October 6, 2009

Dear Readers:

As advertised, this is work-in-progress. Feel free to rip it, but please do not cite, circulate further, etc, without permission.

Since this is part of a larger book project, a note of explanation may be in order. The book-in-progress, *Arguments that Count: Physics, Computing, and Missile Defense, 1949-89*, analyzes the history of public debate and scientific advising on missile defense in the United States, and shows how the establishment something called "software engineering" shaped that debate. It weaves together three distinct, but mutually influential, narrative threads: the changing place of missile defense in nuclear policy (both official and de facto); the shifting place of physicists as political actors during the Cold War; and finally the rise of computing and software engineering.

A brief outline of the book is below. Chapter 7, "The Contrary Networking of Software Engineering," begins on the next page Chapter 7 is something of a fulcrum for the book. The missile defense debates that take place in chapters 8 and 9 are very different than those that take place in Chapters 5 and 6. Chapter 7 aims to explain how changes in computing contributed to some (not all) of those differences. To some extent, Chapter 7 also aims to show how the mid-1970s U.S. missile defense project (the most complex software project to that time) influenced software engineering. This is a minor theme in the chapter.

All of it is very drafty, but the section on network analysis (see the appendix) is extra, extra drafty. I've scattered references to networks throughout the chapter, but figures, charts, tables, and the like are located at the end.

Thanks for reading, and I look forward to your feedback!
Rebecca Slayton
rslayton@stanford.edu

Arguments that Count: Physics, Computing, and Missile Defense, 1949-89 (working title)

Table of Contents (draft)

Chapter 7

The Contrary Networking of Software Engineering

The air was chilly when Edsmund Dijkstra caught his ride from the airport in Montreal in May, 1974.[1] His driver, a manager from IBM, greeted the Dutch computer scientist with a complement: “So you are the world expert on structured programming and chief programmer teams.” The two were en route to a conference on software engineering education, sponsored by IBM at a luxury resort, and the software manager aimed to start on a high note. He missed his mark. Dijkstra concluded that [he] was out in the wilderness,” and “felt very low” by the time they reached the conference. He noted: “The conference was very instructive for me, although learned a lot without which I would have been happier.”[2]

Indeed, though he coined the phrase “structured programming” in the optimistic spirit that followed the 1968 software engineering conference, by 1974 it had taken on new meanings that Dijkstra found insidious. Whereas Dijkstra wished to structure *programs*, many managers focused on structuring teams of *programmers*. Proposals for chief programming teams struck some programmers as an attempt to “deskill” workers and subject them to strict managerial control.[3] Worse yet, whereas Dijkstra and other technically-oriented software experts were focused upon perfecting software, managers were often more focused on cost.

Dispute between managers and programmers, costs and quality, became frequent themes in a broader debate about the meaning and direction of software engineering in the mid-1970s. As computer experts in industry, academia, and government began to converge in the name of “software engineering,” starting new professional journals, special interest groups, newsletters,

[1] History data may be found online at: http://www.almanac.com/weather/history/postalcode/H4J%202K9/1974-05-03

[2] Dijkstra, Edsger W. (1982) *Selected writings on computing: a personal perspective*: Springer-Verlag New York, Inc.).

[3] In 1977, sociologist Philip Kraft drew on David Noble’s and Edward Layton’s classic works in the history of technology when he argued that structured programming was an attempt to control workers in the name of efficiency. Greenbaum, Joan M. (1979) *In the Name of Efficiency: Management Theory and Shopfloor Practice in Data-Processing Work* (Philadelphia: Temple University Press), Kraft, Philip (1977) *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York: Springer-Verlag). See also Baker, P. T., “Chief Programmer Team Management of Production Programming”, IBM Systems Journal, (11,1), 1972, pp . 56-71. Available online at: http://www.research.ibm.com/journal/sj/111/ibmsj1101E.pdf

and conferences, they were hard pressed to define the field. Practitioners advanced disparate research agendas, including programming languages, environments, tools, formal verification, database management, and complexity measures, to name a few. Conflicts were as frequent as conferences on software engineering.

Despite this diversity, there was a guiding hand in the formation of software engineering, and it was the one that Dijkstra despised: the visible hand of management. It is no mere coincidence that the software engineering began to congeal in the mid-1970s with the generous support of the U.S. Defense Department, an institution which was then beset with budgetary and management dilemmas. As we saw in the previous chapter, experts came to recognize software as the “pacing factor” in a deployment of missile defense in the late 1960s and early 1970s. The International Conferences on Software Engineering (ICSE) started up around the time that the “Safeguard” system was deployed in 1975, and as we will see shortly, presenters at early meetings of ICSE were more likely to be associated with missile defense than any other program or system. More generally, defense department managers in the early 1970s confronted opposing demands: cutting budgets while modernizing and unifying computerized command and control systems. A few key computer experts called attention to the growing costs of software, and proved persuasive. By the mid-1970s, the Army, the Navy, the Air Force, and the Defense Advanced Research Projects Agency, had each institutionalized centers of software engineering research and development, justifying it as a cost-saving measure. Economic anxieties marked “software engineering” indelibly with the interests of management.

Though economics helped justify and institutionalize new centers of research, software researchers conceived of their new discipline in much broader terms. Ultimately, the most influential publications addressed the challenges of management and technology seamlessly, rather than treating these issues in oppositional terms. Professional meetings were most likely to address issues of design, performance, and management, only occasionally invoking economics explicitly. And while management was a common theme, even one of Dijkstra’s most “pure” research agendas—formally proving the correctness of computer programs—was principally supported by a U.S. Defense Department anxious about securing computer networks. Defense officials kept their eye on the bottom line; networking itself was largely driven by an interest in cutting costs by sharing expensive computing resources. But whatever economic rationales underwrote their projects, software researchers were able to pursue a broader set of research agendas.

The result was that software engineering became established as a kind of contrary network in the 1970s. Would-be software engineers did not achieve the ideal-types of professionalism. Though debates about certifying software engineers (and software) were ongoing, software engineers did not succeed in agreeing upon or establishing a means of controlling entry to, or work in, the field. What they did establish was a social and intellectual network of common knowledge and experience about managing—and failing to manage—complex hardware-software systems.

1. **Evolving Command and Control Systems: The Software Problem**

Defense Department interest in software research grew rapidly in the early 1970s, largely in response to efforts to reform the U.S. World Wide Military Command and Control System (WWMCCS). In the late 1960s, three significant failures of the system cost American lives and prompted Congressional investigations.[4] These generally underscored the problem of non-standard computer systems and languages throughout the system.

Efforts at reform had begun under McNamara, who in 1966 ordered the Joint Chiefs to develop a proposal for integrating off-the-shelf commercial computers into WWMCCS.[5] When they finally submitted their recommendations in 1969, the system included 131 computer systems and 23 programming languages for 74 distinct command missions.[6] Herbert D. Benington, a veteran of the SAGE programming effort who had gone on to work in the Directorate of Defense Research and Engineering, emphasized the need to “standardize computer operations,” noting that the “push over the next five years will be in software, programming and support.”[7]

[4] These included failures to protect the U.S.S. Liberty from an Israeli attack, failure to rescue the U.S.S. Pueblo (an intelligence vessel) from North Korean waters, and failures to respond promptly to a North Korean shoot down of the Navy EC-121 reconnaissance plane over the sea of Japan. See discussion in Pearson, David E (2000) *The World Wide Military Command and Control System: Evolution and Effectiveness* (Maxwell Air Force Base, Alabama: Air University Press).

[5] In 1966, he requested his joint chiefs to propose a plan that would make use of off-the-shelf commercial equipment, and by 1969 they returned a plan that emphasized the need for a standardized system.

[6] Senate (1972) 'Department of Defense Appropriations for Fiscal Year 1973: Part I', in, *Subcommittee of the Committee on Appropriations, U.S. Senate* (Washington, D.C.: GPO).

[7] Staff (1969) 'Can Vulnerability Menace Command and Control?' *Armed Forces Management* (July): 40-3.

In November 1969, the new Secretary of Defense, David Packard generated considerable excitement in the computing industry when he announced a plan to purchase a minimum of 34 standard computers through a competitive bidding process, with an option to buy 35 more. Cost savings were a central rationale for standardizing hardware and its associated software. The non-standard software used on multiple computers required programming and debugging time, as well as retraining time for military personnel. As defense department managers and officials struggled to grapple with changes in command and control technology and operations, they feared that the cost of non-standard software would become prohibitive.[8]

Ironically, Congressional pressures to cut spending and reform defense acquisition processes soon undermined efforts at long-term cost savings. In the late 1960s, the highly visible failure of weapons systems like the Air Force’s C-5A bomber became a lightning rod for public criticism of the U.S. defense department.[9] As Thomas McNaugher has argued, the basic challenge lay in the tendency for research, development, and production to overlap, revealing the need for design changes after contracts were awarded. The problem was strikingly similar to what Edward E. David and Fraser had named as a fundamental cause of the software crisis in Garmisch, Germany: “the unfortunate telescoping of research, development, and production of an operational version within a single project effort.”[10]

When Nixon took office in 1969, he appointed a Blue Ribbon Defense Panel to recommend reforms to the defense acquisition process. The group levied a harsh critique of the defense department’s computer systems, including its command and control systems. More broadly, the panel concluded that McNamara’s procurement processes had not enabled sufficient flexibility in the weapons procurement process, prompting much discussion of the need for

[8] See Pearson, David E (2000) *The World Wide Military Command and Control System: Evolution and Effectiveness* (Maxwell Air Force Base, Alabama: Air University Press), Weiss, George (1971) 'Restraining the Data Monster: The Next Step in C3', *Armed Forces Journal* 108 (July 5): 26. Interestingly, Lee Paschall, the Director of Air Force Command, Control and Communications, participated in the CCIP-85 study that emphasized the rising cost of software, and placed a high emphasis on this problem in a 1973 talk. However, this does not appear to have stuck. Not long after Kissinger began to emphasize the need for a “flexible response” he once again returned to an emphasis on improving hardware. Compare Paschall, Lee M (1973) 'USAF Command Control and Communication Priorities', *Signal* 28 (November): 13, Paschall, Lee M (1974) 'Command and Control: Why the Air Force's Systems are Revolutionary', *Air Force Magazine* (July): 60.

[9] The C-5A cargo aircraft may be most notable. Rather than admit failure, the Air Force attempted to hide shortfalls in the program, and rather than absorb the growing cost, Lockheed, the contractor, attempted to cut corners. See discussion in McNaugher, Thomas (1989) *New Weapons, Old Politics: America's Military Procurement Muddle* (Washington, DC: Brookings).

[10] Naur, Peter and Randell, Brian (1969) 'Software Engineering: Report on a conference sponsored by the NATO Science Committee', (Garmisch, Germany): 136.

"evolutionary" development of weapons systems.[11] In fact, McNamara had long emphasized the need for evolutionary development in the context of computerized command and control systems.[12]

In some ways, this renewed emphasis on evolutionary development and life cycle costs, including the costs of training personnel to use new computer systems, encouraged investments in software research. But in other ways, defense cutbacks and intensified Congressional scrutiny undermined efforts at reform. They encouraged the services to lease computers rather than purchase them outright, since leased computers could be classified as operating expenses rather than procurement. Budget cuts also forced Packard to reduce the scale of the standardized computer purchase, from a minimum of 34 standard computers to a minimum of 15.[13] Worse yet, budget cuts encouraged the defense department to purchase affordable but outdated computers. Air Force officials were especially concerned when they learned that the defense department was not placing a high priority on purchasing computers that could operate in a real-

[11] Packard responded by introducing a new Defense Systems Acquisition Review Council (DSARC). Comprised of representatives from both the services and the OSD, the DSARC would approve the beginning of project development at Milestone I, decide whether the project would go to full-scale development at Milestone II, and finally decide whether the project should go into production at Milestone III. Packard kept McNamara's Planning, Programming, and Budgeting System (PPBS). Though it was not popular among the services, it enabled the OSD to control defense spending by requiring the services to elaborate goals and costs of their proposals. See discussion in McNaugher, Thomas (1989) *New Weapons, Old Politics: America's Military Procurement Muddle* (Washington, DC: Brookings). Packard's response to the panel report, see http://www.lexisnexis.com/congcomp/getdoc?HEARING-ID=HRG-1970-OPH-0030 Though it was a well-intended set of reforms, it did not resolve the fundamental challenges associated with developing complex technology. Growing Congressional scrutiny did little to rationalize the process. The services continued to experience incentives to push for "full scale development" prematurely, telescoping the process of research, development, and production even beyond what might be expected. This was similar to the challenges under McNamara. As Thomas McNaugher has argued, the weapons development process is most efficient when risks are taken early in the process, yet political and bureaucratic pressures pushed in the opposite direction. As the services faced growing Congressional scrutiny in the 1970s, they built more internal consensus on weapons systems and sought to limit options between Milestones I and II, prior to a point when major design problems might be uncovered. The U.S. Congress gained increasing incentives and means for intervening in weapons procurement through the 1970s. Between 1970 and 1985, the number of reports and studies requested by Congress increased by 1,172 percent, and the number of detailed program adjustments to service budgets increased from 830 to 3,163. McNaugher, Thomas (1989) *New Weapons, Old Politics: America's Military Procurement Muddle* (Washington, DC: Brookings). Quoting Weinberger.

[12] See Editor (1965) 'Evolution and Compatibility: 1965's Key Words in Tactical C&C', *Armed Forces Management* 11/10 (July): 44, Editor (1966) 'How Not to Build a C&C System is Still Unanswered', *Armed Forces Management* 12/10 (July): 109.

[13] Schemmer, Benjamin F (1970) 'DoD's Computer Critics Nearly Unplug Ailing World Wide Military Command and Control System', *Armed Forces Journal* 108 (December 21): 22, Volz, Joseph (1970) 'Revamped World-Wide Computer Net Readied', *Armed Forces Journal* 107 (June 27): 21.

time manner. But despite objections, in October of 1971, Burroughs won a contract to provide fifteen outdated batch-processing computers for the WWMCCS.[14]

No doubt anticipating the decision, in September of 1971 the Joint Chiefs of Staff directed the development of a Prototype WWMCCS Intercomputer Network (PWIN).[15] Though computer networking was relatively new, the successful demonstration of packet switching through the ARPAnet in 1969 was promising. By 1972, the Defense Communications Agency began coordinating the development of the PWIN using packet-switching and WWMCCS computers.[16]

1.1. Computer Networking, Economics, and Security

A major goal of networking in the WWMCCS was not only to increase the availability of timely information, but also to reduce costs associated with storing data on multiple computers.[17] But ironically, efforts at computer time-sharing were already beginning to contribute to growing software costs in industry. In 1968 Werner Frank, then senior vice president and co-founder of Informatics, may have been the first to note the consequences of a "startling" "proliferation of remote access computing." While the cost of peripheral equipment such as terminal displays continued to drop, the cost of the software that aimed to make such remote access terminals user-friendly was either constant or growing. As a result, the ratio of software to hardware expenses was increasing, and might be predicted to rise to 80% of the total cost of computer systems by 1978 (see chart). Frank recommended that expert programmers take the lead on standardizing

[14] Dirnbauer, Frank R, Goertzel, Herbert B and Miller, James B (1976) 'WWMCCS ADP: A Promise Fulfilled', *Signal* 30 (May/June): 60, Pearson, David E (2000) *The World Wide Military Command and Control System: Evolution and Effectiveness* (Maxwell Air Force Base, Alabama: Air University Press). The resulting systems software was so cumbersome that many users wrote their own patches to work around it, contributing ironically to de-standardization.

[15] GAO (1979) 'The World Wide Military Command and Control System—Major Changes Needed in its Automated Data Processing Management and Direction, Report to the Congress', in (Washington, D.C: General Accounting Office).

[16] The prototype used three interface message processors like those built for ARPA, and was to link three WWMCCS computers. This would be transitioned into a system-wide network, the WIN. See discussion by Lukasik in: Senate (1972) 'Department of Defense Appropriations for Fiscal Year 1973: Part I', in, *Subcommittee of the Committee on Appropriations, U.S. Senate* (Washington, D.C.: GPO). This was apparently related to, but distinct from, Autodin, another system developed by the DCA. See discussion in Pearson, David E (2000) *The World Wide Military Command and Control System: Evolution and Effectiveness* (Maxwell Air Force Base, Alabama: Air University Press).

[17] GAO (1979) 'The World Wide Military Command and Control System—Major Changes Needed in its Automated Data Processing Management and Direction, Report to the Congress', in (Washington, D.C: General Accounting Office).

software packages to cut costs, for he had little faith that either computer manufacturers or the government would take the lead.[18]

In the defense department, resource sharing also posed critical security concerns. In the late 1960s, there was no reliable way to prevent a computer user from accessing any information on the system; access controls were weak. As Donald MacKenzie has discussed, this problem became a matter of concern to several computer experts associated with government computing and time-sharing in the late 1960s.[19] At the Spring Joint Computer Conference in 1967, a special session addressed the problem of security in resource-shared systems explicitly. The session was opened by Willis Ware, of Rand, and followed by Bernard Peters of the National Security Agency. Peters took the unusual step of acknowledging his affiliation to the National Security Agency (then a secret agency known to insiders as "No Such Agency").

Ware soon led a study of this problem under the sponsorship of ARPA's Information Processing Technologies Office. Ware's group addressed "the most difficult security control situation - a time-sharing multi-access computer system serving geographically distributed users, and processing the most sensitive information." The group noted that it was "impossible to address the multitude of details that will arise in the design or operation of a particular resource-sharing computer system in an individual installation." Thus, it concluded, "*the security problem of specific computer systems must, at this point in time, be solved on a case-by-case basis, employing the best judgment of a team consisting of system programmers, technical hardware and communication specialists, and security experts*." Without offering any silver bullet, Ware's group recommended that the defense department fund several areas of research.[20]

Ware's study appears to have prompted relatively little action.[21] In 1972, as work started up on a prototype network for command and control, the Air Force commissioned an independent consultant, James P. Anderson, to assess the security problem.

[18] Frank, Werner L (1968) 'Software for Terminal-Oriented Systems', in, *Datamation*: 30-34.

[19] See MacKenzie, Donald (2001) *Mechanizing Proof: Computing, Risk, and Trust* (Cambridge, MA: MIT Press).

[20] Ware, Willis (1970) 'Security Controls for Computer Systems: Report of the Defense Science Board Task Force on Computer Security', in (Washington, D.C.: Office of the Director of Defense Research and Engineering, Washington, DC). Ware's study involved a task force under the auspices of the Defense Science Board.

[21] Anderson's study noted that Ware's work may have inadvertently caused problems by specifying "necessary but not sufficient" ways of improving security. It also noted that ARPA's IPTO had followed Ware's study by sponsoring work on computer security, but did not see this as sufficient. See Anderson, James P (1972) 'Computer Security Technology Planning Study Vol 1', in (Hanscom Air Force Base, Bedford, MA: Electronic Systems Division of the Air Force Systems Command).

Anderson's report distinguished itself from Ware's in two key respects. First, whereas Ware had outlined a somewhat piecemeal research agenda, Anderson's group emphasized the need to take a holistic approach, designing security "into a system from its inception." Since the WWMCCS operating systems were "not built in accordance with a unified set of principles of computer security," there would be no way to make them secure.[22] The report also became quite specific, recommending four clearly related research programs totaling nearly $30 million, and including an $8 million effort to develop a secure, open-use multilevel prototype system.[23] It took special note of academic work by Dijkstra and others, which aimed at developing programs that could be formally proven correct.[24]

Second, whereas Ware had expressed concern about the costs of ensuring that computer systems would be secure, Anderson focused on the potential for cost savings. Since computer systems could not reliably prevent users with one level of clearance from accessing information that was classified at a higher level, the Air Force was forced to isolate and replicate computer systems to prevent unauthorized access. With the growth of terminal systems and other expensive equipment on each computer system, this was costing approximately $100 million a year.[25]

As Donald MacKenzie has discussed, Anderson's study helped spur the Defense Department to action. The National Security Agency and ARPA funded significant research into formal verification throughout the 1970s.[26]

[22] Ibid.

[23] Anderson's panel was charged with developing a "comprehensive plan for research and development leading to the satisfaction of requirements for multi-user open computer systems which process various levels of classified and unclassified information simultaneously through terminals in both secure and insecure areas." Anderson, James P (1972) 'Computer Security Technology Planning Study Vol 2', in (Hanscom Air Force Base, Bedford, MA: Electronic Systems Division of the Air Force Systems Command). The description of the prototype system is contained in Vol. II, Section III. Work on formal verification and structured programming is discussed under the heading of "Exploratory Development," and is described in Section VII. The group recommended spending $10.95 million on exploratory development, including both systems studies and hardware architecture studies; See Anderson, James P (1972) 'Computer Security Technology Planning Study Vol 2', in (Hanscom Air Force Base, Bedford, MA: Electronic Systems Division of the Air Force Systems Command).

[24] Dijkstra's work is mentioned, along with Floyd's work: Anderson, James P (1972) 'Computer Security Technology Planning Study Vol 2', in (Hanscom Air Force Base, Bedford, MA: Electronic Systems Division of the Air Force Systems Command).

[25] Anderson, James P (1972) 'Computer Security Technology Planning Study Vol 1', in (Hanscom Air Force Base, Bedford, MA: Electronic Systems Division of the Air Force Systems Command).

[26] MacKenzie, Donald (2001) *Mechanizing Proof: Computing, Risk, and Trust* (Cambridge, MA: The MIT Press). MacKenzie focuses in particular on the notion of "mechanizing proof." This built on work in in artificial intelligence by John McCarthy, Naur, and Floyd, all sponsored by ARPA. For more on the ARPA-sponsored AI agenda, see

While concern about security was real, it is important to note that Anderson’s research agenda was justified by cost-savings, and that this theme continually emerged in Defense Department efforts to explain research on computer security and software. Efforts at formally proving programs to be correct—the gold standard in achieving perfect software—were one research agenda swept up into “software engineering” in the 1970s, but the economics of evolving large and complex software systems would become much more dominant.

1.2. Software Engineering as Management Science

Perhaps the most notable advocate of software engineering economics was Barry Boehm. Initially trained as a mathematician, Boehm began working as a Programmer Analyst at General Dynamics in 1955. He moved to RAND in 1959, eventually becoming the head of Rand’s information sciences department. While at RAND in the late 1960s and early 1970s, he contributed to a study of the Air Force’s computing needs for future command and control systems. Sponsored by the Air Force’s Space and Missile Systems Office (SAMSO), the “Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980s,” or CCIP-85, was a multi-volume study that was finally completed in 1972. The most shocking conclusions of the study were contained in one volume on software.[27]

Boehm described the study conclusions in the trade journal *Datamation*: “for almost all applications, software (as opposed to computer hardware, displays, architecture, etc.) was the ‘tall pole in the tent’ – the major source of difficult future problems and operational performance penalties.” However, as he went on to explain, “we found it difficult to convince people outside of the software business of this…primarily because of the scarcity of solid quantitative data to demonstrate the impact of software on operational performance or to provide perspective on R&D priorities.”[28]

Boehm put the growing costs of software front and center in his efforts to persuade the defense department of the need for a discipline of software engineering. He pointed to estimates

Chapter 5 of Norberg and O’Neill Norberg, Arthur L and O'Neill, Judy E (1996) *Transforming Computer Technology: Information Processing for the Pentagon, 1962-1986* (Baltimore: John Hopkins University Press).

[27] The CCIP-85 study saw discussion in several military and trade journals: Boehm, Barry (1973) 'Software and its Impact: A Quantitative Assessment', *Datamation* /May: 48-59, Paschall, Lee M (1973) 'USAF Command Control and Communication Priorities', *Signal* 28 (November): 13, Ulsamer, Edgar (1971) 'The Military Decision-Makers' Top Tool', *Air Force Magazine* (July): 44-49, Ulsamer, Edgar (1972) 'Command and Control is of Fundamental Importance', *Air Force Magazine* (July): 42-46.

[28] Boehm, Barry (1973) 'Software and its Impact: A Quantitative Assessment', *Datamation* /May: 48-59.

that in the fiscal year 1972, the Air Force had spent between $1 and $1.5 billion on software, three times its expenditure on computer hardware. This amounted to 4 to 5 percent of the entire Air Force budget. He also noted the high cost of the World Wide Military Command and Control System that was coming under Congressional scrutiny. By 1971, Pentagon planners expected that software would account for over 75% of the total cost of the WMCCS upgrade, $722 million out of $959 million. These high costs were coming under scrutiny by the General Accountability Office, which accused planners of using "highly questionable" cost estimates, allowing 'fragmented responsibility for planning and direction of acquisition,' and unnecessarily upgrading equipment.[29]

Boehm also pointed to the expense of past systems: the SAGE air defense system had cost the U.S. government $250 million, OS/360 reportedly cost IBM $200 million, and software from the manned space program had cost approximately $1 trillion during the 1960s. Nationwide, he estimated that software costs amounted to more than $10 billion each year, over 1% of GNP. Drawing a projection into the future, CCIP-85 predicted that software would consist of 90% of the Air Force's Advanced Data Processing budget for the AF by 1985.

Boehm's vision of software engineering thus focused on management concerns. He concluded that "the problems of software productivity on medium or large products are largely problems of management."[30] He noted that management experience was not typically transferred from one project to the next: "many of the lessons learned as far back as SAGE are often ignored in today's software developments, although they were published over 10 years ago…"[31]

Neither reliability nor security was a central theme in the CCIP-85 study. Boehm's *Datamation* article devoted only two of eleven pages to these issues, and remained focused on measurements. The article showed the results of tabulating software errors in major command and control systems, in an effort to discern the most common causes of failure. The solution to problems began with developing measurement tools, such as a "usage-measuring compiler"—a program "for keeping track of error rates and other software phenomena."[32]

[29] See also: Hirsch, Phil (1971) 'GAO Hits Wimmix Hard; FY'72 Funding Prospects Fading Fast', Ibid. (March 1): 41.
[30] Boehm, Barry (1973) 'Software and its Impact: A Quantitative Assessment', Ibid. /May: 48-59.
[31] Ibid.
[32] Unexpected side effects due to changes; logical flaws in original design; logical flaws in changes to design; Inconsistencies between design and implementation; Clerical errors; Inconsistencies in hardware. Though the analysis focused on command and control systems, Boehm also addressed the prospect of unskilled users trusting

Boehm concluded: "Until we establish a firm data base, the phrase 'software engineering' will be largely a contradiction in terms." He went on to emphasize the importance of quantitative measures, quoting Lord Kelvin:

> When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science.[33]

As this suggests, Boehm conceived of software engineering as a heavily quantitative kind of "applied science," with a heavy emphasis on the science of management.

Boehm's approach appears to have persuaded many Defense Department officials of the need to invest in software. Growing Congressional scrutiny likely contributed to their interest in Boehm's managerial approach to improving software. Along with ongoing defense acquisition reform, confusion about the status of software – as research, development, or procurement – made confrontation between the U.S. Congress and defense officials all the more frequent.[34] By focusing on the challenges of managing "software engineering," Defense officials could respond, or at least appear to respond, to ongoing calls for accountability.

1.3. ***The Great Software Management Experiment: Site Defense***

In 1973, Boehm went to work for TRW as "Director of Software Research and Technology," where he quickly became involved with the Site Defense software project. As discussed in the last chapter, the Army's "Site Defense" project was launched in 1971 as a supplement to the proposed "Safeguard" deployment. To answer the critiques made of "Safeguard," by Pief Panofsky, Sidney Drell, and other presidential science advisors, the Army launched Site Defense as a prototype system capable of being deployed "responsively" to an

flawed results: "by the 1980s, perhaps 40% of the labor force will be trusting implicitly in the results produced by computer software."

[33] Boehm p 59

[34] In 1973 hearings, the Congress criticized the Air Force for including software upgrades for its 485-L Tactical Air Control System, in "procurement" rather than research and development. House (1973) 'Department of Defense Appropriations for Fiscal Year 1974: Part 6 Procurement', in, *Subcommittee of the Committee on Appropriations, U.S. House of Representatives* (Washington, D.C.: GPO). In 1974, Congressional representatives were critical of the DCA for attempting to fund satellite surveillance software development (for MIDAS?) as research and development, rather than procurement. See House (1974) 'Department of Defense Appropriations for Fiscal Year 1975: Part 4', in, *Subcommittee of the Committee on Appropriations, U.S. House of Representatives* (Washington, D.C.: GPO): Apr. 25, 29, 30, May 1, 1974. See also confusion over the meaning of "off-the-shelf" software: GAO (1971) 'Potential Problems In Developing The Air Force's Advanced Logistics System; Report to the Committee On Appropriations, House Of Representatives', in (Washington, D.C: General Accounting Office).

evolving Soviet threat. Over time it became a central part of ongoing research at the Army's Ballistic Missile Defense Advanced Technology Center (BMDATC), in Huntsville, Alabama.

Though Boehm was not primarily responsible for Site Defense, he inevitably became involved because, as he recalled, "Site Defense was the biggest thing going."[35] Boehm's managerial approach to software fit naturally with that of other software developers at TRW. As the contractor for the Space Command and Control system, TRW helped promote what came to be known as the "waterfall" approach to software development. In this model, each successive step in software development would involve some iteration in relation to the preceding (higher level) and succeeding (lower level) steps.[36]

While TRW's experience helped it win a contract for Site Defense, the missile defense software brought unique challenges. Unlike the Space Command and Control software, which helped scientists track relatively predictable orbits, missile defense software tracking missiles that might move in deliberately unpredictable ways, and be hidden by decoys or chaff. Also unlike the space Command and Control software, testing was highly constrained for missile defenses; computer simulation of an attack was a key part of "testing." Finally, as we discussed in the previous chapter, the strategic goals of the Army's prototype system were a moving target; this was the whole point of "responsive deployment." All of this turned Site Defense into an unprecedentedly large software management experiment.

By the same token, with a hundred million dollar budget, the Site Defense project offered software engineers unprecedented resources. The project managers, William Besser and Robert Williams, decided to invest in automated tools, since this promised "economies of scale" in complex software development. As Boehm noted, "on a $100,000,000 budget you can spend $200,000 on a simulator and still have a lot of money left – a $500,000 thing you can't do that." It also allowed managers to say: "We're going to do this right. We're going to do all the planning up front. We're going to invest in tools. …We're going to build in increments rather than a big

[35] Author Interview with Barry Boehm, July 3, 2007, USC. In 1971, TRW won a contract to develop software for the Site Defense project. McDonnell was the prime contractor on Site Defense; TRW was one of many subcontractors. According to TRW's historian, Site Defense "resulted in the biggest and most complex software system in history, a feat so daunting that many experts doubted whether it could be achieved at all. Bell Labs, the original contractor for the program, abandoned the challenge as not feasible under existing requirements." As we have seen, Bell Labs had other reasons for getting out of the missile defense business.

[36] Royce, Winston (1970) 'Managing the Development of Large Software Systems', in, *Proceedings*, *IEEE Wescon*: 328-39. As others have pointed out, Royce himself never used the term "waterfall," though the diagrams in his paper are certainly suggestive.

bang. We’re going to have separate increment managers. We’re going to get the smartest people in the company to work on this.”[37]

The tool-oriented approach to software development was well-described in a paper by Site Defense manager Williams, which he presented at an International Conference on Reliable Software in 1975. Williams focused on four phases of the development process: requirements engineering, software design, implementation, and testing. In each case he suggested developing automated tools that could help the project manager. For example, he promoted “an innovative and extremely powerful technique called Engagement Logic” to help visualize connections between high-level software requirements.[38] Moving on to design, emphasized the usefulness of Harlan Mills’ approach to structured programming and top-down design, and described a system that could automatically update a management chart.[39] As for implementation, he focused on the managers’ need to “hear the process talk,” and described the Process Construction Program (PCP), an automated system to assist in the development of real-time processes.[40] Finally, he suggested improving testing with procedures and tools that could handle “tedious, error-prone, and difficult testing tasks.”[41]

This tool-oriented approach to software development was popular among managers of large software development projects, and especially among major defense contractors. However, it could not necessarily find broad application among software developers with more limited means. TRW’s software development tools worked well at the Army’s advanced research facilities in Huntsville, because they ran on the Texas Instrument Advanced Scientific Computer. This machine was even more advanced than the CDC 7700 that engineers planned to use for the Site Defense prototype command and control system. It was also more advanced that the computers at TRW’s headquarters in Redondo Beach. Boehm recalled: “when we first imported

[37] Author Interview with Barry Boehm, July 3, 2007, USC. In 1971, TRW won a contract to develop software for the Site Defense project. McDonnell was the prime contractor on Site Defense; TRW was one of many subcontractors. According to TRW’s historian, Site Defense “resulted in the biggest and most complex software system in history, a feat so daunting that many experts doubted whether it could be achieved at all. Bell Labs, the original contractor for the program, abandoned the challenge as not feasible under existing requirements.” As we have seen, Bell Labs had other reasons for getting out of the missile defense business.

[38] Williams, R. D. (1975) 'Managing the development of reliable software', in, *Proceedings of the international conference on Reliable software* (Los Angeles, California: ACM).

[39] Ibid.

[40] In particular, the program supported “three major development functions of data base, system component, and system (process) building.” Ibid.

[41] Ibid.

the requirements engineering methodology back to TRW we used it on one proposal… the first run of the tool cost $60,000 dollars of CDC 7700 time and blew half of the budget proposals." [42]

Despite such difficulties, the Army's Site Defense project nurtured a particular kind of agenda within the nascent field of "software engineering." It ultimately produced "the TRW software series," a set of texts discussing software management. When the second International Conference on Software Engineering convened in 1976, over ten percent of the presenters were associated with the ballistic missile defense software research.[43] When we consider the general terms associated with papers from these individuals, it would seem that interests were widely dispersed through the field (see Graph I and appendix on methods). Yet abstracts from these presenters were more than three times as likely as those from other presenters at the conference to emphasize management *tools*, and more than seven times as likely to emphasize user *requirements*, suggesting a strong managerial agenda. They were less likely than other papers to emphasize *specifications*, the term preferred by those dealing with computer programs rather than programmers (See Table I).

Managers working on missile defense had good reason to be concerned about both requirements and specifications. As Williams noted:

> There is nothing, absolutely nothing, that can cause a more devastating outcome of a development activity than an incomplete and/or incorrect knowledge and corresponding specification of system and software requirements. If the customer does not have a good understanding of what needs to be done or if the contractor can't figure out what the customer wants, then there is good reason to believe that serious trouble lies ahead.

Arguably, this was precisely the dilemma of software development for missile defense. Without a complete specification for the attack that would confront the missile defense, the number of targets it would confront, and so on, the "correct" software specifications were inherently uncertain. The ultimate system requirements – coping with a nuclear attack from an intelligent adversary – were inherently unpredictable. The system requirements also depended significantly upon the strategic role assigned to missile defense by the presidential administration, which changed relatively frequently and was often challenged by the U.S. Congress. In the mid-1970s, managers developed automated project management tools to grapple with rapid change, but none could resolve the fundamental problems of volatile and unpredictable requirements.

[42] Interview with Boehm, July 2007.

[43] Over ten percent of the presenters were associated with missile defense software research. http://portal.acm.org/citation.cfm?id=808418

2. The Contrary Networking of Software Engineering

2.1. Developing a Lingua Franca: Standardization in the Defense Department

Site Defense was just one example of the defense department’s growing concern with managing the evolution of complex software systems. By the early 1970s, virtually every branch of the military was sponsoring significant research into software engineering.[44] Concerns about managing software costs also contributed to the development of a single standard programming language.[45] The project was initiated and overseen through its formative years by Air Force Colonel William Whitaker. First trained as a nuclear physicist, Whitaker had experienced the challenges of non-standard programming languages while working at the Air Force Weapons Laboratory in the 1960s. In 1973, he joined the Electronics and Physical Sciences Section of the Office of the Director of Defense Research & Engineering (ODDR&E).[46]

That summer Whitaker heard Boehm present his findings as the keynote address at a “Symposium on the High Cost of Software,” at the Naval Post Graduate School in Monterey. Whitaker recalled that Boehm’s estimates growing software expenses were so “shocking” that they were “rejected out-of-hand.” The Air Force refused to publish the findings until Whitaker’s boss, the Director of the Electronics and Physical Sciences Division, intervened.[47]

With numbers in hand, Whitaker went on to make the case for launching an effort to develop a new, general-purpose language for the defense department.[48] He felt that the case for a

[44] Boehm felt that the CCIP-85 study had begun “to reorient air Force information processing much more toward software,” and noted similar trends in other government agencies. Boehm, Barry (1973) 'Software and its Impact: A Quantitative Assessment', *Datamation* /May: 48-59. The Air Force’s history of software measurement at its Rome Air Development Center suggests the influence of Boehm’s ideal of engineering as a measurement-intensive, management science. Funding for software research increased following Boehm’s study in 1972, and in 1975, the Air Force institutionalized concerns about software measurement with its Data & Analysis Center for Software (DACS). http://www.dacs.dtic.mil/techs/history/toc.html In 1968 the NRL’s Ship-Shore radio division reorganized into a Communication Sciences division and became the “Information Technology Division” in 1981. By 1972 David Parnas began consulting for the Navy, becoming the head of its Software Engineering Research Section from 1979-82, and then working on its Software Cost Reduction Project from 1982-86.

[45] For more on ADA, see: http://archive.adaic.com/pol-hist/history/holwg-93/8.htm

[46] Whitaker, William A (1996) 'ADA - The Project: The DoD High Order Language Working Group'.

[47] Ibid. Whitaker went on to request that the Institute for Defense Analyses conduct an independent study to validate Boehm’s results. This was soon done, using a different method.

[48] Some issues he mentioned were the fact that while “short-term management goals (contractor and government) encouraged proliferation” of programming languages, DARPA’s primary charge was to foster research and development with long-term payoff. Furthermore, by 1974 computer hardware was increasingly being shaped by commercial markets, but software markets were somewhat more subject to the needs of government. Ibid.

general-purpose language was bolstered when each of the services independently proposed a standard language that it could use as a supplement to existing languages in 1974. Though the proposal for a common language "initially met almost universal opposition," by 1975 Whitaker had helped establish a Higher Order Language Working Group (HOLWG), a group that included representatives from each of the three services. The group refined language requirements in a series of specifications that began with STRAWMAN, and ended with STEELMAN.[49] These specifications were then opened to competitive bidding.

Of six technical characteristics listed for the language, one was primary: "the language should facilitate the reduction of software costs," and in particular the long-term costs of maintaining software. This goal demanded "transportability," "flexibility," "efficiency," long-term "readability," and even "reliability." As the group noted, "reliability is an extremely severe requirement in many Defense systems and is often reflected in the high cost of extensive testing and verification procedures." As Whitaker recalled, such "ilities" were but did "not lend themselves to a quantifiable or rational assessment of programming languages."[50]Accordingly, as the chair of the group, Whitaker's job was simultaneously political and technical, as he attempted to convince the services that one language could meet the needs of all.[51]

Efforts at standardization also met resistance from computer scientists who rejected the defense department's emphasis on cost management. Notably, Edsmund Dijkstra wrote a rather critical review of the STRAWMAN language requirements in 1975:

> A considerable part of the given justification is financial. Now, although coins are very concrete objects, money is a highly abstract notion, and the author does not seem to understand it…
>
> At various places – and that is a much more forceful justification – the author expresses his concern about the quality of today's software ('Programming errors can have catastrophic consequences.')…[52]

Thus Dijkstra chose "to read the document as a proposal aimed at improving the quality of DoD software." While he expected that this would reduce costs by reducing mistakes, he insisted upon

[49] Ibid.
[50] Ibid.
[51] As one commentator noted: "Certainly, the Navy felt that only the Navy could design a language that would meet Navy needs. Only the Air Force could design a language that would meet Air Force needs. Only the Army for Army needs. Overcoming that kind of thinking was a major bureaucratic in-fighting win of considerable talent." Ibid.
[52] Dijkstra, Edsger W. (1975) 'On a language proposal for the Department of Defense', in, *SIGSOFT Softw. Eng. Notes*.

viewing "possible cost reduction as a (possibly considerable) fringe benefit, the higher quality as the main goal…"[53]

Dijkstra went on to critique the proposal for its tendency to favor management considerations over quality. He felt that the "worst shock" came in the statement that "benefits will grow if new efforts adopt the standard whatever it is." He went on to question bureaucratic imperatives:

> …the text continues ominously: "The ultimate measure of success must be in cost-savings and failure of a project to adopt the common language should only be for demonstrable economic reasons." I don't like that sentence at all. The use of the word "failure" is incriminating in this context, "refusal to adopt the common language" would have made the sentence less offensive…[54]

As this suggests, Dijkstra rejected the managerial emphasis of programming languages research.

In some respects, the development of a defense department language continued a long tradition of attempting to improve programming languages. But in other respects, it reflected the tool-orientation of managers eager to gain control over unwieldy, complex software projects. Dijkstra was among several software experts who rejected the notion that any language or tool would render software design a simple matter for management.[55] Despite such objections, the development of a unified Defense Department programming language continued, and in April of 1979, the new language was dubbed "Ada" in honor of Charles Babbage's "programmer." [56]

2.2. Software Engineering: Discipline or Fad?

[53] Ibid.

[54] Ibid.

[55] By 1984 he would dub this the quest for the Stone and the Elixir: "The quest for the ideal programming language and the ideal man-machine interface that would make the software crisis melt like snow in the sun had —and still has!— all the characteristics of the search for the Elixir and the Stone." Dijkstra, E. W. (1984) 'The threats to computing science', in, *ACM South Central Regional Conference* (Austin, Texas. Parnas' critique of the emphasis on programming languages, and Fred Brooks' famous "No Silver Bullet" essay in the mid-1980s made much the same points.

[56] Whitaker, William A (1996) 'ADA - The Project: The DoD High Order Language Working Group'. Although Babbage's proposed calculating machine was never built, his designs were given popular currency by Ada Augusta, the daughter of Byron King and the Countess of Lovelace in the early nineteenth century. By February of 1978, the group was on the verge of choosing a name for the new language, but they waited to obtain permission from the family. In May of 1979, , the group selected a contractor for development: CII-Honeywell Bull. The same month, a group known as the "Ada Implementors," began to meet informally, and they soon chose to affiliate themselves with the ACM (rather than the IEEE), eventually becoming a Special Interest Group, SIGAda, in 1984. Whitaker, William A (1996) 'ADA - The Project: The DoD High Order Language Working Group'.

Even as the defense department invested in new programming languages, software engineering began to emerge as a new and distinct field of research and development.[57] The new field partly reflected the interests of a new IEEE (Institute of Electrical and Electronics Engineers) Computer Society, formed in 1971. The Special Interest Group on Programming Languages (SIGPLAN) continued to find its home in the Association for Computing Machinery, the oldest and most academically-oriented computing association.[58] In 1973, the SIGPLAN helped establish a new symposium on the Principles of Programming Languages (POPL), which met annually beginning in 1975.[59]

Around the same time, both the IEEE and the ACM began to sponsor separate conferences focused on “software engineering.” The IEEE Computer Society was one principle sponsor of a 1975 “National Conference on Software Engineering,” which met under the general chairmanship of Harlan Mills of IBM and Dennis Fife of the National Bureau of Standards.[60] That same year, the IEEE Computer Society joined with the ACM in co-sponsoring the International Conference on Reliable Software (ICRS). The Computer Society also established the *Transactions on Software Engineering* for the publication of serious, peer-reviewed articles in the new field.[61]

Nonetheless, the “field” remained relatively ill-defined. Some of the most frank discussions were encouraged by a new ACM Special Interest Committee on Software Engineering (SICSOFT), formed in 1975. Thomas B. Steel, the program chair for the 1975 NCSE, also served as the first chair of SICSOFT.[62] Prefacing the group’s inaugural newsletter, *Software Engineering Notes*, Steel attempted to put the field of software engineering “in perspective” with a short anecdote about a priority dispute among professional groups:

[57] The split between these communities has also been discussed by Ryder, Barbara, Soffa, Mary Lou and Burnett, Margaret (2005) 'The Impact of Software Engineering on Modern Programming Languages', *ACM Transactions on Software Engineering and Methodology* 14/4 (October): 431-77.

[58] Nathan Ensmenger has discussed the academic orientation of the ACM in the 1950s and 1960s. Ensmenger, Nathan J (2003) 'Letting the ‘Computer Boys’ Take Over: Technology and the Politics of Organizational Transformation', *Int’l Review of Social History* 48: 153-80. This persisted in the 1970s; see: http://doi.acm.org/10.1145/359842.363633

[59] POPL was co-sponsored by the ACM Special Interest Groups on Programming Languages (SIGPLAN) and on Algorithms and Computation Theory (SIGACT). SIGPLAN first formed as a special Interest Committee on Programming Languages in 1966. See first newsletter at: http://portalparts.acm.org/1320000/1317439/fm/frontmatter.pdf

[60] The NCSE was cosponsored by the National Bureau of Standards and IEEE Computer Society.

[61] See listing of first volume at: http://www.informatik.uni-trier.de/~ley/db/journals/tse/tse1.html It does not appear to be online.

[62] The first newsletter published two articles from the NCSE, suggesting that this group may have emerged from the conference.

> SURGEON (confidently): I need only quote Genesis II, 21-22, to prove that the surgeon's art was prerequisite to that which is traditionally viewed as the world's oldest profession. ‘And the Lord God caused a deep sleep to fall upon Adam, and he slept; and He took one of his ribs, and closed up the flesh instead thereof…”
>
> ENGINEER (patronizingly): You forget, sir, that according to Genesis I, 1, ‘In the beginning God created the heaven and the earth’. Certainly, constructing the whole universe out of chaos is a feat of engineering.
>
> PROGRAMMER (triumphantly): Who do you suppose created chaos?

As Steel explained, the aim of SICSOFT was “to reduce the chaos in every way possible.”[63]

Though complex (and often chaotic) systems served as a common focus, the field remained ambiguous. Peter Neumann, the editor of *Software Engineering Notes*, invited a broad range of contributions in his introductory letter:

> I eschew setting explicit boundaries on what is an acceptable contribution. General relevance, brevity, and good taste are perhaps the only essential criteria. Contributions such as ‘Software Engineering, Photosynthesis, and Religion’…and ‘A Methodology for Developing Precisely Specified Reliable Software using Hierarchically Structured Design, Top-Down Implementation, Modeling, Simulation, Testing, and Proving, with Applications to Predicting the Performance of an On-line Interactive Multinational Computing Network Programmed in Natural Hopi without GOTOs by Indian Chief Programmer Teams’ are neither suggested nor excluded. Debate as to whether software engineering includes or is included by other disciplines seems worth avoiding. Personal attacks should be avoided, although you may SIC SOFTly and carry a big shtick. More would-be controversy over unstructured programming without GOTOs versus structured programming with GOTOs is probably unnecessary, but good technical controversy is welcomed.

While implicitly acknowledging divisions, Neumann concluded in hopes that SICSOFT might “make real the use of the word ‘engineering’ in software engineering.”

As an informal newsletter, *Software Engineering Notes* typically carried letters from the Chairman and editor, a section on new abstracts in software engineering, some technical papers, and news about upcoming conferences, funding opportunities, and ongoing business of the special interest group, and a smattering of humor throughout. The Special Interest Committee on Software Engineering grew rapidly, to 2000 members within its first two years. This made it one of the largest such groups within the ACM, and in 1977 it transitioned from a Special Interest Committee to a Special Interest Group, SIGSOFT.[64]

[63] Steel, Thomas (1976) 'Letter from the Chairman', *SIGSOFT Softw. Eng. Notes* 1/1 (May): 1.
[64] Ibid.

Nonetheless, questions about the identity of “software engineering” remained. In April of 1977, a new chairman, Anthony I. Wasserman, confronted the question directly: “Is software engineering a fad?” Wasserman was hearing the question “rather frequently,” and acknowledged that it was “perfectly legitimate…since the field appears to have largely sprung up in the last two years.” It suddenly seemed that “any activity or organization with the words ‘software engineering’ (or something vaguely related to that) has…a virtual guarantee of success.” Wasserman felt that “the field has attracted a number of people who think (or hope) that it will provide a panacea for all of their problems in software production…” Acknowledging that “software engineering remains more of a wish than a reality,” he anticipated that some early interest would diminish. Nonetheless, he felt that “the discipline seems like it is here to stay,” in part because it was so new: “We have only begun to scratch the surface of the subject … There is still a way to go before we can certify the correctness of software products.” He anticipated that reaching the goals of software engineering would “take a long time; it is definitely not for faddists.”[65]

In addition to publishing ongoing commentary on the field, SIGSOFT began to co-sponsor a variety of conferences on issues surrounding software design and development. The most regular of these continued the 1975 National Conference on Software Engineering. Beginning in 1976 as the 2nd International Conference on Software Engineering (ICSE), the ICSE continued to meet yearly or every other yearly thereafter.

Like the NATO-sponsored Software Engineering Conferences, the new conferences represented an international community that centered on the United States and Europe, with growing involvement from Japan. But the software engineering conferences of the mid-1970s shared little else with their predecessors. The late 1960s conferences were initiated by the NATO science committee, a quasi-governmental organization, and their elite participants were hand-picked. By contrast, the mid-1970s conferences came at the initiative of professional groups who sought government support only occasionally. Their organizing committees put out an open call for papers, and acceptance rates tended to hover around twenty percent or less.[66] The mid-1970s conferences represented significantly different, and ultimately more lasting, constituency than

[65] Wasserman, Anthony I (1977) Ibid. 2 (January).

[66] According to Neumann, the 2nd International Conference on Software Engineering accepted less than fifty out of 225 papers. There are more than fifty “papers” online, though some are actually only abstracts. Neumann also reported that only 19 out of 80 proposed papers were accepted to the Conference on Language Design for Reliable Software. Neumann, Peter Ibid. 'Letter from the Editor', 5 (October): 1-2.

the NATO-sponsored meetings (see Graph IIA-D, Table II, and discussion in the Appendix). Wasserman observed in 1977: "Few people track software engineering back to Garmisch in 1968."[67]

In 1977, SICSOFT was permanently established as a new Special Interest Group, SIGSOFT, and with over 4000 members, it became the third largest special interest group within the ACM, behind only the Programming Languages and Operating Systems groups.[68]

But just what was software engineering? Nobody knew for sure. When Neumann asked readers of SEN what they wanted to read more about, the answers were diverse. He noted that the most requested topic was "design methodology, followed by programming methodology." But the list went on:

> **A**lso frequently mentioned were tools, then "practical experience" and management topics. Also worthy of note were proofs, psychological and philosophical aspects of software engineering, software metrics, reliability, and security. Several readers wanted more on interfaces, e.g., between theory and practice, between "computer science" and "software engineering", and between technical and management issues.[69]

With such diverse demands, Neumann chose to keep the content of SEN broad.

2.3. Designers, Managers, and their Tools

Distinctive approaches to software engineering were not always easy to reconcile. Divisions between the programming languages community and software engineering partly marked a broader split in the priority given to tools (whether languages or management tools). For example, consider the dispute engendered in 1977 when SIGSOFT joined the two other largest Special Interest Groups to co-sponsor a "Conference on Language Design for Reliable Software."[70] One panel devoted itself to discussing the *limits* of language design.

James C. King of IBM's Watson research labs, began his contribution to the panel: "Offering a critique on programming language design provides one the open opportunity for cheap shots."[71] Although all of the panelists took their shots, the most thorough going crack

[67] Wasserman, Anthony I Ibid. 'Letter from the Chairman', /1 (January): 1.
[68] Wasserman, Anthony I (1978) 'Letter from the Chairman', *SIGSOFT Softw. Eng. Notes* 3/1 (January): 1.
[69] Neumann, Peter Ibid. 'Letter from the Editor'.
[70] These were the Special Interest Groups on Programming Languages, or SIGPLAN, and Operating Systems, SIGOPS. See discussion in Ryder, Barbara, Soffa, Mary Lou and Burnett, Margaret (2005) 'The Impact of Software Engineering on Modern Programming Languages', *ACM Transactions on Software Engineering and Methodology* 14/4 (October): 431-77.
[71] King, James C (1977) 'Panel: Limitations of Language Design for Reliable Software', *SIGSOFT Softw. Eng. Notes* 2/3: 5-6.

came from David Parnas, who we will meet again in Chapter 9. Parnas had good reason for interest in programming languages. He was a student of Alan Perlis, who won a Turing award for helping design ALGOL programming language, was a veteran of the SAGE project, and a central figure at both of the inaugural Software Engineering Conferences. By 1977, Parnas was a rising star in the field of software, and a professor of computer sicence at the University of North Carolina at Chapel Hill.

In a paper entitled “Designing Reliable Software in BLOWHARD,” Parnas argued that “improvements in software reliability are best obtained by studying software design not by designing new languages.” BLOWHARD, an acronym for “Basic Language – Omnificent with Hardly Any Research or Development” was a farce – its principle feature was the lack of options offered to programmers. It was described to highlight the dangers of an “alternative farce:” “a language with so much built into it that any system was but an invocation of one of its features.” Parnas argued that most languages at the conference would do better to move towards the extreme of BLOWHARD.

As Parnas argued in the panel discussion, it was crucial to think about software design before language design: And, as Neumann paraphrased the discussion, “the choice of construction tools is clearly influenced by whether one is working with wood or with metal!” Additionally “a good designer can produce a good product with all but the worst tools. A bad designer can produce a bad product with the best of tools.”[72] In a follow-up letter written to Neumann as the editor of *Software Engineering Notes*, Parnas explained that neither he nor others on his panel were “against research in language design per se,” but felt that there were “so many projects designing new languages and so few evaluating languages and methodologies by building useable software.”[73]

Parnas was among those who gave software engineering a more pragmatic bent than the programming languages community. A relatively simple comparison of paper titles presented at ICSE and POPL meetings between 1976 and 1985 shows that the software engineering meetings were much more likely to focus on management, production, costs, and reliability issues. By contrast, the programming languages community was more concerned about methods of formally proving that software was correct (see Table III).

[72] Neumann, Peter Ibid. 'Panel Discussion on the Limitations of Language Design for Reliable Software', (April): 13-15.
[73] Parnas, David L. Ibid. 'Building reliable software in BLOWHARD': 5-6.

2.4. Pragmatism and Proof

Perhaps the sharpest disagreement in the late 1970s emerged around efforts at formal proof. As discussed previously, concerns about securing military computer networks were one chief rationale for funding research into formal proof of program correctness. But lofty standards of formal proof were repeatedly lowered by their confrontation with the formidable complexity of real-world systems.

For example, in 1973, Peter Neumann began leading an ARPA-sponsored effort to develop a Provably Secure Operating System (PSOS). Seven years later, the final report noted that "PSOS...might be considered an acronym for a 'potentially secure operating system,' in that PSOS has been carefully designed in such a way that it might someday have both its design and its implementation subjected to rigorous proof."[74] At UCLA, researchers aimed to verify a kernel that would allow Unix to run securely on a minicomputer; this encountered similar challenges. An effort to develop a provably secure military computer network as part of the WWMMCCS, Autodin II, was cancelled in 1982 in favor of encryption.[75]

Although the holy grail of provably correct computer programs was largely unachieved, software developers frequently endeavored to make their software development verifiable. Nonetheless, many pragmatists recognized that formal methods were limited, not only because of complex mathematics, but because of the social nature of mathematics. In 1975, at the International Conference on Reliable Software, Harlan Mills had noted a crucial "distinction between correctness and capability." He noted: "Determining what a program should do is usually a much deeper problem than writing a program to do a predetermined process. It is the latter task that you can do correctly." He went on:

> There is no foolproof way to prove that a program is correct. This fundamental difficulty is not in programming, but in mathematics - because the schoolboy idea of mathematics (or logic) as a body of

[74] This was an ARPA initiative, no doubt sparked by Ware's report. According to PGN, Larry Roberts from DARPA called him up in 1971, looking for a study on fault tolerance; PGN takes this on (see e-mail on 6/13/07) MacKenzie, Donald (2001) *Mechanizing Proof: Computing, Risk, and Trust* (Cambridge, MA: MIT Press).

[75] Between 1977 and 1981, a group at the University of Texas, Austin, succeeded in specifying, designing, and verifying a prototype Encrypted Packet Interface for the Navy's computers to use with the ARPAnet, but it was never deployed. Ibid.

> super-natural vertices and infallible procedures is simply not so. Mathematics is a human activity subject to human fallibility. It has no basic secrets of truth or reason.[76]

Mills went on to emphasize that mathematics was useful, and should be applied as much as possible to ensuring that programs were correct. Nonetheless, he admonished software experts not to go “looking for some schoolboy magic to replace your own responsibility.”[77]

Few, if any, software experts took issue with Mills’ comments. However, a similar set of observations sparked strident debate in the late 1970s. At the 1977 Principles of Programming Languages meeting, Perlis and two younger computer scientists—Richard Lipton and Richard DeMillo—published a paper reflecting on the social nature of proof. Lipton, a student of Parnas’ (and therefore an intellectual “grandson” of Perlis), was then a professor of computer science at Yale. He and DeMillo, then a professor at Georgia Tech, began collaborating during the mid-1970s. As we will see in Chapter 9, it was a lasting collaboration.

In their paper in the late 1970s, DeMillo, Lipton, and Perlis reiterated that mathematics and programming were social processes. But they went further by suggesting that program proving would not be subject to the social processes that made most mathematical proofs trustworthy:

> The theorems are neither simple nor basic, and the proofs of even very simple programs run into dozens of printed pages. Thus, the incentive for a community to access and assimilate the “proof” of an individual piece of software is no longer present; proofs simply will not be read, refereed, discussed in meetings or over dinner, generalized, or used. They are too specialized for any of those activities. The sad fact is that the kind of mathematics that goes on in proofs of correctness is not very good.[78]

This was the basis for their primary conclusion: “program verification…is bound to fail in its primary purpose: to dramatically increase one’s confidence in the correct functioning of a particular piece of software.” They argued that software engineers would do well to follow the example of “mature engineering disciplines,” where “‘reliable’ never means ‘perfect’ in the monolithic, classical notion of ‘perfection’.” Instead, they suggested that engineers should establish limits of confidence in programs.[79]

[76] Mills, Harlan (1975) 'How to write correct programs and know it', in, *Proceedings of the international conference on reliable software* (Los Angeles, CA: 363 - 70.
[77] Ibid.
[78] DeMillo, Richard A, Lipton, Richard J and Perlis, Alan J (1977) 'Social Proceses and Proofs of Theorems and Programs', in, *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*: 206-14.
[79] They went on: “Second, engineers tend to counteract the human tendencies to try “untested” designs by limiting the amount and kind of innovation in designs. This suggests that engineers tend to “recycle” designs and to thus subject them to some of the same social processes as are present in mathematics.” Ibid. in.@ 211-12.

All of this caused Dijkstra much consternation. He dubbed the paper "A Political pamphlet from the middle ages," objecting to its premise:

> How do they know that [program] proofs are not communicated and subjected to the judgment of others? Simply by painting them as so long, ugly, and boring that they are, now almost by definition, not fit for communication. They just ignore that how to prove —not in the silly ways they depict, but more elegantly— "the correct functioning of particular pieces of software" is the subject of a lively interchange of experiences between scientists active in the field.[80]

Feeling accused of irrelevance, Dijkstra accused the group of being anti-intellectual:

> Unaware that the 'problems of the real world' are those you are left with when you refuse to apply their effective solutions, they confirm the impression of anti-intellectualistic reactionaries by sentences such as "real programs deal with real human activity and are thus detailed and messy" (their italics and their conclusion!). They boldly postulate that '...the transition between specification and program must be left unformalized'.

This letter soon occasioned an extended discussion in *Software Engineering Notes*.

In replying to Dijkstra's critique, DeMillo, Lipton, and Perlis insisted: "We must begin with by refusing to concede that our confidence in a piece of real software has ever been increased by a proof of its correctness – real software, such as airline ticketing programs that issue real airline tickets or classroom scheduling programs that schedule real classes." While acknowledging that questions of confidence were subjective, they pointed to "the general practice" of others in the field of software: "Do they act as if program verifications increased their confidence in programs? Do adherents of program verification verify their own programs? Our impression is that they do not…." While acknowledging that formal verification was "the subject of lively interchange among scientists in the field," they insisted that it was irrelevant: "the verifications themselves are long, ugly, and boring, no matter how concise, elegant, and fascinating the idea of verification may be."[81]

The exchange between Dijkstra, deMillo, Perlis, and Lipton sparked considerable discussion in the software engineering community. Parnas found the conflict "distressing" because the two sides were "driven to extreme positions." Like Harlan Mills, he suggested that

[80] Dijkstra, Edsger W. (1978) 'On a political pamphlet from the middle ages', *SIGSOFT Softw. Eng. Notes* 3/2: 14-16.

[81] DeMillo, R. A., Lipton, R. J. and Perlis, A. J. Ibid. 'Response to: "On a political pamphlet from the middle ages"': 16-17.

mathematics could be useful without being fool-proof or elegant. Others noted that efforts to verify programs had proven useful in real-world applications.[82]

As this suggests, dispute over the value of proof reflected strongly held beliefs about competing goals and ways of life, especially, between pragmatists and purists. Software engineering remained pragmatic, but it did not dismiss program verification altogether. When we consider “general terms” assigned to papers at ICSE conferences, we find that design, management, and performance were among the most common terms, suggesting the pragmatic, end-use orientation of software engineering (See Chart II and appendix). Nonetheless, these terms were commonly used in connection with verification (See Graphs III.A-E, and Graphs IV.A-E). Although verification tended to be less central to the conferences, and theory less central still—suggesting the pragmatic, end-use orientation of software-engineering—the connections between these interests is significant. Nonetheless, individuals did not tend to cluster around nodes, or to be located in extreme positions.

3. Managing Complex Systems: Software Engineering Goes Public

While diverse (and frequently conflicting) research agendas continued to appear in formal articles and papers on “software engineering”, a new theme began to emerge in the informal pages of Software Engineering Notes during the late 1970s: learning from failure. One of the first individuals to emphasize the value of learning from failure was Robert Glass, computing consultant and industry writer who was then affiliated with Boeing. At a Software Quality and Assurance Workshop in November of 1978, Glass presented a short paper on “Computing Failure, which was published along with the other proceedings in a special edition of Software Engineering Notes the same month. Glass noted that while “most professional papers” focused on success, “it is well known that some of our most lasting learning experiences are based on failure.” He went on to provide an amusing account with “an underlying message that sharing the sometimes embarrassing truths about What Goes Wrong In Our Field is at least as illuminating as more serious discussions about Things That Look Promising.” He drew five morals from the story—that in the software world:

A. All that glitters is not gold.

B. Never count your breakthrough chickens before they're hatched.

[82] See for example: Van Ghent, Roger (1978) 'Social Proceses and Proofs of Theorems and Programs', *Software Engineering Notes* (July): 20-21.

> C. Tools cannot substitute for talent.
>
> D. Breakthroughs cannot be scheduled.
>
> E. Quality control is often only skin deep.[83]

Glass went on to publish several books about computing fiascos and other interesting anecdotes.

While the failures dealt with in Glass’s article primarily involved a corporate failure – something with financial stakes – the columns of Software Engineering Notes soon began to reflect concern about safety-critical software systems. In April of 1979, Neumann took note of some “SENsitive NEWS ITEMS.”

> Software engineering issues have recently been creeping into the news. Computer system security penetrations have been increasing (e .g, the 23 March 1979 news mentioned a $1.1 million fraud at the City National Bank in Beverly Hill. (Peanuts compared to the $10 .2M Security Pacific scam last October!) The March media have also reported a bug in a computer program used to design nuclear reactors. In the long run, this is an enlightening bug – although its immediate effect is not (the shutdown of five nuclear reactors).[84]

Neumann went on to publish an editorial revising the theme of “Program correctness and social processes.” He felt that the reactor shutdown pointed to three problems: “program bugs (in simulations, analyses, real-time control and operating systems), fundamental inaccuracies in the underlying requirements and models, and the intrinsic gullibility of the computer illiterate— including many administrators and engineers as well as laypersons.” In conclusion, Neumann noted: “As I am writing this, a radiation leak has just occurred at the Three Mile Island nuclear plant #2 near Harrisburg PA. Although probably not computer-related, this accident resulted from operator error and multiple faults.”[85]

The “normal accident” at Three Mile Island occurred on March 28, and Neumann’s editorial went to press in April, 1979. In the months and years that followed, not a single edition of *SEN* was published without Neumann’s including commentary about glitches. In July, he was spurred on by the request of Anita Jones, who, in connection with an upcoming Symposium on Principles of Operating Systems, was seeking “true stories on system (mis)behavior, e.g., that "surprises, baffles, spoofs, or even hoodwinks its designers and keepers.” Neumann reframed the request for the software community: “Humorous or socially relevant tales of system software are desired, pertaining to bugs, reliability, security, the human condition, social disasters, etc.”

[83] Glass, Robert Ibid. 'Computing Failure: A Learning Experience', 3/5 (November): 18-19.
[84] Neumann, Peter (1979) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 4/2 (April): 2.
[85] Neumann, Peter (1979) 'An Editorial on Software Correctness and the Social Process', *SIGSOFT Softw. Eng. Notes* 4/2 (April): 3-4.

In the October edition of *SEN*, Neumann revisited the theme of safety-critical software systems, discussing a pacemaker containing a microcomputer.[86] He was prompted by a letter from James Horning, a prominent software expert. Having graduated from Stanford with a PhD in 1969, Horning went on to become a founding member of the Computer Systems Research Group at the University of Toronto, quickly distinguishing himself with his work on languages for writing programs that could be verified.[87] By the time he wrote to Neumann, he had moved to Xerox Palo Alto Research Center.

Horning made what he felt was a “SAFE (but appalling) PREDICTION”: that within a decade “there will be several major disasters that are directly (and properly) attributed to faults in computer programs: airplane crashes, or nuclear accidents, or bridge collapses, or ....” He described “a recurrent nightmare in which one of these disasters has led to a court case,” with the prosecutor or plaintiff accusing the program developer of negligence, and the defendant accusing the user of negligence: the user “relied on the correct functioning of a large program, and no large program is fault-free.” In the nightmare, Horning confronted three choices:

> Testify for the prosecutor that the developer was negligent.
>
> Testify for the defendant that the negligence was in relying on the program.
>
> Chicken out and refuse to testify.

Delivering these remarks before an audience of about 200 people, Horning asked what they would do. He recounted:

> 25% indicated that they would testify for the prosecutor,
>
> 30% indicated that they would testify for the defendant, and
>
> 45% indicated that they would chicken out.

With this in mind, Horning confronted the cost-centric nature of software engineering in the late 1970s:

> Most of those who actually pay for computer programs (and who therefore determine the priorities for their development) do not care significantly about their reliability:
>
> They have not effectively demanded reliability from program developers.
>
> They have not indicated a willingness to pay a premium for reliability.
>
> Until they do, improvements in reliability will merely be “side effects” of efforts to improve things that they do care about, such as the cost of programming.

[86] Neumann, Peter (1979) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 4/4 (October): 2.
[87] Horning was co-author on one of the few papers to be accepted to the 1977 Conference on the Design of Reliable Software.

Horning challenged readers to find ways of increasing the importance of reliability to “those who pay the bills.”[88]

For his part, Neumann continued to publish short notes about glitches and other software problems in each edition of Software Engineering Notes.[89] By October of 1980, “Computer-Related Disasters” had become a regular theme in the editor’s letter, though that month Neumann offered some unusual “encouraging news”—despite “volcanoes, hurricanes, earthquakes, wars, primaries, an erupting missile silo, and so on this summer…computer systems seem to have done fairly well.”[90]

3.1. Computer-Related Risks: Expanding User Groups, Heightened Stakes

Though there were encouraging signs in the summer of 1980, virtually every edition of SEN brought more news of computer glitches in the ensuing years. Significantly, many reports of computer-related risks came from popular accounts, rather than peer-reviewed publications. No doubt this was partly because, as Glass suggested, software researchers tended to report on “successes” of their companies. But in the late 1970s and early 1980s, growing popular attention to computer-related risks followed two broader changes.

First, the rise of the microelectronics industry changed the popular meaning of software, even as it made computing more accessible to the general public. The changes began as early as 1969, when IBM announced that it would “unbundle” software services from hardware. Whereas before IBM had sold hardware with all forms of software included – meaning programmers to set up the computer with various applications – now users would be charged for specific applications. This helped open the market to competitors and fueled a small but growing “packaged software” industry.[91]

[88] Horning, Jim (1979) 'A Note on Program Reliability', *ACM SIGSOFT, Software Engineering Notes* (October): 6-8.

[89] October 1979
The journals are filled with much noise
about modules and specs and such joys.
Although it's endearing,
software engineering
won't help if you're just building toys.
The Naval Research Laboratory is expanding its software engineering research and development activities. NRL develops and studies advanced techniques in the context of non-trivial applications.

[90] Neumann, Peter (1979) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 5/4 (October): 1.

[91] For more on the unbundling decision, see: Campbell-Kelly, Martin (2003) *From Airline Reservations to Sonic the Hedgehog* (Cambridge, MA: MIT Press).

This in turn shifted public conceptions of the meaning of software. In 1971, concerns about the economic implications of the changing software industry prompted the Government Accountability Office to recommend that the General Services Administration take a leading role in negotiating favorable prices on behalf of government user agencies, that it encourage computer program sharing, and even consider “buying outright the computer software products which are used widely…throughout the Federal Government…”[92] As these comments suggest, the growth of the packaged software industry contributed to popular perceptions that software was an artifact rather than a social process of setting up a computer for use.

As Thomas Haigh and Martin Campbell-Kelly have discussed, change did not come overnight. In 1971, two years after the unbundling decision, experts noted that while packaged software was a “potential market” of “tens of thousands” of systems, this remained unrealized: “even highly successful packages have at this point sold only in the neighborhood of fifty systems with a mere handful of outstanding success stories claiming sales in the hundreds.”[93]

Nonetheless, by the late 1970s and early 1980s, software engineering was morphing to match the changing industry. When the ICSE met in Tokyo in 1982, a new emphasis on microelectronics and experiments appeared in its papers.[94] This marked a significant shift from its early focus on major military-industrial computer systems.

As early as 1979, one admittedly “presumptuous” presentation at ICSE discussed the “role of software engineering in the microcomputer revolution.” As the author noted, this title presumed not only “that software engineering exists and that you and I have a reasonably common understanding of it,” but also “that there is a microcomputer revolution and we know what it is.” Despite such ambiguities, the author was convinced of a “conflict in trends.”

> We are expanding so fast that industry is robbing the university cradle (the number of PhD’s granted is dropping each year), as well as pressing into service a large number of people without much formal training. The average skill level in the industry seems to me to be dropping, and along with it the quality of software.

[92] http://archive.gao.gov/f0302/095611.pdf GAO (1971) 'Acquisition and Use of Software Products for Automatic Data Processing Systems in the Federal Government; Report to the Congress', in (Washington, D.C: General Accounting Office).

[93] Quoted in Haigh, Thomas (2002) 'Software in the 1960s as Concept, Service, and Product', *IEEE Annals of the History of Computing* 24/1 (January-March): 5 - 13.. Original source is: R.V. Head, *A Guide to Packaged Systems*, Wiley Interscience, New York, 1971, p. 66.

[94] This observation comes from an analysis of the Classification codes assigned to papers at ICSE. Codes corresponding to microcomputers and microelectronics only began to appear during this period. The data is not shown here in the interests of brevity.

> At the same time the user community is expanding to approach the entire literate population with programs running microwave ovens, automobile engines, hand calculators, telephones, airplanes, banks, gas pumps and heart patients. It is a very inelastic audience. Nearly any problem is beyond the skill of the user to understand, or work around, much less to repair.[95]

Neumann’s discussion of computer-related risks partly tracked the conflict in trends. The editor’s page reflected upon the risks of microprocessors for pacemakers, breakdowns of the Bay Area Rapid Transit system, and other failures in computer systems for popular use (broadly construed), even as they made headlines in the mainstream news. The same month that he reported on the first system-wide shut down of the ARPAnet—generally known as the ultimate in reliability and survivability—he took note of a news item: “Microcomputers are beginning to be used quite widely in Europe for safety purposes in rail transport, medical, elevator, power, oil, and chemical industries.”[96]

At the same time that the scope of potential computer users expanded, the most long-standing and significant “user group” for complex, real-time computer systems—operators of military command and control systems—was confronted with new challenges that captured mainstream media attention. As we will discuss in the next two chapters, concerns about the survivability and responsiveness of nuclear command and control systems grew in the early 1980s, as members of the Reagan administration began to speak openly about winnable nuclear war, and as tension grew with the Soviet Union.

Public commentary would be dominated by a well-established advisory elite—mostly physicists and economists. It would focus on issues of blast resistance, Electromagnetic Pulse issues, and other physical or economic problems. Nonetheless, evidence strongly suggested that the system would face massive organizational challenges, in part due to an exceedingly complex, computerized system that was prone to failure. The fallibility of command and control was not new. However, for the first time, a community of software experts was taking notes on these kinds of problems.[97] Computer experts, including Horning, Neumann, and others who had been studying the risks of complex computer systems, would begin to add their voices to the mix.

[95] McKeeman, W.M. (1979) 'The role of software engineering in the microcomputer revolution: An overview', *Proc. Fourth Int'l Conf Software Eng*: 340.

[96] Neumann, Peter (1981) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 6/1 (January): 1-2.

[97] For example, in July of 1980, when the World Wide Military Command and Control System experienced significant glitches, Neumann posed a question to his readership: “where should we find editorials on system reliability in mankind-critical hardware-software systems?”[97] He continued:

4. Conclusion

This is work in progress. To be continued.

Appendix: Notes on Network Analytic Methods and Qualitative History

[EXTRA DRAFTY]

Social scientists have long used network data to analyze the structure of communities, the place of individuals within communities, and the consequences of such relationships. For example, network analysis can demonstrate who is most central to a community, who serves as a broker for relations between other individuals, and who is “on top” in non-reciprocal patterns of exchange. It can also be used to measure cohesiveness, centralization, or other community-wide attributes.[98] In principle, such methods should be useful for scholars in science and technology studies, because they allow for the emergence of community through *practices*.[99] This aligns well with recent work in STS, which has emphasized the need to move beyond analyses of science and technology as well-bounded “disciplines” or “professions”, and to focus on the material practices that allow for communities to emerge and change over time.

In practice, dynamic network analysis has received somewhat less attention than its static counterpart.[100] This is not the place for a full review, but several studies which draw attention to the potential for such methods to help explain *causes* of change in a field are worth mentioning

“Nowhere”, you might say, because nobody really seems to worry much about it? "Elsewhere", you might say, since managers tend to blame the applications people, the applications people blame the operating system people, and the OS people blame the hardware people (and vice versa), and now everyone can blame the cosmic gremlins. Nevertheless, very few systems people seem to worry about human issues. (Just look at the user interfaces of some of your favorite systems!) ‘In SIGSOFT’, you might say, "because our editor keeps poking his nose into everyone's business.” Neumann, Peter (1980) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 5/3 (July): 2.
OK, folks, what about the reliability of missile detection and launch systems? How about the reliability of air traffic control systems? Both were prominent in the news in the weeks before this issue went to the printer.

[98] For a comprehensive book on methods, see: Wasserman, S. and Faust, K. (1994) *Social network analysis: Methods and applications*: Cambridge Univ Pr).

[99] For a discussion of the relationship between structure, agency, and network analysis, see: Emirbayer, Mustafa (1994) 'Network Analysis, Culture, and the Problem of Agency', *The American Journal of Sociology* 99/6 (May): 1411-54. For a discussion of the conditions that allow a “social movement” to become institutionalized as a field, see: Frickel, Scott and Gross, Neil (2000) 'A General Theory of Scientific/Intellectual Movements', *American Sociological Review* 70 (April): 204–32.

[100] I have not conducted my own survey of the field, but the point is made persuasively in: Powell, W. W., White, D. R., Koput, K. W. and Owen-Smith, J. (2005) 'Network Dynamics and Field Evolution: The Growth of Interorganizational Collaboration in the Life Sciences I', *American journal of sociology* 110/4: 1132-205.

in the present context. Within organizational studies, recent work has shown how specific kinds of relationships contribute to the expansion of networks within fields such as biotech.[101] International relations scholars have also placed special emphasis on network dynamics as an alternative to traditional explanations for international organization (which have tended to treat power and resources in more static terms).[102]

Finally, it should be noted that historians of science and technology have long deployed network analysis in an effort to understand the changing structure of scientific knowledge. Derek DeSolla Price, commonly referred to as the father of scientometrics, used such methods to study scientists’ citation patterns in the mid-1960s, and similar analyses continue today.[103] However, in so far as such analyses tend to rely upon publication databases, they tend to encourage a “history of ideas” approach to scientific and technological change. By contrast, the past three decades of work in science and technology studies have emphasized the importance of material culture, informal exchanges, and other kinds of practices that are not neatly captured by publication databases. Ethnographic and other qualitative methods have gone a long produced rich accounts of change in the cultures of science and technology, deploying a “thick description” that is easily flattened by quantitative methods.[104]

How then, can these tools aid STS scholars dedicated to thick description? In this chapter-in-progress, I have used network analysis of bibliographic data as a way of testing hypotheses and questions that arose in the process of conducting qualitative historical research. The following two questions were of particular interest:

1. Did the sudden rise of “software engineering” in 1975-6 correspond to real changes in publishing and research networks—was a new group of people working together for the first time? Or did this term simply rebrand work that was already under way?

[101] Ibid.

[102] For a review, see: Hafner-Burton, Emilie M., Kahler, Miles and Montgomery, Alexander H. (2009) 'Network Analysis for International Relations', *International Organization* 63/3 (Fall): 559–92. See also Montgomery, Alexander H. (2004) 'Ringing in Proliferation: How to Dismantle an Atomic Bomb Network', *International Security* 30/2 (Fall).

[103] Price, Derek J. De Solla (1965) 'Networks of Scientific Papers', *Science* 149/3683: 510-15.; more recently, see: Bettencourt, L. M. A., Kaiser, D. I. and Kaur, J. (2009) 'Scientific discovery and topological transitions in collaboration networks', *Journal of Informetrics* (July): 210-21.

[104] A minimal set of readings is: Galison, Peter (1997) *Image & Logic: A Material Culture of Microphysics* (Chicago: University of Chicago Press), Knorr-Cetina, Karin (1999) *Epistemic Cultures: How the Sciences Make Knowledge* (Cambridge, MA: Harvard University Press).

2. Did commonly discussed divisions in software engineering actually map onto practices of research and publication?

In order to address these questions, I gathered and analyzed several kinds of network data related to presentations at the International Conferences on Software Engineering (ICSE). When I asked Barry Boehm to name one thing (agency, grant, project, school, anything at all) that had most contributed to the institutionalization of software engineering, he pointed to ICSE. As mentioned in the chapter, the 2nd ICSE began in 1976 as a continuance of the 1975 National Conference on Software Engineering (NCSE). While several other software engineering conferences took place during the 1970s, ICSE has persisted as the most regular conference. It was discussed and well-attended by key individuals in the book (David Parnas, Barry Boehm, and Peter Neumann, to name a few). Thus, I chose these forums as a center of analysis.

Persistence and Change at ICSE

It should be noted that while many disciplines regard conferences as relatively informal forums, where work-in-progress is acceptable or even encouraged, computer science conferences tend to accept only work of peer-review, publication-ready quality. For example, acceptance rates at ICSE between 1995 and 2009 ranged from 10 to 19 percent. As indicated in the chapter, acceptance rates in the 1970s were similarly low. Thus, presenters at these conferences should be understood as leaders in computing. Individuals who presented at multiple conferences should be understood not only as leaders in computing, but as that much more central and recognized in the specialized field of software engineering. Names of such individuals are listed in Table V.

One way of analyzing change and continuity in a field is to consider the persistence of particular personalities in such venues. Of the individuals who present work at any conference, how many return to present work again, and how often do they return? As discussed in the chapter, relatively few individuals from the late 1960s appeared in the mid-1970s version of “software engineering,” and leaders in the field at that time recognized themselves as engaged in something new. To systematically analyze continuity and change, data were gathered on presenters at each conference. Individuals who presented work more than once were classified by the year in which they first presented (marking, roughly, the period in which they became recognized in the field). This data is shown in Table II, and is used in the network analyses described below.

Co-authorship networks (1-mode data)

Rather than examining citation patterns (which can indicate many different kinds of relationships between individuals), I chose to analyze co-authorship relations. Co-authorship tends to indicate a common research agenda and shared interests. Thus I hoped it might reveal some of the structure of the field in an organic way.

A relational database containing data on co-authorship was compiled using the DBLP server:
http://www.informatik.uni-trier.de/~ley/db/
In 2008-09, Sai Prashanth (then a master's student in Computer Science at Stanford), did the work of compiling the database (written in SQL, Structured Query Language). He then developed a query that allows for an analysis of networks surrounding the ICSE conferences. For purposes of analysis, three mutually-exclusive classes of authors were defined:

P = presenters at any ICSE conference in year 19ab

C= Co-authors of P during years 19cd-19ef, who were not a member of class P

O= Co-authors of co-authors, during years 19cd-19ef, who were not a member of class C.

The search script requests input on the years to be searched, and then outputs network data. Network data were compiled as lists of vertices and weighted edges, the format accepted by Pajek network analysis software.

The results of this analysis are instructive, but are limited in two ways. First and foremost, co-author relations indicate such tight ties that the field appears more fragmented that it probably really was. An analysis of co-authorship in the two years following the ICSE conferences indicates a large number of components (defined as unconnected clusters in the network). Some individuals at the conferences frequently co-author papers, and as a result, they find a place in a relatively large component of the network. However, some of the most influential and well-regarded figures in the field are not members of these large components. For example, David Parnas and Barry Boehm, among the top presenters at these conferences, both appear in relatively small components of the field. Thus, large numbers of co-authored papers is not necessarily a sign of special prominence in the field of software engineering.

Additionally, it should be noted that many individuals who presented at these conferences – especially early conferences – worked primarily in fields *related* to software, but not necessarily conceive of themselves as centrally engaged in software engineering. Their patterns of publication do not necessarily reflect software engineering, but nonetheless strongly affect the results gathered in the manner described above. For example, among presenters at the ICSE 1976 meeting, Kapali Eswaran coauthored the most papers between 1976 and 1978 (sixteen co-authorships). However, his work was primarily associated with database management rather than software engineering (H.2.4, or "Systems" within "H.2 Database Management." See more on classification codes below). The next highest ranking presenter at ICSE 1975 was Erol Gelenbe, who co-authored 14 papers between 1976 and 1978. However, his paper was primarily classified under "G.3 Probability and Statistics" a subset of Classification node G, "Mathematics of computing."

Thus, although data on network structure were analyzed for these conferences (see Graphs V.A-C, and Table IV), it is not discussed in detail here. Publication patterns vary too widely for these metrics to prove meaningful. In principle, one or more components of the network data might be extracted to yield more meaningful data, but I have not yet pursued this line of analysis.

Author-subject affiliation networks (2-mode data)

The ACM computing classification system was used to characterize individuals' areas of research. It should be noted that the current structure of this system was adopted in 1981-82, near the end of the period under discussion in this chapter.[105] The preceding classification system was established in 1964. Though it has been periodically updated since 1982, the system has proven relatively stable since the early 1980s.

[105] For the scheme that was finalized in 1981, and announced in 1982, see: Sammet, Jean (1982) 'The New (1982) *Computing Reviews* Classification System--Final Version', *Communications of the ACM* 25/1 (January): 13-25. Several historical documents related to past classification schemes can be found online here: http://www.acm.org/about/class

The core of the system is a three-level, hierarchical classification tree with three levels of hierarchy. At the top level are eleven nodes with letter designations A through K. Subcategories and sub-subcategories are designated numbers appended to letter. For example, D is “Software,” D.2 is “Software engineering,” and D.2.5 is “testing and debugging.” At the fourth level of the tree are unnumbered “subjects.” For example, a paper assigned to “D.2.3 Coding Tools and Techniques” may also be affiliated with the subject “program editors.” “Nouns” hold a similar place in the hierarchy, and tend to refer to specific tools, programming languages, and so on.

Papers are assigned to one and only one “primary classification,” at the second or third levels of the tree. They may also be assigned one or more codes as “additional classification.” Finally, there are sixteen “general terms” that may be applied to any paper, regardless of its classification scheme. The number of general terms in papers I analyzed varied between zero and four.

In 1982, reviewers assigned classification codes to papers as they were published. Today authors are instructed to assign their own classification codes when submitting papers. I have yet to learn who assigned codes to papers written before 1982. Most of the classification codes and all of the general terms used for this analysis were existent in 1982. However, in some cases, classification codes added as late as 2009 (for example, “D.2.11 Software architecture” was established just this year, but it now appears on some papers published prior to 1982). I am guessing that their authors requested the re-designation.

Using the ACM Digital Library, I gathered data manually for each individual who presented work at the following ICSE conferences: 1976, 1978, 1979, 1981, and 1982. In the tedious process, I discovered some mistakes in both the DBLP and ACM databases, and discrepancies between the two databases. Although the co-authorship data described above is mostly reliable, there were occasions when names at ICSE conferences were missing from the results gathered using the SQL relational database and query. These mistakes were generally due to missing data in the DBLP server, though I also found a few mistakes in the ACM database. The most trustworthy data were found by viewing the original paper as posted online in pdf format. It’s quite possible that the database is simply incomplete.

For each individual at each of these conferences, I gathered the following data:

- Primary classification
- Additional classifications
- Subjects
- General Terms
- Title
- Abstract

In some cases, I also gathered information on affiliation and sponsors. Sponsorship information was not available for all papers. This data is complicated and has yet to be fully analyzed.

Although I did analyze the classification code data, these proved to be quite complex, and are not discussed here (in the interests of time and brevity). The general terms provided a useful framework for analysis. On average, "design" was mentioned most frequently, followed by "management" and "performance," as shown in Chart II and Graphs II.A-D.

Network analysis reveals interesting connections between these general terms, and between authors and terms. Graphs III A-E, and Graphs IV.A-E show how terms were connected by ICSE presenters. Both graphs represent the two-mode data shown in Graphs II as one-mode data; a line between two nodes (general terms) indicates that an individual used both terms at the conference. In Graphs III. A-E, the weight of the lines is not shown, but the Fruchterman Reingold (FR) algorithm is used to show, intuitively, the strength of the connections. Roughly speaking, the FR algorithm treats nodes as points that physically repulse one another, but are connected by springs. This draws closely related points together in space, and pushes out points that are not closely related. In some cases, I moved unconnected nodes closer to the cluster for the sake of getting all the data in the frame. I did not move any connected nodes in these graphs; the geometry of the cluster was optimized using a 2D FR algorithm.

In Graphs IV.A-E, the weight of the line is shown; heavier lines indicate strong connections between the terms. These graphs indicate strong and consistent connections between design, performance, and management, though the weights tend to vary by year. Additionally, they suggest diminishing connections to security. Somewhat surprisingly, given the dominance of economics in justifications for software engineering, economics was only rarely mentioned in the terms lists. Similarly, though standardization was a dominant theme in the defense department, it only appeared significant in Munich. Growing interest in experimentation is evident, along with diminishing interest in “security”, in the late 1970s and early 1980s conferences. And despite arguments about the value of verification, it appears to be a persistent theme in these papers.

REFERENCES

Anderson, James P (1972) 'Computer Security Technology Planning Study Vol 1', in (Hanscom Air Force Base, Bedford, MA: Electronic Systems Division of the Air Force Systems Command).

Anderson, James P (1972) 'Computer Security Technology Planning Study Vol 2', in (Hanscom Air Force Base, Bedford, MA: Electronic Systems Division of the Air Force Systems Command).

Bettencourt, L. M. A., Kaiser, D. I. and Kaur, J. (2009) 'Scientific discovery and topological transitions in collaboration networks', *Journal of Informetrics* (July): 210-21.

Boehm, Barry (1973) 'Software and its Impact: A Quantitative Assessment', *Datamation* /May: 48-59.

Campbell-Kelly, Martin (2003) *From Airline Reservations to Sonic the Hedgehog* (Cambridge, MA: MIT Press).

DeMillo, R. A., Lipton, R. J. and Perlis, A. J. (1978) 'Response to: "On a political pamphlet from the middle ages"', *SIGSOFT Softw. Eng. Notes* 3/2: 16-17.

DeMillo, Richard A, Lipton, Richard J and Perlis, Alan J (1977) 'Social Proceses and Proofs of Theorems and Programs', in, *Proceedings of the Fourth ACM Symposium on Principles of Programming Languages*: 206-14.

Dijkstra, E. W. (1984) 'The threats to computing science', in, *ACM South Central Regional Conference* (Austin, Texas.

Dijkstra, Edsger W. (1975) 'On a language proposal for the Department of Defense', in, *SIGSOFT Softw. Eng. Notes*.

Dijkstra, Edsger W. (1978) 'On a political pamphlet from the middle ages', *SIGSOFT Softw. Eng. Notes* 3/2: 14-16.

Dijkstra, Edsger W. (1982) *Selected writings on computing: a personal perspective*: Springer-Verlag New York, Inc.).

Dirnbauer, Frank R, Goertzel, Herbert B and Miller, James B (1976) 'WWMCCS ADP: A Promise Fulfilled', *Signal* 30 (May/June): 60.

Editor (1965) 'Evolution and Compatibility: 1965's Key Words in Tactical C&C', *Armed Forces Management* 11/10 (July): 44.

Editor (1966) 'How Not to Build a C&C System is Still Unanswered', *Armed Forces Management* 12/10 (July): 109.

Emirbayer, Mustafa (1994) 'Network Analysis, Culture, and the Problem of Agency', *The American Journal of Sociology* 99/6 (May): 1411-54.

Ensmenger, Nathan J (2003) 'Letting the ‘Computer Boys’ Take Over: Technology and the Politics of Organizational Transformation', *Int’l Review of Social History* 48: 153-80.

Frank, Werner L (1968) 'Software for Terminal-Oriented Systems', in, *Datamation*: 30-34.

Frickel, Scott and Gross, Neil (2000) 'A General Theory of Scientific/Intellectual Movements', *American Sociological Review* 70 (April): 204–32.

Galison, Peter (1997) *Image & Logic: A Material Culture of Microphysics* (Chicago: University of Chicago Press).

GAO (1971) 'Acquisition and Use of Software Products for Automatic Data Processing Systems in the Federal Government; Report to the Congress', in (Washington, D.C: General Accounting Office).

GAO (1971) 'Potential Problems In Developing The Air Force’s Advanced Logistics System; Report to the Committee On Appropriations, House Of Representatives', in (Washington, D.C: General Accounting Office).

GAO (1979) 'The World Wide Military Command and Control System—Major Changes Needed in its Automated Data Processing Management and Direction, Report to the Congress', in (Washington, D.C: General Accounting Office).

Glass, Robert (1978) 'Computing Failure: A Learning Experience', *Software Engineering Notes* 3/5 (November): 18-19.

Greenbaum, Joan M. (1979) *In the Name of Efficiency: Management Theory and Shopfloor Practice in Data-Processing Work* (Philadelphia: Temple University Press).

Hafner-Burton, Emilie M., Kahler, Miles and Montgomery, Alexander H. (2009) 'Network Analysis for International Relations', *International Organization* 63/3 (Fall): 559–92.

Haigh, Thomas (2002) 'Software in the 1960s as Concept, Service, and Product', *IEEE Annals of the History of Computing* 24/1 (January-March): 5 - 13.

Hirsch, Phil (1971) 'GAO Hits Wimmix Hard; FY'72 Funding Prospects Fading Fast', *Datamation* (March 1): 41.

Horning, Jim (1979) 'A Note on Program Reliability', *ACM SIGSOFT, Software Engineering Notes* (October): 6-8.

House (1973) 'Department of Defense Appropriations for Fiscal Year 1974: Part 6 Procurement', in, *Subcommittee of the Committee on Appropriations, U.S. House of Representatives* (Washington, D.C.: GPO).

House (1974) 'Department of Defense Appropriations for Fiscal Year 1975: Part 4', in, *Subcommittee of the Committee on Appropriations, U.S. House of Representatives* (Washington, D.C.: GPO): Apr. 25, 29, 30, May 1, 1974.

King, James C (1977) 'Panel: Limitations of Language Design for Reliable Software', *SIGSOFT Softw. Eng. Notes* 2/3: 5-6.

Knorr-Cetina, Karin (1999) *Epistemic Cultures: How the Sciences Make Knowledge* (Cambridge, MA: Harvard University Press).

Kraft, Philip (1977) *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York: Springer-Verlag).

MacKenzie, Donald (2001) *Mechanizing Proof: Computing, Risk, and Trust* (Cambridge, MA: The MIT Press).

MacKenzie, Donald (2001) *Mechanizing Proof: Computing, Risk, and Trust* (Cambridge, MA: MIT Press).

McKeeman, W.M. (1979) 'The role of software engineering in the microcomputer revolution: An overview', *Proc. Fourth Int'l Conf Software Eng*: 340.

McNaugher, Thomas (1989) *New Weapons, Old Politics: America's Military Procurement Muddle* (Washington, DC: Brookings).

Mills, Harlan (1975) 'How to write correct programs and know it', in, *Proceedings of the international conference on reliable software* (Los Angeles, CA: 363 - 70.

Montgomery, Alexander H. (2004) 'Ringing in Proliferation: How to Dismantle an Atomic Bomb Network', *International Security* 30/2 (Fall).

Naur, Peter and Randell, Brian (1969) 'Software Engineering: Report on a conference sponsored by the NATO Science Committee', in (Garmisch, Germany: 136.

Neumann, Peter (1977) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 2/5 (October): 1-2.

Neumann, Peter (1977) 'Panel Discussion on the Limitations of Language Design for Reliable Software', *SIGSOFT Softw. Eng. Notes* 2/3 (April): 13-15.

Neumann, Peter (1978) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 3/1 (January): 1.

Neumann, Peter (1979) 'An Editorial on Software Correctness and the Social Process', *SIGSOFT Softw. Eng. Notes* 4/2 (April): 3-4.

Neumann, Peter (1979) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 5/4 (October): 1.

Neumann, Peter (1979) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 4/4 (October): 2.

Neumann, Peter (1979) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 4/2 (April): 2.

Neumann, Peter (1980) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 5/3 (July): 2.

Neumann, Peter (1981) 'Letter from the Editor', *SIGSOFT Softw. Eng. Notes* 6/1 (January): 1-2.

Norberg, Arthur L and O'Neill, Judy E (1996) *Transforming Computer Technology: Information Processing for the Pentagon, 1962-1986* (Baltimore: John Hopkins University Press).

Parnas, David L. (1977) 'Building reliable software in BLOWHARD', *SIGSOFT Softw. Eng. Notes* 2/3: 5-6.

Paschall, Lee M (1973) 'USAF Command Control and Communication Priorities', *Signal* 28 (November): 13.

Paschall, Lee M (1974) 'Command and Control: Why the Air Force's Systems are Revolutionary', *Air Force Magazine* (July): 60.

Pearson, David E (2000) *The World Wide Military Command and Control System: Evolution and Effectiveness* (Maxwell Air Force Base, Alabama: Air University Press).

Powell, W. W., White, D. R., Koput, K. W. and Owen-Smith, J. (2005) 'Network Dynamics and Field Evolution: The Growth of Interorganizational Collaboration in the Life Sciences I', *American journal of sociology* 110/4: 1132-205.

Price, Derek J. De Solla (1965) 'Networks of Scientific Papers', *Science* 149/3683: 510-15.

Royce, Winston (1970) 'Managing the Development of Large Software Systems', in, *Proceedings, IEEE Wescon*: 328-39.

Ryder, Barbara, Soffa, Mary Lou and Burnett, Margaret (2005) 'The Impact of Software Engineering on Modern Programming Languages', *ACM Transactions on Software Engineering and Methodology* 14/4 (October): 431-77.

Sammet, Jean (1982) 'The New (1982) *Computing Reviews* Classification System--Final Version', *Communications of the ACM* 25/1 (January): 13-25.

Schemmer, Benjamin F (1970) 'DoD's Computer Critics Nearly Unplug Ailing World Wide Military Command and Control System', *Armed Forces Journal* 108 (December 21): 22.

Senate (1972) 'Department of Defense Appropriations for Fiscal Year 1973: Part I', in, *Subcommittee of the Committee on Appropriations, U.S. Senate* (Washington, D.C.: GPO).

Staff (1969) 'Can Vulnerability Menace Command and Control?' *Armed Forces Management* (July): 40-3.

Steel, Thomas (1976) 'Letter from the Chairman', *SIGSOFT Softw. Eng. Notes* 1/1 (May): 1.

Ulsamer, Edgar (1971) 'The Military Decision-Makers' Top Tool', *Air Force Magazine* (July): 44-49.

Ulsamer, Edgar (1972) 'Command and Control is of Fundamental Importance', *Air Force Magazine* (July): 42-46.

Van Ghent, Roger (1978) 'Social Proceses and Proofs of Theorems and Programs', *Software Engineering Notes* (July): 20-21.

Volz, Joseph (1970) 'Revamped World-Wide Computer Net Readied', *Armed Forces Journal* 107 (June 27): 21.

Ware, Willis (1970) 'Security Controls for Computer Systems: Report of the Defense Science Board Task Force on Computer Security', in (Washington, D.C.: Office of the Director of Defense Research and Engineering, Washington, DC).

Wasserman, Anthony I (1977) 'Letter from the Chairman', *SIGSOFT Softw. Eng. Notes* 2/1 (January): 1.

Wasserman, Anthony I (1978) 'Letter from the Chairman', *SIGSOFT Softw. Eng. Notes* 3/1 (January): 1.

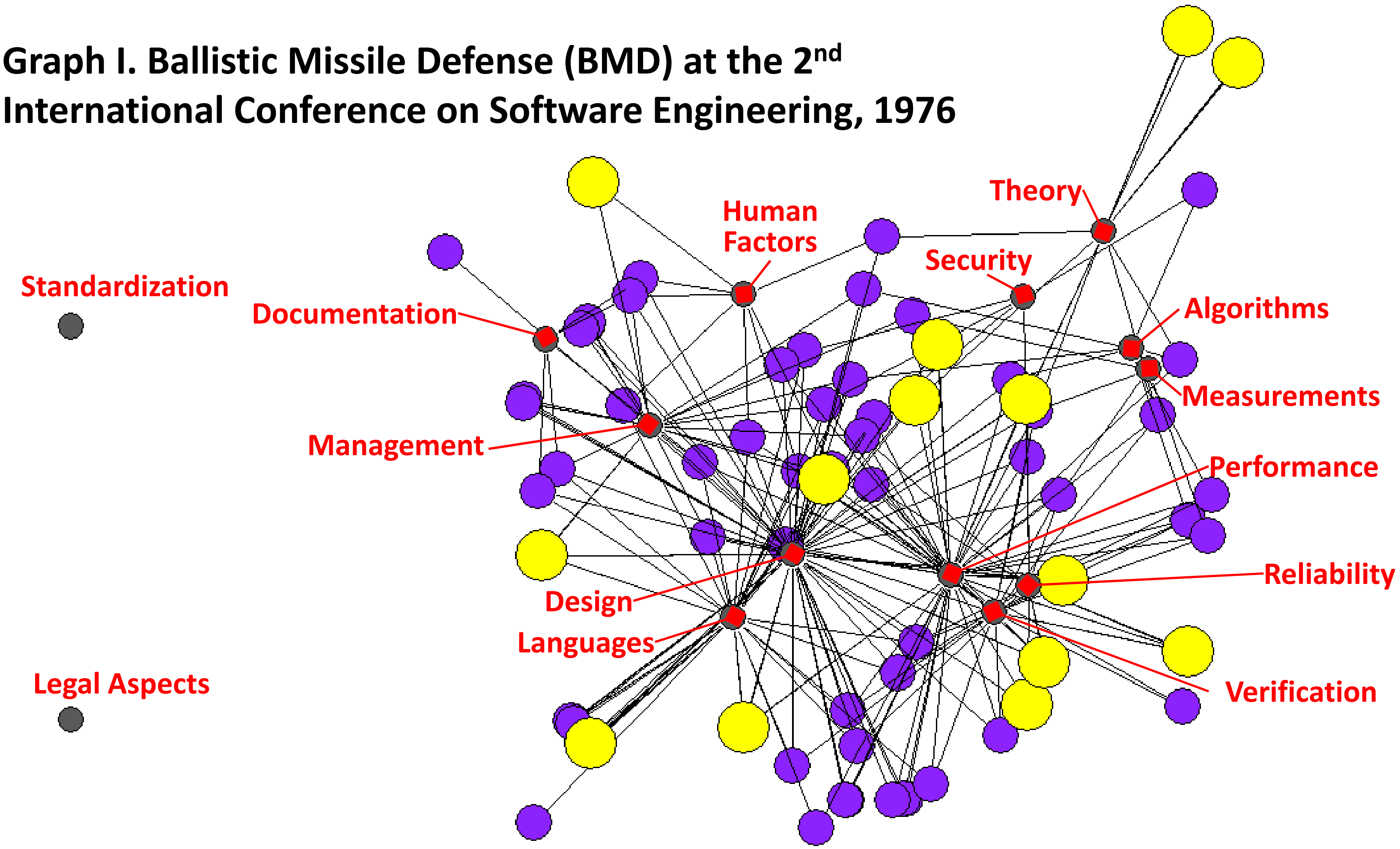
Wasserman, S. and Faust, K. (1994) *Social network analysis: Methods and applications*: Cambridge Univ Pr).

Weiss, George (1971) 'Restraining the Data Monster: The Next Step in C3', *Armed Forces Journal* 108 (July 5): 26.

Whitaker, William A (1996) 'ADA - The Project: The DoD High Order Language Working Group'.

Williams, R. D. (1975) 'Managing the development of reliable software', in, *Proceedings of the international conference on Reliable software* (Los Angeles, California: ACM).

Graph I. Ballistic Missile Defense (BMD) at the 2nd International Conference on Software Engineering, 1976



This two-mode networks show affiliations between authors and "general terms" assigned to their papers. The general terms are represented as small dark grey nodes, and labeled in red. Presenters working on ballistic missile defense are shown in yellow; all others are shown in purple.

The network layout uses a two-dimensional Fruchterman Reingold algorithm; it treats nodes like points that repel one another, but are pulled together by springs (the edges). The arrangement shown minimizes energy. Layering was used to bring the BMD partition (yellow nodes) forward.

Table I. Keyword analysis of paper abstracts at ICSE (1976).

The numbers shown here correspond to the percentage of abstracts mentioning the terms shown, one or more times within the abstract (multiple counts of a keyword within the same abstract are not counted more).

Keywords	Tools	Requirements	Specifications
90 Papers not explicitly related to BMD (titles not listed)	16% (14/90)	14% (13/90)	34% (31/90)
12 Papers Explicitly Related to BMD A laboratory for the development and evaluation of BMD software quality enhancement techniques A methodology for decomposing system requirements into data processing requirements A requirements engineering methodology for real-time processing requirements Adaptive testing An analysis of the resources used in the SAFEGUARD system software development An extendable approach to computer-aided software requirements engineering Process Design engineering a Methodology for Real-time Software development Process design system an integrated set of software development tools Research towards a technology to support the specification of data processing system performance requirements Software requirements: Are they really a problem? Specifications a key to effective software development The Software Development System	42% (5/12)	67% (8/12)	33% (4/12)

Graphs II.A-D. Continuity and Change at ICSE

These two-mode networks show how individuals were affiliated with “general terms” by reviewers of their papers at ICSE. The general terms are represented as small dark grey nodes. Individuals who presented work only once between 1976 and 1989 are represented as larger, light grey nodes. Other individuals are represented by the year they first presented in these venues, as follows:

1968-69 NATO Conferences

1976 ICSE

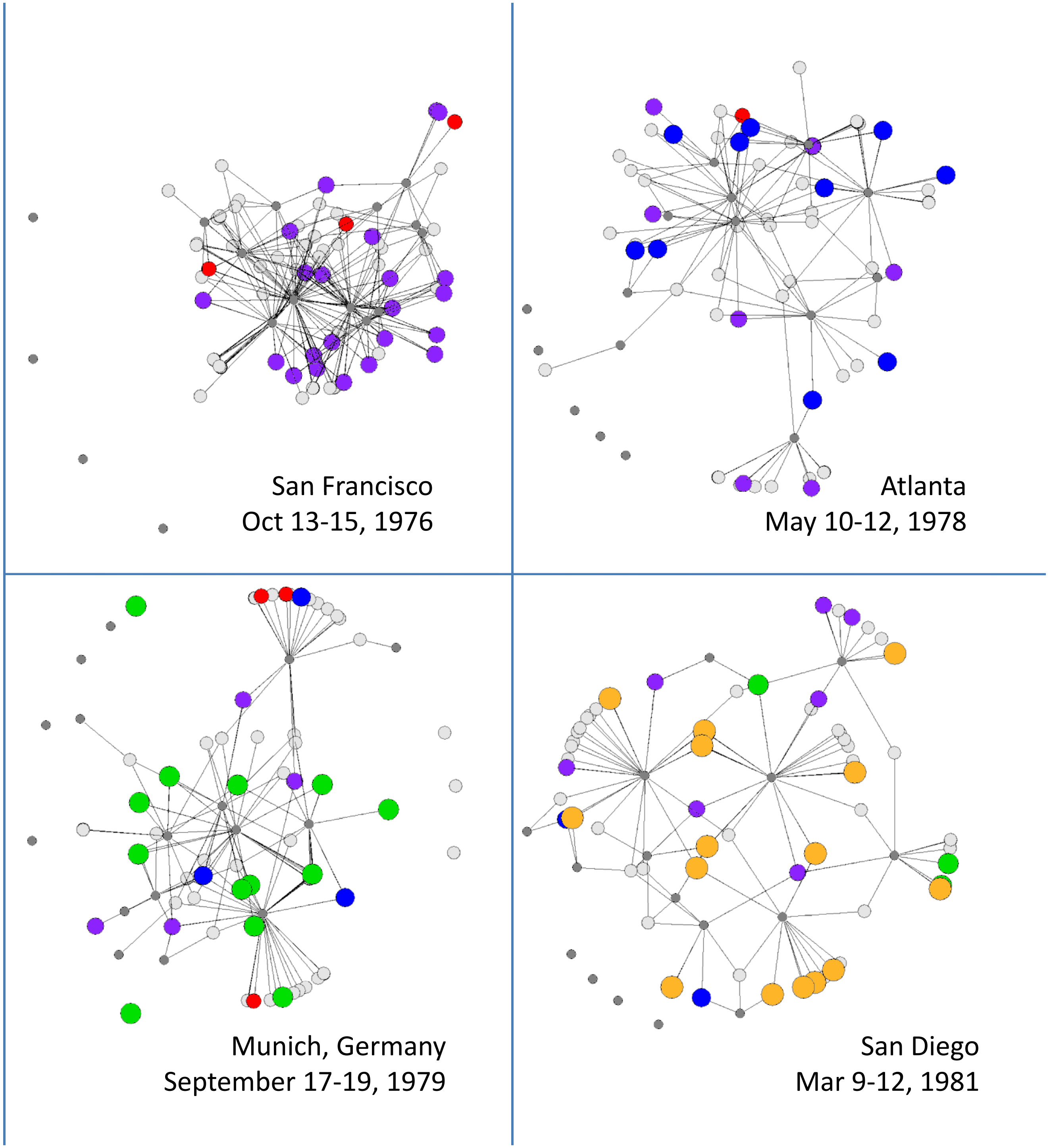
1978 ICSE

1979 ICSE

1981 ICSE

See the list of names in Table II for further detail.

The network layout uses a two-dimensional Fruchterman Reingold algorithm; it treats nodes like points that repel one another, but are pulled together by springs (the edges). The arrangement shown minimizes energy.



San Francisco
Oct 13-15, 1976

Atlanta
May 10-12, 1978

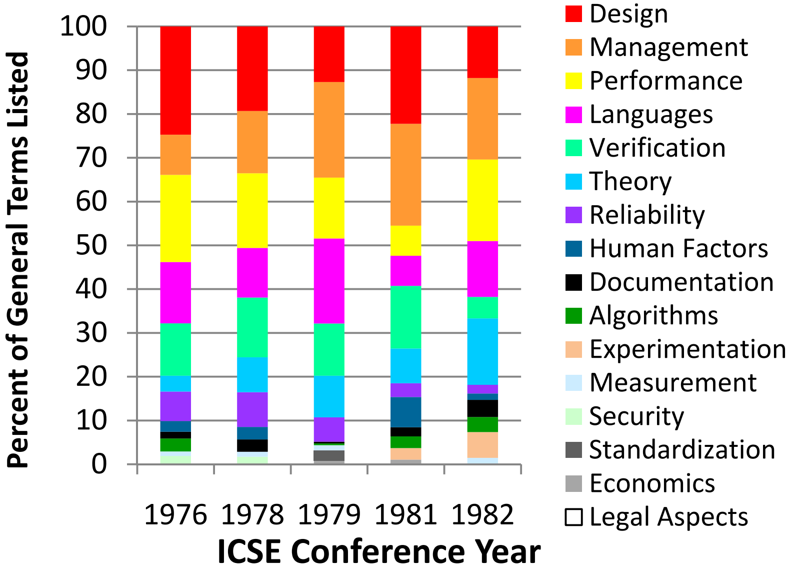
Munich, Germany
September 17-19, 1979

San Diego
Mar 9-12, 1981

Table II. Repeat Presenters in NATO-sponsored and ICSE Conferences Partitioned by first year of presentation

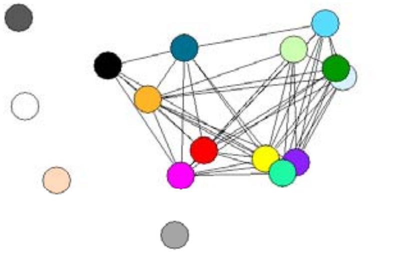
Partition 1. 1968-9	Partition 2. 1976	Partition 3. 1978	Partition 4. 1979	Partition 5. 1981
Alan J. Perlis (1968, 69, 85) Bernard A. Galler (1968, 69, 89) Brian Randell (1968, 69, 79, 84) C. A. R. Hoare (1969, 1978) C. T. Clingen (1969, 1978) David Gries (1968, 76, 89) Douglas T. Ross (1968, 69, 76, 89) Edsger W. Dijkstra (1968, 69, 79) Friedrich L. Bauer (1968, 69, 76, 82) Jerome A. Feldman (1969, 76) N. Wirth (1969, 81) P. Lucas (1969, 82) Rudolf Bayer (1969, 1979)	Anita K. Jones (1976, 79) Anthony I. Wasserman (1976, 78, 81, 88) Barry W. Boehm (1976, 79, 82, 84, 87, 88, 89) C. V. Ramamoorthy (1976, 79) Charles R. Vick (1976, 79) David J. Panzl (1976, 78) David Lorge Parnas (1976, 78, 81, 84, 85) David R. Barstow (1976, 82, 85, 87, 88) David S. Wile (1976, 81) Ellis Horowitz (1976, 85) Isao Miyamoto (1976, 78, 88) J. P. Benson (1976, 81) J.-C. Rault (1976, 79) James C. Browne (1976, 78) John W. Brackett (1976, 79) Leon J. Osterweil (1976, 81, 82, 87) M. M. Lehman (1976, 87) Marvin V. Zelkowitz (1976, 78, 81) Mary Shaw (1976, 89) P. C. Belford (1976, 79) Peter Freeman (1976, 78, 81, 89) Peter Wegner (1976, 78) Raymond T. Yeh (1976, 81) Richard B. Kieburtz (1976, 78, 85) Richard J. Waldinger (1976, 78) Robert Balzer (1976, 81, 85, 89) Susan L. Gerhart (1976, 84) Zohar Manna (1976, 78)	Allan M. Stavely (1978, 85) Gerald Estrin (1978, 79, 81) Guy T. Almes (1978, 84) Jack C. Wileden (1978, 85) John D. Musa (1978, 84) John H. Sayler (1978, 88) Laszlo A. Belady (1978, 85) Michael Jackson (1978, 1989) Nachum Dershowitz (1978, 1981) Pierre Azéma (1978, 84) R. R. Willis (1978, 79) Victor R. Basili (1978, 81, 85, 87, 88) William E. Howden (1978, 81, 89) William E. Riddle (1978, 79, 85, 87)	Bill Curtis (1979, 81, 84, 87, 89) D. M. Lasker (1979, 81) D. Ross Jeffery (1979, 88) Daniel M. Berry (1979, 81) Franco Sirovich (1979, 82) Glenn H. Holloway (1979, 81) Harald Wertz (1979, 82) Judy A. Townley (1979, 81) Karsten Schwan (1979, 87) Ken Sakamura (1979, 85) M. J. Lawrence (1979, 82) Maria Heloisa (Lolo) Penedo (1979, 81, 85) Roy H. Campbell (1979, 88) Sylvia B. Sheppard (1979, 81, 84) Thomas E. Cheatham Jr. (1979, 81) Walter F. Tichy (1979, 82, 89) Wladyslaw M. Turski (1979, 84, 85)	A. G. Duncan (1981, 84) David M. Weiss (1981, 84, 85) Debra J. Richardson (1981, 85) Elizabeth Kruesi (1981, 84) Hassan Gomaa (1981, 89) Ira R. Forman (1981, 82, 84, 88) J. S. Hutchison (1981, 84) Krithi Ramamritham (1981, 84) Lee J. White (1981, 85) Lori A. Clarke (1981, 85, 89) Mark Weiser (1981, 85) Motoei Azuma (1981, 85) Pamela Zave (1981, 87) Peter H. Feiler (1981, 87) Scott N. Woodfield (1981, 87, 88) Simeon C. Ntafos (1981, 84) Steven J. Zeil (1981, 84, 85, 88) T. S. E. Maibaum (1981, 84) Y. Futamura (1981, 84)

Chart I: General terms on papers at ICSE 1976-82.

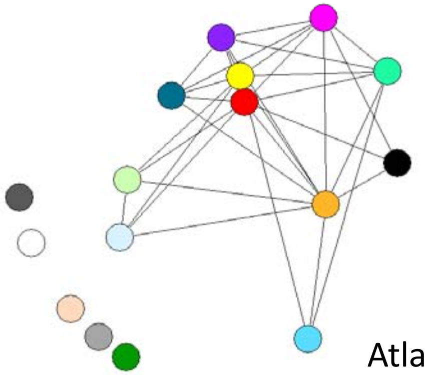


Graphs III.A-E. General term networks, ICSE 1976-82.

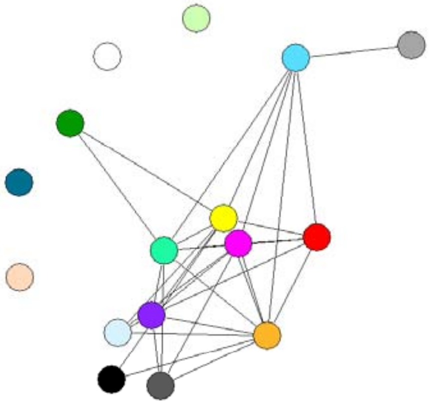
These graphs show how "general terms" were connected by individuals' publications at ICSE conferences. They were produced by transforming the two-mode data on author-term affiliation (see Graph II.A-D) to one-mode data. Edges between two different general terms represent an individual who listed both terms on one or more papers at ICSE (loops not included). General terms are color coded in Chart I, which shows the prevalence of general terms at each meeting. The Fruchterman Reingold algorithm was used to graph points; strongly tied terms are located close together in space.



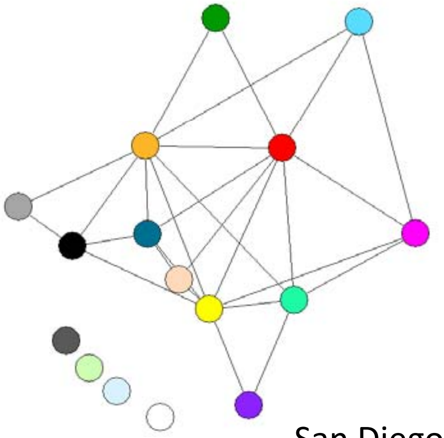
San Francisco
Oct 13-15, 1976



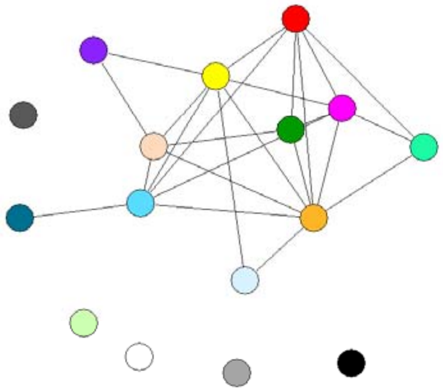
Atlanta
May 10-12, 1978



Munich, Germany
September 17-19, 1979

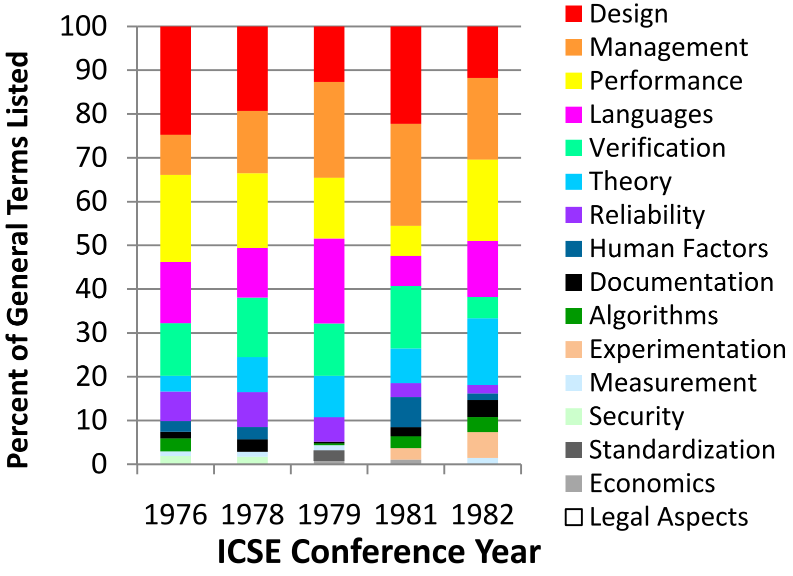


San Diego
Mar 9-12, 1981



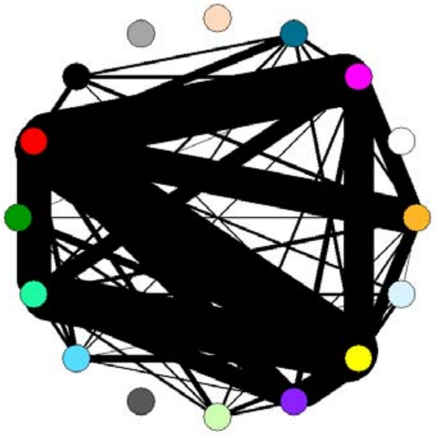
Tokyo, Japan
September 13-16, 1982

Chart I: General terms on papers at ICSE 1976-82.

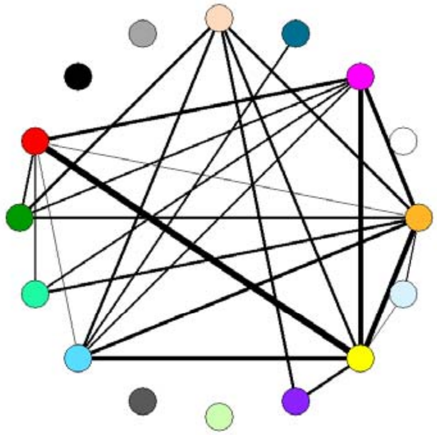


Graphs IV.A-E. General term networks, ICSE 1976-82.

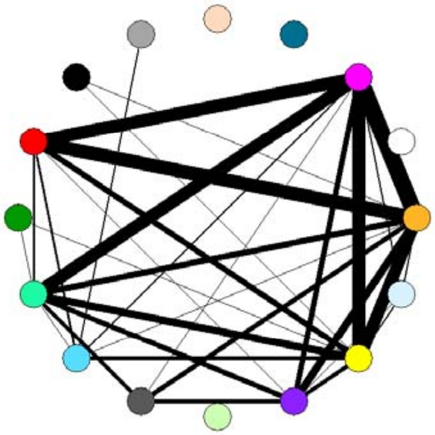
These graphs show how “general terms” were connected by individuals’ publications at ICSE conferences. This is the same data presented in Graphs III A-E, but in this case the points were placed around a circle, and the weight of the edges (number of authors) is shown. Color code is shown in Chart I.



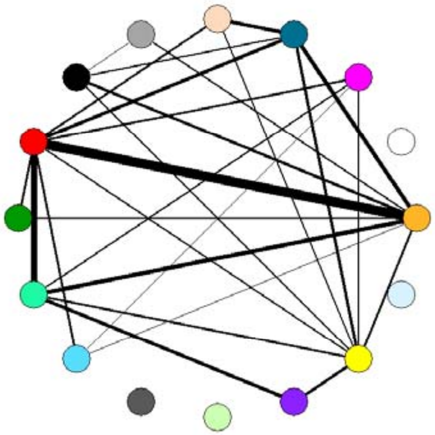
San Francisco
Oct 13-15, 1976



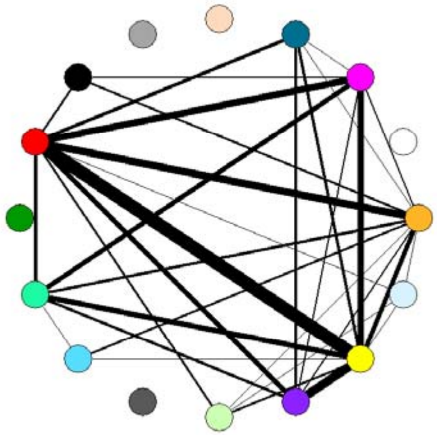
Atlanta
May 10-12, 1978



Munich, Germany
September 17-19, 1979



San Diego
Mar 9-12, 1981



Tokyo, Japan
September 13-16, 1982

Table III. Comparison of Titles of Papers at the Symposia on the Principles of Programming Languages (POPL) and International Conferences on Software Engineering, 1976-85.

Note: these data were gathered from the DBLP server. I have discovered a few omissions from the database since compiling these data, but they are not significant enough to undermine the general findings.

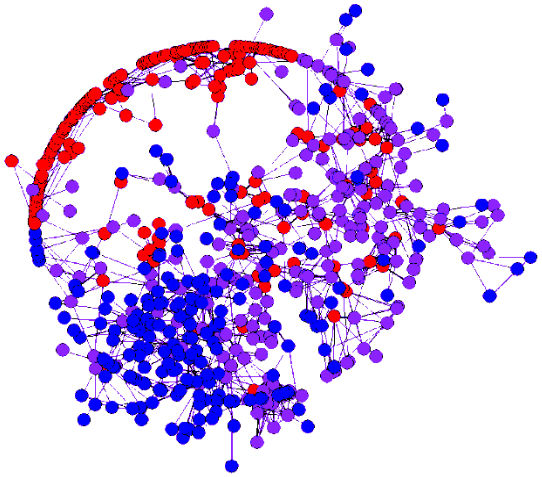
Keywords	**POPL (297 Titles Total) Keyword count**	**ICSE (409 Titles Total) Keyword count**
Reliability or reliable	0	16 (3.9%)
Manage or management	1 (0.3%)	15 (3.7%)
Product, productivity, or production	0	9 (2.2%)
Prove, proof, or proving	10 (3.4%)	6 (1.5%)
Cost	0	2 (0.5%)

Graphs V.A-C. **Co-authorship networks surrounding ICSE Conferences (see appendix). FR algorithm.**

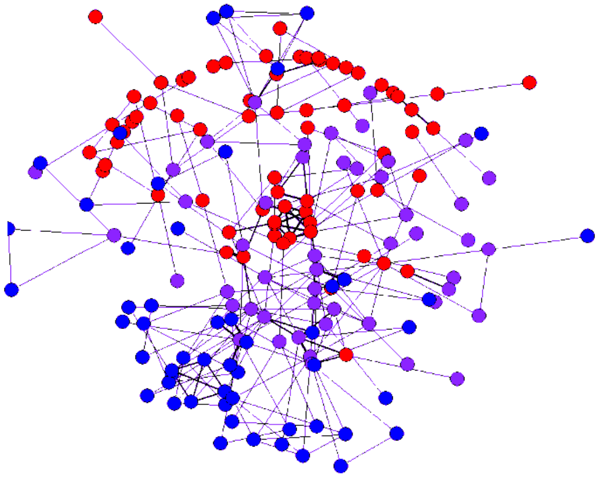
Presenters at ICSE in year shown

Co-authors of presenters outside of ICSE, in subsequent two years

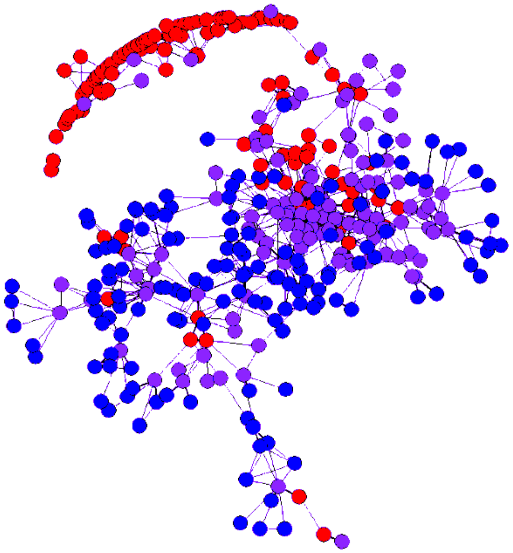
Co-authors of co-authors outside of ICSE, in subsequent two years



San Francisco, Oct 13-15, 1976



Atlanta, May 10-12, 1978



San Diego, Mar 9-12, 1981

Table IV. Cohesion in networks shown in Graphs IV.A-C. Cohesion here is measured as the average degree of nodes (the degree of a node is defined as the number of connections to other nodes). Variance in degree tends to be about as large as the degree itself. Not terribly useful.

ICSE year	Co-authorships thru:	Nodes	Components	All degree centrality	Average degree	Standard Deviation
1976	1978	592	82	0.03025	3.18	3.07
1978	1980	171	35	0.03867	2.50	1.85
1981	1983	372	39	0.09677	4.29	5.57

Table V. Most Prominent Individuals at International Conferences on Software Engineering, 1976-89. Not listed are thirteen individuals who presented work three times in this period, or 80 individuals who presented papers twice. This list is not the only measure of prominence; some of these individuals were more prominent in terms of awards and so on, but I have not systematically gathered all this data yet.

Number of Presentations between 1976-89	**Names**
7	Barry W. Boehm
5	Bill Curtis David Lorge Parnas David R. Barstow Victor R. Basili
4	Anthony I. Wasserman Gruia-Catalin Roman Ira R. Forman Koji Torii Kokichi Futatsugi Leon J. Osterweil Peter Freeman Robert Balzer Steven J. Zeil William E. Riddle