CHAPTER 1

Software and Software Engineering

Learning Outcomes

In this chapter you will learn:

- Historical origins of software, software engineering, and the related disciplines
- Types of software applications and system software
- Stakeholders in software products
- Three Ps in software engineering
- Software engineering as a discipline and its code of ethics and professional practice
- Recurring concepts in software engineering and the desirable software capabilities
- Pioneers in software engineering and their main contributions

Software

- Software is omnipresent in the lives of billions of human beings.
- Software is an important component of the emerging knowledge based service economy.
- Software or computer software consists of the **computer program and its related documentation.**
- The word ‘software’ was coined by John Tukey in 1958.
- The theory behind the concept of a computer program were established by **Alan Turing** in the 1930’s.
- The concept of a program as a sequence of steps to solve a problem is a realization of the concept of **algorithm** which was introduced by **Muhammad Al-Khawarezmi**, a 9th century mathematician.

Software (2)

- An algorithm became concrete when it was programmed by **Ada Lovelace, the first computer programmer**.
- Software = computer program + documentation

Software (3)

- **Computer program**
  - instructions that perform certain tasks on computer hardware.
  - can be written at different levels of closeness to the hardware.
  - Low level to high level languages
- **Documentation**
  - plays a crucial role in the success of software.
  - of interest to the people using the software or to the people developing and maintaining it.
  - User manuals, installation procedures, and operating manuals are written mainly for the software users.

The software crisis (1967)

- Characterized by the inability of existing techniques, tools and processes to deal with the increasing complexity of the needed software.
- Main reasons: **complexity of the software, changing and misunderstanding of requirements and the lack of tools and skilled professionals.**
- Produced software - **low quality, hardly maintainable, and not meeting the stakeholder’s requirements**.
- Software projects were most of the time running **over-budget and over-time**, and many never delivered a functioning product.

The crisis persists ...

- According to the Standish Group, a software market research firm, **17% of software projects were complete failures in 2002**. Moreover, **50 % of projects were not completed within the planned schedule, ran over-budget, or were missing some of the required features.**
- There are many concerns about the quality and reliability of the software we use. Existing software is plagued with **millions of defects**. Some of these defects are known and have already been detected, others are yet to be uncovered. These defects have caused **many disasters leading to financial losses, physical harm to humans and life threatening situations**.

The crisis persists ...

- Tools, techniques, standards and appropriate software engineering education programs at all levels are needed.
- In the US alone, it was reported in 2004 that about 750 thousand software engineers are employed, compared to about 1.5 million practitioners in all other engineering disciplines.
- It was also reported that **most software practitioners do not hold degrees in software engineering**. Currently, most people working as software engineers hold either a degree in computer science or computer engineering.

Software engineering

- Software engineering is a term that was coined during the NATO Software Engineering conference held in Garmisch, Germany, in October 1968. The term was introduced by the conference chairman Friedrich Bauer.

Software engineering

- **‘the application of a disciplined approach for the development and maintenance of computer software’**
- **‘deals with the establishment and use of sound engineering principles to economically obtain software that is reliable and works efficiently on real machines’ (by IEEE)**
- **‘encompasses the use of tools, techniques and methods that are useful during the execution of the steps needed for the development of the software and its future maintenance’**

Types of software

- System software
  - operating systems, language compilers, assemblers, device drivers, debuggers, and networking tools and software
- Application software or end-user software

Figure 1.1 A layered view of hardware and software.

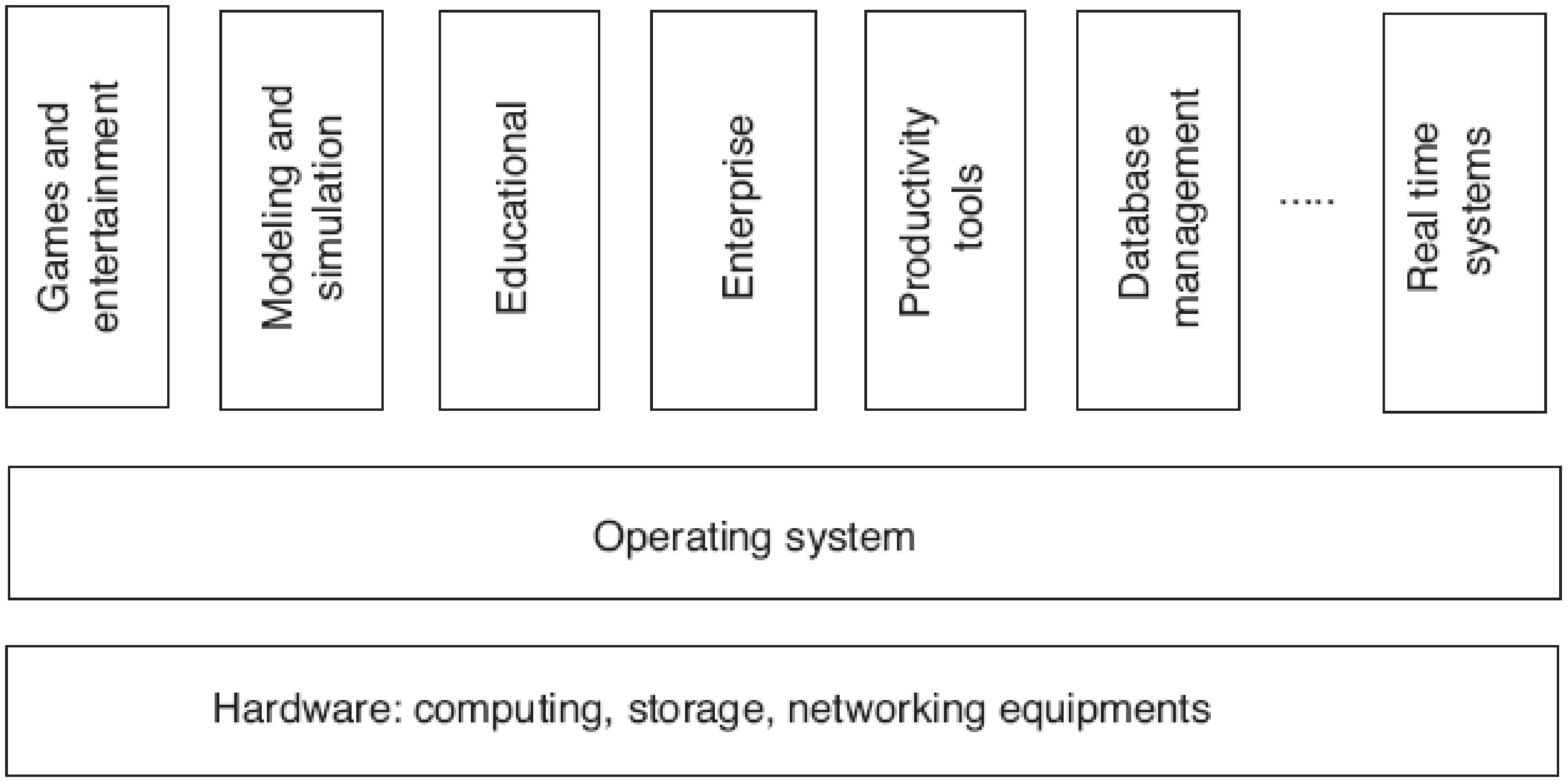


Figure 1.1 A layered view of hardware and software

Types of application software

- **Games and entertainment software:** games for handheld devices including mobile phones, PC or stand alone games, and distributed collaborative games.
- **Intelligent software:** specialized domain specific expert systems, mobile agent systems, learning systems, robot vision software, business decision and intelligence software, and mining software.
- **Modeling and simulation software:** domain specific modeling and simulation packages for military, financial, medical, educational and training uses.

Types of application software (2)

- **Real-time software:** industrial plant monitoring and control systems, missile control systems, air traffic control systems, telephony software, and network security software like firewalls and intrusion detection systems.
- **Embedded software:** home appliance controllers, mobile phone software, and vehicle controllers.
- **Productivity (information worker) software:** tools that implement proven methods and techniques to help specific types of users doing their tasks with ease and high productivity.
- **Enterprise software:** business workflow management software, customer relation management software, and supply chain management software.

Types of application software (3)

- **Web-based software:** content management software, web publishing software, electronic commerce software, web services, web portal software, and web browsers.
- **Educational software:** school and university management software, online and distance learning software, training management software, and educational software for children.
- **Multimedia software:** video, image and sound editing and management software, 3D and animation software, and virtual reality software.
- **Domain-specific software:** banking, finance and stocks, accounting, medical, airline reservation, hospital management, and human resource management software.

3 Generic stages in software development and maintenance

- Structured, disciplined approach
- Aiming at enhancing quality and dealing with complexity

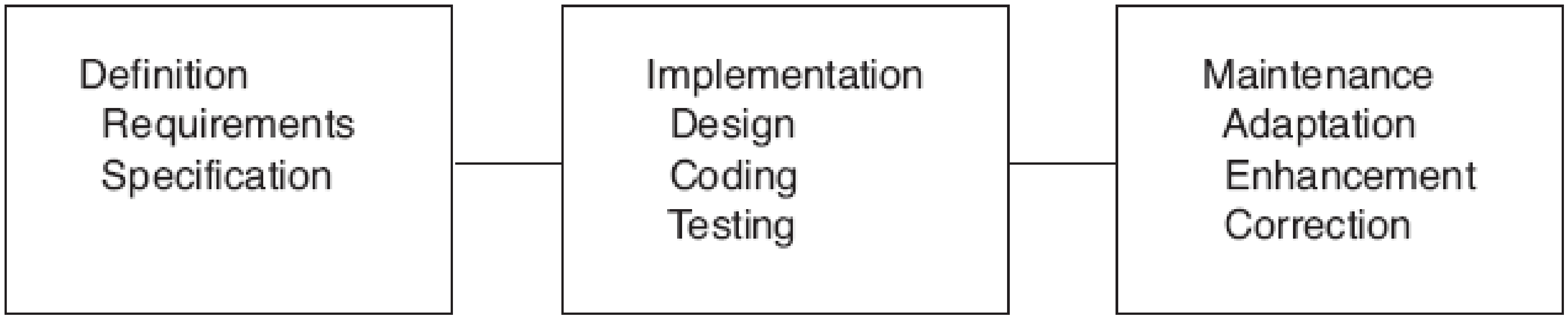


Figure 1.2 The three generic stages for software development and maintenance

Software errors

- Discovered and others are yet to be uncovered
- 25% are definition errors (requirements & specification)
- 25% design errors, 10% coding errors
- It costs more to fix a definition error in the maintenance phase – better discover them early!

The software triad

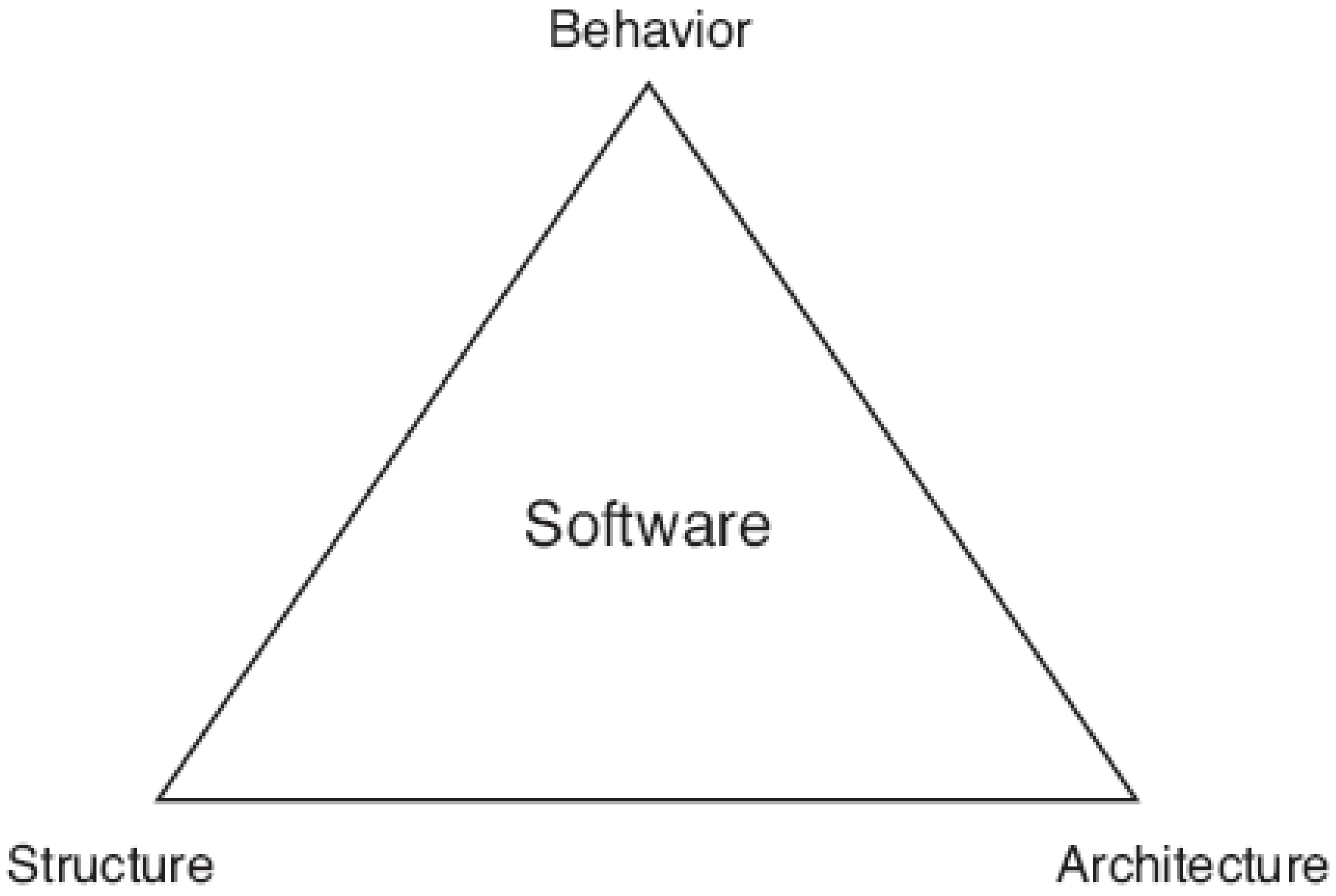


Figure 1.3 The software triad

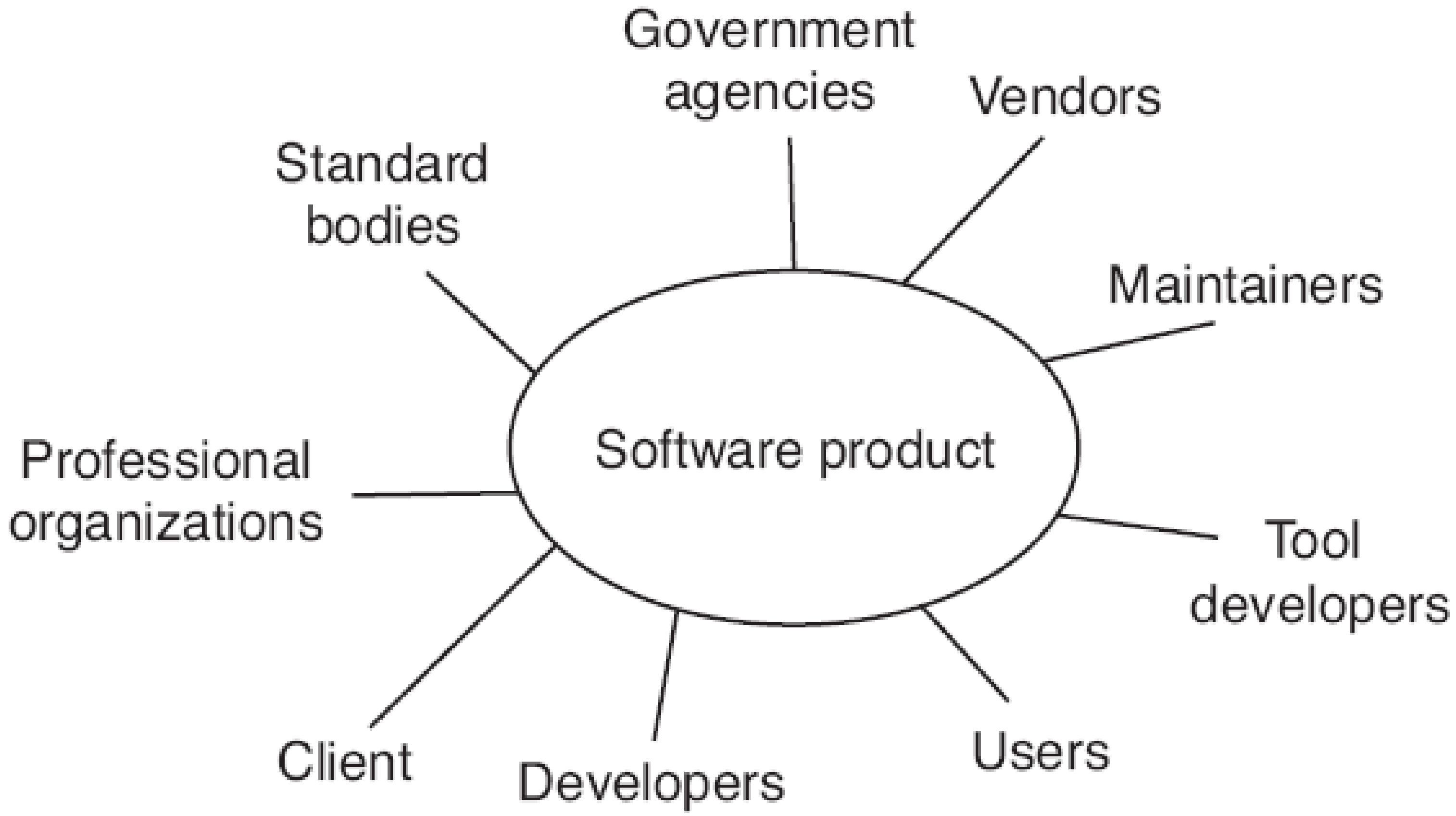


Figure 1.4 Stakeholders in a software product

Table 1.1 The stakeholders in a banking software

Stakeholder	Description
Client	Bank (paying for the software)
Developer	Software Company X (selected to develop the software)
Users	Bank customers having online access
Technical writers	Can be employees of the bank or the software company
Government bodies	Banking regulations related to privacy protection
Standard bodies	Basel II for IT risk management
Professional organizations	Banker's association
Software maintainers	Can be the bank, the software company, or a third party

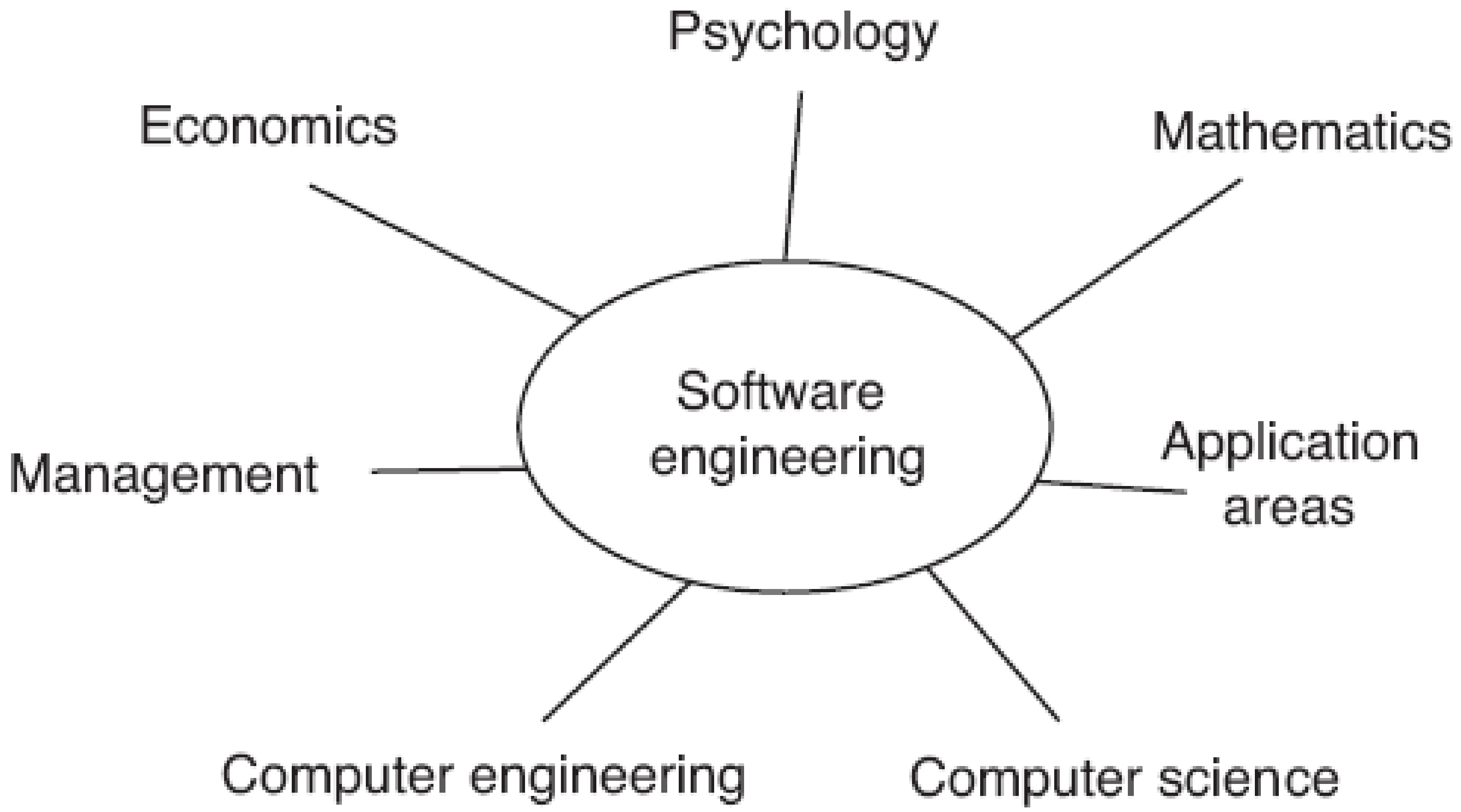
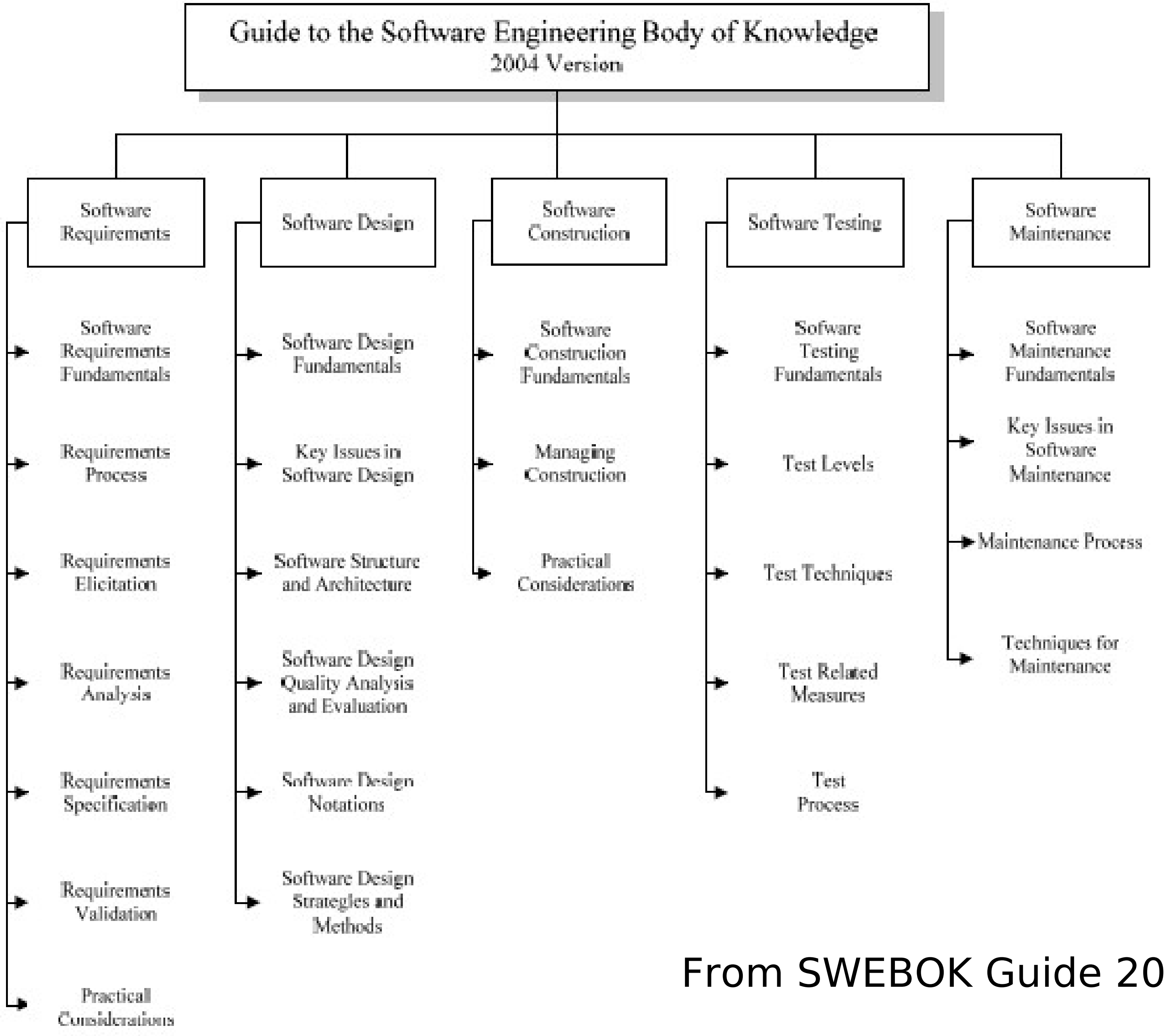


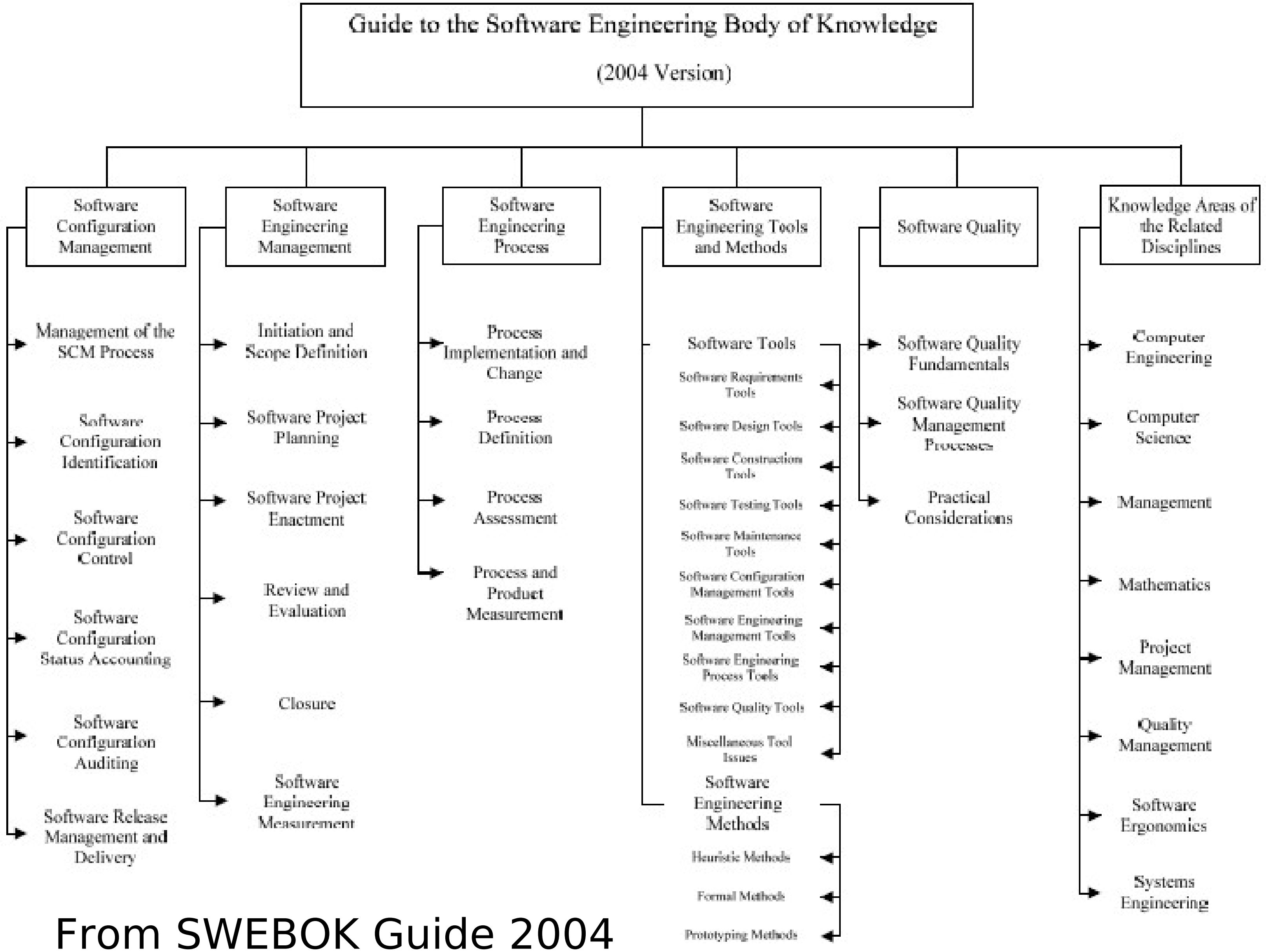
Figure 1.5 Software engineering and related disciplines

Education, Training and Certification

- Young discipline – B Sc in Software Engineering only in the last decade (1990’s)
- Body of knowledge
- IEEE Certified software development professional (CSDP)


Guide to the Software Engineering Body of Knowledge
2004 Version
Software Requirements
Software Requirements Fundamentals
Requirements Process
Requirements Elicitation
Requirements Analysis
Requirements Specification
Requirements Validation
Practical Considerations
Software Design
Software Design Fundamentals
Key Issues in Software Design
Software Structure and Architecture
Software Design Quality Analysis and Evaluation
Software Design Notations
Software Design Strategies and Methods
Software Construction
Software Construction Fundamentals
Managing Construction
Practical Considerations
Software Testing
Sofware Testing Fundamentals
Test Levels
Test Techniques
Test Related Measures
Test Process
Software Maintenance
Software Maintenance Fundamentals
Key Issues in Software Maintenance
Maintenance Process
Techniques for Maintenance

From SWEBOK Guide 2004


Guide to the Software Engineering Body of Knowledge
(2004 Version)
Software Configuration Management
Management of the SCM Process
Software Configuration Identification
Software Configuration Control
Software Configuration Status Accounting
Software Configuration Auditing
Software Release Management and Delivery
Software Engineering Management
Initiation and Scope Definition
Software Project Planning
Software Project Enactment
Review and Evaluation
Closure
Software Engineering Measurement
Software Engineering Process
Process Implementation and Change
Process Definition
Process Assessment
Process and Product Measurement
Software Engineering Tools and Methods
Software Tools
Software Requirements Tools
Software Design Tools
Software Construction Tools
Software Testing Tools
Software Maintenance Tools
Software Configuration Management Tools
Software Engineering Management Tools
Software Engineering Process Tools
Software Quality Tools
Miscellaneous Tool Issues
Software Engineering Methods
Heuristic Methods
Formal Methods
Prototyping Methods
Software Quality
Software Quality Fundamentals
Software Quality Management Processes
Practical Considerations
Knowledge Areas of the Related Disciplines
Computer Engineering
Computer Science
Management
Mathematics
Project Management
Quality Management
Software Ergonomics
Systems Engineering

From SWEBOK Guide 2004

Code of ethics and professional practice for software engineers

Public:

- Software engineers shall act consistently with the public interest.
- Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.

Product:

- Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

Profession:

- Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

Code of ethics and professional practice for software engineers (2)

Peers and self:

- Software engineers shall maintain integrity and independence in their professional judgment.
- Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
- Software engineers shall be fair to and supportive of their colleagues.
- Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

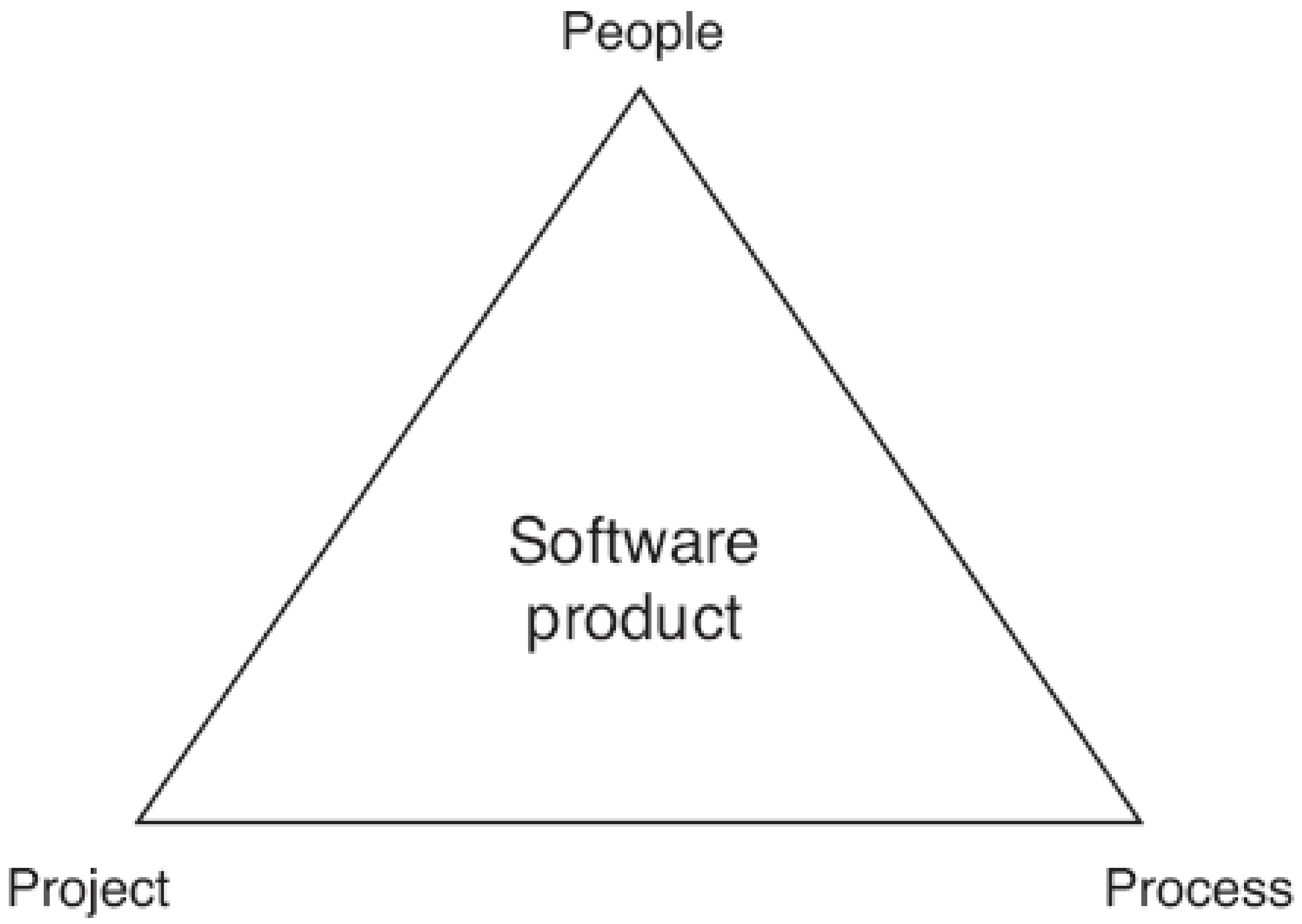


Figure 1.6 The three cornerstones of a software product

Desirable software abilities – user-centered

- **Availability** is the degree to which the software system is available when its services are required. It can be quantified as the ratio of the actual availability of the software services to the expected availability during the same period.
- **Correctness** is the degree to which the software meets its requirements specifications. Correctness is affected positively by the completeness, consistency, and traceability of the software. Accuracy is a qualitative assessment of correctness.
- **Efficiency** is the degree to which the software system performs its functions using the least amount of computational and memory resources.
- **Integrity** is the degree to which the software system prevents unauthorized access to information or programs. Both integrity and availability contribute to the security of the software system.

Desirable software abilities – user-centered (2)

- **Reliability** is the ability of a software system to perform its function under stated conditions and for a specified period of time. Reliability can be quantified using the mean time between failures. Reliability is positively affected by error recoverability and fault tolerance, modularity and simplicity, and other quality factors.
- **Scalability** is the ability of the software system to handle a growing number of user requests up to a certain specified limit.
- **Usability** is the degree of ease with which the software system can be used. Usability is positively affected by **learnability, readability, understandability**, and other quality factors

Desirable software abilities – developer-centered

- **Flexibility** is the measure of how easily the software system can be extended or expanded. Flexibility is positively affected by the simplicity, generality, and modularity of the software, and other quality factors.
- **Interoperability** is ability of the software system to exchange information with other external systems.
- **Maintainability** is the measure of how easily the software system can be modified to fix errors, to improve its functions or to adapt it to different environments. Maintainability is positively affected by adaptability, simplicity, modularity and traceability, and other quality factors.

Desirable software abilities – developer-centered (2)

- **Portability** is the ease with which the software system can be moved to a different software or hardware environment or platform. Portability is positively affected by generality and modularity, and other quality factors.
- **Reusability** is the degree of ease with which a software component can be reused in the same software system or used to build new systems. Reusability is positively affected by generality and modularity, and other quality factors.
- **Testability** is the measure of how easily test cases can be selected and executed to test the software. Testability is positively affected by **modularity, simplicity** and **traceability**, and other quality factors.

Pioneers in software and software engineering

- **Alan Kay** is known for his pioneering work on window based graphical user interfaces. He is also known for coining the term object-oriented programming in 1966.
- **Ali Mili** contributed to the formalization of software fault tolerance, now a major concern for developing secure software systems.
- **Barry Boehm** contributed to the area of software engineering economics and software metrics. He has also introduced the spiral model for software development.
- **Bill Joy** contributed to the development of the Unix operating system and the Java programming language.
- **Brian Kernighan and Dennis Ritchie** were instrumental in developing the Unix operating system and the C programming language.

Pioneers in software and software engineering (2)

- **C.A.R. Hoare** introduced the concepts of assertions and program proof of correctness, designed and analyzed well-known algorithms, and developed communicating sequential processes, a formal language for the specification of concurrent processes.
- **David Parnas** introduced the concepts of information hiding, software interfaces and software modularity. He has also contributed to software engineering education and the ethical responsibilities of a software engineer.
- **Donald Knuth** is known for the design of many well-known computer algorithms and the use of rigorous mathematical techniques for the formal analysis of the complexity of algorithms.
- **Edsger Dijkstra** introduced the concept of structured programming and carried out in-depth studies of the problems of concurrency and synchronization needed in complex distributed systems.

Pioneers in software and software engineering (3)

- **Erich Gamma** with Richard Helm, Ralph Johnson and John Vlissides introduced the concept of object oriented design patterns to facilitate software design reuse.
- **Fred Brooks** contributed to the development of the OS/360 operating system software. He is also known for introducing the mythical man-month in software project management.
- **Friedrich Bauer** introduced the use of stacks for the evaluation of expressions in programming language compilers. He also contributed to the development of the ALGOL programming language. Moreover, he coined the term software engineering in the NATO conference held in Germany in 1968.
- **Grace Hopper** contributed extensively to the first compiler and the COBOL programming language in the 1959’s.

Pioneers in software and software engineering (4)

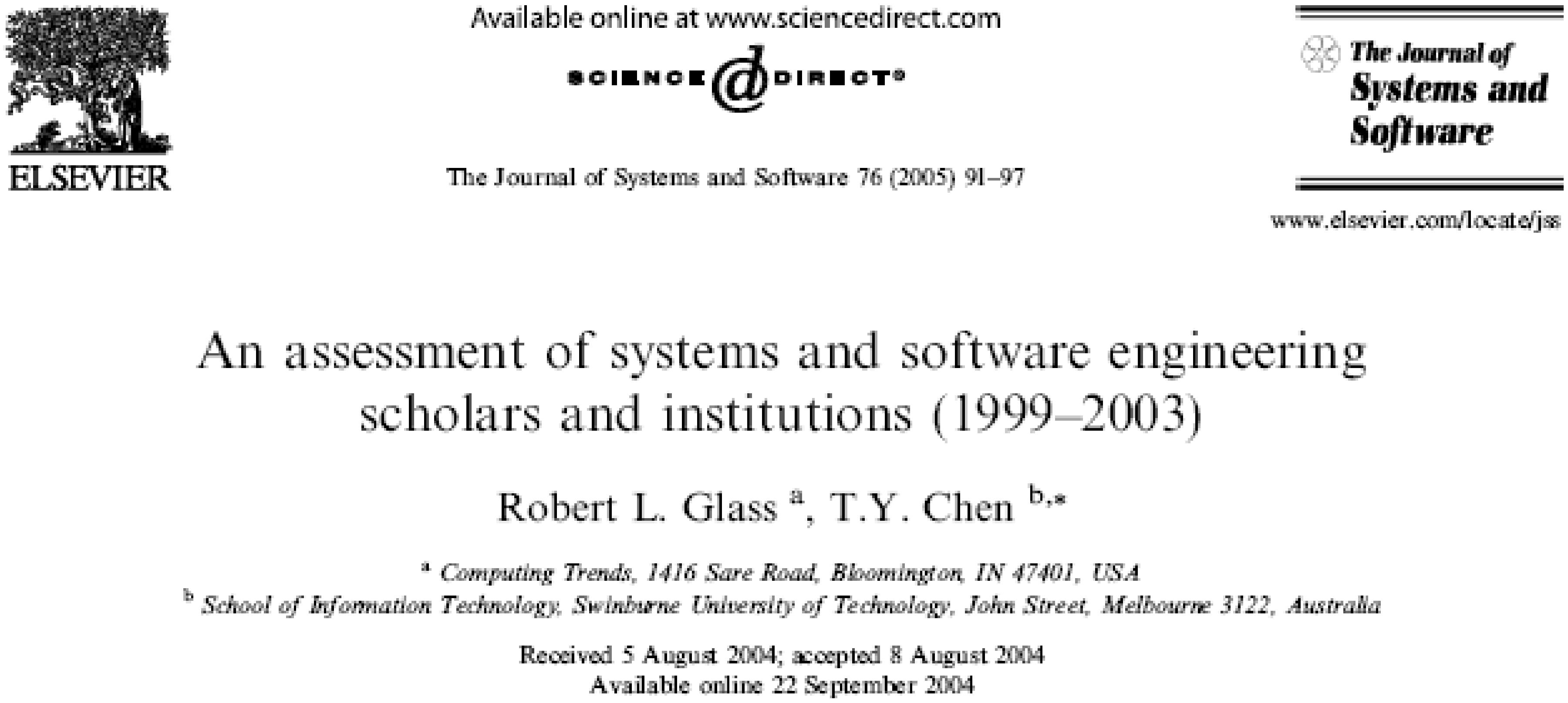
- **Grady Booch** is known for his method for object oriented analysis and design and his co-development of the unified modeling language (UML).
- **John Backus** is well known for the invention of FORTRAN, compiler optimization, and the Backus-Naur Form for the formal description of programming language syntax.
- **Michael Fagan** contributed extensively to the area of software inspection.
- **Niklaus Wirth** introduced the Pascal programming language and other languages, and contributed to the idea of decomposition and stepwise refinement.
- **Ole-Johan Dahl and Kristen Nygaard** introduced the Simula language which was the first object-oriented programming language.

Pioneers in software and software engineering (5)

- **Peter Naur** is known for his contributions to the creation of the ALGOL programming language and the introduction of formal syntax description language.
- **Tom DeMarco and Edward Yourdon** introduced the structured analysis and design approach for specifying and designing software systems.
- **Watts Humphrey** is known for his work on the capability maturity model developed by the software engineering institute and the personal software process.
- **Winston Royce** contributed the well-known model for the management of large software systems which was later named the Waterfall model.

An assessment of top systems and software engineering scholars and institutions

- The annual assessment considers the most published scholars in six software engineering journals during a sliding window of five years.
- The first and the last five year assessment covering the years 1993-1997 and 2001-2005, appeared in 1998 and 2007, respectively.
- Nine scholars appeared five or more times in the nine assessments



Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 76 (2005) 91–97

The Journal of Systems and Software

www.elsevier.com/locate/jss

An assessment of systems and software engineering scholars and institutions (1999–2003)

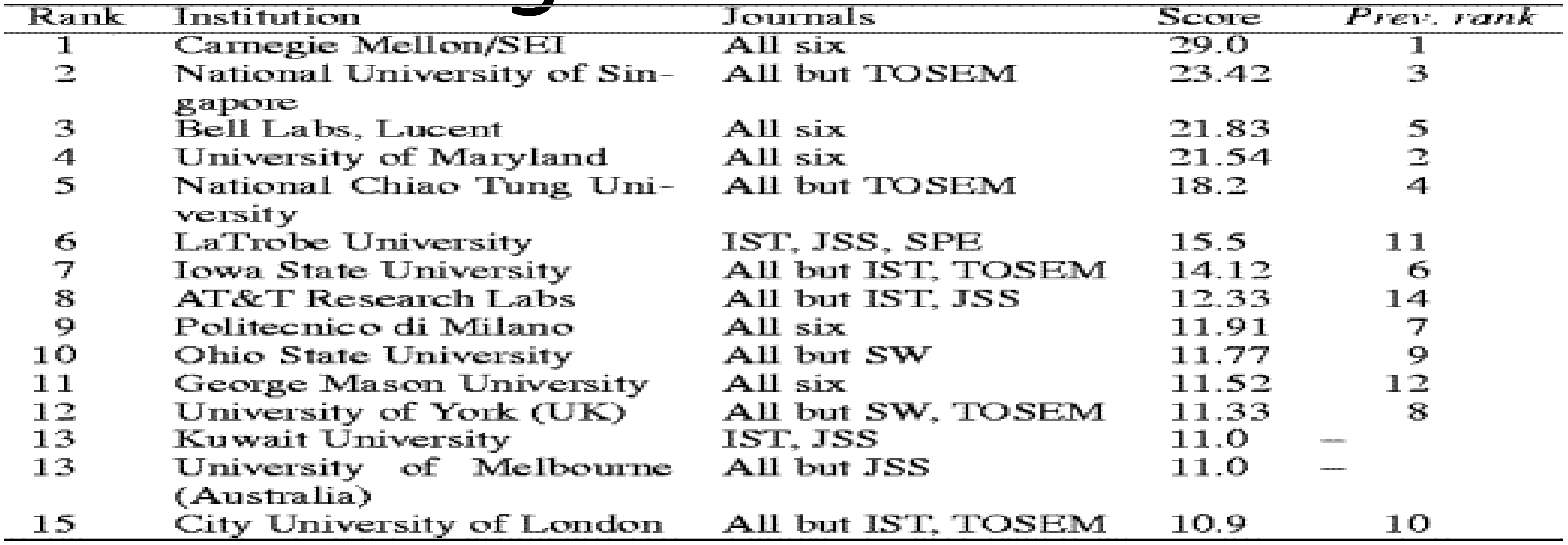
Robert L. Glass [a], T.Y. Chen [b,*]

[a] *Computing Trends, 1416 Sare Road, Bloomington, IN 47401, USA*

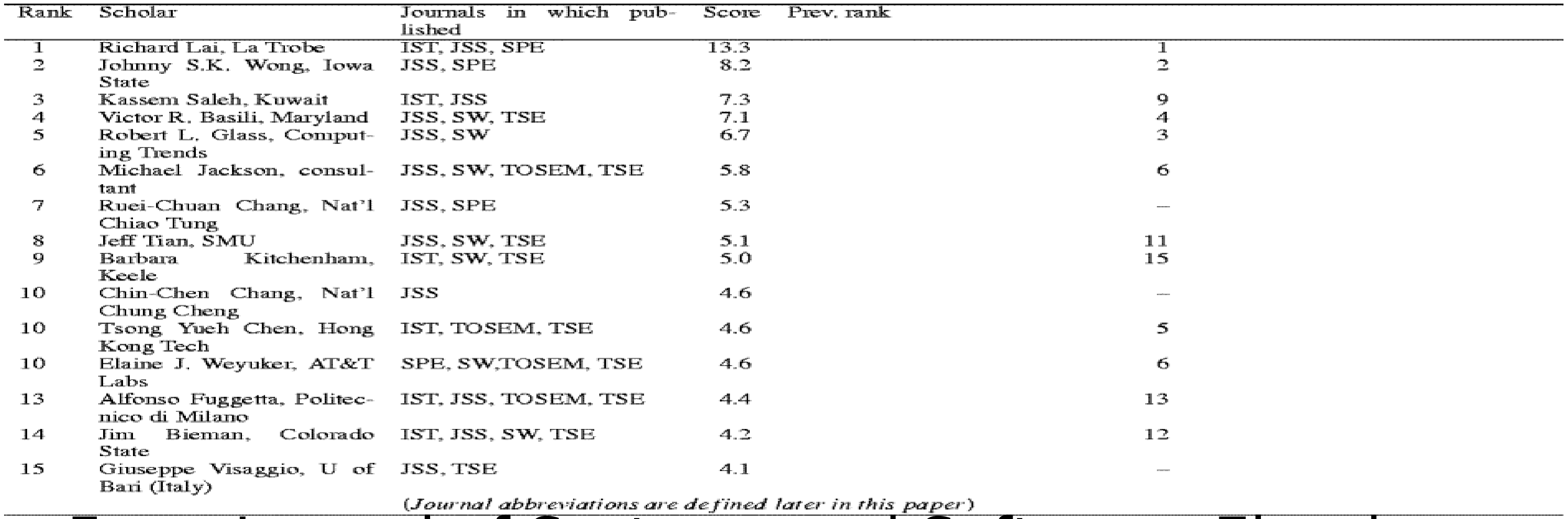
[b] *School of Information Technology, Swinburne University of Technology, John Street, Melbourne 3122, Australia*

Received 5 August 2004; accepted 8 August 2004
Available online 22 September 2004

Rankings for 1995-1999



Rank	Institution	Journals	Score	*Prev. rank*
1	Carnegie Mellon/SEI	All six	29.0	1
2	National University of Singapore	All but TOSEM	23.42	3
3	Bell Labs, Lucent	All six	21.83	5
4	University of Maryland	All six	21.54	2
5	National Chiao Tung University	All but TOSEM	18.2	4
6	LaTrobe University	IST, JSS, SPE	15.5	11
7	Iowa State University	All but IST, TOSEM	14.12	6
8	AT&T Research Labs	All but IST, JSS	12.33	14
9	Politecnico di Milano	All six	11.91	7
10	Ohio State University	All but SW	11.77	9
11	George Mason University	All six	11.52	12
12	University of York (UK)	All but SW, TOSEM	11.33	8
13	Kuwait University	IST, JSS	11.0	–
13	University of Melbourne (Australia)	All but JSS	11.0	–
15	City University of London	All but IST, TOSEM	10.9	10



Rank	Scholar	Journals in which published	Score	Prev. rank
1	Richard Lai, La Trobe	IST, JSS, SPE	13.3	1
2	Johnny S.K. Wong, Iowa State	JSS, SPE	8.2	2
3	Kassem Saleh, Kuwait	IST, JSS	7.3	9
4	Victor R. Basili, Maryland	JSS, SW, TSE	7.1	4
5	Robert L. Glass, Computing Trends	JSS, SW	6.7	3
6	Michael Jackson, consultant	JSS, SW, TOSEM, TSE	5.8	6
7	Ruei-Chuan Chang, Nat'l Chiao Tung	JSS, SPE	5.3	–
8	Jeff Tian, SMU	JSS, SW, TSE	5.1	11
9	Barbara Kitchenham, Keele	IST, SW, TSE	5.0	15
10	Chin-Chen Chang, Nat'l Chung Cheng	JSS	4.6	–
10	Tsong Yueh Chen, Hong Kong Tech	IST, TOSEM, TSE	4.6	5
10	Elaine J. Weyuker, AT&T Labs	SPE, SW,TOSEM, TSE	4.6	6
13	Alfonso Fuggetta, Politecnico di Milano	IST, JSS, TOSEM, TSE	4.4	13
14	Jim Bieman, Colorado State	IST, JSS, SW, TSE	4.2	12
15	Giuseppe Visaggio, U of Bari (Italy)	JSS, TSE	4.1	–

(*Journal abbreviations are defined later in this paper*)

From Journal of Systems and Software, Elsevier

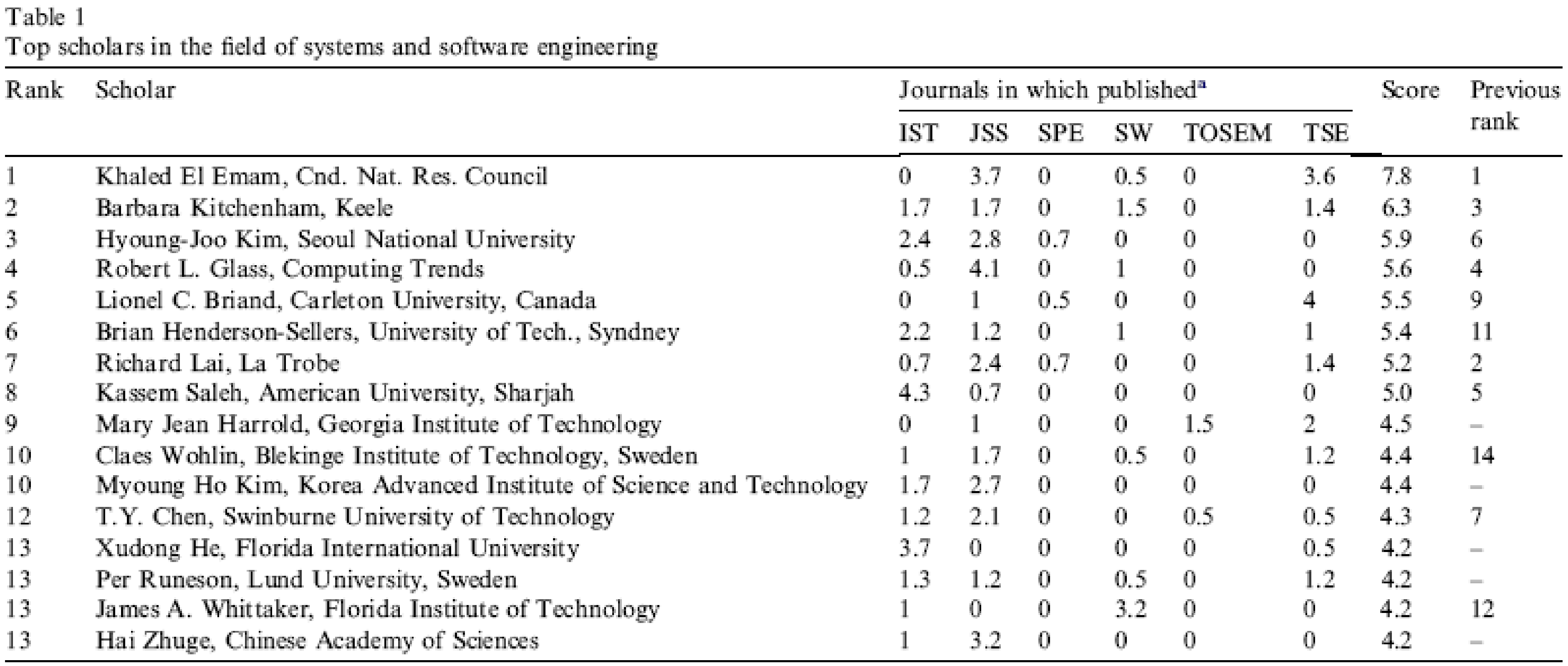


Table 1
Top scholars in the field of systems and software engineering

Rank	Scholar	Journals in which published[a]						Score	Previous rank
		IST	JSS	SPE	SW	TOSEM	TSE		
1	Khaled El Emam, Cnd. Nat. Res. Council	0	3.7	0	0.5	0	3.6	7.8	1
2	Barbara Kitchenham, Keele	1.7	1.7	0	1.5	0	1.4	6.3	3
3	Hyoung-Joo Kim, Seoul National University	2.4	2.8	0.7	0	0	0	5.9	6
4	Robert L. Glass, Computing Trends	0.5	4.1	0	1	0	0	5.6	4
5	Lionel C. Briand, Carleton University, Canada	0	1	0.5	0	0	4	5.5	9
6	Brian Henderson-Sellers, University of Tech., Syndney	2.2	1.2	0	1	0	1	5.4	11
7	Richard Lai, La Trobe	0.7	2.4	0.7	0	0	1.4	5.2	2
8	Kassem Saleh, American University, Sharjah	4.3	0.7	0	0	0	0	5.0	5
9	Mary Jean Harrold, Georgia Institute of Technology	0	1	0	0	1.5	2	4.5	–
10	Claes Wohlin, Blekinge Institute of Technology, Sweden	1	1.7	0	0.5	0	1.2	4.4	14
10	Myoung Ho Kim, Korea Advanced Institute of Science and Technology	1.7	2.7	0	0	0	0	4.4	–
12	T.Y. Chen, Swinburne University of Technology	1.2	2.1	0	0	0.5	0.5	4.3	7
13	Xudong He, Florida International University	3.7	0	0	0	0	0.5	4.2	–
13	Per Runeson, Lund University, Sweden	1.3	1.2	0	0.5	0	1.2	4.2	–
13	James A. Whittaker, Florida Institute of Technology	1	0	0	3.2	0	0	4.2	12
13	Hai Zhuge, Chinese Academy of Sciences	1	3.2	0	0	0	0	4.2	–

From Journal of Systems and Software, Elsevier

An assessment of the top software systems and engineering scholars and institutions (2)

- **Robert Glass**, Computing Trends, is a software visionary interested in strengthening the discipline and in software practice and project failures.
- **Barbara Kitchenham**, Keele University, specializes in empirical methods in software engineering, software cost estimation and software metrics.
- **Richard Lai**, LaTrobe University, specializes in web services, protocol engineering, component based software engineering and software testing.
- **Kassem Saleh**, Kuwait University, specializes in distributed systems, communications software engineering, software testing, and information systems security.
- **T.Y. Chen**, Swinburne University of Technology, specializes in software testing, debugging and software quality.
- **Victor Basili**, University of Maryland, specializes in empirical software engineering and model building.
- **Johnny Wong**, Iowa State University, specializes in system and network security, information assurance and multimedia and medical applications.
- **Jeff Tian**, Southern Methodist University, specializes in testing and quality improvement, measurement and risk management and web quality engineering.
- **Khaled El Emam**, University of Ottawa, specializes in software measurements and quality improvement.

Table 1.2 Analogy between software and building development

Phase	Software as a product	Building as a product
Requirements specification	Requirements specifications	Owner's requirements
Design	Design	Design
Implementation	Coding or construction	Construction
Testing	Testing and integration	Pre-delivery approval
Maintenance	Maintenance	Maintenance