Pro
ware
ness+

Agile Metrics

Let the Numbers tell the Tale

Table of Contents

Introduction

Whether we realize it or not, we all use metrics on a daily basis. When we get into our cars in the morning we keep an eye on our speedometer to make sure we're not going too fast, we hoist ourselves on a scale every now and then to keep the weight in check, teachers and peers have been grading our abilities since an early age, the list is endless. So why do we measure? We measure so that we can take action when whatever it is we're measuring is deviating from that what we expect or desire. Which brings us to the key message of this whitepaper; metrics should always lead to actionable results. It should be clear what the measured data means, and it should be clear which actions to take based on the measurement. There are hundreds of metrics available for software development and Scrum, not all of them make sense.

Metrics should be a key aspect of every (software) development process. Thankfully, Scrum, with its iterative foundation, makes collecting these measurements and responding to them extremely straightforward. In this whitepaper we will provide the reader with a set of metrics that we have selected based on years of experience coaching and working with Agile teams. We do not think that this is list is exhaustive. The collection of metrics an Agile team should use is context dependent, but it should provide the reader with enough knowledge and insight to get started.

The most important questions to answer about your Scrum implementation

We need some sort of structured approach to help us compile a minimal set of metrics from the long list of metrics that we have composed over the years. There are several models that can be implemented when deciding which metrics to use to assess software characteristics. An approach commonly used is the Goal Question Metric (GQM) approach. This approach originates from research into defect evaluations in the NASA Goddard Space Flight Center in 1984[1]. It has since been adapted to function as a quality improvement paradigm in software development.

In short, the GQM approach takes a top-down perspective for quality metrics; an organization needs to achieve certain goals, the metrics should be able to show whether or not those goals have been met, and they should be able to guide an organization towards fulfilling those goals[2]. A goal is broken down into a specific format, so that it contains a purpose, an issue, the object or process of focus and the viewpoint of who or what the goal should be related to. A set of questions is selected that provide context to the goal; when these questions have been answered we should know whether the goal has been reached. The metrics provide the answers to the questions.

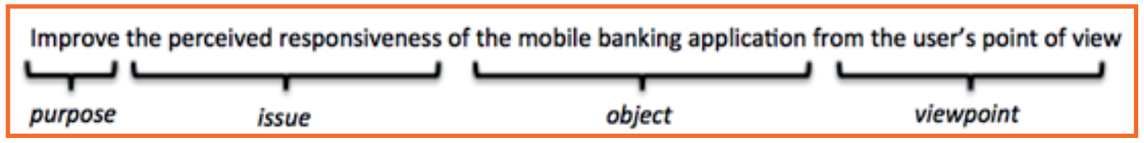

Figure 1, A breakdown of a GQM goal.

To illustrate, take the following goal for a banking app on mobile devices: Improve the perceived responsiveness of the mobile banking application from the user's point of view.

1 R. Basili, D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, vol. SE-10, no.6, November 1984, pp. 728-738.

2 Van Solingen, Rini; Egon Berghout (1999). The Goal/ Question/Metric Method. McGraw-Hill Education. ISBN 0-07-709553-7.

A question, which we can link to this goal, would be ‘what is the relationship between boot-up time and perceived responsiveness?’ the metric we use to answer this particular question would be the startup time of every iteration of the product to date and the responsiveness as judged by the users of the app. Another question that could be linked to the goal is ‘how fast do the apps of our competitors load?’ as this might be the criterion customers use to judge your application.

So we need to set a goal for usage of Scrum metrics. Why do we want to measure our Scrum process? In fact, why are you reading this whitepaper? We think this goal can be described somewhere along the lines of: To improve the results of our Scrum implementation from the business point of view. Of course a Scrum implementation is multifaceted, so we need to make sure that our set of metrics covers all of the aspects of the Scrum implementation. The questions that we link to our goal need to have the same multifaceted characteristic.

We converted those aspects, key areas of a Scrum implementation, into the following questions: Satisfaction of stakeholders (how happy are our customers? How happy is the team?), the maturity of the team (are we constantly improving?), the alignment with the business (are we focusing on delivering value for the business?), the delivered quantity of work and the quality of the work delivered.

These five key areas are inter-connected: it makes no sense to focus on an increase in productivity (Quantity) if that means delivering buggy or low value software (Quality). Similarly, delivering high value work (Alignment) but not spending time on education and knowledge transfer (Maturity) might be beneficial on the short term but will come back to haunt you in the long run. And of course Satisfaction is a general characteristic that should always be achieved. We have defined a number of questions to align our goal to the GQM model:

- How satisfied is our Scrum Team and Stakeholder?
- Are we constantly improving our team maturity?
- Is our development effort aligned with the business?
- Are we increasing the quantity of work delivered?
- Is the quality of the work compliant to the norm?

The following paragraphs describe the metrics we propose to answer the questions mentioned above. It should be noted that there are many, many more metrics than the selection you are about to read through. The aim is to keep the set of metrics limited, but all-inclusive with respect to describing the key areas of Scrum. There must also be a causal relationship between a metric and the performance of a team or company.

The interpretation of the resulting answers when measuring help create actionable improvements. These improvements should lead to improved results.

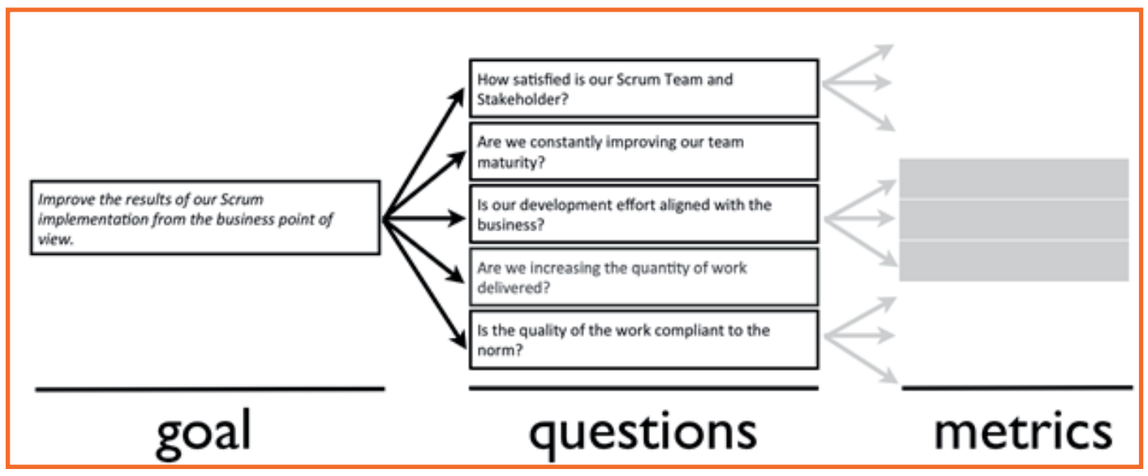

Figure 2, this paper’s goal and questions visualized.

How satisfied is our Scrum team and Stakeholder?

We measure satisfaction from two perspectives, from the 'supply' side and from the 'demand' side. The Scrum Team is the supply side in this case, we use the Team Member Happiness metric for the supply side and the Net Promotor Score, or NPS, for the demand side, the business in this case (please see Appendix A for a summary of these metrics).

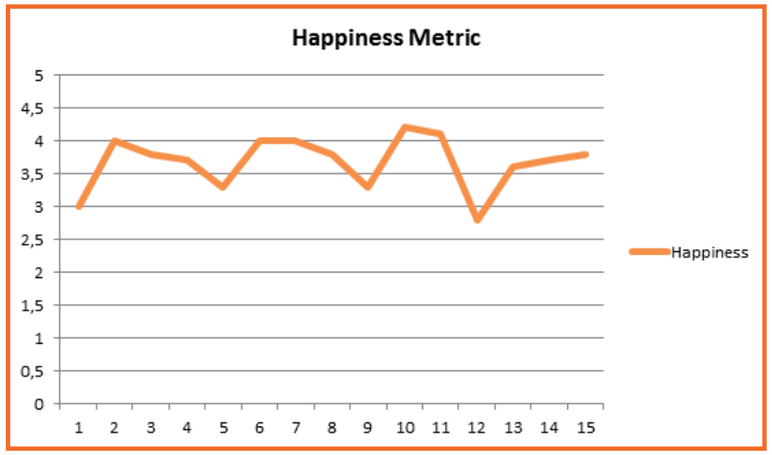

Figure 3, a Happiness metric spanning 15 sprints.

Team Member Happiness is an important metric as things that are going wrong and right during development will almost always reflect on the happiness of everyone involved. Team Member Happiness is commonly measured at the end of a sprint. Five post-its are placed on a wall (or door, any surface will do), each team member is asked to place a sticky next to the numbered post-it that best reflects his or her current happiness, where 1 is very unhappy and 5 is very happy. Every team member is also asked to speak out on what is needed to improve their happiness for the following sprint. It is this characteristic of revealing introspective elements that makes the Happiness metric especially actionable, while remaining lightweight. It doesn't necessarily take a lot of time and resources to identify and solve lingering issues or to put those things that are working out well in the spotlights.

Figure 3 shows Happiness Metric data that one of Scrum coaches has gathered over a period of 15 sprints. Note the dips at Sprint 1 and 12, and the smaller dips at sprint 5 and 9. Without additional data these dips could have been caused by variety of reasons. We know that the dip at Sprint 1 was caused because the team just started Scrum, which was a cause for great anxiety and stress. The dip at Sprint 12 was caused because a number of team members called in sick and so the team wasn't able to successfully complete the sprint. Sprints 5 and 9 mark the completion date for a release. As the release approaches the team is starting to feel the stress. Reading such a collection of metrics data requires this background knowledge, without it no useful data can be extracted from the graph.

The NPS is a metric that is used by many organizations in a variety of fields. Where the Team Member Happiness places focus on the Scrum Team, the NPS is focused towards the customer. It is used to gauge customers how likely they would recommend the services or products of a company to others; this is seen as a major indicator of customer satisfaction. The customers/stakeholders grade the performance of the team on a scale from 1 to 10. Scores ranging from 8 and up are usually seen as preferable. In Figure 4 we can see that the NPS is relatively low at the start, this is not strange, because the first few Sprints of a team tend to be chaotic and are usually associated with a stress and unhappiness. We see that the NPS quickly picks up until there is a sudden decline, this could have been caused by a decrease in quantity, but it could also have been caused by the appointment of new Stakeholder unfamiliar with Scrum. Again, we need the context in order to make sense of the data.

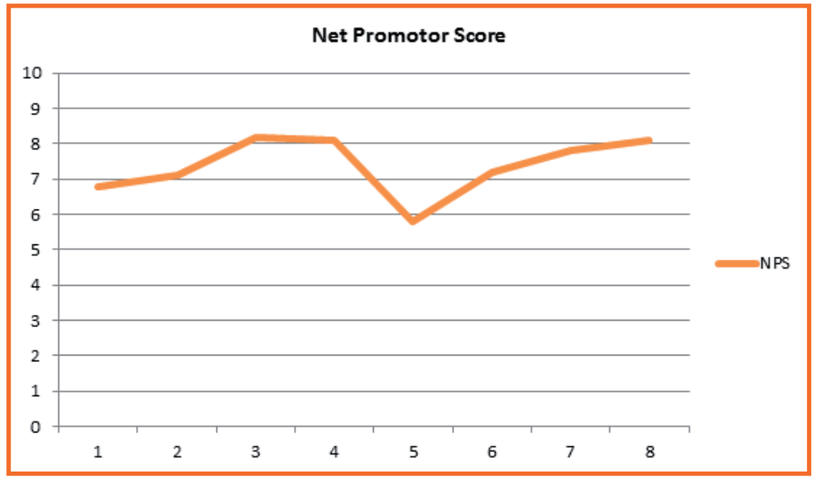

Figure 4, The Net Promotor Score of a team

Are we constantly improving our team maturity?

The inspect-adapt adagio does not only apply to the software. The Scrum team itself must constantly inspect and adapt its own processes and skills and strive for continuous improvement. Teams that fail to learn will stagnate and are doomed to fail. Hence, new knowledge must be created and shared between team members and dependencies on individuals must be resolved to guarantee continuity. Within some organizations this is part of the company DNA and these sorts of processes will happen naturally, at others it might be necessary to schedule some time in every sprint for knowledge transfer. We propose a quartet of metrics to make the transparency process transparent (please see Appendix B for a summary of these metrics).

The Ratio of Successful Sprints shows how many sprints have been finished successfully as compared to the total number of sprints. Obviously as the Scrum process and team matures this ratio should increase over time. The Ratio of Successful Sprints will only be of value when the Scrum Team has completed a minimum amount of Sprints.

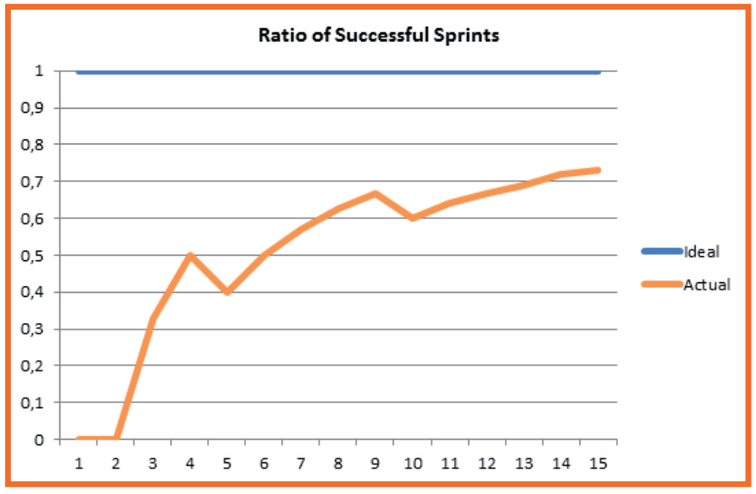

Figure 5, the Ratio of Successful Sprints

The Focus Factor shows the percentage of time the team spends on committed work. This number should also increase over time as team members spend less time on non-committed activities when they grow accustomed to the Scrum way of working.

More formally, it calculates how much time was spent on requested, completed and accepted software. The Focus Factor can be calculated by dividing the Work Capacity by the original sum of estimated Story Points (the Velocity) of a team. The Work Capacity is simply the total reported amount of work. Because we use percentages and relative values here, the Focus Factor can be used to compare teams. As with the Ratio of Successful sprints we expect the Focus Factor to increase and stabilize over time (see figure 6), a Focus Factor of 0,8 is considered to be acceptable.

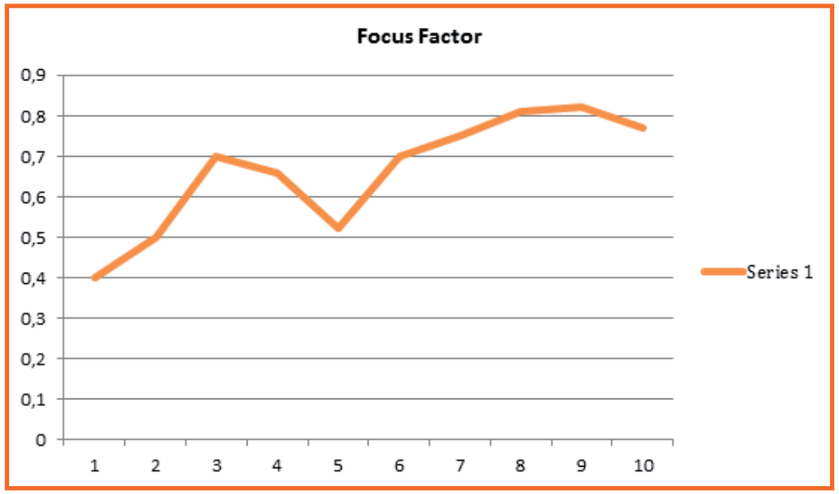

Figure 6, The Focus Factor slowly progressing to the preferred 0,8-1,0 bandwith

The team's ability to accurately estimate the required amount of effort to finish the Sprint Backlog is another maturity metric. This Estimation Accuracy will also increase as a team becomes more mature. The Estimation Accuracy can be calculated by dividing the actual amount of time spent on completing the committed User Stories by the original estimation of time. This means that the accuracy can overshoot and undershoot the optimal bandwidth, as can be seen in Figure 7.

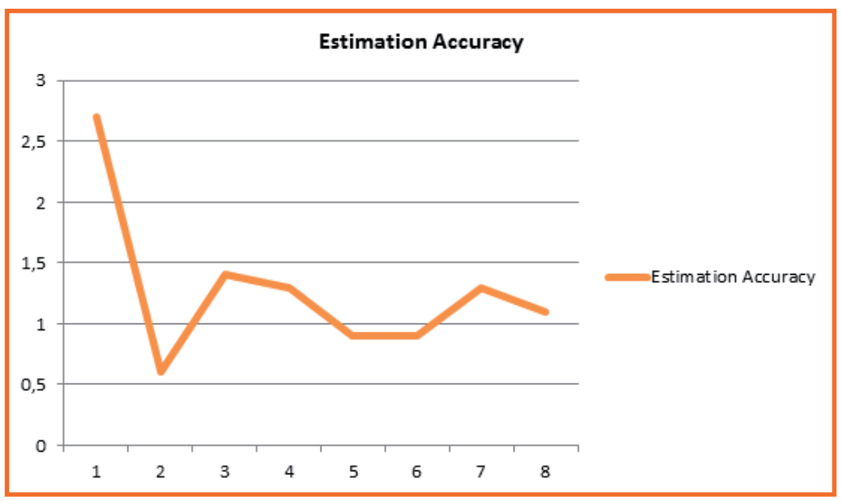

Figure 7, the Estimation Accuracy is slowly stabilizing

The final metric is the Reliability metric, which compares the amount of earned Story Points versus the amount of committed Story Points. Finishing all but one Task of a User Story will still cause that User Story to remain 'not done', although many hours of effort have been spent on the Story. The ability of teams to finish those tasks they have committed to, is reflected in the Reliability, it is simply the amount of estimated Story Points for the completed Stories divided by the total amount of Estimated Story Points for the Backlog.

We further pose that Team Member Happiness is also an indication of the maturity of teams. Especially during the inception of teams there will most likely be some conflict due to the fact that team members have to get used to working with each other.

Is our development effort aligned with the business?

Great teams can deliver great software, but when the software delivered isn't of high value the Business will still not be content. Therefore we encourage a tight alignment of the development team with the business. We measure the business alignment using two metrics, the Return on Investment (ROI) and the Total Business Value Earned. We further propose an end-to-end metric that shows whether the team is tackling all key areas of the product (please see Appendix C for a summary of these metrics).

In order to improve the results of a team, we have to make the value they produce transparent. In linear software development methodologies this is a complex matter, thankfully Scrum's incremental method of software delivery makes these calculations a little easier. We make the business value Agile teams deliver visible by calculating the ROI for Epics and User Stories. Broadly speaking the ROI can be calculated in two different ways; by taking the actual monetary value of User Stories, or by calculating a relative value. Naturally both methods have their pros and cons. In order to come to a toolkit with lightweight metrics we prefer to work with relative estimations, especially for organizations that are relatively new to Scrum.

We use an estimation technique similar to the one used for Effort Estimation to associate a number of Value Points to a work item, for example using the Fibonacci range. We pick one User Story (or a small set of Stories) to set the baseline for the value of a Value Point. It is important to note that the Business Value should be estimated by the representatives of the Business, i.e. the Product Owner (PO) and business stakeholders, and not the development team. At the end of a sprint all the Value Points of the User Stories and Bugs can be accumulated to gather the *Total Business Value Earned*. The ROI can then be calculated by dividing the total value by the total Estimated Effort of the sprint.

A good Product Owner will always take multiple metrics into account when assessing a situation. Take for instance Figure 8 below. Figure 8 displays only the Velocity of a Scrum team, we see tat the Velocity is decreasing rapidly after Sprint 5, a reason for concern, right? But what happens when we project the Business Value earned over this data? As we can see, the Velocity was increasing, but apparently that was due to the fact that the team was delivering items of little Business Value. Perhaps the Velocity was artificially high because the team was only tackling smaller items. Thanks to the combination of the Velocity with the Business Value Earned we can see that it is unwise to aim solely at raising the Velocity when that means hurting the Business Value. That is why the Business Value should be taken into account when prioritizing a Backlog.

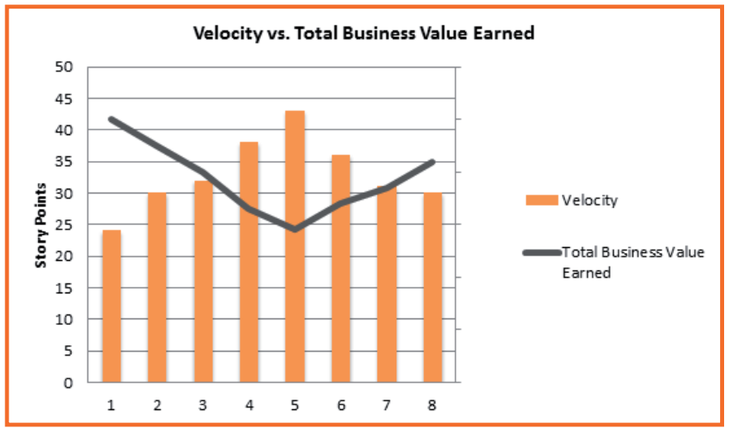

Figure 8, a eam's Velocity mapped against the earnedBusiness Value.

We propose an End-to-End metric that shows whether or not effort is being spent on the most important aspects of the software product. The software product can be divided in a number of components, i.e. the login component, the payment portal, but also less tangible characteristics, like usability. Every User Story the team commits to and finishes is added to one of these components. After

a number of Sprints this metric will be able to show you whether or not the effort is spent on the right items.

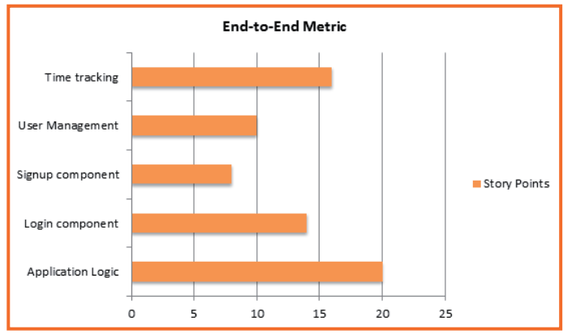

Figure 9, the End-to-End metric

Are we increasing the quantity of work delivered?

We propose four metrics to measure whether the team delivers a consistently increasing amount of work, Velocity, Process Efficiency, Release Burndown and Release Burnup (please see Appendix D for a summary of these metrics).

Scrum teams should always try to achieve the highest possible Velocity (without sacrificing the quality of the work). A team's Velocity is based on the amount of Story Points a team earned during a sprint. If the team finished three User Stories, worth 2, 5 and 13 points, the total Velocity for that sprint is 20 points. It should be noted that the time spent on unscheduled work is not captured in the Velocity. Similar to Value Points he team settles on a User Story (or a set of Stories) to be used as a baseline norm for effort estimation. This means that the Velocity is a relative metric that cannot be compared across teams. The trend in the Velocity growth over time, however, is comparable across teams.

The figure below displays the Velocity of a team set out to the Forecast of the team. As you can see the Velocity has a positive trend but is subject to a lot of changes that cause it to dip periodically. In the Scrum team where this data was gathered from these dips were caused by sick leaves of team members. In this particular case there is not much that can be done to mediate this. However, if these dips were caused by User Stories that were too complex and hence couldn't be finished within Sprint, that would be a sign for the Product Owner to step in and review the Product Backlog.

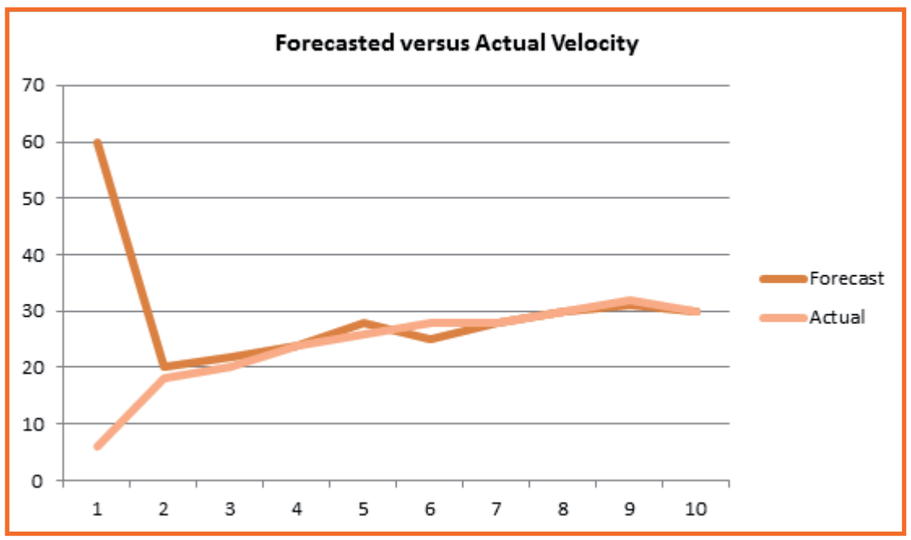

Figure 10, The Forecasted Velocity versus the Actual Velocity

The Process Efficiency shows how efficiently team members work on the tasks they have committed to. For example, when someone starts working on a User Story with an estimated amount of effort of 4 hours at 9 am, we would expect this User Story to be finished at around 1 pm. If the User Story is pushed to done at 2 pm, our Process Efficiency is 80% (4/5 hours). People work most efficiently when they can focus on a single task, thus a high Process Efficiency percentage is preferred if the quantity of work delivered should increase.

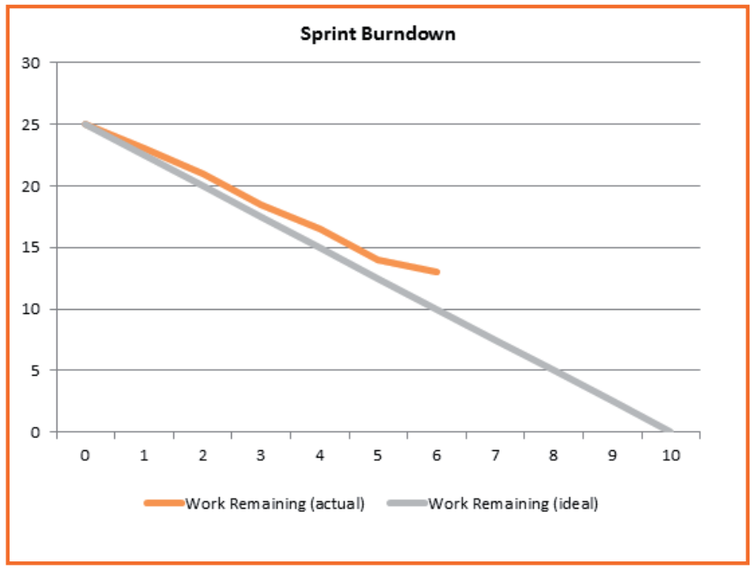

Figure 11, The Sprint Burndown

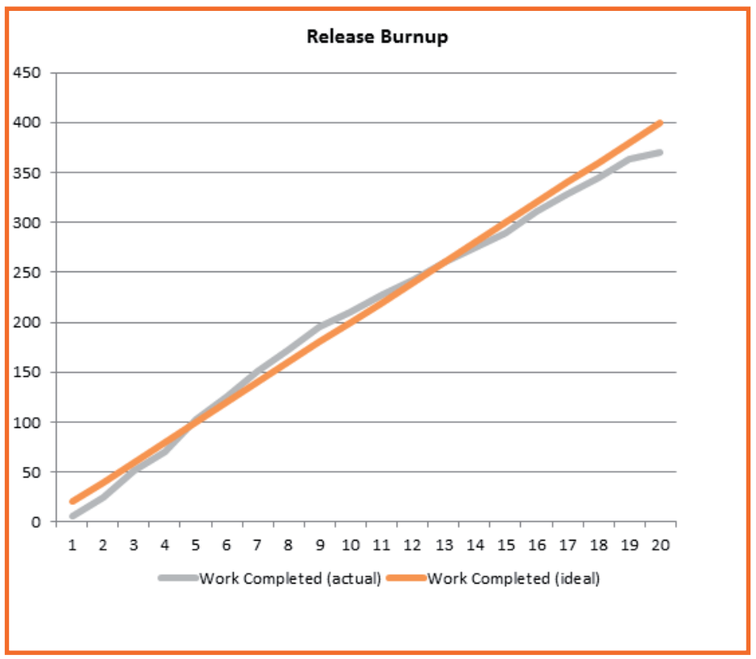

Figure 12, the Release Burnup

Another quantitative metric is the Sprint Burndown, the Sprint Burndown shows how much work (in Story Points or in hours) has been completed during a Sprint (the Stories which are marked as done), how much work is left, and how much work should ideally be left. Displayed in Figure 11 is a Sprint Burndown graph of a two-week sprint (10 working days). We say that as time progresses, the team deviates further away from the ideal line. This would be the right time to step in and make sure the team hasn't taken up more work than it can chew. Note that when the Actual line is below the Ideal line, this is also not a good sign as this means the team didn't accurately estimate the amount of work it can take up.

Another variation of the Sprint Burndown metric is the Release Burnup, displayed in Figure 12, which shows a team's progression towards completing not the Sprint Backlog, but the Release Backlog. The value on the x-axis are sprints, not days, the value on the y-axis is in Story Points.

Is the quality of the work compliant to the norm?

Estimating the quality of a piece of software is a difficult task, as quality is often based on intangible characteristics, such as look and feel. To make software quality more tacit, quality characteristics usually focus around bugs and defects, as these are commonly agreed upon to be visible artifacts of low quality software (please see Appendix E for a summary of these metrics).

We use two metrics to judge whether the software we deliver is of high quality, the Defect Count and the Fault Severity. The Defect Count is simply an accumulation of all known defects in the software. Obviously high quality software should contain a low number of bugs. It must be noted that the amount of defects detected is also largely dependent on the rigorousness of the testing procedures; the fact that no bugs were found does not guarantee that they aren't there. So one should be careful when judging the quality of the work delivered based on the Defect Count.

As we can see in figure 13 the Defect Count for this particular Scrum team is quite low at first, this might be due to the fact that in the first few Sprints not much functionality has been delivered. As the Velocity picks up, so do the known defects. After some time we see the time starts to slowly get in control, perhaps due to an improved Definition of Done and testing procedures, and the Defect Count starts to decline.

The Fault Severity metric can be used to judge the severity of known defects, as the quantity of known defects in itself does not give much insight in the underlying quality of the work. After all, one critical bug in the underlying business logic that takes 2 days to resolve is obviously much more troublesome than 5 CSS layout issues that can be fixed in a matter of minutes. We pose that the poorer the quality of a piece of software, the longer it takes to fix individual defects, so we check the severity of the fixed defects, called faults. We make the distinction between minor faults, major faults and critical faults.

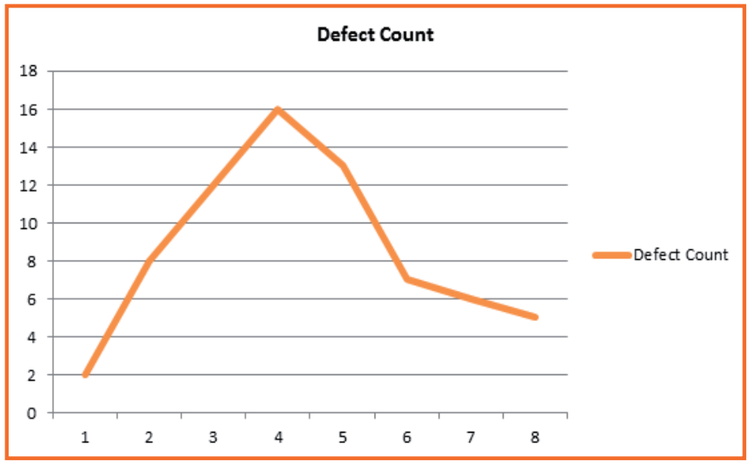

Figure 13, the known defects in the Backlog

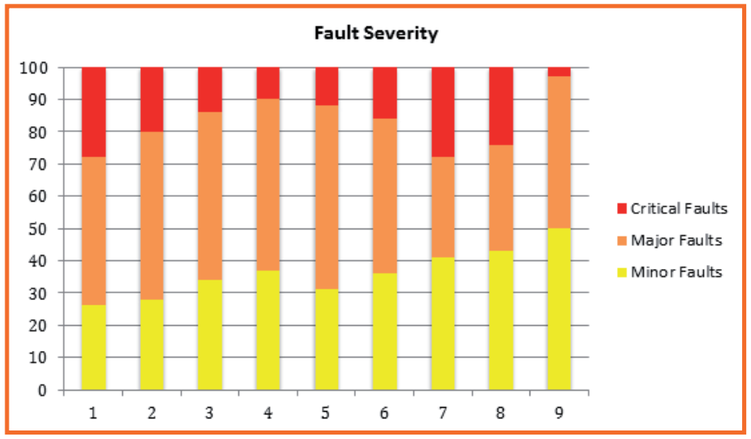

Figure 14, the Fault Severity changes in composition over time

The Scrum team can judge which of the labels applies to which defect. Preferably, the amount of defects fixed with a critical nature should be smaller than the amount of defects fixed with a less severe nature. In the figure below we see that the team is slowly progressing towards that state.

Conclusions

In this whitepaper we provided a lightweight and understandable set of metrics that can be used to track a Scrum team's performance. Although there are many more metrics to be found in literature, we feel that the ones included in this paper are a good set to start out with. What you should take with your from this paper is that it is essential to adopt the right metrics that match your situation, as metrics are only 'right' when they are actionable. The data which you gather should naturally lead to actions that influence what was measured, if needed. Another key thing to note is that metrics can be interpreted in a variety of ways. This makes it necessary to combine data from several metrics in order to be able to draw the most accurate conclusions. The Goal Question Metric model introduced in this whitepaper can help you to pick the right metrics. We encourage everyone who's involved with Scrum on a daily basis to apply metrics to their Scrum processes, after all, the numbers tell the tale.

Appendices

Appendix A, Scrum Metrics for Satisfaction

Metric	Team Member Happiness
Parameter	The Happiness of the Scrum Team Members.
Unit of Measurement	Happiness is commonly measured on a linear scale that ranges from 1-5, where 1 is very unhappy and 5 is very happy.

Metric	Net Promotor Score (NPS)
Parameter	Likelihood of recommendation of services to others.
Unit of Measurement	NPS is measured on a 10 point scale.

Appendix B, Scrum Metrics for Maturity

Metric	Ratio of Successful Sprints
Parameter	The degree to which the team is able to meet their commitment.
Unit of Measurement	We measure the Ratio of Successful Sprints as a percentage by dividing the number of successful Sprints by the total amount of Sprints.

Metric	Focus Factor
Parameter	The percentage of the team's effort that results in finished stories.
Unit of Measurement	We measure the focus factor as a percentage by dividing the Velocity by the team's Work Capacity.

Metric	Estimation Accuracy
Parameter	The ability of teams to accurately estimate their work.
Unit of Measurement	We measure the Estimation Accuracy as a percentage by dividing the estimated amount of effort by the actual amount of effort.

Metric	Reliability
Parameter	The ability of teams to meet the Story Points they committed to for a Sprint.
Unit of Measurement	The Reliability is measured as a percentage by dividing the amount of committed Story Points by the amount of earned Story Points.

Appendix C, Scrum Metrics for Alignment

Metric	Relative Return On Investment (Relative ROI)
Parameter	The value of the completed work compared to the invested costs, here the amount of Story Points.
Unit of Measurement	A amount in Value points per Story Point.

Metric	Total Business Value Earned
Parameter	The Total Business Value Earned is an accumulation of all the Business Value that was earned during a sprint.
Unit of Measurement	An amount in Value Points.

Metric	End-to-End metric
Parameter	The amount of effort spent on specific components of the spftware product.
Unit of Measurement	An amount in Value Points.

Appendix D, Scrum Metrics for Quantity

Metric	Velocity
Parameter	The velocity is a relative measure of the amount of work delivered during a sprint.
Unit of Measurement	We measure the Velocity in Story Points.

Metric	Process Efficiency
Parameter	The Process Efficiency shows how efficient the team members spend their time working on committed stories.
Unit of Measurement	The Process Efficiency is measured as a percentage by dividing the actual work time by the calendar time.

Metric	Sprint Burndown
Parameter	The Sprint Burndown shows how much work the team has left versus the ideal amount of work remaining.
Unit of Measurement	We measure the Burndown in Story Points (or hours). Every day, the remaining amount of Story Points is updated.

Metric	Release Burnup
Parameter	The Release Burnup show how much a team has left to complete a release versus the ideal amount of work remaining.
Unit of Measurement	We measure the Release Burnup in Story Points (or hours). At the end of every Sprint the Release Burnup is updated.

Appendix E, Scrum Metrics for Quality

Metric	Defect Count
Parameter	The amount of defects in the software.
Unit of Measurement	We calculate the amount of identified bugs in the Backlog.

Metric	Severity of Faults
Parameter	We measure the severity of the fixed defects.
Unit of Measurement	We divide the faults into three different categories: Minor faults, major faults and critical faults. The Scrum team itself can categorize these faults.