Winning success from Babbage's failure

'Software factories' are set to revolutionise the complex and thorny business of building effective packages rapidly, efficiently and on budget.

Alan Woodward

The year 1991 saw the successful conclusion of one of the most remarkable enterprises in the history of invention. A team of engineers at the London Science Museum managed to complete the construction of a cogwheel computer that had been designed, but never built 170 years earlier by the Victorian scientist and mathematician, Charles Babbage.

During his lifetime, Babbage insisted that his cogwheel brain, which he named the Difference Engine, would be an ultra-reliable cogwheel calculator for printing mathematical tables devoid of any risk of human error. His efforts to build it, however, met largely with ridicule, and Babbage died a disappointed man in 1871.

Why couldn't Babbage build a Difference Engine in his lifetime? Today, with the benefit of hindsight, we know that the biggest problem that stymied him was the lack of a precision metal industry. It wasn't that the overall design of his machine was faulty, or that cogwheels can't be computer components. The successful modern initiative to build a Difference Engine deliberately made use of components that were no more precisely engineered than Babbage himself could have produced back in the 19th century.

However, the 4,000 parts - most of them cogwheels needing to be fashioned to be as alike as possible - were manufactured for the modern build using contemporary industrial manufacturing processes which ensured consistent high quality components to a standard pattern. This was the result of the difference between the small-scale 'craftsman' approach for building precision components that Babbage was obliged to adopt, and the modern 'industrial' method used by the Science Museum's engineers.

Development is slow, expensive and error-prone

In the important and influential business of computer software development - a business which, in a very real sense, has made the modern world possible - the craftsman-like way of doing things has prevailed for a long time, but that may be coming to an end, to be replaced by an industrial way of designing software that is faster, less risky, less expensive and more efficient.

There is widespread agreement throughout the computer software business today that software development as it is currently practiced is in a bad way. As Jack Greenfield, an influential software architect at Microsoft points out: "Software development... is slow, expensive and error-prone, often yielding products with large numbers of defects, causing serious problems of usability, reliability, performance, security and other qualities of service."

Greenfield went on to quote research from the computer industry research house, the Standish Group, which states that in the US businesses spend about $250 billion annually on the software development of about 175,000 projects.

The research also indicates that only about 16 per cent of these projects finish on schedule and within budget, while another 31 per cent are, on average, cancelled each year, mainly due to quality problems, with overall losses of about $81 billion. A further 53 per cent exceed their budget by an astonishing average of 189 per cent, incurring a total loss of about $59 billion. The same research suggests that even those projects which do get completed deliver an average of only about 42 per cent of the originally planned features.

These figures are disconcerting, to put it mildly. And certainly, anyone with practical experience today of being involved in software development, whether as a developer or as a client, is likely to testify to the sheer difficulty and stress of the convoluted progress, or lack of it, which these projects undergo.

As well as the problems unearthed by the Standish Group research, there is also the brutal fact that software development is extremely labour-intensive. Indeed, Jack Greenfield suggests that software development is consuming far more human capital than we expect of a modern industry. The whole point of an 'industry' is that it brings a concerted and methodical approach to the task of creating something or putting raw materials through a particular process to achieve a final result.

But in so many cases, software development is *not* systematic. Instead, it is a strangely haphazard matter, involving methods, approaches and outputs that bear far more resemblance to the brilliant but ill-equipped genius Babbage, painstakingly struggling with a few hired craftsmen to manufacture precisely-made cogwheels, than anything resembling a modern industry. No wonder the track record of success of modern software development is so, well, unsuccessful.

'The ultimate aim would be to remove the 'crafts' element from routine development and confine it to leading-edge initiatives'

However, there is no denying that, despite the shortcomings of the software development business, the products of software development obviously do provide significant value to the organisations that commission them, and the users who benefit from them.

But does this mean that we should be complacent about the effectiveness of modern software development techniques? Certainly not. Instead, all it means is that despite the ever-present practical, logistical and financial problems relating to software development today, computer software remains so much valued that, by and large, the organisations paying for development projects are willing to suffer large risks and losses in order to obtain the benefits.

Can things get better?

Object-orientated programming (OOP) was supposed to introduce a new way of managing software development by basing programs around certain 'objects' - a bundle of features associated with a particular application - that could be regularly re-used as and when required to facilitate rapid software development. But whilst OOP was widely used, it didn't fulfil its wider promise; it was more a question of *oops!* because developers didn't find the objects as helpful as the original author initially hoped would be the case. Most developers concluded that the level of modification and customisation of the objects necessary was simply too great for the objects to be useful; so they wound up building their software from scratch instead. Albeit using OOP techniques.

More progress has been achieved in recent years from new kinds of tools which, rather than seeking to revolutionise software development, aim instead to make developers more productive. A particularly useful tool here has been application development frameworks (ADFs). These are a form of open-to-all software packages that facilitate rapid customisation of the software for a particular application to the precise requirements of the organisation in question. But even ADFs are really still a set of generic building blocks, with the objective of being used across a wide variety of applications.

Today, the reason why software development is such a laborious, potentially risky and fraught matter is that the way in which software is

designed means that opportunities to re-use code (ie pre-written programs) are much rarer than one might imagine. Software development mainly takes place using generic programming languages that render every project essentially a bespoke one. There has been limited infrastructure and markets that have encouraged developers to supply potentially re-useable programming components. But maybe re-use of pre-packaged functionality wasn't the way to do it after all and a different perspective was required.

'Software development is consuming far more human capital than we expect of a modern industry'

That different perspective appears to be a concept known as the domain-specific language (DSL). A DSL is a special kind of language designed to be used for a particular application or solution. It is not a new concept to computer scientists, but it is one that is now entering the mainstream with the launch of tools that support this type of language. As one might imagine, a DSL will be all the more useful the more tightly the particular application or solution to which it caters is focused, because this will tend to increase the likelihood that another programmer, building a solution to a similar problem, will be able to use the same features of the language to specify what it is s/he wants the computer to do.

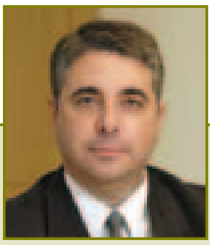
In fact, some DSLs have relatively wide domain remits such as retail banking, while others have narrow remits - operation of a retail banking ATM network. Incidentally, Microsoft has recently developed a special kind of DSL whose particular domain is helping developers to design other DSLs! In many ways this is the type of catalyst that will see DSLs enter the mainstream because it will now be possible for organisations (or their suppliers) to define languages aimed specifically at providing solutions for problems within their business domain.

The software factory

DSLs are at present in their infancy, but already they are creating the possibility of the development of software 'factory'. It is not some new form of business, or a special type of organisation. In essence, it is a set of tools, a development environment if you will, that is optimised for building solutions in a particular business domain. Hence, a software factory could be used by a traditional software supplier (assuming he adopts the new tools and techniques) or an organisation's own internal IT department if they are developing applications for that organisation.

A software factory provides solutions not through using a pre-written code, or following best practice for a particular type of solution (although these can be a part of it) but through the use of a programming language (a DSL) that is inherently designed to write computerised solutions to particular types of problems.

The ultimate aim for software factories would be to remove much of the small-scale 'crafts' element from routine development and confine such craftsmanship to the really sharp end of the most demanding, leading-edge initiatives. This will allow development to become less expensive, lower risk and far more reliable, while still giving organisations all the scope they need to establish and maintain a competitive advantage.



Alan Woodward is chief technology officer at the business and information technology consultancy Charteris plc. Telephone 0207 600 9199 or e-mail alan.woodward@charteris.com www.charteris.com