Productivity metrics and their integration into DevOps

Master of Science in Technology Thesis
University of Turku
Department of Computing
Software engineering
2024
Rasmus Riihimäki

UNIVERSITY OF TURKU
Department of Computing

Rasmus Riihimäki: Productivity metrics and their integration into DevOps

Master of Science in Technology Thesis, 55 p., 8 app. p.
Software engineering
June 2024

This thesis addresses the challenge of measuring productivity in a DevOps environment, focusing on identifying effective metrics, understanding their impact on software quality, and assessing their influence on developer well-being. Productivity and its various definitions are analyzed from historical, modern, and practical perspectives. The modern and practical findings lead the thesis into the realm of DORA metrics and the annual DevOps report. The thesis was started with the goal of improving the current productivity measuring inside the case company.

After conducting a literature review of the current research into productivity metrics, a case study where the current state of productivity metrics is assessed and improved within the DevOps teams at the case company is conducted. The methodology of this case study involves planning, executing, and validating the implementation of a productivity metrics dashboard. This approach sets an example for other organizations aiming to measure and enhance productivity using similar metrics.

The findings from the case study at the case company indicate that adopting a modern productivity metrics approach enhances software quality. This is achieved by promoting a fast deployment process and agile DevOps practices. The data collected and analyzed show a positive correlation between the use of DORA metrics and improvements in both software quality and developer well-being.

The case study suggests that implementing DORA metrics not only improves software quality but also supports developer well-being and self-reported productivity. By following the example set in this thesis, other organizations can plan, execute, and validate their productivity measurement processes, leading to enhanced software quality and a healthier, more productive development team.

Keywords: devops, productivity, dora, developer experience, metrics, software, self-reported productivity, software quality

Contents

List of Figures

List of Tables

1 Introduction

It is hard to improve things that are not clearly quantified and measured. This is the case in many things, especially DevOps. Which metrics have the most correlation with wanted outcomes such as software output and software quality are to be investigated and identified. Once these bottlenecks of production are identified by the metrics found to be the most valuable, improvements can be targeted more efficiently at the root causes of these metrics. There might be some insights that are useful to record and measure that aren't even known about or valued as highly as some others. Continuous improvement is a core element of DevOps, and it is much easier to do so with the right metrics in place.

1.1 Goal

The goal of the thesis is to gather a comprehensive understanding of the different existing software development metrics. As well as to figure out which metrics have more academic basis or if following these metrics is even useful in the first place. It is important to gain an understanding of the limitations certain metrics have in providing insights and considering what context is required to truly analyze the metrics. It is not enough to just follow these metrics; they also need to be analyzed and the results should somehow be applied in the DevOps environment. So, the feasibility of building incentive structures around these metrics is examined, considering whether it would lead to more developer burnout or incentivize the writing of poor-quality code.

From the company's perspective, the goal is to find suitable metrics to follow that will ultimately benefit the business or even the developers themselves. Also getting rid of unnecessary metrics, data or processes surrounding them is always beneficial. Another goal is to provide the company with the tools to properly analyze the metrics once they are being recorded to get the most understanding out of them as possible.

1.2 Research questions

The research questions guiding this thesis are:

1. Which metrics have the strongest academic backing in terms of correlation with productivity?
2. Will an increase in productivity through metric based methods negatively affect the DevOps process in terms of quality?
3. Can strictly improving these metrics have negative effects on developers?
4. What metrics should be integrated in the case company?

These questions provide a framework for investigating the relationship between productivity metrics and their impact on both the DevOps process and developer well-being. These questions are answered in the conclusions 7.

1.3 Methodology

Reading academic papers about these topics including developer experience, productivity metrics, repository mining and metrics implemented in DevOps is the first step. The way to go about gathering these papers and getting an overview of the topic is by doing searches on google scholar with the above-mentioned keywords and reading whichever papers are connected to the subject.

The next phase of the process is to apply the understanding gained from the research by using the case company's DevOps as a case study. First, the data required for the metrics deemed the most fitting in this case is gathered. Once this is done, the metrics are analyzed to see if some key insights can be found using the metrics employed. The findings or the lack thereof throughout the process is being documented.

Once the metrics are ready and presentable, It is demonstrated alongside with insights to the relevant parties. After this feedback is gathered from the participants using a survey. The survey results are then be analysed and used to gleam insights about the metrics and relevant opinions on related topics from the participants themselves.

1.4 Structure

The structure of this thesis starts with an abstract of the most important conclusions and a short summary of what the thesis is all about. In the introduction 1, the goals, research questions, methodology and structure are clarified.

The first background chapter delves into productivity as a concept, investigating it's many definitions, how it's measured and what it means in different contexts. In the productivity chapter 2, the general concept of productivity is outlined in the productivity in software 2.1 and DevOps chapters 2.2. The context of both software development as a whole and then DevOps processes are introduced to converge on the most precise and useful definition for this thesis.

The productivity metrics chapter 3 goes over how productivity is, has and can be measured. In the repository mining chapter 3.1 the more historical ways of measuring productivity are delved into. Then in the DORA 3.2 and the SPACE framework 3.3 chapters the more modern applications that are the main focus in this thesis are analysed. The evolution of productivity metrics is summarized in the Evolution of productivity metrics 3.4. Then finally the application of the metrics into a DevOps environment is detailed in Applying productivity metrics in DevOps 3.5.

Software metrics in the case company 4 examines the state of metrics and productivity in the case company before the changes done in the case study. This is done to help identify what is the right option for the next steps of the case study. These steps are then put to action in the case study chapter 5 with a detailed description of the planning phase in the expectations and plan 5.1 and defining the needed metrics chapters 4.1. That plan is then executed and described in building the metrics 5.3 and hosting the dashboard 5.4. Finally the finished project is validated in the validation chapter 5.5 and examined in the results 5.6

The final chapter survey 6 goes over the planning, designing and execution of the survey capturing participants opinions on the project and relevant questions in regards to this thesis's research questions. The results are captured in survey results 6.1 and analysed in survey insights 6.2. To close out the thesis the conclusions 7 go over the key findings and answers the research questions set in the introduction.

2 Productivity

This thesis delves deep into the complexity of productivity and it's many definitions. To put productivity in the simplest terms it is defined as the ratio of outputs divided by the inputs. This is an easier task in assembly line factories where they can count strictly the number of hours worked and capital invested in materials as input and the number of objects produced as the output. Having such a simple calculation for productivity makes productivity very easy to measure and manipulate. Productivity measures how efficiently production inputs, such as labor and capital, are being used in an economy to produce a given level of output. One of the most widely used measures of productivity is Gross Domestic Product (GDP) per hours worked, capital invested or materials used. (Pilat and Schreyer 2001)

The reason why productivity is to be measured is because companies want to be as productive as possible, and you can't change something if you don't know how to measure it. The more clearly, defined the inputs and outputs are the easier it becomes to identify which corrective actions work and which don't. By comparing to other companies, internally between teams or just tracking change over time it will be possible to tell when corrective action is needed. If for example, it is noticed that some team's productivity is declining there is probably some issue that needs to be addressed in that team specifically and not necessarily company wide.

Productivity growth is the basis for improvements in real incomes and welfare. Slow productivity growth limits the rate at which real incomes can improve, and also increases the likelihood of conflicting demands concerning the distribution of income (Englander and Gurney 1994).

2.1 Productivity in software

When it comes to developing software the general understanding about inputs and outputs stays the same, but it becomes significantly harder to identify those inputs and outputs. Software development is often a costly endeavor and therefore ways to increase to increase productivity are always being searched for. Junior et al. state that different organizations and persons have distinguished notion about the concept of what is produced and its associated value. Therefore, the appropriate model for measuring productivity in software organizations needs to be tailored uniquely for each type of organization. (Junior and Meira 2009)

The ideal way to calculate productivity in software projects should measure how to effectively and efficiently turn ideas into software products. Physical code base size metrics like lines of code, number of functions or classes are very limited measurements of output and can sometimes even be entirely misleading. The concept of value is often talked about as a measurement of software products, but it has its own limitations since it is very much up to interpretation and can change depending on the stakeholder. A project manager can evaluate a project by amount of products released compared to the developer effort required. Where as, a CFO might look at it more broadly and compare the amount of funds generated based on the resources consumed. Since this size of change is nearly impossible to measure objectively, you need to be careful about applying the productivity principles of manufacturing to software development. (Forsgren, Rothenberger, Humble, Thatcher, and D. Smith 2020)

2.2 Productivity in DevOps

DevOps is much more than just a new type of production line for software. It is a new way of thinking and a shift in software development culture. DevOps is also constantly improving itself which makes earlier DevOps quite different from current DevOps. With this shift in culture Software production is no longer about individuals and their produced code, but rather a team and systems. This introduces difficulty into the simple calculations of inputs and outputs, since things like quality, scalability, resilience and stability must now be more carefully controlled for. Resilience in a DevOps context meaning having systems in place that can catch and correct mistakes before they cause any larger harm and stability meaning the downtime of your applications being as minimal as possible for the end-users. (Ravichandran, Taylor, and Waterhouse 2016)

It is not enough to measure created value; metrics that give us information about things such as stability and quality must also be looked at. If every second deployment destroys the entire software and requires more time to fix, the productivity suffers heavily. Therefore, it is important to have good practices for quality control, testing and reviewing the software you plan to produce.

A core practice of DevOps is continuous development (CD). This includes making sure the required changes are developed at a rapid pace. Items should not take a long time from being introduced to being in production. With a working CD loop your software team is able to deploy to production at any time, enables constant checking of quality and makes it possible to prioritize blocking deployment (*2022 State of DevOps Report* 2023).

Forsgren et al. state in their paper that "the performance clusters demonstrate that organizations achieve throughput and stability concurrently. Thus, it is possible to achieve high throughput and high stability without any trade-offs" (Forsgren, Rothenberger, Humble, Thatcher, and D. Smith 2020). This means that these different values of DevOps are not necessarily at odds with each other, but usually a highly functioning system has a combination of all these performance indicators.

An important note is to recognize that a productive system focuses outcomes not output. Productivity isn't really useful for anyone if you are productive at creating bad products and services. The productivity systems should not incentivize putting lots of hours into work that doesn't ultimately bring value to anyone. (Forsgren, Humble, and Kim 2018)

The evolution of the word productivity encapsulates things like developer satisfaction, performance, activity, collaboration and flow in addition to the typical idea of engineering outcomes. This is the basis for the SPACE framework which is discussed deeper in the SPACE framework chapter 3.3. In DevOps productivity has to be thought of in a more multidimensional way. You can not follow just activity metrics defined in the SPACE-framework as theoretical inputs and outputs of a standard production line, because high activity could be indicative of working longer hours or producing fast poor quality code. High activity could also be indicative of your systems working and developers having all the tools they need to do their job effectively or having a highly collaborative and cohesive team who help each other get their work through the DevOps pipeline as fast as possible. Productivity shouldn't be only about the individual in DevOps since work is done as teams. If your most experienced developer is being very productive but not helping the team by communicating, participating in code review or helping with up-keeping the engineering systems your team will suffer in the long run. These trade-offs have to be counted for and sacrificing the team in terms of personal productivity should not be incentivized. A common misconception is that productivity is only about engineering systems and developer tools. While they are a large part of what can be measured there are a lot of practices like moral building, mentoring and knowledge transferring which can not be measured but play an integral part in productivity.

(Forsgren, Storey, Maddila, Zimmermann, Houck, and Butler 2021)

2.3 Self reported productivity

One significant dimension of productivity in DevOps involves self-reported productivity (Beller, Orgovan, Buja, and Zimmermann 2020). Self-reported productivity measures how developers perceive their own productivity and the factors that influence these perceptions. This subjective measure is almost as crucial as the traditional objective metrics, which might miss some of the human element of productivity and developer experience.

Beller et al. research indicates that factors contributing positively to self-reported productivity include having ample time to code, the ability to work without interruptions and a healthy lifestyle outside of work, that includes for example proper diet and sleep. The largest negative impact was caused by traveling. The next largest effects being having no time to code or getting interrupted a lot while working. Surprisingly weekday did not seem to effect self-reported productivity much on average even Fridays were as productive as other weekdays. One key finding of the paper was that the mere act of reflecting on their productivity engineers felt more productive. (Beller, Orgovan, Buja, and Zimmermann 2020)

It's also important to consider that there might be differences between team and individual self-reported productivity. Optimizing for individual productivity over team productivity will hurt the overall team performance. This can look like avoiding team activities like code reviews or knowledge sharing. This is especially important in DevOps which is built around collaboration. This is supported by the findings of Storey et al. in which they found a very strong correlation between job satisfaction and productivity. (Storey, Zimmermann, Bird, Czerwonka, B. Murphy, and Kalliamvakou 2021)

The interplay of job satisfaction and productivity feed into each other. Higher job satisfaction generally leads to increased productivity, which in turn can enhance job satisfaction. This is why it is important to keep in mind developer experience and well being to foster a productive DevOps team. While technical aspects like code infrastructure and available tools are important, the factors most correlative with job satisfaction tend to be non-technical, like relationships with colleagues and managers, opportunities for growth and workplace culture. (Storey, Zimmermann, Bird, Czerwonka, B. Murphy, and Kalliamvakou 2021)

In conclusion productivity in a more specific context like DevOps can mean much more than the traditional productivity definition of inputs versus outputs. Combining both objective metrics with the more subjective ones of asking the developers themselves, can create a more holistic view of productivity within an organization. This in turn makes it easier to understand where issues are truly stemming from and where to focus improvement efforts in hopes to enhance productivity.

3 Productivity Metrics

When it comes to measuring productivity, it is integral that what is being measured is productivity or at least related to it. In this chapter I go through all the different types of possible routes to take when measuring productivity. The goal is to validate the metrics scientifically and possibly do comparisons on what is the most meaningful thing to measure. There might also be a larger context around why something should be measured. For example, it might be important to measure something that is not directly linked to productivity to avoid adverse effects of trying to maximize productivity.

To get a good grasp of the entire picture it is important to do qualitative surveys in the company alongside quantitative data collection. This is done because each software company is different in for example, their work methods, team structures, goals, strengths and weaknesses. This qualitative data is then used to pinpoint the reasons for the quantitative data. However, you must be careful not to rely on either too much, since often it is easy to retrospectively come up with reasons why the data shows what it does. Trends and data over time is more valuable than a single snippet of data.

3.1 Repository mining

One way to get a picture of productivity is to dig deeper into the developers' code contributions. This practice is often called repository mining. There are many insights you can gain from looking at code changes directly. Code commits are the thing closest related to physical products produced by a factory in a software development environment, whereas issue tracking systems, which will looked at in the next chapter **??**, tend to be more intangible.

The simplest thing to look at straight away would be the amount of code produced, but doing so will instantly cause you to run into problems, since not all code is created equal. Measures like lines of code (LOC) don't provide us with very useful information, since it doesn't take into consideration the complexity, quality or nature of the code. Measuring LOC could even encourage developers to intentionally write less concise and lower quality code. An example of a metric that tries to avoid this issue of encouraging bad code is Halstead Effort by Time (HalsteadEff/Time). In HalsteadEff/Time the code's complexity is calculated using the number of operators and operands in the code, therefore the measure considers more than just the size of the code files like LOC. The HalsteadEff/time metric was however criticized as not measuring actual complexity of code by team leaders in Oliveira et al. study. (Oliveira, Fernandes, Steinmacher, Cristo, Conte, and Garcia 2020)

Other metrics to evaluate could be metrics like average complexity of methods, introduced bugs, bugs fixed. Each of these metrics have some issues, especially when analyzed in isolation. The complexity of methods metric could be used to find developers who create unnecessarily complex code. Complex code is an issue because it makes future adjustments more difficult to do. It is hard to differentiate when complexity is necessary and when it isn't and some developer might be tasked with more complex tasks, so the average complexity doesn't really tell the whole story. Bug related metrics also seem to be very heavily affected by outside forces like who is tasked with bug tickets, who is tasked with the hardest tasks and delivery time pressure. (Lima, Treude, Filho, and Kulesza 2015)

Commit based metrics can be very imprecise. It depends heavily on the commit policies of the developers and how strictly they are followed. In some cases, a developer's commit might contain 1000 lines of code and in another only 2. Also, some developers are stricter about only committing code they plan on releasing and others might just use it to save their progress.

Although repository mining has it's uses, the problem in general is that it requires a lot of labor-intensive manual analysis to go along with the data itself. And often the findings aren't very concrete. Oliveira et al. did find that their results suggest that productivity metrics, especially code-based metrics, can complement the subjective perception of team leaders (Oliveira, Fernandes, Steinmacher, Cristo, Conte, and Garcia 2020). By this they mean that these code-based metrics can be used to suss out undervalued developers in your team.

Velocity is another measure for productivity. It has been used in the past, especially during the first introductions of agile into the software development ecosystem. Velocity is measured using a story point based system, where the teams estimate the effort of a task and assign it a certain amount of points. The intended use for story points is for time estimation and planning, but some managers decided to use it as a productivity metric. There are many flaws with using Velocity as a productivity measure the first of which is the relativity of the estimations, since different developers will estimate tasks very differently based on their existing skills and knowledge. Another flaw is that Velocity is very easily gamified by overestimating tasks and then focusing on completing as many points as possible disregarding other things such as collaboration between teams. (Forsgren, Humble, and Kim 2018)

3.2 DORA

As the software sector has been incorporating Agile and DevOps methods into their way of working, the measurement and even definitions of productivity need to be updated to a more modern standard. A lot of the studies around measuring lines of code as proxy for productivity date to the earlier 2000s and in software development the landscape is almost unrecognizable when comparing to today. In DevOps the focus is on bringing value to the stakeholders, so it should follow that that is what should be measured.

DORA (DevOps Research and Assessment team) metrics are the key metrics that the Google research group found to be the best for measuring software development and delivery performance. These four key metrics include Deployment Frequency, Lead Time for Changes, Mean Time to Recovery and Change Failure Rate. The goal of the DORA metrics is to break down the abstract processes that exist in software delivery and make them more tangible.

These metrics are split into two groups stability and throughput metrics. Mean time to Recovery and Change failure rate falling into the former category and Deployment Frequency and Lead Time for Changes into the latter. The DORA team also categorized teams into different levels of performers from low to high level performers. These values will be referenced using table 3.1 created based on the results gathered in *2022 State of DevOps Report* 2023. As you can see in table 3.1 the differences between low and high performers in most categories are very drastic. In the 2021 state of DevOps report it was calculated that the elite performers are 6570 times faster in lead time and time to recover than low performers. The amount of companies in the better performing categories has gone up and ones in the lower categories have gone down every year that these surveys have been conducted excluding 2022. This is because the DORA researchers decided to reshape the categories they were using in previous years due to the data not clustering into four categories and because there wasn't such a clear difference between high and elite level companies that there had been in previous years. They hypothesize this might be because of the pandemic slowing down innovation. (*2022 State of DevOps Report* 2023)

These key metrics are often calculated manually since lots of the necessary data is

Table 3.1: Different level DORA performers benchmarks

SDF metric	low	medium	high
Deployment frequency	once per month - once every 6 months	once per week - once per month	On-demand, Multiple times a day
Lead time for changes	one month - six months	one week - one month	one day - one week
Time to restore service	one week - one month	one day - one week	Less than one day
Change failure rate	46%-60%	16%-30%	0%-15%

often scattered around different services. But Sallin et al. were able to use a method with which these calculations can also be done more automatised and still come away with useful results (Sallin, Kropp, Anslow, Quilty, and Meier 2021). There are also a few emerging software solutions to help with the problem of visualizing and gathering all the required data for these metrics. These will be gone over in more detail in section 5.

3.2.1 Deployment frequency

Deployment frequency measures how often changes in code are deployed to production. By decreasing the size of batches and increasing the velocity of deployments it ensures that new features and fixes are constantly being pushed out. (Sallin, Kropp, Anslow, Quilty, and Meier 2021) Deployment frequency is also a good proxy metric for properly sizing tasks, since you can't have fast deployments if batch sizes get too large. In the state of DevOps 2019 report by Google high level companies are bench marked at deploying even multiple times every hour, whereas lower level companies could be as slow as once every month to once every six months 3.1. A way to calculate Deployment frequency is to monitor your deployments using a pipeline tool like Jenkins or Azure DevOps and extract the amount of successful deployments.

While at first glance Deployment frequency seems like a metric to measure speed. It is more indicative of proper batch sizing. You should not attempt to increase Deployment frequency by trying to push out the same types of changes at a faster speed. Unless your developers are intentionally slacking off this kind of pressure will only lead to negative outcomes like lack of testing, poor code quality, burnout and decreased developer satisfaction. (Forsgren, Humble, and Kim 2018)

3.2.2 Lead time for change

Lead time for change measures how long it takes on average from the start of a task to get it to the end. More specifically the measurement starts from the initial commit to the code running in production. The shorter your lead time for change is the faster you are able to react to feedback and change course if necessary. A fast Lead time for change also enables fast reactions to fix outages and defects. (Sallin, Kropp, Anslow, Quilty, and Meier 2021) A high level of Lead time for changes is bench marked at less than one day. The low level bench mark is around a month to six months. (*The 2019 accelerate state of devops: Elite Performance, productivity, and scaling | google cloud blog* 2019)

Other related metrics are Lead time and Cycle time, which measure the process of new changes form even further back than the first commit. Cycle time refers to the time it takes from the first moment a team starts to work on a task to the final deployment. Lead time is an even longer metric which measures how long it takes from a customer request arriving to that change being deployed. These metrics can be useful if you want to take an even deeper look at how responsive your team is being. (Forsgren, Humble, and Kim 2018)

3.2.3 Mean time to recover

One of the four key metrics is Mean time to recover also referred to as Time to restore service. What it measures is how long does it on average take to fix a failure and return the system state back to normal. There are also many different opinions on how this should be measured and defined (Sallin, Kropp, Anslow, Quilty, and Meier 2021). The most useful and common definition go by is the time it takes between incident creation to closing the incident. The less time a system is in a failed state the more reliable the service can be called.

The reason mean time to recover is measured is that it can be quite indicative of many different issues like your alert systems, diagnostics or your repair process. While Mean time to recover doesn't directly diagnose the problem it can provide a starting point by indicating it exists. Once you know of the problem you can implement more specific metrics like Mean time to respond or Mean time to repair, to figure out how much time the failure spends in each part of the recovery process. (Forsgren, Humble, and Kim 2018)

3.2.4 Change failure rate

The Change failure rate metric measures how often deployed changes fail. There is some conversation around what is considered a failed change, since different software teams have different ways of dealing with failed deployments. To avoid confusion around this metric it is important to define what a failure is in the specific use-case. Some examples of different ways to count failures for a Change failure rate that Sallin et al. found in their review are listed below. (Sallin, Kropp, Anslow, Quilty, and Meier 2021) (Forsgren, Humble, and Kim 2018)

– Percentage of releases that were followed by a "fix release".

– Count of hot fixes in commit messages.

– Detect failures by using monitoring metrics and divided by deployments.

– Manually mark a deployment as successful or not.

– Count rollbacks divided by deployments.

Why measuring the change failure rate is important is because it directly correlates with the quality of code being developed. It is not possible to avoid all kinds of failure in software development, however elite to medium level companies all stay below a 15 percentage Change failure rate (*The 2019 accelerate state of devops: Elite Performance, productivity, and scaling | google cloud blog* 2019).

3.3 The SPACE framework

The larger idea of developer productivity is not totally encapsulated by the DORA metrics by themselves so in Forsgren, Storey, Maddila, Zimmermann, Houck, and Butler 2021 the authors create a larger framework around these metrics to get a better understanding of how to consider all the dimensions of productivity at once. The idea that productivity could be measured with the one metric that matters is just sadly not the case. The SPACE framework is meant to help you choose more carefully which metrics suit your context and what their limitations are especially when used alone or in the wrong context. In the SPACE framework productivity is split into five different categories

-Satisfaction and well-being: Satisfaction is how fulfilled developers feel with their work, team, tools or culture. Well-being is how healthy and happy developers are and what impact their work has on their health. These can be measured using survey data asking questions about employee satisfaction, developer efficacy and burnout. While high satisfaction on it's own doesn't mean you are being productive it has a predictive power about how possible upcoming burnout or general dissatisfaction which will lead to reduced productivity.(Maslach and Leiter 2008)

-Performance: Performance is the outcome of a system or process. With performance specifically outcomes instead of output should be measured, meaning that the developers code doing what is was supposed to do ought to be measured. If output is measured as created customer value it is highly dependant of what kind of work you do, for example you might not have high output if you are assigned less impactful tasks. Some metrics that can be used for performance are reliability, absence of bugs, customer satisfaction and feature usage.

-Activity: Activity is the count of actions or outputs completed while working. What makes measuring activity complex is because the sheer variety of different actions developers can take makes it hard to quantify activity comprehensively. Some measurable activities are commits, code reviews, deployments, count of incidents and incident mitigation. Not all activities are as easy to measure like participation in brainstorming and helping team members with their issues.

-Communication and collaboration: Communication and collaboration is quite self evident. It measures how well people collaborate and work together in teams. Teams that communicate and collaborate well with a high level of transparency and awareness have good outcomes related to working on the right problems, brainstorming new ideas and choosing the best solutions out of multiple options. Collaboration and communication is often quite difficult to measure objectively, since it contains lots of so called invisible work that isn't documented. However some proxy metrics for this are for example quality of documentation, quality of reviews and on-boarding time.

-Efficiency and flow: Efficiency and flow consists of the ability to complete work and make progress with minimal interruptions or delays. Interruptions have a clear negative effect on productivity by increasing the amount of errors made, causing stress and taking time to get reoriented into the work. The longer the interruption the more disruptive they are. (Brumby, Janssen, and Mark 2019) Getting into a flow is also reported as very important for self-perceived productivity (Murphy-Hill, Jaspan, Sadowski, Shepherd, Phillips, Winter, Knight, E. Smith, and Jorde 2021).

DORA contains some effective system level metrics for flow including deployment frequency and lead time for changes. Other flow metrics could be number of hand-offs in a process, length and amount of interruptions and perceived ability to stay in flow. While efficiency is extremely important it is also necessary to make sure your efficient at the right thing and don't start to decrease the quality of your system as a sacrifice for good flow. While working completely uninterrupted sounds like a good idea, it might stifle collaboration within the team. To avoid this it can be beneficial to block out uninterruptible work time.

The researchers suggest that to have a holistic productivity measurement you should have metrics in place that cover at least three of the productivity categories. This is because with three different dimensions covered you create some inherent tension between the metrics so you can not just focus on one aspect of productivity without also considering the rest. The different key four DORA metrics slot into slightly different categories covering system-level Performance, Activity and Efficiency and Flow, but they do not cover satisfaction or collaboration. It is also recommended to have at least one measure that is based in survey data and so that you are not relying on system data alone. Surveying developers can often help you explain lots of the gaps that system data inevitably leaves. This does not mean however that you should overwhelm and confuse people by implementing a way too large number of metrics at once, but rather you should always justify whatever metrics you are putting in place. (Forsgren, Storey, Maddila, Zimmermann, Houck, and Butler 2021)

3.4 Evolution of productivity metrics

Measuring productivity in software development is constantly evolving and expanding. While the earlier repository mining based metrics fall short in their usefulness, in certain ways they have paved the way for the more modern metrics like DORA.

DORA is becoming more and more a staple of DevOps and it has a growing consensus around it for good reason. The key four metrics will receive the most focus in this paper as they are the most established and studied. This does not mean that they are the only useful metrics, but rather a good baseline for starting to measure productivity in DevOps.(Forsgren, Humble, and Kim 2018) An example of other metrics that are often used along side the key four are flow, culture, continuous integration and customer happiness metrics. There are also other metrics that fill in important gaps in the key four metrics that they might not pick up like passed automated tests, bugs fixed and defect escape rate.

In the *2022 State of DevOps Report* 2023 they introduce a category of metrics along side software delivery performance labeled operational performance which measures modern operational practices. It is established that software delivery performance aka. the key four metrics when paired up with reliability are predictive of organizational performance. In this most recent report they introduce a category of metrics along side software delivery performance labeled operational performance which measures modern operational practices. The measure of operational performance is reliability which measures how well your services meet user expectations. Some examples of reliability are availability, performance, latency and scalability. This was measured by asking survey respondents to rate their ability to meet or exceed reliability targets. When operational performance was prioritized along side software delivery performance aka. the key four, it lead to less burnout. It is also mentioned that if reliability is poor then software delivery performance does not predict organizational success. This phenomenon is explained by the fact that users don't really benefit from getting faster code updates when the service is inherently unreliable. (*2022 State of DevOps Report* 2023)

3.5 Applying productivity metrics in DevOps

Flow metrics are often used in tandem with DORA to get a deeper understanding of the end-to-end flow of software value. Detecting bottlenecks becomes much easier when the whole flow of value is being tracked. Flow metrics consist of velocity, time, efficiency, load and distribution. They are often taken to use once your basic DORA metrics have been established and you want to dig deeper into the specific bottlenecks you might have in your DevOps pipeline.

An important aspect of the key four metrics are that they are quite resistant to gamification because of them being system-level outcomes. For a long while speed and stability were thought of as trade-offs to each other. However through further study Forsgren, Humble, and Kim 2018 concluded that this line of thinking was flawed and actually the companies performing the best in speed of releases and deployments, where actually also the best performing in stability on a larger scale. You can only get short term gains in throughput by adding pressure onto the developers to develop faster, but this will always bite back in the way of poor quality code or developer burnout. Cutting corners might increase throughput momentarily but it will be shown overtime in the stability metrics. This is a reason why these metrics need to be looked at regularly and too many conclusions can't be drawn from a limited time frame of data. Many people have had security concerns in regards to increasing deployment frequency, but as it turns out having a system which enables deploying new developments quickly also benefits from having integrated security practices. It is important to remember that the key four metrics are indicators of where you are at and not a goal in and of itself. The way you address your shortcomings in these metrics is to find the root cause of the issue and then give your developers tools to improve on said issues. Losing sight of why these metrics were put in place in the first place ought to be avoided, which is to improve productivity through which the quality of our product and satisfaction of customers is improved.

(DeBellis, D. Smith, and Castillo 2021)

Something to keep in mind is that putting in place metrics always signals what you find important. Even just starting to measure metrics can sometimes change behaviour. Most metrics are also very heavily context dependent. For example having a good MTTR doesn't mean much for a team that doesn't implement many new features that could cause failures. There can often be a reasonable or even expected reason for why a metric might be lower or higher for some team than another. Often this is tied to the nature of the work they do. This is why comparing teams directly by their DORA results should not be done. You should be comparing your teams previous results to see how you are improving. Improving DORA metrics specifically should also not be strictly tied to objectives or incentives since they are not a goal in and of themselves. The DORA metrics are trailing indicators of who well you are doing so you should think of them as health indicators for how well you are practicing productive DevOps. They measure systems not people. Improvements in DORA metrics are meant to correlate with reducing batch sizes, increased quality, faster feedback, happier customers and happier development teams, which all feed into productivity. (Forsgren, Humble, and Kim 2018)

It is not enough to just start measuring and hope that now the rest will take care of itself. If your goal is to improve the metrics you have to start taking steps to address them. The first step is keeping your developers in the loop of what is being measured and why. During the analysis of the results it is also necessary for the developers to give feedback and explanations for everyone to get a better understanding of the systems in play. In order to see improvement in the metrics you need to guide your developers in possible ways to improve processes, but also listen to their suggestions. The developers working with the systems daily probably have good ideas on what needs improving and you should absolutely work together to improve. Once you have made improvements to your systems you should monitor the DORA metrics to see how the new changes affect your productivity. Sometimes the solutions to these issues are not evident and they require experimentation, which should be encouraged.

4 Software metrics in The case company

The case company is a Finnish company dealing in the manufacturing and service of cranes and other lifting equipment. As it is such a large company there is also a quite large and growing software sector inside the company. Some of these software solutions that the case company provides include remote monitoring and maintenance assistance of lifting solutions. I work in the software group that mostly focuses on the company's customer portal and related services which is called the Digital Center of Excellence (DCoE). This thesis was started to dig into possible future improvements of the DevOps practices and productivity of the software teams in DCoE. More specifically to look deeper into software metrics and what the academic research has to say about different options and their validity. Another purpose of this thesis is to see where there might be challenges or limitations in the practical application of the academically studied metrics and to set a guideline on how to set up a digestible dashboard that can be used for metric analysis and tracking.

DCoE contains multiple feature teams which focus on the customer portal experience layer each in charge of their own specific applications. In addition to the feature teams there is a team dedicated to processing the data required called the Cloud datalake team and the Edge enablement team which focuses on IoT solutions and machine data collection. All of these teams are able to work independently, but constant collaboration and knowledge sharing is required between teams. There are also other DevOps teams inside of the entirety of The case company as it is a large global company. For this work the scope will be limited to the feature teams since it is the work I am most familiar with and have the most access to. Each feature team will be treated as a separate DevOps team in terms of the measured metrics, but metrics combining all teams might also be used and studied depending on possible limitations and practicalities.

There are some existing talks and conversations about software metrics inside the DCoE. Some of the internal goals, objectives and intended outcomes regarding development metrics were mentioned in a presentation presented by a manager at DCoE. The purpose of the presentation was to provide a common framework to measure the efficiency and effectiveness of the digital development teams and define the metrics necessary for that purpose. The idea is that once the metrics are defined they are to be measured and analyzed continuously as a means to define the continuous improvement actions for the development teams. The final outcome will be to have impactful and actionable metrics defined to the digital development teams. The four categories of metrics that need to be defined are Business delivery efficiency, DevOps process Efficiency, Quality practices effectiveness, and value generation and protection metrics.

4.1 Metric definitions

Relating to business delivery performance different flow metrics, scope creep, and predictability are mentioned. First is flow time which measures time to market by following Jira ticket status updates from start to completion. The purpose of this measurement is to analyse how time-to-market is changing over time and especially in reaction to changes in practices or systems. The next metric mentioned is flow efficiency which measures the time wasted on waiting during an items flow. It's calculated by comparing the time a ticket is in an active in progress state to the total lead time. It's purpose is to answer questions about what is most holding up the delivery of features. The third flow metric mentioned is flow velocity which gauges whether value delivery is accelerating. This metric is quite similar to DORA deployment frequency except it is measured by amount of releases per month rather than avg time between releases. Some of these flow metrics are measured already in the case company, but that will be covered in more detail in the collected metrics chapter 4.3.

The last two business delivery efficiency metrics have to do with scope. Scope creep is a measure of how much the scope of one project expands and changes in the course of the project. Often scope changes happen due to increased stakeholder requirements or realisations from the technical development team. The suggested metric for calculating scope creep is tagging new requirements with a "scope creep" tag and measuring the amount over time. It is important to be aware of and in agreement about scope and schedules to ensure you are not wracking up unexpected costs in development. To measure how well the ability to predict and hold onto an agreed scope is maintained, predictability accuracy or hit rate is measured. This is measured by counting the differences between sprint estimations. A typical sprint velocity variance should be within 10 percent.

The next category of metrics measure the effectiveness of quality practices in the company. The first category is defect detection effectiveness(DDE) which is calculated by the percentage of defects detected in the testing phase of the software divided by the total amount of defects. This measurement needs detailed labeling of bugs based on the environment they are found in. Once in place DDE measures how effective your quality assurance practices are in the catching of bugs and other defects. The next metrics measure test automation coverage and effectiveness. Most of the code should be covered by high quality automatic tests, reducing the amount of manual testing required for a high quality product. Coverage is counted by simply how much of the code is covered by automated tests. Effectiveness is calculated by the amount of bugs found by automated tests compared to the total tests executed. It is important that the ROI of automated tests is positive so that resources are not wasted running unnecessary tests. How quality is measured today in the case company will be continued in chapter 4.3.

Value generation and protection distribution metrics were also mentioned as a category. Feature distribution measures the distribution of different types of items released from your value stream. Items are categorized into features, bugs, continuous maintenance and technical debt. The amounts of each item type are then compared to ensure that too much focus is not placed on only some types of development. If only new features are released, future issues of huge technical debt and larger failures might arise, but if the focus is only on bugs and continuous development, new business value isn't generated. One way to measure maintainability is to estimate then count the size of the items labeled under technical debt. The larger your technical debt is the harder it gets to maintain your code base.

The final category in the presentation is DevOps process Efficiency. The four key metrics of DORA deployment frequency, lead time for changes, change failure rate and mean time to recover are used as example metrics for DevOps process Efficiency. As discussed in 3.4 DORA metrics are a good way to measure the DevOps process. Especially when counter balanced with the metrics around workplace culture and satisfaction.

4.2 From collection to leveraging

Now that the definitions of the metrics in each category have been discussed, the collection phase as depicted in figure is moved on to 4.1. For the collection phase, data sources need to be identified and the system toolchain analyzed, which can be

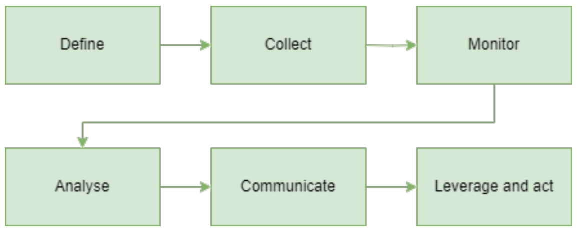


Figure 4.1: Metric planning process

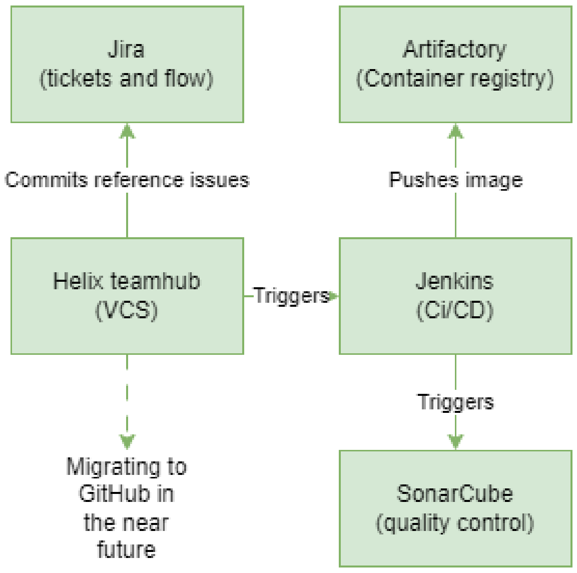


Figure 4.2: The case company system toolchain

seen in figure 4.2. Once the key data points that can be extracted from each part of the toolchain are figured out, the collection can begin. When the data has been defined and collected, the monitor stage is moved to, where the frequency of data refresh and how to visualize the data is explored.

Once the visualization is in place, the analysis of the results can be started. From this phase onward the business side gets more and more involved. Especially with the communication and leverage phases, where insights are shared with all the affected stakeholders and then used to justify investments and future development.

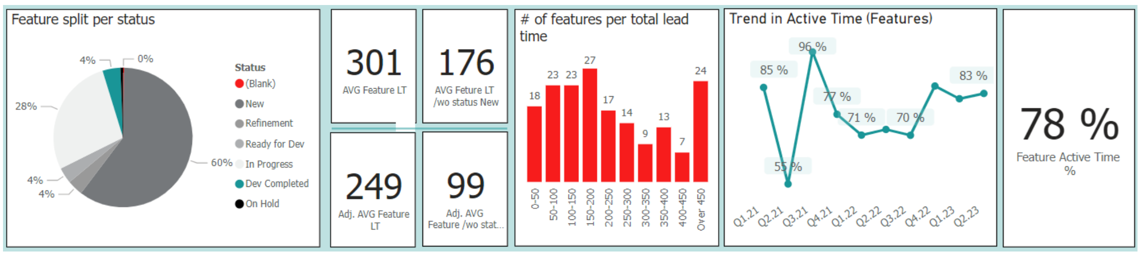


Figure 4.3: Lead time and flow dashboard

4.3 Collected metrics

Let's then look at the current status of the metric categories and their representative metrics mentioned in the presentation. Starting with business delivery performance, it can be seen that the expected things to measure are FLOW and scope. For flow metrics, an up and running dashboard is available in the case company from which things such as feature split, feature lead time, and active time can be seen. All of the metrics are calculated based off of Jira tickets and their flow. In the figure 4.3 you can see a snippet of the presented metrics for all of the 5 feature teams which can be used for analysis.

From here, it can be gathered that on average it has taken 306 days to get a feature from being requested to being completed in the years 2022-2023. Excluding the new status helps us calculate the time it takes for a feature to be finished once a task is actually starting to be developed. In this case the result for that is on average 89 days for all teams. This is quite an important distinction, because as can be seen in the feature split status, most features spend almost three quarters of their lifetime as "new". But what is more interesting is the trend graph for lead time over time to see if the company is improving their lead time or the opposite. This was not directly available on the existing dashboard so I created my own trend graph based on each quarters adjusted average lead time and lead time without new status results since 2021 when the data gathering began. The findings from figure

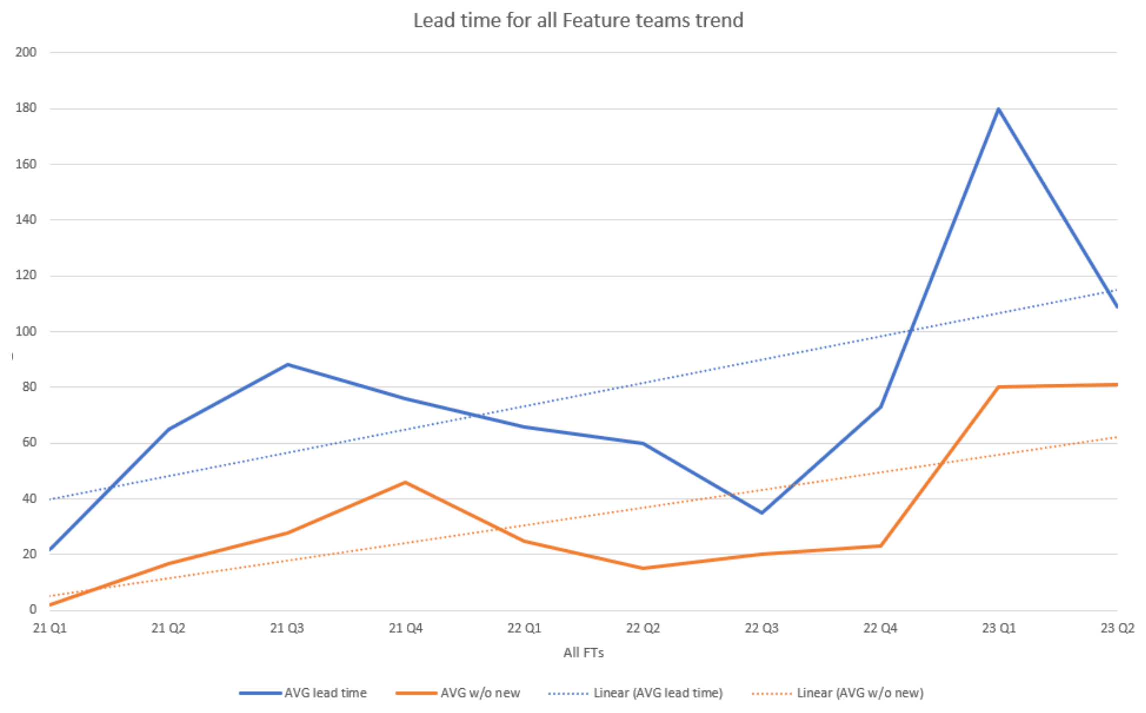


Figure 4.4: Lead time trend for all Feature teams

4.4 is that the lead time has actually increased over time. From figure 4.5 it can be seen that the average time spent in progress is nearly the same, but there are less tasks waiting to be worked on. The median is however surprisingly different as the length of the refinement phase is quite long and has also increased.

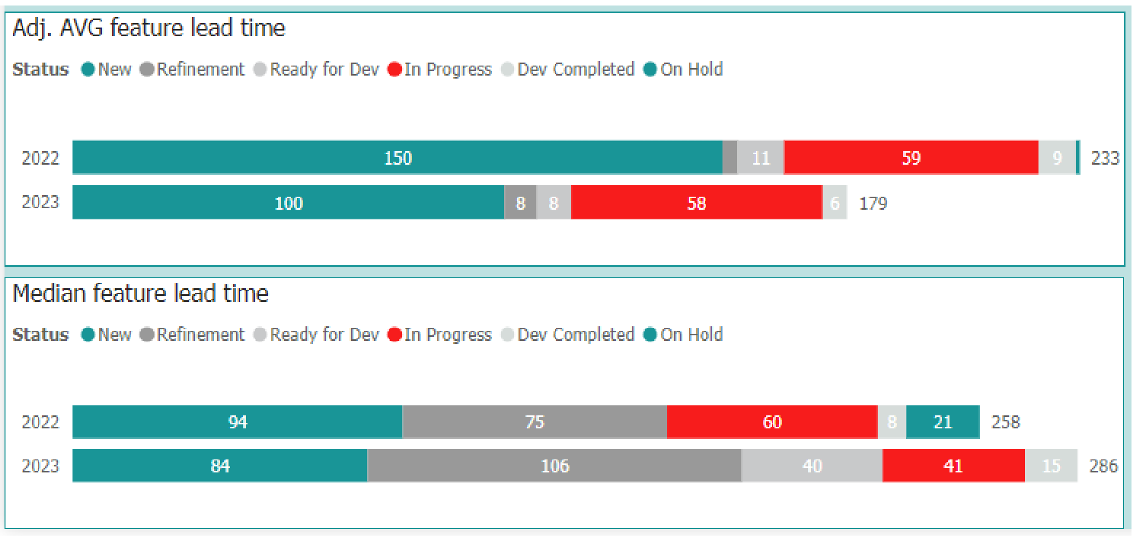


Figure 4.5: AVG and Median feature lead time

5 DORA dashboard

The goal of this thesis was to understand the existing software metrics through the academic lens and then to use that understanding for a practical application. Since the DORA metrics were such a staple in the available productivity measuring literature, the obvious practical approach was to collect, measure and display them. Through this thought process came the next step of creating a DORA dashboard for the teams inside the case company, where the DORA metrics would be accessible by the managers and software teams themselves. Having these DORA metrics would then help to identify pain points in our shared pipelines and perhaps other insights as well.

To execute this idea would require the collection of data, calculating of the metrics, displaying the results visually and gathering feedback of the final result. Using the academic papers as a base would help avoiding pitfalls many would run into in the processes and also help creating a final outline of what to gather from the results of the metrics. The user guide so to say will be an important part of the project itself. I will be referring to this project as the DORA dashboard throughout this thesis.

5.1 Expectations and plan

In this chapter I will go over the expectations and plan for building the DORA dashboard. The project has multiple parts to it so each of them will be tackled one by one. To start I planned on taking the knowledge accumulated from the literature review and tried to draw definitions for each of the metrics that I required for the final dashboard. My expectations were that there would be many different ways of working that differed between the teams and I would have to use my judgement especially in the definitions of bugs and recovery. To get all four of the key four DORA metrics defined I would need a specific definition for the following concepts deployment, initial commit and production incident. These definitions would need to be specific enough to build measurements around them. I would also need to identify what information I am privy to through the various software development pipeline tools used in the company.

The next part of the project was to figure out a way to gather all of this information from different data sources and to unify them to be used in calculations. For this purpose I would have to use an existing tool meant for the job or in the worst case do the calculations manually. Thankfully most of our company's tool-chain seen in figure 4.2 is quite standard so it is possible that there are existing tools for this purpose. Manual calculations would make it much more difficult to automate the process and therefore make the end result less useful and time consuming.

Visualization of the results should be quite straight forward. If the rest of the pipeline works well it's just a matter of storing the data somewhere, doing the calculations and drawing graphs. To add functionality such as filtering by team or changing the data processing on the fly would be a possible cherry on top.

Once the dashboard is up and running it should be validated to some level to confirm the results are as they seem. This could be done by manually calculating some snippet of data and comparing or just by getting a second look by someone who has knowledge of the team's inner workings.

Finally after the validation the results can be presented to the affected stakeholders. The presentation should contain the current situation and perhaps a bit of a retrospective on how the team's DORA metrics look and how they have looked. It should also contain information and insights that I've gleamed about the teams and how I came to these conclusions. Then there should be a mention of how to go forward with the dashboard information and what it should be used for and my general understanding of DORA and the other possible metrics that could be recorded in the future.

The last step after everyone has seen and understood the dashboard and my findings it would be good to survey the stakeholders on how useful they find this information and software metrics in general for their own work. It would also be interesting to see if the agree that this is the best way to measure productivity and if not what ideas they have. There are also important findings in papers like (Storey, Zimmermann, Bird, Czerwonka, B. Murphy, and Kalliamvakou 2021) where they discuss perceived productivity and how it differs but also feeds into the pure mathematical idea of productivity.

5.2 Defining the needed metrics

Once the plan had been laid out it was time to execute it. Reading through the literature I was faced with multiple slightly differing opinions and definitions for the key four DORA metrics. Starting with the most straight forward one Deployment frequency, the definitions were quite unified in that deployments would be counted as successful production deployments. I identified that this information would be the most accessible in our Jenkins CI/CD tool. Production deployments are identified in Jenkins by the "prod" tag in the respective deployment job for each application. There is some variation between applications on the naming convention of deployments, since sometimes they are referenced to as releases or production deployments instead of just the word deployment, this is due to some of the applications being built up to a decade apart from each other.

For Lead Time the definition for deployments needs to be incorporated with the definition of an initial commit or pull request. Some way for the Jenkins deployment to be connected to when the first commit was pushed to the correlating feature branch is needed. The information of the commits and pull requests are stored in our repository tool Helix TeamHub, which is the most non standard tool in our pipeline. This caused some issues since there was no good way of extracting the information out of the tool due to our development practices. Since the developers always deleted the initial feature branch when merging to new one the information of the initial commit was not saved anywhere and made calculating the lead time quite difficult. Thankfully during the writing of this thesis the repository tool was migrated from Helix TeamHub to GitHub so the calculations were possible again.

The last two metrics Mean time to recover and Change Failure rate both require definition for an incident or failure. Incidents were the most difficult to define, since there are many ways to interpret them (Sallin, Kropp, Anslow, Quilty, and Meier 2021). This information is definitely contained in Jira the ticket system in which bug tickets are created. However, including all tickets labeled as a bug as incidents is not enough, because minor bugs or non-production bugs should not be included. Additional filtering is required to include only major or critical production-based bugs. Once this is done, the mean time to recover can be calculated by using the time these major production incident tickets are open and dividing the time by the amount of incidents reported.The change failure rate is calculated by dividing the number of deployments that cause incidents by the total number of deployments, as seen in figure 5.1. It is especially important for the change failure rate incidents to be tagged by application so they can be properly connected to the deployments in the right application. If the bug tickets are created properly with all the necessary information attached this should be quite a rigid definition for incidents.

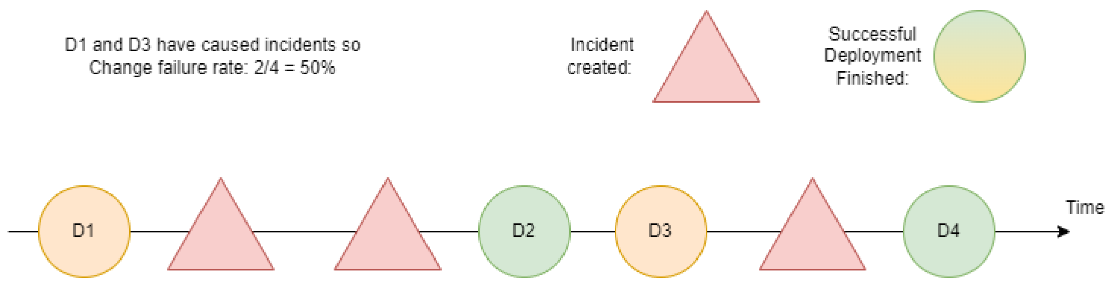

D1 and D3 have caused incidents so
Change failure rate: 2/4 = 50%
Incident created:
Successful Deployment Finished:
D1
D2
D3
D4
Time

Figure 5.1: Change failure rate definition

5.3 Building the metrics

Before starting to build something from scratch I decided to get an understanding of the existing tools available for this purpose. This turned out to be quite a crucial decision since building from the ground up would require creating lot's of data queries, difficult and labor intensive unifying of different data and a large amount of database storage which was both expensive and overly complicated sounding. I started by comparing about 10 different DORA metric tools I could find with simple search engine searches. The tools I compared included Swarmia, Linear B, Sleuth.io, LeanIX and Apache Devlake. Apache Devlake stood out as a free open source tool which supported most of our toolchain and seemed the most customizable out of the options. It also had good documentation with step by step guides and clearly defined queries and processes. It also supprted webhooks which I assumed could be used to integrate HelixTeamhub which is not very industry standard and therefore not supported by any other tool either, this however turned out to be a mistaken assumption. It also included Graphana which was a tool available for visualizations in the company toolkit.

Once it was confirmed as a secure and safe to use tool by our internal security team, I proceeded to start the integration of Jenkins and Jira into the software. All the testing was done in a sandbox environment locally in order to avoid any possible safety concerns with sensitive data such as emails and names of employees. There were some initial difficulties with configurations, but thanks to the quick response of the lead developers at Apache Devlake it was all sorted. The only issue left was with the Helix Teamhub configuration, but that only affected the lead time metric. Since an available proxy metric was already mentioned in 4.3, it was decided to move on without that data until the GitHub migration had taken place.

The Apache Devlake graphical interface is quite intuitive. First I had to configure the data connections for Jira, Jenkins and later once the GitHub migration was finished Github was configured aswell. My initial attempts at configuring Helix Teamhub failed, but thankfully I got lucky with the timing of the migration and was able to proceed. Then I created a separate project for each feature team so that I could track each team's metrics separately. One team at a time I scoped which Jira board contained the team's tickets, which Jenkins builds contained the production builds for each team's applications and finally which Github repositories were under each team's ownership. The next step was to add transformation's to the scope. Transformation's contain details about and define for example bugs, deployments and code-reviews. These transformations then help the software identify the proper data to create accurate graphs and tables. In my Jenkins transformations I identified that not all production deployment's in each pipeline were named identically, therefore I had define deployment's as (deploy to production|deploy gateway|deploy|release|deploy kc to production) instead of the default (deploy|push-image). In the Jira transformations I had to enter which tag is used to identify bugs in the bug tickets. Once all of this was done I started the process to build the dashboard and once it finished I had a functional DORA metrics dashboard.

One issue I ran into with this method was shared repositories. Since there are a couple of repositories in which multiple teams release and develop features and this process does not identify which deployment comes from which team. I decided to contact the developers at DevLake to get their opinion on the matter. I did not get a resolution from them so I decided to continue with just accepting that some deployments in shared repositories will count for multiple teams. This actually is not that big of an issue currently since most deployments contain changes from multiple teams usually anyway. This might become relevant if the teams start increasing their deployment speed and therefore decreasing the likelihood of deployments containing work from multiple teams.

During the validation phase, which is gone over in more detail in 5.5, I found that the metrics relating to incidents were not valid enough, because of the way incidents were being picked up by the Devlake platform. Because of this I had to calculate the two stability metrics manually. This was done by pulling a dump of all feature team related bug tickets in Jira. These bug tickets were then filtered to get a more accurate sample of what would be considered incidents in our applications. This filtering was done based on environment, severity, team associated, status, and date. In this way, only fixed production bugs of major severity or higher that were done by the correct team and within the wanted time-frame could be precisely filtered. This was not possible with Devlake since their filtering options were limited.

Once I had gathered all of the incidents in this way, I pulled deployments directly from the Devlake platform since they were captured correctly. With both of these available I then put them in the same table in excel referring combining them based on the date-time field of either the incident being created or the deployment being deployed to production. Once in date order I went through manually and labeled each deployment as either successful or failed depending on if it was followed by an incident or another deployment like is depicted in 5.1. Then I calculated the change failure rate by comparing the number of failed deployments to all deployments.

MTTR was much simpler to calculate manually by simply taking the difference between each incident's opening date and closing date, then taking the mean. I did this on a quarterly basis rather than monthly like the other metrics to get a more descriptive mean number since some months for some teams might've only had a couple incidents.

5.4 Hosting the dashboard

With the configurations done the next task was to deploy this dashboard to make it accessible. Up until this point I had configured and ran everything on my local machine for both security and practicality reasons. This dashboard needs to be available for both the developers and their overseers for it to be truly useful. As can be seen from the documentation provided by DevLake (DevLake 2023), DevLake is a platform composed of multiple components. These components run separately in their own docker containers. This means that only the database and dashboard layers need to be hosted in our own secure server. For this purpose, AWS is used as a hosting platform for both the database and dashboard layers.

Going over the hosting options I decided on an EC2 instance to run Devlake from. The type of instance I used was c5a.large, since smaller instances ran into performance issues. Once everything was up and running I reconfigured Devlake the same way I had done it before locally. One new issue that propped up was the existance of two factor authentication on our company Jira configuration. This made it impossible to access Jira with the Devlake api call without some work around, since the EC2 instance was not in the company network which is the way I got around that issue locally. At this point I had to contact the technical support team who handles the Jira and two factor configurations. They ultimately decided that it is not safe for security reasons to host the dashboard in an online EC2 instance, which threw a bit of a wrench into my work. Options like whitelisting or blacklisting ip addresses, adding lots of security steps onto the EC2 instance itself and using a proxy server for Jira were all considered, but deemed not worth it for this project.

As soon as this decision was made I had to take the dashboard into a different direction to still get it accessible by all of the users. The simplest way was to simply make an excel table copy version of the Devlake dashboard. So I extracted all the DORA metric related information manually into the excel. Once the excel was finished I could host the file on our company cloud service OneDrive and gave individual access to all of the feature team members. The future plan is to get this process somehow automated so that I would not have to manually create a new excel whenever this data needs to be looked at. One way to do this is to skip the calculation process on the Devlake side and use it as a data collection tool and then dumping that data into our existing Power-BI reports which could have the calculations integrated.

5.5 Validation

This data is not helpful and can even be dangerously misleading if it is not accurate to reality. I took multiple steps to validate that the data is being handled correctly. My first step was to do a manual review of the data connections. In the available dashboard options there was a dashboard for each data connection detailing what data it had received in the end. Using this to my advantage I could open for example a single Jenkins job or GitHub repository and compare it manually with the actual job or repository to confirm it has correct data.

My next step was to validate with the developers and product owners that the data is accurate in their opinion as well so that I had not overlooked anything in the process. This is also where I demonstrated the dashboard for the first time. I presented exactly how I had configured my data connections and transformations. Finally I had a short review of the results, what they mean and how you should understand and improve them. At the end of the presentation I then handed out a survey to everyone involved that had some questions about the usefulness and validity of the results. The survey is covered in more detail in 6.

Through these user sessions I ran into some data validity issues. Firstly I already caught in my initial manual review of the Devlake data that while the deployments and pull requests were what I expected, incidents were not. The incidents were all over the place some being completely unrelated to the team that it was assigned to in the dashboard. Since the Devlake configurations for incidents don't have lot's of options in the version I was using I was unable to pinpoint what kind of tickets are of a high enough severity level to be assigned the incident label, rather now it was catching all bugs, which is a huge issue since incidents are a small subset of bugs. The way Devlake expects to identify incidents is by using a label in each Jira ticket and manually labeling incidents, since this is not in use in any of our feature teams this option was simply infeasible. The way I got around this was doing a manual data dump of each team's Jira tickets and doing the filtering myself. I could then compare these ticket's in excel to the team's deployments to calculate their change failure rate. I also used the raw Jira data to calculate the MTTR using the ticket's closing time compared to the opening of said ticket.

After the manual calculations I once again checked the validity of the metrics and if they were showing what was expected. Since change failure rate is meant to depict how often the software team deploys poor quality production code a measure that does not depict that is not useful. It was determined that since the feature teams work on a very wide variety of different repositories and are also responsible for any incidents found in any of these repos, taking all deployments and all incidents at the same time for change failure rate calculations did not yield proper results. If for example a team released multiple successful deployments in one repo while there were incidents found in a completely separate one in the time between those successful deployments they would get labeled as failed, which would then yield in a very high failure rate. This goes against the spirit of change failure rate so the next manual calculations have to take into consideration connecting incidents to the application and it's deployments were the incident was actually found in.

To get on with the project these findings were noted and also mentioned in the presentation of the dashboard for each team, so that everyone was aware of these existing limitations. With the go ahead of my supervisor I continued to the next phase of the project.

5.6 Results

Once I had fought through the many hardships of this project I finally had a functional DORA metrics dashboard available for all the feature teams and their management to see and use to their benefit. Although not as automated, easily accessible and containing more limitations as I would have liked, it was still a usable metrics dashboard.

The validation phase was now finished and I went ahead with scheduling a half an hour meeting with each affected feature team separately so that the barrier for feedback was as low as possible. A PowerPoint presentation was prepared first detailing what Dora metrics are, why they are being measured, and how the construction of those metrics has been done. Once everyone had a good idea of what was being presented the link to the dashboard was shared so that everyone could have access and then the results along with what kind of steps could be taken to improve and what insights there are to be drawn from each metric were analyzed.

As for the results themselves, since it is somewhat sensitive company information it will not be in too much detail. What it showed was that there are some differences between the teams in deployment frequency and as a metric it is always worth improving, however most teams fell into the high range of the DORA benchmarks. Faster and more frequent deployments is ultimately what DORA and increasing productivity is all about so aiming higher even though there is no issue is not a bad idea.

Next going into the lead time for changes it fluctuated between one week to two weeks on average, which makes sense somewhat since each feature team works in two week long sprints. This in the bench-marking falls in the medium category. An improvement idea for this, if after further research it is determined to be because of the sprint cycle, could be that teams are guided to divorce deploying from the sprint cycle and encourage on demand deployments. Another explanation for this metric is just that the code review and testing phases are taking an extended period of time. If this is the case then improvements should focus on those areas. As can be seen here and highlighted in 3.4 these metrics are health indicators and require further research after some finding is pointed out. There are many ways to improve these metrics it's all about finding the one that solves the team specific issues causing lower level results. In 6.1 one team already identified that this metric helped them notice an issue with too many features sitting in a ready state waiting for testing.

MTTR results for teams were on average in the more than a week category however there are large fluctuations between quarters for all teams. Since some teams have results of even over a month to recover in some quarters this tells us that either there is some very major issue or that more likely incidents that either should not be captured or are just not prioritized by teams are captured. Improving the definition of what is an incident would help keep these fluctuations from happening, for this purpose taking in use the Jira incident tag is not a bad idea. Once the incidents are properly defined it is important that once an incident happens other work can be left on standby and end user usability prioritized instead of taking them in on the typical sprint pace. This is in the end a value question which the managers should ultimately answer, since the trade off could be perhaps a higher lead time for new feature development.

Due to the validity issues discussed in 5.5 change failure rate requires some work on the dashboard side to draw proper conclusions from. While in theory the calculating of this metric didn't seem more difficult than the rest in practice it turned out to be quite the challenge. The connection of Jenkins and Github were simple since commit hashes and deployment ids are shared between the two, but since Jira is more external it would require some kind of sophisticated mapping table to connect the proper deployments to the ticket related. Even with a mapping table there is the issue of tickets being more abstract and sometimes multiple deployments can be part of one ticket or the other way around.

6 Survey

With the dashboard finalized it was time to get opinions from the people affected by the dashboard. These opinions are important for confirming the usefulness of the dashboard and researching the effects of it on the developers and related parties. Ultimately the purpose of this thesis work is to improve productivity, but it has to be confirmed that this productivity does not come at the cost of developer well-being and potential future negative impacts.

To get these opinions out of the affected parties a survey was conducted. The survey consisted of questions regarding demographics, opinions on the DORA metrics dashboard, perceived productivity and work well-being. All of these questions were designed with an aim in mind to either to test the research questions and hypothesis or to gauge the quality and usefulness of the finalized product. The full survey can be found in A.1. The survey was created using google forms and distributed to each member of the feature teams which are included in the metrics dashboards. Before the survey was handed out a meeting was hosted with each team separately going over how to read the dashboard and what kind of insights can be drawn by it's use. I refrained from asking or answering the questions that are related to the questions included in the survey since I wanted that information to be untainted by the meeting. This meeting was necessary so that the respondents had an idea of what is included in the dashboard and an understanding of what it could be hypothetically used for. It was also a good place to verify that the configurations for each team had been done correctly.

The first portion of the survey asks questions about the respondent's position, age, experience in software engineering and which team they are a part of. This differentiation is done to see if your position, experience or type of work affects your answers throughout the survey. My original assumption would be that the answers should be quite similar between your average developer, but there might be large differences with the outliers like someone who only started working or has been working for over ten years.

To start off the next section about productivity the respondents were given a clear definition of productivity. The definition given was "productivity is a measure of how effectively and efficiently you are able to turn ideas into software products". The participants were then instructed to answer the questions regarding productivity using this definition. This was done since everyone has an idea of productivity in their head and for research purposes it is best to have a consistent definition shared by all respondents.

In the second part of the survey questions about the dashboard itself are asked. These questions included things such as do you understand what, why and how the dashboard works and questions about the usefulness or correctness of the dashboard and it's metrics. Sallin et all. conducted a similar type of survey in regards to the key four DORA metrics (Sallin, Kropp, Anslow, Quilty, and Meier 2021). Their survey inspired my own questions since I wanted to both corroborate their findings from over a decade ago and validate the dashboard based on their findings. This section ended in an open question for improvement ideas for the dashboard in the future.

The questions in the next section started with questions about how the respondents felt about their personal and team level productivity and if they perceive the recording of these metrics can have an effect on it. These questions were based in the idea taken from both (Storey, Zimmermann, Bird, Czerwonka, B. Murphy, and Kalliamvakou 2021) and (Beller, Orgovan, Buja, and Zimmermann 2020) that perceived productivity is an important part of so called "objective" productivity. Next they were asked if they think the dashboard measures productivity accurately; this question is crucial for my original research question RQ4 about these metrics being fitting for the current use case 1.2. In their paper Umarji et all. talk about the acceptance of new metrics frameworks form both the managerial and developer perspectives (Umarji and Seaman 2009). In my survey I wanted to have these main findings addressed so that developers do not feel they are negatively impacted by the recording of these metrics. This also relates directly to RQ3 about the metrics having a negative effect on developers 1.2.

6.1 Survey results

Once I had the answers to my survey I gathered them in an excel file and did some analysis. I calculated the average's of my quantitative questions about opinions on DORA metrics and the dashboard into 6.1. In general the results are very promising and there seems to be a high level of understanding what, why and how the DORA metrics are being measured. These findings also heavily correlate with the findings in (Sallin, Kropp, Anslow, Quilty, and Meier 2021). The confidence in the results themselves also shows promise regardless of the limitations that that are talked about in detail in 5.5. These limitations were talked about along side the presentation of the dashboard itself so the respondents were aware of them before answering the survey.

The last three bars in 6.1 move the conversation into the general perception of productivity and developer experience in the case company. It's not surprising that most people feel that they are highly productive as an individual and even more productive as a team, however the surprising result is seeing that, even though they

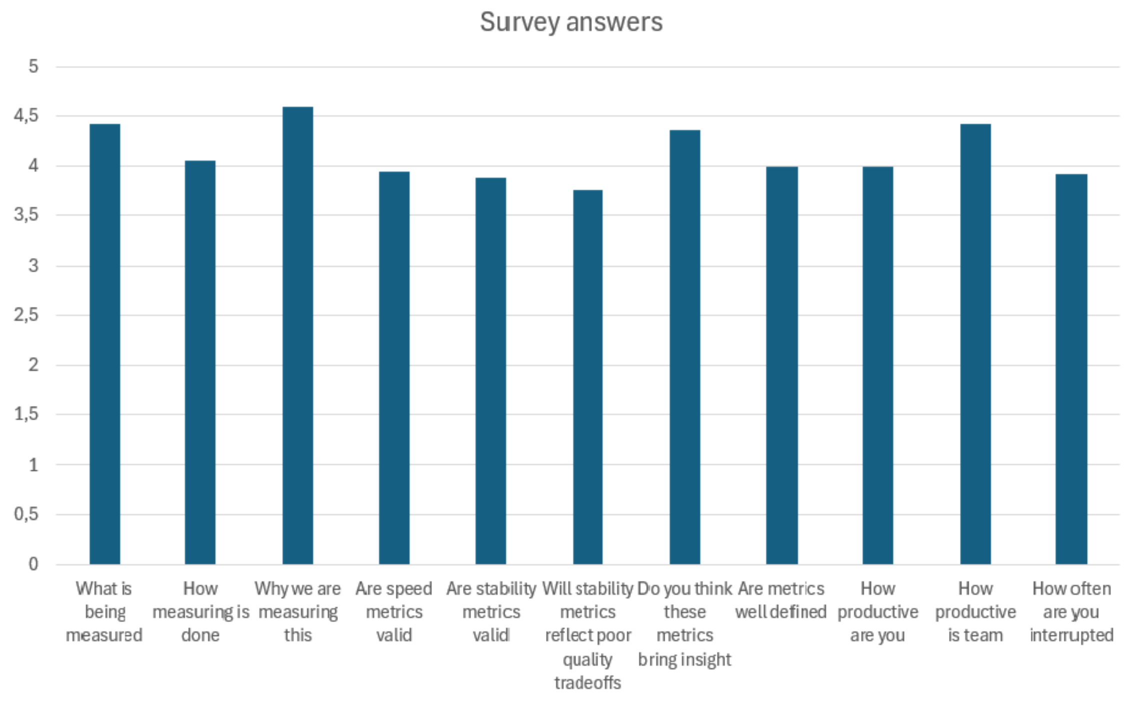


Figure 6.1: Survey results chart

feel very productive their work is interrupted very often. From this it can be inferred that the interruptions are either not very large or so commonplace that they are an expected part of work. In the figure 6.2 it can be seen what kind of interruptions survey participants face. The top three reasons for interruptions in this case were meetings, confirming requirements and helping others. Like Meyer et all. found in (Meyer, Fritz, G. C. Murphy, and Zimmermann 2014) most people find meetings an unproductive activity, so minimizing time spent in non important meetings is crucial for peoples perceptions of productivity. Excluding non coder answers on the question of interruptions brings the percentage of coders who answered they spend too much time in meetings is nearly a hundred percent, which should raise some alarm bells. Since perception of productivity is already so high on average maybe taking a look into the biggest issue almost everyone seems to face might be the key of bumping productive into extremely productive.

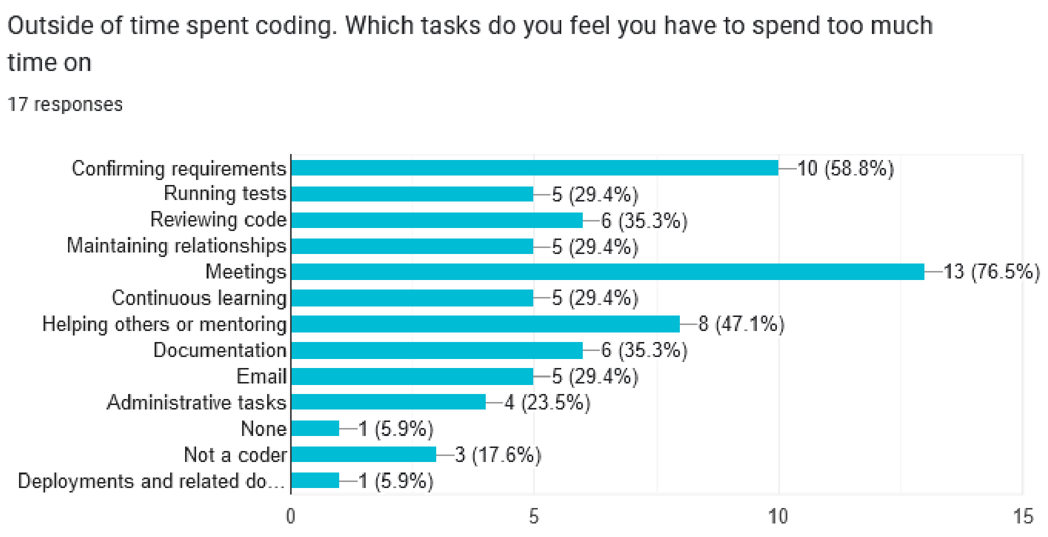


Figure 6.2: Time spent not coding

The survey ended with an open question about additional thoughts or questions which had some interesting answers. A couple participants voiced their small concerns of the data validity and how important it is in these types of projects, but they also expressed that once these small issues are corrected they see the dashboard as extremely valuable. One participant commented that these metrics helped them identify an issue with too many features sitting and waiting for testing and plan to address this. Someone expressed how integral it is to implement metrics like these into the team's normal flow and informing team members of what are the important things to focus on in improving the metrics. A suggestion came up of creating some sort of list of tasks on each ticket which detail how much time spent outside programming was required for this ticket, this would then help identify what is the actual bottle neck. This could be a nice future improvement and falls under the flow metrics which was discussed in 3.4, which are often a nice extra metric to include after the DORA metrics are in place.

6.2 Survey insights

From the survey results it can be seen that the opinions surrounding the DORA metrics, both why and how they are being measured, are mostly positive. Although the sample of developers from a single organization, and even within the same larger team, may not be highly generalizable, the presence of two studies with similar results increases confidence in the findings.

Most developers felt highly productive both individually and as a team, despite frequent interruptions such as meetings, confirming requirements, and helping others. Interestingly, while these interruptions were common, they did not seem to significantly detract from the overall perception of productivity. The feedback indicated that minimizing time spent in non-essential meetings could further enhance productivity.

The open-responses and even my own general opinion is that the concerns about validity and process of refreshing the dashboard data need to be addressed in the future. Even in spite of this participants still found the dashboard extremely valuable. This keen interest in the dashboard encourages to continue in improving and updating it.

7 Conclusions

The objective of the thesis was to answer questions about productivity and the metrics related to it in a DevOps environment. Referring back to 1.2 this thesis strove to dig into the academic findings of productivity measuring, the adverse effects of productivity measurement on both the developers and the software quality, and the practical application of productivity metrics into an ongoing DevOps operation. In this chapter the answers to the research questions 1.2 will be systematically laid out and supporting context will be given to each answer.

Starting with RQ1 "Which metrics have the strongest academic backing in terms of correlation with productivity?". The answer is multifaceted and answered in detail in 3.4. The clear winner in simple volume of academic backing are the key four DORA metrics, however their specific implementation, the surrounding frameworks and supporting metrics are constantly expanding and evolving to this day. DORA is an active research group and their latest report (*State of DevOps Report 2023* 2023) was released merely months before this thesis was concluded, which goes to show that the DORA project is a living and breathing project rather than an established framework. Other supporting metrics are usually built in conjunction with or around the key four DORA metrics, such as FLOW metrics which are important in finding more detailed information about the flow and bottlenecks of the DevOps pipelines used in agile teams. The SPACE framework is a larger framework that DORA metrics are a key part of. In SPACE other factors such as satisfaction, well-being, communication and collaboration are taken into consideration alongside the typical productivity measurements of activity, efficiency and performance. This framework however is so new that it has not been in the scrutiny of academic literature for long enough for it to be considered backed by academic literature.

RQ2 was defined as "Will an increase in productivity through metric based methods negatively effect the DevOps process in terms of quality?". To answer this question the evolution of productivity metrics chapter 3.4 can be referred back to. While it was the case in the past that simple changes were counted in lines of code or methods produced, which seem to clearly lead into incentives of pushing poorly developed, untested and large amounts of code to production fast, that is not the case with these more modern and sophisticated productivity metrics. DORA metrics have built in checks for software quality and long term code stability. This is the reason for splitting the DORA metrics into speed and stability metrics. The fast release cycle which the DORA metrics incentivize also makes it so that code is being pushed out more consistently and in small manageable chunks which makes the job for testers, code reviewers and automation much easier. The DORA metrics also require highly functional pipelines, systems and tools to hit the elite levels of deployments, which in turn feeds into reliable and stable code. So in fact quite counter-intuitively increase in these productivity metrics effects not only quality, but security and reactivity to customer demands positively.

This is also supported by the results of the DevOps productivity dashboard survey 6.1. When developers were asked if the stability metrics are a valid way to measure software stability, they responded positively, indicating a high level of understanding and confidence in these metrics. When asked if the stability metrics would reflect if a team made trade-offs in quality to increase speed, the responses suggested that the metrics indeed help in identifying and maintaining a balance between speed and quality.

To answer RQ3 "Can strictly improving these metrics have negative effects on developers?" in 2.2 the relation of productivity and the well being of developers is brought up. The differences and similarities between developer's self-reported productivity and so called objective productivity are compared. Like mentioned in 3.4 it is extremely important for the people putting new metrics in place, that what is measured is also signalling to the everyone what you find important, therefore when measuring productivity to keep in mind the well-being of your developers is not only important to maximize productivity itself, but it signals to your developers that you find their well-being, effort and improvement important. The key four DORA metrics are also built to be analysed on a team level and trying to use them for singling out less productive individuals is highly discouraged as this most likely will lead into negative outcomes. An important note is also that while the metrics are implemented on a team level that doesn't mean that the results of different teams should be compared between each other to find the most productive team. There are many good reasons why some team's metrics would not look the same as another's and because of this teams should only be compared against themselves at a different point in time. An easy way to lose the usefulness of these metrics is to just measure them and look at the final result. To use these metrics properly they need to be properly analysed and your developers should be given the proper tools, solutions and support to help them eventually improve the metrics. In 6.2 the feelings of the developers in the case company project are gone over and analysed. The survey revealed that respondents generally have a high level of understanding and confidence in the DORA metrics dashboard. Most developers felt highly productive both individually and as a team, despite frequent interruptions such as meetings, confirming requirements, and helping others. To answer RQ3 directly it can be said that negative effects on developers brought by just throwing these metrics at them and asking for improvements is quite likely, but if these metrics are handled with care and you have proper systems in place these negative effects can be avoided and in turn the effects should be quite positive for developers.

The last research question RQ4 is "What metrics should be integrated in the case company?". To answer it simply the key four DORA metrics were clearly the right step in this case, but more improvements to these metrics can and should be added in the future. The way in which they were implemented is perhaps the more interesting part of this thesis and for that the DORA dashboard chapter 5 can be referred back to. In the DORA dashboard chapter 5 the required steps of setting up an automated DORA metrics dashboard using the open source platform Apache Devlake are gone over. These steps are both general in the way that other companies could use this as a guide for the required thought and steps involved for creating a dashboard of their own and specific enough that if someone were to run into similar issues as I have they can be used for the solving of them. From the results of the survey in the results chapter 6.2 this project can be labeled a success and conclude that these metrics were indeed the right ones for this specific use case.

References

2022 State of DevOps Report (2023). Google Cloud. URL: https://cloud.google.com/devops/state-of-devops (visited on 02/08/2023).

Beller, Moritz, Vince Orgovan, Spencer Buja, and Thomas Zimmermann (Dec. 30, 2020). "Mind the Gap: On the Relationship Between Automatically Measured and Self-Reported Productivity". In: *IEEE Software* PP. DOI: 10.1109/MS.2020.3048200.

Brumby, Duncan P., Christian P. Janssen, and Gloria Mark (2019). "How Do Interruptions Affect Productivity?" In: *Rethinking Productivity in Software Engineering*. Ed. by Caitlin Sadowski and Thomas Zimmermann. Berkeley, CA: Apress, pp. 85–107. ISBN: 978-1-4842-4221-6. DOI: 10.1007/978-1-4842-4221-6_9. URL: https://doi.org/10.1007/978-1-4842-4221-6_9 (visited on 05/25/2023).

DeBellis, Derek, Dustin Smith, and Kim Castillo (2021). *DORA | DORA Reseach: 2021*. URL: https://dora.dev/research/archives/2021/ (visited on 05/11/2023).

DevLake, Apache (2023). *Apache DevLake Documentation*. URL: https://devlake.apache.org/docs/Overview/Introduction.

Englander, A Steven and Andrew Gurney (1994). "MEDIUM-TERM DETERMINANTS OF OECD PRODUCTIVITY". In.

Forsgren, Nicole, Jez Humble, and Gene Kim (Mar. 27, 2018). *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.* IT Revolution. 248 pp. ISBN: 978-1-942788-35-5.

Forsgren, Nicole, Marcus A Rothenberger, Jez Humble, Jason B Thatcher, and Dustin Smith (2020). "A Taxonomy of Software Delivery Performance Profiles: Investigating the Effects of DevOps Practices". en. In.

Forsgren, Nicole, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler (Mar. 2021). "The SPACE of Developer Productivity: There's More to It than You Think." In: *Queue* 19.1, pp. 20–48. ISSN: 1542-7730. DOI: `10.1145/3454122.3454124`. URL: `https://doi.org/10.1145/3454122.3454124`.

Junior, Gibeon Soares de Aquino and Silvio Romero de Lemos Meira (Sept. 2009). "Towards Effective Productivity Measurement in Software Projects". In: *2009 Fourth International Conference on Software Engineering Advances*. Porto, Portugal: IEEE, pp. 241–249. ISBN: 978-1-4244-4779-4. DOI: `10.1109/ICSEA.2009.44`. URL: `http://ieeexplore.ieee.org/document/5298403/` (visited on 11/10/2022).

Lima, Jalerson, Christoph Treude, Fernando Figueira Filho, and Uira Kulesza (Sept. 2015). "Assessing developer contribution with repository mining-based metrics". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Bremen, Germany: IEEE, pp. 536–540. ISBN: 978-1-4673-7532-0. DOI: `10.1109/ICSM.2015.7332509`. URL: `http://ieeexplore.ieee.org/document/7332509/` (visited on 11/07/2022).

Maslach, Christina and Michael P. Leiter (2008). "Early predictors of job burnout and engagement". In: *Journal of Applied Psychology* 93. Place: US Publisher: American Psychological Association, pp. 498–512. ISSN: 1939-1854. DOI: `10.1037/0021-9010.93.3.498`.

Meyer, André N., Thomas Fritz, Gail C. Murphy, and Thomas Zimmermann (Nov. 2014). “Software developers’ perceptions of productivity”. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. New York, NY, USA: Association for Computing Machinery, pp. 19–29. ISBN: 978-1-4503-3056-5. DOI: `10.1145/2635868.2635892`. URL: `https://doi.org/10.1145/2635868.2635892` (visited on 11/21/2022).

Murphy-Hill, Emerson, Ciera Jaspan, Caitlin Sadowski, David Shepherd, Michael Phillips, Collin Winter, Andrea Knight, Edward Smith, and Matthew Jorde (Mar. 2021). “What Predicts Software Developers’ Productivity?” In: *IEEE Transactions on Software Engineering* 47.3, pp. 582–594. ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: `10.1109/TSE.2019.2900308`. URL: `https://ieeexplore.ieee.org/document/8643844/` (visited on 11/17/2022).

Oliveira, Edson, Eduardo Fernandes, Igor Steinmacher, Marco Cristo, Tayana Conte, and Alessandro Garcia (July 2020). “Code and commit metrics of developer productivity: a study on team leaders perceptions”. en. In: *Empirical Software Engineering* 25.4, pp. 2519–2549. ISSN: 1382-3256, 1573-7616. DOI: `10.1007/s10664-020-09820-z`. URL: `https://link.springer.com/10.1007/s10664-020-09820-z` (visited on 11/09/2022).

Pilat, Dirk and Paul Schreyer (Jan. 1, 2001). “Measuring productivity”. In: *OECD Economic Studies* 2001, pp. 13–13. DOI: `10.1787/eco_studies-v2001-art13-en`.

Ravichandran, Aruna, Kieran Taylor, and Peter Waterhouse (2016). “DevOps Foundations”. In: *DevOps for Digital Leaders: Reignite Business with a Modern DevOps-Enabled Software Factory*. Berkeley, CA: Apress, pp. 27–47. ISBN: 978-1-4842-1842-6. DOI: `10.1007/978-1-4842-1842-6_3`. URL: `https://doi.org/10.1007/978-1-4842-1842-6_3`.

Sallin, Marc, Martin Kropp, Craig Anslow, James W Quilty, and Andreas Meier (2021). “Measuring software delivery performance using the four key metrics of devops”. In: *Agile Processes in Software Engineering and Extreme Programming: 22nd International Conference on Agile Software Development, XP 2021, Virtual Event, June 14–18, 2021, Proceedings*. Springer International Publishing Cham, pp. 103–119.

State of DevOps Report 2023 (Oct. 2023). URL: https://services.google.com/fh/files/misc/2023_final_report_sodr.pdf.

Storey, Margaret-Anne, Thomas Zimmermann, Christian Bird, Jacek Czerwonka, Brendan Murphy, and Eirini Kalliamvakou (2021). “Towards a Theory of Software Developer Job Satisfaction and Perceived Productivity”. In: *IEEE Transactions on Software Engineering* 47.10, pp. 2125–2142. DOI: 10.1109/TSE.2019.2944354.

The 2019 accelerate state of devops: Elite Performance, productivity, and scaling | google cloud blog (2019). URL: https://cloud.google.com/blog/products/devops-sre/the-2019-accelerate-state-of-devops-elite-performance-productivity-and-scaling.

Umarji, Medha and Carolyn Seaman (Oct. 2009). “Gauging acceptance of software metrics: Comparing perspectives of managers and developers”. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. Lake Buena Vista, FL, USA: IEEE, pp. 236–247. ISBN: 978-1-4244-4842-5. DOI: 10.1109/ESEM.2009.5315999. URL: http://ieeexplore.ieee.org/document/5315999/ (visited on 11/09/2022).

Appendix A Blank Survey

DevOps Productivity Metrics Survey

This is a survey to get Developer and PO opinions on the DevOps Metrics Dashboard. These answers will be anonymously analyzed in my thesis

* Indicates required question

1. Which of these most closely describes your own position in the company *

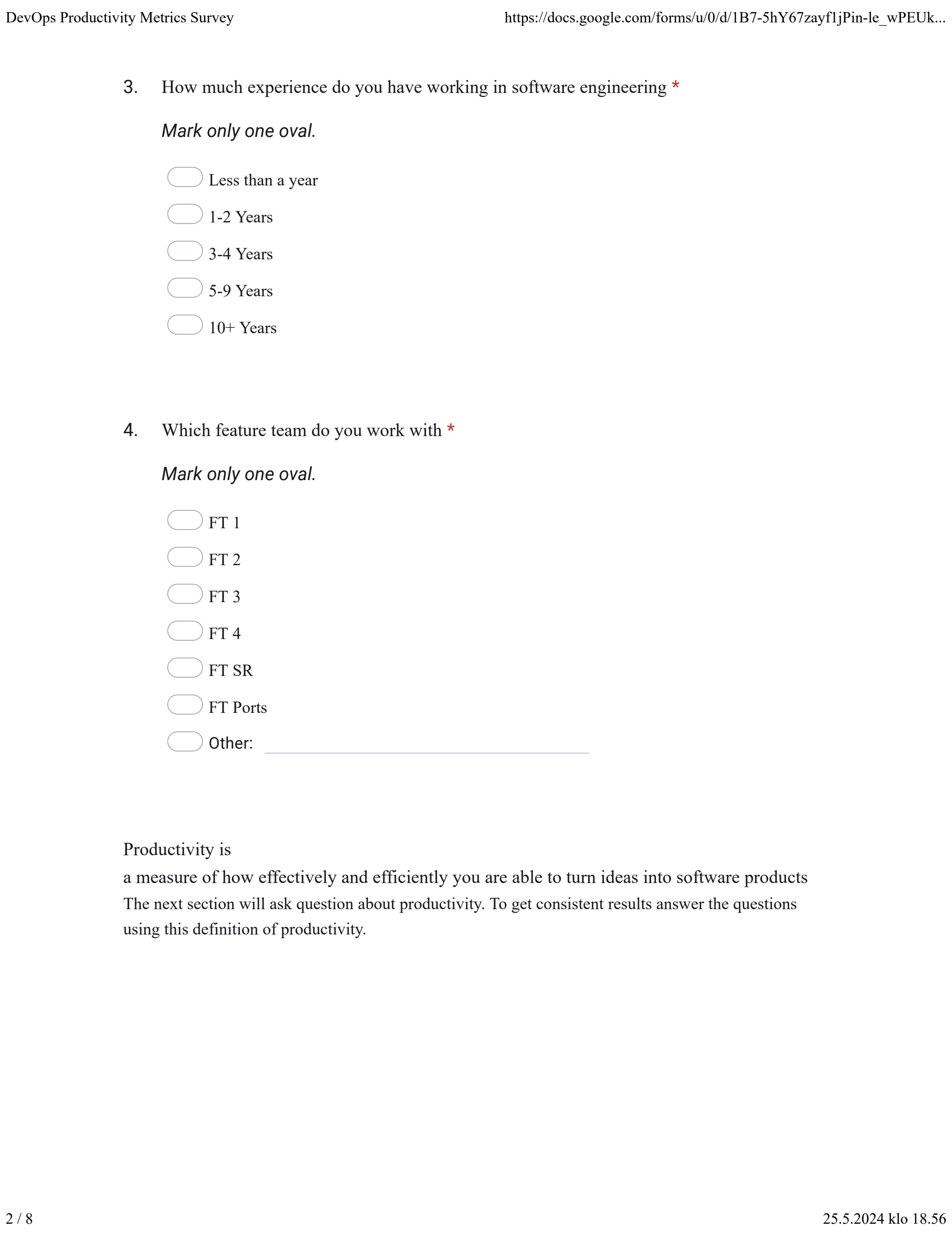
   Mark only one oval.

   ◯ Developer
   ◯ Tester
   ◯ Product Owner
   ◯ Other: ____________________

2. Age *

   Mark only one oval.

   ◯ Under 18
   ◯ 18-24
   ◯ 25-34
   ◯ 35-44
   ◯ 45-54
   ◯ 55-64
   ◯ 65 and older



DevOps Productivity Metrics Survey https://docs.google.com/forms/u/0/d/1B7-5hY67zayf1jPin-le_wPEUk...

3. How much experience do you have working in software engineering *

Mark only one oval.

- ◯ Less than a year
- ◯ 1-2 Years
- ◯ 3-4 Years
- ◯ 5-9 Years
- ◯ 10+ Years

4. Which feature team do you work with *

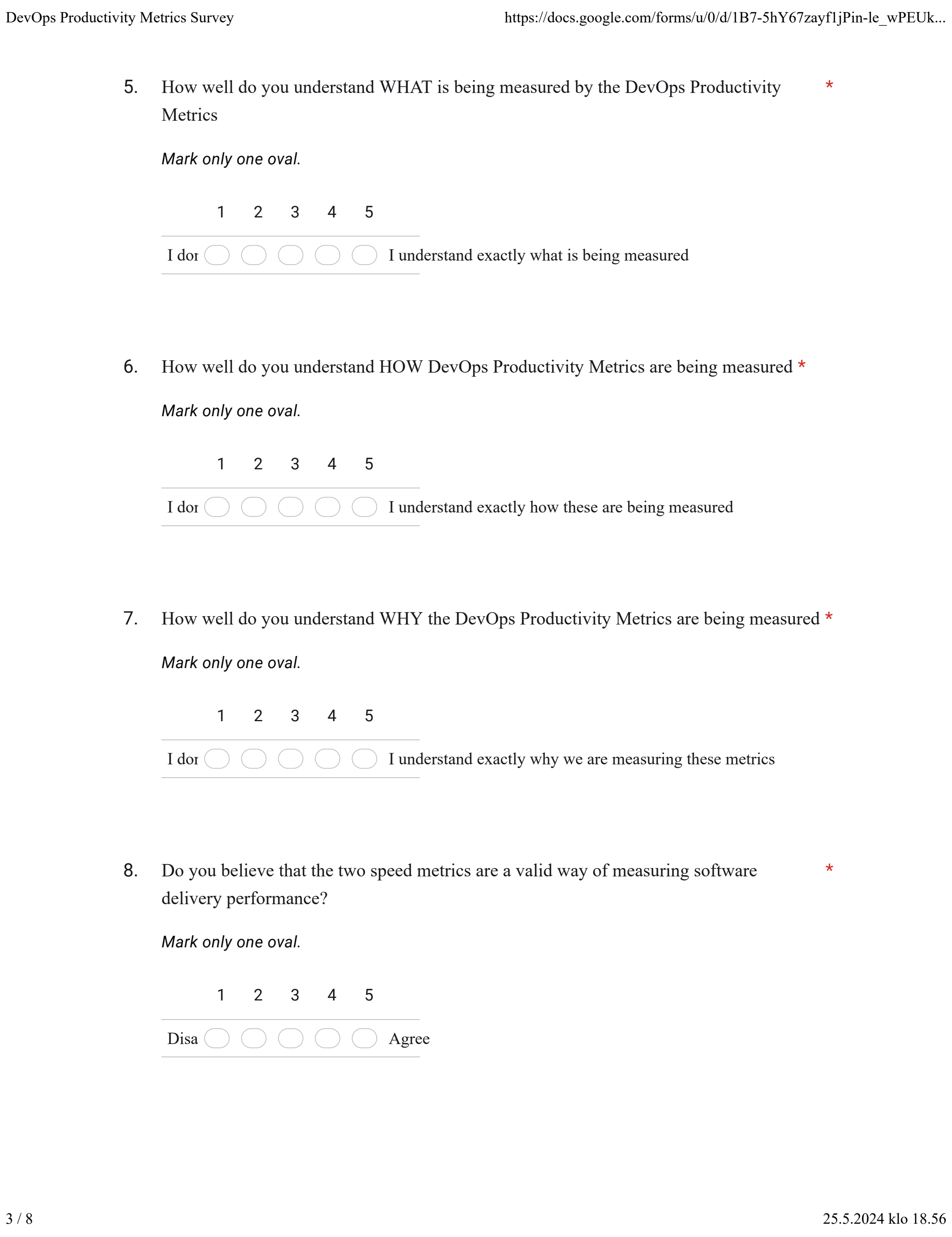
Mark only one oval.

- ◯ FT 1
- ◯ FT 2
- ◯ FT 3
- ◯ FT 4
- ◯ FT SR
- ◯ FT Ports
- ◯ Other: ______________

Productivity is
a measure of how effectively and efficiently you are able to turn ideas into software products
The next section will ask question about productivity. To get consistent results answer the questions using this definition of productivity.

2 / 8 25.5.2024 klo 18.56

Figure A.2: Blank Survey page 2



DevOps Productivity Metrics Survey https://docs.google.com/forms/u/0/d/1B7-5hY67zayf1jPin-le_wPEUk...

5. How well do you understand WHAT is being measured by the DevOps Productivity Metrics *

Mark only one oval.

	1	2	3	4	5	
I dor	◯	◯	◯	◯	◯	I understand exactly what is being measured

6. How well do you understand HOW DevOps Productivity Metrics are being measured *

Mark only one oval.

	1	2	3	4	5	
I dor	◯	◯	◯	◯	◯	I understand exactly how these are being measured

7. How well do you understand WHY the DevOps Productivity Metrics are being measured *

Mark only one oval.

	1	2	3	4	5	
I dor	◯	◯	◯	◯	◯	I understand exactly why we are measuring these metrics

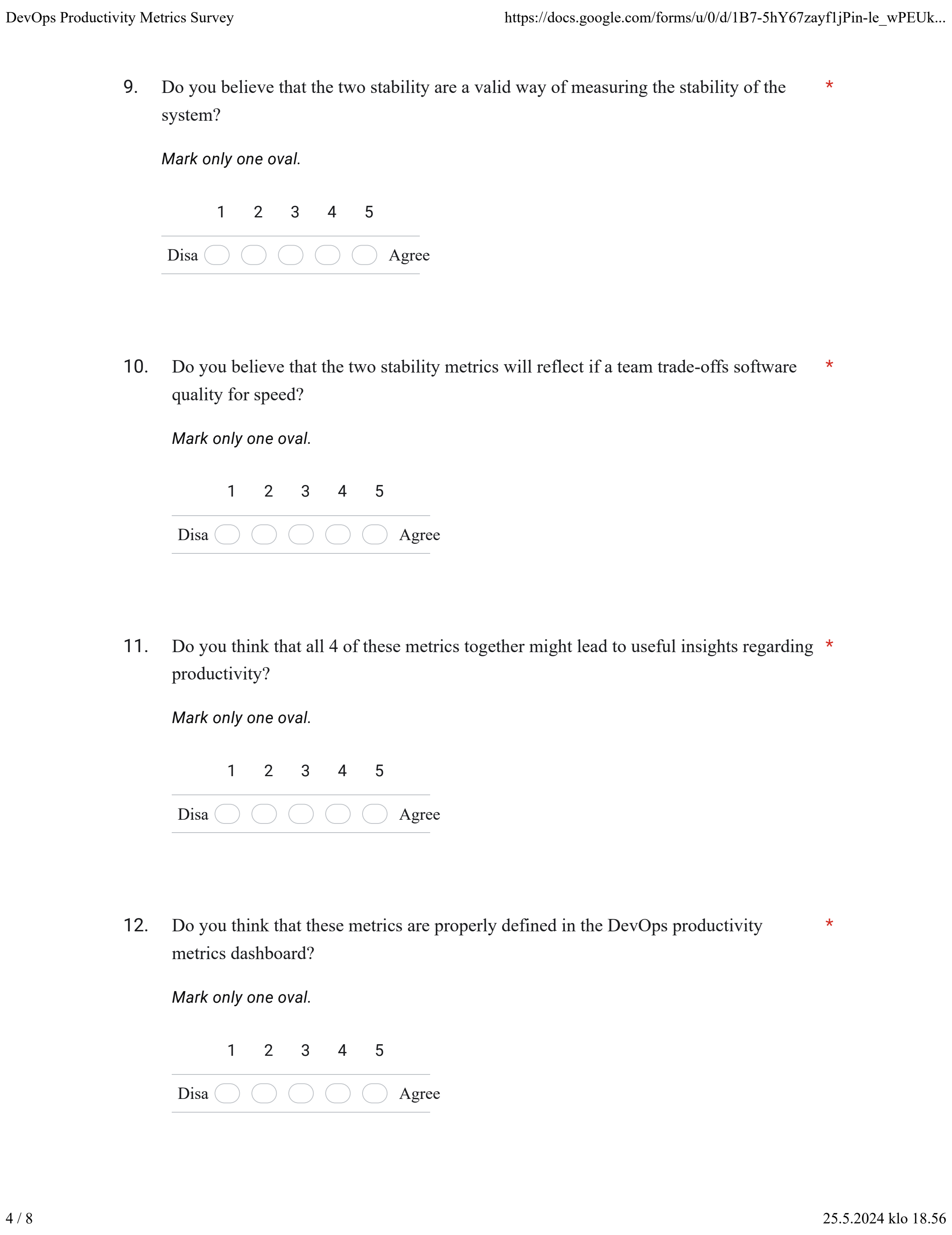
8. Do you believe that the two speed metrics are a valid way of measuring software delivery performance? *

Mark only one oval.

	1	2	3	4	5	
Disa	◯	◯	◯	◯	◯	Agree

3 / 8 25.5.2024 klo 18.56

Figure A.3: Blank Survey page 3



DevOps Productivity Metrics Survey https://docs.google.com/forms/u/0/d/1B7-5hY67zayf1jPin-le_wPEUk...

9. Do you believe that the two stability are a valid way of measuring the stability of the system? *

Mark only one oval.

	1	2	3	4	5	
Disa	◯	◯	◯	◯	◯	Agree

10. Do you believe that the two stability metrics will reflect if a team trade-offs software quality for speed? *

Mark only one oval.

	1	2	3	4	5	
Disa	◯	◯	◯	◯	◯	Agree

11. Do you think that all 4 of these metrics together might lead to useful insights regarding productivity? *

Mark only one oval.

	1	2	3	4	5	
Disa	◯	◯	◯	◯	◯	Agree

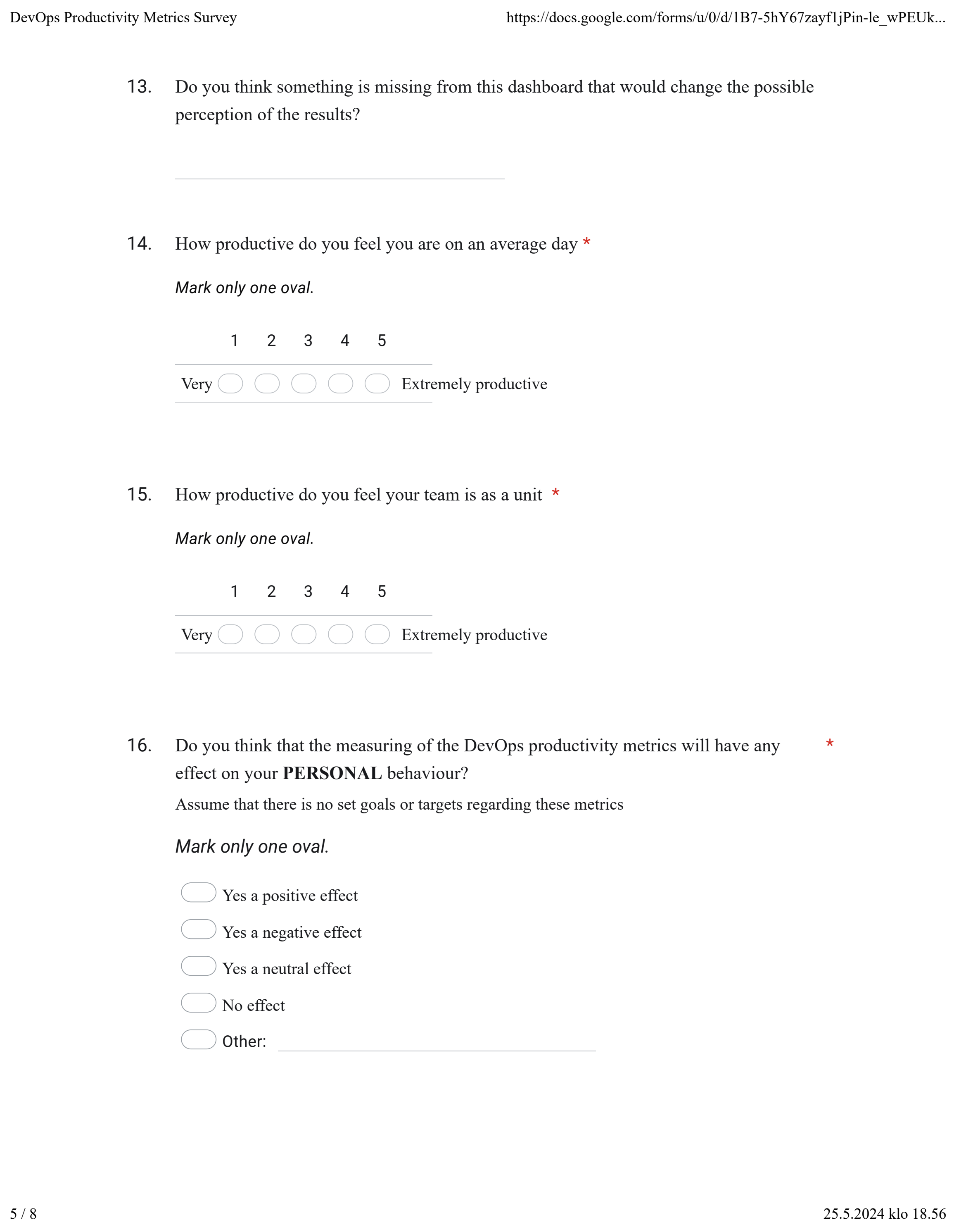
12. Do you think that these metrics are properly defined in the DevOps productivity metrics dashboard? *

Mark only one oval.

	1	2	3	4	5	
Disa	◯	◯	◯	◯	◯	Agree

4 / 8 25.5.2024 klo 18.56

Figure A.4: Blank Survey page 4



DevOps Productivity Metrics Survey https://docs.google.com/forms/u/0/d/1B7-5hY67zayf1jPin-le_wPEUk...

13. Do you think something is missing from this dashboard that would change the possible perception of the results?

14. How productive do you feel you are on an average day *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Extremely productive

15. How productive do you feel your team is as a unit *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Extremely productive

16. Do you think that the measuring of the DevOps productivity metrics will have any effect on your **PERSONAL** behaviour? *

Assume that there is no set goals or targets regarding these metrics

Mark only one oval.

◯ Yes a positive effect

◯ Yes a negative effect

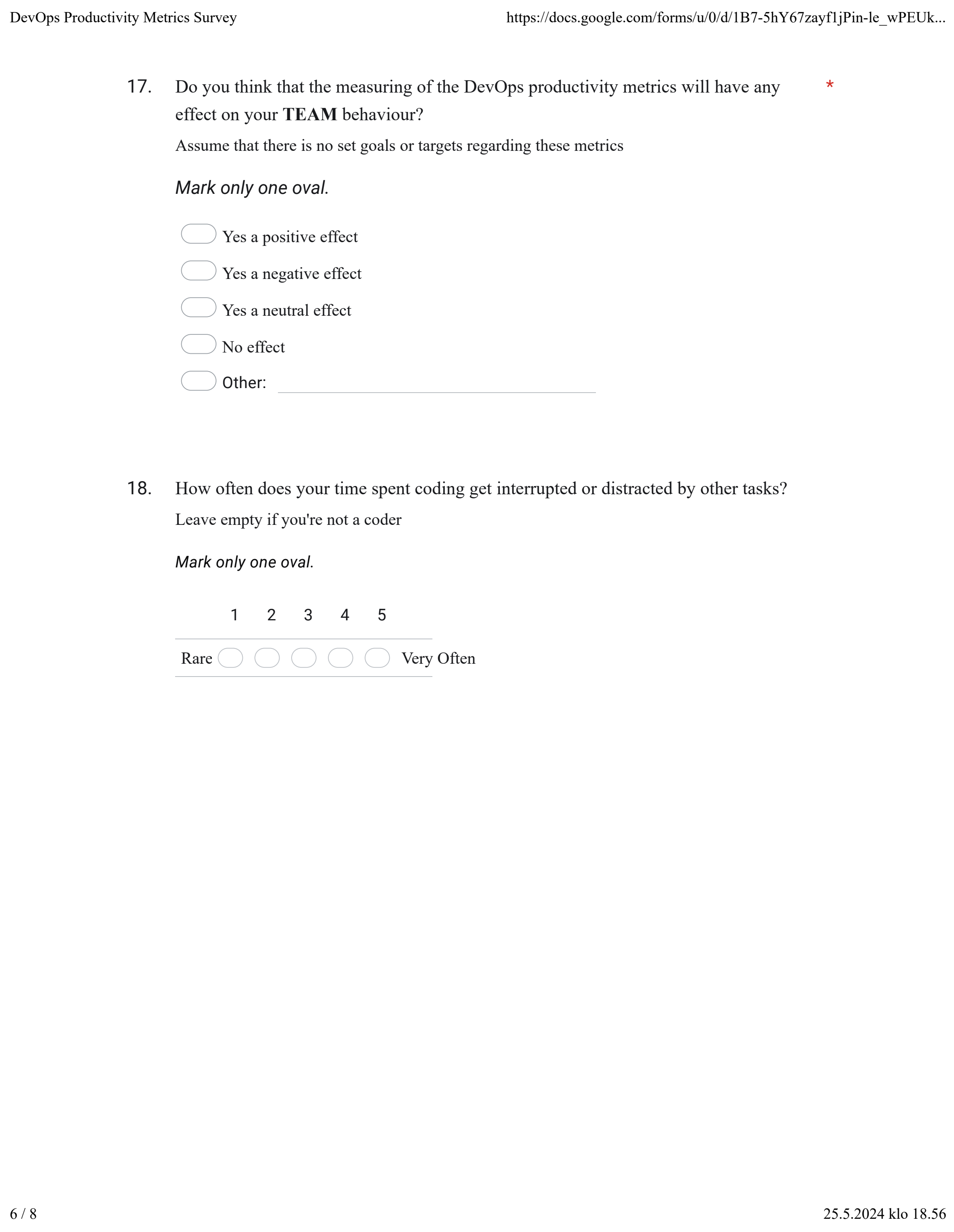
◯ Yes a neutral effect

◯ No effect

◯ Other:

5 / 8 25.5.2024 klo 18.56

Figure A.5: Blank Survey page 5



DevOps Productivity Metrics Survey https://docs.google.com/forms/u/0/d/1B7-5hY67zayf1jPin-le_wPEUk...

17. Do you think that the measuring of the DevOps productivity metrics will have any effect on your **TEAM** behaviour? *

Assume that there is no set goals or targets regarding these metrics

Mark only one oval.

- Yes a positive effect
- Yes a negative effect
- Yes a neutral effect
- No effect
- Other: ____________________

18. How often does your time spent coding get interrupted or distracted by other tasks?

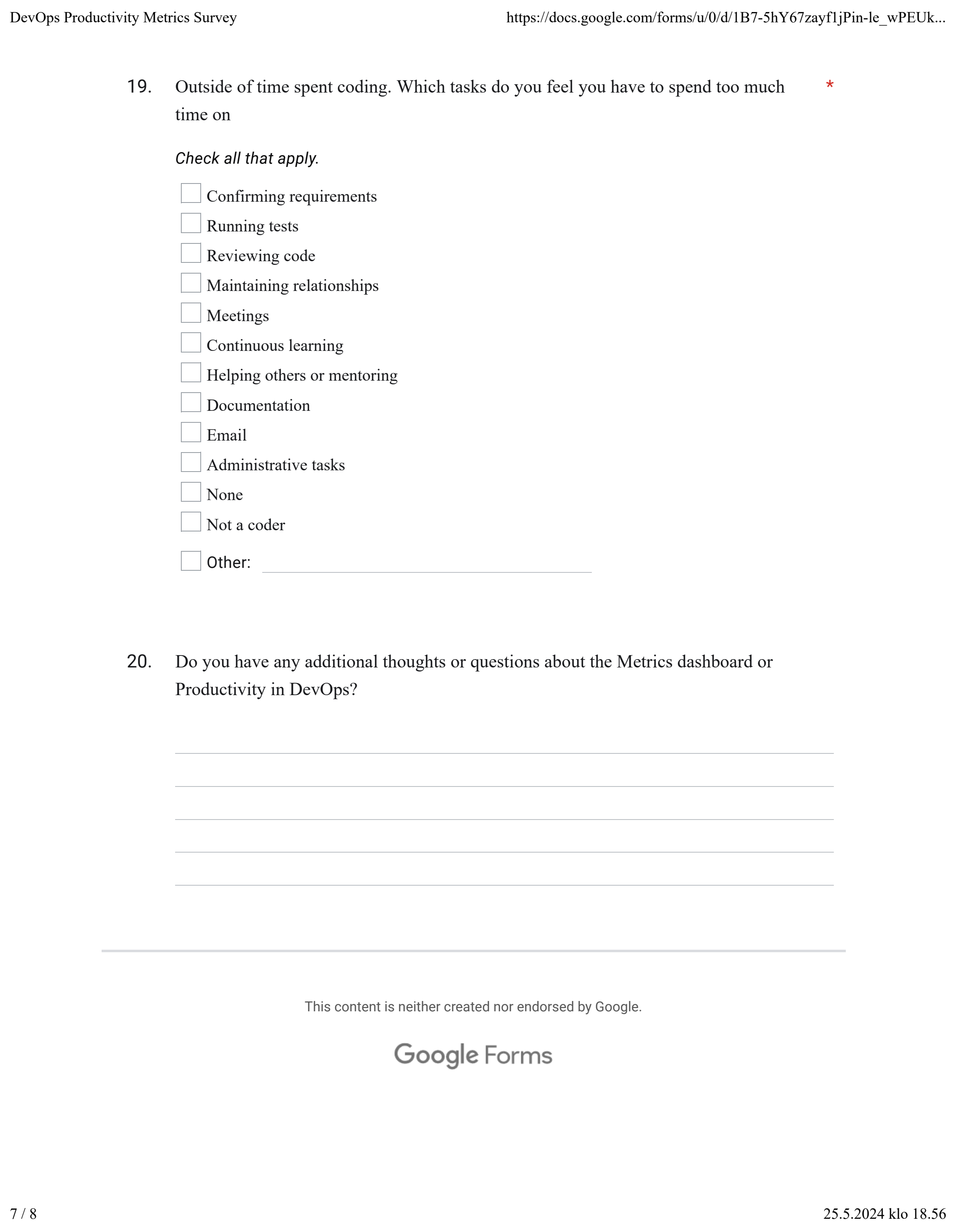
Leave empty if you're not a coder

Mark only one oval.

	1	2	3	4	5	
Rare						Very Often

6 / 8 25.5.2024 klo 18.56

Figure A.6: Blank Survey page 6



DevOps Productivity Metrics Survey https://docs.google.com/forms/u/0/d/1B7-5hY67zayf1jPin-le_wPEUk...

19. Outside of time spent coding. Which tasks do you feel you have to spend too much time on *

Check all that apply.

- ☐ Confirming requirements
- ☐ Running tests
- ☐ Reviewing code
- ☐ Maintaining relationships
- ☐ Meetings
- ☐ Continuous learning
- ☐ Helping others or mentoring
- ☐ Documentation
- ☐ Email
- ☐ Administrative tasks
- ☐ None
- ☐ Not a coder
- ☐ Other: ____________________

20. Do you have any additional thoughts or questions about the Metrics dashboard or Productivity in DevOps?

7 / 8 25.5.2024 klo 18.56

Figure A.7: Blank Survey page 7