VELOCITY

Outcome

Velocity is defined as speed in a given direction. In software development, this translates as the rate at which we are completing a number of units of work in a given time interval. Normally, we would measure this as the number of requirements or changes completed per iteration.

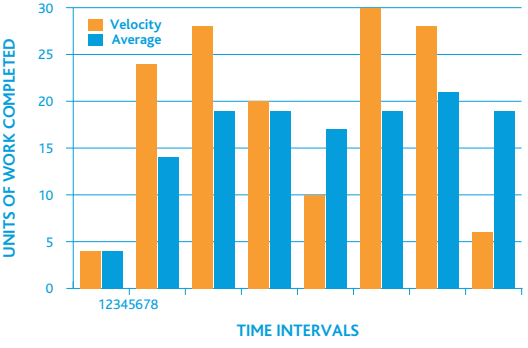
Function

Velocity helps us make relatively accurate predications of future progress. It is not perfect, but in general, the single best indicator of how much progress the project team will make in the future is how well they have performed in the immediate past.

This is sometimes known as using 'yesterday's weather' to predict 'tomorrow's weather'. The everyday analogy is deliberate – while we can predict that on average a day's weather will be similar to the one before, we know it is not always similar, and certainly never the same. Unexpected storms can always appear.

Benefit

Predictions from yesterday's data may be imprecise, but it is often better than the alternatives – perhaps asking what the weather was on this date a decade ago, or what it is like at the moment in another continent. In product development terms, the alternatives are parametric models and tools that have been fed with data from other teams on other projects (often in other organisations). Since individual circumstances are so important, an obsession with 'benchmarking' based on data of dubious relevance is a mistake.



Who

We need to move 1000 bricks from our front drive to our back garden, and we want to know how long it will take us to finish the job. We can see how many bricks we can transport using a wheelbarrow in 15 minutes. This measurement becomes our 'velocity'. For example, 50 bricks every 15 minutes means we can calculate that it would take us 5 hours to complete the job.

Scaling Factors

Bricks are of uniform size and weight – not so product requirements! Rather than just counting the number of requirements completed, we might decide therefore to take their relative sizes into account. We can count our velocity in terms of story points. If our velocity is 30 points per iteration, then for the next iteration we can accept requirements up to a total of 30 points.

Implementation

Difficulty

We need to be working in a way that means regularly repeating the same kind of activity within similar time periods. The team would work in iterations that are similar-sized time boxes, usually ranging from 1 to 4 weeks in length. Within each iteration the team would work on a number of requirements or changes to a product.

To calculate velocity it is sensible to use points to give a relative size to the requirements (see Points Estimating). Then the team count how many points are achieved in the first iteration. This becomes the base-line for the second iteration, accepting a number of requirements that add up to the expected velocity. The actual data of the latest iteration is added to the previous velocity scores to calculate a new average velocity.

The team should define what state a requirement needs to be in before counting it as done (see Definition of Done). Just as a brick that fell off the wheelbarrow and broke could not be counted, so requirements that have critical outstanding defects can't be viewed as completed for that iteration.

Calculating velocity

1. If we are using relative sizing, we need to assign points or sizes to each of the requirements that we will work to implement in the next iteration.
2. At the end of the time period, we count how many points we have completed according to our definition of done, and this is recorded as the team's velocity for the completed development iteration.

Potential pitfalls

- **Inadequate 'definition of done'** – if the definition of done does not include rigorous verification and validation of what has been produced, and correction of any problems, velocity-based predictions will give a false sense of optimism about how long it takes to complete a usable product.
- **Failing to set a sustainable pace** – if velocity is seen as an end in itself, rather than a helpful predictor, the team can initially rush to establish a high velocity. In later iterations their velocity slows significantly, as the product becomes less and less maintainable. The team should always strive for engineering excellence, refactoring during every iteration to ensure that it can be added-to or changed in the next. This steady pace is more helpful for predicting eventual completion times.
- **Lack of continuity** – if there are significant changes to the team, the length of the iterations or the nature of the work being done, then velocity becomes hard to measure consistently and therefore less valuable as a predictor.
- **Statistical analysis** – a single velocity reading is a poor statistical basis for future expectations. Five consecutive similar velocity readings represent a far better sample. In practice, significant velocity variations are normal as the team 'get up to speed'. When measuring velocity, use sound statistical analysis practices to assess and communicate a level of confidence, including appropriate rolling averages and standard deviation analysis.

If you want to learn more, consider reading:
The Principles of Product Development Flow by Donald Reinertsen
Kanban Successful Evolutionary Change for Your Technology Business by David Anderson