MEASURING THE IMPACT OF SOFTWARE CRAFT

BY BENJAMIN SCOTT

EBOOK - MAY 2021

ippon

BENJAMIN SCOTT

Ben is a craftsman with more than a decade of experience designing and delivering software solutions for a multitude of clients. He is passionate about Agile development and aligning business with tech to ensure the right solution is built at optimal scalability. His clients include startups to Fortune 100 companies. He is a passionate mentor, a leader, and a conference speaker.

Find his articles on the Ippon blog:
blog.ippon.tech

> WHAT IS SOFTWARE CRAFT? SIMPLY PUT, IT IS A MOVEMENT THAT FOCUSES ON CODE QUALITY AS OPPOSED TO COMPLETING BUSINESS FEATURES AS FAST AS POSSIBLE.

It's typically presented as an option or a trade off. 'You can have it done well or you can have it done fast.' I believe that this is a misconception.
Software Craft isn't a new movement, as its *manifesto* dates back to 2009 with its inception many years prior. For a good recap of where it came from and where it is now, read *What Happened to Software Craftsmanship*. The movement also hasn't aged well with its lack of inclusiveness by choosing a non-inclusive term. I'll be using "Software Craft" or "Crafter" when talking about Craftsmanship, as others have already started doing.

I recently did a tour speaking at several Agile and developer conferences on how to *Instill a Culture of Craft* in the organization. This talk was primarily aimed at developers, product owners, and engineering managers. The consensus feedback I got in regards to my presentation was positive. Attendees had a lingering question: How do I convince my own company to behave in this way? For software engineers that have spent time in large enterprises, you have probably noticed that there are a lot of metrics thrown around. Everyone above the team level (i.e., doers) seems to be obsessed with them. I am not really here to discuss the merit of metrics. It is just a fact that they exist and leadership tends to use metrics to make decisions. So what should we measure?

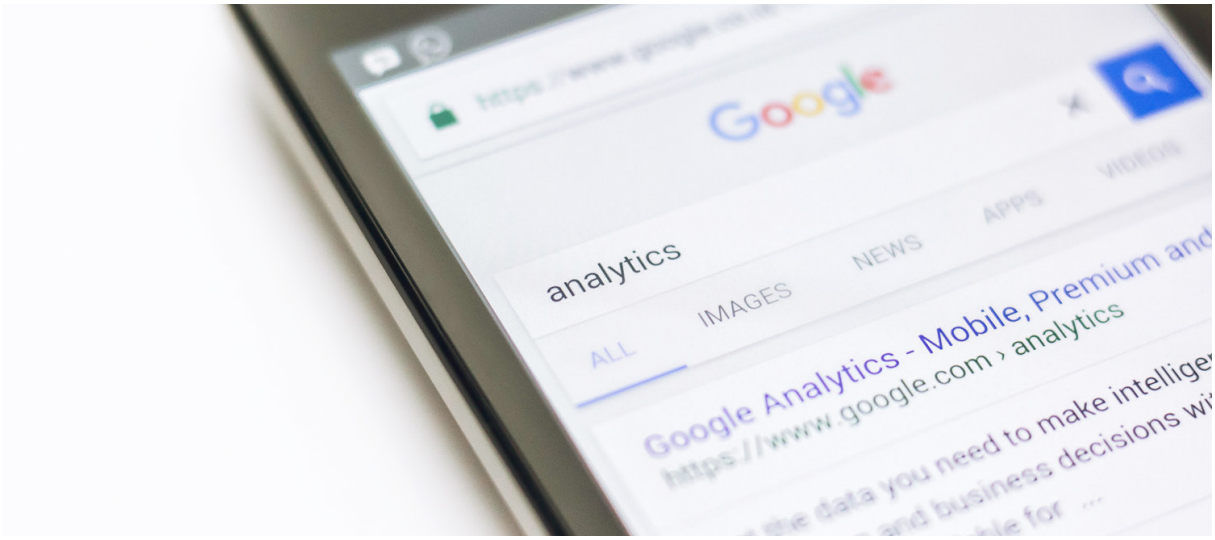
A typical way to measure the health of a codebase would be using tools such as *SonarQube* to measure code coverage, complexity, and the number of issues left to be resolved. While those metrics are certainly important, they are still a bit too technical to get real business buy-in.

Whenever deadlines loom, software craft goes out the window and the focus is right back to deadlines. It is crucial to use a measurement that caters to your target audience, and to be as factual as possible of its impact. Crafters do not care about code quality for the sake of designing beautiful systems; we care about code quality because it helps solve business problems, reduces cost, and enhances the developer experience.

When I join a company as a technical coach, I tend to focus on three metrics at the business level: Time-to-Market (TTM), Mean Time-to-Discovery (MTTD), and Mean Time-to-Resolution (MTTR). There are plenty of other metrics you may find more useful within your specific situation, but without any context, these three provide me with enough to decide where to focus. All of these metrics are linked to direct financial impact. Reducing TTM means reducing the cost of developing new features, whereas MTTD and MTTR combined show how long a bug takes to be fixed.

It has been shown that bugs have an *exponential cost to fix*, depending on where they are detected in the Software Development Life Cycle (SDLC), as well as how long the bug is out there in the wild. The practices of a Crafter aim to improve such metrics.

I will decompose each metric into their respective significance, what business impact it serves, and which Software Craft practice you can use to improve the metrics. You will notice that I propose techniques that go beyond the scope of merely writing software because quality software starts with quality requirements. I will not go into the specifics of each technique, as I am just trying to link a crafter practice to a business metric. By applying these techniques, you should see an improvement with the related metric.



01 — TIME TO MARKET (TTM)

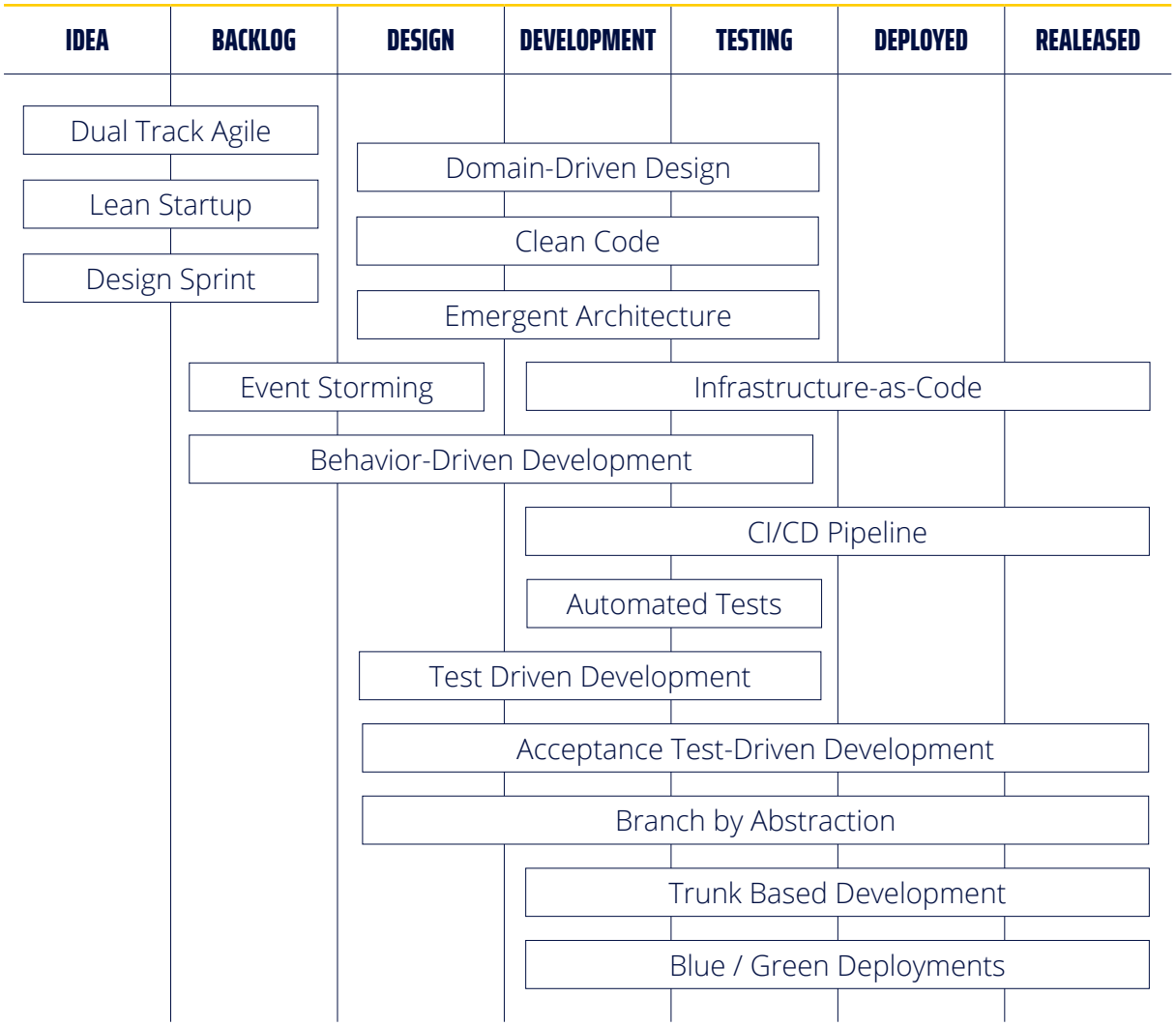
THIS IS SIMPLY THE MEASURE OF TIME BETWEEN THE START OF A PROJECT TO WHEN AN END USER INTERACTS WITH YOUR SOFTWARE.

THERE ARE OTHER METRICS THAT ARE SIMILAR, SUCH AS LEAD TIME AND CYCLE TIME.

For this article, I will simplify it by calling everything "TTM." Regardless of which delivery process you use, you can measure TTM and improve it by reducing the time it takes to go from one stage of the SDLC to the next. This metric is important because the quicker your TTM is, the quicker you can deliver features, bug fixes, and your overall responsiveness to change.

Plenty of companies had to *adapt* to the realities of COVID-19, some completely changing focus. This ability is reflected within your TTM. The more obvious business impact would be the economic cost associated with delivering a set of features.
The slower you are, the higher the cost. Not just in the financial cost of your employees delivering the work, but also in the opportunity lost by not generating new revenue from a feature being late.

The following tables show stages of a SDLC with some practices that a person might try to adopt to improve their TTM. They aren't in any particular order, but simply listed with which stages in which they primarily impact.



One of the best ways to improve TTM is to combine segments of the SDLC. For example, a Crafter would combine the act of coding a feature with writing the automated tests. This process is called Test-Driven Development (TDD). Another good way to improve TTM is to automate as frequently as possible.

02

MEAN TIME-TO-DISCOVERY (MTTD) AND MEAN TIME-TO-RESOLUTION (MTTR)

MTTD is a metric frequently used by DevOps teams to measure the time between when an issue starts within a system and when it is detected. MTTR is the time between an issue being detected and resolved. I always measure MTTD and MTTR in tandem. While you may focus on improving one as opposed to the other, it is always important to know both, as that will provide you the necessary context to suggest improvements.

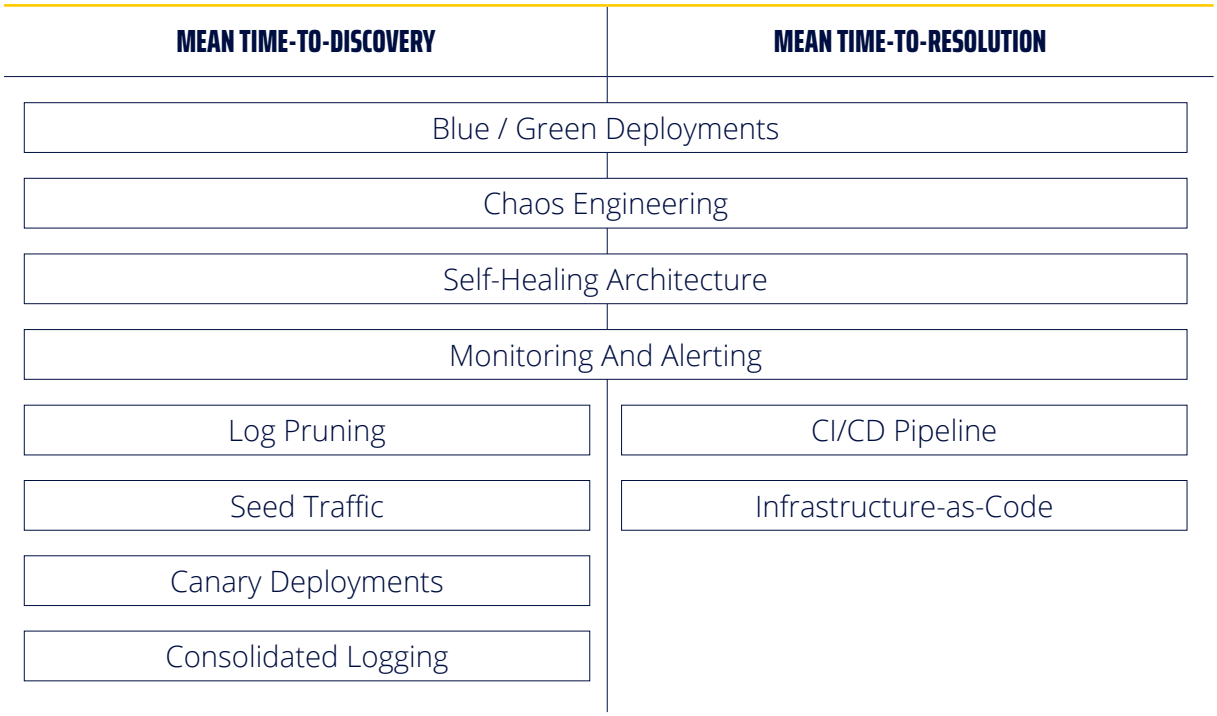
No one likes to use buggy software. We have all had to use it here and there. We feel differently about a company that identifies and resolves their issues in a timely manner compared to one where the issue lingers. MTTR has been a key metric used by a lot of support staff, as it directly impacts *customer satisfaction.* This metric is used when a customer reports an issue and its speed to being resolved. The real success is when you identify and resolve the issue prior to the customer even noticing it!

I like to do some triaging of the different issues and measure them independently :

— Technical issues (anything not business-related; can even be code issues, such as missing images on a web page or back-end business rules firing incorrectly)

— Business issues (code and infrastructure are behaving correctly, but the business outcome itself is wrong)

Technical issues can typically be solved by improving the technical side of the SDLC, but business issues require a different approach. This is more about communication between the product and development teams. Techniques like BDD are very good at improving those issues.
For those metrics, I would start with your QA / pre-prod, or whichever is your first non-prod, stable environment. We want to push the discovery towards that first non-prod environment. We want to avoid an end user discovering an issue as much as possible, and all the tools and processes that help with discovery can also be applied to your non-prod environment. This also allows for testing and validating



your incident response protocols.
MTTD/MTTR are important metrics because, together, they define how you can develop according to your risk tolerance. Your service-level agreements (SLAs) around defect resolution and your MTTR will have a direct impact on your release strategy and how rigid and/or flexible it can be.

If we look at it from a speed perspective, TTM is how fast you are going and MTTD/MTTR are your speed limits. Going too fast for your speed limit will result in issues and loss of revenue.

03 — HOW DO I CONVINCE MY BOSS?

Convincing your boss, particularly a business-minded boss versus a technical one, is always going to go back to measuring the impact to the business itself. First, attempt to measure the current situation, which may not even be possible if issue resolution isn't even tracked! Then, identify your circle of influence. Do not attempt to change the organization if you only have an impact on your team.

> THE FIRST LESSON TO LEARN ABOUT A CRAFTER'S JOURNEY AT TRANSFORMING THEIR TEAM AND ORGANIZATION IS THAT THEY MUST FIRST ADOPT IT THEMSELVES.

If you are trying to introduce better testing practices for your team, but still do not commit tests with your pull requests, you will have a difficult time convincing others. There will always be a gap between theory and practice, and using these practices within the context of your organization will bridge that gap and allow you to actually start a transformation. Then expand your area of influence.

Move to the team, promote your successes, and share your knowledge with the organization.

As you transform yourself and your team, and if you've been tracking TTM and MTTD/MTTR, then you should be able to start measuring the team's rate of change. Are they improving? Worsening? Be prepared that any significant change usually makes delivery worse before improving. Once you have demonstrated that these metrics have improved with Software Craft techniques, then you can escalate to your boss and present your story. It will have a much bigger impact coming from someone within the organization as a success story rather than any blog or talks that you can find.

ippon

CONTACT US!

us.ippon.tech
blog.ippon.tech
medium.com/ippon-technologies-usa

contact@ipponusa.com
+1 844.477.6687
@ipponusa