Agile metrics

short overview

Intro

One of the elements of Agile and Lean software development is constant improvement and adaptation. But how can we improve if we don't know the "current state of affairs" in interested areas? And here, at this point, metrics step into the game.

Metrics help measure different aspects of the development process, visualise progress, keep track of goals, measure efficiencies, etc.

In this paper I will list the commonly used metrics in Agile and Lean software development context and touch on what insides (if applied correctly) they might provide.

It also must be mentioned that using metrics and deriving results from them depends on the project specifics and the type of work done. So in every case metrics have to be applied and interpreted mindfully.

Also, some of the metrics mentioned here are not strictly tied to Agile frameworks (like Technical Debt or Defects count). However, they are very beneficial to Agile teams and, I think, really should be considered.

Agile context metrics

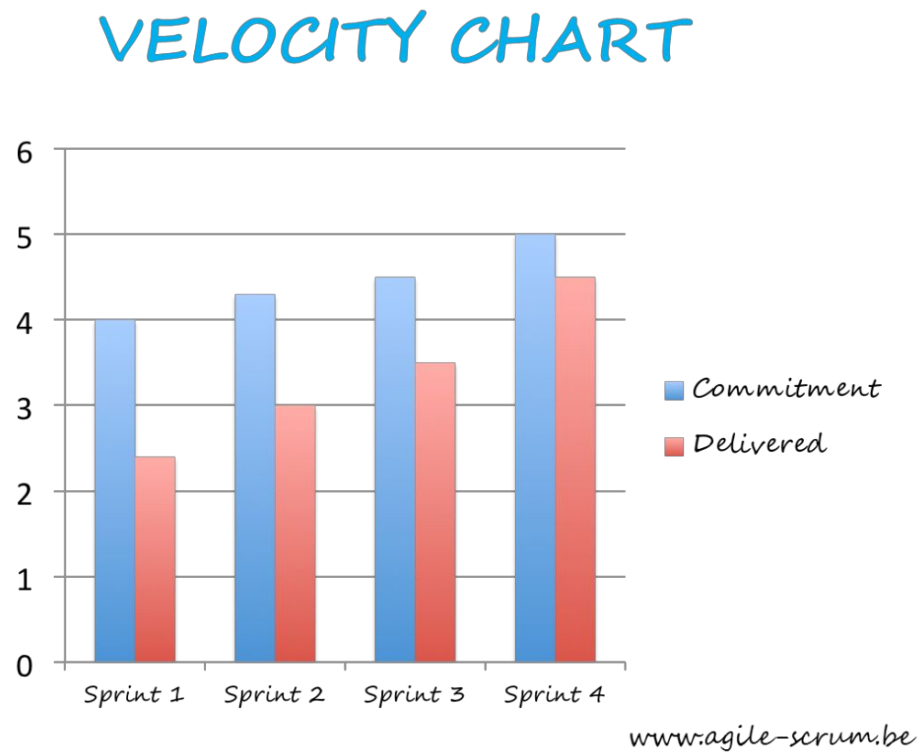
Team Velocity

Velocity is one of the most popular Agile metrics, usually considered a "must have" for Scrum process.

During each Iteration the Team can measure:

- how much work (e.g. in Story Points) they **committed to deliver** as a result of Planning meeting;
- how many Story Points **were actually delivered** and accepted by Product Owner during Review meeting.

Both of these values can be measured, documented and tracked over time.



Possible insights

If the amounts of "planned" and "delivered" work items in any chosen Iteration are too different (or they dramatically differ over time) we can look into:

a. whether we estimate correctly and maybe are being (are pressured to be) too optimistic on Planning meeting;
b. (if many stories turn out to be technically challenging) whether we need to improve technical level of the Team (bring in experts, consultants, hire senior level specialists, provide trainings or learning opportunities) so that our initial estimates are more accurate in future;
c. (if done User Stories are not being accepted by PO during Review) whether Stories/AC are formulated correctly and understood by the Team.

Note: the best practice is **not** to count into "delivered" Story Points those User Stories which were not finished within the Iteration (were just partially done) or which were rejected by Product Owner.

Actual Completed Stories Completed vs. Planned Stories

The number of **Actual Completed Stories Completed vs. Planned Stories** shows the Team's ability to understand and predict its capabilities. This metric is related and supplementary to Velocity, but still can provide additional insights. And since it is super-easy to calculate alongside Velocity - there is almost never a good reason not to have it handy.

Possible insights

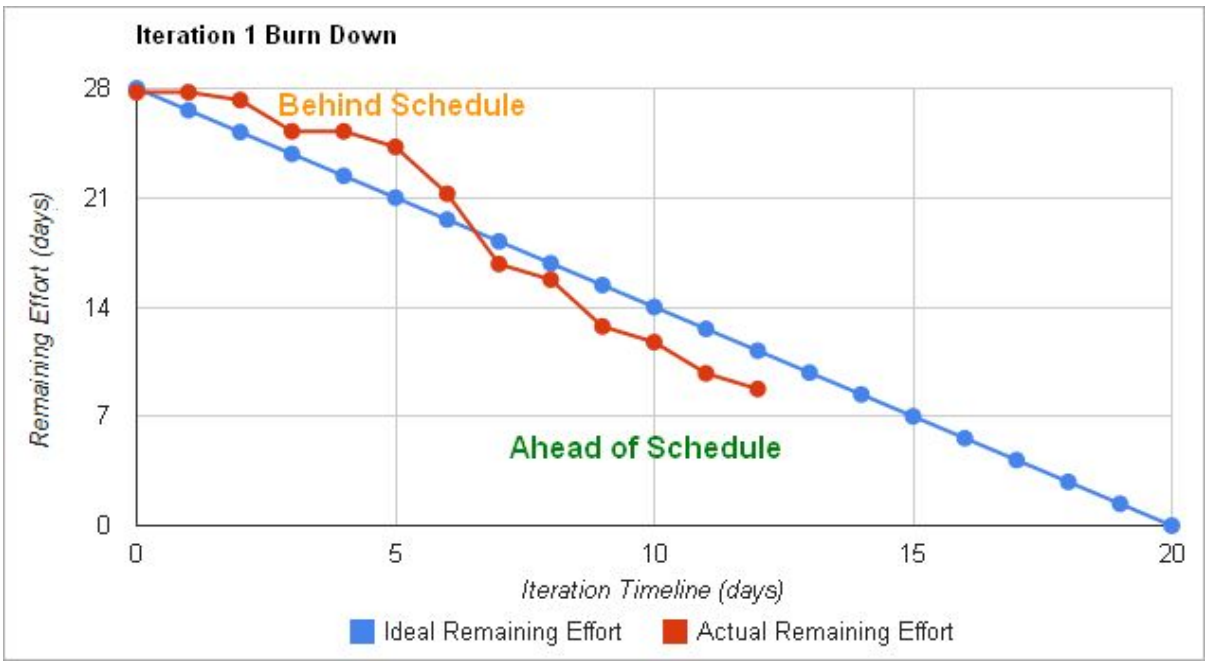
As a result of this metric the Team can visualise (and consequently try to improve) their:

- estimation and planning skills;
- level of User Story understanding required for estimation and planning;
- ability to resist the "Scope creep"; etc.

Iteration Burndown

Iteration Burndown is a chart updated after every Daily Standup meeting and **reflecting how Tasks, User Stories and/or Story Points are becoming "done"** as a result of Team's work during the Iteration.

Usually, Scrum Master is responsible for Iteration Burndown update, but this activity can be performed by anyone in a Team.



At the beginning of the Iteration Sprint Burndown only shows the "ideal burning trajectory", but every day the real data is coming in.

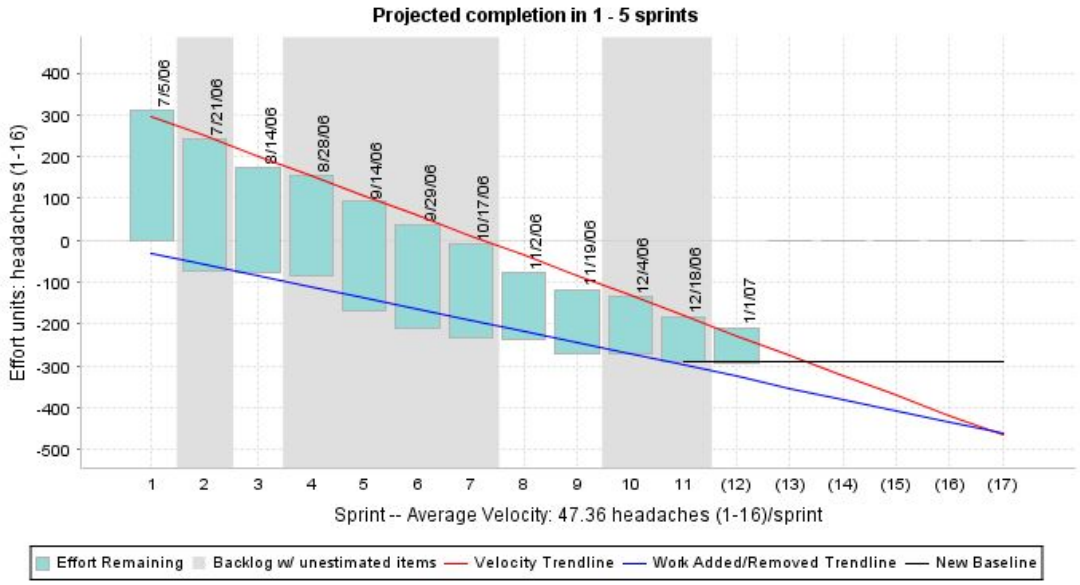
This way the Team can conclude whether:

1. They are on track to complete the Sprint commitment fully;
2. Raise early the issue of "over/under-commitment" with PO and manage the scope accordingly.

Release Burndown chart

Release Burndown tracks **the progress of entire release development** over several Iteration and possibly over several Teams.

This chart is especially interesting since it explicitly shows the moments when the scope of the Release is being added. On the example below it happens when the Y axis (the one representing the planned effort) plunges into values under "zero".

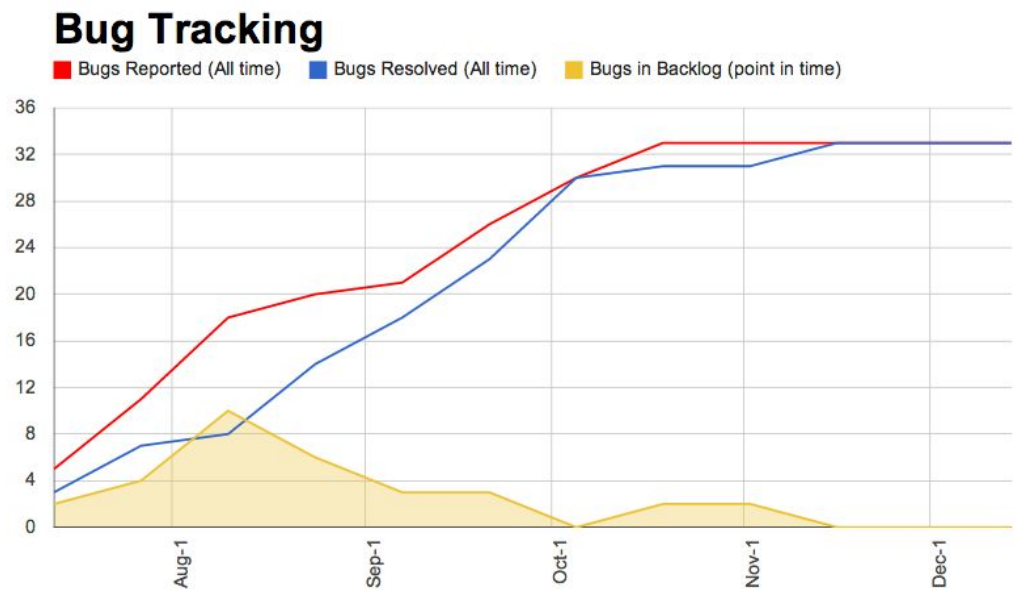


Possible insights

This metric helps catch moments of lagging behind and/or "scope creep" situation (when scope items are being added into previously agreed release definition without renegotiating other parameters).

Number of defects (especially from UAT/Production)

This metric shows **how good the QA process is**.



Possible insights

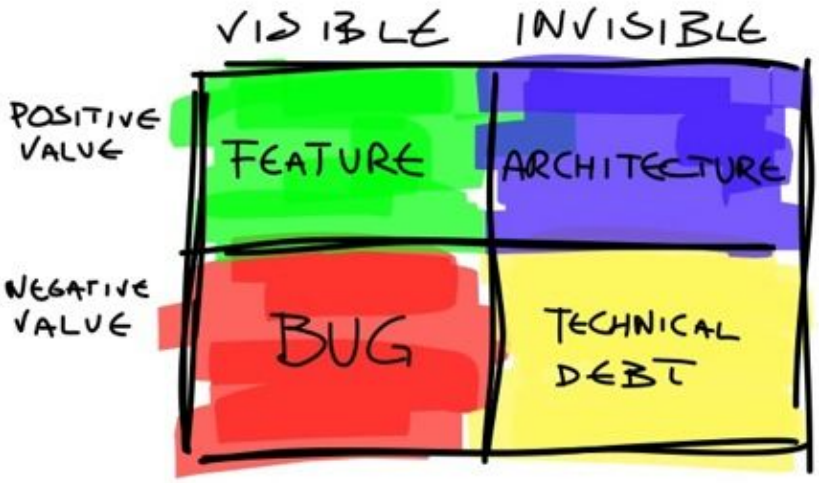
If defects are leaking to UAT or (God forbid) to Production environments, this may uncover:

- unequal environments for testing, UAT and/or Production;
- inefficient acceptance procedures during Review meeting;
- or even low quality of Product Backlog.

Technical Debt

Maintaining a **Technical Debt list** may be crucial to the project's success.

When development Team has to forgo optimal technical solutions in favor of quick fixes, all such instances should be well documented including the technical description, date/release/commit ID, business reasoning behind the decision and possible risks (both business and technical).



Possible insights:

The size of Technical Debt list and the type of issues it contains can shed some light on the development process and the negative influences it sometimes endures.

It helps not just in registering technical issues in need of fixing, but also serves as a point of reference if we're making a comprehensive argument (e.g. in case technical side of things are constantly neglected in favour of "urgent business" priorities).

Retrospective Issues identified vs. fixed

Retrospectives should be effective and therefore issues which are being discussed should be tracked.

There are many techniques for identifying issues (as well as noting good things) during Retrospectives. One of the popular is "Mad, Glad, Sad" and there are many others. But when issues are identified - they also have to be tracked. Otherwise, there is no point even to have a Retrospective, except maybe "a good team building vent" :)

The Team can create a list in organization-wide-accessible Jira and assign issues to appropriate owners. Or other tools and mechanisms can be used.

Possible insights:

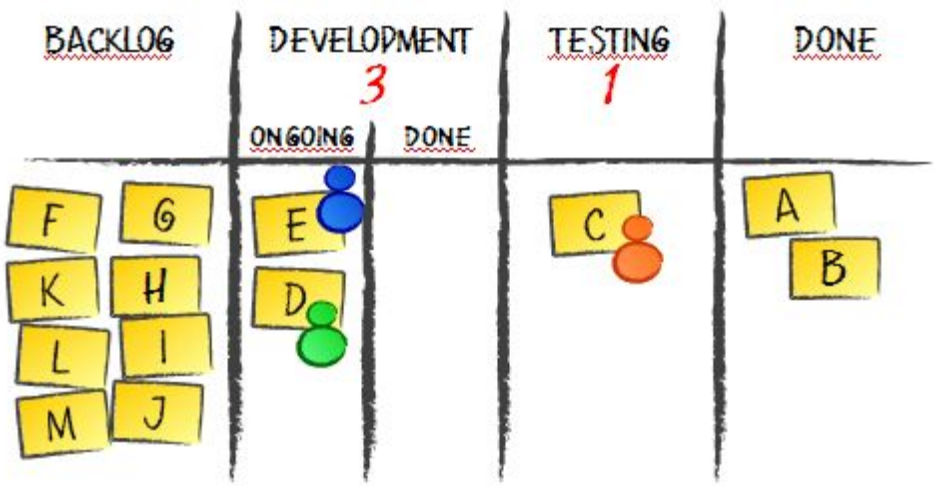
The key here would be accessible data on how quick issues are being resolved (if they even are), whether they re-surface with time, who is stalling (if anyone), etc.

Lean context metrics

WIP

Tracking and managing "Work in Progress" (WIP) items is a key instrument in Lean software development. When the Team starts adopting e.g. Kanban methodology, defining and managing WIP is one of the first on the list.

It starts with defining the number of work items allowed to be in progress in every stage of production flow. Specific numbers are first assigned by heuristic formulas or using "educated guess", and are indicated on the Kanban board. With time these numbers are refined to facilitate the Team's efficiency.



Possible insights

Obviously, as Kanban instrument, WIP can show the Team the bottlenecks in the process (details), but not only that.

For example, if we are documenting WIP over time, we can see how it changes as the Team becomes more experienced, as people with different specialization or level of knowledge join or leave the Team, as the Product matures or the nature of the tasks changes, etc.

Also, WIP is used to manage teams effectiveness in terms of "finishing vs. starting new", but this topic is outside the scope of this paper.

Lead Time and Cycle Time

These two metrics are related and often used interchangeably. However, Lean and Kanban gurus point out that there is a significant difference between them.

Cycle time is a frequency in which units (work items) are produced (time from producing one unit to the time of producing the next one). E.g. if we have a Cycle time of 3 days - this means that on the average every 3 days one unit of work is done.

Lead time is a measurement of how long it takes for one unit of work to make its way through operations from start to end. For example, the Team can produce new unit of work every 3 days (**Cycle time**), but production of one such unit takes 6 days (**Lead time**). This just means that 2 subteams are working simultaneously thus making Cycle time two times shorter than Lead time.

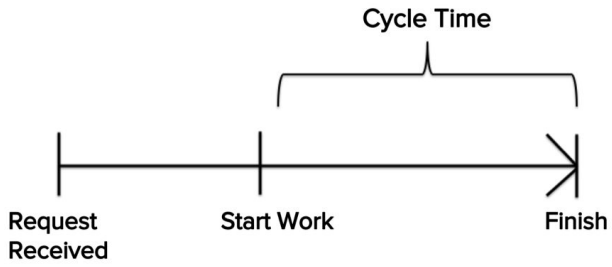
Possible insights:

Understanding the type of tasks and having the correct measurements of Lead and Cycle times, the Team can optimize its performance.

For instance, we can add more people to the Team. If they start working on additional work items - the Lead time will stay the same, but Cycle time will decrease (less time to wait for the next item to be done) and vice versa.

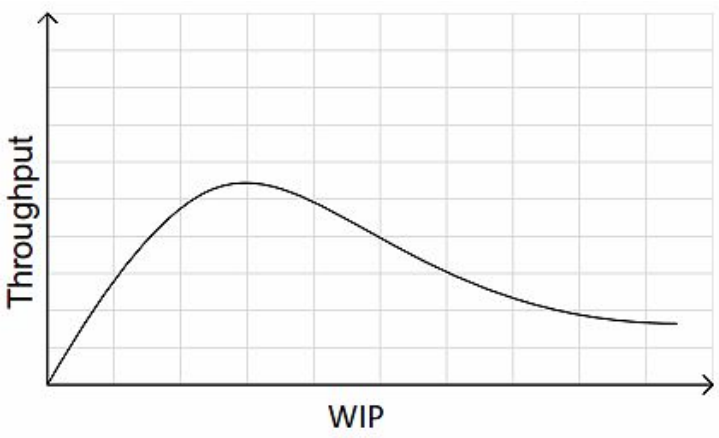
Note: In specific software development context sometimes Lead Time and Cycle Time are defined as:

> **Lead time** - time *from* registering a request from the Customer (or bug from production system or defining a User Story, etc) *until* it is available (on production servers, in stores, etc).
>
> **Cycle time** - time from the moment that the Team starts working on the task to the time it reaches the "done" state.



Throughput

Throughput is the **amount of work items delivered per time period** (week, month, iteration, etc).



This metric may be calculated using a Little's Law:

$$\overline{\text{Lead time}} = \frac{\overline{\text{WIP}}}{\overline{\text{Throughput}}}$$

Possible insights:

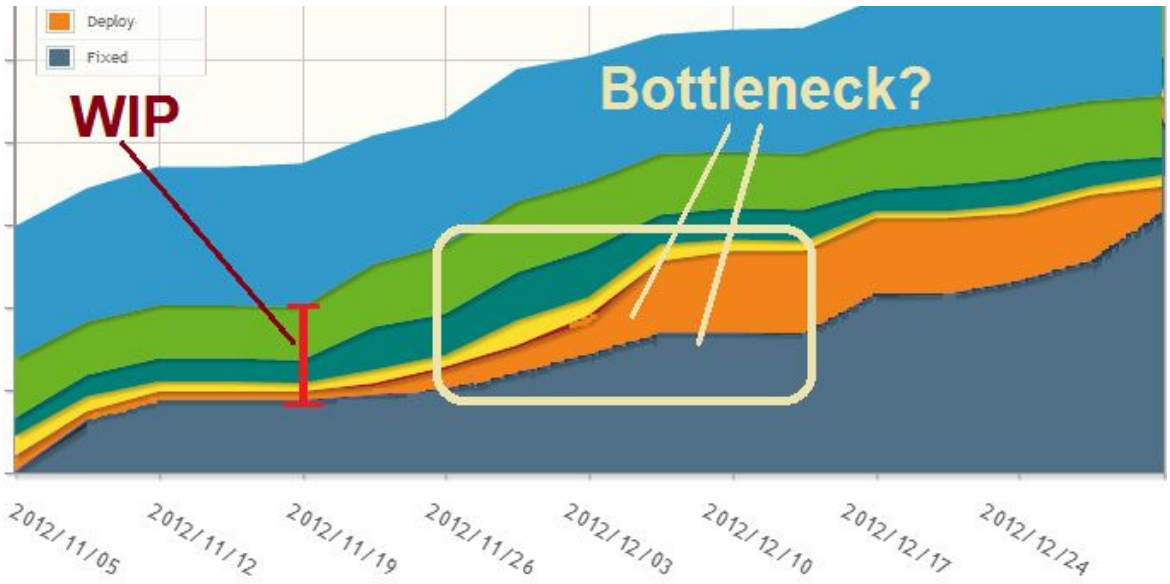
This metric is not 100% accurate and is usually calculated for the average values. But it can be used to predict delivery times, for planning and for balancing the efficiency of the Team.

Cumulative Flow diagram

The Cumulative Flow diagram helps **understand the flow of work** across the Team over time.

It is build by indicating the number of Work Items on the Y axis and the Time on the X axis, and Work Items are shown in different colors depending on the workflow states they are currently in.

The Cumulative Flow diagram is one of the most informative and productive metric for Lean software development framework (details). E.g. vertical projection of WIP area gives us the **WIP in estimation points** (in that particular time)**,** and horizontal projection indicates **the lead time**.



Possible insights

The Cumulative Flow diagram should look smooth(-ish) from left to right. If not, different anomalies may indicate problems. For example:

- **jumps** might signal of significant changes (e.g. adding or removing Team members) as well as inconsistencies in production flow;
- **thickenings in WIP area** might indicate the bottle necks;
- if the area of **Backlog Items thickens over time** - it means that scope is being added (or estimations of existing scope are being increased);
- projected value of done/fixed area reaching the top right corner of the chart shows an **anticipated final delivery date** for the Backlog in current state.