ConnectALL

The Value Stream Metrics Playbook

SCOPE & FOCUS

ConnectALL exists to help organizations achieve higher levels of agility, traceability, predictability and velocity. We do this by connecting people, processes and technology across the software development and delivery value stream, enabling companies to align digital initiatives to business outcomes and improve the speed at which they deliver software. ConnectALL's value stream management solutions and services allow companies to see, measure and automate their software delivery value streams. This guide is a supplement to ConnectALL's value stream mapping & assessment service.

Part of ConnectALL's comprehensive value stream management offerings, ConnectALL's Value Stream Insights is a customizable, packaged framework of metrics, analytics, and visualizations designed to help you accelerate the flow of business value through your value stream. Including an extensible data model and general purpose analytics and visualization application, Insights can graph any metric that you have data for. The ConnectALL's Value Stream Management Platform can pull information from any system with a ReST interface or a database. Further, additional data can be placed into the Insights database via other mechanisms. However, ConnectALL's focus is on software development, Product Management, DevOps, and IT operations. Therefore, this guide's focus is on the software delivery ecosystem and metrics that support the continuous improvement of such systems.

There are an endless number of measures and metrics. We have no intention of cataloging all of them. Nor do we claim to know "the n must-have metrics" because that depends on the "for what." The recommended set of metrics for a large enterprise adopting an agile methodology would not be the same set for an organization trying to adopt a DevOps mindset, which would not be the same set that a small agile team would use to diagnose and improve their own processes. A portfolio team focused on business value and time to market would have yet another set.

Rather, our intent is to spell out an approach, Goal-Question-Metric (GQM), to decide what to measure and to give a sufficient number of examples to get started quickly. And also to catalog many common metrics.

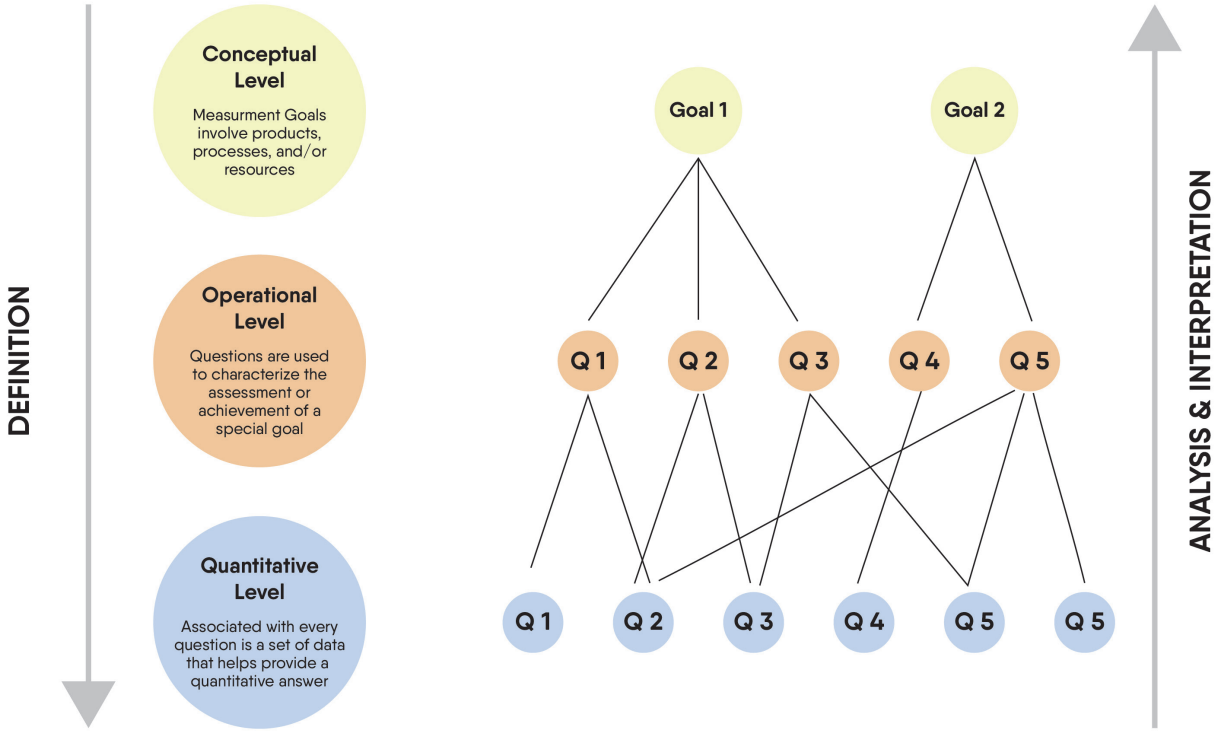
Although this guide addresses financial matters as it relates to the software delivery ecosystem, metrics beyond the scope of that ecosystem are outside of the scope of this guide. For example, this guide does not address corporate finance, CAPEX/OPEX, cash flow, profitability, pirate metrics (customer acquisition, activation, retention, revenue, referral), customer satisfaction, net promoter score, product/market fit, shopping cart abandonment, employee retention, and the like. Such data can be brought into the system and included in the Insights dashboard, but this guide's focus is on recommending metrics that pertain to improving software development and delivery.

TABLE OF CONTENTS

HOW TO DECIDE
WHAT TO MEASURE

When deciding what to measure, the place to start is with a goal. First, ask yourself what outcomes you are after; your goals. Then consider what is needed to meet those goals. And finally, what metrics indicate whether you have what you need. You may recognize this as the Goal-Question-Metric (GQM) approach.

GQM is based on the theory that all measurements should be goal-oriented. There should be some rationale and need for collecting measurements, rather than collecting for the sake of collecting. Each measurement collected must inform one of the goals. Questions are then derived to help refine and articulate the goal. Questions specify what we need to know that would indicate whether the goal is being achieved. Finally, choose metrics that answer the questions in a quantifiable manner. Leading indicators are preferred, but you can use trailing indicators when necessary.



Based on Basili, V. R Caldiera, G., and Rombach, H.D. (1992) *The Goal Question Metric Approach*

The goals you chose should come from an analysis of your value stream. What problems are you encountering? What do your users, customers, or support personnel complain about? What improvements does the business need from the IT or software development organization? What do your existing metrics indicate?

Goals should change over time. A metrics program should drive improvements. As the system improves, reevaluate the system and decide on new goals. Avoid collecting more and more and more metrics over time. Retire old metrics that no longer answer the questions that pertain to your new goals. Maintain only a small set of the most valuable metrics so that the organization will be able to focus on the current improvement goals.

This guide suggests typical goals, suggests questions for those goals, and lists metrics to answer those questions. At the end of this guide are definitions for most of the metrics mentioned in this guide. Where necessary, this guide gives tips on how to measure or collect such data.

Remember that the objective is not to implement all of these metrics. Start with a goal, choose a few questions that if answered would inform your progress to that goal or that would be a good diagnostic, then select a few metrics that would answer those questions. ConnectALL's Consulting & Services organization will gladly assist you with that effort.

TYPICAL GOALS

At the highest level, as it pertains to software development and operations, our clients tend to care about predictability, early ROI, fast time to market, improved quality, or lower cost.

Predictability Questions

Predictability seems to be paramount. Companies want teams to get good at making and keeping promises, consistently delivering working, tested, remediated code at the end of each sprint. A team that is not predictable isn't "bad" – they just aren't predictable. Without stable, predictable teams, we can't have stable, predictable programs, particularly when there are multiple dependencies between teams.

Can the team meet its sprint and release commitments? Can they deliver the functionality they intended each sprint or release? Should we trust them?

Metrics
- Story completion ratio
- Point completion ratio
- Velocity variance
- Throughput variance
- Blockers
- Missed release date history
- Due-date performance

Will the team meet their SLAs?
The following metrics can be useful to monitor predictability-related SLAs such as MTTR and due-date performance.

Metrics
- Lead time for production issues (Mean Time to Repair (MTTR))
- Flow time for funded or approved enhancement requests
- Throughput variance
- Blockers

Do the engineering teams and product owner teams have everything they need to perform the work?

Metrics
- Environment availability
- Team-member availability
- Blockers
- Feature roadmap visibility
- Ready backlog
- % ready backlog with open dependencies (a leading indicator)

Is the team confident that they can deliver the requested functionality and meet the release commitment?

Metrics
- Release confidence

Is the team's throughput or velocity stable?

Metrics
- Velocity variation
- Throughput variation

Can the team control their WIP? Can we discourage excessive context switching?
WIP is an abbreviation for Work In Process, or Work in Progress.

Metrics
- WIP to Throughput ratio
- Team member WIP or pair WIP
- The average sprint backlog item cycle time from In Progress to Done

These metrics are related via Little's Law. The inability to control WIP and cycle time in sprint will increase the likelihood of missing a sprint commitment, leading to throughput variation and lack of predictability. The inability to control WIP at higher levels, such as for features or epics, will increase the lead/flow time for those items, decrease predictability, increase risk (i.e. of changing requirements, priorities and competition), and could decrease quality.

Is the next release on track to be delivered on schedule as planned?
If you have to ask this question, consider whether your releases, sprints, and epics are too big. Nevertheless, use traditional project management techniques to answer this question. Compare percent complete versus the percentage of time elapsed. A release burn-up chart is a great visual indicator. Another effective measure is release confidence.

Metrics & charts
- Release confidence
- Release burn-up chart

Are we managing risks?

Metrics
- Risk score

Are we controlling scope?
Strike a balance between trying to know everything in advance, preventing change, and over planning on one end, and under planning on the other end. Look for an appropriate response to change.

Metrics
- Unplanned work ratio
- Investment mix
- Epic effort estimate versus actual

Are our epic effort estimates good? Are we able to constrain work to budget?

Metrics
- Epic effort estimate versus actual

Early ROI, Time to Market Questions

A lean principle is to favor small batch sizes: smaller epics, smaller releases, smaller user stories, shorter sprints. Smaller items flow through the value stream more quickly, have fewer dependencies and blockers, and are less complex to code, test and debug. These dynamics allow for a faster time to value.

Can the team frequently deliver working, tested, remediated code?

Metrics

- Lead time
- Flow time
- Epic size (effort estimate)
- Release/deployment frequency
- Sprint duration

As for all metrics, carefully choose which classes of work to measure. Usually, time to market only matters for certain classes of work, or it's only a useful indicator for certain classes of work. Flow time for epics is useful if epics represent valuable and minimal increments of prioritized business value. You might have a prioritized backlog of epics, so since some epics wait in the backlog behind higher priority work, flow time is more appropriate than lead time.

If you have a class of work in which certain customer requests need to move quickly from the customers' point of view, be able to identify just that class of work in your systems and use lead time. For example, you may need to distinguish those priority customer requests that need to be delivered quickly versus all other customer requests.

Technical Quality, Availability and Infrastructure & Operations (I&O) Questions

Can the process catch issues?

Metrics
- Escaped defects
- Production impact, latent defects

Are we able to deliver value?

Metrics
- Business value to maintenance
- Investment mix

Are defects being addressed in a timely manner?

Metrics
- Defect aging and defect backlog

Is the code testable, malleable, and maintainable? Are we incurring technical debt?

Metrics
- Cyclomatic complexity
- Code coverage
- Investment mix

Are we meeting our uptime expectations?

Metrics
- Uptime
- Impacted minutes
- Mean Time to Repair (MTTR) (as a diagnostic)

Are we providing good availability?

Metrics
- Planned downtime
- Impacted minutes

Is the system performing as expected?
- Response time
- Memory & CPU utilization as a diagnostic metric

Are we able to recover quickly?
- Mean Time to Repair (MTTR)

Lower Cost Questions

There is enough truth to the maxim "you get what you pay for" that average cost per headcount is not an ideal measure. Nor should organizations compare velocity (story points) per person or points per team. Nor lines of code. Function Point (FP) Counting is the best approach to comparing the output of multiple teams, but requires a trained and experienced FP counting professional, stable teams, and comparably sized projects. FP counting doesn't work well for ongoing products being maintained and enhanced with small agile user stories.

Comparing your IT Spend to others in your industry can be informative, but remember that investing in IT can be a good strategy and give a company a competitive edge. Such a metric is rarely tooled up in a metrics dashboard, but is often evaluated manually on a quarterly or annual basis using information from outside analysts.

Can we control scope?

Metrics
- Unplanned work

Can we release a minimum viable product?

Metrics
- Unplanned work
- Flow time (for epics)
- Batch size (stories per epic)

Are we wasting time?

Metrics
- Abandoned work

Are we over or under spending on maintenance?

Metrics
- Investment mix
- Supported release WIP

Investment & Financial Questions

Are we actually investing as intended?

Metrics

- Investment mix

Are we making good investment decisions?
In his book Product Development Flow, Don Reinertsen says if you quantify only one thing, quantify cost of delay. Make prioritization decisions based on cost of delay divided by duration (CD3), a Weighted Shortest Job First (WSJF) approach. If CD3 is used for prioritization, it wouldn't likely appear on a metrics dashboard.

After delivering an epic, after it has been in production long enough to evaluate the results, it's good for the product, program, or portfolio team to evaluate the effectiveness of that epic. Did it deliver the intended result? Was the decision making effective? What should we do differently going forward? If the process evaluates such questions for every epic, there may be no need for a metric. Nevertheless:

Metrics

- Planned Outcomes Score

METRIC DEFINITIONS

This chapter is a large glossary of metrics, grouped by type of metric. ConnectALL does not recommend starting out by perusing this list. Although skimming through this list might give you useful thoughts, the best approach is top-down, using the GQM method explained above.

Lean Metrics

Several lean metrics are measures of time, and are thus measured and reported similarly and have many of the same usage concerns and modes of misuse: Lead Time, Flow Time, Cycle Time, Process Time, Queue Time, Non-Value-Added Time, Blocked Time and Wait Time. This guide gives a longer discussion of these issues under Lead Time, and a shorter treatment of the others.

Lead Time (80th Percentile, Variation & Trend)

Lead time for a given class of work is the duration between when the request was made and when the solution is available to the requestor. Lead time is always from the customer's or end user's perspective. For only certain classes of work will you want to use lead time. For other classes of work, flow time or cycle time will be more appropriate.

While it may be okay to use average lead time in order to see if you have an improving trend, we strongly advise against using the average because someone will misinterpret or misuse the metric. Instead, use a percentile, such as the 80th percentile. For illustration, the 25th percentile is the point at which 25% of the observations fall below that point. The 80th percentile of your lead time observations (measures) is the point at which 80% of your historical lead time observations fall below that point. The 50th percentile is not always the median, but for the sake of this guide we can say it's close.

Average, however, could be materially off. It's worse to use the average than the median because the average can be thrown off by outliers more than would the median and 80th percentile. If you are good enough with statistics to correctly identify and remove outliers, that's great, but few people do that at all, much less with statistical precision, and it's really not necessary for most IT work.

Again, it may be okay to use the median for monitoring the trend -- to see if the median is improving. But using it for forecasting or expectation-setting would be bad. You see, when using median, 50% of observations took less time, but 50% took longer. You wouldn't want to tell a customer that he has a 50/50 chance of getting a fix in two weeks. Telling them they have an 80% probability of getting a fix in three weeks, in my experience, is more palatable. You want to be able to tell your customers or marketing or management or support or program management that "80% of the time we resolve this kind of issue in n weeks." Most people are happy with those odds. Anything higher takes in too much of the "long tail" of the distribution and makes forecasts not be terribly useful for planning. Anything less increases risk of disappointment.

You don't need a ton of data. Depending on scale (number of observations and how long lead time actually is), data beyond 24 weeks is most likely out of date.

What to do with your Lead Time chart? I publish my "80% lead time expectation". I talk about it with the people who are anxiously waiting for the delivery. I talk about it with my engineering team. I talk about it with my management team, PMO, project managers and program managers. I talk about it with my lean and agile coaches and consultants and Scrum Masters. I talk about it with my team leads. I want everyone in the loop and on board with the improvement goal. I use it to explain how certain behavior, such as expedites and high WIP, work against improving the lead time expectation.

What to look for:

- Look at your bar chart showing the changes in your lead time expectation over time. See if it's moving in the right direction. Use A3s, Toyota Kata, lean principles, and systems thinking to improve the system. Engage your upstream and downstream neighbors in the improvement process and in making process policies explicit.
- Fixes for production bugs should have a short lead time. If the average lead time is unsatisfactory, look for an improving trend.
- For predictability, look for a narrowing spread (standard deviation) on a control chart.
- For forecasting, use monte carlo simulation.
- For expectation setting for individual items, use the 80th percentile.

Flow Time, Process Time (80th Percentile, Variation & Trend)

Sometimes called process time, flow time for a given item is the duration between when the request was approved or when the work was started to the time that the work was completed. Flow time is an internally focused measure, from the perspective of the software development value stream.

Whereas lead time is from the customer's perspective, flow time is focused on the software delivery value stream, and excludes time that the item sits in the backlog waiting to be released into the software development flow. It may also exclude time after a build is complete or release is available, but not yet installed at the customer's location. That is, this metric excludes factors outside of the control of the software development process.

Just like for lead time, we recommend using the 80th percentile instead of the average or mean lead time.

What to look for:

- Most companies want a short flow time for epics, for approved customer requests, or other enhancements. If the flow time is unsatisfactory, look for an improving trend.
- For predictability, look for a narrowing spread (standard deviation) on a control chart.
- For forecasting, use monte carlo simulation.
- For expectation setting for individual items, use the 80th percentile.

Cycle Time (Variation & Trend)

Average cycle time for a given class of work is the average duration between any two states in the value stream. Stated differently, it's the average time between point A and point B.

Cycle time is an internally focused measure, from the perspective of the software development value stream. It is usually used to examine a particular phase of the value stream. For example, the development cycle time or QA cycle time can be a useful diagnostic for stories, defects, and epics. If not pair-programming, the peer-review cycle time might be of interest.

'Median' won't tell you if you have a problem with some extremely high or low values. 'Average' doesn't really help you with that either. Nor will the 80th percentile. A control chart is a very good visualization of what's really happening with your cycle time.

What to look for:

- Agile teams, or those teams using an iterative process, should want a short cycle time for all backlog items in their sprint. Agile organizations should also want a short cycle time for their epics to be "in progress." If the average cycle time is unsatisfactory, look for an improving trend.
- For predictability, look for a narrowing spread (standard deviation) on a control chart.

Queue Time, Non-Value-Added Time, Wait Time (Average & Trend)

Queue time is a cycle time measure. Queue time is the amount of time work sits waiting for actions to be performed. This could be the time for a single queue, or the sum of times waiting in multiple queues across a value stream. This could be the average wait time per ticket, or average wait time per month, or ratio of wait time to value-added time. The latter (ratio of wait time to value-added time) is best if you have the data. The first (average time per ticket) is susceptible to changing work sizes and splitting stories. Such behavior can make the metric improve without improving the system's wait time per feature. But wait time per month might be susceptible (up or down) to fluctuating throughput due to fluctuations in team member availability.

Many organizations do not model all of their significant wait states in their kanbans or ALM tooling and as a result cannot see the magnitude of delay. A value stream mapping session with a careful eye out for queues and delays can help identify additional states to add to your kanban.

What to look for:

- Queue time is crucial for items that must move quickly through the value stream, such as production issues. Queue time is usually less useful for stories waiting in a release or PI backlog.

Blocked Time (Average & Trend)

Similar to cycle time, time blocked is the average amount of time that items stay blocked by something or someone outside of the team that is being blocked. Blocked time is a measure of the negative impact of dependencies outside of the team. This metric can include only those items that were blocked, or can be averaged over all items (i.e., including those that were never blocked).

What to look for:

- Blocked work interrupts flow, breaks concentration, and introduces delays. Time blocked may indicate work that wasn't sufficiently ready and shouldn't have been started. It may indicate that more backlog refinement or more rigorous dependency management is needed.

Blockers (Average & Trend)

The blockers metric is a count of blocking events that happen over a period of time, such as per month or per sprint. A blocking event is something that happens outside of the team's control that impacts the team's ability to move forward.

The blockers metric can be used as an alternative to, or as a compliment to, the blocked time metric. Depending on your source data, one of these might be easier to collect than the other.

What to look for:

- Blocked work interrupts flow, breaks concentration, and introduces delays. Blocked work may indicate that the item wasn't sufficiently ready and shouldn't have been started. It may indicate that more backlog refinement or more rigorous dependency management is needed.
- Consider whether to count or to ignore blockers that do not impact the outcome of the sprint. Weigh the cost tradeoff between more thorough refinement and dependency management versus blocked work. If a Scrum team is able to remove the blocker and finish the story or sprint as planned, then maybe that blocker doesn't need to be counted. If this is the case, then count blockers at the end of the sprint. (Count items that remain blocked at the end of the sprint.)

WIP to Throughput (Ratio & Trend)

Lead time, flow time, and cycle time are negatively impacted by increasing amounts of work in process (WIP). Shoving more work into the system slows everything down. Building a large inventory of untested code typically increases the costs and time associated with fixing defects. Building up too much ready backlog can lead to wasted effort when priorities, requirement details, or the market changes.

An appropriate level of WIP is relative to the average throughput, and to the definition of "in progress" necessitated by the specific GQM in question, and the people or team involved. I suggested two different scopes for "in progress" in the prior paragraph: one included just development and QA, which would be useful for diagnosing testing backlog or lag. The other also included PO/BA and team efforts to ready a backlog. "In progress" for epics would naturally include a larger period of time than "in progress" for user stories.

What to look for:

- Large agile organizations trying to deploy every 2 weeks should not have more than 6 weeks' worth of throughput (user stories) active in a team from Ready to Delivered. That's 3 or 4 weeks of ready backlog, 2 weeks for the current sprint, and maybe a week of post-sprint verification. That would be a ratio of 3. Two weeks of ready backlog might be sufficient for a smaller, more nimble organization with no dependencies and little structure or overhead. A team practicing continuous deployment should have an even lower expectation for this ratio. If your ratio is high, look for an improving trend.
- At a sprint level, the WIP to throughput ratio should be much less than 1. For example, if a team has an average throughput of 20 items per sprint and if they, on average, have 20 items in progress (actually being developed), that means about all of their work is in progress for almost the whole duration of the sprint. The number of team members is also a factor, but to put some bounds on it, 10% is probably very good and 50% is not that good.
- A related measure is the ratio of the percentage of planned work completed to the percentage of time consumed. For example, with iterative development, 80% of the story points should be completed by the time the iteration is 80% through.

WIP (Quantity)

We previously discussed the WIP to Throughput Ratio, but sometimes the ratio isn't needed. Sometimes the raw amount of WIP is a sufficient metric.

What to look for:

- The number of epics or initiatives each team is working on should be 1. On your metrics dashboard, list teams with more than 1 epic in progress. But remember, it's not the team's fault. Fix the system. Don't blame the team. For any given software product, the organization producing it should strive to have a very small number of epics in progress (actually in development), typically only 1 or 2.
- The number of items being worked on per individual should be one per individual, or less. It's usually easier to gather this data at a team level. At a team level, WIP should be less than the number of individuals. Encourage working together. If you are pair-programming most of the time, your WIP could be less than the number of pairs: With TDD, good test coverage, and good Continuous-Integration practices, you should be able to get multiple pairs on one user story.
- The number of open sprints should be 1 per team.
- You may want to monitor the number of releases being maintained (fixed, patched, level 3 support) or the number of releases being supported (help desk, service desk, level 1 and 2 support).

Batch Size

Another lean principle is to favor small batch sizes: smaller epics, smaller releases, smaller user stories, shorter sprints. Smaller items flow through the value stream more quickly, have fewer dependencies and blockers, and are less complex to code, test and debug. These dynamics allow for a faster time to value.

Two batch size metrics, epic size and sprint duration, are discussed below.

Epic Size (Average & Trend)

Epic size could be a measure of effort estimate, as in story points or team-months, and can also be measured in terms of the number of stories the epic contains.

As a measure of estimated effort, there is inherent inaccuracy in this metric. Such error can be offset by using a measure of actual time, such as flow time. Nevertheless, epic size is a leading indicator whereas flow time is a trailing indicator. Therefore, it can be valuable to accept the error in exchange for an early indicator of what your future flow time might become. Also, if your epic size is trending up (to larger epics), expect some other metrics to also worsen in the future, such as blockers and quality.

Sprint Duration (Quantity)

Like release/deployment frequency, if sprint duration is consistent across your organization, stable (not variable), and known, then there is probably no need to automate the collection and display of this metric. This metric would be useful if you are in a very large organization with diverse sprint lengths that is in an effort to shorten and standardize.

As of 2020, month-sized sprints have been falling out of favor for many years. The two-week sprint duration still seems to be very common.

Abandoned Work (Quantity)

Abandoned work is any item (epic, feature, story) that is thrown away or discarded. A small amount of abandoned work can be healthy, if it's abandoned early enough. Abandoning items earlier in the value stream is, of course, much better than abandoning them in later phases. It's much worse to throw away some developed feature once it is in QA. It's much better to throw it away before any coding is done. And it's even better if it can be abandoned before it is fully "ready" (meeting the team's definition of ready) as we don't want to waste the Product Owner Team's time either.

Report the raw number of items abandoned by phase. It's usually sufficient to record whether the item was abandoned before being ready, after being ready but before development starts, or after development started.

Predictability Metrics

Throughput (Variation & Trend)

Throughput is the amount of work a team can complete in a defined time period, typically a month. Average throughput can be used for forecasting, but monte carlo simulation is a better approach.

The throughput variation metric helps teams become stable in their performance. This will encourage organizations to manage risks and dependencies before starting the work. Recent throughput within 20% of average throughput is good. We want to see a reduction in the standard deviation of the throughput over time.

Variation can be computed as the standard deviation divided by the average. We recommend including data from at least the last 4 periods, but no more than 6 months of data.

What to look for:

- Organizations that are after predictability should look for low throughput variation (lower than 0.20), and a trend that is stable or gradually increasing.

Velocity (Variation & Trend)

Velocity is an alternative measure of throughput. Velocity is the measure of story points completed in a sprint. Average velocity can be used for forecasting, but monte carlo simulation is a better approach.

The velocity variation metric helps teams become stable in their performance. This will encourage organizations to manage risks and dependencies ahead of the sprints, and to not overcommit within the sprint. Recent velocity within 20% of average velocity is good. We want to see a reduction in the standard deviation of the velocity over time.

Variation can be computed as the standard deviation divided by the average. We recommend including data from at least the last 4 periods, but no more than 6 months of data.

What to look for:

- Organizations that are after predictability should look for low velocity variation (lower than 0.20), and a trend that is stable or gradually increasing.

Story and Point Completion Ratio

These ratios are computed as:

- Number of Committed Stories Delivered / Number of Committed Stories
- Number of Committed Points Delivered / Number of Committed Points

This metric helps teams become predictable in their estimating and sprint planning. It encourages smaller stories and more effort in getting work ready prior to the sprint. We like to see delivered points and stories to be within 10% of the commitment.

Team-Member Availability (Ratio & Trend)

Team-member availability ratio is computed as: headcount available / headcount expected

You may want to quantify the extent to which planned team members aren't available. Stability is critical for teams to be able to make and keep release commitments. When people are pulled across multiple teams – or are not available as planned – it is unlikely that the team will be able to deliver predictably. We like to see this be within 10% of the plan.

Release Confidence

A great leading indicator of the likelihood of meeting a release date (time and scope) is to just ask the people involved. Use the team's insight and their own sense of their record of performance to evaluate the team's confidence that the release objectives can be achieved.

You can use a simple 3-, 4-, or 5-point scale ranging from very unconfident to very confident. You can poll everyone involved with an online survey, or just get a consensus vote from certain people such as a technical lead, test lead, and scrum master. If a team has heavy dependencies, they should include a vote from the agile project manager of the team handling the dependencies. You can ask this just once, at the end of PI planning, or more frequently, such as each sprint or weekly.

Risk Score (Trend)

A common way to quantify risks is to identify risks, score each with a probability and impact, multiply those two scores, add up all the values, and track the trend over the course of the release. And, of course, work the risks, preferably early.

An (better) alternative is to evaluate the overall risk of a program across a small number of dimensions, such as technical risk, business risk, quality risk, schedule risk and organizational risk. Risk management is outside the scope of this document.

Unplanned Work (Ratio & Trend)

Some amount of unplanned work added to a release or to the top of the backlog can indicate appropriate response to learning or to a fast moving/shifting competitive landscape. Too much unplanned work can cause issues when predictability is needed, such as when other groups like operations, marketing, or sales are involved with roll out plans.

An unplanned work metric is useful once you have determined that your organization or some team is too far to one extreme and needs to move to a more moderate position. Tracking the trend can give you an indication of movement in the right (or wrong) direction. This metric can also be useful as a diagnostic tool if you are having trouble with a lack of predictability and other metrics haven't helped resolve the issue.

Don't let this metric (or any metric) outlive its usefulness or take on a life of its own.

Feature Roadmap Visibility (Quantity)

Program teams (sometimes called product owner teams or the 3 amigos) need to make work ready ahead of the team doing the work. For program teams to make work ready, there needs to be a roadmap of prioritized features sufficient to drive the program team's work. How far out this planning horizon needs to be depends on many factors, such as: the stability of the market or industry; prioritization stability; needed agility (the ability to respond quickly to changing circumstances); the level of dependencies the program team has on outside SMEs; the level of dependencies the engineering team doing the work will have on outside teams or SMEs; and the lead time needed to decompose work into ready stories. The program team needs to plan far enough ahead so that work doesn't get blocked and so that the engineering teams don't get starved for work or have to work on stuff that really isn't ready, but not so far ahead so as to create waste due to change or uncertainty.

There are many ways to measure this. An easy approach is to "right-size" features so that they are each 1, 2, or 3 sprints big. You could also use "team-weeks." Record the estimates (budget) in your tooling.

Decide what the target range for roadmap visibility should be. In the dashboard, list the teams that are operating outside of the target range.

Feature Roadmap Visibility is a measure of ready backlog of features. Ready Backlog (below) refers to the ready backlog of user stories.

Ready Backlog (Quantity)

Software development teams need a steady stream of ready work fed to them. They also need some visibility into work that is coming up in the near future for a myriad of reasons: staffing, planned time off, holidays, special skill or SME availability; to plan for outside dependencies; to make informed design decisions. Too little ready backlog can lead to scrambling to make work ready, unclear stories or acceptance criteria, and fewer options when the sequencing of work needs to be adjusted. Too much ready backlog can lead to waste, especially in the face of shifting priorities or changing requirements or market needs.

One sprint, or two-weeks worth, of ready backlog is on the low side. Two sprints, or one month, of ready backlog is about right in our experience. Your needs may differ from these guidelines.

How much is "one sprint worth" or "two-weeks worth", of course, depends on team throughput or velocity and varies by team. Decide on a target range, such as 2- to 3-sprints worth. Have a means to mark stories as "ready" in your tooling. Divide the size of the ready backlog by the throughput and compare against the target range. List the teams that are operating outside of the target range.

Epic Effort-Estimate Versus Actual

Don't compare story point estimates to actual hours. Estimating duration for epics, however, is a good practice. Don Reinertsen, in his book Product Development Flow, says to quantify the cost of delay and to take expected duration into account for prioritization. Estimate epic duration in terms of sprints or team-months. If you estimate an epic to require, say, 3 sprints, and that epic is on the near-term roadmap, that estimate then becomes a budget. The product or program team should endeavor to constrain the work to that budget.

Quality Metrics

Production Impact, Latent Defects (Quantity & Trend)

Production impact, sometimes called latent defects, are issues over a certain severity (i.e., ignoring minor, cosmetic) that are found in production that were introduced per release or per month. If it is difficult to identify which release or build introduced the defect, it is often sufficient to attribute a bug to the product version in which it was found.

Graph the quantity over time so that the history and trend can be observed. How much is acceptable depends on many factors, such as the number of users and the definition of severity levels. Defining a target for each application is a good practice.

Escaped Defects (Quantity & Trend)

Similar to production impact, the escaped defects metric is a measure of issues over a certain severity. Escaped defects, however, are caught before production, but after the end of the sprint in which they were introduced. Such defects should have been caught during the sprint, but were caught during later phases, such as during integration testing or regression testing. They have "escaped the sprint".

Graph the quantity per sprint so that the history and trend can be observed. Defining a target is a good practice.

Defect Aging and Defect Backlog (Trend)

Organizations don't want to build up a backlog of unresolved defects. The backlog of defects should not be increasing for any severity level, except perhaps for the lowest severity. The absolute number of the highest severity defects should be extremely low.

Also, the age of the defects in the highest category should not be increasing. If it is, then there may be a process or prioritization issue. It may be that certain issues are incorrectly classified.

Uptime (Percentage & Trend)

Likely measured on a weekly or monthly basis, uptime is relative to the planned (scheduled) application availability. This metric is the total available minutes divided by the planned available minutes. A graph showing history and trend is sufficient, but adding the SLA threshold is a very good practice.

Planned Downtime (Quantity & Trend)

Graph the planned downtime (such as in minutes per month) for all applications or services over time in order to see the trend and determine if planned downtime is improving or if it is out of line for some application or service.

Impacted Minutes (Quantity & Trend)

Likely measured on a weekly or monthly basis, this is a measure of time that a service or application is up, but is impacted by high severity issues. This metric is the total time in minutes that the system was up but impacted. A graph showing history and trend is sufficient, but adding the SLA threshold is a very good practice.

Adding the number of such incidents to the graph can be informative.

See also (related metrics)

- Investment mix
- Business value to maintenance

Investment & Financial Metrics

Investment Mix (Percentages)

Many organizations plan to invest across multiple categories or investment themes, such as maintenance, innovation, enhancements, roadmap, and usability. Such organizations should check their actual investment against the plan on a monthly or quarterly basis. Getting a complete picture may require careful categorization of all user stories and defects; that is, at a low work item level. Other organizations might only be interested in such categorization at the feature or epic level. In any event, an investment mix pie chart is ideal.

Business Value to Maintenance (Ratio)

Organizations, of course, want their teams to deliver lots of business value, but maintenance shouldn't be neglected. Neglecting maintenance can cause teams to slow down, negatively affecting the rate of business value delivery. Product lifecycle stage, average effort required for each class of work item, and numerous other factors influence what an acceptable business value to maintenance ratio should be.

The CAPEX to OPEX ratio is similar, though not addressed in this guide.

A simplistic approach to this metric would be to compare the count of user stories versus the count of defects. However, not all user stories provide new business value and not all defects should be categorized as maintenance. Most defects should be considered rework instead of maintenance. A better approach would be to categorize all work according to a set of investment themes. Distinguish between rework, maintenance, enhancements, and the like.

Planned Outcomes Score

Organizations that use an opportunity canvas or epic canvas approach have questions on the canvas about the intended outcomes expressed in terms of the business need (revenue, market share, shopping cart abandonment, etc.) that should be moved and by how much. How to score such results is organizationally specific.

Others

- Resource spend by resource type (dev, QA, BA, architecture, manager)
- Financial variance (plan versus actual)

DevOps, Build & DORA Metrics

Some of the Quality Metrics listed above could arguably be listed in this section instead: uptime, planned downtime, impacted minutes. Such metrics reflect quality, and are not repeated here.

Release/Deployment Frequency (DF)

Release frequency is a gating factor for time to market and early ROI. Companies can gain value from developed software only if it's released (deployed) into production. Companies whose releases are a big deal and a subject for much project management, those who have infrequent releases and large epics, those who bring everyone together for big-room PI planning, probably know their release frequency and don't need to automate the collection and display of such metrics. Automating the collection and display of this metric makes sense when different teams have different frequencies, once frequency is more often than quarterly, and when there is a strong desire to increase the frequency across all teams. (You get what you measure.)

Mean Lead Time for Changes (MLT)

Also known as Change Lead Time, this metric is sometimes defined as: mean lead time for any change, defect or enhancement to go from idea to production. Others define it as the mean (cycle) time from code commit to production. See Lead Time and Cycle Time above.

Mean Time To Recover (MTTR)

Mean Time To Recover/Repair/Restore represents the average time required to repair a failed component, system, service or defect (i.e,. in production). This is yet another measure of Lead Time, discussed above.

Change Failure Rate (CFR)

The change failure rate is simply the percentage of deployments that fail.

Others

- Pull requests
- Build failures
- Security scans
- Cyclomatic complexity
- Code coverage

For more information visit or contact

www.connectall.com
+1 800 913 7457
sales@connectall.com