What makes an expert?

Ian Thompson

May 4, 2017
Department of Mathematical Sciences,
University of Liverpool

In the early days of computing, one would sometimes encounter individuals who knew *everything* about a particular device or piece of software. Single programmers wrote entire applications or games, and some could debug their work by looking directly at a core dump (a printout of the numbers stored in the computer's memory). Some even managed to take computers beyond their specifications by exploiting design loopholes that the manufacturers hadn't foreseen or intended. It would be fair to classify such individuals as 'experts'.

Fast forward twenty five years, and the picture is far less clear. The complexity of computers and software has grown to such an extent that even relatively small smartphone applications are created by teams of developers, and nobody understands every aspect of a CPU chip, much less an entire PC or tablet. Who now should be classified as an expert? One possibility is that an expert is a person who may sometimes need to look up the details of a rarely used command or feature, but who is never confused or frustrated by the behaviour of the system or software in question (except where there is a bug), and never needs help from anyone, except perhaps on rare occasions from its creators.

This rather stringent definition makes me an expert in only two areas of computing: the Fortran programming language, and the mathematical computation system Maple. An argument could be made for the typesetting system LaTeX, but whilst this has a large number of expert users, there is also a much smaller group of more exalted experts, who maintain the system and develop new packages and extensions. It would be fair to say that I fall into the first category, but not the second.[1]

How does one achieve expert status? Some software actively prevents this, by hiding its workings to such an extent that fully understanding its behaviour is impossible. Where it is possible to gain expert status, I have experienced two very different routes, both starting during my time as a research student, when it became clear that Fortran and Maple would be useful in my work. There were several parallels. I knew a little about both, having used them for basic tasks as an undergraduate. However, working out why things went wrong and how to fix them was time-consuming and unrewarding, since it often relied on magic recipies obtained from unreliable sources, and in many cases I didn't really understand why

[1] One can always identify an expert LaTeX *user* by the fact that they know enough to realise that they are not a 'real' expert.

these worked, any more than I understood why my own attempts had not. I realised then that *knowing a little* was at the root of these problems. Partial knowledge, supplemented by contradictory, outdated and even downright bad advice from websites and well-meaning individuals (some of whom invariably labour under false pretences of their own expert status) is not an efficient way to approach scientific computing. In fact it's just a recipe for more frustration. In the case of Fortran, fixing this turned out to be easy, because there are lots of good books on the subject. Reading one of these eliminated all of my problems with the language at a stroke. I can't claim that I remembered every command and its syntax, nor do I know them all now. This is hardly surprising — the Fortran Language Standard (a very terse document that sets out everything the language provides) now extends to more than 600 pages. Instead, the book provided a general picture of how things work in Fortran, and showed the right way to go about tackling a problem. This investment in time has since paid itself back hundreds of times over.

The route to expert status in Maple was far more painful. Its own help pages give a very comprehensive description of individual commands, but they are intended as a reference guide, and if it's possible to become an expert using these alone, then the order in which to read them is certainly not obvious. To make matters worse, none of the books in the university library seemed likely to meet my needs. Most were too basic to be useful, and others focused on particular applications. None seemed likely to give me the general picture — the feel for how things work — that would change Maple from a maddening source of frustration to a time-saving resource. Years of anger, failed attempts at solving problems (usually followed by a reluctant return to hand calculations) only came to an end after I had taught Maple to students on three different courses. Investigating the problems they experienced gave me new determination to properly understand the system, and eventually the few remaining gaps were filled in by biting the bullet and reading the Maple Programming Guide. This is a highly technical document aimed at advanced users; it sets out the capabilities of Maple in a manner analogous to the Fortran Language Standard. It is no longer available as a standalone document (recent versions are accessed through Maple's help system), but the pdf version produced in 2013 extends to over 660 pages. It certainly isn't an easy read, and it's not something that should be recommended to anyone who doesn't have a degree in computer science, or a lot of experience with programming language specifications. Students now started to ask how I came to know so much about Maple, and whether there was a book that would teach them the same. I would answer that no such book existed, and that Maple could only be learned through sheer determination; some variation on the theme of 'you bang your head against it until it breaks, or your head does'. After years of hoping for a book to appear, I decided to write one; after all, if you want something doing, do it yourself. The project soon began to evolve as I tried to set down everything that the majority of Maple users need to know. I've always hated books that skirt around important but difficult topics, so where before I might have used a dirty trick to circumnavigate a problem, now I felt compelled to research exactly what was going on, and to try to explain it in a simple, concise way. When the first draft was complete, I approached Cambridge University Press (CUP). The editor arranged for reviews by four anonymous referees,[2] and by Maplesoft's own programming team. This led to several major improvements. My colleague, Dr Martyn Hughes also deserves a mention for his efforts in reading and commenting on four different

[2]Actually not wholly anonymous; the name of the referee who made the most suggestions for improvements was left in one of the documents sent to me by CUP, so a grateful mention can be made here of Michael Monagan, one of the original creators of Maple.

drafts. Meanwhile, Maplesoft continued to release new editions of their software, and the drafts had to be revised to keep up with these. The cover was created by one of CUP's designers, with instructions that it should not look too 'treeish' — one might be surprised by the number of books about Maple syrup, and it would be a shame for Understanding Maple to be mixed up with these by potential readers browsing the internet. Then there were the minor details: how wide should the pages be? What font should be used? Should disk be spelled with a 'c' or a 'k'? Could quotes from other sources be used without the threat of legal action over copyright infringement? One rights holder laughably tried to charge $200 for a fragment of text from *LaTeX A Document Preparation System* by Leslie Lamport. Needless to say, no greenbacks were forthcoming.

The resulting book is concise, with all the key concepts needed to gain an understanding of Maple, alongside numerous examples, packed into a mere 228 pages. It gives new users a solid introduction, and doesn't avoid difficult topics. It isn't perfect (in fact I have already started to list revisions that will be made if a second edition is published in the future[3]) but I've seen very few problems that can't be solved with the material it contains. Will it create new experts? Only time will tell. At the very least, I would certainly like to think that it will lead to a reduction in the number of computers smashed in frustration.

[3] pcwww.liv.ac.uk/ itho17/understanding_maple