On the 12th of March 2009, the UK government announced that an Information Technology (IT) system for tracking prisoners from sentence to release, NOMIS, had been abandoned. Why? Because the National Audit Office had determined that the IT system’s main aim of supporting end-to-end offender management would not be fulfilled. In addition, this unfinished IT system was two years behind schedule. Yet, the original £234 million estimate had blossomed to £690m of which the taxpayer had already put in £155 million.

Sadly, this was not a one-off, expensive, IT-system disaster. It was just the latest in a long line as the London Ambulance Service, the International Stock Exchange, the Performing Rights Society and the Child Support Agency (to name but a few) have all learned to their great cost – millions of pounds spent, years wasted and no usable IT system. Why does this keep happening?

Reports on the NOMIS debacle variously blame sloppy project management, overoptimistic planning, unclear financial responsibility and underestimation of technical complexity. The Editor of the technical magazine *Computing* was quoted as laying the blame squarely on this last issue – the proposed IT system was too complex.

Technical complexity, I shall argue, is at the heart of the problem of large IT-system development (and usage), but complexity can come in many forms, and, in this case, compounds to unavoidable unmanageability. The precise nature of this inevitable complexity is analysed and explained in this book. Conventional IT-system technology lures all parties – the technical developers, the system designers, the project managers, and the clients – into grappling with the unmanageable system behaviours that emerge. The inevitability of the consequent emergent chaos is the focus of this book – primarily its technical basis, but also its knock-on effects for project management, cost estimation and client satisfaction.

In 2005, the British Computer Society (BCS) produced a report on the problem of IT-system failure. After noting significant cost overruns on conventional engineering projects (Concorde, the Channel Tunnel and the Scottish Parliament building – none of which was abandoned, incidentally), they explicitly blame IT-system failures on management reasons, not technical reasons. The cynic might be tempted to observe that the BCS would say this wouldn’t they. In the further elaboration of their argument the BCS did, however, adduce management’s ignorance of the
technology, and inability to reduce program complexity as two of the main reasons behind the problem.

The fundamental nature of the underlying technology – programming – is the single main reason behind all of the acknowledged difficulties. Its detrimental effect derives from a pernicious melding of factors from various sources: psychological and sociological problems coupled with unfulfilled mathematical promise and an excessive demand for mastery of fine detail. It is the inevitable escalation of the resultant complexity that will be exposed.

Further evidence that technological difficulties are at the root of the problem comes from the observation that in the USA, where management practices are somewhat different but the technology is exactly the same, the situation is no better. Annual costs of IT-system error avoidance and mitigation are estimated, by the US Department of Commerce, to range from $22.2 to $59.5 billion, and this is, of course, primarily for the IT systems that do get operational. A doubling, it is said, would probably cover world-wide costs, and a further doubling is anticipated in the next decade.

It is difficult to view our computerised society as anything but an ever-growing phenomenon – both reaching out to embrace new aspects of our lives, and simply expanding current activities to provide a more comprehensive coverage. This, I’m sure, you know already. What you are unlikely to know is that all these computer systems are, and will always be, imperfect, which is a polite way to say that they contain errors and will thus go wrong – sometimes irretrievably so. All IT systems will sometimes fail. Over half of the huge cost of IT-system error avoidance and mitigation in the USA is borne by the IT-system users – they are grappling with the failures of the system. Not all IT systems are abandoned as total failures, far from it, but all IT systems are failures to some extent.

As consumers (if that’s the word) of this burgeoning technology, we really ought to know what we are getting into. One of my basic claims is that an understanding of how computer systems work (and, more importantly, don’t work) is not way beyond anyone, and that possession of a “logical mind” (whatever that may be) is largely irrelevant. This book offers an explanation of modern computer technology that is readily comprehensible to anyone who is also prepared to devote a little effort to the endeavour.

This means that I have to delve into details. How much detail, and what detail? First, what detail? I shall find no need to bring in bits, bytes, binary arithmetic, formal logic, AND gates, or any other of the similarly irksome objects that are traditionally part of the initiation rights for the computationally naïve. A sufficiently detailed understanding of computer technology is necessary, but it can be gained much less painfully, I believe. I am not hoping to turn you all into programmers, but I shall co-opt you into a small program-development exercise in order to provide the necessary depth in a simple and straightforward explanation of this technology.

How much programming detail? Some programming, involving just seven precise instructions for controlling a machine, is necessary. We gloss over this level of detail at our peril, I believe. Let me illustrate this.
The generally excellent book *Emergence* (Penguin, 2001), by Steven Johnson, is weakest when the author switches from ants and cities to the achievements and future implications of computer technology. Johnson makes much of a neat example of a non-traditional approach to sorting lists of numbers into ascending order. For Hillis, his sorting algorithm was a small boost to the case for his invention of a massively parallel computer. As a technologist, Hillis knew that he had done no more than apply a well-known technology (adaptive hill-climbing en masse, aka simulated evolution) to a special case of number sorting using an innovative procedure to avoid dead-ends. Johnson, a non-technologist, unacquainted with the long history of (and well-known problems associated with) adaptive hill-climbing procedures, misses the details and jumps to the conclusion that Hillis’ demonstration has opened a new route to the stars. Hillis, however, was under no such illusion; he knew that if his little demonstration had pushed away from the earth-bound morass of conventional programming, it had only shown how to climb one very small and very specific type of tree. It is technical details that lay bare the non-technologist’s misconception.

But exactly how much detail do you need to avoid such pitfalls? Whatever level I aim for, it will be too much for some readers, and too little for others. So, I have introduced a variety of strategies for making this a variable level-of-detail book.

Firstly, I have picked up on the advice that Mack of Palace Flophouse fame gave John Steinbeck about how best to make *Sweet Thursday* an improvement on *Cannery Row*.

I ain’t never been satisfied with that book *Cannery Row*. I would of went about it different ... Sometimes I want a book to break loose with a bunch of hooptedoodle ... But I wish it was set aside so I don’t have to read it ... Then I can skip it if I want to, or maybe go back to it after I know how the story come out.

So, in the chapters where I feel compelled to break loose with a bunch of hooptedoodle (although Mack and I attach somewhat different meanings to this word), I’ve set them aside. These diversionary excursions can then be skipped, without real loss, pain or guilt, by the reader who feels so inclined. There is not even a need to return to them when you know “how the story come out”, but feel free to do so, if you wish.

Secondly, each chapter concludes with a bullet-point list of the specific technical points that it has championed. The reader thus has various options with respect to every chapter: to accept the bullet-points on trust and forego the preceding technical arguments (with the opportunity to go back at some later stage if so motivated); to accept the bullet points as well-known and so just skip or skim the chapter; to accept the bullet points as foci of particular importance, interest or error, and read the associated chapter closely.

Thirdly, the book’s penultimate chapter is explicitly structured to permit the browsing of detail at whatever depth the reader feels comfortable with, or feels is necessary. This is explained below. Lastly, each chapter is associated with detailed endnotes that, once more, can be selectively read, if so motivated, or totally ignored by Salinger’s “amateur reader who just reads and runs”. In addition, the endnotes
often provide a hook into the published sources where the interested reader can find even more detail.

The book is divided into four parts: Part I is designed to be a relatively painless but sufficiently-detailed attempt to provide the non-computer literate reader with the necessary basics of modern computer technology. The computer literate might also skim it with profit, because it provides the basis for many subsequent examples. This is not a ‘How to program’ book; it delves into the essential nature of programming just as far as is necessary to properly appreciate both the opportunities and difficulties involved. Becoming a programmer and appreciating the difficulties (as well as the attractions) of the craft are two different things. It is this latter competence I am aiming to convey.

Part II addresses the consequent system-level problems. It begins by examining computer addiction, or less dramatically, computer dependency which is a widely acknowledged syndrome within the community of computer enthusiasts. Investigations of this phenomenon have tended to concentrate on the detrimental effects it has on the dependents themselves, or on their nearest and dearest. These relatively few and extreme cases are the tip of an iceberg, the vast bulk of which (as is the way with these chill leviathans) is largely invisible, yet capable of causing great damage. Importantly, this damage is not so much to the enthusiasts or technologists themselves, but to the rest of us – the IT-system-dependent society. So, as more properly befits my expertise, I shall be probing in a new direction. I shall direct your attention away from the adverse effects on computer people. Instead we will look towards the repercussions for IT-system development and so to the consequences for IT-system users – consequences that we must all endure.

Part III adds a more positive line of argument: innovative approaches to tackling the problems are described. The first three are potential patches for current practice; the last two are more speculative, but hold some hope for major renovation of current technological practice. Neither is a reiteration of the popular suggestion that from bottom-up rule-driven anarchy something wonderful will emerge. This is the fast (and easy) track to emergent chaos. Biological systems, which provide a framework for one innovation, are both complex and reliable. How is it done – incremental growth and redundancy? The Internet is an (accidental) example of the ‘organic growth’ approach to IT-systems development. It may be that the organic world is hiding a completely new framework that computer technologists might usefully adopt, adapt, and exploit. The second innovative technology might be termed a digital revolution, but a reversal of the ongoing revolution in UK TV broadcasting – the proposal offered is to abandon total reliance on digital and move to admit some analogue.

In Part IV, the main chapter provides an overall summary but one that is designed to provide an alternative entry point into the book for the eclectic reader. Each of the earlier chapters concludes with a bullet-point list summarizing its contribution to the overall case I am making. This penultimate chapter repeats these summary lists, and further summarizes them. My idea is that an efficient read (indeed a strategy for deciding what to read in detail) is to look at this chapter and see how “the story come[s] out.” This super summary, which is linked back to the individual
chapter summaries, should quickly clarify where what I have to say fits in your personal spectrum of interest, from thought-provoking and unexpected to simple and obvious. Then from this personal positioning, it should be possible to decide which chapters might be read with profit and which might be skimmed or even skipped (due to either a pre-existent competence, or a reluctance to engage fully with technical detail on a first read). The final chapter presents the totality of my case in soundbites whose justification is all of the foregoing.

All IT systems harbour errors that nobody knows about. IT systems control much of your life (and it’s only going to increase). Now you should feel suitably motivated to read this book in order to learn why this highly disturbing state of affairs exists.
The Seductive Computer
Why IT Systems Always Fail
Partridge, D.
2011, XVI, 323 p. 56 illus., 6 illus. in color., Softcover