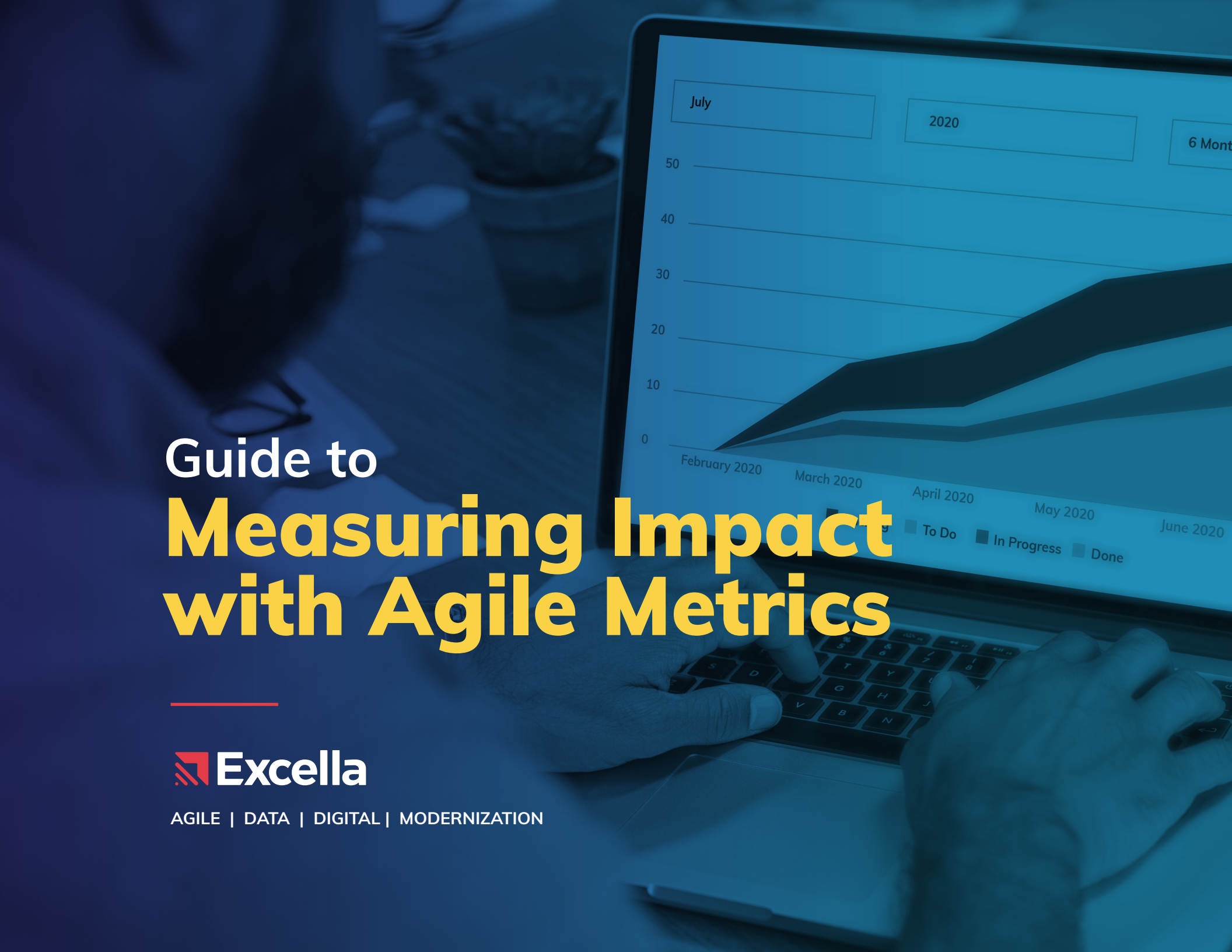


Guide to Measuring Impact with Agile Metrics

Excella

AGILE | DATA | DIGITAL | MODERNIZATION

In an Agile context, it's important to collect data to inform decision-making and enable continuous improvement. Not all metrics are created equal; however, some metrics can do more harm than good. One of the best recent sources of information about what consistently high-performing organizations measure themselves against is the 2019 Accelerate State of DevOps Report, compiled by Data Scientist Nicole Forsgren, who is also one of the authors of the book Accelerate. The "big four" metrics from that report are among those that we recommend, and which we'll describe further, along with many others: 1. Deployment frequency; 2. Lead time for changes; 3. Change failure rate; 4. Time to restore service.

At Excella, we have partnered with numerous federal agencies to help them on their Agile journey. We follow an adaptive approach, continuously seeking to improve, by listening to our business partners, running experiments, and making data-driven decisions based on the results from those experiments. Knowing what to measure, and what those measurements can and cannot tell us, is an Excella core competency. Our expertise is based on our own experiences as practitioners, along with careful application of metrics such as those that are included in the State of DevOps Report.

Let's focus our attention on four categories of Agile metrics that can provide actionable and meaningful insight, and help us focus on business outcomes:

Collaboration Metrics

Delivery Metrics

Quality Metrics

Value Metrics

Collaboration Metrics

Collaboration Metrics provide visibility into whether an Agile team is working together effectively, both internally (intra-team), and externally (cross-team), and also provide a pulse check on team morale.

Unlike many of the other metrics categories that we describe, Collaboration Metrics tend to be ***leading indicators***, offering early warning signs of potential areas of trouble.

At Excella, we place great importance on partnering with our clients to gather data that provides visibility into team health and organizational health, and this sets us apart from many of our competitors.

Examples of Collaboration Metrics:

Effectiveness

- **Number of Cross-Team Dependencies Identified/Closed.** Count the number of dependencies that exist, where each additional dependency significantly increases the risk that work will not be finished when expected; and count the number of cross-team dependencies closed, where each elimination of a dependency significantly decreases the risk that work will not be finished when expected. (**Focused Objective: Dependency Types and Impact**)

- **Mean Cycle Time of User Stories with Dependencies:** Collect data on how long it takes to complete each user story (its "cycle time"), and compare the cycle times for user stories with and without dependencies. (**Focused Objective: Calculating Throughput and Cycle Time History**)

Happiness/Satisfaction

- **Organizational Satisfaction Score:** Collect data from a broad cross-section of people in the organization and generate scores that highlight which things are going well, and which things need attention, at the organizational level. (**Crisp: Leadership and Team Health Checks**)

- **Team Satisfaction Score:** Collect data from individual teams and generate scores that highlight which things are going well, and which things need attention, at the team level. (**Atlassian: Health Monitor**)

Questions that Collaboration Metrics help answer:

Do team members proactively seek to minimize the number of dependencies that they have on other groups/teams?

Do team members feel empowered to discover and develop solutions to business problems?

Do team members feel that they can safety express their views, and are they confident that those views are heard and acted upon?

Collaboration Metrics to Try and Collaboration Metrics to Avoid

Try This	Avoid This
Evaluate cycle times, on an ongoing basis	Compare the cycle times of one team with the cycle times of another team
Collect data about the happiness of people in the organization	Single out individual teams because their morale appears to be lower (instead, look for root causes of broadly applicable organizational dissatisfiers)

Delivery Metrics

Delivery Metrics provide visibility into the relative frequency of deployment of software features to Production. As such, Delivery Metrics focus on the ability of an organization to deliver at a consistent, sustainable pace.

Delivery Metrics tend to be ***lagging indicators***, giving us insight into the maturity of our tools and processes that govern the management of code artifacts and monitoring of physical or virtual infrastructure.

Examples of Delivery Metrics:

Delivery Cadence

- **Deployment Frequency:** Collect data on the relative frequency with which new functionality is made available to users. **(**CircleCI: How Often Does Your Team Deploy?**)**

- **Lead Time:** Collect data on how much time elapses from the moment a customer first requests a feature to the moment that the feature is available to that customer for use. (GoCD:What Lead Time Is)

System Availability

- **Change Failure Rate:** Collect data on the relative frequency with which a deployment results in the immediate need to correct a problem caused by that deployment. (CircleCI: What does change fail rate tell us?)

- **Mean Time to Detect (MTTD)/Mean Time to Repair (MTTR):** Collect data on how long it takes to detect that a problem exists, and how long it takes to correct a problem after it has been detected. (AlertOps: MTTD vs MTTF vs MTBF vs MTTR)

Questions that Delivery Metrics help answer:

How many running, tested units of work can we reliably deliver during a Sprint?

How stable and reliable is our deployment pipeline?

How stable and reliable are code and other artifacts that we deploy to Production?

Delivery Metrics to Try and Delivery Metrics to Avoid

Try This	Avoid This
Evaluate lead times, on an ongoing basis	Compare the lead times of one team with the lead times of another team
Leverage system availability data to identify areas for improvement	Use a deployment or system failure as a reason to single out an individual or team for blame

Quality Metrics

Quality Metrics provide visibility into the extent to which the products that we deliver work as they are intended to work.

Some quality metrics, such as unit test coverage, are ***leading indicators***. For instance, if we take it as a given that unit tests are testing something meaningful, execution of unit tests can prevent problems from surfacing later. Other quality metrics, such as Defect Escape Rate, are ***lagging indicators***, because they tell us after the fact that we have a gap in our testing approach.

Examples of Quality Metrics:

Defects

- **Defect Density: Number of Defects by Module:** Count the number of defects that exist in a particular software module (often counted based on number of defects per 1,000 Lines of Code (KLOC). (SeaLights: Defect Density: Context is King)

- **Defect Escape Rate:** Count the number of defects that are opened in a particular component, where the defects are opened after the Sprint has ended during which the component was created or modified. (LeadingAgile: Escaped Defects)

Technical Debt

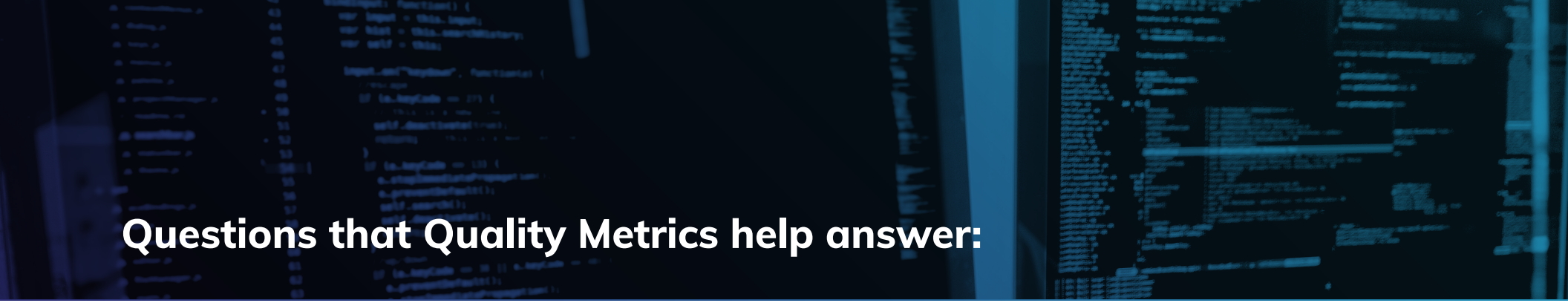
- **Amount of Time Set Aside for Code Refactoring (per User Story):** Capture data on the amount of time during which code refactoring is done, during initial development and/or during peer code review. (Mountain Goat Software: The Economic Benefit of Refactoring | Tushar Sharma: How to Track Code Smells Effectively)

- **Number of User Stories or Defects Created to Address Technical Debt:** Count how many user stories or defects are opened to address shortcuts that had to be taken due to time constraints; related metrics that can help uncover technical debt via code analysis tools include cyclomatic complexity, code coverage, SQALE rating, and rule violations. (Excella: The Technical Debt Management Plan | Christiaan Verwijs: How to Deal with Technical Debt in Scrum)

Test Coverage

- **Test Automation Coverage Level:** Collect data on the percentage of the code base that is covered by automated tests, by dividing the total test coverage by the test automation coverage. (LogiGear: 5 Useful KPIs for Test Automation)

- **Unit Test Coverage Level:** Collect data on the percentage of the code base that is covered by unit tests. (SeaLights: Code Coverage vs Test Coverage)



Questions that Quality Metrics help answer:

To what extent does the software work as intended?

To what extent do we give ourselves sufficient time to build things right the first time?

To what extent are there automated checks in place to surface anomalies before they become significant problems?

Quality Metrics to Try and Quality Metrics to Avoid

Try This	Avoid This
Seek to detect and correct anomalies in real time, during a Sprint	Open defects for anomalies that are detected during a Sprint (fix them on the spot!)
Use multiple testing approaches to check for correct software/ system behaviors	Create incentives for opening as many defects as possible
Make time available for code refactoring, which greatly reduces the likelihood that technical debt will be created	Place blame on teams when it becomes difficult to improve or maintain existing code, due to the existence of technical debt
Consider the extent to which different parts of the code base have sufficient test coverage	Assume that a high test coverage number is a guarantee of success (some tests may have been commented out or lack sufficient depth)

Value Metrics

Value Metrics provide visibility into the extent to which the products that we deliver enable customers to achieve their goals, and the extent to which our investments provide the returns that we expect.

Value metrics, especially those associated with Customer Satisfaction, are ***lagging indicators***, because some time needs to elapse to give customers sufficient time to interact with a new product, or to experience changes to an existing product.

Examples of Value Metrics:

Customer Satisfaction

- **Customer Effort Score (CES):** Capture data about a customer's experience, based on how much time was required by that customer to achieve a goal, such as getting a question answered, an issue resolved, or a product purchased/returned. (Retently: What is CES and How to Measure It)

- **Customer Satisfaction Score (CSAT):** Capture customer survey data, divide the positive responses by the negative responses, and multiply by 100. (Emolytics: CSAT: The Happy Customer KPI)

- **Net Promoter Score (NPS):** Capture customer survey data for a given group, total the number of people in the group and divide it by the number of survey responses, and subtract the percentage of respondents who are Detractors from the percentage of respondents who are Promoters. (Hotjar: What is NPS?)

Value Delivery

- **Return on Investment (ROI):** Divide the amount of money produced by a new product or improvement to an existing product by how much money was spent to build or improve that product. (Xpedia: Calculating ROI for Software Development)

- **Units of Value Delivered per Release:** Count the number of finished work items in a Release, and assign a value score based on that number. (Folding Burritos: Measuring Customer Value in a Software Product)

Questions that Value Metrics help answer:

To what extent do we deliver solutions that customers consider usable?

To what extent are customers satisfied with the solutions that we deliver?

To what extent are internal stakeholders satisfied with the solutions that we deliver?

Value Metrics to Try and Value Metrics to Avoid

Try This	Avoid This
Collect customer satisfaction data on a consistent cadence	Measure "success" only on the basis of traditional project management measures, such as "on time and on budget"
Collect data on how much work (and by extension, how much value) each team finishes during a particular time frame	Compare how much work one team completed with how much work another team completed

Conclusion

By taking a balanced perspective, being sure to include Collaboration Metrics (which often get little if any attention), and including Delivery, Quality, and Value Metrics, we have a strong foundation on which to build. As the authors point out on page 9 of the **2019 Accelerate State of DevOps Report**:

> Remember to accelerate your transformation by starting with a solid foundation and then focusing on the capabilities that are constraints: What capabilities cause the biggest delays? What are the biggest headaches? Where are the biggest problems? Pick three to five and dedicate resources to solving these first. Don't worry if you still have problems; by focusing on the biggest problems now, you remove bottlenecks, discover synergies, and avoid unnecessary work.