EVOLUTION OF SOFTWARE DEVELOPMENT ENVIRONMENTS

David G. Belanger, Stephen G. Chappell, and Mike Wish

David G. Belanger, ***Stephen G. Chappell,*** *and* ***Mike Wish*** *are with AT&T Bell Laboratories. Mr. Belanger is head of the Advanced Software Department at Murray Hill, New Jersey. He is responsible for research into concepts and tools to increase productivity and quality in software development. He received a B.S. from Union College, an M.S. from Case Institute of Technology, and a Ph.D. from Case Western Reserve University, all in mathematics. He joined AT&T in 1979. Mr. Chappell is director of the Software Development Laboratory in Liberty Corner, New Jersey, which is responsible for technology and processes to improve quality and productivity of software development. He has a B.S. in electrical engineering from the Georgia Institute of Technology, an M.S. in electrical engineering from Northwestern* (continued on page 6)

Software development technology is, increasingly, a key element in AT&T's ability to compete across all our products. This article introduces a sequel to the 1988 issue of this journal on software productivity. New technology and methodology being brought to bear on problems of specification, design, implementation, testing and maintenance are outlined. Three dominant trends are emerging in software development environment technology: tools are growing more powerful, technology is being applied across the software life cycle, and development environments are becoming more closely integrated within an open architecture.

History

More than 10 years ago the *Bell System Technical Journal* published a classic issue[1] on the UNIX® operating system. During the next few years, the computing environment described in that issue, including the UNIX system, the C programming language, and the Programmer's Workbench[2] became the *de facto* standard for the development of new software products.

By the early 1980s, the UNIX system provided a large collection of tools for the creation of software, including debuggers, scanner and parser generators, and a powerful shell programming language. Utilities such as `make` and SCCS (source code control system) supported large software projects by automating the building of systems from individual units and allowing for version control.

These tools evolved, and a new generation of software technology emerged, as described in a special *AT&T Technical Journal* issue on software productivity published in July/August 1988. The object-oriented language C++ extended C's expressive power, maintainability, and reusability, particularly for large systems.[3] SABLE and `nmake`,[4] which replaced SCCS and `make`, improved software project management by supporting products throughout their life cycle. Likewise, Stage[5] advanced the technology for building application generators beyond that of `lex` and `yacc`,[6] and the Korn shell enhanced the

command language interface to the UNIX system.[7] Though the new generation of tools is not yet universally used in the development of software, this technology is becoming the choice for an increasing number of projects.

Challenge

The cost and potential impact of new technology is becoming more and more critical to the corporate bottom line as software accounts for an ever-increasing part of AT&T products. Software is responsible for much of the functionality in these products, as well as for a large share of their design and development cost. Over 60 percent of AT&T R&D employees were involved in software in 1989.

As a measure of the size of the software problem to be overcome, AT&T research and development delivered 20 million lines of new and changed code in 1987 on a base of 35 million lines of code. As Figure 1 indicates, these numbers are growing rapidly without corresponding expansion of the software development staff. The size and complexity of software in products continues to increase rapidly at an average compound annual growth rate, of delivered lines software, of about 25 percent.

To meet this challenge, AT&T initiated a program to increase the productivity of software development and the quality of its resultant products. The triangle in Figure 2 illustrates key areas in which change is being driven—organization, process, and technology. These correspond to the "who," "how," and "with what tools" of software development. A significant change in any of these will impact the others. Although the papers in this issue are oriented toward advances in software development technology, each has substantial impact on the process used in a part of the software life cycle.

Perspective

In order to acknowledge and advance the current state of software development within the company, we present a sequel to the 1988 software productivity issue

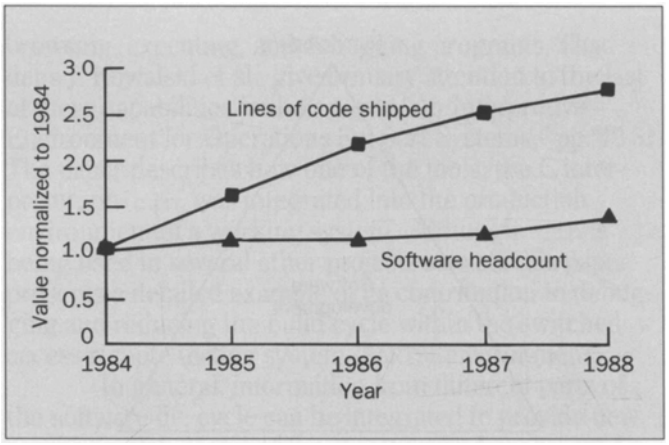

Figure 1. Software growth. Lines of code shipped has climbed steadily in recent years and in 1988 almost tripled the 1984 level. Meanwhile, software head count—the number of people engaged in software development—increased only slightly.

of this journal. The thrust is on some of the advanced techniques with which our developers are gaining significant experience. Although only a sample is represented, at least three dominant trends can be identified:

- Individual tools are becoming more powerful.
- Technology is being applied across the software life cycle.
- Software development environments are becoming more integrated.

The first item means that we have more of the same, but better. Most of the tools described in the 1988 issue have evolved considerably in the last 2 years in response to feedback based on intensive use. They now do more with less work, and do things previously not possible. Moreover, there has been a maturation and healthy flow of new technology into the software development process. As an example, the C information abstractor,[7] along with software visualization capabilities, is now used in over 200 projects. The rapid movement of

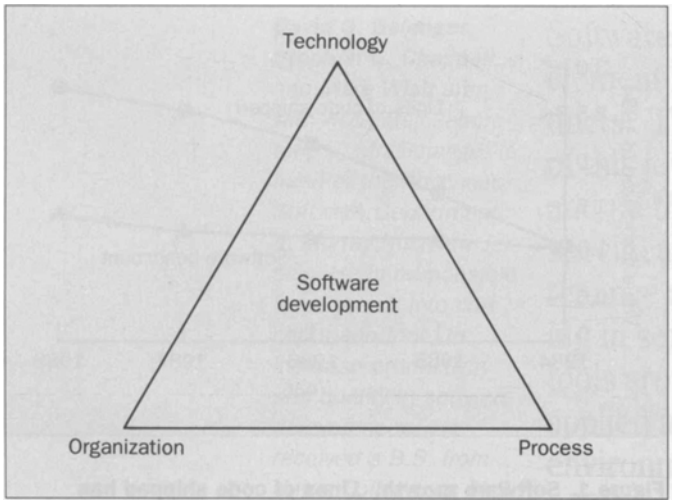

Figure 2. A triangle relates the key elements in software development. Processes and organization underlie the creation, transfer, and integration of new technology. All three elements are undergoing changes aimed at improving productivity continuously.

software from research into practice reflects increased emphasis on making early use of new technologies.

The second and third items in the list above represent more fundamental extensions of effort, with increased focus on organization and process. In this regard, a variety of studies have shown that coding is only 10 to 15 percent of the total effort involved in a large project. The time-consuming, labor-intensive activities that precede programming (e.g., specification) and follow it (e.g., testing) have been supported only by ad hoc tools. Although the cost of an error discovered late in the life cycle can be 1000 times greater than one found early, little technology has been available for uncovering errors in specification or design.

More resources are now directed at problems with the largest potential cost savings for software development. Technology is being coordinated with process, reaching out to the full range of software development activities. Likewise, the individual software "tools" are being integrated to work together within an open software development environment architecture.

In this issue, authors describe ongoing efforts to address weak spots in the software life cycle, and to create more integrated, manageable software development environments (SDEs).

Supporting the Software Life Cycle

The initial tasks in building a software product involve understanding in detail what the product should do, describing that functionality in a way that can be understood for downstream activities (e.g., implementation, test) and by specifiers of future releases, and defining an architecture for the product. A number of techniques, such as prototyping, have been used effectively to understand the requirements for products. These have the advantage of allowing the customer and systems engineer to view the proposed functionality and make adjustments before large resources are committed to design and development.

In "Improving the Front End of the Software-Development Process for Large-Scale Systems," pp. 7-21, G. David Bergland et al. describe a new approach to this problem for a very large, complex project, the 5ESS® electronic switch. This system is AT&T's premier central office switch, containing millions of lines of software with extremely stringent requirements for performance and reliability. In order to create and convey requirements information, scenarios and constraints are employed. For example, a scenario might be to process a telephone call. Constraints would refer to the conditions under which the call can be made (e.g., one must take the phone off hook). The approach, which uses CASE (computer-aided software engineering) technology, will reduce the errors that pass through the specification process and, because of its formality, reduce ambiguity in the passage of the specifications to designers and testers. In the future, this may allow automatic generation

of some of the switch code, leading to cost savings and quality improvements.

At the other end of the life cycle, testing is inevitably a large part of the development of a software product. It is also very labor-intensive, generally taking between 20 and 40 percent of the effort in a large, complex system. The Buster test management system, described by Kent C. Archie and Robert E. McLear in "Environments for Testing Software Systems," pp. 65-75, provides an approach to the problem that is now finding wide use within AT&T. It includes a semiautomated text execution facility that supports the tracking of individual tests as well as test suites. Buster standardizes and controls the testing process, facilitates reuse, simplifies training procedures, and helps to eliminate waste and redundancy.

In "COMPAS: A Development Process Support System," pp. 52-64, H. Jack Barnard and Robert B. Collicott describe a system that integrates measurement and development processes and ties them to process improvement. Although COMPAS was initially created for document management, it supports a significant portion of the product life cycle. For example, COMPAS' facilities for inspections and reviews are heavily used, especially on the architecture, design, and coding of a product. By querying a relational database, developers can obtain needed information about plans, methodologies, requirements, architecture, design, and test for a product. By providing access to the knowledge needed to produce better specifications more quickly, COMPAS promotes the capture of errors as early as possible, so that they can be fixed at minimal cost.

Creating Integrated Environments

Every software project has a large collection of tools that address individual tasks, but the information used by individual tools is now, more and more often, shared among them. A key approach to integrating such tools is the C Environment System (`cens`).[7] While the overall system includes facilities for creating, editing, browsing, executing, and debugging programs, Thaddeus J. Kowalski et al. give primary attention to the last of these capabilities in their paper "An Interpretive Environment for Operations Support Systems," pp. 42-51. The paper describes how one of the tools, the C interpreter, or `cin`, was integrated into the production environment of a working system. Although `cin` is being used in several other projects as well, this paper presents a detailed example of its contribution to debugging and reducing the build cycle within the switched access remote testing system (SARTS) environment.

In general, information from different parts of the software life cycle can be integrated to provide new, more complete views of the structure and progress of a software product. Integration of software information started with developers using processes, such as inspections, and manually comparing information from different sources, such as design documents and code. It is continuing with applications that share data from several development environment systems such as the program database described by David G. Belanger et al. in "Toward a Software Information System," pp. 22-41. Because the activities in a software project overlap, the information flow between them is bidirectional rather than a simple hand-off. Information captured in the early stages of a project—such as the specification and design phases—is required not only for producing code but also for testing and documentation. Similarly, there is a reverse feedback in which information based, for example, on code is part of the input into the design of the following release. This is known as *reverse engineering*.[9] Understanding these concepts has led to a further integration of tools in the Software Development Environment.

One of the side effects of the increased size and life span of software products is that individuals are continually called upon to modify designs or code that they did not create. Add to this the fact that software does not have a common visual instantiation and we face the problem of *discovery*. Discovery refers to the process of

understanding software design before making changes. Statistics indicate that the average discovery time required to answer a modification request is about 30 percent of the total time required. "Toward a Software Information System" describes tools designed to provide automated support for making software more "visible" and, therefore, reducing the time for discovery. With one of these tools, the C Information Abstractor, developers can investigate and view the structure of their software in a matter of seconds even for systems of over a million lines of code.

Finally, the project described by Charles C. Hayden et al. in "The Software Development Assistant," pp. 76-90, integrates a large variety of tools into a powerful software development environment for a specific type of software development. In this case, it is for the development of the Definity™ telecommunications system. Included within this integrated environment are some of the tools discussed in the other papers, for example, the C Interpreter and C Information Abstractor. The paper describes the power of integrating a variety of tool capabilities and a single graphic user interface, while giving proper attention to issues of process. It is an excellent view of the functionality to which software development environments are moving in response to the requests of developers.

The Future

Change in software development is accelerating. This issue describes part of that change, representing the present or near future of some of the more progressive of AT&T's software developers. Space does not permit us to touch on many other exciting new directions being tested in improving software development. The progress we are making in improving software productivity and quality depends on matching all the vertices of the triangle in Figure 2. It also depends on a variety of technologies for reuse in products as well as for the development environment itself.

We expect, on the basis of technologies now being prototyped in research, to repeatedly reenact this improvement process.

References

1. UNIX Time-Sharing System, *Bell System Technical Journal*, July-August 1978, Vol. 57, No. 6, Part 2.
2. T. A. Dolotta, R. C. Haight, and J. R. Mashey, "The Programmer's Workbench," *Bell System Technical Journal*, July-August 1978, Vol. 57, No. 6, Part 2, pp. 2177-2200.
3. S. Cichinski and G. S. Fowler, "Product Administration through SABLE and NMAKE," *AT&T Technical Journal*, July/August 1988, Vol. 67, No. 4, pp. 59-70.
4. J. C. Cleaveland and C. M. R. Kintala, "Tools For Building Application Generators," *AT&T Technical Journal*, July/August 1988, Vol. 67, No. 4, pp. 46-59.
5. S. C. Johnson and M. E. Lesk, "Language Development Tools," *Bell System Technical Journal*, July-August 1978, Vol. 57, No. 6, Part 2, pp. 2155-2176.
6. M. I. Bolsky and D. G. Korn, *The Korn Shell Command and Programming Language*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
7. D. G. Belanger, G. D. Bergland, and M. Wish, "Some Research Directions for Large-Scale Software Development," *AT&T Technical Journal*, July/August 1988, Vol 67, No 4, pp. 77-92.
8. C. Bachman, "A CASE for Reverse Engineering," *Datamation*, July 1, 1988, pp. 49-56.

Biographies (continued)

University, and a Ph.D. in computer science from Northwestern. He joined AT&T in 1969. Mr. Wish is head of the Computer-Aided Information Systems Research Department in Murray Hill. He is responsible for research and technology transfer of tools and concepts that give productivity and quality leadership in the development of software products. He holds a B.S. in psychology from Case Western Reserve University, an M.S. and Ph.D. in mathematical psychology from the University of Michigan, and an M.S. in computer science from Columbia University. He joined AT&T in 1967.

(Manuscript received December 1, 1989)