THE OBJECTIVES OF THE SOFTWARE ENGINEERING INSTITUTE

A.N. Habermann

Rapporteurs: Mr. L. Mancini
Mr. A.M. Koelmans

1. Introduction

Our society is becoming more and more dependent on computer software. The production of software systems has increased by an order of magnitude but the demand has increased even more and keeps rising [g]. This situation puts severe pressure on the software manufacturing community which is plagued by a severe shortage of well-educated and well-trained software engineers. An additional aggravating factor is the fact that customers demand much higher standards of quality and reliability than ten years ago.

It is generally recognized that the software production practices have not kept up with the rising demands for quantity, size and quality. Many software systems are produced with technology that dates back to the sixties and early seventies. Systems are often written in special purpose programming languages, depend on peculiar hardware features and make little use of software engineering principles such as information hiding and data abstraction [a]. Such systems are not portable, error-prone when modified and critically dependent on what their implementors can remember.

The U. S. Department of Defense, a major consumer of software systems, has taken the initiative to improve the software production process [b]. The DoD Software Initiative started in the mid-seventies with the design of the Ada programming language [c], which largely solves the portability problem. The language also provides direct support for software engineering principles through its package mechanism.

The Ada language was primarily designed for supporting the individual programmer in writing programs that depend on program modules written by others [d]. In contrast to Ada's elaborate support for what is generally known as "programming-in-the-small", the language provides only limited support for "programming-in-the-Large" and practically no support for "programming-in-the-many" [e]. Programming-in-the-large is the term for the integration of program modules into systems and for matters such as version control and configuration management. Programming-in-the-many refers to the fact that software systems are typically generated by teams of people which necessitates task coordination and project management.

It has frequently been stated that improving the software production process is largely a matter of developing good methods for programming-in-the-large and for programming-in-the-many. This insight made the DoD follow up on its Ada initiative with various efforts to stimulate the use of good software engineering principles, methods and tools in practice. One of these efforts gave rise to the Software Engineering Institute at Carnegie-Mellon University.

The main objective of the Software Engineering Institute is technology transition. Its main occupation is not basic research, but the transition of existing technology into routine practice. The SEI will have achieved its goal if it succeeds in reducing the time lag between the development of software production methods and tools in the research environment and their application in practice. Thus, the main task of the Software Engineering Institute is to find out what kind of promising methods and tools exist in the research environment, to understand the production environment and explore which of the methods and tools are applicable and, last but not least, to demonstrate their feasibility through the construction of prototype programming environments and through the training of personnel.

The Software Engineering Institute is to a limited extent involved in basic research and education. Its role in these areas is primarily one of stimulating research and education in software engineering by inviting visitors for extended periods of time and by coordinating projects undertaken with affiliates from government, industry and academia. Although these activities do not constitute the major occupation of the SEI, they do amount to an effort that is comparable in size to that of an average computer science department in the U. S.

The SEI has established a five year plan and has started a number of projects that address the issues described in that plan. The following sections present the essence of that plan and describe in a little more detail the state of Ada-related projects of the SEI. The basic ideas behind the five year plan are described in an earlier paper written when the projects were just started [f]. That paper also describes the SEI's organization and its project planning procedures.

2. The SEI's Basic Themes

The Software Engineering Institute will focus on issues involving the production and maintenance of large software systems. The SEI has decided not to promote a particular software development methodology or philosophy, but to review the various issues involving the production process and product quality. The SEI's major task is to demonstrate the available software development support technology and show how it can be applied. A major decision has been to achieve improvement of software production and maintenance by becoming more technology-intensive instead of labor intensive. We at the SEI believe that production and maintenance can be improved considerably if they are directly supported by the programming environment that provides the software development tools instead of having to rely on labor intensive management procedures and tenuous conventions applied by humans. One of the major benefits of the technology-intensive approach is that the programming environment can apply consistency checks and can enforce management rules which is a time-consuming task for humans and hard to do consistently, instantly and accurately all the time.

For the initial five year period of its existence, the SEI has chosen three areas and three topics which together constitute the six themes that serve as guidelines for selecting projects. Subdivided into areas and topics, the six themes are:

Areas of Interest	Topics of Interest
1. Technology Identification & Assessment	1. Reasoning about the Software Production Process
2. Nature of the Transition Process	2. Tools & Environments for Software Development
3. Education & Training	3. Reusability & Automation

2.1 Areas of Interest

All three areas concern the essence of the SEI: technology transition. The first area includes the task of finding promising software development technology, of exploring who may benefit from it and of showing how it can be applied. We emphasize that the SEI's task is *assessment* and not *evaluation*. The SEI has the task of showing which existing technology can be applied when and where, but not of publishing consumer reports of commercial products. The motivation for the second area of interest, the Nature of the Transition Process, is that there is ample evidence that new technology is often not applied because of reasons that have nothing to do with the merits of the technology itself, but with the organization of the site of its potential users. Existing management procedures are often difficult to change and project deadlines must be met, while application of the new technology may have legal implications and requires retraining of personnel.

The SEI has set a task for itself in education and in training. The purpose of the education program is to increase the number of qualified software engineers in the U. S. The purpose of the training program is to provide potential users of specific new techniques with hands-on experience in the use of the new tools or methods. The two tasks are clearly distinct. The first has a long range goal that cannot be achieved by the SEI alone. The SEI can take the initiative and play a major role in the coordinator of educational activities, but the work must be done in close collaboration with academic institutions. In order to make this happen, the SEI has started an academic affiliates program in which a dozen institutions work with the SEI on the development of educational material for software engineering. The status of the SEI's ongoing education

project is discussed further on in this paper in the Section describing the currrent projects. In contrast to the broad scope of the education program, the SEI's task in training is directed to teaching the use of specific techniques that are ready for transition. Training is provided for the methods, tools and environments that the SEI prepared for transition and that are on display in the SEI's software laboratory.

2.2 Topics of Interest

The three main topics of interest concern the existing tools and environments for software development, the issues of management and quality control and the development of a programming attitude that has the potential of saving a considerable amount of programmers' time. The three topics are not particularly intended to lead to specific projects that separately explore the issues involved, but are thought of as desirable aspects of projects undertaken by the SEI. The first two topics relate to existing technology that can be demonstrated by the SEI in prototype programming environments.

Reusability and automation are at this time the most promising techniques that in combination with effective tools and programming environments, may substantially increase program productivity. Reusability is routinely practiced by sharing of operating system facilities and by the use of tools and libraries. However, most software systems contain large sections of code with very similar functionality that are written for each system from scratch. Reusability is often difficult to achieve because of poor documentation or programming language pecularities or because of dependencies on the underlying operating system. These problems are difficult to avoid when programs are directly written in a programming language. Research into the topic of reusability tends to move in the direction of program descriptions at a level of abstraction that does not force the detailed bindings of a programming language. Tools, programming environments and automation play an important role in transforming such abstract descriptions into concrete programs that can be compiled or interpreted by existing language systems.

3. The SEI Plans

At this time it may be too early for the SEI to pursue reusability and automation because this topic has only recently received the attention of the research community. Instead, the SEI has scheduled its activities according to the relative maturity of the available technology. This criterion has led to the following sequence of phases planned for the initial five year period:

phase 1: the use of the Ada language and associated tools;

phase 2: techniques for integrating Ada and Ada tools with existing tools of other programming environments;

phase 3: the transformation of programming environments to intelligent programming assistants;

phase 4: the application of reusability and automation.

The phases will undoubtedly overlap in time. Each phase needs a period of preparation before specific projects can be started that lead to technology transition. We envisage that the focus of the SEI's activities will gradually shift from one phase to the next during the five year period. Phases will also not die out abruptly. It is for instance almost certain that the transition of Ada technology is still of great interest at the end of the five year period.

The SEI has started its activities with a series of workshops on the software factory concept which led to a broad discussion of the nature of software engineering. The first two workshops took place in March and April of 1985 with a group of approximately ten well-known scientists and engineers. The third workshop was held in October of 1985 with an attendance of approximately 50 people. The fourth and last workshop was held in February of 1986 with approximately 250 people attending.

One of the major results of the first two workshops was the decision to stand firmly behind the Ada language and to work on issues of programming-in-the-large related to Ada. The attendants of these workshops generally agreed that no major breakthrough is to be expected similar to the invention of the transistor for computer hardware that will all of a sudden solve the production problems associated with the development and maintenance of large systems. This opinion was shared by the attendants of the October and February workshops. It is generally believed that improvements must come from improved programming environment support combined with specific techniques that enables us to write reusable code and automate the program generation process.

Although there is general agreement among the experts in the field that a revolutionary breakthrough is not to be expected in the foreseeable future, there is no general agreement on the direction in which to push for an evolutionary improvement of the software production process. The various opinions were clearly stated by the members of a discussion panel at the February workshop. That particular panel consisted of two representatives of industry, one from

the government and one from academia. The first panel member, representing the large software producers working on government contracts, took a strong position in favor of Ada based on the strong desire to make software portable. The representative of the government did not oppose the use of Ada, but was much more interested in the successive phases of the software production process, starting with requirement specifications and moving through design and functional specifications to coding, testing and maintenance. The government representative expressed as his strongest desire the standardization of the various phases of the production process. The representative of the software industry took a strong position in favor of development tools and integrated programming environments for programming-in-the-Large. The representative of the academic world took a position close to that of the representative of the software industry, but made a strong pitch in favor of Artificial Intelligence Technology, particularly for the rich interpretive environment of Lisp.

The best course is for the SEI to combine these ideas into a strategy that is based on the use of Ada, but emphasizes programming-in-the-Large and that moves the evolution of programming environments in the direction of the intelligent assistant in the style of Artificial Intelligence. The large software producers are right in stressing portability. The use of a standard Ada is a step in the right direction to achieve portability. Standardization is undoubtedly desirable, but may come too early for issues that have not stabilized. Standardization is possible for the Ada language and its compilers. It is, however, too early to standardize programming environments or the phases of the software production process. There are important developments taking place in the design of programming environments for Ada that each should be given a chance to mature. Regarding the Lisp environment, there is no reason why an Ada environment cannot provide a similar functionality and a similar smooth user interface. The issue is not whether creating such an environment is possible for Ada, but whether someone will do the work. The Lisp environment took more than ten years to evolve!

4. Current Projects

The SEI has organized most of its activities in the form of projects that are based on the areas and topics of interest and on the phases which determine the shifting emphasis over the years. In the eighteen months of its existence, the SEI has started eight projects on the following subjects:

1. the Software Factory concept
2. the Showcase Environment (with the Ada Browser)
3. Software Licensing
4. Software Engineering Curriculum
5. Ada Applications

6. Ada Technology
7. Ada Environments
8. Intelligent Programmer Assistants

The Software Factory concept was the central topic at the series of workshops discussed in the preceding section. The workshops have been very helpful for the SEI to shape its five year plan and to determine its areas and topics of interest.

4.1 The Showcase Environment

The Showcase Environment and the Software Licensing project were started soon after the SEI was established. The showcase project is a continuing effort to build up and maintain a software laboratory in which software development tools and environments are on display. The laboratory is used for demonstrations and for training. It has been used to show the merits of software development environments such as DSEE [h], the planning tools on MacIntosh, the object oriented programming style of SmallTalk [i], the evolution of user interfaces as in Andrew [j], etc. A project particularly worth mentioning is the Ada Browser which provides editing and tracing facilities for the visible parts of Ada packages. The designer of a software system written in Ada can look at his or her design in different views that show the interdependency of packages and can zoom in on packages to inspect details such as the types of subprogram parameters.

4.2 Software Licensing

The legal issues project is a first step in the area of the Nature of Technology Transition. The government is for instance faced with the fact that software it owns was written with tools it does not own. Questions arise as to how the government can assure that these tools will remain in existence for system maintenance. Serious problems exist with regard to intellectual property rights, software licensing, copyrights, etc. The first year of this project was concluded with a workshop held in April 1986 which was attended by many of the legal experts on software in the U. S. The results of the project are presented in a report that has been submitted to the SEI's sponsors.

4.3 Software Engineering Education

The Software Engineering Curriculum project is the main project in the academic affiliates program. Its initial purpose is the design of material that can be used in software engineering courses. In a later phase, the project will focus on programming environments for software engineering education. The team working on the course material consists of a small permanent SEI staff and a number of visiting faculty from various universities who spend a sabbatical semester or

the summer at the SEI. The relationship between these visitors and the SEI will be extended over a number of years which provides the opportunity for visitors to work on long-range plans.

The team working on the curriculum design came to the conclusion that specialization in software engineering fits best at the senior undergraduate and at the master's graduate level. The course material is designed for a masters degree in software engineering. The conclusion is based on the observation that software engineering must rest on a solid undergraduate education in computer science that contains components of discrete mathematics and electrical engineering. The design includes a description of the prerequisites for the software engineering curriculum.

The initial plans of the education team were discussed in detail with a group of approximately 25 experts at a workshop in April 1986. The attendants were unanimous in their opinion that an important part of a software engineering curriculum must consist of hands-on experience with real software systems. Small homework problems are of little interest because their solutions hardly ever extrapolate to large systems. Students should participate in an on-going project for an extended period of time during the summer or while taking the software engineering courses.

It was generally agreed that a software engineering curriculum should not be based on an enumeration of topics, but on lasting principles. The attendants distinguished between the general scientific priniciples of theory and experiment and the discipline-specific principles that characterize the activities in a field. A collection of the principles that are believed to distinguish software engineering from disciplines such as physics, mathematics and psychology are:

a) the embedded character of software systems
b) the discrete character of software
c) the limited knowledge of the resources
d) the decomposability of software problems

Software engineering may share each of these principles taken separately with some other discipline, but the collection is what makes it clearly distinct from other disciplines.

Software systems usually perform a particular function in a larger organization that exists outside of the software world. This embedded character imposes evaluation criteria on software that depend on extraneous factors determined by the larger organization in which the embedded software system may play only a minor part. Software engineering is in this respect quite different from physics which explicitly states as one of its principles that its laws are universal. (There is in this respect more resemblence with biology if one considers the relationship between a particular animal and its habitat.)

Another distinction between software engineering and physics is the discrete character of software engineering problems in contrast to the continuous view physicists have of the natural world. The discrete character of software enineering brings the discipline closer to mathematics with which it has in common the basic concepts of formal models, enumeration, abstraction and inductive proof. Software engineering is, on the other hand, quite distinct from mathematics in that its objects do not have complete control of the resources they use. System behavior may depend on operating system facilities and hardware that was designed independently, or on the degree of concurrency which may vary from time to time. Such factors do not play a role in mathematical theories.

Decomposability might also be called the principle of divide and conquer. In this respect software engineering is again comparable to mathematics, but different from physics or psychology. Mathematician and software engineers believe that they can decompose a problem, solve the parts separately and combine the solutions of the subproblems into a solution of the initial problem. Physicists and psychologist work under the hypothesis that the whole is larger than the sum of the parts which implies that there is always an aspect of the initial problem that has escaped consideration when one attempts to partition a problem of their domain.

The discussion on the principles of software engineering has given the education team a guidance as to how to present various topics in the curriculum. Other sources of information that contributed to shaping their plans were the experience at Wang Institute and the perspective on future developments which largely coincides with the observations of the Software Factory workshops. A digestion of all the material and ideas collected by the education team has led to a contract with a well-known publisher for a series of monographs on a variety of topics in software engineering.

4.4 Ada Applications

The Ada Application project resulted from the October workshop on the Software Factory concept. A review of the state of Ada at that workshop resulted in a three part conclusion:

1) The initial difficulties with writing compilers for Ada have been overcome. More than a dozen Ada compilers have passed the validation test. Although Ada compilers now compile correctly, a lot of work is still needed in making the object code more efficient and in building optimizing compilers.

2) The Stoneman Report on the APSE Ada environment is based on outdated technology of the midseventies. It is too early to freeze the design of programming environments for Ada. There are several developments going on that should be tested on the commercial market.

3) There is a lack of good teaching material for Ada. There are several beginners texts, but most take a bottom up approach. The Ada reference manual is of good quality for the expert Ada compiler writer, but not suitable as instruction manual. Teaching material is needed that takes a top down approach starting with the central modularity concept of Ada implemented by packages.

It seems that software producers have trouble introducing Ada, because the time can hardly be spared for teaching programmers the proper use of Ada. The lack of time for Ada instruction is primarily caused by the firm time commitments to government contracts that exclude all activities from the budget that do not directly contribute to the goal of that contract.

A working group at the October workshop discussed a plan to create a government contract specifically for the purpose of introducing Ada. The discussion resulted in an SEI project in collaboration with the STARS program (which is another component of the DoD's Software Initiative). The project involves the introduction of Ada in a number of industries that are interested to compare the use of Ada with other languages they have used and are willing to experiment with Ada support tools or Ada environments. The industries will write a proposal for rewriting a system of more than 1oo,ooo lines of code in Ada. Part of the task is to observe what kind of development tools are used to support Ada and how effective these have been. The task of the SEI consists of monitoring the projects, organize meetings to let the participants "compare notes" and to summarize the results.

4.5 Ada Technology

The SEI tries to make use of its comparative advantage that it has because of being part of Carnegie-Mellon University and because of the particular expertise of the people at the SEI. A field in which the SEI has particular expertise is in compiling techniques and automatic compiler generation. Since there is still room for improvement in the area of building Ada compilers, the SEI has started an Ada technology project that focuses on compiling techniques and tools for compiler construction.

The tools provided by this project center around the IDL description language that has been used to write the well-known DIANA description of Ada. The project is undertaken in collaboration with the University of North Carolina. The results will be made available to compiler writers and training in the use of the tools will be provided by the SEI.

4.6 Ada Environments

The Request for proposal that came out in the spring of 1984 asked for an evaluation of the ALS and AIE Ada environments that had been contracted by the U. S. Army and the U. S. Airforce respectively. At the time the proposal was submitted in the summer of 1984, it became clear that the ALS system would still be in its early stages and that the AIE system would not be available at all. (The AIE Ada Compiler has finally been announced for September 1986 without an Ada Environment!) Since by this time several Ada compilers were reaching the market, we proposed to explore the environments that these compilers were using instead of evaluating a particular environment. Final discussions resulted in the SEI's Ada Environment Evaluation project that has as its main goal to create a method for evaluating these vastly different environments.

The term "evaluation" in the project title is somewhat misleading, because no specific evaluation is contremplated, but a study is planned of how Ada environments can be evaluated [k]. The project has just concluded its first year with interesting results and plans to continue with a study of how Ada runtime and dynamic debugging environments could be evaluated. The Ada environments project has primarily focused on issues of programming-in-the-Large. It did not want to repeat the elaborate validation process that Ada compilers have to go through to get official approval. The project did also pay little attention to issues of code efficiency or optimization. The project looked at an Ada compiler as just one of the tools in a programming environment. If one looks at a compiler that way, issues of importance are things such as the quality of the error messages, the compilation speed, the linking procedures, etc. The general issues of primary interest in the project were those of system version control, configuration management, access control and project management.

It was clear from the start of this project that programming environments are hard to compare. Instead of basing the study on a comparison, we categorized the issues into four areas that respectively concern

1) functionality
2) user interface
3) context utilization
4) performance

One can easily generate a list of functions that an environment must provide in order to handle source versions, documents and object code. One can also specify in general terms what kind of facilities must be available for system configuration management and for system construction.

Environments may offer these functions in different forms or in different sets, but expert programmers will soon find out whether or not the necessary facilities are present.

The situation is slightly different with the user interface. There are some general characteristics that every user interface must have, but in contrast to the functionality, user interfaces tend to have specific characteristics that strongly depend on the type of input-output device used. Among the general characteristics are properties such as "the response to an editing command must be less than a second", "the interface must distinguish between an expert user and a novice" or "for long operations, the interface must show at regular intervals that the system is still alive". The specific characteristics, however, depend on factors such as having a bitmap display or character terminal, whether mouse control is included, etc. Although the facilities offered by these devices differ too much to make a useful comparision, it is not hard to list for each system separately some properties one expects a good user interface to have. We did that in the preliminary report of the project that was submitted to the SEI's sponsors [I].

Another criterion that determines the quality of a programming environment is the degree to which the available hardware and software resources are utilized. A good programming environment will make use of the potential execution speed of a large machine, will not do superfluous paging when sufficient memory space is available, will make use of existing software and existing network facilities, etc. For each host machine and each host operating system one must determine which specific facilities are provided and how well the programming environment makes use of these facilities.

Performance has already been mentioned in passing in the paragraph on user interfaces. It is of course a matter of concern for every part and every aspect of a programming environment. Our study shows that at this stage of the development of programming environments special attention should be paid to the performance of linking procedures. Some compilers make the impression of being fast because programs are compiled in small pieces, but the advantage is lost when it comes to linking modules into systems which for some language processors is not much faster than if all modules were compiled together.

In the study of Ada programming environments, we found that designers have introduced three different models which are represented by the Rational machine, the DEC-Ada system and the ALS system. The philosophy of the Rational approach is that everything is Ada and hardware is specifically designed to make this approach feasible. Ada is not only used as the language for writing programs, it is also used for programming-in-the-Large. The interactive command language for is Ada, the user can write Ada command procedures, version control is done with Ada libraries. The only other thing besides Ada is the concept of text which is needed for documentation.

The opposite approach was taken by the designers of DEC-Ada. Here the philosophy was to integrate Ada facilities with an existing environment which already provides extensive support for programming-in-the-Large and also offers the benefit of extensive program libraries. This approach makes it possible for the user to live in a multi-language environment in which subprograms written in various languages such as Ada, Pascal and FORTRAN can be used in each of the language systems. The user of DEC-Ada does not work in the uniform Ada world as the user of the Rational machine, but has the benefit of using a rich environment of mature software.

The ALS approach is in the tradition of operating system design. It is based on the view that an Ada environment should be an interface between an operating system and its users in the style of a Database Management System. In this approach, each user command can be treated as a transaction that is controlled by the environment which performs all the necessary actions that have to do with version control, access control, etc. The idea behind this approach is reflected in the name ALS which stands for Ada Language System. The ALS differs most from the other two in the automatic checks it performs and in including management procedures as part of the standard user interface. A consequence of this approach is that ALS forms a layer on top of an existing system that hides this underlying system from the user.

4.7 Intelligent Program Assistants

We expect that programming environments of the future will show a more intelligent behavior than today's rigid environments. The term "intelligent" is used here in the behavioral sense of "what one expects an intelligent being to do". We expect that the development of the field will lead to a type of environment that interacts with its users more in the style of an assistant than of a toolbox. One of the main differences between an assistant and a toolbox is that the assistant will need only little instruction and apply his own judgement as to what should be done automatically, possibly without involvement of the boss. Another main difference is that the assistant has a large tolerance in understanding his boss. He is able to interpret commands depending on properties of the communication language and on factors such as the context and the history of the interaction between the assistant and the boss.

The technology is emerging that makes this kind of environment possible. It is, however, not yet ready for transition into routine practice. We do believe that this technology has the attention of the research community and that the SEI should prepare for technology transition in this area during the last phase of the current five year plan. An important part of the preparation is planned as an SEI project in which we explore what technology already exists that may lead to the desired goal. It is not surprising that the SEI looked in the first place at AI technology. At this time the project studies in particular the various Lisp environments and the extensive assistance provided by

these environments for program development. The intermediate results are made available in the SEI's software laboratory.

The strength of a Lisp environment is not so much in the language, but in the rich collection of facilities provided by the environment. We believe that the functionality of a Lisp environment could also be provided by an Ada environment. However, Ada is ten years behind the Lisp development. It will take a major effort to create an Ada environment that is as rich as a Lisp environment. We believe that such an effort should be given high priority because programmers' productivity will not increase enough without it.

One feature of the Lisp environment does not easily carry over to an Ada environment: the interpretive nature of Lisp. The great advantage of the interpretive approach is that one does not need a dynamic debugger that operates on the object code. In a Lisp environment one can immediately test a function in the same environment in which it is written. Efficient object code is produced at a later stage by a Lisp compiler when debugging with the interpreter has been completed.

Summary

The Software Engineering Institute was started as the result of the DoD's Software Initiative. The primary purpose of the SEI is the transition of promising software development technology from the research environment to routine practice. The SEI has put together a five year plan and has scheduled successive phases in which it will introduce increasingly advanced technology. The first phase involves the Ada language and the tools that are needed for constructing large software systems.

The SEI discussed its plans extensively with the leaders in the field of software engineering and with the software producers and users at large. As a result of these discussions, the SEI staff found its viewpoint confirmed that it should not adopt a particular software development methodology. Instead, the SEI will show how improvement of the production process can be achieved by replacing the traditional labor-intensive approach to managing a software project by a technology-intensive approach. The SEI demonstrates the technology that is ready for transition in its software laboratory in which it also offers opportunity for training.

The SEI has organized most of its work in project form. It has started eight projects that deal with legal issues, with existing technology and with future developments. Although basic research and education are not the primary concerns of the SEI, they constitute an integral part of the program. In education, the focus is on a software engineering curriculum at the graduate level.

The education project is a joint effort of the SEI and a number of participating academic institutions.

The first results of the work at the SEI are becoming available. The software licensing project has submitted a report that has served as the basis for a public debate on software ownership and licensing arrangements. The Ada Environment project has produced a report in which it publishes the results of the experiments it conducted for the validation of its method for the evaluation of Ada environments. The showcase project has built up the SEI's software laboratory in which several programming environments and development tools are on display.

In conclusion, the SEI has started a number of activities that directly serve its objective of technology transition. The five year plan specifies definite goals that are being achieved by a number of projects that initially focus on technology around the Ada language. The SEI will be successful if it can maintain and extend its contacts with the potential users of the advanced software development technology it is able to demonstrate.

References

[a] Parnas, D.L.,
"On the Criteria to be Used in Decomposing Systems into Modules"
Comm. ACM, Vol 15, no 12 (Dec 1972)

[b] Martin, E.W.,
"Strategy for a Department of Defense Software Initiative"
Computer, Vol 16, no 3 (Mar 1983)

[c] "Reference Manual for the Ada Programming Language"
U.S. Department of Defense (Jan 1983)

[d] DeRemer, F., H. Kron
"Programming-in-the-Large versus Programming-in-the-small"
ACM SIGPLAN Notices (June 1975)

[e] Kaiser, G.E., A.N. Haberman
"An Environment for System Version Control"
Proc. IEEE Spring CompCon, San Fransisco (Feb 1983)

[f] Barbacci, M.R., A.N. Habermann, M. Shaw
"The Software Engineering Institute: Bridging Practice and Potential"
IEEE Software, Vol 2, no 6 (Nov 1985)

[g] Boehm, B.,
"Software Engineering Economy"
Prentice Hall, Englewood Cliffs (June 1981)

[h] Leblang, D.B., R.P. Chase, Jr
"Computer-Aided Software Engineering in a Distributed Workstation Environment"
Proc. ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments
Pittsburgh, Pa (April 1984)

[i] Goldberg, A., D. Robson
"Smalltalk-80: The Language and its Implementation"
Addison & Wesley, Reading, Mass (March 1983)

[j] Morris, J., J. Howard, F. Hansen
"The Andrew Windowing System"
Techn. Rep. Carnegie-Mellon University (Jan 1986)

[k] Weiderman, N.H., A.N. Habermann, M.W. Borger, M.H. Klein
"A Method for Evaluating Environments"
submitted to the second ACM SIGSOFT/SIGPLAN Symposium
on Practical Software Development Environments

[l] Weiderman, N.H., A.N. Habermann, M.W. Borger, M.H. Klein
"Preliminary Evaluation of the Ada Language System"
Technical Repc SEI-86-MR-6, Carnegie-Mellon University (Apr 1986)

DISCUSSION

The discussion was postponed to the end of the second lecture. Nevertheless some remarks arose during the presentation which are worth mentioning.

Professor Randell asked whether, in the speaker's opinion, the variance of software quality and expertise across industry is so great that increasing the present average would be revolutionary. The reply was that indeed such a variance exists, so that different situations may be found that accordingly require different measures, ranging from managerial means to formal techniques.

Professor Lee asked whether any measure of software quality and productivity has been developed within the projects carried on at the Software Engineering Institute (SEI) at Carnegie-Mellon University. Professor Habermann stated that particular emphasis is being placed on this aspect.