Agile can be perceived in different ways: as a manifesto of values or list of principles, as an emphasis on collaboration, as a repackaging of spiral and iterative development concepts, or as the overarching prioritization of adapting to change over following a plan. I’ve been around software development long enough to see several of these shifts in perception and I’ve come to think about all such shifts as reactions to environmental changes, similar to what occurs in nature. Take the flounder, for example. A flounder’s environment is the ocean floor. He lies on his side and has adapted so that both eyes are on one side of his head, because he only cares about feedback from above.

The movement between the different stages of software’s lifecycle used to be very expensive. Compilers ran for hours. Testing was labor-intensive. Distribution of a completed product involved physical media and could take months. In this environment, it was critical to minimize the number of times you went through these costly transitions. Fittingly, the emphasis for feedback was on the process: you could improve the checklists you used to stage-gate each of the transitions, with the hope of reducing rework that crossed backward over these expensive boundaries. Similarly, the success of a project minimally required that it be finished before funding ran out, so there was similar emphasis on the plan feedback.

Then the environment changed. The costs of compilation, testing, and distribution have been driven close to zero. The biggest threat to success is not that you’ll run out of money, but that you’ll miss your market window. “Rework” is no longer verboten. In fact, it’s much better to build something minimally usable – and then “rework” it based upon usage feedback – than it is to try to “build it right the first time.” Like the flounder who no longer needs feedback from below, we no longer value feedback on the process or the plan as much. Our most valuable feedback is feedback on the product.

The old metrics systems we used before Agile had the wrong feedback loop emphasis, so agilists fittingly threw them out. They replaced them with qualitative insight, which works well on smaller teams. But Agile is going through another environmental shift. It’s scaling...
up to larger projects and being adopted by larger organizations. In these environments, qualitative insight alone is insufficient – it must be complemented with appropriate quantitative insight.

I have been working hard to identify the key perspective shifts necessary to successfully introduce measurement into an Agile development environment, and I’ve coined the phrase, “Seven Deadly Sins of Agile Measurement” to capture this learning. Let’s look at the first of these Seven Deadly Sins.

**Sin #1 – Using measurement as a lever to drive someone else’s behavior**

If feedback emphasis is key to the success of Agile, the key to effective Agile measurement is to think of measurement in terms of feedback, not as the traditional lever to motivate behavior. Using measurement as a lever often devolves into keeping score, which is where the dark side of measurement starts.

There is a subtle, but important, distinction between “feedback” and “lever.”

An example will help illustrate this point.

Below is a chart found in an ALM tool. It’s an attempt to borrow a concept from the manufacturing world that I believe has been misapplied to our domain. (There are several problems with this chart, but for Sin #1, I want to highlight the use of the red line and the red dots.)

Feedback is something you seek to improve your own performance. Levers are used to influence others. The difference is in how you use the measure more than the measure itself.”

Each dot on the chart represents a particular user story. How high up on the chart they appear is proportional to how long the story took to be completed. The higher the story, the longer it took. That red “upper control limit” line and those red dots say, “this is bad!” The stories in red took too long; they are literally “out of control.”

What’s going to happen the next time you show this chart? There probably will be fewer red dots, but why? I’d like to think it will be because people have deeply analyzed their process
and made necessary changes, blah-blah-blah ... but what’s more likely is that they will just game the system to make sure they don’t have any red dots. Maybe they’ll split stories artificially instead of where they deliver value. This is bad because it’s wasteful and doesn’t improve the process; but what really hurts is that you’ve now hidden data from yourself. You’re making decisions with a distorted picture of reality.

As an alternative, I propose this chart:

It’s conceptually very similar to the first chart. The primary difference is the lack of a red line and red dots. This visualization is designed to allow teams to explore their evidence, enabling learning and improvement. You can hover over a dot and get the details about what happened, which should enable discussion. You can talk about probabilities to help gauge risk. You’ll see that 95% of all stories finish in 28 days; maybe that will help you make a service-level agreement commitment.

So, the heavenly virtue here is to inspire pull and resist the push. Use metrics as feedback to improve yourself: never as a lever to alter someone else’s behavior.

Sin #2 – Unbalanced metrics

If you try to measure Agile development in a pre-Agile way, you’re bound to fail. We refer to these anti-patterns as the Seven Deadly Sins of Agile Measurement. The second sin has to do with the need for a balanced metrics regimen. The need for this is fairly readily apparent. If you focus on one aspect of performance (say productivity), other aspects will go down (quality, customer satisfaction, etc.).

I recommend that you launch a metrics feedback effort with at least one measure from each of these four areas: 1) Do it fast, 2) Do it right, 3) Do it when expected, and 4) Keep doing it. In the diagram, you can see, we have populated each of these with at least one outcome sub-dimension.

These outcomes dimensions form the foundation of the Software Development Performance Index (SDPI), which quantifies
insights about development work and provides feedback on how process and technology decisions impact a team’s performance. Of the metrics in each of the quadrants shown above (productivity, responsiveness, quality, predictability, customer satisfaction, and employee satisfaction), the first four can be extracted from ALM tools like Rally. The fifth and sixth metrics we recommend you gather via lightweight (maybe even single-question) surveys. (Note: we’re also working on a seventh dimension, what I’m calling the “build the right thing” dimension. Stay tuned.)

With unbalanced metrics, you may not be measuring the right outcomes in the right proportions. The heavenly virtue that complements the sin of unbalanced metrics is, of course, to implement a balanced measurement regimen.

**Sin #3 – Believing that metrics can replace thinking**

Lord Kelvin says, “...when you cannot express it in numbers, you knowledge is of a meagre and unsatisfactory kind...” But your teams hear, “We don’t trust you.”

If you believe that quantitative insight can replace qualitative insight, you are committing Sin #3. One of the primary principles of Agile is to trust the folks closest to the work, the ones with the most context to make decisions. If you think metrics can replace thinking, you are essentially telling your team, “We don’t trust you.” In an Agile world, the sort of quantitative evidence that you get from metrics must complement qualitative insight, rather than seek to replace it.

In the best of circumstances, you can move in a virtuous cycle between qualitative and quantitative. Frequently, you will want to validate a performance hypothesis you have (qualitative insight) by using metrics (quantitative insight). The results of that analysis will lead to more questions. Creating a cycle of hunches that you validate via metrics is very powerful and can lead to huge leaps forward in decision-making. This requires support for ad-hoc analysis. Rally connectors like the Excel plugin as well as APIs, data access toolkits, custom grids, custom reports, and custom dashboard panels all help you achieve this heavenly state. Remember, metrics don’t steer you: they illuminate the road so you can decide which way is the best way to go.

**Sin #4 – Too-costly metrics**

Sometimes it’s too expensive or burdensome to get the exact metric you want. Even when the actual cost – in time it takes for your developers to record extra information that helps calculate a certain metric – is low, the perceived burden can be much greater, especially in an Agile environment. Developers often are your most valuable resources and you do not want to burden them unnecessarily. You should only ask your developers to input manual data if they believe it leads to a measurement whose value exceeds the perceived burden (see Sin #1).
What this means is that in many cases, qualitative insight is your best alternative, but that’s OK, maybe even desirable (see Sin #3). However, you can turn a qualitative perception into quantitative insight via a survey. That’s why two of the SDPI dimensions are customer satisfaction and employee engagement. We recommend that you use a very lightweight metric (maybe one question like net promoter score) for minimal burden while capturing a lot of value.

When assessing the value of a particular metric, make sure you include the “cost” of the perceived burden on your developers to record that metric. Qualitative insights can become valuable quantitative data.

**Sin #5 – Using a convenient metric**

Better measurement leads to better insights, which in turn lead to better decisions and eventually better outcomes. With this measurement chain of effect in mind, many people start by choosing measurements that are easy to acquire. However, measuring what’s easy often can drive the wrong behavior. Let’s use a sports analogy to illustrate this point.

In 2010, Monta Ellis of the Golden State Warriors was the ninth-highest scorer in the NBA. Carmelo Anthony of the Denver Nuggets was the eighth-highest scorer. Measuring individual scoring totals is easy, and you might assume that because these players were prolific scorers, their teams were winning games.

However, it turns out that the more they played, the less their teams won. Scoring is itself a function of two measures: the number of shots taken, and the percentage of those shots that go in the basket. These two “stars” have high measures for the former but low measures for the latter: they are high scorers because they take more shots, but their shooting percentages are quite low. Since they’re not making more opportunities for shots – their teams take almost exactly the same number of shots per game no matter how much they play – they are literally stealing shots from their teammates who might have a better chance of scoring.

So while the flow of learning goes from measures to outcomes, the best way to make measurement decisions is to start by identifying the desired outcomes. That’s why we call this ODIM:

Applying this to our sports analogy, the NBA stars should focus on the outcome of winning more games rather than being high scorers. If they used insights gained from measurements, such as the overall likelihood of the team scoring under various conditions, it would help them make better game-time decisions and achieve the right outcome.
Even the best strategy can fail in the execution, one small decision at a time. Another way to think about measurement in this context is that it provides the feedback to help your people know which decisions best align with your strategy.

**Sin #6 – Bad analysis**

Imagine setting a service level agreement, thinking that you would miss it in just 0.2% of your cases, but in reality you miss it in 1.0% of your cases. If you had allocated $200,000 as a contingency, you’d actually need $1,000,000. Yikes! Though the percentages seem small, you can see that their impact in dollars is not. This kind of bad analysis is actually a fairly easy mistake to make.

Here’s the same chart we discussed in Sin #1 – using measurement as a lever to drive someone else’s behavior – where we looked at how coloring the dots red is a “lever” rather than “feedback”. Each dot represents a user story: the higher the dot, the longer it took to be completed. The upper control limit is supposed to be a 2 standard deviation, or 97.7% threshold, but the data are not normally distributed so it’s really only about a 92% threshold.

Many phenomena in the knowledge-work domain exhibit fat-tail behavior, and the tail is frequently what interests us. Careless statistical analysis will hugely underestimate this. This is just one example of the way in which bad analysis can drive misinformed decisions. The heavenly virtue to go with this sin is to invest in the expertise that enables you to do correct analysis.

**Sin #7 – Forecasting without discussing probability and risk**

I’ve often seen teams get burned by giving their boss a linear projection as the anticipated completion date for their current backlog. Inevitably, the boss takes this as a commitment; after all, they have little other information on which to base their plans. The numbers say this is when we should finish, right? However, there’s a lot of uncertainty in a linear estimate: at best, it’s an average projection with about a 50% degree of certainty. It’s much better to think in terms of probability and shared risk between the team and the stakeholders. I call this “probabilistic forecasting.”
In any innovation process, there’s a limit to what can be known ahead of time. Although software projects start with a vision, Agile development allows software projects to unfold rather than be planned upfront. We can forecast what we know, but the unknowable often causes a larger impact on delivery time. In other fields of commerce, this is understood: insurance companies, for example, have to manage cash-flow in the face of uncertainty about future risks. We accept that risks and delays are a natural part of software innovation, mitigate the associated costs, and forecast accordingly. Linear forecasts of only what is currently known will always fail to give an acceptably accurate result in the face of high uncertainty and risks. Managing uncertainty by measuring how much remains in the project is the only way to know if a linear forecast projection (burn-down) is an accurate prediction.

**Probability**

Probability is how certain an outcome is likely to be. If we throw a common six-sided die and forecast a “three,” the probability is one chance in six: there is only one outcome that matches our forecast. With software projects, it’s more complicated to quantify the odds of any particular result. Hundreds of decisions are made that alter the eventual outcome, each changing the probability of any initial forecast. What may be more important than the forecast date is the likelihood that the forecast will match reality.

For any forecast there are multiple possible outcomes, some more likely than others. Each piece of remaining work for a project has uncertainty. One to five days might be an estimate for one story, five to ten days for another. Forecasting software projects requires computing how the remaining work may play out. This could be as simple as summing the lowest estimate for each piece of work for the most optimistic forecast, and the largest estimate for each piece of work for the most pessimistic forecast. This gives us a range of results, but both ends of the spectrum are unlikely.

A common technique for resolving the chance of values within the optimistic and pessimistic range is called a Monte Carlo simulation. Monte Carlo simulation hypothetically completes the software project hundreds or thousands of times, and then plots the outcomes in a histogram. The probability of any date between the optimistic and pessimistic range is calculated as the number of outcomes on or before a value (dates, in our case). By forecasting in this way, you avoid committing to an overly optimistic date and the odds of achieving a chosen date are clear to all and can be tracked accordingly.

Figure 1 shows one such histogram computed by hypothetically completing a software project – with story estimate ranges – 1,000 times. In this case, the x-axis shows work days to completion. Although the most likely is 70 days, there were some simulated outcomes as high as 77 days. The black line graph shows the cumulative probability that a date will be achieved; for example, there’s 96% likelihood at 73 days.
Forecast and certainty are often easier to understand in table form. The table in Figure 2 shows how knowing different likelihood levels, the last responsible likelihood start date can be computed.

Risk

Monte Carlo simulation accurately portrays how combining multiple pieces of uncertain work gives a range of results, and the relative odds of any one of those results. But often there is often unknowable work or knowable delays that may or may not occur. This is risk. Take, for example, the challenges of unacceptable performance, trouble installing hardware, or waiting on another team or vendor: none of these challenges is certain to occur, but any one or combination of them could delay your project. Often these delays are longer than the uncertainty in our story estimates. A simple forecast based on resource availability often fails to demonstrate the impact of technical and delivery risks. Ignoring these risks, or not proactively managing their likelihood by mitigating their impact, is signing up for disaster. Development teams are left holding the bag, blamed for a flawed initial estimate.

Monte Carlo simulation helps here as well. Simulating the work we know, combined with the impact of risks on delivery time, shows more possible outcomes. The likelihood of one, two, or all risks coming true is dependent on the likelihood of occurrence for
each risk: for example, a risk with a 70% chance of coming true would be included in the simulation 70% of the time. The histogram shown in Figure 3 demonstrates the result of combining the work we know with two risks that each has a 30% chance of coming true. The result is ugly, but interesting. It shows that depending on which risk factors come true (none, either, or both,) the outcome date can be wildly different. Managing the risk likelihoods and impact would eliminate and reduce the high outlier modes, making an outcome on the left side more likely.

(By the way, a histogram like the one shown in Figure 3 is more common than you think – especially in sectors like pharmaceutical drug discovery, where testing progresses through phases. Most drugs fail early laboratory testing; a few move to human testing, and assuming they don’t do more harm than good, even fewer of those actually make it to market. Properly managing risk yields better returns.)

**Key Points**

When forecasting a project, spend some time determining what can go wrong, the impact it would have, and how long that would take to resolve. Typically, the impact of the first three risks on a project will be bigger than the variation of the known work. This is a good point to ponder when setting your developers on the task of detailed work breakdown and estimation.

- All forecasts have multiple possible outcomes, some more likely than others.
- An outcome at the pessimistic or optimistic end of the spectrum is least likely.
- Use probabilistic forecasting techniques like Monte Carlo simulation to identify which results are more likely than others.
- Risk factors probably pose a bigger chance of impact than the uncertainty of any one story estimate, or even the cumulative effect of all story point estimates. Spend more time asking your teams, “What can go wrong?” and less time asking them, “How long will it take to …”

**Learn how Agile can quantifiably improve your software delivery**

Now that you know the seven deadly sins of Agile measurement, it’s time to get real data about the incredible benefits of adopting Agile, make an economic case, and drive better results. Go to rallydev.com/agilemetrics to get started!
About the Author

Larry Maccherone is the Director of Analytics for Rally Software. He has been with Rally since 2009, first as an Agile coach and then as a Product Owner of the analytics team based in our Raleigh office. Prior to his current role, Larry served as a software process coach focusing primarily on Scrum and the SEI’s Team Software Process (TSP). He obtained qualifications and certifications in a number of software process and quality practices including the SEI’s Personal Software Process (PSP) instructor, TSP launch coach, CMMI Assessor, ISO-9000, Rational Unified Process, and Certified Scrum Master (CSM).

Prior to Rally, Larry served as Chief Engineer and CEO of Comprehensive Computer Solutions, a systems integrator for factory floor automation, and was founder of QualTrax, software for measurement and management for ISO-9000 and other standards compliance.

Larry is currently finishing work on his Ph. D. in Software Engineering at Carnegie Mellon University. His research focuses on Agile measurement, analysis, and visualization for software and systems engineering.

About Rally

Rally inspires companies to deliver the right software faster with Agile and Lean practices. Our SaaS platform connects strategy and execution to steer businesses through chaotic, fiercely competitive markets. Rally drives over a hundred thousand users in countries around the world.