Summary of the paper **'Software industry performance – 'what you measure is what you get',** published in 'IEEE Software', November/December 2010**.**

For the full paper which contains the evidence, examples and references supporting the assertions made in this summary, go to
http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5235133

Charles Symons, April 2010

Summary

The purpose of the paper is to examine the software industry's overall performance and to explore how commonly-used performance metrics, estimating methods and processes, appear to contribute to producing this performance.

My observations are drawn from publicly-available surveys of overall industry performance and from my own experience. The paper does not discuss the achievements of exemplar organizations.

The software industry performance indicators used are:

- **Productivity** = size of software delivered / effort
- **Speed of delivery** = size of software delivered / elapsed time
- **Quality** of delivered software = defects recorded post-delivery / size of software (i.e. 'defect density')
- **Delivery to budget** = actual cost / estimated cost
- **Delivery to time** = actual elapsed time / estimated elapsed time

Software industry performance on these five indicators is very uneven.

- Delivery to time and budget is notoriously bad
- Productivity does not show much sign of improvement over time, and little is known about industry speed of delivery trends
- Delivered software quality is astoundingly good in several domains but variable in others
- All this is in spite of SPI activities having been practised for 20 years
- Customers pay heavily for the software industry's failings, but it is very rare that an external supplier actually suffers serious financial harm. Internal suppliers do not survive perceived poor performance for long; their services get outsourced.

Many factors contribute to explaining these observations. But bearing in mind the old adage 'what you measure is what you get', I explore how industry performance might be influenced by current practice in software metrics and estimating methods and processes.

First the paper briefly reviews the evidence supporting these assertions.

1. Industry performance – the evidence

Delivery to time and budget

Data from many sources, covering US and European software projects, all show high percentages of projects that fail altogether or over-run on time and budget

An important point from a survey of such projects in the UK public sector is that they were all undertaken by external suppliers that operate world-wide and would claim in their marketing to be 'world-class'.

The *annual* cost to customers of these project failings has been estimated at 100 Billion US Dollars in the US and 100 Billion Euros for Europe. To put these figures in context, these *annual* costs of write-offs due to software system project failures and over-runs are comparable to the *one-off* collective losses in 2008 of a few large investment banks due to the financial crisis.

Productivity (and speed of delivery)

One widely-quoted study concludes that US software industry productivity has actually declined over the last 15 or so years. In the follow-up discussions, several software suppliers claimed that their technology has led to major increases in 'real' productivity and/or that the 'complexity' of software has increased in recent years and this explains the apparent lack of progress in productivity. This implies that productivity metrics aren't working properly.

Another study found 'no trace of on-going improvement' in software productivity in the period 1995 to 2005. However, over this period the average team size of projects roughly doubled which, the authors suggest, may be due to the 'growing intricacy' of software development.

There is no real industry-trend data on speed of delivery of software (though it may be as important as productivity), nor on software maintenance productivity.

Quality

In contrast to the above, much delivered software is of astonishingly high quality in terms of freedom from defects. The world's communications, transportation and financial services systems operate largely defect-free. For these industries, the supplier has no choice but to produce defect-free software, e.g. to meet international communication standards, or to satisfy safety-criticality needs, or to avoid the risk of huge financial losses, respectively.

Defect-free software can also be produced when the supplier is suitably incentivised by a contract that makes it expensive to deliver defects. However, when the customer's bargaining power is not so strong, quality does not necessarily get such a high priority. Much widely-used software is full of defects and can be frustrating to use.

Industry investment in Software Process Improvement

The ‘quality movement’ pre-dates formal SPI models and much of the focus of SPI has been on improving quality. Early-adopters of SPI ideas often operated in those domains that must aim to produce defect-free software. But there has not been the same emphasis on improving productivity or delivering on time and budget.

Industry financial performance

The cost of the failings of the software industry seems to fall almost entirely on its customers. Examples are given.

2. The influence of software metrics and estimating methods on performance

Various studies have shown that few organizations succeed long-term in gathering software metrics and using them to improve performance and predictability. Often the metrics that are gathered are those that are easiest to collect.

Of our five main performance indicators, measuring quality by counting defects is one of the easiest and therefore one of the commonest metrics activities. Defects are also easily understood by all parties to a software contract.

Contrast this observation with the difficulty of reliably measuring productivity and speed of delivery of software activities, the difficulties for customers in understanding the metrics involved, and the corresponding poor industry performance. Both performance parameters rely on using a technology-independent measure of the size of the delivered software, as a measure of project work-output. The paper then discusses the use of traditional Function Points as the measure of work-output, their strengths and weaknesses.

The result of all the weaknesses is that all large collections of productivity measurements show a huge spread of results.

Estimating methods rely on such data collections to derive their algorithms but generally they do not advise the user on the inherent uncertainty in any particular estimate due to the spread of the raw data used to derive the method’s algorithms.

The paper asks if we seriously expect estimates for new projects to be realistic if the approach to estimating consists of:

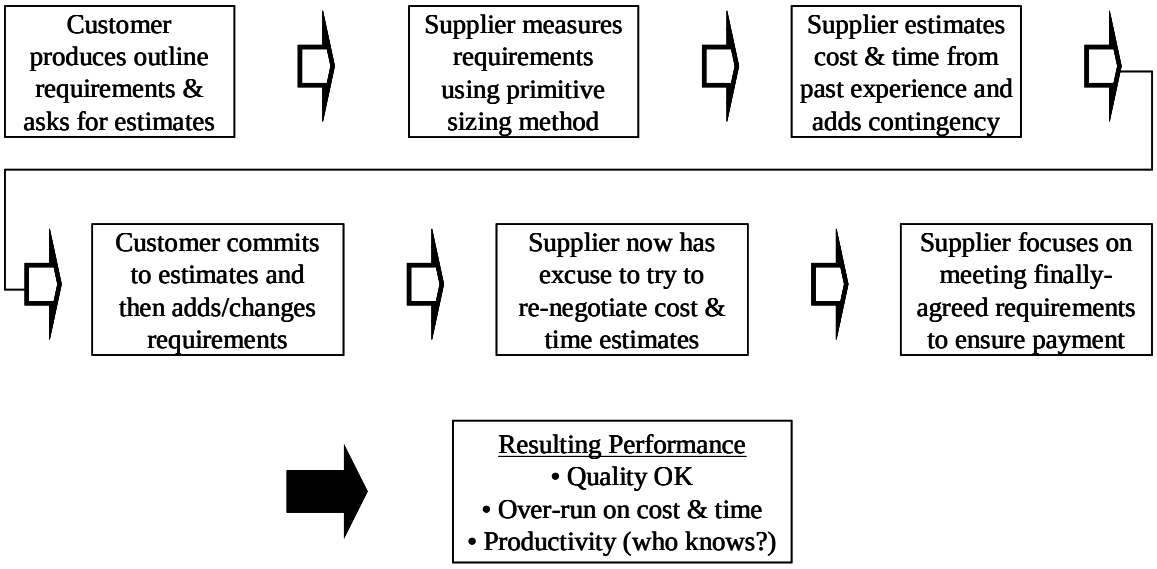
- applying a primitive method of measuring a functional size, or hazarding a guess at the SLOC.
- to measure an incomplete and unstable set of requirements,
- then entering this size (or even worse, a functional size converted to SLOC) into a ‘black box’ estimating tool that has not been calibrated to local performance,

- and finally setting the estimates of effort and time in stone, with no understanding of the real uncertainty….

The paper concludes that the problem is not that industry is necessarily poor at delivering to time and budget; it is that customers and suppliers agree to unrealistic estimates that are not much better than guesswork in the first place.

The reason why rough, initial estimates get set in stone too early in software projects is due to the poor way in which the process of using metrics for estimating is integrated with the processes of determining requirements and project decision-making, coupled with the interests of the parties involved in these processes.

Examples are given of problems on both the supplier and customer side of weaknesses in this process. I conclude there is a causal chain that links current practices in software metrics, estimating methods and processes to observed project performance, as summarised in the figure below.



A causal chain linking measurement and estimating to actual performance?

Of course not all 'challenged' projects follow this path exactly. For example, where the need to meet an agreed delivery date becomes paramount, testing may be curtailed and quality suffers. The general lesson is that a customer who makes an early commitment to an unsound estimate is storing up trouble for later.

3. The way forward

The paper proposes three main elements to overcome these problems. These elements largely already exist. I do not suggest that adopting these elements will solve all the performance problems described above. But when we have a set of known weaknesses,

and remedies largely exist that are designed to overcome the weaknesses, then it seems sensible to at least consider the remedies. The three elements are as follows.

A credible, open method for sizing the functional requirements of software, for use in performance measurement and as the primary input for effort estimation.
The COSMIC method is proposed.

Method(s) of estimating development, enhancement and maintenance effort and duration, starting from a size of the functional requirements, ideally open and transparent to both customer and supplier.

An open process for applying the estimating methods, integrated with the process for determining requirements and controlling project scope that is fair to customers and suppliers.
The Australian 'Southern Scope' process meets this need.

Examples are given of how adopting the COSMIC method reveals performance improvements that were 'invisible' when measured with traditional Function Points and also of the improvements from using the Southern Scope process in terms of improved project control, hence improved delivery to time and budget, and improved unit cost.

The paper then discusses why, if these metrics methods and processes are so good, does the market not rush to adopt them? It concludes that the balance of incentives and knowledge between software customers and suppliers does not help adoption, and there is huge inertia in the field of software metrics.

Software customers ought to be driving their suppliers to obtain improved all-round performance, but they are mostly ignorant of the possibilities for improvement and their decision-makers do not understand software metrics. Suppliers have no incentive to improve their customer's ability to control them whilst the suppliers continue to make healthy profits.

The hope must be that those in the software industry who profess to the status of 'professionals' will give a higher priority to improving software metrics and estimating practices, and to educating their customers on the levers that are available.