AGILE TESTING METRICS
QUALITY BEFORE VELOCITY
Some people never weigh themselves. They may say, “I just look at my clothes. If they don’t fit, then I know I should lose weight.” On the other hand, some people follow a regimen where they monitor their health regularly through diet and exercise. We all know that especially as you get older, even a few pounds can be difficult to lose. It’s much easier to catch a few pounds early in the game than to let it build up. “Shift left” applies to more than just software development!

For agile development, many proponents for ‘working software’ say there is no need for metrics. But if they don’t track metrics other than velocity, how do they know that the quality of their ‘working software’ is truly better? They may even say “tracking defects early is a waste of effort”. If we can just have a conversation and fix it, there’s no need for the overhead.” However, there is limited documented research regarding differences or any proven reduction in defect potentials, defect removal efficiency, delivered defects or customer satisfaction from pre and post agile implementation, but maybe it’s just hard to find. We often wondered what data and metrics agile proponents are using to support their claims that quality has improved using an agile methodology?

We’ve been working with our clients over the past 5 years as many of them have been migrating to an agile methodology and have always stressed with them that it is important to develop a baseline and some key indicators to see if things are improving.

This white paper presents some of the testing metrics that we have used with clients during this timeframe as we have partnered with them in their evolution toward agile and in reaping the benefits from agile. In this paper:

» We discuss the need for testing metrics even when using an agile methodology despite agile’s call for simplicity in “maximizing the amount of work not done”.

» Then we discuss some general principles to lay out a foundation of ‘what’ and ‘why’ for collecting metrics in an agile environment. In particular we discuss the key agile objective of quality.

» Finally, we list out some metrics from a testing perspective that we have used on various projects to help our clients not only keep track of their testing effort, but as a means to analyze and determine where they can improve. For brevity purposes, we list out quality related metrics and leave speed (velocity) related metrics for another white paper. We start with quality metrics because as indicated in the paper’s title, we’ve experienced that going for speed alone does not result in quality, but going for quality usually results in greater speed.
THE NEED FOR TESTING METRICS **EVEN WITH AGILE**

According to Capers Jones: “defects must be tracked throughout the life cycle in order to have high quality software”. Defect metrics can provide value for any development process including agile:

» Defects are data and can be analyzed to provide valuable information and insight into how we can improve.

» Tracking defects facilitates communication. In agile we are all co-located right? But we all know that co-located teams, while optimal are often not reality. And what about someone who quits? What if he was the one who knew how to reproduce the defect and was working on it? What if we want to examine the types of defects that we create and see how to improve over time?

» Although many agile teams say they’ll fix things as they go, and cite customer satisfaction as a measure of quality, it’s hard to have customer satisfaction with defects. So in a way, tracking defects is akin to “Shift Left”.

» Defects in production are simply the result of defects pre-production that don’t get removed. That’s important!

Actually, defects are the ultimate rework!
Doing something over, or fixing something is definitely not what we want. From every perspective, it costs us both speed and quality in agile.
CONCEPTS WE CAN AGREE ON - AGILE OR NOT

1. Focus on the most valuable information by tying to goals

The danger of trying to collect too much data is that it can be distracting and discouraging. So make sure what you collect is valuable. And make sure, as we’ll see later in the article, that each metric has interpretations with potential actions that are tied to related goals.

2. Fix defects early

Finding and fixing bugs early in the development life cycle is much more efficient and cheaper than finding them later so this reduces rework. Finding a bug later in the cycle more expensive just simply because of the overhead of tracking, fix, and regression testing.

3. Production defects – No one wants them

Once in production, defects have a cost to fix, plus other costs that are difficult to quantify such as those incurred by the end user, compounded by reputation damage, or even lost customers. Even so, for various reasons, both business and technical, sometimes we purposely let them go into production.

Relating Metrics Back to Our Agile Objectives

Let’s keep those concepts above in mind, while also recognizing that not all metrics are fit for all organizations. It wouldn’t make sense to measure blood sugar everyday for a patient not prone to diabetes who shows no symptoms, so it wouldn’t make sense to buy any of the special equipment to do so nor spend the time to do it. The point is that to determine the optimal metrics for your organization, you need to examine your current processes and determine where your problem areas are as well as your goals. If we are an organization that uses an agile development process, let’s start by looking at two key agile objectives:

» **Quality Software**: One of the main objectives of agile, to provide not only working software, but software free from defects. So any defect, especially those that would impact the customer or end user are important to stop in their tracks prior to production. If key problems enter production that impact our end users and how they view our software quality, we need to jump on it.

» **Velocity**: One of the main concepts is that we are supposed to produce good quality software, FASTER. So anything that measures elements that would slow us down or keep us from meeting our delivery schedule is important. Anything that makes us faster, keeps us from wasting time, or makes us more productive is important for agile. Additionally, the concept of working software early is a key part of agile. Each step is supposed to provide software that works so that feedback can be obtained quickly and nimbly switch directions or add/subtract requirements if needed. So any metrics that indicate software is not working, at each step/iteration is critical.

As we continue our discussion, we’ll relate each concept and metric back to **Quality in Agile**. Speed, or velocity, which is also a key objective, will be covered in another paper.
Getting Bang for the Buck on our Metrics Effort

The Pareto principle, otherwise known as the 80/20 rule will tell us that 80% of the defects are caused by 20% of the problems or located in 20% of the software. Most organizations have a gut feel for the most common types of defects and may even have a good idea when these defects are being introduced into the product. Given that, our data and information gathering efforts (metrics) should verify or complement what we suspect rather than start from scratch. In other words, we don’t want overkill. Less is more. We need to minimize the amount of data you collect thus focusing on the most valuable data.

To know what specific data from your defects you need to collect, or the types of defects you need to collect information on, we should answer these questions:

1. **What are most common types of defects causing the greatest impact on software quality?**
   a. Where are they located in the software?
   b. What are the reasons they occur, and what resolution?

2. **When are these defects most often injected into the product?**
   a. Finding defects early is good, but it is more important to find the most critical and costly defects at the moment they are injected into the software. This could be later in the cycle, not just in requirements.
   b. With poorly defined or ambiguous requirements that get injected early, you need to track requirements defects and take action early to eliminate those defects. If problems are primarily in detailed design or coding and requirements tend to be stable, then dig deeper later in the cycle.

3. **How will metrics and information be used and by who? What actions will be taken and by who?**
   a. You need to have indicators that say when you are in trouble and what you’ll do if trouble occurs.
   b. For an agile team focused on schedule, rework tends to drag things down. So what will constitute rework and what will you do when rework reaches a certain level?
   c. Technical debt also plagues agile teams in undocumented and unmeasured ways. How will you measure and track this, and what will you do when you reach what level?

4. **What defects should we worry about and fix, versus not fix? These days, many teams don’t fix all preproduction defects for a variety of reasons:**
   a. Because they are not critical
      i. They will not impact many customers
      ii. They will not impact important customers.
   b. It is too expensive/difficult to fix the defect and the cost of fixing is greater the penalty/cost of a customer finding the defect.
AGILE TESTING METRICS

Now that we’ve outlined the ‘what and why’, let’s dig into some actual metrics. Many of the metrics presented here are what you would use in a waterfall effort. However, it’s how you examine and interpret them that helps you reach your agile objectives. For this paper, we examine metrics that impact the agile objective of quality. Another paper will address speed or velocity related objectives. In general, without burdening the collection effort too much, organizations should consider collating the following information on defects: phase found (requirements, design, code, user documents and bad fixes), phase origin, type of defect, and reason for defect. All metrics should be tracked over time for trending purposes.

» Open Defects by priority, % for each priority, at any moment and over time. Of course we want higher priority defects to drop to 0 over time as the software gets close to release. In an agile environment, this is critical. We’d want to set a threshold for % high priority. If over 5 percent for example, then the software is not ‘working software’ and we are therefore, not ‘agile’.

» Defects by issue type, % for each type at any time, and over time.

» Perhaps you want to see if you are fixing the same type of defect over and over again. See what troubles our software the most. This is low hanging fruit for reducing rework if we can figure out what type of defect, and where it is injected.

» This can also tell us where to focus our testing in the future, or where to develop more test cases.

What do you think is happening in Figure 1 above?

Although this team may be doing well at reducing the open P1 defects, it seems that the P4 defects are piling up. And even if P4 is not important at the time, they certainly do ‘add up’ in the eyes of the end user. Thus, examining the definition of ‘working software’ may include a threshold for each kind of defect priority.
How you decide to categorize the issue types is up to you. We’ve laid out a few examples here. In this particular instance, we can see that program logic seems to be getting worse across the sprints. In this case, we probably need to examine the requirements to make sure they are easily understood by the developers.

- Defect aging, or time to fix a defect, average at any moment, and over time. This would be time between open and resolved. And time between resolved and closed. Or could be time between open and closed.
- Look for patterns and work to prevent the most expensive defects. Naturally, if a defect takes a long time to fix, from open to resolved, then that is expensive.
- So for those defects taking a long time to resolve, what is the issue type of those defects.
- Perhaps for those defects taking more than 72 hours to resolve, do a percent chart for those issue types so we can alert that they are preventing us from delivering ‘working software’.
- Total time spent fixing defects, from open to close is another metric if logged labor hours, rather than elapsed time is used. Then can be called rework. We can put this rework number as part of our burn-down work in our planning meeting, just to make sure it is counted and people see it.
Although defect aging is not necessarily an ‘agile’ quality metric, we think it’s critical to end user satisfaction and therefore instrumental in evaluating the quality of the software. If you are taking a long time to fix high priority defects, then your users won’t be happy, and especially with mobile apps, they’ll leave and never come back.

- Status (open, resolved, and closed), in %, at any moment and over time.
- In general we want to see lots of defects opened in the beginning of the project (defects found early is better).
- We often set thresholds when evaluating our performance for having a high percentage (90%) of the high priority defects found in the first 20% of the testing effort.
- Also, we can set a threshold for either an absolute number, or %, for open defects to make sure we are hitting our target of ‘working software’in agile.

Figure 3: Defect aging
Pre-production and post-production defects

By tracking pre-production defect issue types versus post-production issue types, we can determine where we need to focus more effort, and possibly raise our knowledge. For instance, if most post-production defects are found on certain platforms then perhaps we can look closer at deployment configurations or particular platforms. On the other hand, if certain functional areas have more defects in production, then we know somehow these escaped and that we need either more training in these areas or more focus. This type of analysis helps us to get even better and focus on those parts of the software that have the most risk.

Defect removal efficiency (DRE). We all agree that defects in production are bad, so let’s track it. Software quality is of course, about more than defects. Many say that user satisfaction is the most important. However, with too many defects in production, it’s hard for users to be satisfied! So we believe that focusing on processes and skills that reduce defects in the field, or increases defect removal efficiency is the best way forward.
With agile, you need to adapt the production defect period according to your sprint intervals. For instance, if your sprints are 2 weeks, but you release monthly, then you’d have a post-production defect count for the month following the release. In any case, you want to see this trend going up rather than down.

**Beware**

Note that comparing performance amongst individuals can have unforeseen behavioral results and therefore should be used with caution. Some simple examples:

- Testers who are measured by defect counts will seek easy bugs to raise their numbers. You can counteract that behavior by assigning greater weight to deeper defects just like you’d assign a higher difficulty rating to a triple somersault versus a simple dive.
- Programmers who are punished directly for defects will waste time arguing about whether it is a defect or not unless you have this well defined.

However, we can still use the information not to penalize, but to support training or strengthening our team. If we find for instance, that we have a lot of test design defects, or that one person seems to have a lot of these problems, we can put more effort in this area or send that individual to training so they have the tools/knowledge to get their job done.
CONCLUSION

Agile is a frame of mind, a way of thinking. Just because you are ‘agile’ doesn’t mean you don’t track and don’t measure. Otherwise how can you improve? You can try some different things and just guess that they might work, or you can measure. In this paper, we’ve outlined some metrics that we think you can use as a baseline. We are not advocating the use of all of them, and this is by no means the entire list of metrics.

We cannot claim that these metrics are ‘best practice’. Rather, that you should keep it simple. Start with a handful of metrics and tie each of them to your agile objectives. Start simple there too, starting with quality. If you feel things are not improving, change some metrics out. If you feel you’re getting better and faster, examine why, and keep doing it or do more of it. Make sure you are collecting metrics that tell you more about your weak spot(s) so you can improve in a targeted way yet not overburdening yourself in collecting information. This means that after a while, you may find that a metric has served its purpose and you need to move on to other metrics.