Most Software Development Metrics are Misleading and Counterproductive

The software development industry as a poor track record for developing and employing effective software metrics. This is because most of the metrics selected are tangential to the true goal of software development - delivering business value, and instead focus on software attributes and accounting measures.

Metrics such as lines-of-code per developer week, function points created, hours worked, or budget consumed appear important measures, but they have dangerous and counterproductive implications. The use of these metrics reward the wrong behaviour, the phrase "you get what you measure", highlights the problem. By tracking lines of code written, visible and unconscious incentives to generate lots of code are established. On the surface this may seem attractive, as a manger of a project it is gratifying to see lots of code being written, but what is really required is functionality completed, business value generated, and customers satisfied.

The more code generated the harder a system is to maintain and extend. With incentives like lines-of-code written, how do value-adding activities like refactoring simplifications appear? Reducing 20,000 lines of code to 15,000 is a good thing, but from the lines-of-code perspective it looks like the project is going backwards.

Be aware of the Hawthorn Effect

During the 1920's Elton Mayo conducted a number of worker productivity experiments at the Western Electric Company in Chicago. These experiments included taking a selection of office workers into a controlled environment where the lighting brightness could be varied and then measuring their productivity. Mayo found that in the new brighter conditions the test group were more productive than they had been in the main office. After increasing the lighting levels further their productivity went up yet again. (These findings must have seemed promising for Western Electric shareholders who sold electrical lighting systems to offices). However, as a last control experiment, Mayo reduced the lighting brightness to the default level and measured worker productivity one last time. Despite the dimmer conditions productivity went up yet again and it was concluded that it was not the lighting levels at all, rather, the fact that people were being measured that increased their productivity.

These experiments and the widely accepted phenomenon that by measuring something you will influence it gave rise to the Hawthorne Effect (named after the Western Electric factory). Because we will likely influence what we measure, we should be very careful about what we visibly measure. Lines-of-code written and hours-worked are great examples of bad things to visibly measure on software projects, as they will likely lead to bloated code and quickly consumed budgets. Instead, we should use the Hawthorn Effect to our own advantage and measure things we would happily influence.

Metrics like features completed compared to features remaining is an example of a good metric. It is simple and relevant to the true goal of the project, getting the project finished to the satisfaction of the customer. Donald Reinertsen, author of "Managing the Design Factory", offers this advice for metric selection: *"First, they should be simple, the ideal metrics are self-generating in the sense that they are created without extra effort in the normal course of business. Second, they should be relevant. One test of relevance is whether the metrics focus on things that are actually controllable by the people being measured. Psychologists have found that when people think that they can control something they are more motivated to control it. Measuring people on things they can not control simply causes stress, dissatisfaction, and alienation. A third characteristic is that they should be focussed on leading indicators. Accountants like lagging indicators that can be measured very accurately, but these point to things that have past. Managers, on the other hand prefers an imperfect forecast of the future to a perfect report on the past. It is better to measure the size of a test queue than it is to measure the processing times of individual tests because test queue size is a leading indicator of future delays in test processing."*

Scoring Our Metrics
Given these guidelines for metrics, let's assess how traditional and agile metrics perform.

Traditional metrics:

- Lines of code written – poor, does not reward simplification, leads to code bloat
- Hours worked – poor, leads to long hours, burn-out, defects, consumed budgets
- Function points delivered – poor, not relevant to true goal of project, effort to generate

It is also fair to say that these three common metrics are also backwards looking, perfect views of the past that. On today's high change projects they are not good indicators for the future. Running projects by relying on backwards facing views is a little like driving your car using only the rear view mirror and fuel gauge. You can see what you have run over and how much longer we have to go before the fuel runs out. However, we need to look ahead and make adjustments (steer) in light of leading (forward looking) metrics.

Agile metrics

- Features completed verses features remaining – good, simple and self generating
- Customer satisfaction – good, simple and relevant to true goal
- Cycle times – good, simple yet forward looking

Agile Metrics Explored
Features completed verses features remaining is a vital metric. At the end of the day it is the business value generating features rather than lines of code or function points that add value to the customer. "Features completed" means accepted by the customer (or proxy customer) not just coded and unit tested as features do not add any business value until they are really done. In fact written but unsigned-off code represents inventory in lean think, effort invested for no return yet, and should be minimized.

Cumulative Flow Diagrams (CFDs) are great ways to show Features completed verses Features remaining.

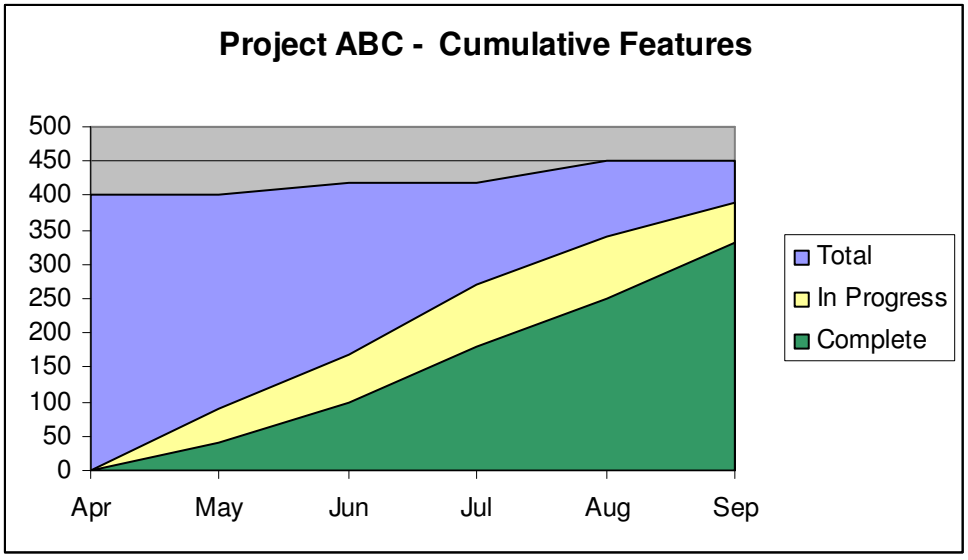

Figure 1 – Sample Cumulative Flow Diagram

Figure 1 shows the features completed verses the features remaining for a fictional project that is still in progress. The blue area represents all the planned features to be built, this number has

risen from 400 to 420 in June and then 450 in August as additional features were added to the project. The yellow area plots the work in progress, and the green area shows the total number of features completed. (CFDs are a little more complicated than burn-down charts but have the advantage of illustrating scope variances separately from productivity. A flat spot on an "Effort Remaining" burn-down chart could be the result of lower productivity or additional scope being added to the project.)

In CFDs the rate of progress is indicated by the gradient of the green features completed line. Cycle time (how long features take to be coded and accepted) can be determined by assessing the size of the yellow in progress area. (I will create a separate post on creating and interpreting CFDs including Little's Law and constraint identification in the next week.)

Customer Satisfaction can be captured via a "Green", "Yellow", "Red" satisfaction ranking from the ambassador user at regular meetings. It is better to learn of a potential issue when it is still a vague spidy-sense from the customer than after it manifests its self as a real problem. If "Green", "Yellow", and "Red" is too defined, then a line drawn on a range from happy face to unhappy face as recommended by Rob Thomsett in his Radical Project Management book can be an effective and more subjective measure. We need to be aware of how intrusive this canvassing for satisfaction is and the personalities of the stakeholders. If you keep pestering a happy person to tell you how satisfied they are, then pretty soon the pestering will reduce satisfaction. This assessment needs to be frequent, but not intrusive and adjusted to fit the customers work style.

Cycle Times are imperfect, but important forward looking metrics. The amount of work in progress, number of bugs not fixed, time from new feature identification to implementation all allow assessment and steering to be made on the project. While these queue sizes do not allow for precise forecasts of completion dates, they do allow provide the best insights available for tracking and planning.

Summary

Mary Poppendieck lists: "Cycle Time", "Customers Satisfaction", and "Conformance to the Business Case" as the key lean metrics. We can see that these are very similar to the Agile metrics and comply with Reinertsen's recommendations of being simple, relevant, and leading. Unfortunately many of the metrics used on software development projects are neither simple, nor relevant, nor leading and because of the Hawthorne Effect they contribute towards counter productive behaviours.

References

1) Reinertsen D, (1997) Managing the Design Factory, Free Press
2) Poppendieck M., T. (2003) Lean Software Development: An Agile Toolkit, Addison-Wesley

Mike Griffiths is an independent consultant specializing in effective project management. Mike was involved in the creation of DSDM in 1994 and has been using agile methods (Scrum, FDD, XP, DSDM) for the last 13 years. He serves on the board of the Agile Alliance and the Agile Project Leadership Network (APLN). He maintains a leadership and agile project management blog at www.LeadingAnswers.com.

