SOFTWARE TECHNOLOGY MATURATION

Samuel T. Redwine, Jr.
Institue for Defense Analyses
1801 North Beauregard
Alexandria, Virginia 22311
703 845 2368

William E. Riddle
software design & analysis, inc.
1670 Bear Mountain Drive
Boulder, Colorado 80303
303 499 4782

ABSTRACT

We have reviewed the growth and propagation of a variety of software technologies in an attempt to discover natural characteristics of the process as well as principles and techniques useful in transitioning modern software technology into widespread use. What we have looked at is the technology maturation process, the process by which a piece of technology is first conceived, then shaped into something usable, and finally "marketed" to the point that it is found in the repertoire of a majority of professionals.

A major interest is the time required for technology maturation -- and our conclusion is that technology maturation generally takes much longer than popularly thought, especially for major technology areas. But our prime interest is in determining what actions, if any, can accelerate the maturation of technology, in particular that part of maturation that has to do with transitioning the technology into widespread use. Our observations concerning maturation facilitators and inhibitors are the major subject of this paper.

A. INTRODUCTION

Technology transition activities are the heart of any technology improvement program. These are planned, overt actions taken to move a piece of technology into widespread use. They involve primarily the packaging of the technology so that it is well-received, the active insertion of the technology into some initial, high-impact arenas, and the ultimate marketing and dissemination of the technology to a broad community.

This report was prepared as part of work performed for the Department of Defense STARS Joint Program Office by the Computer and Software Engineering Division of the Institute for Defense Analyses. It originally appeared as part of IDA Paper P-1788, Institute for Defense Analyses.

Technology transition follows technology development within an overall process of technology maturation. The study of technology maturation is complex (see, for example, references 1 and 2). To determine some characteristics of the maturation process for software technologies, and some of the facilitators and inhibitors affecting the transition segment, we have analyzed several case studies[3] which individually treat specific software technologies. These case studies were prepared by people highly knowledgeable in the technologies, thus avoiding some of the more obvious pitfalls in trying to conduct a broad-based study. From these studies we have extracted data concerning the time required for major steps. We have also identified activities critical to the transition steps as well as activities which speed or slow these steps. These summary observations are discussed in this report along with our general conclusions concerning the processes of technology maturation and transfer.

B. TECHNOLOGIES INVESTIGATED

The case studies covered the following technologies:

- knowledge-based systems
- software engineering principles
- formal verification
- compiler construction
- metrics
- abstract data types
- structured programming
- software creation/evolution methodology
- SCR (Software Cost Reduction) methodology
- software development/acquisition standards

- DoD-STD-SDS (Department of Defense Software Development Standard)
- AFR (Air Force Regulation) 800-14
- cost models
- automated software environments
- Smalltalk-80*
- SREM (Software Requirements Engineering Methodology)
- Unix**

For some technologies, very specific instances, such as the Smalltalk-80 system, were investigated. For others, such as verification technology, the investigation was in general terms considering the technology area as a whole.

Four different types of software technology are present in our sample. First, our sample includes *major technology areas* such as knowledge-based systems, metrics, software engineering, compiler construction and verification. Advancements in major areas such as these depend on coordinated advancement in several inter-related areas, many of them frequently theoretical in nature. In addition, only a small segment of the technical community will directly use the technology in these areas. For example, few professionals actually generate compilers but virtually all of them use a (semi-automatically generated) compiler in their work.

The second type of technology that our sample includes is *technology concepts* such as abstract data types and structured programming. This type of technology is conceptual in nature and usually ends up as a basis for a variety of usable pieces of technology. For example, structured programming led to the development of structured analysis.

Methodology technology is the third type found in our sample. This technology addresses how to most effectively create and evolve software and is a mixture of technical and managerial principles, practices and procedures. This is basically "second-level" technology that provides the rules and guidelines for how other technology can be best employed. The instances of this type of technology in our sample are: DoD-STD-SDS, the emerging Department of Defense (DoD) software life cycle model; AFR 800-14, the Air Force policy regarding the acquisition of embedded software systems; and the SCR methodology that underlies the Navy's Software Cost Reduction program.

Consolidated technology, the fourth type of technology found in our sample is also "second-level" technology. In this case, the underlying technology is collected and made to work together so as to provide something significantly better than any of the individual pieces of technology alone. A simple example is software cost estimation technology that brings together metrics, statistical prediction techniques, and empirical study. The other example within our sample is automated software development environments, for which we have considered three specific systems: Unix, Smalltalk-80, and the Software Requirements Engineering Methodology (SREM).

C. TECHNOLOGY MATURATION

The individual case studies provide considerable detail about what happens during technology maturation. To provide some overall comparisons, these individual histories must be placed on a common scale and we use the one displayed in Figure 1. This figure defines six major phases for software technology maturation by fixing the time points that indicate passage between phases. Our interest in technology transition leads us to focus primarily on the period following the emergence of usable capabilities (time point 2). For all types of technology except consolidated technology, this is the point at which overt, planned action can start to have some definite impact on technology maturation. In the case of consolidated technology, the process of fitting individual pieces of technology together may require some basic concept formulation and development and extension, and these preliminary activities may themselves also be amenable to acceleration.

The major time points for the individual technologies in our sample, as reported in the individual case studies, are indicated in Figure 2 and used in the graphs of technology maturation in Figure 3. Each time line in Figure 3 ends at 1984 and the time lines are adjusted so that the points for the appearance of a clear solution definition (time point 1) are lined up. The time lines are organized first by type of technology and second by the length of the development and extension phase. Since our interest is primarily in the period after

Smalltalk-80 is a trademark of Xerox Corporation.
Unix is a trademark of AT&T Bell Laboratories.

BASIC RESEARCH
investigation of ideas and concepts that later prove fundamental
general recognition of problem and discussion of its scope/nature

0

Appearance of a Key Idea Underlying the Technology or a Clear Articulation of the Problem

CONCEPT FORMULATION
informal circulation of ideas
convergence on a compatible set of ideas
general publication of solutions to parts of the problem

1

Clear Definition of Solution Approach Via a Seminal Paper or a Demonstration System

DEVELOPMENT and EXTENSION
trial, preliminary use of the technology
clarification of the underlying ideas
extension of the general approach to a broader solution

2

Usable Capabilities Become Available

ENHANCEMENT and EXPLORATION (Internal)
major extension of general approach to other problem domains
use of the technology to solve real problems
stabilization and porting of the technology
development of training materials
derivations of results indicating value

3

Shift to Usage Outside of Development Group

ENHANCEMENT and EXPLORATION (External)
Same activities as for ENHANCEMENT and EXPLORATION (Internal) but they are carried out by a broader group, including people outside the development group.

4

Substantial Evidence of Value and Applicability

POPULARIZATION
appearance of production-quality, supported versions
commercialization and marketing of the technology
propagation of the technology throughout community of users

4a

Propagation Throughout 40% of the Community

4b

Propagation Throughout 70% of the Community

Figure 1: Software Technology Maturation Phases

usable capabilities become available (time point 2), the time lines are presented again in Figure 4 for those technologies that have achieved some degree of usage outside of their developing group, adjusted to emphasize the phases following the availability of usable capabilities (time point 2).

D. GENERAL OBSERVATIONS

Figures 3 and 4 indicate a wide variance in the time that it takes for a technology to mature from the emergence of a key idea to the point that it is used by professionals outside of the developing group or by the technical community at large. Nonetheless, the figures do indicate some general characteristics of the technology maturation process.

1. Major Technology Areas

For this type of technology, maturation requires a very long period of time. This is presumably caused by two things. First, since the area is broad, many advancements in specific pieces of technology are needed before a general advancement can take place in the area as a whole. Second, because the technologies we investigated in this area are ones that do not naturally move into widespread use because of their specialized nature, full maturation must await the "pulling" effect of a recognized need for the product technology derived from the technology within the major area.

For several of the major technology areas investigated, maturation was guided by an external force. Verification technology was essentially

```
0 1 2 3 4
: : : : :
: : : : V
: : : V   Substantial Evidence of Value and Applicability
: : V   Shift to Usage Outside of Development Group
: V   Usable Capabilities Available
V   Definition Via Seminal Paper or Demonstration System
Emergence of Key Idea
```

Knowledge-based Systems

0-1965-	appearance of artificial intelligence systems providing intelligent assistance (for example, Dendral)
1-1973-	appearance of systems containing a knowledge base (for example, Hearsay)
2-1978-80-	appearance of knowledge-based systems that can be routinely used for problem-solving tasks (for example, R1)

Software Engineering

0-1960-	inadequacy of existing techniques for large-scale software development noted in several projects (for example SAGE)
1-1968-	concept of software engineering is articulated at Workshop on Software Engineering at Garmisch Partenkirchen
2-1973-74-	general collections of papers appear and policy guidelines are established in various communities
3-1978-79-	texts and generally usable systems supporting software engineering appear (for example, the SREM system)
4-1983-	use of software engineering shifts to a larger community through actions such as the DeLauer directive and the definition of a Software Engineering Institute

Verification

0-1966-	Floyd's paper on program correctness analysis
1-1971-	King's demonstration system appears
2-1975-	multiple systems are available
3-1979-	usage of some systems shifts to application groups

Compiler Construction

0-1961-	Iron's paper on compiler generation
1-1967-	review paper by Feldman and Gries
2-1970-	usable systems appear (such as the XPL system at Stanford)
3- -	(cannot be determined)
4-1980-	appearance of production-quality compiler-compilers

Metrics

0-1972-	publication of book on Halstead metrics
1-1977-	results of trying to measure various empirical and analytic measures appear

Abstract Data Types

0-1968-	initial report on information hiding
1-1973-	appearance of some languages using idea of abstract data types (for example, TOPD design language)
2-1977-	major publication on the subject and frequent appearance of the concept in new programming languages (for example, CLU)
3-1980-	use of abstract data types in other technologies (such as in the Affirm program verification system)

Structured Programming

0-1965-	Dijkstra's paper on programming as a human activity
1-1969-	paper on structured programming by Dijkstra at the First NATO-sponsored Workshop on Software Engineering
2-1972-73-	concept is widely discussed and presented in papers
3- -	(cannot be determined)
4-1976-	publication of first introductory text based on structured programming

SCR Methodology

0-1968-	appearance of concepts such as information hiding and communicating sequential processes
1-1976-	completion of feasibility demonstration by NRL with positive experiences
2-1978-79-	appearance of training material and models of usage
3-1982-	methodology moved to a variety of other organizations

DOD-STD-SDS

0-1967-	initial articulation of phased approaches to software development
1-1980-	contract signed for development of DOD-STD-SDS

AFR 800-14

0-1972-	basic need for policy and specific guidance is documented
1-1973-	initial draft policy is published
2-1974-	policy guidance is published
3-1974-	final draft is available
4-1975-	regulation and instructions for its use are officially published

Cost Models

0-1966-	appearance of first collection of cost-related data
1-1976-	appearance of a usable system (Price S)
2-1978-	alternative systems are available (for example, COCOMO)
3-1981-	publication of Boehm's text

Smalltalk-80

0-1965-	Kay's thesis defines concept of a personal computerized notebook
1-1972-	preliminary version of Smalltalk is available
2-1976-	major new version of Smalltalk appears
3-1981-	other companies start porting the Smalltalk-80 system to their computers
4-1983-	Smalltalk-80 available as a commercial product

SREM

0-1968-	ISDOS system demonstrates applicability of attribute-value-relation approach to pre-implementation activities
1-1973-74-	first concrete definition of the SREM system appears
2-1977-	first release of the SREM system
3-1981-	Vax version available

Unix

0-1967-	appearance of the Multics system
1-1971-	initial versions of Unix available
2-1973-	Unix system debuts at Sigops conference
3-1976-	collection of papers appears and system begins to be widely used in academic community
4-1981-	announcement of Unix System III

Figure 2: Software Technology Maturation Points

languishing in the development and extension phase until its value for life-critical and secure application areas was recognized and special attention and focus was directed toward application areas with these characteristics. Compiler construction technology experienced a similar phenomenon when the general focus on high-level languages in the late 1970's forced attention to capitalizing on the technology that had been developed so far and developing the capability to economically produce production-quality optimizing compilers.

2. Technology Concepts

The two examples considered in this area share the characteristic that they are simple ideas with a fairly easily understood conceptual basis. Thus they have matured fairly rapidly; but they themselves have not spread throughout the technical community as fast as some of the technology based on them. For example, structured design, which is based on the concept of structured programming, is probably used by a larger segment of the technical community than is structured programming itself. This is to be expected -- ideas can mature fairly quickly but it takes their unambiguous definition as a specific technique, perhaps supported by tools, to make them usable by the majority of the technical community.

3. Methodology Technology

Methodology technology concerns the rules and guidelines governing use of other technology for the creation and evolution of software systems. As such, several things have to happen before this type of methodology can transition into widespread use. First, the underlying technology has to mature. Second, the rules and guidelines have to be developed. Finally, demonstrations of value have to be prepared in order to prove the worth of the technology.

The concept formulation phase for methodology technology can, therefore, be rather long, as evidenced by the time line for DoD-STD-SDS. The technology supporting full life cycle development methods was just not in place for this technology to mature faster. Once the technology is in place, then maturation can occur fairly fast as evidenced by the time line for AFR 800-14. But the definition and standardization process does not necessarily go quickly -- in the case of AFR 800-14 there was a relatively small, homogeneous target community, but in the case of DoD-STD-SDS the target community is much more diverse and the process of gaining approval is considerably slower. The value of good planning and the careful development of convincing cost/benefit demonstrations is shown by the experience in maturing the SCR methodology -- this project sets a standard for work in this area and indicates what can be expected in the "average" case that

```
Metrics
            ...**0****1.*******
Knowledge-based Systems
        ...**0*******1.******2****
Software Engineering
        ...**0*******1.*****2****3****4*
Verification Technology
            ...**0****1.***2***3*****
Compiler Construction Technology
          ...**0*****1.**2*********4****

Abstract Data Types
            ...**0****1.***2**3****
Structured Programming
              ...**0***1.***2**4a**b****

SCR Methodology
        ...**0*******1.**2**3**
DOD-STD-SDS
...*0************1.**2*
AFR 800-14
                  ...**01.24***********
                        3.
Smalltalk-80
         ...**0******1.***2****3*4*
SREM
          ...**0*****1.**2***3***
Cost Models
  ...**0*********1.*2**3***
Unix
              ...**0***1.*2**3****4***

Note: the last point on each time line is 1984
```

Figure 3: Software Technology Maturation Time Lines

```
Software Engineering
 ...**0*******1*****2.****3***4*
Verification Technology
           ...**0****1***2.***3*****
Compiler Construction Technology
           ...**0*****1**2.*********4****

Abstract Data Types
           ...**0****1***2.**3****
Structured Programming
            ...**0***1***2.**4a**b****

SCR Methodology
         ...**0*******1**2.**3**
AFR 800-14
                   ...**012.4***********
                        3.

Smalltalk-80
         ...**0******1***2.****3*4*
SREM
           ...**0*****1**2.***3***
Cost Models
        ...**0*********1*2.**3***
Unix
                 ...**0***1*2.**3****4***

Note: the last point on each time line is 1984
```

Figure 4: Software Technology Transition Periods

the methodology technology is well-developed and carefully transitioned into the professional community.

4. Consolidated Technology

The situation is similar for consolidated technology -- many things have to come together in order for the technology to fully mature. From the case studies it appears that the enhancement and exploration phases take longer than for methodology technology, presumably because of the need to build the "glue" that fits various pieces of technology together. In fact, the phasing seems rather the same as for technology concepts but one must realize that consolidated technology must, of necessity, lag the maturation of the technology on which it is based.

E. ACCELERATION OF TECHNOLOGY MATURATION

We have too few case studies and the areas are too disparate to be able to determine the nominal case for technology maturation. In fact, one suspects that special considerations will make each instance rather unique and that it will be hard, if not impossible, to predict the maturation time line for a technology by investigating the time lines for other technologies even if they are quite similar.

Our case studies do, however, indicate a number of factors that can inhibit or facilitate the maturation of technology. These give us some insight into what we should and should not do in order to assure that technology matures as smoothly as possible. These factors are discussed in this section.

A word of caution: the various factors are discussed individually but they will interact in very complex ways for any given technology and we do not address these interactions here. An interesting study that reflects the overall effect of all factors impinging on the maturation of technology has been done by Graham[4] who notes that there has been a 50-year cycle in the popularization of technology and makes some interesting observations as to why this is so and how it will affect the transition of software technology.

1. Critical Factors

The case studies indicate a number of factors that are critically necessary in the sense that trying to move a technology into widespread use is almost pointless unless these factors are present. Our case studies show a few instances of failure when these factors have not been present. In many cases, however, total failure was avoided by realizing the problem and correcting the situation -- therefore, many of our examples in this section really show the slowing effect of failing to establish the right context for technology transition.

Conceptual Integrity. The technology (or the base technology in the case of methodology or consolidated technology) must be well developed. In particular, there can be no major outstanding questions about the conceptual basis underlying the technology --- the resulting controversy will just slow things too much. The area of metrics indicates how controversy can make any meaningful progress impossible. On the other hand, the conceptual clarity of the Unix operating system (due in part to the preceding work on Multics) made its maturation relatively quick. As another positive example, the clean separation of concerns within the area of compiler construction technology has led to that area being one of the few major technology areas that has matured to popularization.

Clear Recognition of Need. The technology must fill a well-defined and well-recognized need. Often this need just materializes, but in many cases the need must be articulated by a well-respected "salesperson." Time points 1 and 2 in our maturation process reflect the critical role played by clear enunciation of a need and a solution.

Tuneability. It must be possible to mold and tune the technology to the specific practices of a variety of technology user groups. Both SREM and Unix provide examples of highly tuneable technologies that could be molded to a variety of "ways of doing business." The transparency of the COCOMO (Constructive Cost Model) cost estimation model is another example of how a technology can be open to modification and therefore more easily incorporated into diverse situations.

Prior Positive Experience. Reports on prior positive experiences with the technology should be readily available. Of particular importance are reports showing demonstrable cost/benefit. This is evidently a major contributing factor in the successful transitioning of the SCR methodology. The success of the Price-H hardware cost estimation model was evidently a primary factor in the Price-S software cost estimation model being almost immediately accepted and used. On the other hand, the lack of demonstrable success in applying SREM to software development caused its maturation to be considerably slowed. (The SREM experiences demonstrate a severe problem what will exist for most of the technology surrounding early life cycle phases -- clear demonstrations of value must await the completion of a reasonable sized project and this will delay the results.)

Management Commitment. Management must be committed to the introduction of new technology. And this means that they must actively work to introduce the technology rather than just not oppose its introduction. The case study of the SCR methodology cites at least one case in which the technology introduction failed because of a lack of management commitment.

Training. Training in the use of the technology must be provided and this training should include a large number of examples. This training is particularly critical when new, modern concepts are involved, since then the users must be put in the "right frame of mind" before they can be effectively taught or can effectively use the technology. The studies of SREM, the SCR methodology, and verification technology all cite the criticality of high-quality training.

2. Inhibitors

The factors discussed in the preceding section are critical and if they are not present then the situation must be corrected. Even when they are present, and therefore maturation can proceed, the case studies show that a number of inhibiting factors can slow down the process (as opposed to bringing it to a standstill).

Internal Transfer. It may take additional time to propagate a technology throughout an organization. Our own case studies do not uncover the need for this additional time, but it is demonstrated by a particularly complete and fairly quantitative study of technology transition done by Willis[5]. He notes two important factors that affect the internal propagation of a technology but do not show up in our case studies. First, he notes that the influence of new hires, who are knowledgeable in the new technology and less reluctant to use it, can be important in facilitating the transition of technology. Second, he notes the general requirement that there be some person or group of persons who are personally committed to successfully transitioning the technology -- this is consistent with Boehm's suggestion that there be a technology transition agent within companies to aid and guide the infusion of new technology[6].

High Cost. The cost, either in money or in the time needed to grasp the technology, must be reasonable. The cost estimation technology study cites the high monetary cost of Price-S as inhibiting its acceptance and the intellectual difficulty of COCOMO as having a similar effect. The study of SREM relates the SREM developers' belief that the cost of using the initial version, although reasonable, was above a preconceived threshold and that this delayed the acceptance of SREM. And certainly, the difficulty of performing verification, even with automated aids, has slowed its transition into general use. (One would expect that a high cost would be all right as long as the derived benefit was high, leading to a high benefit-to-"pain" ratio. Our case studies do not support this; rather they indicate that the cost must be reasonably low whatever the benefit gained.)

Contracting Disincentives. Acquisition and contracting practices can serve to slow the spread of technology. The case study of SREM points out that private industry may be reluctant to support the development of new technology when the result will be in the public domain and the developing organization will not gain a competitive edge as a result of their efforts. And the SCR case study mentions the possibility that modern technology for developing maintainable systems will not be used when the possible result will be the reduction of follow-on work.

Psychological Hurdles. Many practitioners feel threatened by new technology, especially when it is advertised as changing (or worse, automating) processes that they have been competently doing for years. And the computer science community seems particularly afflicted by the "Not Invented Here" disease that makes practitioners think that they have developed or can develop something much better than what is being offered for use.

Easily Modified Technology. If something can be changed or fiddled with, then it is almost axiomatic that a computer scientist will change it or fiddle with it. Therefore, if a technology is easily modified (as opposed to just tuned), its introduction will be slowed because it will be modified. This was cited as a debilitating factor in at least one case of introducing the SCR methodology.

3. Facilitators

Technology will spread more quickly when the inhibiting factors mentioned in the previous section are absent. But, our case studies indicate that there are also a number of factors that tend to speed the dissemination of technology.

Prior Success. A "good track record" for the technology's originator(s) will not only make it easier to "sell" but may lead to practitioners seeking out a technology when they read or otherwise hear about a recognized expert's new developments.

Incentives. Software acquisition contracts can specify that new technology must be used -- for example, the Ballistic Missile Defense Advanced Technology Center is currently requiring the use of SREM on some of its contracts. In some cases, the incentive can be indirect -- contract bidders were evidently quite interested in using particular cost estimation models when it was learned that the government was using these models in their proposal evaluation.

Technically Astute Managers. In two case studies, it was noted that adoption of a new technology went more quickly when the decision-makers were well versed in modern software technology.

Readily Available Help. Knowledgeable, articulate advisors and consultants will help in explaining a complex technology, in getting over the almost inevitable misunderstandings that will arise and in dispelling any initial hostility. They can also serve indirectly as salespersons who can make needs visible, explain the benefits of the new technology, and relate the technology to the needs of potential users.

Latent Demand. If there happens to be a well-recognized critical need for the technology, then its adoption can be almost immediate -- this was experienced in the case of cost estimation models where the need was a longstanding one and the early models were almost immediately used.

Simplicity. While the technology and its underlying basis can be quite complex, adoption will move more certainly and smoothly if the instances of it that are available for use are easy to comprehend and are only minimally disruptive to the state of practice. The slowness in maturing verification and knowledge-based system technology is partially due to the absence of simple, easily comprehended systems that deliver this technology. On the other hand, the relative simplicity and conceptual clarity of the Unix operating system contributed to its rather quick adoption by the academic and research community.

Incremental Extensions to Current Technology. This factor is closely related to simplicity. Technology that requires large cognitive switches, such as the SREM and verification technology, will transition relatively slowly, whereas technology that is an incremental enhancement of previous technology, such as the Unix system, will be adopted rather quickly.

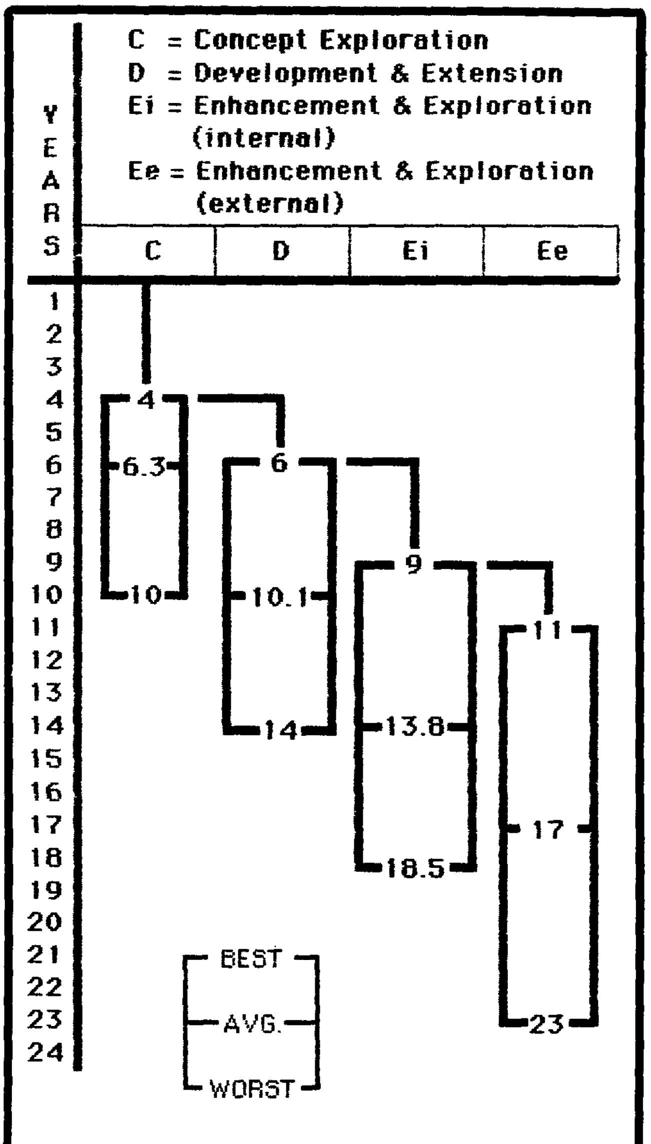

Figure 5: Maturation Times on a Case-by-Case Basis

E. CONCLUSIONS

The case studies show that it takes on the order of 15 to 20 years to mature a technology to the point that it can be popularized and disseminated to the technical community at large. Figures 5 and 6 provide insight into the best (that is, shortest), average and worst times needed for specific phases in the overall process and the variance evidenced by our case studies. In preparing these figures, the government standards (AFR 800-14 and DoD-STD-SDS) have been deleted because they are not representative of the general trend. In addition, data after time point 4 has not been included because it is too sparce to allow general conclusions.

Figure 5 indicates the best, average and worst times when the case studies are considered on a case-by-case basis. The time to reach the end of each phase is shown in terms of the number of years from time point 0, the clear recognition of a problem or articulation of a solution. Thus, the figure shows that in the worst case present in the case studies, it took 23 years to go from time point 0 to time point 4, with 10 years to achieve time point 1, another 4 years to get to time point 2, 4.5 years to go from time point 2 to time point

3, and an additional 4.5 years to achieve time point 4.

In Figure 6, the data is considered on a phase-by-phase basis and charted cumulatively, again with time point 0 as the initial point. Thus, the figure shows that over all of the cases for which there was data for the Enhancement and Exploration (internal) phase, the least amount of time needed for this phase was 3 years, the average time was 3.8 years and the greatest amount of time was 5 years -- when charted as cumulative to the average amount of time needed for the previous phase, this gives the quoted figures of 13, 13.8 and 15, respectively.

In special cases, these times can be considerably shorter than the average. One such case is technological concepts and ideas that do not need computerized support or a change in a user's mind set. Another special case is methodology technology that can transition very quickly when the need is clear and the target community is a close knit and homogeneous one.

The case studies show a very mixed situation with respect to government coordination as a means of facilitating technology transition. AFR 800-14 was established very quickly, mainly because of the homogeneity of the target community. DoD-STD-SDS, on the other hand, has gone through at least one major revision because of the difficulty of arriving at a consensus among the technical community -- this has considerably slowed its maturation. Even in the case that there is a clear need and a strong desire to agree on a piece of technology that can be used throughout the government, the decision-making process can slow the maturation considerably -- the need for a common high-level language within the DoD was articulated as early as 1971[7] but the initial definition of Ada was not chosen until eight years later and it took another four years to finalize the language definition.

No technology will transition into widespread use unless there is a recognized need, a receptive target community and believable demonstrations of cost/benefit. In addition, bringing a technology into widespread use requires a well-defined channeling of attention accompanied by concentrated support. It also needs an articulate advocate who will argue both for the need to support the technology's development and the value of the technology once it is developed.

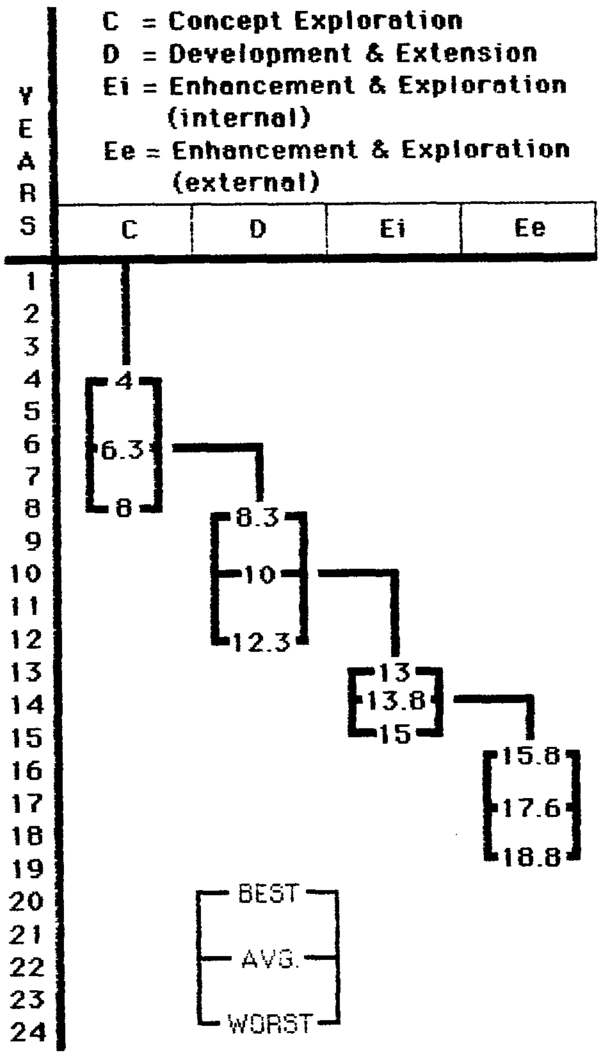

Figure 6: Maturation Times on a Phase-by-Phase Basis

The best process for transitioning technology seems to be incremental expansion in small steps with trial use and the careful gathering of empirical evidence concerning the technology's value. The transitioning of Unix is the epitome of this approach. Unix was initially an incremental improvement over Multics and the Unix system appeared as a series of systems, each being an incremental improvement over the previous instance. Government constraints upon AT&T contributed to the failure to commercialize Unix and this slowed what might have been a relatively quick transition. The carefully managed exploration of Smalltalk-80 by professionals outside the development group is a good example of how the initial part of technology transition can be done in controlled, small steps.

Compiler construction technology provides a slightly different example of this process. Early in the development of compilers there was a clean separation of the problems into a number of inter-related concerns: parsing, optimization, code generation, etc. This separation made it possible to make improvements that were relatively small with respect to the area as a whole and that could be relatively easily delivered to the compiler development community.

For all of the cases we looked at, technology transition was inhibited not by making major conceptual or strategic blunders, but rather by making small, relatively simple mistakes that were easy to correct once they were identified. SREM provides a good example. With 20-20 hindsight, it was obviously a mistake to initially implement SREM on a super computer. The decision was a reasonable one because of the risk associated with preimplementation support and the availability of the TI ASC computer to support the project. But the restrictions that this imposed on the target user community, and the difficulty in porting that resulted, caused a significant slowing in demonstrating and thus transitioning this technology. Once the problem was appreciated and the resources obtained to fix the situation, SREM was ported to a much more widely available host in less than a year and its transition to wider use put back on track.

Our case studies also show that, in general, technology transition is facilitated by actions that improve the context in which the transition is taking place. Acceleration will come more from actions such as providing a focusing goal, improving the technical capabilities of the target community, assuring that training materials and personnel are available, and assuring that the technology "packages" that are provided for use are conceptually coherent and relatively simple improvements over the technology already in use.

Our study shows that a number of factors can affect the speed at which technology matures and is transitioned into widespread use. Even without the inhibiting effect of not providing the basic context or making mistakes, the technology will take a long time to mature. The degree to which the maturation can be speeded seems limited, but there are several actions, relating to the context in which the technology is maturing, that can be taken to accelerate technology maturation.

G. ACKNOWLEDGEMENTS

We are indebted to the authors of the individual case studies for providing the basis for our survey and for helpful critical reviews of our assessments and conclusions.

REFERENCES

1. R.F. Rich (editor), The Knowledge Cycle, Sage Publications, 1981.

2. William D. Garvey. Communication: The Essence of Science, Pergamon Press, 1979.

3. In Redwine, Becker, Marmor-Squires, Martin, Nash, and Riddle. DoD Related Software Technology Requirements, Practices, and Prospects for the Future. IDA Paper P-1788, Institute for Defense Analyses, Alexandria, Virginia, June 1984.

- John Bailey. Cost Model Technology Transition.
- Paul C. Clements, *et al.* Case Studies of Software Engineering Technology Transfer.
- Richard A. DeMillo. Compiler Technology Insertion Network Study.
- John H. Manley. Technology Case Study: Software Engineering Concepts.
- John H. Manley. Technology Case Study: Software Metrics.
- John H. Manley. AFR 800-14 History.
- Ann Marmor-Squires. Formal Software Verification as an Example of Software Technology Transfer.
- Ronnie J. Martin. DoD-STD-SDS: The Development of a Standard.
- Samuel T. Redwine, Jr. Structured Programming: A Technology Insertion Case Study.
- William E. Riddle. "The Magic Number Eighteen Plus or Minus Three: A Study of Software Technology Maturation." [Also appeared: ACM SIGSOFT Software Engineering Notes, 9, 2 (April 1984)].
- William E. Riddle. Knowledge-based Systems as a Case Study in Software Technology Maturation.

- William E. Riddle. Abstract Data Types as a Case Study in Software Technology Maturation.
- David Weiss. Time Line for Development and Transfer of SCR Methodology.

4. Alan K. Graham. "Software Design: Breaking the Bottleneck." IEEE Spectrum, March 1982, pp. 44-50.

5. R.R. Willis. "Technology Transfer Takes 6 Plus/Minus 2 Years." Proc. IEEE Workshop on Software Engineering Technology Transfer, April 1983, Miami, Florida.

6. Barry W. Boehm. "Keeping a Lid on Software Costs." Computerworld, January 18, 1982.

7. Paul Cohen. Early Software Technology Efforts Within DCA. March 1984.