Article from the web-site of SCL (the Society for Computers and Law)

Towards a New IT Model Contract: Changing the Pricing Method

Richard Stephens and Charles Symons offer some original and challenging thoughts on one of the basic aspects of IT contracts.

Over the years, there has been much written (and debated) about the typical problems encountered with IT development projects - and it is not long before such terms as 'scope creep' start to be used. Of course, the discussion then moves on to what can be put right: some say we need a different kind of contract, others that we need a wholesale shift to agile methods as against waterfall methods and others again insist on tighter management and governance.

Perhaps the elephant in the room is that the charging method is all wrong - whatever development or management method is adopted. This article is intended to demonstrate how a different approach could help in dealing with existing problems: perhaps it would be too bold a statement to say that all problems could be resolved at a stroke, but perhaps this is a very significant start.

Illustration of the problem

There seems no better place to start than with the case of *De Beers v Atos Origin* [2010] EWHC 3276 (TCC). There are not that many cases in the law reports on failed development projects and this one seems to point up the problems well. In fact, much of the factual background of the case will seem very familiar to those involved in this area.

Briefly, De Beers, the well-known diamond mining company, tendered for a new software system to handle its operation. Atos was the successful tenderer, and the parties started on an 'Initiation and Analysis Phase' to let Atos investigate and analyse the business requirements so as to provide a fixed price. The judge found that this was a high-level exercise and did not turn up the depth of detail the project later required.

However, following that initial phase, the parties entered into a fixed price contract providing for a fixed completion date. It did not go well. De Beers was unhappy when Atos announced that it could not complete on the agreed date and that it needed a large injection of cash. As against which, the evidence showed that Atos' management was deeply concerned about the cost overruns on the project. Atos claimed that De Beers' staff had failed to cooperate in the work of requirements analysis (which the judge partially upheld). Part way through the project, Atos seems to have moved from agile to waterfall in an attempt to tie down requirements.

The case shows up many familiar problems.

Description of the problem

The essential problem as shown in *De Beers* is this: the parties thought they had agreed on what they had to do (produce a working system to manage a diamond business); they agreed

on the budget available (a fixed price) and they were agreed on when it should be completed by. What they could not agree on was how to tailor the effort available to meet the requirements and both the budget and the date. Crucially, Atos had not captured enough detail to be able to estimate the project accurately. Atos tried both agile and waterfall methods, but neither seems to have worked.

As shown in *De Beers*, the fixed price and fixed completion date gave De Beers a false sense of security. On the other hand, if they had opted for time and materials (which many agile contracts do), it could have been far worse for De Beers, because the customer in those cases has no visibility of value for money.

Although this particular case went against Atos, overall it is the customers who tend to lose out - witness the fact that major software suppliers' profit margins are generally healthy and relatively few of them actually go out of business. This should be seen against industry statistics which show an alarming rate of failures and overruns for IT projects. How can it be that an industry which under-delivers to this extent still seems to be so profitable? How can such an industry be the lynchpin for the future of the world's economy?

What's wrong with fixed price contracts?

Fixed price software contracts are often signed off early in the development cycle when detailed requirements are still very uncertain: *De Beers* is a good example of this. The lack of detail may make the software to be delivered appear to be smaller than is needed in reality and harder for the supplier to estimate. To allow for inevitable scope creep, the supplier has to add a contingency to its basic costing, but has to weigh this against the risk of losing the contract by bidding too high. Inevitably the customer must make changes in its original requirements which get charged as extras and the result is cost and time overruns on the customer's original budget. Another risk for the customer with this type of contract is that the supplier bids low for the development work in order to secure the follow-on maintenance and enhancement work after the system goes live, which will be more profitable.

What's wrong with Time and Materials (T&M) contracts?

The response of the software industry to the difficulties of specifying requirements in waterfall projects was to introduce agile methods. These have proven very attractive, especially for smaller projects and where designing the human/computer interface is critical. However, this approach usually leads the parties to adopt a T&M contract. Many projects, on the other hand, especially (but by no means uniquely) in the public sector, are large and involve business system change. In these cases in particular, agile + T&M has no way to help the customer to establish an early total cost estimate, nor to control the final cost. The other point worth making is that there is evidence that agile projects producing code at speed with limited documentation can result in systems which are hard to maintain long-term, leading to a higher total cost of ownership.

Towards a solution: unit pricing

Imagine you needed some unleaded petrol (very precisely defined) and your nearest garage was advertising 'petrol £1.00 today'. Sounds good, but then you realise that you have no idea of how much petrol you would get for £1. Unimpressed, you try the next garage. Their pumps

serve petrol for £20/minute. The problem here is that you have no idea how fast the pump delivers petrol, so again you have no idea about value for money.

Unlikely scenarios? Not at all: that's how billions of pounds are spent annually on software in legally-binding contracts. All software is delivered in measurable units such as lines of program code or some other countable unit, but very few software customers actually measure *how much* software they get, so do not have the slightest idea of the productivity of the supplier and therefore of the real value-for-money. This applies whether they are buying software in a fixed-price contract (the equivalent of 'petrol is £1 today'), or in a T&M contract (the equivalent of buying petrol on a price/per minute).

The one area where software customers and suppliers quite often measure the quantity of software delivered is in the domain of real-time, and mainly defence systems. However, the units used, counts of 'source lines of code' (SLOC), are a poor measure of software size . There are no proper standards, the measure is programming-language dependent, and you cannot estimate the software size in SLOC until a project has progressed into detailed design, too late for many contracting needs. More recently, agile methods have introduced 'Story Points' as a method for sizing software increments but these too are ill-defined and no basis for contracting.

Over 35 years ago, Allan Albrecht of IBM developed a method for measuring a size of software from its 'functional requirements', ie what the software must do. It was a brilliant piece of lateral thinking. The method, known as 'Function Points Analysis' (FPA), was particularly applicable for measuring business application software. It produced software sizes in units of 'function points' (FP) that are independent of the technology to be used for the software development and could be estimated early in the life of a project from its emerging requirements. The method was quickly standardised, and performance data were collected on completed projects. For example, development productivity = (delivered FP)/(actual work-hours). For the first time, software customers had a means of measuring the performance of their suppliers, whether internal or external. The measurements could be used to guide performance improvement and for estimating the effort for future projects from statements of requirements.

In the UK, around 1990, the Central Computing and Telecommunications Agency introduced a variant of the IBM method (known as 'MkII FPA') to the public-sector, where it became widely used. Unfortunately, politics intervened in the form of long-term outsourcing contracts. Along with all the development capacity, expertise in measuring performance was literally handed over to the suppliers (like asking children to mark their own homework), and lost by most of the customers.

In Australia, around the year 2000, the Government of the State of Victoria introduced a process to contract for software on a 'price-per-FP' basis. The process starts with the customer issuing an ITT to suppliers, accompanied by an outline statement of requirements. The ITT requires bidders to estimate the total size of the software to be delivered and to bid a fixed unit-price per FP. The supplier is selected on the basis of the unit-price and all the other usual parameters.

As the project progresses, the total size may vary as the customer adds or changes requirements but the unit-price remains constant. On hand-over, the delivered FP size is measured, perhaps by an independent 'quantity surveyor of software'. The customer pays an

amount computed from (delivered FP size) x (unit-price). Project risk is now much more evenly balanced. The supplier takes the risk on its bid unit-price whilst the customer takes the risk on the total size of its requirements – and gets to understand supplier performance from their bid unit-prices.

Users of unit-price contracting have claimed huge benefits in terms of reduced project cost overruns and steady reductions in unit prices (see the downloadable conference presentation 'Measures to get the best performance from your software suppliers' from Charles Symons and this short paper about a project involving the Finnish Ministry of Justice). So why has this process not caught on widely?

There are really two reasons. First, the original FP sizing method was not easy to adapt to modern software (distributed systems, web-servers, mobile access, etc) and to modern development methods (re-usable objects, agile, etc). Consequently, its use has steadily declined. Second, although all the major software suppliers use these methods to size requirements so that they can estimate for new projects (how else can they make sensible bids?), it is not in their commercial interests to share their performance data with their customers. It seems most customers have never grasped the potential of these methods.

More recently, customers have started to renew interest in methods of software sizing and their use in unit-price contracting. In part this is due to the development of the COSMIC method of software sizing that is applicable to all almost types of software (business, real-time and infrastructure), and is usable for any type of development approach including agile. It is an ISO standard and 'open' method, with all documentation freely available.

In response to the history of software project overruns, public sectors in several countries have introduced policies requiring measurement of software, eg Brazil, Finland, Italy, Mexico and Poland. And some outstanding case histories are emerging of the success of using unit-pricing to control supplier performance.

For example, Renault, the French car manufacturer, can now predict from its requirements the unit price it will pay for software in the electronic control units (ECUs) of its vehicles, typically 50 – 100 per vehicle to control the engine, air-conditioning, brakes, driver controls, etc (see 'Manage the automotive embedded software development cost & productivity', Oriou, A., et al. the International Workshop on Software Measurement, Rotterdam, 2014). Renault can also predict the chip memory size needed for its ECUs at the design stage. Finally, its data are now sufficiently robust that it can negotiate across-the-board annual reductions of unit prices for software in the same way as manufacturers do for other materials and components. Other major vehicle manufacturers are understood to be following suit.

There are also signs of new interest in software measurement in the UK. For example, HMRC has rebuilt its in-house expertise in early life-cycle software cost estimating so that it can properly carry out cost-benefit analyses of new proposals, and is in a strong position to evaluate supplier bids.

The time is ripe to consider how to introduce unit-pricing into the mainstream of software contracting.

What would a unit price contract look like?

The most obvious matters which would need to change in a unit pricing contract would be to set out the basis of the software unit price and any deliverables that are not included in the software unit price, eg training, implementation roll-out tasks, etc. Defining the software unit price is not technical, and does not require complex software engineering knowledge, just references to the appropriate standard for the software sizing method. As noted, the major industry suppliers are already using these methods.

Some typical and much-beloved elements of development contracts would need to change. In particular, the idea of a fixed scope stated at a high level leading to a presumption (express obligation?) that the whole of it will be delivered, no matter how much detail is uncovered during the project, will have to disappear. With the disappearance of fixed scope, those provisions for fixed drops of software at various contractual milestones become more topics for planning than for contracting.

So a unit pricing contract has something in common with an agile contract. The offered fixed unit price multiplied by the estimated size enable the customer to set a budget, whilst the target delivery date would remain.

What would change is the need to maintain accurate information regarding the evolving software size as the project progresses. This task could be undertaken by the customer or by a 'neutral' quantity surveyor of software, or be an obligation on the supplier. Knowing the size of any proposed new or changed requirement and the unit price enables the customer to make an informed decision on the cost/benefit of the proposal and to use that information to control the scope of individual milestones and, ultimately, the whole project. The supplier's role is to advise on what is possible, having regard to the constraints of resource, budget and time. Again, there is much overlap here with emerging agile contracts. This would need to be written into the governance provisions, which should become less prolix and more in accord with what the parties are actually going to do.

Change control would remain only for those items genuinely out of scope of the originally-published outline requirements - if something completely new comes up during the project, additional budget might need to be found and there might be knock-on effects on the ultimate timescales. However, as with agile projects, since the customer is in charge of directing the work based on its needs and on the information provided to it, this should not be a major issue, subject to the availability of the supplier's resources.

The great advantage to the customer is that it has visibility of the whole process and of the costs both incurred and estimated, allowing it to make sensible decisions as it goes along.

It might be said that the parties could fall out over the actual unit price, eg if a change requires a major reworking of already accepted functionality which might require a higher or lower unit price. Perhaps one approach on larger projects would be to have some ongoing adjudication process as in domestic construction projects. This would operate as a temporary fix, providing a binding decision, leaving it to the parties to dispute the ultimate consequences if they so chose. The evidence from the construction industry is that this has been an acceptable way forward.

Conclusion

The development of agile methods has led to changed expectations and forms of contract. The only thing missing is that the customer is not currently made familiar with the basis of the pricing which is being used by the supplier: one key piece of information is therefore lacking.

Going back to *De Beers*, it would perhaps be ambitious to assert that using unit pricing would have solved all problems as they emerged in that troubled project. However, from the start, the customer's expectations would have been different. With a fixed unit price, De Beers could have assessed the situation as the project progressed, would have been better able to decide between what seemed like a good idea and what it was actually prepared to pay for, knowing how the total cost was evolving compared to its budget.

In summary, unit pricing gives the customer critical information for the initial supplier selection, an understanding of value-for-money which is missing from all other types of contracts, and the ability to make fully informed decisions as a project progresses.

Richard Stephens is Principal of LORS Online. As well as running his own law practice, he works as a mediator and arbitrator in IT disputes. He is a Fellow of the Chartered Institute of Arbitrators, a Fellow of the Society for Computers and Law and a Fellow of the British Computer Society.

Charles Symons has over 50 years' experience in most areas of IT, ranging from scientific programming at CERN to advising on IT strategy as a Partner with KPMG Management Consulting. He co-founded COSMIC in 1998 with other international software measurement experts.

Published: 23/03/2016