DevOps Ideals, Metrics and KPIs

Disclaimer

DevOps is NOT about Docker, Kubernetes or Terraform. It is also not about Cloud, Frameworks or Automation. Instead, it is a cultural mindset with the very fundamental idea:

You build it, you run it, you own it.

DevOps Ideals

The basic idea behind DevOps is packed in the "five ideals" of DevOps. These ideals cannot be translated directly into KPIs, but will give a good idea where to take care of.

I will provide some examples of metrics and KPIs alongside the ideals.

Simplicity and Locality

The **first ideal** is stating, that we need to design things so that we have locality in our systems, processes and organizations that build them. We also need simplicity in everything we do.

This ideal, and measurements based on it, indicate the degree to which a development team can make local code changes without impacting various teams. It also indicates how independent tools and software can be used.

- Measurement: How many teams are affected by a Feature Request?
- Metric: How many teams are affected by single Feature Requests in one month?

Focus, Flow and Joy

The **second ideal** is all about how our daily work feels. Is our work marked by boredom and waiting for others to get things done on our behalf? Do we blindly work on small pieces without seeing the outcomes of our work? How often are we entangled in meetings, fire fighting or see punishment and burnout? Or, do we work in small batches and single work items and get feedback immediately?

The measurements of this ideal indicate how productive, effective and responsible our work is used. Providing new features and fixing bugs can be focused, fun and joyful.

- Measurements: How many meeting time is needed to align work for a feature?
- Metric: How many meeting time is consumed per week to align work?

Improvement of Daily Work

The **third ideal** addresses paying down technical debt, building knowledge and improving architecture. It is not about technical improvement alone. Improvement can and should be

done for all parts of the work. Be it tooling, processes, feedback rounds or communication - improvement should be done regularly. This also includes time to learn new things. It will result in continuously improved and modernized workflow. Teams can delivering better value sooner, safer, and happier.

- Measurements: Which improvement was made after the last sprint/release.
- Metrics: How many improvements can be tracked over one month?

Psychological Safety

The **fourth ideal is** Psychological safety. It is one of the top predictors of team performance. When team members feel safe to talk about problems, problems can not only be fixed but prevented. Solving problems requires honesty, and honesty requires an absence of fear. In knowledge work, psychological safety should be treated with the same importance as physical safety is in manufacturing.

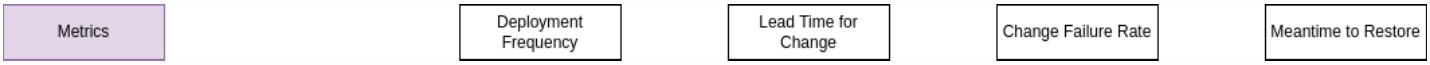
- Measurements: Which problem was raised openly and freely?
- Metrics: How many problems were addressed from the team members in a month?

Customer Focus

The **fifth ideal** relates to customer focus and the difference between core and context work. Core work is, what a customer is willing to invest in or even pay for. Context is what customers don't care about, but that needs to be done to get work done or improve the workflow. You can think of your product as Core Work and everything else, including marketing, internal IT or development infrastructure as context.

Reducing the amount of context work and focusing on core work will enhance the overall performance. Furthermore, the interference of context work should be limited to a minimum and never kill core work.

- Measurements: Which context work is interrupting your work?
- Metrics: How much delay is created by context work in one week?



DORA Metrics

The DevOps Reasearch and Assement (DORA) metrics can be used to indicate the performance of DevOps. They are also known as the **Four Keys**.

Aspect of Software Delivery Performance*	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per day and once per week	Between once per week and once per month	Between once per month and once every six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one day	Between one day and one week	Between one week and one month	Between one month and six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day[a]	Less than one day[a]	Between one week and one month
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0-15%[b,c]	0-15%[b,d]	0-15%[c,d]	46-60%

Deployment Frequency

The deployment frequency metric gives an insight into how frequently an organization releases software to production successfully. With the help of CI/CD, Cloud, configuration management and sane quality gates you can ensure that software is in a proper state and ready for use.

It is a good indicator of process maturity, code quality and story sizes. Small, well tested features can be released more often than complex and huge changes. this also provides a faster feedback loop, where customers and users can provide valuable insights.

Tracking this metric will allow the teams to identify any underlying issues that may be causing delays in release or service. It also often considered, the single most important metric to measure success of a team in different literature, ranging from management best practices, to DevOps or Agile.

Lead time for Changes

If you measure the time from committing code and release to production, you will get the lead time for changes. The general rule of thumb is, that a shorter LTFC is better, since it provides valuable context for changes. Team members don't need to remember features introduced months ago and commits made weeks ago.

The measurement also allows to identify why the lead time is long and which parts can be improved to release faster.

Change Failure Rate

The change failure rate is the ratio of failure and successful deployments in production. This can also indicate the amount of critical bugs and defects that will be visible for customers. A

low failure rate is requested to have a high quality of software that is shipped to the customer and deployed in production.

There are two values required to calculate this metric; the number of deployments attempted and the number of failures in production.

Meantime to Restore

Calculating the time taken by the organization to recover from failure in production is the meantime of restoring service. One of the most crucial DevOps quality metrics, calculating MTTR, should be a standard practice in every DevOps environment.

Since this metric can also be adapted to indicate the fix rate of production bugs, it can be considered as a combination of the above measurements.

DevOps KPIs

In the DevOps world, you can identify a set of important KPIs, that are often used to indicate the success. These are mostly used to provide a more detailed picture of the development process and adoption of the DevOps mindset.

In general, each of them can be applied on the organization, teams and smaller projects. They are not suitable to indicate the success of a single person or commit.

Feature Prioritization

The rate of feature releases is often used to answer multiple questions. For example, you can answer questions like:

- How often do we provide value to the customer?
- How mature is our release process?
- How fast can we get feedback?

If you also indicate the direct value for customers by adding questions like:

- How much of these features were requested by customers?
- How much of these features are solving a real customer challenge or problem?
- How many customers are affected by our changes?

You will get an awesome picture how valuable your product is for your customers.

Customer Ticket Volume

If customers are coming to you with lots of tickets, it is often a sign of optimization potential. Asking questions like:

- How much tickets do we generate per month per license?

- What kind of tickets are openend?
- How many tickets are actually handled?

These questions can help to guide the way forward. For example:

- If the customer often asks for support, the documentation is lacking.
- If the customer finds lots of defects, you need to improve the quality of releases.
- If you don't get any tickets at all, give the customer more options and ways to provide feedback.

Measuring customer satisfaction can be done in many ways. The ticket volume is one that is simple, straight forward and easy to implement.

Defect Escape Rate

We often hear, that "the customer found a bug". This is obviously a very bad signal. Anyway, this can happen and will happen, especially in an innovative process. Measuring the escape rate (how often do we release bugs to customers) is way more interesting than measuring bugs that are found in QA. You can answer so many questions with this single KPI:

- How good is our quality assesment?
- How good are we at detecting bugs before release?
- How critical are these bugs?

And you can start to work on these withquestions like:

- How do we implement quality gates earlier?
- Why do we produce so many bugs?
- Where are these bugs coming from?
- How mature is our testing queue for development/staging/production?

Unplanned Work

Another, very interesting metric is about unplanned work. If you work on unplanned things like critical bugs detected by a customer in production, on-call, last minute features, support cases and whatever, you will not be able to work on planned work (aka the roadmap).

Asking questions like:

- How much time is my team spending on support cases?
- How often do we add unplanned work to a sprint?
- How often is the team interrupted by outages?
- How much work is actually, explicitly planned?
- Do you plan room for job interviews, clarifications, accidental findings?

Will lead to a better picture of the work and how it is used in the team. In the end, it boils down to a simple measuring:

Every minute, that the team spends on unplanned work, is a minute wasted on planned work. Even worse, unplanned work will interrupt planned work and you can easily double the effort to fix something unplanned, compared to the same work if it is planned.