Turning points in software development

by R. Goldberg

The early days of software development consisted mainly of writing program code for computers, i.e., programming. I began to program very shortly before the first issue of the *IBM Systems Journal* was published in 1962. Programming, still very new at the time, was the method by which computers were directed to manipulate large amounts of data or perform very large calculations. The goal was to make computers perform some operation that could not be done in any other way, and any nicety of quality or ease of use was certainly secondary.

The *IBM Systems Journal* was established by a company that then contained one of the largest groups of professionals dedicated to the creation of software. Reading through the cumulative index[1] of the *Journal* provides a sense of what was of concern to a very select group of programmers, spanning 38 years of evolution of a very dynamic new discipline. Some papers are groundbreaking, others represent the crest of a wave identifying the forces that come together to produce a new approach, and most represent a definitive view of the state of the practice in software development. As part of a retrospective look at computing developments featured in the *Systems Journal*, a selection of papers on topics considered to be turning points in software development are discussed in this section, and some representative examples are reprinted.

Software development as an art

The earliest sequence of published topics was concerned with examples of what could be done with software, such as data analysis and large calculations. Programming was then a very new tool that was still being learned. Programming languages evolved, and assembler languages gave way to higher-level languages. During this time, as papers dealt with what could be done with software, this evolution extended to better ways of organizing programs to use hardware resources more optimally, i.e., operating systems and various other resource management tools.

The production of software was an art. The practitioners needed to be talented to perform this art and, in parallel with other creative professionals, suffered through periods of unproductive effort. The potential benefits of computers were being blocked by the inability to manage development predictably. But forces were gathering to create a "sea change" in the way software would be developed. The notion that a methodology could be defined to provide a recipe-like definition of the sequence of steps that a program developer should go through in creating software, dictating what to do even when the solution was not evident, was becoming popular.

My career at IBM brought me to a remarkable institution, the IBM Systems Research Institute. It was an internal graduate school created at about the same time that the *IBM Systems Journal* was launched. As a student there in the early 1970s, I took a course on software development taught by Larry Constantine. The material in the course was the precursor for the structured methodologies that dominated the mid-1970s of software development. I joined the faculty several years later and met Glen Myers who was already writing

internal reports on composite structured design. Wayne Stevens was a student at the IBM Systems Research Institute and in conjunction with Myers and Constantine wrote one of the earliest public papers on the application of an analysis technique to create a structured design. Appearing as the first of the papers reprinted in this section on software development, it is entitled "Structured Design" and was published in 1974.

Many other names have been associated with the structured methodologies and with software development methodologies in general, and for a time the family of structured techniques was the recommended approach to be used in developing complex systems. For a fuller description of many of the methodologies and a more complete list of references, see my review paper[2] in the 25th anniversary issue of the *IBM Systems Journal* that appeared in 1986.

The use of methodologies was not the answer. Applications and systems continued to be plagued by low overall productivity rates and an unacceptable number of errors. The search turned to how software was being developed rather than what was being developed. The search was introspective and looked at the attributes of the programmers as well as at the attributes of the sequence of process steps used to create software. Although most of this came to fruition after methodologies became popular, the search had begun earlier.

One of the more significant papers appeared in 1972 and focused on the way programmers organized themselves to create their work products.[3] The concept of the chief programmer and associated team as a combined unit of specialized effort was introduced in the first of the papers, which looked at the way software was produced and how improvements in process could lead to both improvements in product and a decrease in the cost of production.

It was observed early that defect detection and repair was a dominant cost in software production. Many strategies were devised to create an effective portfolio of test cases to find errors, but these were usually put into effect late in development and involved actual costly execution of code. Groups in IBM began to implement early manual reviews of the artifacts created as part of systems and application development. These artifacts included everything from requirements documents, to code, to documentation. Michael Fagan made the intuitive connection between software quality assurance and the quality assurance techniques used in manufacturing operations. He measured effectiveness, codified process, set step goals, and instituted a detection and correction flow for defect tracking. He called his approach the "inspection process," a name derived from quality assurance processes used in manufacturing. His paper "Design and Code Inspections to Reduce Errors in Program Development," reprinted in this section, was truly a turning point in software development.

When the inspection process was being defined, I was teaching software engineering at the IBM Systems Research Institute. Fagan was a frequent guest speaker in my classes, and it was almost possible to chart the students' change in attitude as the use of inspections picked up. When the idea was first presented, they vehemently objected to the thought of subjecting a personally produced product to the public scrutiny of their peers, but after five years or so, students then attending classes could no longer conceive of a development project that did not plan for the managed use of resources to conduct inspections. Even as late as 1998, newly published papers can be found extolling the benefits of inspections and statistically documenting the cost savings that occur because of the ability to detect errors earlier in the software development process.

Inspections, however, focused only on single steps of development. Attention then shifted to what it would take to move software development from a hand-crafted process to a predictable and manageable engineered process. That transition was not going to be easy and has still not been completed. To achieve this formidable goal, it would be necessary to first understand enough about the quantifiable aspects of the artifacts produced in development to be able to create formulas or tables that could be used both to plan and to measure progress.

In 1977 Walston and Felix analyzed 60 development projects in order to estimate the rate at which code had been produced, so as to come up with an algorithm to plan future projects. Their words, taken from their paper published that same year,[4] describe the effort:

> This paper discusses research into programming measurements with emphasis on one phase of that research: a search for a method of estimating programming productivity.

They looked at many of the project conditions in order to arrive at a statistical result that could be used to predict future project duration and effort. The idea of understanding the statistical nature of the elements of production, size of the effort, and rate of completion, is a necessary prerequisite for converting software production into an engineering discipline. You cannot plan where you want to be until you understand where you are today.

Software development as a science

The evolution of software engineering as an accepted field took a major step forward with the publishing of the landmark work directed by Harlan D. Mills at the former IBM Federal Systems Division. The five-part sequence that appeared in Volume 19, Number 4, of the *IBM Systems Journal* laid out the pragmatic foundations of how we should manage the production of high-quality software. As Mills says in the introduction of his paper, "Principles of Software Engineering," which is included with the reprints in this section:

> Software engineering has as one of its primary objectives the production of programs that meet specifications, and are demonstrably accurate, produced on time, and within budget.

This sequence of papers began a transition in both the management of software development and in how we look at software quality. The relationship between a software fragment, a requirements specification, and the ability to demonstrate the equivalence between the two using predicate calculus techniques, described in a paper by Linger,[5] was the second significant concept that these five papers discussed. As part of the sequence, the Mills and Linger papers presented turning points—one on the management of software development and one on the creation of very high-quality components. Each concept has had a profound effect on the way in which software development is done today.

The management sequence, the view that a disciplined engineering approach must be taken, leads to the use of project management techniques, the creation of a defined process, a work breakdown structure of known steps, associated verification and validation steps, and the use of statistical control techniques to ensure quality and process improvement.

Early in the 1980s I joined the IBM Software Engineering Institute (SEI) to put together a curriculum directed at improving the skills of the professionals who manage software development. The IBM SEI started from the ideas of Mills and his group and worked on the technology transfer of these ideas into the mainstream development areas of the IBM Corporation. I worked extensively with a group, directed by Watts Humphrey, whose mission was to improve the software development process. The entire contents of Volume 24, Number 2, describes their work and should be reviewed by anyone who wants to understand how to improve his or her own organization. A technique that was used extensively, described in the 1985 paper, "A Programming Process Study" by Ronald Radice, John Harding, Paul Munnis, and Richard Phillips, included as a reprint, involved the evaluation of eight large-system programming locations within IBM and evaluated them according to a set of process stages. The "maturity" of the organizations along those specific attributes pointed at where and how process improvement could be effected. The Software Engineering Institute set up at Carnegie Mellon University used this idea and created the Capability Maturity Model (CMM) to measure the effectiveness of development organizations. The application of the techniques described in the above paper by Radice et al., codified and expanded in the CMM, and applied in the Space Shuttle work done at the IBM Federal Systems Division, was further described by Billings et al.[6] in 1994 in the *Systems Journal*.

A paper published in 1985 described how one could move from defect detection to defect prevention. This paper, by C. L. Jones,[7] described how to institutionalize a set of process steps that will extinguish complete classes of defects. The process is triggered by the discovery of a defect, which leads to the identification of why it occurred, and culminates in process changes that would have prevented that defect from occurring at all. This idea was described in expanded detail by Mays et al.[8] and extended by Bhandari et al.[9]

The second sequence of papers, which derived from the landmark Mills issue, is associated with the ideas of provably correct programs. The method described to create such programs depends on choosing logical design structures from among a set of patterns that have been shown not to alter the intent of the specification from which they have been derived. To this was added the state machine concept that isolated a particular method from possible side effects that could otherwise seep in from the external environment.

A 1987 paper on box-structured information systems by Mills, Linger, and Hevner[10] increased the emphasis on a hierarchy of data abstractions and the separation of the methods from the environments in which they would be embedded. Although not the only source of object orientation, the initial idea of the provably correct program was moving steadily in that direction. This direction was cemented even further in the 1993 paper by Hevner and Mills,[11] which specifically called out the connection between object-based system development and box-structured methods.

The use of object orientation to create applications and the beginning of a definition of a methodology that could be used in customer service engagements was also described in the *IBM Systems Journal* in the 1993 paper entitled "The Impact of Object Orientation on Application Development," by Alistair Cockburn, reprinted in this section. Although this paper is not a direct descendent of the initial work by Mills and Linger referenced earlier—the ideas of data abstraction and encapsulation, and incremental and iterative development—all were featured in the turning point papers of more than ten years earlier.

The use of patterns as paradigms to be reused in order to increase both productivity and quality, introduced in 1980, was reemphasized within object-oriented methods and at a more complex level in a 1996 paper.[12] The emphasis here is on design patterns described in an abstract sense and then instantiated for a particular problem. It is precisely the idea suggested earlier for the creation of provably correct programs.

The existence and use of a library of well-defined objects that could then be reassembled for specific purposes later is still being echoed, whether as design patterns or as class libraries of application-specific components. The use of encapsulation and data abstraction to isolate processes from their environments is the enabling feature of the Java** language. These methods all have their roots in the earlier work so well documented in the *IBM Systems Journal*.

Summary

My intent in this commentary was not to describe every paper related to software development in the 38-year publication history of the *Journal*, nor to claim that they were all significant. There are many more papers that fit in the sequences described. There are also other important topics that have not been mentioned—for example, the papers describing the methodologies and economic benefits of software reuse.

This journal does represent the ebb and flow of ideas as practiced by dedicated professionals whose careers are devoted to the effective and efficient production of software. Along the way it recorded key turning-point ideas that were expanded and modified and that still play a crucial role in how we build software.

Cited references

1. 1962–1994 Cumulative Index, *IBM Systems Journal* **33**, No. 4 (1994).
2. R. Goldberg, "Software Engineering: An Emerging Discipline," *IBM Systems Journal* **25**, No. 3/4, 334–353 (1986).
3. F. T. Baker, "Chief Programmer Team Management of Production Programming," *IBM Systems Journal* **11**, No. 1, 56–73 (1972).
4. C. E. Walston and C. P. Felix, "A Method of Programming Measurement and Estimation," *IBM Systems Journal* **16**, No. 1, 54–73 (1977).
5. R. C. Linger, "The Management of Software Engineering, Part III: Software Design Practices," *IBM Systems Journal* **19**, No. 4, 432–450 (1980).
6. C. Billings, J. Clifton, B. Kolkhorst, E. Lee, and W. B. Wingert, "Journey to a Mature Software Process," *IBM Systems Journal* **33**, No. 1, 46–61 (1994).
7. C. L. Jones, "A Process-Integrated Approach to Defect Prevention," *IBM Systems Journal* **24**, No. 2, 150–167 (1985).

8. R. G. Mays, C. L. Jones, G. J. Holloway, and D. P. Studinski, "Experiences with Defect Prevention," *IBM Systems Journal* **29**, No. 1, 4–32 (1990).
9. I. Bhandari, M. J. Halliday, J. Chaar, R. Chillarege, K. Jones, J. S. Atkinson, C. Lepori-Costello, P. Y. Jasper, E. D. Tarver, C. C. Lewis, and M. Yonezawa, "In-Process Improvement Through Defect Data Interpretation," *IBM Systems Journal* **33**, No. 1, 182–214 (1994).
10. H. D. Mills, R. C. Linger, and A. R. Hevner, "Box Structured Information Systems," *IBM Systems Journal* **26**, No. 4, 395–413 (1987).
11. A. R. Hevner and H. D. Mills, "Box-Structured Methods for System Development with Objects," *IBM Systems Journal* **32**, No. 2, 232–251 (1993).
12. F. J. Budinsky, M. A. Finnie, J. M. Vlissides, and P. S. Yu, "Automatic Code Generation From Design Patterns," *IBM Systems Journal* **35**, No. 2, 151–171 (1996).

Robert Goldberg *IBM Learning Services, 590 Madison Avenue, New York, New York 11210 (electronic mail: bobgold@us.ibm.com).* Dr. Goldberg is an Engagement Manager supporting the IBM Software Group. In this capacity he works with the Software Group to assure the availability of training to maintain and enhance the skills of their professionals. He is one of the architects of the IBM Object Technology University and has been curriculum manager with broad responsibility for software engineering and object technologies. He joined IBM in 1968 in a branch office as a systems engineer but very shortly thereafter moved into education as a faculty staff member of the IBM Systems Research Institute, then as a manager in the IBM Software Engineering Institute, and then managed the Software Development Technology Transfer Center. He is the coauthor with Mark Gillenson of *Strategic Planning, Systems Analysis, and Database Design—A Continuous Flow Approach* and coeditor with Hal Lorin of *Economics of Information Processing,* Volumes I and II. He is fascinated by the possibilities that technology brings to advanced training. Dr. Goldberg is an ADCOM member of the IEEE Education Society. He received his B.S. in physics from the Polytechnic University of New York in 1960, and his doctorate in physics from Rutgers University in 1969.

Reprint Order No. G321-5701.