Risky Business

Peter Hearty, PhD student, Queen Mary University of London

"Billions wasted on failed IT projects", "Passport agency chaos", "MPs slam ID database costs". Scarcely a month goes by without this kind of headline. If the press are to be believed, the IT industry does nothing but burn cash, especially in the public sector. Yet the UK government does have some IT success stories to boast about. The very same passport system which caused such chaos in 1999 now has a user satisfaction rating of 97% - better than many private sector sites, while the online self-assessment tax return system is rated positively by over 90% of users.

The commercial sector suffers from exactly the same problem as government departments. What exactly is it that makes software such an inherently risky business, and why are its costs so difficult to judge? The problem has been known about for decades, but the answer has always seemed elusive, at least until now. The Risk Assessment and Decision Analysis research group (RADAR) at Queen Mary, University of London, think they have the answer.

Most software cost models use simple formulas to make predictions. Project managers enter some values, representing the size and complexity of the project say, and the formula returns a time or a cost. The trouble is, these formulas are based on the "average" of many hundreds of different projects. The figures can vary wildly from one project to another making single value predictions of doubtful value. But there is another, more fundamental problem with existing models.

"In software development we normally have to make tradeoffs between quality and cost," says Prof. Norman Fenton, head of the RADAR group at Queen Mary. "A product might be ready for delivery when the number of known defects has fallen to some acceptable level, but the lack of known defects could equally well be caused by poor testing rather than high quality code. Simple algebraic models cannot distinguish between the two."

With the help of EPSRC grants, Fenton's group has gradually been developing a new class of models to help with software prediction. These "causal models" employ an intuitive and easy to use technique called "Bayesian Nets". Each node in the net represents a property of the software project, such as its size, or the experience of the development team, but unlike previous models, these are not just fitted into a one size fits all formula; the various factors influence each other, as they would in real life. This model won't be misled into thinking that, just because no defects have been found, the product is ready to ship.

The benefits of causal models don't end there, however. According to Fenton, "A lot of software engineering data is subjective and therefore couched in uncertainty. This uncertainty rarely makes it through into project estimates. Causal models maintain that uncertainty, allowing the risks in the project to be quantified and mitigated." It is this ability to make statements about the riskyness of software projects that should be of interest in large, expensive programmes, whether commercial or government run.

The recent EPSRC funded eXdecide project has exploited another feature of this style of model. Many small and medium size projects now use a highly iterative approach to software development. The software is developed in a series of small releases, which the customer gets the chance to use and comment on. These “agile” development methods lack the data needed by many RADAR models. However, they do provide rapid feedback on how well the model is performing. This allows the model to learn from its errors and quickly correct the quality of its forecasts. The learning takes place without the user even being aware of the process – they simply enter the normal data that they collect at the end of each iteration, and the model automatically makes the necessary corrections. Instead of being based on generic data from a wide spread of previous projects, the model is now specific to *this* project, with all its predictions tailored accordingly.

Being able to learn about the local environment is one of the distinguishing features of causal models. They may begin with the same basic data, but unlike older techniques, they don’t ignore the uncertainty in that data. Every bit of locally available information, whether from quantitative project measurements, or from development team opinion, is used to calibrate the model and adjust the risk assessment. The results are dramatic. Not only do the predictions closely match the behaviour of real software projects, but they also give quantitative assessments of the risks.

The days of the delayed, over-budget, under-performing software project may not be behind us just yet, but thanks to the advances being made in causal models, their end could well be in sight.