Comment on "Capital-Intensive Software Technology"

Peter Wegner's mega-article, "Capital-Intensive Software Technology" (IEEE Software, July 1984, pp. 7-45), is a masterful survey of the local landscape called "software development," but I believe the article is based on an entirely incorrect assumption. He states, "Intuition suggests that a production process is capital-intensive if it requires expensive tools or if it involves large startup expenditures. Software development is becoming increasingly capital-intensive: its tools are becoming more powerful and expensive, and it requires greater early investment to reduce late expenditures." The article points out many interesting side-effects due mainly to the pervasiveness of personal computers, but I believe it fails to link these interesting side effects to anything remotely connected to "capital-intensive software technology."

First, software development is becoming increasingly cheap. Computers are nearly free compared to 10 years ago; compilers cost $395 instead of $20,000; editors, operating systems, and application generators are perhaps two orders of magnitude less expensive than they were a few short years ago.

Second, knowledge is tremendously undervalued. A BS in computer science can be purchased for a few hundred barrels of oil, the profit reaped from a few acres of farmland, the cost of a new car, or the cost of a European vacation for two. The cost of software development has fallen to the point where thousands of cottage industry programmers have glutted the software market with programs of every description. They literally give away their expertise in every field of endeavor from stock market analysis to electronic circuit design. The problem, I believe, is that software development simply costs more than hardware development. This is due to the steep decline in hardware costs—even steeper than the decline in software costs—rather than any comparative surge in software development costs. The amount of advertising and promotional dollars needed to bring out a new product in an overcrowded market is the culprit.

Software development cannot be cast into the mold of capital-intensive manufacturing or any other economic artifice because knowledge and creativity defy economics. One good idea is worth all of the software development tools and methodologies combined.

Do not get me wrong; I am an advocate of better tools and more productive methodologies, but it is a mistake to believe these are a panacea for that all ails software technology.

Ted G. Lewis
Department of Computer Science
Oregon State University
Corvallis, OR 97331

Author's reply

Ted Lewis overlooks the fact that techniques for constructing small programs do not scale up to large programs. This was realized by the computing profession in the late 1960's when the discipline of software engineering was born. Software engineering's primary objective was to develop a capital-intensive framework for software production that would parallel capital-intensive technology for building large physical structures.

Tony Hoare, in his article, "Programming: Sorcery or Science" (IEEE Software, April 1984, pp. 5-16), suggests that small programs are like sandcastles while large programs are like city blocks. His assertion that one would not use even a prizewinning builder of sandcastles to build a city block nicely captures the idea that techniques for building small programs do not scale up to large programs. Lewis appears to suggest that all programming is simply the building of sandcastles and that city blocks do not exist. This is manifestly not the case.

Capital-intensive technology is required both when a software product is too complex to be managed by a small number of individuals and when a simple software product is mass-produced for a large market. Software technology has made Cobol compilers, spreadsheets, and Pacman affordable by distributing their capital development costs over many consumers, just as industrial technology has made automobiles, television sets, and personal computers affordable.

Software products differ from industrial products in that copies of a software product require few raw materials and little labor to produce. But this is balanced by the fact that widely distributed software products incur support and maintenance costs comparable to the production costs of an industrial product. The initial versions of Unix and Lotus 1-2-3 were produced on shoe-string budgets, but their widely distributed successor systems, Unix 4.2 and Symphony, require capital-intensive organizations to support their distribution. Thus, savings in costs of mass production are balanced by expenses incurred by mass distribution, and the overall economics of mass-producing software products is therefore not too different from that of mass-producing industrial products.

The current software boom has spawned many small start-up companies to manufacture specific systems in areas such as automatic typesetting, computer-aided design, and computer-aided instruction. These companies are heavily capital-intensive, typically requiring an initial investment of $1 million to $5 million and an elapsed time of at least two years before their first commercial product is sold.

Large, evolving software systems that control banks, airplanes, and ships are even more capital-intensive. They are generally constructed either by very large (Fortune 500) companies or by the organization they are designed to control. They require careful planning over a long time horizon and must take into account post-
installation maintenance and system evolution. Greater expenditures on specification, documentation, and system structuring early in the life cycle can greatly reduce costs later in the life cycle. The total capital cost of such systems can be billions of dollars (the DoD command and control system, WIS, is budgeted at over $30 billion over 15 years). Our current technology for such systems is inadequate and results in gross inefficiencies in their construction, use, and maintenance. The programming language Ada was an attempt to develop a capital-intensive technology base for large evolving systems and has served to focus attention on this problem. But Ada has not yet demonstrated that it can significantly reduce the cost or improve the reliability of such systems.

The impact of cheaper computer hardware is similar to that of cheaper energy sources. It has made computers affordable in every home and appliance, but has also increased our appetite for building complex capital-intensive systems. Distributed systems with greater hardware complexity, as well as very complex software systems, are becoming technologically and economically feasible. It is precisely because hardware has become so cheap that software has become the limiting bottleneck. Cheaper hardware provides an incentive and economic justification for capital-intensive software technology in much the same sense the cheaper energy provides an incentive for capital-intensive industrial technology.

When I started thinking about capital-intensive software technology in the late 1970’s, I viewed the analogy between the computer and industrial revolutions as a convenient peg on which to hang some philosophically suggestive verbiage. But the analogy is more than superficial. Energy and engines for transforming energy in the industrial revolution have deep analogies with information and engines for transforming information in the computer revolution. The roles of tools and reusable components, of factories that facilitate the manual and automatic application of sequences of tools, and of an integrated product development technology, are strikingly similar in both industrial and computer contexts.

The further analogy between capital-intensive industrial and software technology appears at first to be less compelling, not only to Lewis, but also to traditional engineers. The myth that software is an add-on requiring less professional expertise than engineering design is widely held by upper and middle management in engineering companies responsible for constructing large hardware-software systems. This is illustrated by a middle manager of a large defense system who, when asked by the author whether software was a primary bottleneck, said that the system was built so that if all the software were thrown overboard, system performance would not be affected.

The failure to realize that software technology is as capital-intensive as hardware technology is in fact one of the causes of the present software crisis. Software structures are not as visible as hardware structures, and the effort needed to complete them cannot be estimated as easily as that needed to complete a partially constructed building. The effectiveness of software tools cannot be judged by their visible effects, as can the effectiveness of cranes and steam hammers. Information structures are more subtle than physical structures but are just as real. Their construction, maintenance, and distribution require careful capital-intensive preparation by highly-trained “construction engineers.”

Peter Wegner
Box 1910
Dept. of Computer Science
Brown University,
Providence, RI 02912

INNOVATIVE SOFTWARE APPLICATIONS

Software A&E, a rapidly growing company in Washington, DC, specializes in the development of advanced products and systems which integrate software engineering and artificial intelligence technologies. Current projects include intelligent workstations, expert system development tools, extensions to Ada for AI applications, and more. These projects involve the use of Ada, C, and LISP on VAX, M68000, and LISP-oriented machines.

A CAREER FOR THE PROFESSIONAL COMPUTER SCIENTIST AND SOFTWARE ENGINEER

We are seeking a computer scientist or software engineer who believes, as we do, that valuable gains can be achieved through the intelligent application of state-of-the-art techniques:

SOFTWARE ENGINEERING PRINCIPLES - object/data abstraction, information hiding, abstract specifications, iterative/incremental development.

DESIGN AND IMPLEMENTATION OF COMPLEX SYSTEM COMPONENTS - interactive interface, database management, knowledge-based systems, distributed processing.

DEVELOPMENT SUPPORT - environments, program transformation, prototyping, reusable components.

If you fit this description and are seeking a career in an environment oriented to the software professional, call Andrew Ferrentino at (703) 276-7910 or send your resume to:

SOFTWARE ARCHITECTURE AND ENGINEERING, INC.
Suite 800 1500 Wilson Boulevard
Arlington, VA 22209

MULTIPROCESSING ENGINEERS—PROGRAMMERS

The new Center for Supercomputing Research and Development (CSR&D) at the University of Illinois at Urbana-Champaign seeks experienced technical staff for the development of high-speed multiprocessor systems. With support from the U.S. Government and the State of Illinois, CSR&D is initiating an effort to develop a multiprocessor that will deliver high performance over a wide range of computations. This effort is based on over ten years of research in software, architecture, and algorithm development performed by a nucleus of an experienced research staff. In addition, this project will enjoy the cooperation of some of this country’s leading industrial groups. The project will begin October 1, 1984.

We need approximately twenty practitioners with state of the art skills in:

- Gate array design.
- System integration.
- Compilation for parallel and array computers.
- Operation system implementation.
- Parallel algorithms in numerical and non-numerical applications.

All positions will be permanent and full time. The type of positions to be filled, including duties and qualifications, are as follows:

SENIOR RESEARCH ENGINEER
To take part in a group of research engineers in area of design and systems integration. Minimum qualifications include a Master’s degree in Computer Science (C.S.) or Electrical Engineering (E.E.) or Ph.D. is desirable), and four years of experience.

RESEARCH ENGINEER
To take part in research and development of the above hardware activity. Minimum qualifications include B.S. degree in C.S. or E.E. (Ph.D. is desirable), and four years of experience.

SENIOR RESEARCH PROGRAMMER
To provide programming expertise and technical leadership in development of compilers, or operating systems. Minimum qualifications include a Master’s degree in C.S. or E.E. (Ph.D. is desirable), and four years experience.

SENIOR RESEARCH SCIENTIST
To provide programming expertise and technical leadership in development of multiprocessor numerical and non-numerical algorithms in a variety of applications. Minimum qualifications include a Ph.D. degree in C.S., E.E., or any computer-related field.

RESEARCH SCIENTIST
To take part in research and development of the above numerical and non-numerical algorithms. Minimum qualifications include a B.S. degree in a computer-related field.

The Center for Supercomputing Research and Development will offer industrial-scale salaries, in addition to providing University benefits. Please send resume, including educational background, recent professional experience, and salary requirements to:

D. J. Kuck, Director
Center for Supercomputing Research and Development
University of Illinois
1304 W. Springfield Avenue, Urbana, Illinois 61801
Telephone (217) 333-6150

Applicants should clearly specify the position or positions for which they are applying. To expedite the filling of all positions, qualified applicants will be interviewed during the advertising period. All applications submitted by December 3, 1984, will receive full consideration. Some positions may begin immediately.

The University of Illinois is an Affirmative Action/Equal Opportunity Employer.