Invited Talk

Great Principles in Computing Curricula

Peter J. Denning*
Computer Science Department, Naval Postgraduate School
Monterey, CA 93943 USA
+1-831-656-3603 pjd@nps.navy.mil

ABSTRACT
The nearly three dozen core technologies of computing sit in a simple framework defined by great principles and by computing practices. The great principles are of two kinds, mechanics and design. Computing mechanics comprises computation, communication, coordination, recollection, and automation. Design principles address concerns for complexity, resilience, performance, evolvability, and security. Practices comprise programming, systems, modeling, innovating, and applying. This framework opens many new possibilities for teaching computer science, including new approaches to programming. The new CS curriculum at the Naval Postgraduate School is based on the framework presented here.

Categories and Subject Descriptors
A.0 [General Literature]: organization and structure of computing field. K.0 [Computing Milieux]: organization and structure of computing field. K.2 [History of Computing]: evolution of principles and practices of computing. K.3 [Computing Education]: organization of curriculum, teaching programming. K.4 [Computers and Society]. K.7 [The Computing Profession]: professional practices of programming, systems, modeling, innovating, applying.

General Terms
Algorithms, Measurement, Performance, Design, Reliability, Experimentation, Security, Human Factors, Languages, Theory.

1. INTRODUCTION
The great principles of computing have been interred beneath layers of technology in our understanding and our teaching. This paper is about how to set them free. We propose a great principles framework for computing and discuss some of the reasons it has been so hard to articulate such a framework. The framework suggests new ways to organize a computer science curriculum. The new CS curriculum at Naval Postgraduate School (NPS) adopts the framework.

The great principles framework is a new organizing principle for our field. There are four main reasons to be interested:

- **Understandability:** The field continues rapid growth as new computing technologies emerge and spread into new application areas. Our traditional view of the field as a set of core technologies has become untenable: the number of core technologies tripled from 10 a decade ago to around 30 today. A great principles framework does not depend on the number of core technologies. It will make our field more understandable and approachable, for insiders as well as outsiders.
- **Curriculum complexity:** The Computing Curriculum 2001 (CC2001) committee took 2 years to come to a consensus on a set of topics that could be covered in the time allocated to the core in most curricula. The typical first course is now organized around object-oriented programming (with Java or C++ as the vehicle), a practice that has spread to the Advanced Placement (AP) curriculum in high schools. Many novices find the language details intricate and the course very difficult; they feel under extreme pressure to perform. There are numerous reports of high dropout rates in the first course (35 to 50 percent) and of widespread cheating and plagiarism on programming assignments. The "trauma of the first language" subverts the ideal of multi-lingual computing professionals. A great principles framework can help us resolve these incongruities.
- **Computing Practices.** The competence of computing professionals is judged by their ability to perform effectively with their customers. Computing education currently emphasizes concepts at the expense of competent practices. A great principles framework offers a new balance between concepts and practice.
- **Public Image:** Despite many years of our trying to broaden our image, computing is still widely perceived as a programmer's field. Computer Science departments have added to this perception by making an industry-strength language the centerpiece of introductory courses. Many outsiders wonder whether CS departments will eventually disappear as the technology evolves and other fields take over as the main contributors of new computing technology. A great principles framework can dispel these misconceptions.

The view of the field to be discussed here identifies a fundamental base of principles for all the core technologies and a fundamental set of practices that mark the computing professional. The resulting framework is simple to grasp and provides a stable context for the core technologies. It also suggests ways to organize curriculum to overcome the curriculum difficulties noted above. It puts computing on an equal basis with other traditional science and engineering fields.

* Peter Denning (pjd@nps.navy.mil) is Chairman of the Computer Science Department and Director of the Cebrowski Institute for Information Innovation and Superiority at NPS.

SIGCSE'04, March 3-7, 2004, Norfolk, Virginia, USA.

2. EXAMPLES FROM OTHER FIELDS

Computer science was born in the mid 1940s with the construction of the first electronic computers. In just 60 years, computing has come to occupy a central place in science, engineering, business, and everyday life. Many whose lives are touched by computing want to know how computers work and how dangerous or risky they are; some want to make a profession from working with computers; and most everyone asks for an uncomplicated framework for understanding this complex field. Similar questions have been faced in other fields such as physics, life sciences, and astronomy. Can we learn from them how to answer such questions in a compact, compelling, and coherent way?

The mature sciences such as physics, biology, and astronomy portray themselves with a principles-based approach. Each builds rich structures from a small set of great principles. The great principles are simple ideas that affect the entire field. Examples of this approach are *Lectures in Physics* by Richard Feynman [5], *The Joy of Science* by Robert Hazen and James Trefil [7], and *Cosmos* by Carl Sagan [9]. Newcomers find a principles-based approach to be much more rewarding because it promotes understanding from the beginning and shows how the science transcends particular technologies.

A great principles approach for computing needs to deal with mechanics (how computations work), design (how we organize computations), and practices (what we must be competent at when we build computations).

3. MECHANICS

In the 1950s, our field's founders portrayed their young science as a set of core technologies that supported application domains. They listed their core technologies as algorithms, numerical methods, computation models, compilers, languages, and logic circuits. Over the next 30 years, our field added a few more: operating systems, information retrieval, databases, networks, artificial intelligence, human computer interaction, and software engineering. The 1989 ACM/IEEE report, "Computing as a Discipline," listed 9 core technology areas [3]. Since then, the total number of core technology areas has tripled (see Table 1). Today, learning the mechanics of these technologies and their hundreds of possible direct interactions has become a daunting challenge.

Table 1. Core Technologies of Computing

algorithms	management info systems
artificial intelligence	natural language processing
compilers	networks
computational science	operating systems
computer architecture	parallel computation
data mining	programming languages
data security	real-time systems
data structures	robots
databases	scientific computation
decision support systems	software engineering
distributed computation	supercomputers
e-commerce	virtual reality
graphics	vision
human computer interaction	visualization
information retrieval	workflow

In an effort to stem "curriculum bloat" from this growth, the CC2001 report emphasizes the ideas at the "intersection" of the core technologies [4]. Two books that popularize computing focus on the "great ideas" embodied in these areas [1,8]. None of these authors discusses which ideas are fundamental principles of all the core technologies. A list of core technologies does little to convey the great principles of computing.

Locating the fundamental principles of the field looks, therefore, to be a very attractive project. It calls to mind a picture in which the principles are the foundation of a pantheon with one pillar for each great principle. Unfortunately, as we shall soon see, such a picture is an unsatisfactory portrayal of computing.

Our very first question is: How shall we express our principles? It seems like we are looking for declarative statements such as

"The Turing machine is a universal model of computation."

"All information can be encoded as strings of bits."

"The number of bits in a message source is given by its entropy."

But this quickly becomes contentious. Some people argue over the definitions of terms such as computation, information, or message sources. Others ask whether some of the words ought to be qualified -- such as algorithmic computation, physically represented bits, or discrete message sources. Still others ask why these statements are singled out and not others, such as "Every function imposes a lower-bound running time on all algorithms that compute it." Most everyone demands statements of obvious relevance to the familiar core technologies. But they wrestle over the selection criteria for principle statements, such as universality, recurrence, invariance, action orientation, utility for prediction, or scope of consequences.

How do other fields express their principles? Physicists name key phenomena like photons, electrons, quarks, quantum wave function, relativity, and energy conservation. Astronomers name planets, stars, galaxies, Hubble shift, and black holes. Thermodynamicists name entropy, first law, second law, and Carnot cycle. Biologists name phylogeny, ontogeny, DNA, and enzymes. Each of their terms is actually the *title of a story*! The principles of a field are actually a set of interwoven stories about the structure and behavior of field elements. They are the names of chapters in books about the field [5,7,9].

Astronomy, thermodynamics, and physics use the term *mechanics* for the part of their fields dealing with the behavior and structure of components -- the so-called "cause-and-effect relationships." For example, Celestial Mechanics deals with the motions of heavenly bodies; Statistical Mechanics with the macro behavior of physical systems comprising large numbers of small particles; Quantum Mechanics with wave behaviors of subatomic particles; Rigid-Body Mechanics with the balance of forces within and between connected objects. I adopt this term for computing.

Computing Mechanics deals with the structure and operation of computations -- with the universally occurring phenomena that appear in computational processes and hardware and software components. It does so with stories for algorithm, Turing machine, grammar, message entropy, process, protocol stack, naming, caching, machine learning, virtual machine, and more. The stories group into the five categories of computation, communication, coordination, automation, and recollection (see Table 2). Every core technology expresses all five in its own way.

The lines between these categories are blurry. For example, the Internet protocol stack is an element of both communication and coordination; naming, encoding, and caching are elements of

Table 2: The Five Windows of Computing Mechanics

Window	Central Concern	Principal Stories
Computation	What can be computed and how; limits of computing.	Algorithm, data structure, automata, languages, Turing machine, universal computer, Turing complexity, Chaitin complexity, self reference, approximations, heuristics, non-computability, translations, compilations, physical realizations
Communication	Sending messages from one point to another.	Data transmission, Shannon entropy, encoding to medium, channel capacity, noise suppression, error correcting codes, end-to-end-error correction, Huffman and Reed-Solomon codes, file compression, cryptography, packet networking
Coordination	Multiple entities cooperating toward a single result.	Human-to-human (action loops, workflows as supported by communicating computers), human-computer (interface, input, output, response time, data visualization); computer-computer (concurrency control, races, synchronization, deadlock, serializability, atomic actions)
Automation	Performing cognitive tasks by computer.	Simulation and machine performance of cognitive tasks, philosophical distinctions about automation, expertise and expert systems, enhancement of intelligence, Turing tests, machine learning and recognition, bionics
Recollection	Storing and retrieving information.	Hierarchies of storage, locality of reference, caching, address space and mapping, bindings, naming, sharing, thrashing, retrieval by name, retrieval by content

communication and recollection; concurrency is an element of computation and coordination. Therefore, I found it better to view the categories as *windows* into computing mechanics (Figure 1). Although the views through the edges of windows overlap, the view through the centers is distinctive.

4. DESIGN

Computing Mechanics does not exhaust all the principles used in our field. Computing professionals follow principles of design that enable them to harness mechanics in the service of users and customers. Five concerns drive the design principles:

- **Simplicity:** Various forms of abstraction and structure that overcome the apparent complexity of the applications.
- **Performance:** predicting throughput, response time, bottlenecks; capacity planning.
- **Resilience:** reliability, redundancy, forward error correction, retransmission, majority voting, recovery, checkpoint, integrity, system trust.
- **Evolvability:** adapting to changes in function and scale.
- **Security:** access control, secrecy, privacy, authentication, integrity, safety.

The design principles themselves include abstraction, information hiding, modules, separate compilation, packages, version control, divide-and-conquer, functional levels, layering, hierarchy, separation of concerns, reuse, encapsulation, interfaces, and virtual machines. These principles are *conventions* that we collectively have found to lead consistently to dependable and useful programs, systems, and applications. These conventions are practiced within constraints of cost, schedule, compatibility, and usability.

Design is not the same in computing as it is in other fields. In computing we design *abstract objects* that *perform actions*. Other fields use abstraction to explain or to organize tangible objects. Since design tells us about arrangements of basic components, design sits above mechanics in our picture of the field.

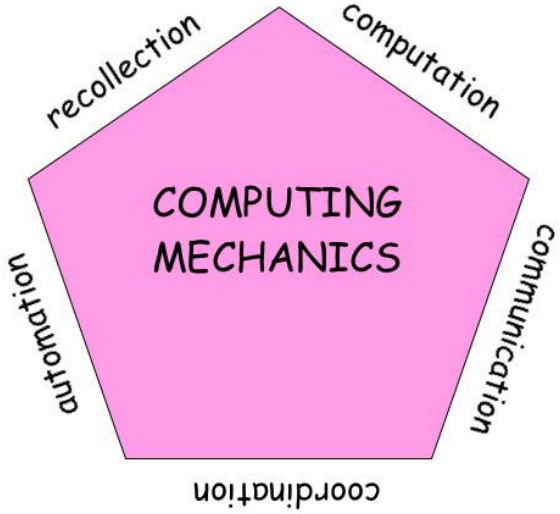

Figure 1. The five windows.

5. COMPUTING PRACTICES

Our picture of computing is needs more than mechanics and design. It needs an account of the computing practices that characterize our skills as professionals. Our competence is judged not by our ability to explain principles, but by the quality of what we do. We found five main categories of computing practice:

Programming -- Using programming languages to build software systems that meet specifications created in cooperation with the users of those systems. Computing professionals must be multilingual, facile with the numerous programming languages, each attuned to its own strategies for solving problems.

- **Engineering Systems --** Designing and constructing systems of software and hardware components running on servers connected by networks. These practices include a design component concerned with organizing a system to produce valuable and tangible benefits for the users; and an engineering component concerned with the modules, abstractions, revisions, design decisions, and risks in the system; and an operations component concerned with configuration, management, and maintenance of the system. High levels of skill are needed for large programmed systems encompassing thousands of modules and millions of lines of code.
- **Modeling and validation --** Building models of systems to make predictions about their behavior under various conditions; and designing experiments to validate algorithms and systems.
- **Innovating --** Bringing about lasting changes in the ways groups and communities operate by exercising technical leadership. Innovators watch for and analyze opportunities, listen to customers, formulate offers customers see as valuable, and manage commitments to deliver the promised results. Innovators are history-makers who have strong historical sensibilities.
- **Applying --** Working with practitioners in application domains to produce computing systems that support their work. Working with other computing professionals to produce core technologies that support attributes common to many applications. Working according to the standards of practice, conduct, and ethics to increase public trust in the profession and its members.

It is best to think of practices and principles in an endless cycle of mutual reinforcement: our practice is shaped by principles, and our perceptions of principles are shaped by practice.

Our portrait is now complete (see Figure 2). It consists of computing mechanics (the laws and universal recurrences that govern the operation of computations), design principles (the conventions for designing computations), computing practices (the standard ways of building and deploying computing systems), and core technologies (organized around shared attributes of application domains). Although not shown in the figure, the entire framework floats in a rich contextual sea of application domains, collectively exerting strong influences on core technologies, design, and practice. Each of the four levels at which we act as designers and users of computation is also a domain of practice with its own supporting technologies (see Table 3). Each level of the picture has a characteristic question that justifies its place in the hierarchy and exposes the integral role of practices.

6. AN IMPLEMENTATION AT NPS

At the Naval Postgraduate School, the Computer Science Department offers a graduate curriculum leading to MS in CS. The students are mostly officers of the US Navy, Marine Corps, Army, and Air Force, and of similar military services in 53 other countries. They are professional leaders who are seeking a solid grounding in the principles of computing and who will be project leaders, program managers, strategists, change agents, and innovators. Most are in mid-career, 5-10 years since their BS degrees, and some are seeking formal education in CS for the first time. They attend for 2 years and take courses continuously for 8 quarters. Their first year establishes a base in CS and their second develops depth in a track supporting a required master's thesis. (Although we have a few PhD students, the bulk of student support for our $18M of sponsored research comes from MS theses.)

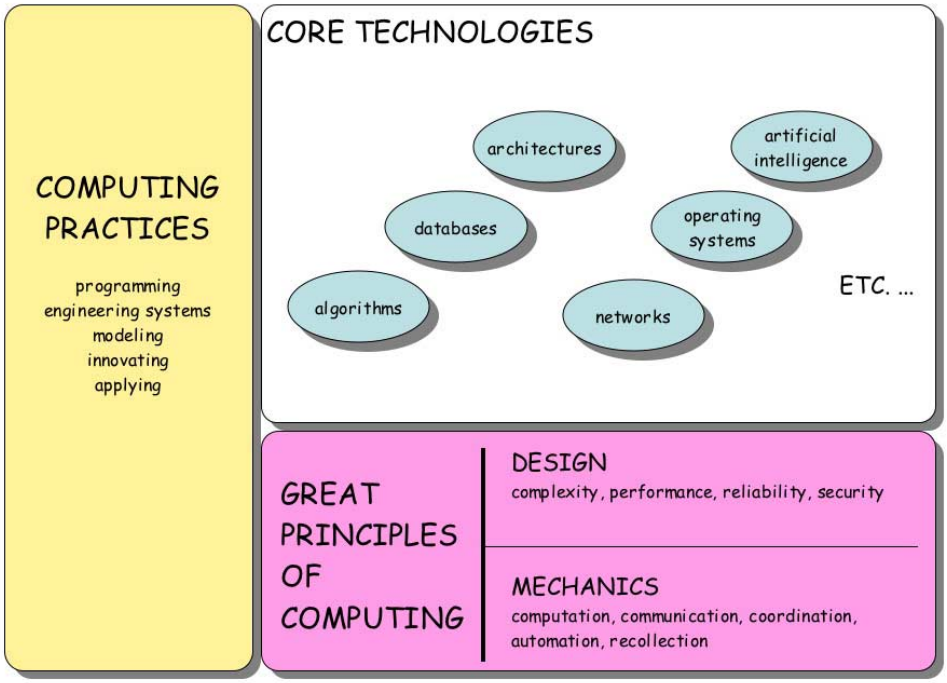

Figure 2. Principles-Based Portrait of Computing

Table 3. Levels of Action in Computing Practices

Levels	Central Questions	Example Technologies
Application Domains	How do we work with others to design computing that serves them?	Supercomputers, grid computing, graphics design, interfaces, ...
Core Technologies	How do we design computations that support common elements across applications?	Algorithms, databases, networks, operating systems, HCI, AI, …
Design Principles	How do we organize ourselves and our thinking when designing computations?	Design tools, object-oriented programming, layering, virtual machines, authentication, …
Computing Mechanics	What are computing machines doing?	Logic simulators, protocol stack, workflow, expert systems, virtual memory, …

We implemented the great principles framework in our context as follows.

- We designed a new course, Great Principles of Computing Technology, which is the opening course for the curriculum.
- We matched it with a final, integrative capstone course that considers contemporary issues interpreted in the light of the great principles.
- We revised our core courses so that they build on the principles introduced at the start, rather than reinventing them from scratch. The content of these courses is consistent with the core content recommendations of the ACM 2001 Curriculum.
- Created a set of computing practices courses for programming, engineering of systems, modeling, and innovating.

The computing practices courses are as follows.

- *Programming Practices* are 3 courses. The first two cover object programming using Java and a rich repertoire of actions that programmers can take. They are similar to standard versions of the first and second CS courses, except they unabashedly focus on programming practice and not on the development of concepts. The third PP course is a survey of languages, introducing CS students to their responsibility to be multilingual.
- *System Practices* is one course, a modified introduction to software engineering, emphasizing the practices of putting systems together from modules, tools for software development, assembly, testing, and risk assessment.
- *Modeling Practices* is a new course created in cooperating with the Operations Research department. It teaches how to use basic queueing tools for predicting throughput and response time of networks of computers, how to set up experiments and discrete simulation, and how to gather and present data.
- *Technology, Innovation, and Leadership* teaches the practices of innovation in computing. Innovation means a change of practices in a community that produces new value for all members. We use relevant case studies to teach the process of innovation, and language-action philosophy to teach the embodiment of the practice of innovation. This course is an excellent fit for our officers, who are expected to be innovators and change agents.

Early in our deliberations, we considered a more radical reshaping, in which we would create full courses (or sequences) for the five windows of computing mechanics and the design principles. We decided against this because our graduates will be practicing in a world of technologies and will be experts in some of the technologies. There are well-established communities of practice around these technologies. We therefore retained the technology-oriented structure in our core courses.

7. REFLECTIONS

By aligning with traditions of other science fields, a curriculum organized around great principles and practices promotes a greater understanding of the science and engineering behind information technology. It improves our students' abilities to discuss risks, benefits, capabilities, and limitations with technicians and with people outside the field. It recognizes that computing is action oriented and has many customers, and that the context in which computing is used is as important as the mechanics of computing. It also clarifies professional competence, which depends on dexterity with mechanics, design, practices, core technologies, and applications.

In teaching and writing about the elements of computing mechanics, we emphasize the way the principles developed over time and what has made them so durable and ubiquitous. For example, Turing machines are not a obscure mathematician's conceit, but are the essence of all computing and the vehicle for seeing why a great many practical problems are intractable. Error correcting codes are not amusing tricks in discrete mathematics, they are essential to making all practical communication systems work in the perpetual presence of noise.

We also emphasize how the context of use has shaped our practices and our perceptions of principles. For example, the different styles of the different programming languages flow out of the application domains that inspired them. The controversies about the limits of machine intelligence cannot be understood without reference to cognitive science and linguistic philosophy. The debates about software development process turn on different ways that engineers and architects use the term "design": does it mean a highly methodical engineering process, or does it mean agile processes promoting systems built for customer satisfaction, artistry, good taste, simplicity, and elegance?

A great principles framework may offer a new approach to the endlessly vexing issue of the role of programming in our field. Computer science is seen by many as "programming exalted." For many years, we have tried to organize our first courses in

computing (CS1, CS2, CS3) around important concepts of algorithms and data structures integrated with programming. Over the years, the growing complexity of the industrial-strength languages has turned CS1 and CS2 into grueling ordeals for many students. Overwhelmed with the mechanics of programming and an almost-obsessive drive to pass these courses, many have resorted to cheating and plagiarism; dropout rates typically run at 35 to 50 percent. Many do not experience the joy of computing: the interplay between the great principles, the ways of algorithmic thinking, and the solutions of interesting problems. The great principles framework creates a distinction between the principles and the practices of using the principles. Programming practices cannot crowded out the learning of principles in this context. Mark Guzdial's recent experiment at Georgia Tech demonstrated this in a new way with a pilot first course on "media computation" that introduced computing in a world of audio and visual media. The course attracted more women than men, had a dropout rate of under 1.5%, and produced enormous interest in the rest of the CS curriculum [6].

Some have asked about the relationship between the great principles framework and the matrix of 9 core technologies versus the 3 processes of theory, abstraction, and design discussed in "Computing as a Discipline" in 1989 [3]. The older model is based on technologies, rather than enduring principles. It mentions practices as lab sections and projects but not as distinguishing characteristics of computing professionals and the bases of professional competence. Theory, abstraction, and design were inherited from mathematics, science, and engineering, but have merged together as our field matures. In the great principles framework, theory appears in every window, abstraction is a design principle, and design has its own layer.

In 1989 we were deeply concerned about the outside perception that we are a field of programmers. We argued that our field has much more breadth and depth than programming. The record of history shows that our argument was not persuasive, not even among ourselves. Programming still dominates the first course, even more so than in 1989, and outsiders still see us as programmers. The 1989 framework was ineffective at helping us escape this millstone. We hope that the great principles framework will be more successful.

Some have asked about the relationship between the great principles framework and the recommendations of Curriculum 2001 [4]. They are complementary with great potential synergy. The great principles framework offers a new way to organize the body of knowledge, demonstrating underlying stability in the face of rapid technological change. It distinguishes principles and practice, permitting individual departments to find a balance where programming does not overwhelm principles. It distinguishes the social conventions of design from laws and recurrences of computing mechanics. It reveals two gaps in CC2001. One is the lack of a modeling practices course to support the important aspect of experimental computer science. The other is a lack of support for the important objective that computing professionals be capable of producing innovations for their customers.

A final question concerns how existing trends will shape our future perceptions of principles. Examples of important trends are pervasive and mobile computing, context-aware computers, self-healing systems, and hyper-computing (beyond Turing machines). These trends will exploit and enrich the give windows. For example, pervasive and mobile computing is a form of distributed computing, which is an aspect of both computation and coordination; context-awareness is an aspect of automation; self-healingness extends design principles for reliability into a world of ubiquitous computers; hyper-computing is a debate within computation. New practices around these new issues will displace older practices.

It is time for us to make ourselves known by saying our mechanics, our design principles, and our practices. It is time to stop hiding the enormous depth and breadth of our field.

8. READINGS

[1] Biermann, Alan. *Great Ideas in Computer Science* (2nd Ed.). MIT Press (1997).

[2] Denning, Peter. "Great Principles of Computing." *ACM Communications 46*, 11 (Nov 2003), to appear.

[3] Denning, Peter et al. "Computing as a discipline". *ACM Communications 32*, 1 (Jan 1989), 9-23.

[4] *Curriculum 2001 Final Report.* <computer.org/education/cc2001/final/>

[5] Feynman, Richard. *Lectures in Physics.* Addison-Wesley (1970).

[6] Guzdial, Mark, and Elliot Solloway. "Computer science is more important than calculus: The challenge of living up to our potential." Inroads (ACM SIGCSE Bulletin), June 2003, 5-8.

[7] Hazen, Robert, and James Trefil. *Science Matters.* Anchor (1991).

[8] Hillis, Danny. *The Pattern on the Stone.* Basic Books (1999).

[9] Sagan, Carl. *Cosmos.* Random House (2002).

9. BIO

PETER J. DENNING is Chairman of the Computer Science Department at the Naval Postgraduate School in Monterey, California. He is also director of the Cebrowski Institute, a research center for information innovation and superiority. He came to NPS in 2002 from George Mason University, where he served as vice provost, associate dean, and chair of the CS Department. He was founding director of RIACS at the NASA Ames Research Center, co-founder of CSNET, and head of the computer science department at Purdue. He received a PhD from MIT and BEE from Manhattan College. He invented the working set model for program behavior and helped establish virtual memory as a permanent part of operating systems. He co-invented operational analysis, an approach to computer system performance prediction. He was president of the Association for Computing Machinery 1980-82. He chaired the ACM publications board 1992-98 where he led the development of the ACM digital library, and chaired the ACM Education Board 1998-2003. He has published 7 books and 300 articles on computers, networks, and their operating systems, and is working on 3 more books. In 2002, he was named one of the top 5 best teachers at George Mason University and the best teacher in the School of Information Technology and Engineering. In 2003, he received one of Virginia's 10 outstanding faculty awards. He holds three honorary degrees, three professional society fellowships, two best-paper awards, three distinguished service awards, the ACM Outstanding Contribution Award, the ACM SIGCSE Outstanding CS Educator Award, and the prestigious ACM Karl Karlstrom Outstanding Educator Award.