# Measuring DevOps

—------------------------------------------------

This module is all about measurements for DevOps. You will learn the importance of measuring what matters, such as social metrics and continuous improvement goals. You will see how measuring social metrics leads to improved teamwork and how measuring DevOps metrics allows you to see progress toward your goals. You will discover that actionable metrics help you take action toward your desired outcome. One of the key aspects of DevOps is creating a blameless culture. Measuring culture is critical for building a culture in which ideas flow openly and people are listened to. You will learn how that type of working culture can be created. You will see the disadvantages of rewarding one behavior while hoping for a different behavior.

### **Learning Objectives**

* Describe the effect on teams of rewarding for A while hoping for B.
* Describe how actionable metrics are commonly used in a DevOps environment.
* Describe how to measure culture.

—----------------------------------------------------------------------------------------------------------------------------

## Video 1: **Rewarding for “A” while hoping for “B”**

—-------------------------------------------------------------------------------------------------------

This lecture explains the significance of measuring specific goals in DevOps, emphasizing that outcomes align with what is measured. Here are the core points:

1. **Measuring for Desired Results**: The principle “you get what you measure” highlights that teams focus on the metrics they’re rewarded for. If a company wants improved code quality or collaboration, it should track metrics that directly encourage these outcomes. For instance, if you measure lines of code, developers may produce excessive code, focusing more on quantity than quality. But if the goal is collaboration, metrics should include contributions to team code reviews or knowledge sharing.
   * **Real-Life Example**: A software company that wants collaborative and reusable code might track the frequency of shared code repositories and the number of times a developer reuses others’ code. This incentivizes developers to write clear, reusable code, benefiting the team as a whole.
2. **Social and DevOps Metrics**: Encouraging social behavior among developers requires measuring interactions. Two effective metrics are (1) the usage of a developer's code by others and (2) the developer’s reuse of existing code. This dual approach motivates sharing while preventing duplicated effort. By measuring these, organizations foster a more collaborative environment.
   * **Real-Life Example**: If a company measures the extent to which developers leverage each other’s code, developers are more likely to collaborate on shared solutions, reducing redundant code. This ultimately improves team efficiency and morale.
3. **Continuous Improvement with Baselines and Goals**: DevOps promotes setting baselines (e.g., the current deployment time or team size) and defining measurable goals for improvement. These baselines enable teams to evaluate progress effectively, iteratively working towards better performance in manageable steps.
   * **Real-Life Example**: If a company initially needs 10 hours and six team members for each product deployment, the DevOps goal could be to reduce it to two hours and one team member, achieving cost and time efficiency while maintaining product quality.
4. **Shift from Failure Prevention to Recovery**: Traditional metrics focused on avoiding failure (mean time to failure), but DevOps emphasizes quick recovery (mean time to recovery). This approach, combined with microservices and containerization, enables faster response times and smoother customer experiences.
   * **Real-Life Example**: A company deploying an app using microservices can maintain uninterrupted service even if one component fails. If a server or container goes down, a new instance can launch immediately, minimizing downtime and enhancing user experience.

In sum, measuring the right metrics can lead to cultural shifts and continuous improvement in DevOps environments. By focusing on shared goals, collaboration, and swift recovery, organizations can drive quality and resilience in their development and operational practices.

—----------------------------------------------------------------------------------------------------------------------------

## **Video 2: Vanity metrics vs. Actionable metrics**

—-------------------------------------------------------------------------------------------------------

This lecture explores the importance of actionable metrics in DevOps, contrasting them with "vanity metrics," which offer limited insights. Here’s a summary of the essential points:

1. **Limitations of Vanity Metrics**: Vanity metrics, like website "hits," can be misleading. While they might seem impressive, they don’t reveal actionable insights about user behavior or business outcomes. Without knowing what drives a high hit count, it’s unclear what actions to take, if any. Vanity metrics might feel encouraging, but they don’t guide effective decisions.
   * **Real-Life Example**: If a company tracks only website hits, it may focus on increasing visitors without improving user experience. However, tracking user actions, such as purchases, reveals which features lead to sales, helping the company target valuable improvements.
2. **Power of Actionable Metrics**: Actionable metrics provide clear cause-and-effect data, offering insight into what drives positive outcomes. For example, in an A/B test, if Group B (who saw a new feature) spends 20% more than Group A, it’s clear that the feature adds value. Actionable metrics guide informed decisions that support goals and improve customer experience.
   * **Real-Life Example**: A streaming service introducing a “skip intro” feature can use A/B testing to measure if engagement increases for users with this feature. If retention rates improve, they can confidently roll out the feature to all users.
3. **Examples of Actionable Metrics**: Actionable metrics help companies measure progress and achieve specific goals. Key examples include:
   * **Time to market**: Measuring the time to introduce new features allows companies to meet customer demands quickly.
   * **Availability**: Ensuring the product is always accessible ensures users can rely on it.
   * **Defect Detection**: Identifying issues early minimizes costly fixes after production.
   * **Resource Efficiency**: Using hardware efficiently can reduce costs, boosting profitability.
4. **Top Actionable DevOps Metrics**:
   * **Mean Lead Time**: The time it takes for an idea to reach production. This metric ensures that development aligns with customer needs and that new features reach users in a timely way.
   * **Release Frequency**: The speed of new releases ensures the company remains competitive.
   * **Change Failure Rate**: Tracking how often updates fail helps maintain system stability as speed increases.
   * **Mean Time to Recovery (MTTR)**: Focusing on quick recovery, rather than failure prevention, ensures resilience and minimizes service interruptions.
   * **Real-Life Example**: A tech company tracking MTTR can quickly deploy fixes to reduce downtime, ensuring customers experience fewer disruptions. By also measuring release frequency and change failure rate, the company can make improvements without compromising quality.

In conclusion, actionable metrics support data-driven decisions, helping organizations improve processes and meet customer expectations. By focusing on meaningful measurements like lead time, defect reduction, and resource efficiency, DevOps teams can achieve resilience, speed, and reliability in software delivery.

—----------------------------------------------------------------------------------------------------------------------------

## **Video 3: How to Measure Your Culture**

—--------------------------------------------------------------------------------------------

This lecture focuses on measuring team culture using statements developed by Dr. Nicole Forsgren at DevOps Research and Assessment (DORA). These statements help teams evaluate their dynamics and identify areas for improvement on a scale from 1 (strongly disagree) to 7 (strongly agree). Here are the key points and their practical applications:

1. **Seeking Information**: Team members should actively seek out information and be curious about why things happen. When teams prioritize understanding, they can uncover potential issues early on and develop solutions proactively.
   * **Real-Life Example**: A project team consistently reviews past successes and failures to identify patterns and improve future projects.
2. **Failures as Learning Opportunities**: Teams should view failures as opportunities for learning rather than punishment. This mindset encourages experimentation, where even if something goes wrong, team members feel safe to discuss it openly.
   * **Real-Life Example**: If a software deployment fails, instead of blaming a team member, the team analyzes what went wrong, adjusts workflows, and ensures everyone is better prepared for future deployments.
3. **Shared Responsibility**: Effective teams operate with a "we’re all in this together" approach, where everyone steps in during crises, creating a sense of collective responsibility.
   * **Real-Life Example**: When a customer-facing issue arises, all team members, regardless of role, come together to resolve it quickly and efficiently.
4. **Encouraged Cross-Functional Collaboration**: Collaboration across departments should be both encouraged and rewarded, allowing team members to develop broader perspectives and contribute beyond their usual responsibilities.
   * **Real-Life Example**: A marketing team works with engineers to understand a product’s features better, ensuring campaigns accurately highlight the product's strengths.
5. **Inquiry into Failures**: Instead of focusing on "who" caused an error, effective teams ask “why” the error occurred. This approach supports continuous improvement and system-based thinking.
   * **Real-Life Example**: After a delay in project delivery, the team examines workflow bottlenecks and improves processes instead of pointing fingers.
6. **Welcoming New Ideas**: An open environment where new ideas are valued and genuinely considered contributes to innovation and team morale. Employees feel empowered when they know their contributions matter.
   * **Real-Life Example**: A manager holds monthly brainstorming sessions where all team members, regardless of experience level, can share ideas for improving workflows or product features.

In summary, measuring team culture through Dr. Forsgren’s statements enables teams to assess their collaborative environment, improve communication, and foster innovation. Each statement provides actionable insights, allowing teams to create a supportive and productive workplace culture that drives continuous growth and team unity.

—----------------------------------------------------------------------------------------------------------------------------

## Video 4: Comparison of DevOps to Site Reliability Engineering

—----------------------------------------------------------------------------------------------------------------------------

This lecture explains the differences and commonalities between Site Reliability Engineering (SRE) and DevOps, and how they can work together to improve software deployment stability and speed. Here’s a summary of the main points, along with practical examples:

### **Key Differences Between SRE and DevOps**

1. **Nature of Responsibilities and Team Structure**:
   * **SRE**: Maintains separate development and operations teams to keep clear boundaries. SRE hires software engineers who focus on automating tasks and reducing repetitive "toil."
   * **DevOps**: Breaks down silos to integrate development and operations into a single team with shared goals, focusing on a collaborative “you build it, you run it” approach.
   * **Example**: In an SRE structure, developers focus on creating new features, while SRE engineers focus on automation and stability. In contrast, a DevOps team would have all team members responsible for both development and operational stability.
2. **Stability Management**:
   * **SRE**: Uses “error budgets” to limit allowed downtimes. Developers can deploy freely until they exceed the error budget, then are restricted from further deployments until stability is restored.
   * **DevOps**: Relies on Continuous Delivery pipelines and a shared responsibility model, ensuring that the entire team is accountable for code in production.
   * **Example**: An e-commerce platform could use an SRE error budget to keep uptime above 99.9%, stopping deployments if downtime exceeds the budget. In DevOps, all team members might collaborate on automated tests to catch issues early and avoid downtime.
3. **Focus on Automation**:
   * **SRE**: Engineers automate tasks to reduce toil (manual, repetitive tasks), aiming to automate themselves “out of a job” by freeing up time for more complex issues.
   * **DevOps**: Also uses automation, particularly through Continuous Integration/Continuous Deployment (CI/CD) pipelines, to speed up and stabilize deployments.
   * **Example**: If an SRE team finds themselves manually configuring servers repeatedly, they might write scripts or use Infrastructure as Code to automate server setup. In DevOps, CI/CD pipelines automate testing, integration, and deployment processes.

### **Commonalities Between SRE and DevOps**

1. **Blameless Culture**: Both SRE and DevOps promote a blameless culture, emphasizing that errors are due to system issues, not individual failings. This approach encourages open discussions about mistakes to drive continuous improvement.
   * **Example**: After a production issue, an SRE or DevOps team conducts a "post-mortem" meeting to find the root cause and discuss improvements without placing blame.
2. **Shared Goal of Stability and Speed**: Both frameworks aim to deploy software quickly without compromising stability, although their approaches differ.
   * **Example**: A company with both SRE and DevOps teams may have SRE maintain the infrastructure's reliability, while DevOps focuses on using it to deliver applications efficiently.

### **How SRE and DevOps Can Complement Each Other**

1. **SRE as Infrastructure Providers**: SRE teams can create and maintain stable infrastructure, which the DevOps team then utilizes to deploy and manage applications.
   * **Example**: In a cloud environment, the SRE team might manage the cloud platform (e.g., configuring load balancers, automating scaling), while DevOps uses the platform to push application updates.
2. **Practical Integration**: SRE can manage Platform as a Service (PaaS) resources, enabling DevOps teams to focus more on application performance and deployment.
   * **Example**: SRE engineers could set up auto-scaling policies in a cloud environment, allowing DevOps to release applications without worrying about handling traffic surges.

### **Summary**

This approach highlights that while **SRE and DevOps have different structures and methods**, they share a commitment to rapid, reliable software deployment. By combining SRE's focus on automation and infrastructure with DevOps' integrated, collaborative development, companies can achieve faster and more resilient software delivery.

—----------------------------------------------------------------------------------------------------------------------------

# **Summary and Highlights**

Congratulations! You have completed this lesson. At this point in the course, you know:

* Measure and reward what you want to improve.
* People seek information on what is rewarded and then seek to do that.
* Measuring social metrics leads to improved teamwork and measuring DevOps metrics allows you to see the progression toward your goals.
* If you want people to be social, then measure them being social.
* DevOps changes the objective of problem resolution from failure prevention to failure recovery.
* Vanity metrics may be appealing at first but offer limited actionable insights.
* Actionable metrics provide meaningful ways to measure your processes and take action toward goals.
* DevOps actionable metrics include mean lead time, release frequency, change failure rate, and mean time to recovery.
* You can rate statements developed by Dr. Nicole Forsgren to measure your team’s culture, including statements about information, failures, collaboration, and new ideas.
* Mean lead time is the measure of how long it takes for an idea to get to production.
* Change failure rate is the rate of failure from pushing new releases out.
* Mean time to recovery is how long it takes to recover from a failure.
* Failures are learning opportunities that should not be punished.
* Dr. Nicole Forsgren developed cultural statements for measuring team culture.