What Has Literature to Offer Computer Science?

Mark Dougherty

In this paper I ask the question: what has literature to offer computer science? Can a bilateral programme of research be started with the aim of discovering the same kind of deep intertwining of ideas between computer science and literature, as already exists between computer science and linguistics? What practical use could such results yield?

I begin by studying a classic forum for some of the most unintelligible pieces of prose ever written, the computer manual. Why are these books so hard to understand? Could a richer diet of metaphor and onomatopoeia help me get my laser printer working?

I then dig down a little deeper and explore computer programs themselves as literature. Do they exhibit aesthetics, emotion and all the other multifarious aspects of true literature? If so, does this support their purpose and understandability?

Finally I explore the link between computer code and the human writer. Rather than write large amounts of code directly, we encourage students to write algorithms as pseudo-code as a first step. Pseudo-code tells a story within a semi-formalised framework of conventions. Is this the intertwining we should be looking for?

In this paper, I will try to explore whether there is a meaningful relationship between computer science and literature studies. Since computer science is very broad, some focus is needed and I have, perhaps not surprisingly, identified artificial intelligence (AI) as a particularly relevant branch of the subject. AI touches on application areas close to literature studies, such as natural language processing and contextual representation. It also involves deep philosophical debates and histori-

Dougherty, Mark. "What Has Literature to Offer Computer Science?" *HUMAN IT* 7.1(2004): 74–91

cally has been an "open" multi-disciplinary activity. A key aim of initiating such a discussion is the difficulties that many computer science students experience when learning to write programs and their related documentation. A key hypothesis is that an educational programme founded purely on mathematics and logic does not provide the necessary tools to communicate to other people the complex ideas and abstractions often represented in computer programmes.

Before embarking upon any detailed discussion of the issues involved, it is interesting to briefly review a long-standing philosophical discussion concerning what engineering and mathematics have in common with the arts. In the preface to his novel *The Picture of Dorian Gray* (1890), Oscar Wilde wrote:

> *All art is at once surface and symbol. Those who go beneath the surface do so at their peril. Those who read the symbol do so at their peril. It is the spectator, and not life, that art really mirrors. Diversity of opinion about a work of art shows that the work is new, complex, and vital. When critics disagree the artist is in accord with himself.*
>
> *We can forgive a man for making a useful thing as long as he does not admire it. The only excuse for making a useless thing is that one admires it intensely.*
>
> *All art is quite useless.*

Wilde is quite uncompromising. There is no relationship between engineering and art; the two activities are totally disjoint. The gulf between stoic and epicurean can never be crossed. Wilde laces this argument with a humorous irony, but the reader is left with a clear impression that although art may be "useless" it is immeasurably more important than practical matters.

Jules Verne in *Paris in the Twentieth Century* (circa 1863; provenance disputed) describes a world in which this supposed gulf between engineering and art is extremely wide. However, Verne argued that science and engineering would dominate by 1960, with art and literature largely abandoned. Verne, of course, would never have subscribed to this philosophical viewpoint, but was nevertheless a little afraid of the power of technology. It is ironic that at around the very time of Verne's future

scenario, Jacques Ellul published his diatribe against technocracy, *The Technological Society* (1948), the same year in which George Orwell wrote *Nineteen Eighty Four*.

Yet at the other end of the spectrum we find the First Proclamation of the Weimar Bauhaus (circa 1920). See *Bauhaus, 1919-1928* 1952 for an introduction:

> *Art is not a profession. There is no essential difference between the artist and the craftsman. The artist is an exalted craftsman. In rare moments of inspiration, moments beyond the control of his will, the grace of heaven may cause his work to blossom into art.*

I lean toward this second view, although I think the first sentence a little extreme. Fine arts, poetry, music and numerous other items I could mention all enrich our lives and are provided to us by professional artists. Yet I think the basic philosophy is sound; there is something of an artist in all of us, regardless of our chosen profession. Likewise there is something of an engineer in every artist; Samuel Florman (1994) relates in *The Existential Pleasures of Engineering* an illuminating conversation:

> *Other artists have found in the machine a pure beauty that seems completely isolated from subjective experience. Fernand Léger tells of visiting an airplane exhibition with fellow artists Duchamp and Branscui. Duchamp, according to the story, turned to Branscui and said "Painting has come to an end. Who can do anything better than this propellor?" "I myself," relates Léger, "felt a preference for the motors…. But I still remember the bearing of those great propellers. Good God, what a miracle!"*

Perhaps nineteenth and early twentieth century technology was more accessible. A modern microprocessor is an engineering marvel, but is pretty unexciting from an external viewpoint. Creating a link between computer science and literature studies is going to require rather more thought; the same kind of flash of insight as described above is perhaps not very likely.

Language and Computer Science

The formalisation of language has been fundamental to computer science. A computer must be able to interpret the meaning of a program;

a description of an algorithm written by a human. In general this task is carried out by a compiler, a special-purpose piece of software which translates the human-readable program into a machine-readable series of binary operations.

In order to ensure reliable and meaningful results, compilers have to be made highly restrictive. They only accept programs which adhere to a certain pre-determined set of syntactical rules. The semantic meaning of these programs (i.e. what algorithm does a particular program represent?) has to be as unambiguous as possible.

One finds that to accurately define allowable syntax is relatively easy. The study of formal grammars (Chomsky 1956) has yielded a set of logical and mathematical tools, which can be used to rigorously define the syntax of programming languages. Furthermore the same tools can be used as a basis for building the necessary parsing machinery within compilers.

Work in this area has also yielded specialist text-processing languages such as SNOBOL and ICON. These have been widely used in the humanities (Johnson 1999) for tasks such as word counting, producing statistics on sentence length and extracting all dialogue for a particular character for further analysis.

Unfortunately, defining semantics accurately is very hard and research efforts in this direction currently occupy many computer scientists. Even when a denotational approach is taken, where we are concerned only with the meaning of programs and put to one side *how* this meaning is to be implemented, pinning down the meaning of even simple programs is fraught with difficulty. Subtle traps abound and the problem belies its apparent simplicity, given the highly restricted nature of compilers and computer languages. Nielsen and Nielsen (1992) give a good introduction to the field.

Natural Language Processing and Artificial Intelligence

More recently, serious efforts have made in the world of artificial intelligence to build programs which can process natural language. By natural language we mean "normal" human speech and writing, as opposed to formalised, restrictive computer languages. Apart from obvious practical applications (wouldn't it be nice if we could communi-

cate with our computers instead of having to learn to type?), solving this problem is widely recognised as the holy grail of artificial intelligence.

The motivation is that speech and language is a key aspect of human intelligence. This was first expounded by Descartes in his great tour de force *A discourse on method of rightly conducting the reason and seeking truth in the sciences*.

> *if there were machines bearing the image of our bodies, and capable of imitating our actions as far as it is morally possible, there would still remain two most certain tests whereby to know that they were not therefore really men. Of these the first is that they could never use words or other signs arranged in such a manner as is competent to us in order to declare our thoughts to others: for we may easily conceive a machine to be so constructed that it emits vocables, and even that it emits some correspondent to the action upon it of external objects which cause a change in its organs; for example, if touched in a particular place it may demand what we wish to say to it; if in another it may cry out that it is hurt, and such like; but not that it should arrange them variously so as appositely to reply to what is said in its presence, as men of the lowest grade of intellect can do.*

In a seminal work, Turing (1950) put this philosophical standpoint into the context of the newly emerging science of computing. In doing so he laid down the foundations of artificial intelligence. Turing's idea was to devise a test of machine intelligence, now known as the *Turing test*. In this test a human interrogator is placed in a closed room and is able to communicate via computer terminals to two other rooms. In the first room is another human and in the second room is a computer running an "artificial intelligence" program designed to converse in natural language. The computer passes the Turing test if the interrogator is unable to determine which terminal communicates with a human and which communicates with a computer.

Fifty years later, no attempt to pass the Turing test has succeeded. Just a few minutes interrogation is sufficient to make a successful distinction, even if the domain of the conversation is limited to a specific topic. Turing's final sentence of the paper has proved more prophetic than he himself realised: "We can see only a short distance ahead, but we can see

plenty that remains to be done." Why is understanding natural language so difficult? The answer supplied by Wittgenstein (1953) is that human speech is neither objective nor amenable to formalisation. This philosophical foundation has inspired some authorities such as Dreyfus and Dreyfus (1985) to deny the possible existence of AI. This may be throwing the baby out with the bath water, but as yet we are not in a position to make a judgement. What is at stake is whether the human mind is restricted by the same fundamental limitations that theories such as Gödel's incompleteness theorem and the Church-Turing Thesis place upon the mechanical and electronic computers. See Lucas (1961) for a fascinating article on this issue.

Computers and Literature

If computers are still so woefully poor at understanding simple natural *language*, is computer understanding of *literature* really a relevant or sensible domain? Or can computer science only provide tools to aid the creation and understanding of literature by humans? Whilst computer tools have yielded several interesting literary concepts (e.g. interactive poetry), this does not amount to the same deep relationship as exists between computing and linguistics, as summarised above. If the role of computing is to be more than a provider of services to the study of literature, I believe we have to dig a little deeper in order to find parallels and a more fundamental relationship which is more reciprocal in nature.

First it must be necessary to define what distinguishes the study of literature from the study of language. A (non-exhaustive) list of criteria might be:

- Higher-order semantic meanings
- Fictional content
- Metaphor and other such abstractions
- Aesthetics
- Emotion
- Style (on the border with linguistics)

In essence, analysis of literature involves a further layer of what we term meta-analysis in the world of computing, over and above linguistic analysis. Meta-analysis involves concepts which themselves are made up of lower-level concepts. As such it is a strictly relative term.

One might ask: is not all meaning wrapped up in the general term "semantics"? One could, but I think this would confuse the issue. To know what pain *is* can never be equated with *experiencing* pain. To quote Frank Zappa: "The computer can't tell you the emotional story. It can give you the exact mathematical design, but what's missing is the eyebrows."

In computer science, semantics are dealt with formally using mathematics, but no such mathematical tools exist for dealing with higher semantic concepts. In fact even the syntax of higher-level concepts has yet to be pinned down; only relatively restricted higher-order logics have been formalised. The contextual knowledge which an average human being has access to and applies on a daily basis, is therefore still very poorly understood. This *frame problem* and the need for higher levels of reasoning is discussed by Dennett (1984):

> *when we think before we leap, how do we do it? The answer seems obvious: an intelligent being learns from experience, and then uses what it has learned to guide expectation in the future. Hume explained this in terms of habits of expectation, in effect. But how do the habits work? Hume had a hand-waving answer – associationism – to the effect that certain transition paths between ideas grew more-likely-to-be-followed as they became well worn, but since it was not Hume's job, surely, to explain in more detail the mechanics of these links, problems about how such paths could be put to good use – and not just turned into an impenetrable maze of untraversable alternatives – were not discovered. Hume, like virtually all other philosophers and 'mentalistic' psychologists, was unable to see the frame problem because he operated at what I call a purely semantic level, or a phenomenological level [....]*
>
> *That is the mechanical question the philosophers left to some dimly imagined future researcher. Such a division of labour might have been all right, but it is turning out that most of the truly difficult puzzles of learning and intelligence get kicked downstairs by this move.*

Thus one answer to the question at the beginning of this section might be that computer understanding of literature is simply irrelevant at the current time; we have no sufficiently powerful tools or theoretical basis upon which to base such a study. However, this is perhaps a little too pessimistic. Although tackling heavy literature head on is beyond us, we can speculate as to whether the world of literary analysis has some insights, which might help the world of artificial intelligence solve some aspects of the above-mentioned frame problem. Just as computer science has "borrowed" much from linguistics in order to solve problems which were pressing and difficult at the time, could borrowing from literature studies yield similar dividends?

Such an approach would use human literature as a layer of abstraction above the mind itself; a sort of mezzanine floor in Dennett's metaphorical staircase, where we make an attempt to catch at least some of the difficult problems on their way to oblivion.

Another possibility is to examine the world of computer science for signs of literary development. Perhaps the seeds are present even if there are no flowers to speak of. I will therefore turn my attention to various aspects of the process of designing, building and documenting computer systems. I will argue that some seeds of literature are indeed present, which perhaps give some hope for interesting developments in the future.

Documenting a Computer System

Some of the problems that the world of computing is still struggling to solve are best illustrated by one or two anecdotes. My father lectures in the Department of Applied Mathematics and Theoretical Physics at Cambridge University. Clearly he is no fool, although perhaps I rather unjustly thought so during my teenage years. Like most older academics he has witnessed a complete revolution in daily working practices. The typing pool is gone, the mainframe has come and gone, most communication is now through e-mail and his main office support is the PC sitting on his desk.

As with most computers, sometimes things go wrong and he has to seek out the help of a computer officer to put things right. He goes to his or her office, explains the problem and the computer officer explains how

to put it right. All well and good, *up to the point when my father arrives back in his office and can no longer remember the details of the answer to his question*. Somehow the information is not presented to him in a form which is easy to assimilate. No wonder, if the conversation follows a path as described by Jackson (1999):

> *If you have a memory like mine, you probably forget your passwords on various machines quite frequently. How does your system administrator, let's call him Sam, reset your password? Our budding knowledge engineer, Ken, tries to find out.*
>
> *Sam: Well, if it's a YP password, I first log on as roon on the YP master.*
>
> *Ken: Er, what's the YP master?*
>
> *Sam: It's the diskfull machine that contains a database of network information.*
>
> *Ken: 'Diskfull' meaning - ?*
>
> *Sam: - it has the OS installed on local disk.*
>
> *Ken: Ah. (Scribbles furiously.) So you log on…*
>
> *Sam: As root. Then I edit the password datafile, remove the encrypted entry, and make the new password map.*
>
> *Ken: …password map. (Attempting humour.) What happens if you forget you password?*
>
> *Sam: On a diskfull system, I could reboot to single user mode, or I could load MINIROOT so I can edit /etc/password. Or I could reload the entire system, which I'd rather not do. Root passwords aren't usually included in YP. On a diskless client I could use the passwd command.*
>
> *Ken: Oh.*

Or consider the experience of Alan Kay (1996) as he relates his first day in graduate school:

Through a series of flukes I wound up in a graduate school at the University of Utah in the fall of 1966, "knowing nothing." […] Head whirling, I found my desk. On it was a pile of tapes and listings, and a note: "This is the ALGOL for the 1108. It doesn't work. Please make it work." The latest graduate student gets the latest dirty task.

The documentation was incomprehensible. Supposedly this was the Case-Western Reserve 1107 ALGOL – but it had been doctored to make a language called SIMULA; the documentation read like Norwegian transliterated into English, which in fact it was. There were uses of words like activity *and* process *that did not seem to coincide with normal English usage.*

Finally another graduate student and I unrolled the program listing 80 feet down the hall and crawled over it yelling discoveries to each other.

If university professors and graduate students can't make head or tail of computer systems, what chance has everybody else? Why are computer manuals and other technical documentation often so hard to understand? Of course all manner of specialists find it difficult to discuss their particular domain of expertise. Try discussing techno/house music with a teenager or rocks of the tertiary period with a geologist. Nevertheless, computer science does seem to suffer from the problem more than most. In addition, most of us can live our lives quite comfortably without knowing anything about techno/house music, but it is increasingly difficult to fall back on that attitude where computers are concerned.

I believe one reason is that many computer scientists become so heavily focussed on the linguistic, formalised aspects of computing that they lose sight of other, equally important aspects of using our own human language. The simple art of telling a story, the use of poetry as a tool to aid our faculties of memory have been forgotten. Many computer scientists talk and *write as though they were communicating information to a computer*, not to a human being. Jerome McGann (2001) discusses some of the differences between "thin" text (for the purpose of communicating information) and "thick" text (more poetic in nature). This is maybe a sensible distinction, but it seems difficult for many people to write in both genres and in many circumstances an overlap

could aid both understanding and memorability. In this sense, many computer scientists have a lot to learn from the study of literature.

We should also remember that in many non-western cultures, traditional story-telling remains a key tool for disseminating knowledge from one generation to the next. It would be unfortunate if the Western-dominated "information age" destroyed some of this heritage.

Writing Code

Is writing a computer program like writing a novel? Strangely enough, in some ways it is. Now the reader *is* to be either a computer or another computer scientist and for many, this seems to be a more comfortable situation. Thus we find a great emphasis on clarity and aesthetics. It is not enough to write program code that works well, it has to appeal to some higher sensitivities. Deitel and Deitel (1998) write:

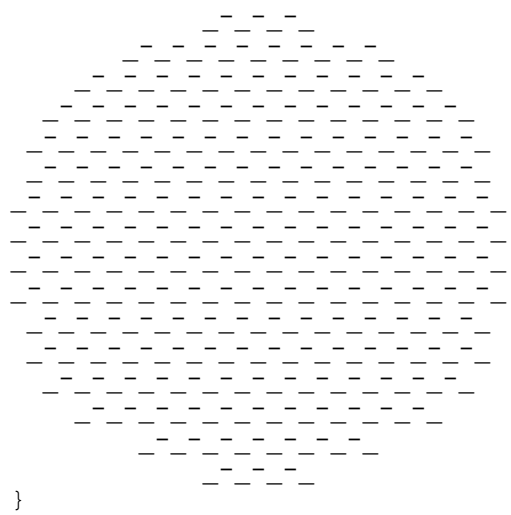
> *Welcome to ANSI/ISO Draft Standard C++! This book is by an old guy and a young guy [….] The old guy wants clarity; the young guy wants performance. The old guy appreciates elegance and beauty; the young guy wants results.*

Knuth (1997) is even stronger in his affirmation that programming is a creative, artistic experience. Note that he also specifically relates programming to poetry:

> *The process of preparing programs for a digital computer is especially attractive, not only because is can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music.*

Humour also abounds in the world of programming. There is an annual "obfuscated C code" competition in which programmers compete to write the most ridiculous programs they can think of. For example:

```
#define _ -F<00||--F-OO--;
int F=00,OO=00;
main(){F_OO();printf("%1.3f\n",4.*-F/OO/OO);}F_OO()
{
```



```
}
```

The code above was a winning entry by Brian Wesley in 1988. This program (in an extremely devious way which defies explanation to a non-technical readership) computes the value of pi by counting the number of dashes in the circle, which gives an estimate of the area. (If we assume the radius to be of value 1, the area will be equal to pi.) As the author says, "for more accuracy, write a bigger program!".

Some artistic and literary seeds therefore seem to be present in the world of computer science. A question which we should ask ourselves is whether we can build on this beginning and produce anything useful (thus proving Wilde wrong!).

Describing Algorithms

Perhaps it is whilst describing problems, and algorithms to solve them, that computer scientists come both closest to, and furthest from real literature. In some books concerning algorithms we find a richness of metaphor and a creativity of expression which outshines factual publica-

tions in many other fields. In others, the prose is, to be frank, both tortuous and poverty-stricken.

Consider the standard graph theory problem of finding the convex hull of a set of points in a two-dimensional plane. The convex hull of a set of points is the smallest polygon which surrounds all of the points, as illustrated in figure one:

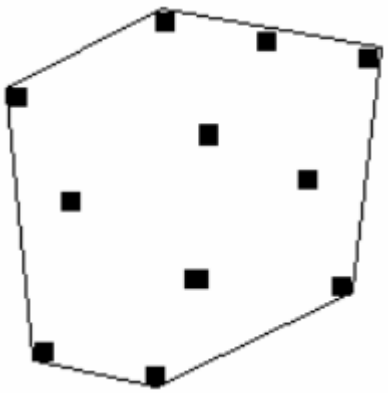


Figure 1. A convex hull

In presenting this problem, Harel (1996) chooses to dress the problem up in a zoological guise. A programmer is to spend the night in an imaginary jungle. The points are a family of fierce, programmer-eating tigers, for the moment asleep. Before the programmer can go to bed, it is necessary to construct a fence around the sleeping tigers. Naturally, the programmer wishes to build the shortest possible fence.

This simple description transforms our problem. It becomes far easier to visualise what must be done and it simplifies the task of remembering the details of both the problem and the solution.

Yet such an approach is all too rare. More typical in a computer science textbook is prose of the following character:

> *Most of our attention in this section has been devoted to transformations involving postfix operations. An algorithm to convert an infix expression into postfix scans characters from left to right, stacking and unstacking as necessary. If it were necessary to convert from infix to prefix, the infix*

string could be scanned from right to left and the appropriate symbols entered in the prefix string from right to left. Since most algebraic expressions are read from left to right, postfix is a more natural choice.

Reading one paragraph of this material is tiring on the brain. The book the above extract comes from, *Data Structures using C and C++* by Langsam et al. (1996), is 672 pages long and in a uniform style! No wonder many computer science lecturers complain that their students are reluctant to open their books.

Finding a middle ground

Despite the occasional glimpses of a closer relationship, there is a large and yawning gap between contemporary computer science and literary studies. How might we try to bring the two subjects together? How can we define a middle ground in which to meet?

My view is that a good meeting ground is what computer scientists term "pseudo-code". Rather than write large amounts of computer code directly, we encourage students to write algorithms as pseudo-code as a first step. This frees us from the harsh, rigorous formality of actually writing real code and provides a "thicker" style of writing than the code itself (although it is still pretty "thin" from a global perspective).

A simple example of pseudo-code might be:

While there are more items on my shopping list
Read next item and cross it off
Place the said item in my shopping trolley
Add the price of the item to my total bill to be paid

Since pseudo-code tells a story within a semi-formalised framework of conventions, it seems well suited for analysis in itself, using tools and techniques usually reserved for the study of poetry and literature. This is particularly pertinent in the light of the following statement by Levitin (2003): "Surprisingly, computer scientists have never agreed on a single form of pseudocode, leaving textbook authors to design their own 'dialects'."

Study of these different dialects and conventions could prove illuminating. Which explanatory devices are the most useful? Would making pseudocode "thicker" make it less or more readable?

We can also try rewriting the above example (which is somewhat dry and functional) in one or two different styles. Here is a rather quaint poetic version:

A pen gently descends death row
Kissing the entries a fond farewell
The tumbril welcomes its latest victim
How much more blood money to escape from this hell?

How about a rap-artists' version?

Ain't you got what you came for?
Check-out on the list you whore.
All the stuff goes in the cart
Count the cash you lazy tart

If nothing else, this kind of exercise is rather fun.

Conclusions

Many aspects of artificial intelligence have resisted all attempts to create a robust mathematical framework. Perhaps we have had the wrong approach? It is not altogether surprising to find that mathematics and logic are not the best tools to represent aspects of the human psyche.

Perhaps these tools are to be found in the area of literature studies. A deep relationship between literature studies and computer science is certainly a possibility. It will not be easy to formalise and this paper is only a very speculative discussion about possible lines of enquiry. A more detailed study of the historical development of the genre of computer documentation would also be a fascinating line of enquiry and is perhaps a more realistic initial line of research.

The possible benefits of carrying out such a study could be considerable, impacting on a number of important areas of computer science; artificial intelligence, code writing, language design and many others. A practical outcome of this paper I would hope for is a move to modify the educational syllabus typically followed by computer scientists to include more study of the humanities. I am convinced that wider reading and a demand to write "thicker" text on occasions would help to overcome some of the problems of communication which I have discussed.

However, I would strongly argue that we should consider exploring these ideas regardless of their possible practical applications. For as Wilde said: "Nowadays people know the price of everything, and the value of nothing."

Mark Dougherty has an MA in computer science from Cambridge University, a PhD in civil engineering from University of Leeds, is docent in traffic and transport planning at KTH and professor in computer science at Högskolan Dalarna. He has taught at university level in all of these subjects, plus mathematics, philosophy and electrical engineering. Interdisciplinary studies are a natural focus for his research work.
Email: mdo@du.se
Web page: http://www.du.se/~mdo

References

Bauhaus, 1919-1928. Eds. Herbert Bayer, Walter Gropius & Ise Gropius. Boston: Charles T. Bradford, 1952.

CHOMSKY, NOAM (1956). "Three Models for the Description of Languages." *IRE Transactions on Information Theory* 2: 113–124.

DEITEL, HARVEY M. & PAUL J. DEITEL (1998). *C++: How to Program*. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

DENNETT, DANIEL (1984): "Cognitive Wheels: The Frame Problem of AI." *Minds, Machines and Evolution: Philosophical Studies*. Ed. Christopher Hookway. Cambridge: Cambridge UP. 129-151.

DESCARTES, RENÉ (1993). *A Discourse on the Method of Rightly Conducting the Reason and Seeking Truth in the Sciences*. [1637]. Project Gutenberg. <http://www.gutenberg.net/etext93/dcart10.txt>

DREYFUS, HUBERT & STUART DREYFUS (1985). *Mind over Machine*. New York: Macmillan/The Free Press.

FLORMAN, SAMUEL (1994). *The Existential Pleasures of Engineering*. 2nd ed. New York: St Martin's Press.

HAREL, DAVID (1996). *Algorithmics: The Spirit of Computing*. 2nd ed. Reading, MA: Addison Wesley.

JACKSON, PETER (1999). *Introduction to Expert Systems*. Harlow: Addison Wesley.

JOHNSON, ERIC (1999). *Computer Programming for the Humanities in SNOBOL4*. Madison: Dakota State UP.

KAY, ALAN (1996). "The Early History of Smalltalk." *History of Programming Languages*. Eds. Thomas J. Bergin, Jr. & Richard G. Gibson, Jr. Reading, MA: Addison Wesley. 511-579.

KNUTH, DONALD (1997). *The Art of Computer Programming: Volume 1: Fundamental Algorithms*. 3rd ed. Reading, MA: Addison Wesley.

LANGSAM, YEDIDYAH, MOSHE J. AUGENSTEIN & AARON M. TENENBAUM (1996). *Data Structures Using C and C++*. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

LEVITIN, ANANY (2003). *The Design and Analysis of Algorithms*. Boston: Addison Wesley.

LUCAS, JOHN (1961). "Mind, Machines and Gödel." *Philosophy* 36: 112-127.

MCGANN JEROME J. (2001). *Radiant Textuality*. New York: Palgrave.

NIELSON, HANNE RIIS & FLEMMING NIELSON (1992). *Semantics with Applications: A Formal Introduction*. Chichester: Wiley.

TURING, ALAN (1950). "Computing Machinery and Intelligence." *Mind* 59: 433-460.

VERNE, JULES (1996). *Paris au XXe siècle*. [1994, posth.]. New York: Ballantine.

WILDE, OSCAR (1992). *The Picture of Dorian Gray*. [1890]. Ware: Wordsworth Editions.

WITTGENSTEIN, LUDWIG (1953). *Philosophical Investigations*. New York: McGraw-Hill.