"Crisis, What Crisis?"
Reconsidering the Software Crisis of the 1960s and the Origins of Software Engineering

Thomas Haigh
thaigh@computer.org
www.tomandmaria.com/tom
School of Information Studies, University of Wisconsin--Milwaukee

DRAFT Version for discussion at the
2nd Inventing Europe/Tensions Of Europe Conference, Sofia, June17-20 2010.
A revised version is projected for inclusion in the book produced by the SOFT-EU project.

Introduction

In 1979 the British government presided over a country plagued by strikes, shortages, and spiraling inflation. Prime Minister Jim Callaghan returned to a gloomy English winter looking tanned and relaxed from an economic summit in the West Indies. Asked how he planned to deal with this crisis he denied the existence of "mounting chaos" and complained that the reporter was "taking a rather parochial view at the moment." This reply was turned by the viciously partisan and unashamedly sensationalistic Sun newspaper into the classic banner headline "Crisis, what crisis?" The phrase entered the lexicon of political reporting, and is widely believed to have helped sink his Labor party into almost two decades in opposition. Callaghan was perhaps hoping to emulate a more successful rhetorical de-escalation accomplished by Harold MacMillan in 1958, who faced the coordinated resignation of his government's three most senior financial officials. His dismissal of this apparently fatal challenge to his authority as "a little local difficulty" helped to cement his reputation for unflappability and strengthen his hold on power.

We see from these two contrasting stories that the existence or non-existence of a crisis in a particular area is very much a matter of perspective and perception. A crisis is constructed to serve the interests of particular groups, and may with hindsight appear more or less real depending on our knowledge of what happened next.

My concern here is with a different kind of crisis, an event known as "the software crisis." This event looms very large in the secondary historical literature on the history of software. Indeed it is probably the most widely and extensively discussed event in that history. According to the consensus view of history, the existence of this crisis was generally recognized at the NATO Conference on Software Engineering, held in Garmisch, Germany in 1968. The conference is said to have featured a diverse range of industrial and academic software experts, who agreed that the increasing complexity of programming work associated with a new and more powerful generation of computers had overwhelmed the technical and managerial ability of software groups. Software was late, over budget, lacked features, worked inefficiently, and was unreliable. Something to be called "software engineering" was proposed as the solution to the crisis. In a less widely discussed coda to the main story, a second conference held in Rome in 1969 was supposed to achieve more of a consensus on the actual concerns and techniques of this new discipline, but in fact ended in acrimony and revealed a lack of agreement on matters of

substance. Today software engineering is fairly popular academic field of study, with conferences, journals, and degree programs. However historians have noted with some frequency that basic debates over its identity were never really resolved and that the rhetoric of a crisis in software development has likewise endured for many decades.

Nothing in the broad outline of this established narrative is altogether false. Yet the increasingly entrenched position of the software crisis and the 1968 NATO Conference in the historical literature has gradually led to the distortion of their actual nature, historical significance, and context. At the same time, surprisingly little attention has been paid to the actual background, experiences and intellectual interests of the conference attendees or to the spread of the "software crisis" concept after the conference itself.

I begin with a review of the software crisis concept and 1968 NATO Conference in the secondary historical literature, from their first appearance in 1988 to the present day. Over time the implied scope of the software crisis has grown, as has the implied importance of software engineering as a new identity for programming practice. In the rest of the paper I go back to the original sources to try to reconstruct the actual significance of the meeting and its associated crisis, and to sketch some neglected aspects of the broader history of software and programming in order to better contextualize them.

Looking first at the origins of the "software crisis" I note that this specific phrase appears only in editorial material within the two conference proceedings volumes, not in any of the quoted dialog between participants. I thus identify the editors of the volumes as key agents in its original promulgation, but find that computer scientist Edsger Dijkstra was responsible for its more widespread adoption in the 1970s during a quixotic campaign to evict almost all practicing programmers from their jobs and replace them with mathematicians. Next I argue that change in time in the meaning of the term "software" has led to widespread misinterpretation of the scope of the crisis, which was initially understood to afflict only operating systems and programming languages. This leads to an analysis of the backgrounds and affiliations of the participants, from which I conclude that almost all were oriented toward research rather than development, and to systems software rather than applications. Among the groups not represented at the conference were data processing managers (responsible for administrative computing program development within computer using organizations), business school experts on computer use, the managers of large industrial software development projects, specialists in data base management systems, and representatives of software product companies. From the perspectives of these other groups, particularly data processing, neither the NATO Conference nor software engineering nor the "software crisis" looms very large. Instead I document a range of computer related crises and chronic complains from the 1950s onward, most of which are constructed as failure to meet the goals of the broader organization rather than being seen narrowly as failures of software.

Next I sketch the development of software engineering in the 1970s from the viewpoint of data processing, arguing that its immediate impact was limited and that the original agenda of Dijkstra and many of his fellow conference attendees, driven by the application of mathematics and theory to programming, was largely replaced by a range of other more congenial approaches. Finally I ponder the

fact that the software crisis and the 1968 NATO Conference on Software Engineering appear to be much more firmly entrenched in the writings of career historians than in the historical reflections of software engineers themselves, before concluding with a few possible explanations for this phenomenon and the outline of a more modest and nuanced way of thinking about their significance.

Historiography of the Software Crisis

Perhaps the first mention of the software crisis in the secondary literature on the history of computing came in Michael S. Mahoney's landmark 1988 paper "The History of Computing in the History of Technology." This was Mahoney's first published paper on computing, though by this point his interest in the topic had been growing for some years and he had already educated himself by auditing the core series of undergraduate computer science classes at Princeton. The paper dismissed existing work on the history of computing as narrow and parochial, setting out instead several broad agendas for the field based on key texts in the history of technology. Mahoney also sketched out his own sense of the field's submerged narrative and of the key questions thus far ignored by historians.

> Central to the history of software is the sense of "crisis" that emerged in the late 1960s as one large project after another ran behind schedule, over budget, and below specifications. Though pervasive throughout the industry, it posed enough of a strategic threat for the NATO Science Committee to convene an international conference in 1968 to address it. To emphasize the need for a concerted effort along new lines, the committee coined the term "software engineering", reflecting the view that the problem required the combination of science and management thought characteristic of engineering. Efforts to define that combination and to develop the corresponding methods constitute much of the history of computing during the 1970s, at least in the realm of large systems, and it is the essential background to the story of Ada in the 1980s. It also reveals apparently fundamental differences between the formal, mathematical orientation of European computer scientists and the practical, industrial focus of their American counterparts.[1]

This captures the key elements of the software crisis as it has appeared in the work of historians over the past twenty years: a crisis emerged around the time of the 1968 NATO conference, the conference was a response to a deterioration of conditions in industry (though it's ambiguous here whether that was the nascent software industry or the much larger computer industry), and the conference led to the founding of software engineering.

Two years later Mahoney published another paper, "The Roots of Software Engineering," which he described as "part of a history of the development of the computer industry from1950 to 1970 focusing on the origins of the 'software crisis.'" The paper expanded on his earlier outline of the software crisis, observing that the software engineering literature continued to complain of the crisis through the 1970s

[1] Michael S Mahoney, "The History of Computing in the History of Technology", *Annals of the History of Computing* 10, no. 2 (April 1988):113-125.

and 80s. He also probed concepts of the assembly line and interchangeable parts he believed to underlie early models of software engineering.[2]

At this point Mahoney saw the software crisis and accompanying rise of software engineering as the central focus of his own work on the history of computing. The online transcript of an interview with Berk Tague, as part of Mahoney's project on the history of UNIX, included "MSM: The major book I'm working on is the origin of the software crisis as it emerged in the '60s and how the industry got into that situation."[3] While his plans eventually changed, Mahoney remained interested in the history of software engineering and served for many years as an advisor to the SIGSOFT Impact Project, an effort to determine the importance of academic work on software methods to the practical accomplishments of the software industry.

In fact Mahoney went little further in the empirical study of software engineering. In 2004 his work on the topic concluded with a second paper, "Finding a History for Software Engineering."[4] As the title suggests, his focus had shifted to the use of historical metaphors and narratives by early proponents of software engineering. The construction of precedent within in a new discipline was Mahoney's great theme, expressed in his historiographic work and his accounts of the origins of theoretical computer science as well as his work on software engineering. He reworked and expanded his 1990 paper around different kinds of historical metaphor expressed by participants at the 1968 NATO conference and in other contemporary sources. This reframing better aligned his sources with the framing of his arguments, as in both versions he relies on the explication of lengthy quotations from a handful of eloquent and prominent researchers rather than offering any evidence on actual practices, representative experiences, or the institutional development of software engineering after the 1968 conference.

The software crisis was a major theme in two dissertations underway as Mahoney readied his initial paper for publication, those of Maria Eloina Pelaez Valdez at Edinburgh University and Stuart S. Shapiro at Carnegie Mellon University. Valdez's title was "A Gift from Pandora's Box: The Software Crisis," though the study ranged rather widely to include extended discussion of ALGOL, the IBM 360 range, integrated circuits, the origins of the packaged software industry, and users of third generation computers as well as a chapter on the NATO conference and another on the origins of software engineering.

The 1968 NATO Conference and the travails of OS/360 also feature prominently in Shapiro's dissertation, an ambitious exploration of "software as technology" encompassing programming language history, professionalization and certification projects, a lengthy examination of early software engineering methods, the institutionalization of software groups within the ACM and IEEE Computer Society, and the role of standards. Again the NATO Conference serves as a central event, which Shapiro

---

[2] Michael S. Mahoney, "The Roots of Software Engineering", *CWI Quarterly* 3, no. 4 1990):325-334.

[3] http://www.princeton.edu/~hos/mike/transcripts/tague.htm Unfortunately the transcript is not dated, but it appears to be part of a batch conducted in or just before 1989.

[4] Michael S Mahoney, "Finding a History for Software Engineering", *IEEE Annals of the History of Computing* 25, no. 1 (January-March 2004):8-19.

even tries to link to IBM's unbundling of hardware and software products ("increasing demand combined with increasing ambition to draw attention away from the program per se and toward the process by which is was made" – p.24).[5]

Another treatment of the software crisis and NATO conference is given in Andrew L. Friedman and Dominic S Cornford's 1989 book *Computer Systems Development: History, Organization and Implementation*. This book deserves to be much better known among those with an interest in the history of software. I would call it a lost minor classic of the field, though in fact it seems to be fairly widely cited outside the community of professional historians. The authors take an organizational perspective, looking at changes over time in the practices and technologies behind computer application systems. In this model the second major phase in the development of computer systems was dominated by software problems. They view the 1968 NATO Conference as a key event in public acknowledgement of these problems, taking from its report a total of six problems around which the rest of the chapter is structured. These are the increasing scale of software projects (most notably IBM's ambitious OS/360 development effort), excessive ambition, difficulties in estimating project cost and length, deficiencies in project management skills, inadequate documentation, skimping on design and testing, and a labor shortage among programmers.[6]

Martin Boogard's 1994 thesis, *Defusing the Software Crisis* was devoted primarily to proposing a technical solution to the crisis but includes a lengthy survey of the literature on the concept, including its use to sell products and methodologies.[7]

In 1996 Martin Campbell-Kelly and William Aspray published their overview of the history of computing, *Computer: A History of the Information Machine*. This did an excellent job of integrating the secondary literature and remains the most widely used introduction to the field. Of the twenty four pages in its chapter on software, five were devoted to the software crisis (almost all of those to the troubles ofOS/360) and four more to software engineering as a response to the software crisis initiated at the 1968 NATO Conference. According to Campbell-Kelly and Aspray (who identify no source for their analysis)

> Although there were some serious industrial and academic papers presented at the conference, the real importance of the meeting was to act as a kind of encounter group for senior figures in the world of software to trade war stories.

---

[5] Stuart Shapiro, "Splitting the Difference: The Historical Necessity of Synthesis in Software Engineering", *IEEE Annals of the History of Computing* 19, no. 1 1997):20-54, 100-105. The material of early software engineering methodologies and concepts has been published, as Shapiro, "Splitting the Difference: The Historical Necessity of Synthesis in Software Engineering".

[6] Andrew L Friedman and Dominic S Cornford, *Computer Systems Development: History, Organization and Implementation* (New York: Wiley, 1989), 102-107.

[7] Martin Boogaard, *Defusing the Software Crisis: Information Systems Flexibility Through Data Independence* (Amsterdam: Thesis Publishers, 1994).

> The Garmish conference began a major cultural shift in perceptions of programming. Software writing started to make the transition from being a craft for a long-haired programming priesthood to become a real engineering discipline. (p. 201)

The other major recent overview of the history of computing, by Paul Ceruzzi, gives a similar but shorter treatment. It references a crisis in software production and the appeal of engineering models of technical discipline.[8]

Discussion of software engineering and the software crisis also dominated two conferences on the history of software held in Germany around this time. The first, at Scholss Dagstuhl in 1996, was dedicated to the history of software engineering and included veterans of the 1968 conference and historians in roughly equal quantities. Participants recall it as rather fractious. The organizer's introduction to the conference report notes that "there may or may not be a software crisis, but there is definitely what might be called an identity crisis…. I underestimated the personal and disciplinary identity problem and I was expecting more historical studies already having been carried out…"[9]

The second conference was conceived, partly in reaction to these problems, as the beginning of a much broader approach to software history. Invited keynote speakers gathered in Paderborn in 2000 to explore five different aspects of software, for example "Software as Economic Artifact" or "Software as Labor Process." Yet, as Mahoney observed in the concluding essay of its proceedings book, "quite independently of each other, Tomayko, MacKenzie, and Aspray/Ensmenger chose to begin with reference to the NATO conference."[10] His own paper had a different opening but did discuss the NATO conferences and the mathematical tradition of reasoning about the behavior of programs. Thus four out of the five main approaches placed the conferences and their associated crisis close to the heart of their narrative. Mahoney invoked the fable of the blind men and the elephant (a group of blind people are trying to determine the shape of an elephant but each mistakes one of its many parts for a known object) to suggest that this intersection might signal the beginning of a more coherent shared vision of software history. He wrote that "their hands might touch where the parts under investigation overlap, and the encounter might cause them to combine their initial impressions into a coherent whole. This is the case with the papers at hand." It seems to me at least as likely that historians of software have mostly positioned themselves around the same leg of the elephant and inadvertently find themselves groping each other while leaving most of the beast neglected.

The software crisis and the NATO Conferences on Software Engineering have been major themes in the work of the authors Mahoney mentioned. James E Tomayko was an academic champion of software engineering, serving for many years as director of the software engineering degree program at Carnegie Mellon University. He also had a Ph.D. in history and a particular interest in the history of aerospace control computers, serving on the editorial board of IEEE Annals of the History of Computing for many

---

[8] Paul E Ceruzzi, *A History of Modern Computing* (Cambridge, Mass.: MIT Press, 1998), 105.
[9] http://www.dagstuhl.de/files/Reports/96/9635.pdf.
[10] Michael S Mahoney, "Probing the Elephant: How to the Parts Fit Together", in *Mapping the History of Computing: Software Issues*, ed. Ulf Hashagen, Reinhard Keil-Slawik, and Arthur L. Norberg (New York: Springer-Verlag, 2002):277-283, page 278.

years. Given his interests in history and in software engineering it is not surprising that he wrote a series of articles on the history of software engineering. Tomayko's work pays relatively little attention to the idea of a software crisis in the late 1960s although his article "History of Software Engineering Education" does credit the 1968 NATO Conference as the key event in popularizing the term software engineering and spreading a shared understanding between participants of the shared problems faced by software developers.[11]

Donald MacKenzie is a sociologist of technology and one of the leading scholars in the science studies field. His paper at the Paderborn conference summarized several threads of research for his book, published in 2001 as *Mechanizing Proof*. The 1968 NATO Conference and associated software crisis are central to its first narrative chapter.[12] They provide much of the impetus to the book's main theme, which is the application of mathematical techniques to prove to correctness of software (and eventually hardware) to formal specifications. This leads him to the related topics of automated theorem provers, computer aided proofs, and the evolution of safety critical systems. MacKenzie's treatment of the conference goes the farthest of any yet published to situate the conference in a broader history and explain the agendas underlying the positions expressed by several of its attendees. He looks particularly at the contribution of Edsger Dijkstra, tying his position at the meeting back to the work of John McCarthy, Peter Naur, Robert Floyd and C.A.R. Hoare on program proofs and forward to the later developments explored in the rest of the book.

Nathan Ensmenger has taken a different approach, tying the NATO conference to more general discussion on the management of computer personnel and broadening the software crisis to encompass problems with programming work of all kinds. His paper from the Paderborn conference, authored with William Aspray, calls the 1968 NATO Conference "perhaps the earliest and best-known attempt to rationalize the production of software development along the lines of traditional industrial manufacturing." [13] This is presented as part of "attempts by corporate managers to address the software crisis by developing new methodologies of project management and project control."[14] Their argument drew in part on work during the 1970s by Philip Kraft, a sociologist of work and acolyte of Marxist labor historian Harry Braverman. Braverman believed that capitalist employers consistently and deliberately reorganized industrial production around assembly lines so as to deskill workers and

---

[11] James E Tomayko, "Forging a Discipline: An Outline History of Software Engineering", *Annals of Software Engineering* 6 1998):3-18, page 6.

[12] Donald MacKenzie, *Mechanizing Proof* (Cambridge, MA: MIT Press, 2001), ch. 2. See also Donald MacKenzie, "A View from the Sonnenbichl: On the Historical Sociology of Software and System Dependability", in *Mapping the History of Computing: Software Issues*, ed. Ulf Hashagen, Reinhard Keil-Slawik, and Arthur L. Norberg (New York: Springer-Verlag, 2002):97-122 and Donald MacKenzie, "A Worm in the Bud? Computers, Systems, and the Safety-Case Problem", in *Systems, Experts, and Computers : The Systems Approach in Management and Engineering, World War II and After*, ed. Agatha C Hughes and Thomas P Hughes (Cambridge, MA: MIT Press, 2000):161-190.

[13] Nathan Ensmenger and William Aspray, "Software as Labor Process", in *Mapping the History of Computing: Software Issues*, ed. Ulf Hashagen, Reinhard Keil-Slawik, and Arthur L. Norberg (New York: Springer-Verlag, 2002):139-165, page 153.

[14] Ibid, page 152.

deprive them of control over their own work.[15] Kraft applied a similar analysis to trends in data processing work in the 1970s, suggesting that the then-novel approaches of structured programming and "chief programmer teams" were intended to deskill and fragment programming work.[16] Kraft himself did not mention the software crisis or the 1968 NATO conference, through his focus on structured programming does provide an obvious tie as the technique was associated with Edsger Dijkstra and C.A.R. "Tony" Hoare, both of whom were at the conference.

This part of their Paderborn chapter was based on material from Ensmenger's dissertation, *From Black Art to Industrial Discipline: The Software Crisis and the Management of Programmers*. [17] As its title suggests, it uses the software crisis as a frame in which to explore the purported transformation of programming over the 1950s and 60s from "a tightly knit family of 'computer people' into a diverse and highly fragmented collection of programmer/coders, systems analysts, and information technology specialists." [18] Ensmenger credits the 1968 NATO Conference with setting "an agenda that influenced many of the technological, managerial, and professional developments in commercial computing for the next several decades." [19] He avoids distinguishing between different kinds of programming work, or treating programming as an activity practiced differently in different social spaces (scientific laboratory, administrative user, computer vendor, etc) although he does suggest that his focus is on "commercial organizations in the 1950s and 1960s, for these were by far the largest users of information technology in the period." [20] Ensmenger diagnoses a labor crisis in programming by the early 1960s and suggests that early work on "automatic programming" tools, professionalization efforts among computer scientists and data processing managers, and efforts to restructure programming management should

---

[15] Harry Braverman, *Labor and Monopoly Capital: The Degradation of Work in the Twentieth Century* (New York: Monthly Review Press, 1974),

[16] Philip Kraft, *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York: Springer-Verlag, 1977), Philip Kraft, "The Industrialization of Computer Programming: From Programming to 'Software Production'", in *Case Studies on the Labor Process*, ed. Andrew Zimbalist (New York: Monthly Review Press, 1979):1-17. In his comments at the Paderborn conference David Hounshell was rather critical of Kraft's theory. David A Hounshell, "Are Programmers Oppressed by Monopoly Capital, or Shall the Geeks Inherit the Earth? Commentary on Nathan Ensmenger & William Aspray", in *Mapping the History of Computing: Software Issues*, ed. Ulf Hashagen, Reinhard Keil-Slawik, and Arthur L. Norberg (New York: Springer-Verlag, 2002):167-176. My own view is that a great deal of programming work was indeed been routinized and entirely deskilled, but that this did not reduce data processing staff to the status of assembly line workers. Any programming job simple enough to reduce to a list of rote instructions had thereby been transformed into an algorithm that a computer could follow more reliably and infinitely more rapidly than any assembly line drone. From the assembler (which eliminated the routine work of specialist coders) onward, we have seen that many technologies were expected to trigger the death of the applications programmer. Each has tended, instead, to reduce the amount of drudge work performed by the human side of the human-computer partnership, and so lower the cost of programming and raise the demand for programmers. Economic forces may yet reduce the status, pay and autonomy of programming staff, and thanks to new software technologies more and more non-specialists undertake some programming as part of their jobs, but no technical developments is likely to transform programming into a specialized clerical chore.

[17] Nathan Ensmenger, "From Black Art to Industrial Discipline: The Software Crisis and the Management of Programmers" (Ph.D., University of Pennsylvania, 2001).

[18] Ibid, vii.

[19] Ibid, 2.

[20] Ibid, 5.

all be understood as attempts to solve, or at least take advantage of, this "software crisis." According to this account the 1968 NATO Conference was a crucial turning point because it made sure that "software engineering emerged as the dominant rhetorical paradigm for discussing the future of software development" and so assimilated these alternative constructions of the software crisis into a shared model that appealed to both "industry managers," "computer manufactures," "academic computer scientists" and "working programmers." Ensmenger concludes "The software engineering model seemed to offer something to everyone: standards, quality, academic respectability, status and autonomy." [21]

When compared with earlier accounts of the software crisis Ensmenger offers an exceptionally broad scope, encompassing all kinds of programming work (and sometimes other activities related to the development and operation of computer applications) and beginning his crisis in the early 1960s (or possibly the 1950s). However he retains from earlier use of the concept the centrality of the NATO Conference and software engineering to its history. Recently the concept of software crisis seems to have become even broader. Adopting the term provides the historian with an easy way to tie in virtually any subject to this relatively well developed historical literature. One team of historical researchers described the challenges posed in localizing software packages for a non-Latin alphabet as part of "the software crisis." The combination of the software crisis and the 1968 conference have become for historians what sociologist of science Michel Callon called an "obligatory passage point" – a concept so entrenched in our collective professional network that it is appealed to by many different actors (in this case historians) with many different objectives.

Who Declared a Crisis?

Returning to the original source suggests that the relationship between the 1968 conference and the idea of a "software crisis" is rather more complex than one might expect from later accounts. The phrase "software crisis" appears only once in the proceedings of the 1968 conference, and that in an editorial aside rather than a direct quotation from a participant. This states that "There was a considerable amount of debate on what some members chose to call the 'software crisis' or 'software gap.'" [22] The phrase "software gap" may have been more prevalent at the conference itself -- it appears four times in the proceedings (all of them extracts from the discussion paper of Edward E. David of Bell Labs and Alexander G. Fraser of Cambridge University) and gaps of various kinds are discussed in four other paragraphs of discussion. The word crisis, but not the exact phrase "software crisis," appears in quoted dialog between Ken Kolence (who does not like it) and Doug Ross (who does). None of the published reminiscences from participants address this issue, and Randell today writes that "My (vague) recollection is that it evolved during discussion." [23] So it seems likely that David and Fraser's declaration of a "software gap" promoted more general discussion a "crisis" which, in the editorial summary, became a "software crisis."

---

[21] Ibid, 251.

[22] Peter Naur and Brian Randell, eds., *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968* (Brussels: Science Affairs Division, NATO, 1969), 120.

[23] Email to author.

David and Fraser have been little discussed in the literature on the NATO Conferences and unlike many of the other attendees did not become well known figures in software engineering. David was an electrical engineer with a Ph.D. from MIT. He became executive director of Bell Labs before serving from late 1970 to January 1973 as President Nixon's science advisor. He continued at the highest levels of scientific management and public advocacy, including election as President of the American Association for the Advancement of Science. Thus while he was not a career specialist in software he had been immersed in the problems Bell Labs was experiencing with Multics and was already shrewd enough in the ways of scientific politics to understand the power of a crisis. Fraser had experience in the early 1960s directing compiler and operating system work for Ferranti. At the time of the conference he was a Ph.D. student and research manager at Cambridge where he continued to work on operating systems. Of this he noted in the discussion, "the designs I have been involved in… have not involved too many people… About large designs I don't know."[24] In 1969, the year after the conference, he joined David at Bell Labs where he worked on computer architecture and rose up the ladder, eventually becoming Executive Director of the labs in 1987 and retiring as AT&T Chief Scientist. He continues to work together with David in a venture known as Fraser Research.

The paper from David and Fraser is listed in the proceedings as "Position Paper" without any other title. It was not included in the conference proceedings and as far as I know was not published elsewhere, so we have only a few short extracts spliced by the editors into the discussion. Both authors contributed separate papers discussing their own experiences in producing systems software, which are also quoted from in the proceedings. David and Fraser saw the "software gap" as having opened "between ambitions and achievements" expressing itself in the unreliability of "large software," failure to deliver promised features and overruns in time and cost.[25] They identified a failure to separate "research, development, and production" into separate projects as the "root cause" of this gap but mentioned "lack of management talents," "employment of unqualified programmers," and "sheer incompetence in software design" as contributory factors.[26]

The phrase "software crisis" had not gained general currency by the follow on 1969 conference in Rome. In the proceedings of this conference it again appears only as an editorial interjection, this time in the introduction: "realization of the full magnitude of the software crisis was the main outcome of the meeting at Garmisch."[27] This editorial declaration of "the software crisis" as the main contribution of earlier meeting (above even the declaration of software engineering as a new field) points strongly toward the role of the editors themselves in promoting the phrase. Brian Randell may have primary responsibility for this. He was coeditor of both volumes, and in recent correspondence suggested of the introduction that "I think I wrote or at least drafted it…."[28] If the participants at the 1969 meeting shared this view of the earlier conference it is not reflected in their recorded comments. The word crisis

[24] Naur and Randell, eds., *Software Engineering*, 50.
[25] Ibid, 120.
[26] Ibid, 122.
[27] J N Buxton and B Randell, eds., *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969* (Brussels: NATO Scientific Affairs Division, 1970), 7.
[28] Email to author, May 5 2010.

appears just once in the body of the volume in a statement from Edsger Dijkstra that "we all tell each other and ourselves that software engineering techniques should be improved considerably, because there is a crisis."[29]

Searching the ACM Digital Library on the combination of NATO and "software engineering" yields 11 matching publications by January 1971, three years after the publication of the report. These are mostly in the literature on programming languages and systems programming. After this relatively slow initial spread a further 12 papers mention the combination in 1972 alone.

Bernie Galler played an important role in publicizing the conference and the ideas of software engineering and the software crisis. He was then President of the ACM, and in 1969 devoted one of his regular columns in its Communications to praise of the conference report and reproduction of some its jucier quotations. He wrote "whether or not one calls it a crisis, these is a serious problem in the software industry." It's interesting that he confines this crisis to the software industry, rather than software itself. Galler also seemed to have computer vendors in mind when he blamed the problem on "manufacturers who promise to produce new systems knowing the work involved has a high research content." [30]

Dijkstra's Crisis

The phrase "software crisis" seems to have gained little traction until 1972.[31] Then Dijkstra made it a central theme of his Turing Award lecture "The Humble Programmer."[32] The Turing award is the most prestigious in computer science, billed by the Association for Computing Machinery as the "Nobel Prize" for its field. During the 1970s recipients put a lot of work into their speeches, which were disseminated to the entire membership of the association. As well as his great technical contributions, Dijkstra was already well known within the emerging computer science community for a short manifesto published in 1968 and given by Niklaus Wirth, then editing the letters section of Communications of the ACM, the memorable title "GO TO Statement Considered Harmful."[33] This is frequently mentioned by historians as the beginning of a new focus on the practical importance of writing programs in an elegant and readable style. The phrase "Considered Harmful" has since worked its way into the title of dozens of computer science articles.

Dijkstra's 1972 lecture is probably the single most important turning point in pushing broader discussion from a general, widely shared sense of problems in large scale system software development into The Software Crisis, a specific malady. He uses the exact phrase four times in this short paper, complaining

---

[29] Buxton and Randell, eds., *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*, 13.

[30] Bernard Galler, "ACM President's Letter: NATO and Software Engineering", *Communications of the ACM* 12, no. 6 (June 1969):301.

[31] One of its few occurrences was http://portal.acm.org/citation.cfm?id=961053.961057&coll=ACM&dl=ACM&CFID=86366026&CFTOKEN=31144452

[32] Edsger Wybe Dijkstra, "The Humble Programmer", *Communications of the ACM* 15, no. 10 (October 1972):859-866.

[33] Edsger W Dijkstra, "Letters to the Editor: Go To Statement Considered Harmful", *Communications of the ACM* 11, no. 3 (March 1968):147-148.

that we are "up to our necks in the software crisis," asking "Is it a wonder that we found ourselves in a software crisis?" and observing that

> With respect to the recognition of the need for greater reliability of software, I expect no disagreement anymore. Only a few years ago this was different: to talk about a software crisis was blasphemy. The turning point was the Conference on Software Engineering in Garmisch, October 1968, a conference that created a sensation as there occurred the first open admission of the software crisis.

So rather than saying that a software crisis was proclaimed by attendees at the 1968 NATO Conference on Software Engineering it might be more historically revealing to say "In 1972 a software crisis was proclaimed by Edsger Dijksra to have been proclaimed in 1968 at the NATO Conference on Software Engineering." Recall that Dijkstra was also the only person quoted in the proceedings of the 1969 NATO Conference as referring to a crisis.

This attribution is particularly significant because of the singular nature of Dijkstra's world view and personality. He cast himself as "the humble programmer" but was neither conventionally humble nor a career programmer. Trained as a physicist he believed that the proper model for programming came from the "mathematical engineer," an occupation he claimed was well established in his native Netherlands but which thoroughly confused his American colleagues. Dikstra was undeniably brilliant, but his personal peculiarities and insistence on expressing honestly his low opinion of most other computing researchers marginalized him even in world of computer science, a community with a high level of tolerance of individual eccentricity. He was particularly critical of the management of industrial operating systems projects and of approaches based on the management of large programming teams. Instead he dreamed of a scientific revolution, in which programming would win recognition as a mathematical discipline and the incompetent hordes currently messing things up would be swept away and replaced by an elite corps of scientists versed in new and more formal methods of software development.

To give a sense of his fondness for polemic and disdain for the status quo it is enough to know that he believed that the following propositions were irrefutable:

> (1) good programming is probably beyond the intellectual abilities of today's 'average programmer'
>
> (2) to do, hic et nunc, the job well enough with today's army of practitioners, many of whom have been lured into a profession well beyond their intellectual abilities, is an insoluble problem
>
> (3) our only hope is that, by revealing the intellectual contents of programming, we will make the subject attractive to the type of students it deserves, so that a next generation of better qualified programmers may gradually replace the current one.[34]

[34] Edsger W Dijkstra, "EWD 611: On the Fact that the Atlantic Ocean Has Two Sides", in *Selected Writings on Computer Science: A Personal Perspective*, ed. Edsger W. Dijkstra (New York: Springer-Verlag, 1982):268-276. I

Dijkstra's outspokenness was unique, but his general sentiments were shared by many of those present. Dijkstra shared three characteristics which were, as we will see later, common among attendees at the 1968 meeting. He was a career researcher, primarily in the academic world though he spent some years conducting very similar work while on the payroll of the Burroughs corporation. He worked on compilers for high level programming languages during the 1960s. He was an important part of the Algol effort, a hugely influential attempt to produce a standard international programming language for scientific use. And he was a leader of a small team producing an academic operating system, again during the 1960s.

These characteristics help us to understand the positions taken by Dijkstra and others at the meeting such as CAR Hoare, Alan Perlis, Peter Naur, and Niklaus Wirth. They were exceptionally gifted men with strong scientific backgrounds who in the early- and mid-1960s had produced effective, elegant and reliable pieces of systems software with small teams and tiny budgets while working in research environments or for marginal computer firms. They were not corporate managers or project management specialists. So their instinct was to recoil as word spread of out-of-control industrial development efforts in which millions were being spend to produce unreliable, bloated, and incomplete operating systems and compilers. Instead their faith was that with the application of suitably rigorous and mathematical methods a small team could produce high quality systems software without the need for massive teams of programmers and elaborate bureaucratic controls.

The software crisis Dijkstra promoted at the 1969 NATO Conference on Software Engineering, in his 1972 Turing Award speech, and in other occasions over the years to come[35] was to him merely the natural result of a fundamental failure to recognize the mathematical nature of software development and the insistence of companies in hiring insufficiently intelligent people to do it. The instrumental utility of the crisis concept to Dijkstra and his colleagues was not lost on those present at the NATO Conference. According to Donald MacKenzie, Albert Endres later expressed the view that "the notion of a 'software crisis' was an exaggeration that theorists had constructed to justify their work."[36]

Thus the most important initial group claiming "software engineering" as a identity was the group advocating what would soon be known in computer science circles as "formal methods." This is the core of Mahoney's interest in the topic, connecting it with his larger research project in the history of mathematical ways of reasoning about the behavior of computer programs.[37] As Mahoney himself wrote "The call for a discipline of 'software engineering' in 1967 meant to some the reduction of programming to a field of applied mathematics."[38]

What Was Software?

---

should follow up on his reports of the Rome conference and look for a translation of the report of the 1968 meeting.

[35] According to Boogaard, *Defusing the Software Crisis: Information Systems Flexibility Through Data Independence* page 48 Dijkstra continued at least until 1982 to advance the software crisis as the consequence of hiring incompetents rather than mathematicians to do programming work.

[36] MacKenzie, *Mechanizing Proof*, 37.

[37] Michael S. Mahoney, "Computer Science: The Search for a Mathematical Theory", in *Science in the 20th Century*, ed. John Krige and Dominique Pestre (Amsterdam: Harwood Academic Publishers, 1997).

[38] {Mahoney, 1992 #4943}.

Today, and for several decades previously, we tend to use "software" synonymously with "computer program." This was not the case in the early 1960s, as I have explored in my article "Software in the 1960s as Concept, Service, and Product."[39] Software first became part of the data processing lexicon in the early 1960s, as the complement of hardware. Examining early usage reveals a variety of implicit meanings and considerable debate as the word's correct meanings. Proposed definitions invariably included operating systems and compilers, but generally did not include application programs produced within a user company. As a 1962 Honeywell advertising supplement entitled "A Few Quick Facts on Software" explained

> Software is a new and important addition to the jargon of computer users and builders. It refer to the automatic programming aids that simplify the task of telling the computer 'hardware' how to do its job. The importance of software lies in the fact that programming a computer can be an arduous, time-consuming and costly operation and the quality of automatic programming aids has become virtually as important as equipment specification in evaluating the total capability of a data processing system…
>
> Generally there are three basic categories of software: 1) Assembly Systems, 2) Compiler Systems, and 3) Operating Systems.[40]

Some definitions suggested than any kind of computer service or program purchased from an external company was software, but that the same service or product would not be considered software if developed in house. As Gerard Alberts has pointed out, this focuses on the "ware" side of software, because the word was widely used by the late 1960s to describe a new and diverse industry of computer services firms. During this era systems programs were increasingly likely to be supplied by computer manufacturers or specialist firms and used without change, whereas applications programs were still overwhelmingly likely to be written within the company using them.

By the time of the NATO Conference on Software Engineering in 1968 the word software was closely associated with systems programs but much less commonly used to describe application programs.

Mahoney himself was well aware of this. In his 1990 paper he noted the rapid growth in employment of programmers during the 1960s and suggested that "programs became 'software' in two senses. First, a body of programs took shape (assemblers, monitors, compilers, operating systems, etc.) that transformed the raw machine into a tool for producing useful applications, such as data processing. Second, programs became the objects of production by people who were not scientists, mathematicians, or electrical engineers."[41] It was the first set of programs, the programs that made programs, which became software. As Mahoney wrote in a later paper "By the mid-60s, the term [software] had taken on a more specific sense of systems software, what people use to construct and

---

[39] Thomas Haigh, "Software in the 1960s as Concept, Service, and Product", *IEEE Annals of the History of Computing* 24, no. 1 (January-March 2002):5-13.
[40] Honeywell, "A Few Quick Facts on Software", *Business Automation* 7, no. 1 (January 1962):16-17
[41] Mahoney, "The Roots of Software Engineering".

run programs."[42] A third article by Mahoney, "Software the Self Programming Machine" makes it clear that his interest is primarily in systems software because "systems software is about getting the computer to do the applications programming." He added that these "systems increasingly realized the ideal of the computer as a self-programming device."[43]

Thus, Mahoney suggested, at the 1968 NATO Conference the suggestions of participants "built on developments in systems software, extending programming languages and systems to encompass programming methodologies. Two main strains are of particular interest here: the use of programming environments to constrain the behavior of programmers and the extension of programming systems to encompass and ultimately to automate the entire software development cycle."[44]

Discussion of software quoted in the proceedings of the 1968 conference largely assumes this sense of the word. Administrative and scientific applications were dismissed in passing by J.N. Buxton, then a professor at the University of Warwick, as outside the scope of discussion: "Of course 99 percent of computers work tolerably satisfactorily; that is the obvious. There are thousands of respectable Fortran-oriented installations using many different machines and lots of good data processing applications running quite steadily; we all know that!"[45]

What specific pieces of software are mentioned in the transcript? IBM's OS/360 is mentioned about as much as every other piece of software put together. A full tabulation[46] would include

- **Operating Systems**: OS/360, CTSS (MIT), Multics, TSS/360, TSS/635 (the OS for an online military system), ECS Scope (CDC) and a range of experimental systems such as Dijkstra's THE, MTS (Michigan), JOSS, CODAS by Parnas and Darringer, and the work of Zucher and Rangall at the IBM Watson Lab.
- **Compilers:** PL/1, AED (an Algol-based MIT string oriented language), FORTRAN, COBOL, Nebula (for ICT Orion), LISP, SIMULA, SNOBOL,
- **Tools to Support Programming**: Autoflow (ADR), Com Chart, Mercury, TOOL, etc.
- **Real Time Control Applications**: "Electronic Switching Systems" (at Bell), SABRE.

Discussion was dominated by problems facing the large scale development of operating systems and (to a lesser extent) programming languages. Most notably IBM's troubled OS/360 and the ill-fated Multics conducted as a joint project between MIT, General Electric, and Bell Labs since 1964. Yet few of the attendees had first hand experience working on the massive commercial operating systems projects of the mid-1960s. The two obvious exceptions were R M Graham from MIT's Project MAC and Doug McIllroy who was then head of the Bell Labs Computing Techniques Research Department. Both were

---

[42] Michael S Mahoney, "Software as Science--Science as Software", in *Mapping the History of Computing: Software Issues*, ed. Ulf Hashagen, Reinhard Keil-Slawik, and Arthur L. Norberg (New York: Springer-Verlag, 2002):25-48, page 43.

[43] Michael S. Mahoney, "Software: The Self Programming Machine", in *From 0 to 1: An Authoritative History of Modern Computing*, ed. Atsushi Akera and Frederik Nebeker (New York: Oxford University Press, 2002):91-100.

[44] Ibid .

[45] Naur and Randell, eds., *Software Engineering* .

[46] Hmm – what is GP? SYSGEN? IPL? Check for others and make sure are properly situated.

institutionally located in research organizations, but both MIT and Bell Labs were deeply immersed in the development of Multics. Graham was responsible for many aspects of the Multics Kernel and McIllroy was, like many of those who went on to create Unix, a part of the Multics team until Bell Labs' final withdrawal from the project in 1969.

A few applications are mentioned, but their presence is negligible compared to the focus on systems software. SABRE comes up just once. SAGE, today perceived as the most influential single system in the history of computing, is not mentioned at all. (This also makes it clear that the participants had no bias toward military applications, despite the NATO sponsorship). RPG, a utility for generating customized reports out of business data files, is the only specific data manipulation or reporting package to be mentioned, and even then only in a table estimating the amount of work needed to test programs written in different languages.[47]

Who Were These Guys Anyway?

For all the discussion of the 1968 conference as a vital historical turning point there has been remarkably little attention paid to who these people were. Some are famous, others have left almost no trace in the easily accessible literature. Documenting their biographies and subsequent career trajectories would be a major contribution to penetrating the fog of hype that has enveloped the conference. I can attempt no more than a spirited stab in this direction, but hope to encourage others to undertake a more systematic effort.

One nice feature of the 1968 NATO Conference proceedings is that it includes as an appendix the full name and address of each participant. This permits an approximate tabulation of the groups represented at the meeting.

Sector	**Count of Attendees**
Universities	21
Govt. Research Centers	9
Computer Vendor Research Groups	13
Other Corporate Research (mostly Bell Labs)	4
Software & Services Firms	5
Military Observers	3
Unknown	1
Total	56

This simple exercise suggests that the academic world was better represented at the conference than has sometimes been suggested. For example James E. Tomayko's assertion that "Software engineering seemed of immediate interest to industry, and industry dominated the population at the NATO

[47] Naur and Randell, eds., *Software Engineering*, 198.

conference" does not hold up well.[48] The second conference (Rome 1969) skewed more heavily in the direction of academic computer science, including such notables as Per Brinch Hansen, Edsger Dijkstra, Robert W Floyd, Bernie Galler, Gerard Goos, J W Graham, C A R Hoare, Roger M Needham, Butler W. Lampson (then a graduate student at Berkeley), Alan J. Perlis, Christopher Strachey, W L van der Poel and Niklaus Wirth.[49] All had worked on operating system or programming language research, and between them they eventually won six Turing awards.

Attempts to extract from the conference transcripts representative opinions of industrial and academic groups seemed doomed. Donald MacKenzie wrote that the invitee list for the 1968 Nato Conference was "carefully constructed to include key figures in academia, in the computer industry, in the emerging 'software houses,' as well as a small number of important computer users." [50] Other historians have used similar phrases to suggest that the impact of the conference stemmed in part from its inclusiveness. MacKenzie's description is not incorrect, but it is easy to read into it a much more diverse group of people than the one actually assembled in Garmisch.

Certainly there were attendees from many computer companies, from many universities, and from a handful of software companies. But most of the attendees from each side were members of a single emerging community: computer scientists researching operating systems.

Valdez points out that so many of the people at Rome and Garmish had moved back and forward between academic and corporate posts. Comparing the affiliations listed for attendees a the 1968 and 1969 conferences she sees that in this short time Randell moved from IBM to Newcastle University and Ross from MIT to Softech. Dijkstra was listed as a faculty member at the Technological University of Eindhoven when he attended the 1969 conference, but soon left for a spell at Burroughs before returning to academic life in the 1980s. She writes

> "The differences were not quite as clear as that: they could not be when it is often difficult to distinguish the industrial people from the academics in a world in which computer scientists frequently move from IBM or the other large corporations to the universities and back, and in which so many from both backgrounds have an interest in a software house or a consultancy."[51]

Almost all the participants were closer to research than to development and followed career paths leading further away from large scale software development. Randell's career path captures a common progression among the attendees. In the late 1950s and early 1960s he worked for English Electric to develop systems software, most notably a much acclaimed Algol compiler. In the mid-1960s he transitioned to a corporate research center at IBM, where he worked on experimental computer

---

[48] Tomayko even wondered how history might have been different it "more than a sprinkling of academics had attended the Garmisch conference?" Tomayko, "Forging a Discipline: An Outline History of Software Engineering", page quote from pages 6 and 7.

[49] Buxton and Randell, eds., *Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969*, 122-125.

[50] MacKenzie, "A View from the Sonnenbichl", page 98.

[51] Maria Eloina Pelaez Valdez, "A Gift From Pandora's Box: The Software Crisis" (Ph.D., University of Edinburgh, 1988), 185.

architectures and methodologies for operating system design. In 1971 he left business altogether, to become a professor of computer science at the University of Newcastle. Dijkstra followed a similar path. He was the coauthor of the first Algol 60 compiler, done early work on interrupt handling, and made a major contribution to operating system design with his experimental THE system of the mid-1960s. His career turned progressively toward mathematics and theory, and though his work on concurrent programming and process synchronization was hugely influential he never took part in the development of a large scale commercial system.

Thus the actually picture is rather less diverse than one might expect merely from looking at the names of the employers listed in appendix A.1 of the conference proceedings. For example IBM was well represented at the conference. Brian Randell and Ascher Opler worked at its Yorktown Heights research center, Albert Endres at its Böblingen programming lab in Germany, C.P. Enlart at its European Program Library in France, Francois Genuys at IBM France (no unit listed, but Genuys was a mathematician active in IFIP and so appears to have been engaged in research of some kind), R. C. Hastings at IBM Rochester, and John Nash at IBM's UK labs in Hursley. Thus a closer look at the roster indicates that only Hastings might have been coming from a mainstream IBM product group rather than one of its flagship research centers. None of the participants made, to the best of my knowledge, major contributions to the OS/360 project, or to any of IBM's other major software products such as IMS, CICS, or Fortran. Nash had led development of the first workable PL/1 compiler.

The attendees from software companies had similar backgrounds. The list of attendees includes four men whose affiliations are given as small companies rather than universities, government agencies, corporate research labs, or computer manufacturers. Of all the attendees one might expect these men to be closest to business issues, production software, and applications. Closer examination verifies this, but also makes it clear that their experience was primarily with systems software and that business concerns were very marginal at the conference.

Two of them did indeed represent small software companies. Both companies originated as developers of systems software, often under contract to hardware companies, rather than creators of application systems. Alexander d'Agapeyeff was a founder of Computer Analysts and Programmers. Its first job was creation of a version of Algol for the English Electric KDF6 and it later produced a real time operating system for the Royal Navy. The firm's other founder eventually became a professor of computer science. However the firm also hired rapidly and diversified into administrative work, and d'Agapeyeff had formerly worked for an accounting firm. So he had relevant knowledge both of running a services business and of data processing applications.[52] Ken Kolence represented the firm Boole and Babbage, of which he had become a cofounder the previous year. Prior to that he had worked for computer manufacturer CDC with responsibility for researching improved methods for its internal production of software. His plan for the new company was to sell specialized systems software to monitor

[52] Harry Baecker, "Biographies: From Cambridge to Calgary by Computer", *IEEE Annals of the History of Computing* 20, no. 1 (January-March 1998):55-66.

performance on OS/360 systems.[53] At the conference Kolence made friends with d'Agapeyeff and signed up CAP as a distributor for his products. Kolence's own firm sold systems tools, but he had early experience at North American Aviation managing a group of technical application programmers and his research at CDC used a project planning application rather than a piece of systems software as its test case. He had also been involved in the SHARE user group.

The contributions of both men showed sensitivity to the issues of application development. D'Agapeyeff's main contribution to the proceedings was his "inverted pyramid" graphic and accompanying explanation, in which he argued the need for "middleware" programs between existing kinds of systems software and applications programs. This was the origin of the term, which is now used widely. But although d'Agapeyeff and Kolence both had some experience with application software their backgrounds were primarily with system software and tools, and in fact d'Agapeyeff's pet project at CAP in the 1970s was the development of operating system and compiler products for microcomputers.[54]

Kolence also stands out in the conference proceedings for his challenge to the idea of crisis and acknowledgement of the importance of applications programs.

> The basic problem is that certain classes of systems are placing demands on us which are beyond our capabilities and our theories and methods of design and production at this time. There are many areas where there is no such thing as a crisis – sort routines, payroll applications, for example. It is large systems that are encountering great difficulties.[55]

In a 2001 oral history interview Kolence discussed his feelings of marginalization at the conference.

> I must admit, after a few days I had become rather disillusioned about the prospect of this conference providing any helpful ideas about how to begin forming a real engineering capability for software.
>
> Part of my reason for this disillusionment was related to the backgrounds of the attendees. About a fourth or so of us were from the business world, and the rest were basically from academia. It was clear that Dr Perlis had invited the business people simply to appear that he was interested in what we might have to say. I had the distinct impression that he had no interest in talking with any of us who disagreed with his main principle for software engineering. This was apparently: that he only way to build software was to only have small (6 or 8) people teams, and not burden them with management reporting responsibilities. He liked to say things like "Don't throw Chinese Armies at development problems", an allusion to both IBM's large and well managed (for its time) System 360 software effort, and the Chinese army's incursion into North Korea in the 1950's. As a matter of fact, he said it so often that it was clear he didn't believe in the main point of my little paper, which was and is that software product

[53] Kenneth W Kolence, *OH 348: Oral History Interview with Jeffrey R. Yost, October 3* (2001); available from http://special.lib.umn.edu/cbi/oh/pdf.phtml?id=316. See also (Aspray interview of Coleman). He did actually have some interest in apps, and had managed their development at NAA.
[54] Kathy Lawrence, "The Alternative Operating System", *Data Processing* 27, no. 6 1985):27-28.
[55] Naur and Randell, eds., *Software Engineering*, 17.

> development requires (but is not itself sufficient) the design of rational and correctly integrated management and technical processes.[56]

Kolence adds that "I was not invited or even told about the conference in Rome the next year. So I guess I didn't make any impression on Dr Perlis."[57]

The other two attendees giving their affiliations as small companies turn out not to represent specialist software firms. James Babcock represented a small company, Allen-Babcock Computing. This was a minor player in the market for online computer services, rather than a software company in the modern sense, though in those days the time sharing business required cutting edge expertise in operating systems. The firm was known for producing an interactive version of PL/1 and an OS/360 timesharing system called Conversational Programming System under contract to IBM.[58] The last of the four, was Robert S. Barton. While he gave his affiliation as "consultant in system design" his day job from 1965 to 1973 was as a professor of electrical engineering at the University of Utah. Barton was an expert in systems programming, having managed a development of an Algol-like compiler for Burroughs and then become an independent consultant for a few years. He won fame as architect of the Burroughs B-5000 computer, the first optimized to run programs created in high level languages. He eventually returned to Burroughs in a research role.[59] Neither seems to have any personal connection to application programming.

Who Wasn't There?

The inclusion of "a small number of important computer users" in MacKenzie's list of groups represented at the conference stands out particularly. First it is not literally true that only a "small number" of the attendees were computer users. As software specialists almost all of them used computers in their work. If we assume that what MacKenzie meant was not "computer users" but "representatives of organizations not involved in the production of software" then the flips from understatement to overstatement.

From the mid-1950s to the 1970s the use of computers for administrative purposes was almost always called "data processing." In fact no representatives of corporate data processing departments attended the meeting. This is surprising, at least for readers liable to equate "software crisis" with "programming crisis", because these firms were the main creators of computer programs. Remember that at this time every computer installation included programmers as well as computer operators, analysts, keypunch operators and managers. Almost all application programs used by large companies were written or extensively revised in house. Thus the vast majority of programs were written inside user companies

---

[56] Kolence, *OH 348: Oral History Interview with Jeffrey R. Yost, October 3* . Kolence appears to think of Perlis as the conference organizer. He was the keynote speaker, but the proceedings themselves show F. L. Bauer as the chairman, H.J Holms and L. Bolliet as the co-chairment, and Perlis merely as leader of the design group. Randell today recalls that Bauer and Helms had the primary responsibility for selecting invitees.
[57] Ibid.
[58] http://www.bitsavers.org/pdf/allen-babcock/cps/CPS_Progress_Report_may66.pdf
[59] http://www.computer.org/portal/web/awards/barton

such as banks, insurance firms, manufacturing firms, scientific research institutes, and civilian government bureaucracies.

The absence was not just of rank and file corporate applications programmers. They who languished some way down the status hierarchy within data processing and did not usually attend international conferences. Also absent were corporate managers of data processing departments, some of whom were responsible for many hundreds of staff and budgets of many millions of dollars. The data processing world had its own minor celebrities, consulting gurus, columnists and other recognized experts. None of these men and women were in attendance. Systems analysis had been well established in American business even before the introduction of administrative computers, but no experts on administrative procedures and their redesign were invited. The study of "management information systems" was already emerging as a significant area of focus within business schools and consulting firms, but no representatives of those communities were present.

Who would have been invited if the conference was devoted to programming as a whole rather being narrowly focused on industrial and academic researchers in systems software? In 1968 the list would have included men such as Dan McCracken (the leading author of programming textbooks and future ACM president), Charles W. Bachman (creator of the first data base management system and future Turing Award winner), Bob Patrick (a prominent author and consultant with deep knowledge of administrative software applications), Robert V. Head (a veteran of SAGE and SABRE and respected author on real time business systems), James Martin (prolific writer of books on business computing), Richard L. Nolan (one of the leading academic writers on IT management, soon to be a professor at Harvard Business School), Dick Canning (publisher and writer of EDP Analyzer, the smartest and best informed industry newsletter), Frances V. Wagner (a former SHARE chairman and founder of leading software firm Informatics), Walter M Carlson (ACM vice president, soon to be president, former McKinsey consultant, and an outspoken advocate for business computing) and the eccentric but ubiquitous Herb Grosch (an industry legend and future ACM president). It might also have included at least one woman, Grace Hopper, who was well on her way to iconic status.

Even the focus at Garmisch on systems software was selective and skewed away from administrative computing. 1968 was a crucial year in the emergence of the data base management system as a new kind of systems software. Charles W. Bachman produced IDS, later recognized as the first data base management system during the early 1960s. By 1968 IBM's competing IMS software was being released as a product and the Data Base Task Group of the CODASYL standards organization was finishing up its landmark report on the topic.[60] During the 1970s DBMS systems became the most important product category of the new packaged software industry.[61] Yet data base management systems are not mentioned in the conference transcript. Not a single member of the CODASYL Data Base Task Group or was present at either of the NATO Conferences on Software Engineering.

---

[60] Thomas Haigh, "How Data Got its Base: Generalized Information Storage Software in the 1950s and 60s", *IEEE Annals of the History of Computing* 31, no. 4 (Oct-Dec 2009):6-25.

[61] Thomas J Bergin and Thomas Haigh, "The Commercialization of Data Base Management Software 1969-85", *IEEE Annals of the History of Computing* 31, no. 4 (Oct-Dec 2009):26-41.

Business data handling needs did surface at one point in the discussion, with an allusion to Informatics' Mark IV data file management system, soon to become the first piece of software to reach a million dollars in sales. Nobody there seemed to know much about this area.

> Perlis: I have a question for McIlroy [who had just presented the idea of a software components factory using parameters and generator programs to produce required subroutines]. I did not hear you mention what to me is the most obvious parametrizations, namely to build generalized business data file handling systems. I understand that Informatics has one out which everybody says is OK, but — . This seems to be a typical attitude to parameterized systems.
>
> McIlroy: My reason for leaving that out is that this is an area that I don't know about.

Other Crises

These other computing communities had their own crises and problems, sometimes parallel to the software crisis popularized by Dijkstra but largely separate. Data processing had been plagued since its origins with both rampant hype and, for almost as long, complaints of failure and crisis. General Electric's appliance plant in Louisville, Kentucky created the first administrative computer application in America when it used a Univac I to process payroll starting in 1954. This was promoted as a showcase for automation, but it also quickly won fame in computing circles as an example of a bungled programming project: late, over budget, and missing crucial features. Cost savings from the replacement of payroll clerks were dwarfed by the new expenses of computerization.

By 1958 several thousand computers had been installed in American businesses, mostly for administrative work, and reports of problems were mounting. Breathless hype and promises of a computer revolution were no longer enough for those with products or ideas to sell. Rhetoric shifts in a new direction. From the late 1950s to the present day a high proportion of articles on the business use of computers can be summarized thus: "We face a crisis. The inherent power of computer technology is enormous, but it has been squandered by [insert scapegoat here]. A new [technology/methodology/product/field/professional identity] shows great promise…."

These persistent complaints were usually couched as evidence managerial, organizational, and cultural failures rather than failures of software or programming. The specifics changed over time, but the rhetorical coupling of failure, acute danger, and potential salvation endured for decade after decade. For a long time it was used to justify freeing data processing groups from the authority of accountants and elevating their status within the organization chart. It was widely used to argue that computers should be used to assist managerial decision making rather than confined to clerical automation.

Often these pleas include explicit declaration of a "crisis." Of course the word was everywhere in those days, thanks to the Suez Crisis (1956), the Cuban Missile Crisis (1962) and a host of other cold war and diplomatic crises. 1968 was a year of political crises around the developed world. Urban crises, environment crises, energy crises and various hostage crises permeated public discourse in the 1970s. The Harvard University library system alone has 3,566 books with the title keyword "crisis" published between 1960 and 1980. So it's not surprising that the word appeared with regularity in the data

processing literature. The rhetoric of crisis was even used with respect to punched card machines, when a 1961 article proclaimed a "Crisis in Machine Accounting" because punched card supervisors had failed to take their managerial duties seriously.[62] In 1963 the American Management Association warned that "management is facing an information crisis" because of the "large gap between the possibilities of EDP as a profit-oriented management tool and its actual use in business today."[63] In 1967 a computing specialist at NASA warned of an "ADP Management Crisis", and advocates for the creation the new field of information science warned through the late 1950s and into the 1960s of an "information crisis" that could only be overcome with new technologies.

1968 appears as a crisis year in the business computing press and literature on managerial computing for reasons largely unconnected to the "software crisis" associated with systems software development and the NATO Conference on Software Engineering. After its original formulation in 1959 the idea of a "totally integrated management information system" spread rapidly through computer consulting firms, the data processing trade press, computer manufacturers and the business school literature. This involved computerizing all of a firm's administrative processes and linking this base of data to report generation, modeling and simulation systems to support managerial decision making. Through most of the 1960s it was by far the most widely discussed concept for the application of computers to business. Yet the idea was far ahead of the hardware and software platforms of the day. Ambitious open ended projects ended years later in outright failure or systems accomplishing only a fraction of what was promised. 1968 was the year in which *Fortune*, the *Harvard Business Review* and elite consulting firm McKinsey and Company all abandoned their support of the concept and began to publicize its failure.[64] None of this was mentioned at the 1968 NATO Conference.

The most elaborate treatment of crises in data processing management came from Richard L Nolan, a professor in the Harvard Business School. His article "Managing the Four Stages of EDP Growth" introduced the so-called "stage model" of data processing development. While an early version of the idea was published in Communications of the ACM in 1973, it was a Harvard Business Review article in 1974 that really launched the model Nolan was to promote and refine for decades to come. [65] The late 1960s and early 1970s were a troubled time for computer departments, as the cost and complexity associated with advanced third generation computers failed to bring promised improvements in efficiency and breakthroughs in applications. Some critics called on data processing managers to put their own house in order and focus on improving the quality of the service they provided, making the computer operation more like an efficient production line turning input data on punch cards into printed reports and other outputs. Others felt that the problem was failure to apply computers to important

[62] Arnold E. Keller, "Crisis In Machine Accounting", *Management and Business Automation* 5, no. 6 (June 1961):30-31.

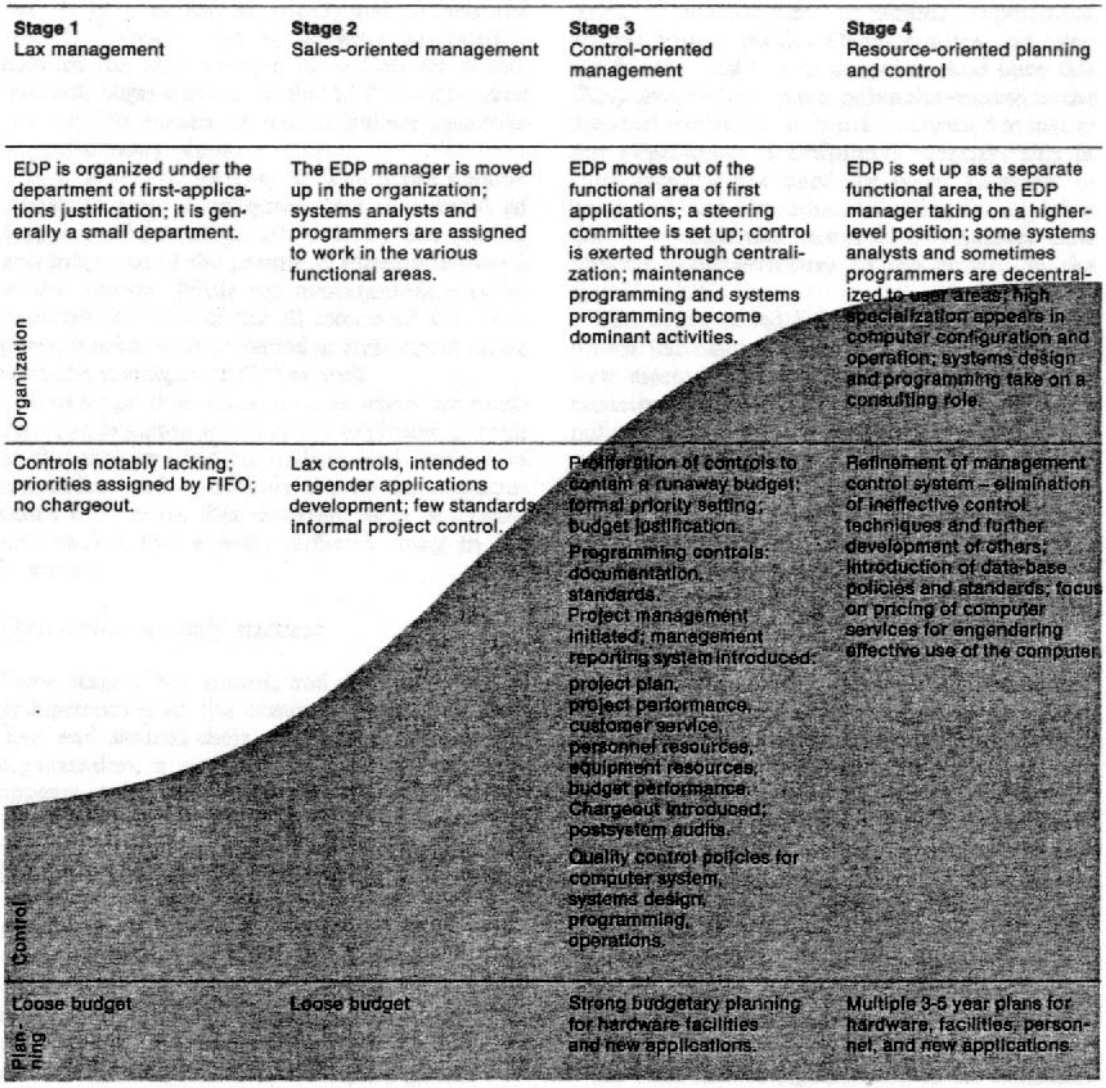
[63] Gabriel N. Stillian, "EDP and Profit Making", in *Control Through Information: A Report on Management Information Systems (AMA Management Bulletin 24)*, ed. Alex W. Rathe (New York: 1963):42-44

[64] Thomas Haigh, "Inventing Information Systems: The Systems Men and the Computer, 1950-1968", *Business History Review* 75, no. 1 (Spring 2001):15-61.

[65] The model received its first public outing in the single-author paper Richard L. Nolan, "Managing the Computer Resource: A Stage by Stage Hypothesis", *Communications of the ACM* 16, no. 7 (July 1973):339-405. The HBR paper was co-authored, and unlike the original paper does not describe the research on which it is based – however there is no evidence that the three firm sample of the original paper was broadened.

problems in managerial decision making, rather than mere clerical automation. Nolan cleverly combined these two critiques, and suggested that they reflected separate stages through which each computer department must pass on its way to the nirvana of the data management function.

According to Nolan, each firm was destined to make its own progression along a fixed trajectory through a total of four stages. Along the way, it would pass through "turning points." During the first of these stages, Initiation, it automated simple accounting and clerical applications. As the range of applications broadened to encompass budgeting, inventory, forecasting and other operational tasks it entered the second stage, that of Expansion (or as Nolan called it elsewhere, Contagion).



Gibson and Nolan's idea of "Management techniques currently applied in each of the four stages." The shaded area represents the overall EDP budget.[66]

[66] Cyrus F. Gibson and Richard L. Nolan, "Managing the Four Stages of EDP Growth", *Harvard Business Review* 52, no. 1 1974):76-88, page 79.

As the expansion stage continued, the company faced a crisis in its computer operations triggered by loss of control and decentralization. Systems analysts would become over confident, projects would cost far more than anticipated and the number of specialists would multiply, triggering self-reinforcing growth. Nolan warned that in this stage "fascination with the computer and its applications as a symbol of progressive management techniques or as a status symbol for a department or individual. This fascination breeds an enthusiasm not moderated by judgment."[67]

These problems triggered the third stage, Formalization. The data processing manager would be reined in or fired, as management imposed stringent controls on the runaway department. Control would be recentralized, and the data processing staff would chafe under elaborate reporting systems, formal planning of new applications, chargeout systems and quality control measures. Nolan cautioned that harsh measures might be needed: "[t]rying to introduce needed formalization of controls with the same personnel and the same organizational structure more often than not encourages conflict and reinforcement of resistance rather than a resolution of the crisis; by refusing to fire or enforce layoffs, senior management may simply prolong the crisis..."[68]

Further rewards lay in the fourth and final stage, "Maturity." Nolan admitted that he had never observed a company arrive at this stage. Like Karl Marx, Nolan employed the logic of historical materialism (and in fact credited Das Kapital as an intellectual justification for this method in his first presentation of the stage model, though not in the Harvard Business Review article). Each firm passed inevitably through a succession of stages, each one creating the conditions for its own replacement. In both cases, the final, utopian, stage was yet to arrive but was expected to appear shortly.

The software crisis mentioned briefly at Garmisch in 1968 and promoted heavily by Dijkstra and other theorists in the early 1970s was conspicuous by its absence in managerially-oriented data processing publications such as Business Automation and in the general managerial press. Business Automation first acknowledged the push to establish software engineering in an article entitled "The Mongolian Hordes vs. Superprogrammer."[69] This article, by a consultant working for a specialist software company, introduced its readers to the main ideas of the NATO Conference and summarized some of the most important early literature on the subject.

Datamation was the liveliest computing publication of the 1960s, reaching an eclectic audience centered on the computer industry itself but including coverage of applications and technologies as well as industry news and gossip. The magazine had paid attention to the management of programming earlier, and published articles by several software engineering pioneers. Even here, however, the phrase "software crisis" did not appear with enormous frequency – and when it did, it was often in another context. For example a 1970 article by real time software expert and software industry analyst Robert V. Head listed no less than a dozen separate crises affecting the young industry, none of them the crisis of large scale system software development associated with "The Software Crisis." Among them are a

[67] Ibid, page 81.
[68] Ibid, page 84.
[69] Jerry L. Odgin, "The Mongolian Hordes versus Superprogrammer", Infosystems 19, no. 12 (December 1972):20-23.

"crisis of program protection" due to an outdated intellectual property regime and a "crisis of marketing expertism" or rather the lack thereof among software firms.[70]

Thus we see that the software crisis beloved of historians had a very marginal place in the awareness of data processing experts in the late 1960s and early 1970s, even though a different and widely discussed set of purported crises gripped their attention. Neither does the 1968 conference appear from the viewpoint of data processing as a particularly early or influential as a venue for the admission of problems in the production of software (specifically operating systems, programming languages, and other system tools). Dijkstra's suggestion that it was blasphemy to speak of software troubles prior to 1968 is clearly not true with respect to the data processing community. Indeed similar complaints can be heard from the moment the word first gained currency. In 1961, for example, Herb Grosch wrote that hardware was generally successful but software problematic, and reminded his readers of "the SOS fiasco in Los Angeles and elsewhere, the Wall Street 501 delays; the Norfolk 220 debacle; gross failures to meet schedule on the part of almost every programming system package whether written in-house or contracted out."[71] Early COBOL compilers were late and ran very slowly, causing such grumbling that in 1963 one early user reported "One of the newest parlor games seems to consist of sitting around in a circle (*inner* of course) and debunking COBOL. I have heard everything blamed on this language but the recent recession in Monaco."[72]

In 1966 Carl Hammer, then director of computer sciences at Univac, noted in a public speech that "Software is considered by many the great enigma in the field of electronic computers. It appears as a source of frustration to programmers and it is frequently a matter of great concern to management."[73] The next year Robert V. Head wrote that "To say that the systems software being delivered with third-generation gear falls short of being an unqualified success would perhaps be the understatement of the decade." He bemoaned the computer "manufacturers' difficulties in providing flexible and efficient operating systems, file management systems, and other software aids" to their customers.[74]

Software Engineering in the 1970s

As the 1970s wore, matters became still more confused as the "software engineering" tag was freely borrowed and applied to a broad range of different ideas and products. Chief among the ideas marketed as software engineering were structured programming, new analysis and charting methodologies, and new software project management techniques. This left many of the original enthusiasts thoroughly disillusioned, among them Dijkstra himself. Brian Randell, one of the editors of the original NATO reports, later recalled that despite the collapse of the original efforts, "the software engineering

[70] Robert V. Head, "Twelve Crises -- Comments on the Future of the Software Industry", *Datamation* 16, no. 3 (March 1970):124-126.
[71] H.R.J. Grosch, "Software in Sickness and Health", *Datamation* 7, no. 7 (July 1961):32-33, page 32.
[72] Stanley M. Naftaly, "Correcting Obfustications By Ordained Linguists", *Datamation* 9, no. 5 (May 1963):24-28.
[73] Carl Hammer, A Hard Look at Software: Keynote Address, American Management Briefing Session 6379-04, 1966, contained in CBI, Minneapolis
[74] Robert V. Head, "Old Myths and New Realities", *Datamation* 13, no. 9 (September 1967):26-29.

bandwagon began to roll as many people started to use the term to describe their work, to my mind often with very little justification."[75]

Somehow by the 1980s the main focus of software engineering had shifted toward these standardized methodologies for analyzing business needs, designing a computer application, and managing its construction and deployment. The history of this transition is yet to be written, but I suspect that it is the 1970s and early 1980s rather than the 1968 NATO Conference that will eventually prove to have been the crucial period in shaping the spectrum of concerns we now think of as software engineering. Over the next few papers I offer a highly incomplete and somewhat speculative sketch of the era.

To simplify matters a little for the present discussion, the occupants of the software engineering bandwagon of the 1970s may can be divided into two groups. The first, closer to computer science and to the Garmisch vision of software engineering, was concerned with providing new and more rigorous methods of programming. These techniques included the mathematical verification of program logic, the use of formal specification languages, the creation of more transparent programming languages, and attempts to create re-usable blocks of program code called "software components." Ideas of this kind were eventually packed in object-oriented programming languages such as C++ and Java for easy consumption by the unwashed masses. [76]

The original, aggressively mathematical concept of structured programming[77] advanced by Dijkstra and his colleagues softened into a general concern with good programming style and the avoidance of GOTO statements. One 1975 article conceded that "since many of the original articles about structured programming either came out of scientific or software programming environments, the question of how well structured programming would work in commercial data processing has remained unanswered in many people's minds." Many promoters of structured programming combined these ideas with new principles for laying out, indenting and modularizing code. This was tied to a new interest in what was amorphously termed the "style" of a good program, which itself implied that programs should be read and critiqued by humans as well as executed by machines. That, in turn, reflected a modest increase in academic attention given to the training of programmers and the improvement of programmer productivity.[78]

---

[75] Brian Randell, "The 1968/69 NATO Software Engineering Reports" (paper presented at the Dagstuhl-Seminar 9635: "History of Software Engineering", Schloss Dagstuhl, August 26 - 30 1996).

[76] The component idea, much discussed by historians, was presented at the NATO Conference as M D McIlroy, "Mass Produced Software Components", in *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*, ed. Peter Naur and Brian Randell (Brussels: Science Affairs Division, NATO, 1969):138-156.

[77] O J Dahl, E W Dijkstra, and C A R Hoare, *Structured Programming* (New York: Academic Press, 1972).

[78] The quotation is from Kenneth T. Orr, "Structured Programming: Not a Fad", *Infosystems* 22, no. 11 (November 1975):36-38. For one of the first discussions of structured programming in the data processing literature, see Edward F Miller and George Lindamood, "Structured Programming: Top-Down Approach", *Datamation* 19, no. 12 (December 1973):55-57. For a discussion of the relevance of these ideas to programmer training see Edward L Schulman and Gerald M Weinberg, "Training Programmers for Diverse Goals", *Data Base* 5, no. 2-3-4 (Winter 1973):16-26 and Fred Gruenberger, 16th Annual One-Day Computing Symposium (RAND Symposium 16), 1974, contained in RAND Symposia Collection (CBI 78), Charles Babbage Institute, University of Minnesota, Minneapolis.

The second school shifted away from the details of coding styles and toward the best ways to manage a programming project and structure a programming team. Sensational claims were made by representatives of IBM for a new approach they called the, "chief programmer team." This was supposed to combine "recent technical advances in programming," with, "a fundamental change in managerial framework which includes restructuring the work of programming into specialized jobs, designing relationships among specialists, developing new tools to permit these specialists to interface effectively with a developing, visible project; and providing for training and career development of personnel within these specialties." The concept, ostensibly modeled on the surgeon, placed a virtuoso programmer in the role of system architect and chief coder. He would be assisted by a number of backup programmers, librarians, secretaries, etc. to handle the less creative parts of the job and – metaphorically speaking – hand over scalpels and scrub up while keeping out of his way.[79] This system was seldom if ever used in practice, but its discussion established an important thread within software engineering research focused on the micro level management of programming practices, small software development workgroups, and tools to support workflow between different kinds of software specialist.

In his Psychology of Computer Programming, a cult classic, Gerald Weinberg had offered another widely discussed team structure for software projects with his more egalitarian concept of the "egoless" programmer. He thought that programming projects worked best when lead in a low-key fashion by a skilled programmer. For Weinberg, programming was a complex and important human activity, a craft best overseen by those skilled in its mysteries. His work signaled the beginning of a new focus on programming practices and their improvement. In what one might think of as a "programmer pride" movement, efforts began to justify programming as a profession (or at least a skilled craft) in its own right rather than as a (low-status) part of data processing, business, scientific work, or computer science. This reinforced the general push for "structured programming" (at least in the very vague sense this concept was used in the data processing press) and the new interest in program code as an engineered artifact or aesthetic object.[80]

While neither egoless programming nor the chief programmer team ever achieved widespread usage, the publicity given to them did trigger a new interest in the integration of design and analysis work with specialized approaches to software project management. Consciously or unconsciously, advocates of

---

The importance of human-readable programs as a centerpiece to the socialization and management of programmers was given an influential early statement in Gerald M. Weinberg, *The Psychology of Computer Programming* (New York: Van Nostrand Reinhold Company, 1971). Similar issues had been raised by the RAND Symposium discussants back in 1963, when M.D. McIlroy (later an influential participant in the NATO Conference) complained that nobody read programs, and that they were "not written with flair, style, elegance, and what have you.... I'd like to see one programmer say of another's program 'that's good; he has a nice style.' We never seem to talk about style in programs, and we ought to." Fred Gruenberger, RAND Symposium 6, 1963, contained in RAND Symposia Collection (CBI 78), Charles Babbage Institute, University of Minnesota, Minneapolis, 43.

[79] The quotes come from F Terry Baker and Harlan D Mills, "Chief Programmer Teams", *Datamation* 19, no. 12 (December 1973):58-61, in which the chief programmer team concept was presented to the world of data processing – though the ideas had circulated previously in an internal IBM report back in January 1970. The concept was also featured prominently in the classic Frederick P Brooks, Jr, *The Mythical Man Month: Essays on Software Engineering* (Reading, MA: Addison-Wesley, 1975).

[80] Weinberg, *The Psychology of Computer Programming* .

this school of software engineering sought to emulate the 1950s projects such as Atlas and Polaris, from which systems engineering and engineering project management techniques had sprung. Techniques for flowcharting and analysis were merged with management tools like PERT charts to create pre-packaged methodologies intended to encapsulate the latest practices. Building on the new enthusiasm for structured programming, by late 1970s, people like Edward Yourden, Michael Jackson and Tom De Marco had built thriving careers as speakers and writers promoting their own trademarked methodologies for "structured" programming, design and analysis.[81]

To adopt these "structured" methods implied a shift toward a project-oriented, engineering approach to management and, more importantly, implied that effective data processing management would demand a body of highly specialized knowledge and a firm understanding of the latest ideas in programming and analysis. Interest in these techniques, and in engineering as a model for data processing careers, represented a new vision of professionalism. In some senses, software engineering extended the techniques of the systems analyst upward into project management and downward into code writing. The systems men of the 1950s had also paid attention to flowcharting and analysis, having invented many of the charting techniques modified by the new generation of software engineering gurus.[82] However they had focused their attention the design of business processes, believing that the actual details of programming could be delegated to lower level team members. Software engineering, on the other hand, paid a great deal of attention to the interplay between the technical architecture of the system being developed, the specific programming techniques used, the analysis techniques adopted, and the managerial methods by which the project was coordinated.

Other emerging areas of interest within software engineering research as it developed during the 1970s and early 1980s included metrics for software project progress and for the quality of the software produced and the processes producing it, automated methods for testing software, cost estimation techniques to gauge the time and budget necessary for a particular project, and tools to manage source code repositories and other software project data.

What did all this mean for the organization and status of programming work? We saw earlier that Ensmenger, following Kraft, has implied that the programmers of the 1950s and 1960s were skilled, independent artisans and that software engineering techniques represented a push by capitalist managers to degrade their work and remove their autonomy.[83] It is true that many software engineering advocates promised to bring industrial discipline to the unruly crowds of programmers they believed were getting in the way of an efficient development process. More fundamentally, however, to view software engineering solely as a managerial device for the control of programmers is to miss at least half

[81] Michael A Jackson, *Principles of Program Design* (New York: Academic Press, 1975), Tom DeMarco, *Structured Analysis and System Specification* (Englewood Cliffs, N.J.: Prentice-Hall, 1979) and Edward Yourden, *Techniques of Program Structure and Design* (Engelwood Cliffs, NJ: Prentice-Hall, 1975).
[82] Haigh, "Inventing Information Systems".
[83] In fact, creative mystique of programming appears very rarely in the data processing literature, and then always as a foil – invoked as a symbol of the bad old days or as something about to be banished. No data processing authorities of the 1950s or early 1960s were ready to concede that administrative programming was either mysterious or artistic, and few believed it unmanageable.

the point. Software engineering was also supposed to be a technocratic device for the imposition of engineering rationality on corporate management. As well as pushing techocratic control downward onto programmers, it pushed a concern for the technical qualities of good program design upward into middle management. Software projects would be headed by qualified software engineers with practical experience and advanced degrees – not by fuzzy generalist managers.

As such, software engineering might be expected to hold considerable appeal to programmers seeking a managerial path in which their experience and world-view would prove an asset rather than a liability. The traditional path of advancement within data processing departments led the ambitious from programming work into systems analysis, and from there into the management of application development projects, management of the data processing department, and eventually into general business management and away from computing entirely. The more successful one was the less closely one worked with computer technology. In contrast many software engineering methodologies involved an implicit blending of technical and managerial responsibilities. An engineering project, like other kinds of professional work, could only be overseen by someone with expert knowledge in the field able to command the respect and evaluate the work of the specialists involved. New approaches created senior positions for system architects and various kinds of software specialist. For this reason, interest in software engineering was strongest in organizations for which the production of software was a primary goal: software houses, consulting firms and computer manufacturers.

Even though many of the software engineering methodologies of the late 1970s and 1980s were explicitly designed to improve the management of ordinary application development projects, they were slow to win acceptance in the non-specialist firms in which most system development work took place. While some administrative programming supervisors may have picked up on fads, such as the elimination of the GOTO statement, or the need for particular code indentation techniques, they usually lacked the power or conceptual tools to reorganize more fundamental aspects of the development process. For corporations their expenditure on data processing remained an administrative overhead, and the pressure placed on data processing managers was to provide better service, control costs and show more awareness of business issues rather than to turn themselves into engineers. In turn, university trained computer scientists and aspiring software engineers have been less likely to seek out jobs as corporate application programmers, which were more likely to be filled by the self-educated, community college graduates, and the commercially trained.

Consulting firms and government agencies were particularly likely to adopt elaborate, standardized methodologies to provide a fixed series of stages, checklists, feedback mechanisms, and tools for each software development project. On the other hand, widespread attempts to produce Computer Aided Software Engineering (CASE) tools to automate such methodologies, and go directly from requirements and flowcharts to code in an integrated environment, proved disappointing. Computerized tools are now widely used to diagram and document program designs, and modern code editing environments

(Integrated Development Environments, or IDEs) include powerful features to automatically check and format code.[84]

Hopes for software engineering as the basis for a new profession, however, have been disappointed. Despite periodic pushes, software engineering has not won recognition as a "true" engineering discipline, relatively few programmers would identify themselves as software engineers, and fewer yet would have anything approaching a rigorous academic training in its principles. Indeed, attempts to define a suitable core of software engineering knowledge have proved highly fractious.

The appeal of software engineering has remained strongest to academics and those with advanced degrees working on complex systems software projects. In the academic world, the term software engineering has been applied to research centers, conferences, journals and degree programs concerned with techniques for the effective production of computer software. In the academic world, software engineering represented a coordinated attempt to bridge the gap between the theoretical concerns of computer science and the managerial orthodoxies of corporate practice. Computer science, as many software engineering specialists saw it, had pursued a few narrow and mathematically interesting avenues which had little bearing on the practical problems of huge development projects.[85] Yet traditional management techniques had equally little to offer these projects, because generalist managers insisted that there was nothing special about computer projects. Though "software engineer" has appeared as a job title or self-description beyond the university, its use has largely been confined to systems software specialists, and to computer experts seeking credibility within technical organizations dominated by engineering culture, for example aerospace firms. As an academic field software engineering has provided a useful umbrella under which academics have been able to research the processes of programming and design, and produce new tools and methodologies intended to support them.

Where did the Crisis Go?

Historians and sociologists of science often distinguish between the "actors' categories" used by our historical subjects to understand the world and guide their own actions and our own "analysts' categories" chosen to facilitate our own investigation of the topic under consideration.[86] As we gain historical distance from the events under consideration the two viewpoints are likely to diverge. We can never disregard actors' categories because they are necessary to understand the motivations and choices of our protagonists. But for the same reason accepting uncritically the ideas used by our subjects to understand their world can lock us into ways of seeing the stories in question that favor the interests of one group over another, erase the evidence of historical change, or render the actions of their opponents incomprehensible. It is just as dangerous to superimpose on our historical actors

[84] On the history of CASE, see Fred Hapgood, "CASE Closed?", *CIO Magazine*, April 1 2000.
[85] On the distinction between software engineering and computer science, see David Lorge Parnas, "Software Engineering Programmes are not Computer Science Programmes", *Annals of Software Engineering* 6, no. (14) 1998):19-37.
[86] Harry Collins, "Actors' and Analysts' Categories in the Social Analysis of Science ", in *Clashes of Knowledge*, ed. Peter Meusburger, Michael Welker, and Edgar Wunder (Amsterdam: Springer Netherlands, 2008):?.

knowledge, assumptions, or concepts unavailable to them as they worked. In an extreme case this can lead to the kind of incomprehension and misinterpretation described by Thomas Kuhn in understanding scientific practice after a paradigm shift.[87] To do their work historians must approach the past carefully, accepting certain assumptions and terms while working hard to historicize others.

"The software crisis" is clearly an actor's category, and as such should be used with great caution as a neutral historical description of an actual event or era. Yet, as I will show below, it is invoked with much greater relative frequency and given far more prominence in the work of historians than in the memoirs and reminiscences of historical actors themselves. Somehow the concept has escaped into the world of historians without ever being properly domesticated.

The NATO Conference itself has not gone unremembered among software engineers, although we should remember that self-identified software engineers make up a tiny fraction of the computing workforce. In 1989 Tomayko organized a panel on the impact of the NATO conference as part of a conference with the theme "Twenty Years of Software Engineering." The panelists were Bernard A Galler, David Gries, Doug Ross, and Mary Shaw. All but the latter had been part of the 1968 conference. Tomayko then published this material in the Anecdotes section of *Annals of the History of Computing*, which he was then editing. His introduction noted "These conferences are considered as signaling the beginning of the emergence of software engineering as a discipline separate from computer science…"[88] Mary Shaw, one of the most important academic advocates for software engineering from the 1980s onward, did recall the proceedings of the 1968 NATO Conference as important to her own understanding of the field.

> I don't think we at Carnegie Mellon were especially aware that Al Perlis had gone to Garmisch, but when the proceedings came out early in 1969 he got a box of them and dispensed them to the graduate students "Here, read this. It will change your life.[89]

Yet even here is notable that neither Tomayko nor the other contributors used the exact phrase "software crisis" though Galler did write that (p.132) "There was a 'crisis' in the software field."[90]

Today the software crisis occupies a much more prominent place in the work of historians than it does within the literature of the software engineering field. It is by no means obscure in the technical literature. A quick search confirms that it remains a staple of cursory historical overviews within the community, such as those given in the opening sections of keynote addresses or papers written by students. Authors continue to advance their own favorite techniques as a solution to the software crisis. But it is a relatively obscure part of the discourse.

The clearest illustration of this comes in comparing two books on the history of software, published by Springer within a few months of each other. The first is the volume *Mapping the History of Computing:*

---

[87] Thomas S Kuhn, *The Structure of Scientific Revolutions (2nd ed.)* (Chicago: University of Chicago Press, 1969).
[88] Bernard A Galler, "Annecdote: Thoughts on Software Engineering", *Annals of the History of Computing* 11, no. 2 1989):132-133.
[89] Ibid.
[90] Ibid.

Software Issues based on a workshop held in 2000. Historians set the agenda and provided the five main papers, though some eminent computer scientists were present and provided some of the comments. The exact phrase "software crisis" appears thirty times in its 283 pages. As I mentioned earlier three of those five main papers began with discussion of the NATO conference.

The other is its mirror image, *Software Pioneers: Contributions to Software Engineering* based on a conference held in 2001 to which a diverse range of pioneers were invited. Among them were many of the key figures of early software engineering and programming methodologies such as Tom DeMarco, Edsger Dijkstra, Barry Boehm, David Parnas, Michael Jackson, Peter Chen, Michael Fagan, C.A.R. Hoare and Frederich L. Bauer. Each pioneer contributed a classic technical paper to be reprinted and most presented new lectures reflecting on the origins and impact of their ideas. The NATO Conference was mentioned only once in the text – in Dijkstra's reflections. Its proceedings were cited twice, once in the reprint of Parnas's 1972 paper and once in Boehm's 2001 reflections. The NATO Summer School on Programming Languages, to which historians have paid little to no attention, is actually referenced more frequently than the NATO conference on Software Engineering.

The phrase "software crisis" does not appear, to the best of my knowledge or that of Amazon's search inside the book feature, on a single one of the 728 pages of this hefty volume. In contrast the software pioneers mention Algol on 64 different pages. One might conclude that the influence of the software crisis and the 1968 NATO Conference on the early software engineering has been overstated and the influence of the Algol effort has been understudied.

One sees a similar pattern in the volume "In the Beginning: Recollections of Software Pioneers" edited by veteran software commentator Robert L. Glass. Among the authors of the short memoirs assembled here are such fixtures of the software engineering community as Peter J. Denning, Barry Boehm and Watts S. Humprey. Both the software crisis and the 1968 NATO Conference pass unmentioned. No doubt most if not all of these men have heard of the software crisis and may well have referenced it or the NATO conference elsewhere in their work. The point is that when asked to identify for themselves the context and influences on their work they looked elsewhere.

Why do We Love Garmisch

Why have the software crisis and the 1968 NATO Conference on Software Engineering so captivated two generations of historians, while occupying a rather more marginal role in the historical reflections of leading software experts?

First there is the name. Historians, particularly historians of science, have been trained to look for crises. Thomas S. Kuhn argued that a crisis with a scientific community is a prelude to a "scientific revolution" in which its members adopt a new paradigm showing potential to overcome the anomalies accumulating around their old paradigm. Mahoney was Kuhn's student, and so was naturally sensitive to the ability of a crisis to expose the usually unspoken assumptions underlying technical practice. Indeed Mahoney liked to remind us of the original medical meaning of "crisis" in classical medicine. In the

Hippocratic school a crisis was a turning point in the progression of disease at which its course would turn either toward death or toward a natural recovery.[91]

Then there is the nature of the conference report. This is neither a conventional collection of papers presented or an actual transcript of the discussion. Portions of it read like a transcript, but in fact it is more of a Platonic dialog constructed with fragments of the precirculated position papers, at least one document written during the conference, and précis capsules from the verbal discussion. In most cases these summaries were made without reference to the recording. According to the introduction "owing to the high quality of the reporters' notes it was the, in general, possible to avoid extensive amounts of tape transcription."[92] A handful of the working papers are reproduced in full, or at least at greater length, in an appendix. So the report as we have it is already a highly processed, well structured document edited to highlight particular topics and publicize certain debates. As the editors note, "points of major disagreement have been left wide open…"

The report is thus a candy box of well turned phrases and witty exchanges, calling out to the historian for inclusion in any paper on a somewhat relevant topic. The mumbles, misunderstandings, digressions, inanities and meanderings of actual verbal discussion have been stripped away to leave a succession of epigrams and bon mots, often presented without enough of their original context for us to really understand their significance. Anyone spending time with the secondary literature on software history will be familiar with Graham's joke about the Wright brothers pushing planes over cliffs, the suggestion from McIlroy that software writers were "receiving the short end of the stick" from their colleagues working on hardware in a clash of crofters versus industrialists, the note in the introduction that the phrase "software engineering" was "chosen to be provocative" and several other choice turns of phrase. Indeed the one about the Wright brothers was so quotable that the editors inserted it into their fabricated dialog twice, in different sections of the original proceedings! Of course the participants in the meeting were an unusually eloquent bunch who made an exceptional number of quotable points. But I think the presence of so many different voices in such a well crafted dialog has surely contributed to the ubiquity of the report in historical writing on software.

Once established in the secondary literature as the key moment in the history of software the conference has naturally attracted further attention and become the obvious conclusion or departure point of any work on the topic. Its influence on particular areas of practice is more often asserted than demonstrated.

Rethinking the Crisis

So what really happened at Garmisch in 1968 and why should we care? A rather crude summary of the arguments given above would go like this: a group dominated by operating systems and programming language researchers from universities and corporate research laboratories was assembled. Many knew each other from previous international efforts, particularly Algol. Participants from Bell Labs gave first

[91] Mahoney made this remark at the SOFT-EU meeting held in January 2008, during discussion of an earlier version of this paper.
[92] Naur and Randell, eds., *Software Engineering*, 6.

hand horror stories of the problems crippling the Multics project and others passed on gossip about the problems facing OS/360, PL/I, large scale timesharing systems, and other ambitious systems software for their generation machines. This was what they had in mind when they talked about "software" – applications programs and even business-oriented systems software such as data base management systems were barely mentioned. The idea that a crisis of some kind afflicted industrial systems software production was mentioned but not universally accepted.

Most participants agreed that dramatic action of some kind was needed, and endorsed the idea that current industrial practices could be improved by the application of their research expertise. Most endorsed the idea that software should be "engineered." Many ideas on how to present this were put forward, though given the background and employment of the participants it is unsurprising that these focused largely on the integration of mathematical theory into programming practice or the development of new kinds of system software. The follow on conference in 1969 failed to produce a consensus on what software engineering might actually be or whether mathematical theories of computation had an important part to play in real world software development.

The immediate impact of the conference was limited, and felt mostly in elite schools of computer science. Despite being used by the editors of the 1968 and 1969 NATO Conference on Software Engineering proceedings volumes the phrase "software crisis" gained little currency until it was adopted by Edsger Dijkstra in his 1972 Turing Award lecture. Since then it has retained a significant niche in the rhetoric of the software engineering movement, and over the past two decades much larger one in the historical literature on software. This was but one of a host of crises proclaimed in many areas of computing since the 1950s.

The vast majority of programmers during this era developed applications programs in end user companies, most commonly in administrative data processing departments. The data processing literature had been full of complaints, crises and grumbles since the late 1950s but neither the software crisis nor the idea of software engineering gained much traction in the data processing world in the decade following the 1968 conference. Its impact on data processing was eventually felt more through the adoption of discrete software technologies and system development methodologies rather than the adoption of new identities or ideologies. The formal methods faction of software engineering never had much impact among programmers of any kind, although some of its ideas filtered into tools such as static code analysis systems. Some ideas mentioned at the 1968 conference and pursued in the early days of software engineering played an important role in shaping modern software tools such as integrated development environments, object oriented programming languages, software components, and generalized tools on the Unix model.

Finally the most important influence of the software engineering movement on application system development has probably come through the widespread adoption of packaged methodologies such as Jackson Structured Programming, Yourdon Structured Design, SSADM, PRINCE and their many descendents. These place the traditional tools of business systems analysis, many of which predate the use of computers, within a strong framework of specialized project management methodologies. This wing of the software engineering movement has little connection to the community represented at

Garmisch or to most of the concerns expressed there. In particular it does not accept the assumption that using suitable mathematical methods or developers of sufficient inherent ability and advanced academic training can keep projects small enough to render elaborate management methods unnecessary. When we begin to investigate actual application development practice we may well discover that these methods are largely an evolution of early practice with administrative "systems and procedures" groups from the 1940s onward with few abrupt discontinuities and little of substance imported from computer science.

What places do the NATO Conferences on Software Engineering and the software crisis deserve in future work on the history of computing? Clearly they will continue to play a significant part in histories of the software engineering movement, though future writers on the topic would do well to recognize the initial association of software with systems tools and pay particular attention its gradual shifting of attention to application development methodologies. It's striking how much more prominent both are in the academic literature on the history of software engineering than in the memoirs of software engineering pioneers themselves. My initial analysis of the backgrounds and interests of attendees suggests that the conference deserves a prominent place in histories of operating system research, when some are eventually written, perhaps as a key moment in the development of a community of operating systems researchers as opposed to operating systems developers. Likewise the conferences may be a turning point in the evolution of programming language research, and perhaps in the refocusing of efforts from Algol to other aspects of systems software and programming theory.

On the other hand scholars interested in application development or the evolution of actual programming practice are likely to find that the current high profile of the software crisis and the NATO conferences in the secondary literature represents an unsustainable bubble overdue for collapse. We have been mixing and matching between the histories of quite different things, seizing on the pronouncements of men like Dijsktra without doing the hard work of social history needed to put the conference in its proper historical context or understand just how isolated his assumptions and experiences were from those of administrative programmers. Perhaps a temporary moratorium on beginning articles with references to software crises or Garmisch might help to push work on the history of programming out of the intellectual eddy in which it seems to be in danger of getting caught. To borrow a formulation with which readers of this literature are surely familiar, we may conclude that without real social histories of computer science and of programming practice the following warning is appropriate: "Quoting Dijkstra considered harmful."