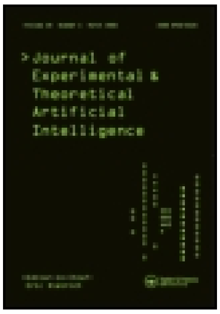

Journal of Experimental & Theoretical Artificial Intelligence
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/teta20

What humans can't do: a review of Derek Partridge, *The Seductive Computer* by Derek Partridge

Chris Fields

Available online: 13 Jan 2012

Journal of Experimental & Theoretical Artificial Intelligence
2012, 1–4, iFirst

BOOK REVIEW

What humans can't do: a review of Derek Partridge, *The Seductive Computer* by Derek Partridge, London, Springer-Verlag London Limited, 2010, 339 pp., US$59.95 (paperback), ISBN 978-1849964975

With just a glance at the title, one might imagine that *The Seductive Computer* is yet another rail against the ambitions of artificial intelligence, a twenty-first century update of Hubert Dreyfus' 1972 classic, *What Computers Can't Do*. Derek Partridge's slim volume is, however, much more interesting than that. It concerns what *humans* can't do, and in particular, the causes and consequences of human inabilities to understand the information technology (IT) systems that increasingly operate the human world. This is a topic that Partridge, who has spent his career researching and teaching software engineering, understands deeply and worries about not just for theoretical but also for utterly practical reasons.

Why should philosophers, cognitive scientists and AI researchers care about, much less read, a critique of software engineering practice written by a computer scientist? The short answer is that IT system development is an imaginative process, and IT system implementation enables the testing of human imagination for self-consistency in a way that few, if any, other disciplines can match. At bottom, self-consistency is a philosophical issue, and IT system development is, from this perspective, a grand exercise in experimental philosophy, a systematic if perhaps unintentional delving into the relationships between certainty and delusion, and between human agency and the world's raw insistence on evolving according to its own principles. It is, moreover, experimental philosophy done under the best possible conditions, experimental philosophy in which all statements must, eventually, be expressed in an uncompromising formal language and rigorously examined for grammatical correctness, self-consistency, and consistency with all statements made previously. It is experimental philosophy done as if logical positivism had won the day, and imposed absolute clarity of expression across the board. Indeed, the operational success of an IT system over an extended period of use that probes its behaviour in response to a wide variety of inputs is arguably the best operational definition of conceptual self-consistency that is currently available.

All IT systems begin with a simple statement of desired behaviour, from printing out 'Hello World' to flying an airliner or running the global financial system. They proceed from desire to specification: under conditions X, do Y. As the level of detail with which the desired behaviour is described increases, the specifications inevitably get defensive: What happens if the user hits 'Alt' instead of 'Shift' when typing an upper-case letter? Do X. What happens if two users issue commands to change a database entry at exactly the same time? Do Y. What happens if the system administrator logs on from an IP address that has never been seen before? Do Z. Eventually the specifications – often tens of thousands of pages of specifications – are detailed enough to be expressed in formal, executable code. Then they can be put to the test: does a machine executing the code actually perform the behaviour that was specified, and is the behaviour that was, at the end of the day, actually specified the behaviour that was originally desired? In principle,

ISSN 0952–813X print/ISSN 1362–3079 online
http://dx.doi.org/10.1080/0952813X.2011.653101
http://www.tandfonline.com

these questions seem straightforward. *The Seductive Computer* goes to some length to demonstrate that in actual practice they are not.

Let us put this into philosophical language. An IT system specification is a long list of conditions and consequences, a precise description of a conceivable, indeed an actually conceived world. This imagined world is meant to be a place where certain desired behaviours occur. Implementing an IT system specification in a formal programming language, and running the resulting code on a machine, is asking two questions that are otherwise seldom asked. First, is this conceivable world *in fact* logically possible, or does it contain contradictions? A computer told 'if X, do Y' on, say line 137,253 of a program, and 'if X, do Z' on line 791,675 will, if run long enough with sufficiently diverse inputs, eventually find the contradiction and either behave erratically or crash. Second, is the world that was actually conceived the one that the designers intended to conceive, or are there hidden contradictions between desired behaviour and actually specified and hence implemented behaviour? Implementation tests the 'possible worlds' defined by IT system specification for contradictions of both kinds, and the judgement is often unexpected and harsh. Word processors crash without warning, databases send insurance payments to the wrong bank accounts, stock markets and airliners spin out of control. Work is lost, money is lost, people die.

The Seductive Computer illuminates the process of specifying, coding and testing IT systems with a probing and unsparing light. It offers cogent and compelling arguments that, except in trivial cases, the 'possible worlds' conceived in software specifications are inevitably and inescapably flawed. Partridge minces no words here: 'IT systems all fail: sometimes immediately and spectacularly, sometimes unobtrusively just once in a while, and sometimes in any combination of these two extremes' (p. 9). He takes apart the comforting formalist argument that computer programs, being well-defined mathematical objects, can in principle be proved correct. Right, says Partridge, but in practice only if they are trivial enough to be hand-executed by us. Given a 10,000-page specification and a 500,000-line program, no human being will be able to prove that the program exactly satisfies the specification. Only another program could keep track of all of the inferences and cross-checking of cases that would be required, and in the absence of a proof to the contrary (none have ever been given), using one complicated piece of software to prove the correctness of another complicated piece of software is circular. Without the possibility *in practice* of formal proofs of correctness, self-consistency of both specifications and the programs written to implement them can only be assured by good engineering: well thought-out high-level designs, incremental development, clean, well-structured code, thorough documentation and 'best practices' guidelines to enforce all of these. All well and good, says Partridge, but insufficient even so.

Partridge offers four mutually-reinforcing reasons why non-trivial software systems, even those developed with the best of best practices, are impossible for human beings to fully understand, and hence potential sources of emergent chaos that are more akin to features of the natural world than to traditional artefacts. The first is that every detail counts: programming languages are formal systems, and formal systems are unforgiving: mistyping ',' for ';' can utterly change the semantics of an expression and hence the behaviour of a computer executing it. The second is non-locality: real-world problems, like quantum systems, do not factor cleanly into independent bits that can be treated with fully-modularised chunks of code. This is a deep point, one that deserves a book on its own, and one that Partridge could have spent more time on. However, it is made evident by political, economic, and even weather news every day, and hence is something that must, even if unpalatable, be assumed in any domain without a strong demonstration otherwise. Third is the vague and slippery nature of specifications, a consequence of the human reliance on shared inexplicit knowledge that Dreyfus and many others have emphasised. This too is a deep point, one closely related to the Frame Problem, which Partridge does not mention. The fourth reason lends the title to this book: it is the seductive simplicity of 'try it and see', the universal strategy of time-pressured software developers struggling to find some solution, any solution, that will get the machine to do what he or she wants. All of these, of course, look like mere technical issues, problems to be solved by more careful editing, better high-level design, thorough, critical and detailed writing of specifications and more disciplined personnel management. Such 'solutions' are routinely advanced, singly or in various combinations, by academic, industrial and governmental review panels every time a major IT project goes wrong. The genius of Partridge's book is to attack this 'mere technical issues' stance head-on, adducing cognitive, emotional and organisational data to show that the information processing requirements of large-scale IT systems exceed the capabilities that human beings can bring to bear, often by many orders of magnitude. This is, of course, in one sense obvious: computers are not merely time-saving conveniences in our twenty-first-century technological culture, but rather do jobs that we humans are not capable of doing. The result of our inability to fully understand the IT systems that we ourselves have constructed, however, is that we cannot predict with certainty how they will behave, and we cannot fix them with complete confidence when they misbehave. All we can do is tweak, patch, restart and hope for the best.

Partridge is a technologist, and in some sense an optimist. He places the primary blame for the incomprehensibility of software not on people, but on the dominant digital technology of practical computing systems, and he ends *The Seductive Computer* with suggestions that a more evolutionary approach to IT system development coupled with the judicious use of at least simulated analogue technology – neural networks and statistical methods reminiscent of 'genetic programming' – might ameliorate the situation. This hopeful finale is the weakest part of this book, as it attempts to dodge the previous arguments by hiding the problems of incomplete specifications and non-local interactions between 'modules' in the language of machine learning. As Partridge points out, however, the code that results from machine-learning methods or other forms of statistical optimisation is typically not understandable by humans: we may know that it works on some set of cases, but have no idea how it will generalise outside the boundaries of the training data. Such unpredictability in the face of actual reality is the very problem that the main arguments of *The Seductive Computer* are about. It is of no help to discover that the training set for a neural network must be re-designed by watching the network classify friendly aircraft as hostile and direct automated weapons to shoot them down, or to realise that one's statistical models are not representative after all only when one's prized bond-trading program has gambled away the entire pension fund. Programming with continuous methods is useful, but only in domains in which small details do not much matter, i.e. more-or-less smoothly varying domains in which human understanding of what the program is doing is not nearly so much of an issue.

Philosophers are typically not technologists, and they are often not optimists, but they are imaginers of worlds who are generally concerned that such conceived worlds are possible. All of science, from the most refined theory to the most straightforward experimental design, requires such imaginative ability. The lesson of *The Seductive Computer* for philosophy and for science in general is that possible worlds, unless they are trivial, are hard to conceive. According to Partridge, when we say 'given conditions X...' we generally don't know what we are talking about, even under the strictest of logical-positivist restrictions on language. When we say 'imagine a world just like this one but...' we are quite likely imagining an impossible world, a world with subtle contradictions that, given time, will generate unpredictable emergent chaos. What is most troubling about this predicament is that we can never, in fact, find out whether our imagined worlds are possible or not except by pursuing them until they screech to a halt, veer out of control or explode. Philosophy itself is thus no better than software engineering: an imaginative discipline that works well enough most of the time, but can fail, either subtly or spectacularly, at any moment.

The Seductive Computer is written for a general educated audience, and builds its arguments from both well-documented historical examples and detailed analyses of an apparently simple programming task – specifying in implementable detail the desired behaviour of a robotic grocery shopper – that throws out one unexpected failure mode after another throughout the book. Every chapter has a bullet-list summary; the last two chapters collect these summaries, and then re-summarise them to assure that the major points are not lost. Four 'hooptedoodle' chapters provide detailed asides or explore relevant secondary topics, such as the likelihood of ever 'understanding' the human genome. There are plentiful pointers towards supporting data and further reading, and an extensive glossary. The multiple themes being simultaneously treated, from the unrealisability of formal proofs of correctness to the odd psychology of the 'happy hacker' give this book a somewhat choppy style, and it exemplifies its own main point by containing a noticeable number of sentences from which key words or possibly entire phrases seem to be missing. *The Seductive Computer* is, however, well worth reading, if only for the *schadenfreude* of realising that practitioners of the high-tech arts have conceptual problems at least as deep as philosophy's own.

Chris Fields
Email: fieldsres@gmail.com