Whiggism in Computer Science: Views of the Field

John A. N. Lee

TR 92-17

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

April 14, 1992

Whiggism in Computer Science: Views of The Field

John A. N. Lee
1992 April 14

Abstract

The teaching of any science is not complete without the inclusion of those elements that constitute the foundations of the field. While the history of computing is an element of those fundamentals in computer science, history should not be taught merely as a diversion from the technology but as a series of case studies supporting scholarship from ethics to virtual reality. Understanding the successes of the field provides only a biased view of the technology and hides the true nature of pioneers; understanding failures provides direction and guidance, and occasionally new insights that were ahead of their time and whose time has come.

Introduction

Whiggism can simply be described as the view of an event with the full availability of hindsight. Anthony Hyman, in his introduction of the term to the history of computing community[1], was concerned, for example, that it was inappropriate to look at the life and work of Charles Babbage from the point of view of modern day computing. Babbage did not have the advantage of the knowledge of the work of Alan M. Turing and Johnny von Neumann, and thus it is unfair to judge his actions in the light of the knowledge of our world. If one is to express opinions on the accomplishments (or failures) of Charles Babbage, then it is only appropriate to express those opinions on the basis of the state of knowledge of the world as it existed in his time. For example, Babbage has been castigated for designing machines that required a capability of machining that far surpassed the capabilities of the time. Michael Wright, curator of the machines exhibit at the Science Museum, showed[2] that precision machining of the quality expected by Babbage was quite commonly used in the construction of intricate clocks and locks. Perhaps the difference between the precision needed in clock and lock mechanisms and that needed for the components of the Difference Engine is that the former is a basically sequential system where there is minimal feedback between

[1] Hyman, R. Anthony. 1990. "Whiggism in the History of Science and the Study of the Life and Work of Charles Babbage", *Ann. Hist. Comp.*, Vol. 12, No. 1, pp. 62-67.

[2] Wright, M. T. July 1991. "Building the Difference Engine: Workshop Practice and Capability in Babbage's Day", *Babbage/Faraday Bicentenary Conference*, Cambridge, UK.

components, while the latter requires multiple interconnections between elements at all times. The construction of the Difference Engine at the Science Museum in 1990-91 showed that Babbage had overcome the need for extreme tolerances by the inclusion of self correcting mechanisms, even though greater precision than was required was possible! Similarly, in a error brought on by Whiggism, authors of Babbage biographies[3] have made a great deal of mileage out of Babbage's opposition to street musicians. A play by Maurice Wilkes[4] opened with a discussion between Babbage and his solicitor concerning the charge of "disturbing the peace" made against a street musician that the judge had deemed to be invalid. The recently discovered autopsy report on Babbage's body[5] revealed that he suffered from a calcification of the carotid and vertebral arteries. His great, great-grandson, Neville Babbage, a general physician, concludes that this would have produced a "recruitment of hearing" and thus Babbage could have been suffering from truly painful sympathetic vibrations when "attacked" by particular notes included in the repertoire of street musicians! This lack of information about the environment in which he lived, led prior authors, such as Mosely[6], to misinterpret Babbage's sensitivities and mislabel him as the *Irascible Genius*. Understanding his problems allows us to view the man very differently. We had put his actions down to general social dysfunction; in fact, he was in pain. There are numerous examples in the interpretation of history, and particularly the history of computing, that suffer from Whiggism. In some, we belittle the accomplishments of pioneers when compared to our present state-of-the-art, when in fact they started from nothing to accomplish the first step in a new line of development or discovery.

Whiggism is also an error in teaching.

Whiggism in Teaching

In teaching computer science[7] thirty years ago we could start from first principles and give the student everything there was to know about computing. We could show them the best route to a solution for a problem; there were very few alternative paths and the wrong paths were quickly obvious. Today, in a Newtonian phrase, *we are standing on the shoulders of giants*; and too often we fail to look down and see what those giants had to go through to achieve their giant-hood. For anyone who has been in the field for more than a few years, the lessons learned are the lessons, not of documentation, but of experience. Experience, that is, of having made mistakes, having made the wrong decisions, and occasionally of having achieved the hoped for goal. James Horning said of the relationship between experience and mistakes:

> *Good Judgement Comes from Experience*

---

[3] See for example, Halacy, Dan. 1970. Charles Babbage, Father of the Computer, Macmillan Co., New York.

[4] Wilkes, Maurice V. 1991. "Pray, Mr. Babbage ...", A Play, *Ann. Hist. Comp.*, Vol. 13, No. 2, pp. 147-154.

[5] Babbage, Neville F. June 1991. "Autopsy Report on the Body of Charles Babbage", *Medical Journal of Australia*, Vol. 154, pp. 758-9.

[6] Mosely, M. 1954. Irascible Genius, Hutchisons, London.

[7] It is, of course, an example of Whiggism to talk about Computer Science in 1960 - the concept may have been there, but not the term!

Experience Comes from Bad Judgement

As in most historical reporting, teaching the history of computing is often the teaching of successes; we fail to learn about the failures because they are either uninteresting or time consuming. The baseline for understanding our science rises each year and as we grow older we forget that the reasons for the support of that currently perceived baseline are not always obvious. We know about something because we stubbed our toes on that rock years ago, and we have impatience for those who do not have the knowledge of that injury as a learning experience.

An examination of central concepts in the science from birth to maturity shows a roughly parabolic complexity curve. In the beginning the initial idea is often expressed in such complex terms that it is overlooked by the majority of readers. The one or two who can see through the veil of complexity simplify the concept and express it in understandable terms that in turn can lead to implementation and application. As the system is used and explored, it is again modified and augmented to achieve a new zenith of complexity in that the level of applicability is high while the degree of understanding (seen as the complement of complexity) has dropped once again to a low setting. Three examples of the mid 1960's come to mind: LR parsing as initially presented by Knuth[8], the basic concepts of well-formed programs expressed by Boehm and Jacopini[9], and the concept of unification described by Robinson[10]. Each missed the mark initially but was re-interpreted by scholars and reduced to a meaningful implementation system before being supplemented and magnified again to create a highly useful but less understood application.

One of the areas in which Whiggism is most prevalent is in the teaching of programming. The art of programming (to use Don Knuth's phrase) did not evolve full blown. It developed over a long period in which the errors of process produced the codes of practice of the next generation. The development of Fortran 9X from (Whiggishly called) Fortran I does not imply that newcomers need only be taught about the additions to the language on the assumption that the present generation of computer science students have the basics of the language embedded in their genes. We teach languages such as BASIC and Pascal, in an incremental manner, starting from an understanding of identifiers and imperatives, and progressively adding frameworks of control until the student can achieve some level of independence. In the process we probably (and most oftenly) emphasize iteration as a fundamental methodology of control, and introduce recursion as an anomaly in passing. Later, we attempt to build on the knowledge of recursion only to find that since it did not appear on the main path, it has been forgotten, or had never really been learned.

---

[8] Knuth, Donald E. 1968. "Semantics of Context-Free Languages", *Math. Systems Theory Journal*, Vol. 2, pp. 127-145.

[9] Boehm, Corrado and Giuseppi Jacopini. 1966. "Flow Diagrams, Turing Machines, and Languages with only Two Transformation Rules", *Communications of the ACM*, Vol. 9, No. 5, pp. 366-71.

[10] Robinson, J. A. 1965. "A Machine-Oriented Logic Based on the Resolution Principle", *Journal of the ACM*, Vol. 12, pp. 23-41.

Over the years we have seen frequent emerging fads and passions for improvements in the techniques of programming - from the use of high level programming languages, through disciplined, structured programming systems to object oriented systems; we have gone from programming in the small to programming in the large; we have gone from a subject area that was familiar to most participants, to a multi-faceted profession that supports specialists who are unable to converse with colleagues in disjoint areas. While only a few decades old, we get the impression that we teach computing as if certain elements are registered in the genes of our students. BASIC comes built-in and thus we can begin our explanation of a topic by jumping in halfway through its development cycle and ignoring the prehistory, thereby denying students the simple concepts that have simple everyday applications. Actually we do not always want to go back "all the way".

Michael Mahoney[11] retells a story, perhaps apocryphal, about Jean Piaget. The child psychologist was standing outside one evening with a group of 11- year-olds and called their attention to the newly risen moon, pointing out that it was appreciably higher in the sky than it had been at the same time the night before and wondering out loud why that was. The children were also puzzled, though in their case genuinely so. In his practiced way, Piaget led them to discover the relative motions of the earth, moon, and sun and thus to arrive at their own explanation. A month or two later, the same group was together under similar circumstances, and Piaget again posed his question. "That's easy to explain," said one boy, who proceeded to sketch out the motions that accounted for the phenomenon. "That's remarkable", said Piaget, "How did you know that?" "Oh," the boy replied, "we've always known that!" Not only children, but people in general, and scientists in particular, quickly forget what it was like not to know what they now know. That is, once you have solved a problem, especially when the solution involves a new approach, it's difficult to think about the problem in the old way. What was once unknown has become obvious. What once tested the ingenuity of the skilled practitioner is now "an exercise left to the student". Graduate teaching assistants and senior faculty are not immune to the error of Whiggism.

In fact, we need to lead students and learners through the possibly winnowed steps of development to understand where they are going and from whence the concepts and axioms have emerged. Winnowed, that is, not to rewrite history as if it were one long trail of triumph, but a trail that shows the major approaches, and the overtaking technologies, glued together with the rationale for not straying too far from the trail. If one wants to have students to have negative experiences then the simple solution is to let them blunder along devoid of supervision and hopefully to have them explore the same tarpits of extinction as did their teachers[12]. We simply seem not to have the time for that, and yet we simply cannot afford not to look in the tarpits occasionally to at least view bad experiences. Just because one technique has now been accepted as the "best" or the "norm" does not imply that we should not look at the alternatives (such as recursion) or on the path of operating systems (such as MS-DOS and VMS!) The process

---

[11] Mahoney, Michael. 1991. "What Makes History?", unpublished manuscript.

[12] Brooks, Frederick P., Jr. 1975. The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley Publ. Co., Reading MA.

of problem solving includes the process of distinguishing between alternatives, and making judgements about the capabilities and shortcomings of those options. However, if we initially place students too far along the trail, then they have not been privy to the myriad paths which led to that peculiar route and do not have a complete knowledge on which to base their future decisions.

A simple example was given to me by one of the early reviewers of this paper. His daughter, a computer science graduate following in her father's shoes, had joined a small company in the North East alongside graduates from three other institutions. One of their first tasks was to develop what amounted to a lexical analyzer, but not having access to a Unix system containing LEX, the other three were stumped! Having access to an education that included studies of compiler technology prior to the implementation of YACC and LEX, my friend's daughter was able to solve her employer's problem. An extreme example perhaps, but like other sciences, computer science education requires studies of the fundamentals of our field in which to build and deduce additional concepts. David Gries[13], for example, working from the first principles of program development through the axiomatic formalisms of algebraic semantics, was able to derive an algorithm for computing maximal subvectors that is an improvement over the previously best known technique. Going back to fundamentals and eliminating some of the basic assumptions may enable us to demonstrate different approaches to old problems.

Going even further back in history we discover that both John von Neumann and Alan Turing were not satisfied with simply devising programs; they accompanied their programs with proofs of correctness - somehow we have forgotten how to do this. After all, do not computers operate flawlessly? Anyone who was taught simple accounting prior to the introduction of the spreadsheet was coached in the necessities of cross-footings as a means of validating the correctness of the mental computations. Yet very few computer programs use redundant algorithms to verify the correctness of the system.

When Fortran was implemented in 1957, it was a purely commercial enterprise, not highly touted in the scholarly journals. The techniques of compilation were not presented as the long lasting primary achievement of the Fortran project, rather the language and the open prairie of applications were the covenant. Sheridan[14] presented a scientific view of the compilation process but the Greek characters and the mathematics hid simplistic solutions that were later shown to be optimal. The follow-up compiler for Fortransit was unable to follow this line of illumination and was forced to reinvent the techniques of translation. But then these were once again lost for the lack of documentation and lack of scholarly publication. Yet this compiler contained one of the most simple of arithmetic statement scanning techniques ever devised. Only a study of the history of programming languages revealed that system. Last week, while teaching my compilers class and while discussing optimization, a student asked why the Fortran group placed the minimization of memory accesses at the top of their

[13] Gries, David. 1989. Lecture during the "Year of Programming", University of Texas, Austin.

[14] Sheridan, Peter B. February 1959. "The Arithmetic Translator-Compiler of the IBM Fortran Automatic Coding System", *Communications of the ACM*, Vol. 2, No. 2, pp. 9-21.

priority list. He was unaware that the IBM 704 in use at the time the Fortran project started used Williams Tube memory in that the access time was several orders of magnitude different from that of the access time of "high speed" registers.

John von Neumann has been blamed by John Backus, creator of Fortran, for the "bottleneck" in computing and the restrictions of the so-called von Neumann machine. Early von Neumann recognized the advantages of parallel computation, but he also recognized in his time that the technology of architectural design was not sufficient to support truly parallel systems, that programming had not developed the techniques of multi-programming, and thus to solve the problems of the day he accepted serial or sequential computation - the very technology for which he is now blamed! On the other hand, von Neumann accepted the credit for the concept of the stored program; the EDVAC report was never completed and the first draft, which only ascribed authorship to von Neumann, was taken to be the original source of the idea. Over the years, von Neumann never took the time to dissuade his creditors of the actual genesis of the stored program concept. Similarly, Grace Murray Hopper has been credited by many writers of her several obituaries with the "development of COBOL". History clearly shows that she would better be credited as the *grandmother* - as the originator of FLOWMATIC, as the *midwife* - as a member of the organizing committee that recommended the evolution of a business oriented programming language, or as the *mentor* - in her capacity as one who promulgated the language and possibly instigated the writing of the first two compilers. Over the years, Dr. Hopper never took any steps to dissuade believers of this misplaced credit, but neither did she personally claim the credit for the language. John Mauchly claimed credit for the invention of the computer, and for its construction in conjunction with J. Presper Eckert. Yet in 1973, the patents were invalidated by the Minneapolis District Court and "one, John Vincent Atanasoff" was, by law, given the credit for the invention of the computer - at least in the US. Clark Mullenhoff[15] went so far as to accuse Mauchly of perjury on the witness stand. Mauchly's widow, Kay Mauchly, has, subsequent to the trial, discovered pieces of early electronic devices that her husband built prior to his visit to Atanasoff's home in 1941. But we are now aware that there was similar work going on in England by Alan Turing and in Germany by Konrad Zuse. After the unveiling of the ENIAC, the University of Pennsylvania asked J. Presper Eckert and John Mauchly to sign over their intellectual property rights to the university. The resulting disagreement led to the departure of Eckert and Mauchly from the university and the subsequent creation of the first electronic computer company. Harvard University never challenged Howard Aiken to give up his rights to the Mark I concepts; Aiken never questioned the university's rights and never formed a commercial organization to market his inventions. These are all topics from our history that are well documented and that can form case studies for students in so-called computer ethics courses.

Business and sociological studies of computer enterprises can reveal practices and procedures that had an effect on the cultivation and extension of the industry. John Hendry of the Cranfield Institute of Technology, England, investigated the impact of government policy on the fledgling computer industry in the United

---

[15] Mullenhoff, Clark R. 1988. Atanasoff: Forgotten Father of the Computer, Iowa State University Press.

Kingdom following World War II when the state of the exploration of the field was on a par with that in the United States. He entitled his book Innovating for Failure[16] . This study contains lessons that have applicability to the current vogue in state government to channel and encourage university research in the names of technology transfer and economic development. Several CEO's have recently written their memoirs in which they attempt to justify their decisions in managing their respective corporations but unfortunately their views are tainted by their perceived successes more than by their failures from which we might learn much more. Shane Greenstein, University of Illinois, has been investigating the methods and techniques of vendor lock-in that were used by computer purveyors in the 1970's and that can be a highly successful corporate strategy. The delivery of upward compatibility through successive machines and deliberate incompatibility with competitor's systems, coupled with the excessive costs of change-over, tended to sustain a virtual monopoly within government agencies that emanated from the initial computer system sale. A similar situation exists today with the need to replace personal computers and workstations, though the evolution of open systems standards has diminished this impact to some extent; Greenstein's studies[17] have an applicability that is worthy of consideration.

Whiggism also prevents us from recognizing artifacts that were originally given some other name but which preceded the classic introduction of the same item. For example, index registers, that many believe were introduced with the IBM 704, in fact appeared in the University of Manchester in the Mark I under the name "B-lines". Conversely, Grace Hopper introduced the concept of a compiler in 1952 and so we give her the credit for the introduction of a whole set of compiler components that apparently were not in her mind at the time! Charles Babbage described a computer-like device that contained a "mill", and a "store"; instead of control, Babbage speaks of "government", and for programs he used "variable cards" and "operation cards". A Whiggish look at history prevents us from seeing innovations that may well have present-day applications but that are not couched in present day terms.

Perhaps one of the most difficult of determinations is the identification of "firsts". As we just discussed, one problem is recognizing the "first" under some other name. So often throughout the history of computing, elements of the technology have "condensed" out of the cloud of postulates that have emerged warm from the sea of perceptions. The question of who *first* invented the computer can be answered several ways. John Atanasoff, in the US, generated the idea of a special purpose electronic system while Konrad Zuse, in Germany, was wrestling with the problem of a general purpose relay driven system similar to that of Howard Aiken at Harvard University. Simultaneously Alan Turing, in Great Britain, was applying a more abstract techniques to conceptualize the Universal Machine that eventually bore his name. Not one of them termed his system a "computer" and thus if we were to perform a contextual search to locate contemporary documents we would not locate references to their work. The term "computer" still referred to a human operator.

---

16 Hendry, John. 1990. Innovating for Failure, MIT Press, Cambridge.

17 Greenstein, Shane. 1990. "Did Installed Base give an Incumbent any (Measurable) Advantages in Federal Computer Procurement?", *Faculty Working Paper 90-1718*, Political Economy Series #42, Univ. of Illinois.

Atanasoff's drum memory had many of the elements of later memory devices both as a rotating contrivance and as a regenerative system; those same concepts were independently discovered and reintroduced in 1950's systems - without contravening the patent or copyright laws.

Technology, Engineering or Science?

The progression of understanding and application of computing has moved from empirical, ad hoc, skill-based activities, through increasing comprehension to a triumvirate position in which there simultaneously exist technologists, engineers and scientists. Distinct from a "true" science, such as physics, the ability to "engineer" a product has preceded the full scientific understanding of at least the software process. Paul Cohen from the University of Massachusetts undertook to survey[18] of the gist of a year's worth of published articles in artificial intelligence, mainly to support his thesis that not enough was being done to develop the theoretical underpinnings of the science. His results, I believe, mirror the state of the understanding of certain aspects of engineering a hundred years ago. This may indicate that we are "on track" in the transmogrification of computer technology into a science and thence into engineering. However, our acceptance as an engineering discipline has been thwarted, in my mind, by the lack of concrete artifacts that can be subjected to external examination by investigators. Perhaps Thomas J. Watson, Sr. was not all wrong when he steadfastly resisted the introduction of electronics into the data processing business because he felt that maintenance of the components could not be achieved readily. He believed that what you could not see could not be fixed! Thus the field of computing, though named "computer science" since the mid 1960's, still has a foot in the domain of technology. It is our responsibility to develop the science so as to understand the artifacts and their properties, and then to engineer reproducible products which have a reliability comparable to that that we expect in products from other engineering disciplines. [However, the word *engineering* carries less than an envious position in the scale of social graces. Nick Metropolis, developer of the MANIAC system for the Los Alamos Laboratory, and author of some of the early programs for the understanding of the processes of nuclear detonation, tells the story[19] of a "distinguished lady ... extolling her admiration for Professor S. 'And what department at MIT does he belong' she finally asked. 'Mechanical Engineering' Metropolis responded as a look of horror crossed the lady's face. 'Why I thought he was a scientist' she blurted out".

Our current scientific understanding of the field of computing is incomplete; our ability to engineer reliable products is also incomplete. Like a canoeist, we sit bobbing on the edge of an eddy, on the one hand, ready to be swept downstream to continue the fight against the flow of nature, and on the other hand, straining to achieve the chance to collect our thoughts in the still waters of understanding. This oscillation between pure and applied science swirls us around, the most effective progression downstream being accomplished by conserving our energy and analyzing the last descent, before we tackle the next obstacle.

---

[18] Cohen, Paul. 1991. "A Survey of the Eighth National Conference on Artificial Intelligence: Pulling Together or Pulling Apart?", *AI Magazine*, Vol. 11, No. 4.

[19] Metropolis, N. 1991. "The Age of Computing: A Personal Memoir", American Academy of Arts and Sciences.

There is a certain amount of *technolust* in our profession today that discards the immediate past in favor of the latest toy; we do not pause long enough either to enjoy the tranquility or to understand what we have accomplished. A well mounted hype that builds desire for glossy new artifacts has begun to win over reason and planning. As the amount of data left behind increases our ability to analyze the ongoing experiment grows more and more difficult. What *really* did work and what not? Is speed taking the place of ingenuity? Is faster so much better? The announcement of a new system in one of the glossy personal computer magazines suggests that all prior models are now to be viewed with contempt and to be regarded as being superfluous. Old systems become museum pieces or are handed down to the "less fortunate" - to the biologists and the educators.

The acceptability of historians of modern and contemporary technology in universities is at a low ebb. Metropolis[20] has pointed out that histories of long obsolete discoveries, such as the steam engine, are much more acceptable. As the reviewer from a mathematical journal pointed out in rejecting a manuscript on algorithms (while recommending it's forwarding to the *Annals of the History of Computing*), the programming language Pascal is by no means as permanent as formulae written in Greek or Latin! Had the same paper included algorithmic expositions in terms of the programming language APL, perhaps it would have been more acceptable!

Too often institutions look upon historians as non-technical humanists who turn to the study of history in the face of their inability to handle the current theories of the field they study. Like the study of ethics or educational applications in our computer science curriculum, contemporary history is thought to be the hiding place for professors approaching senility. There is a place for historians in computer science and the study of historical elements of the field by students, at all levels, is legitimate. The historian cries out to the fleeing back of a scientist "pause to smell the flowers"!

Conclusion

Does this mean that I advocate that every course should include some history? Not necessarily, but I do advocate that some of the historical venues should be visited. I believe that we do students an injustice when we start from a point at which we presume that everything else is just irrelevant prolog.

Does this mean that I advocate teaching the negatives? To a certain extent there is learning in examining negative or complementary aspects of a subject; in fact, the negative approach, as in proof by contradiction, can reinforce the acceptance of the positive. In almost every subject we teach, there was a period of development in which alternatives were tried and rejected. Not all alternatives failed; they were simply superseded. Some alternatives were ahead of their time and now have applications.

Dos this mean that I advocate teaching only the successes? To merely teach the successful algorithms should not suggest that there is no merit in those that are no longer used. In some cases, a quick and dirty solution to a problem (now-a-days called rapid prototyping) can more easily be achieved with simple tools, that are, unfortunately, no longer taught.

I would propose a compromise solution between teaching basic history and only dealing with the state-of-the-art:

---

20 Ibid.

each course could "tithe" 10% of its syllabus to looking backward to learn about what led up to the state-of-the-art - 5 minutes per class, 3 class periods out of 30. The state-of-the-art in Columbus' time was the theory that the world was flat, and so long as one limited travel to within sight of land, that worked within acceptable (or perhaps unobserved) tolerances. Had not Columbus being willing to challenge the underlying assumptions we might not be here; or at the very least someone else would have crossed the Atlantic. To solve many everyday problems we can accept a flat earth theory; to progress we need to understand the limitations of simplifying assumptions and to overcome them to find the "new world".

Richard Hamming once observed about one of the supportive systems of computing that

> *"one should know enough about [it] to protect one's self against it!"*

We and our intellectual offspring need to know enough about the tarpits on either side (and even under) the path to protect ourselves against them.

The 1990's has been labelled as the decade in which we get "back to basics", an era when we look at our educational process and lift the country into an upper percentile of nations with superior accomplishments in science, mathematics, and social studies. I am convinced that walking that extra mile in our educational system starts with the first step - putting the best foot forward by teaching the elements of the subject in a clear, understandable, lucid, intelligible manner. Starting right sets the pace for later understanding and establishes the level of the shoulders on which our students are going to stand. Tony Hoare is reported to have said that in every large problem there is a small problem trying to get out! Small problems need small solutions, and solutions that are not complicated by the Whiggish views of extended scientific development. Too often simple solutions get lost in the rush to the sophistication of scientific method and we find ourselves using a steam hammer to break open a peanut.