# DEVOPS FEEDBACK LOOPS: AN INTRODUCTION

A basic concept of system thinking as well as a vital part of your organization's success, understanding and properly implementing DevOps feedback loops in your business is a "make-it or break-it" kind of thing.

As a DevOps professional, you are focusing on creating a streamlined relationship between development and IT operations. However, to cultivate a successful relationship between and collaboration of these two business units -- focusing on both speed and quality -- you must first know how feedback loops influence your process improvements.

## What is a Feedback Loop?

A common DevOps term, feedback loops are a largely misunderstood concept Let's begin by addressing what the words "feedback" and "loop" mean, and then how they work together to complete useful work.

- Feedback is defined in the Oxford Dictionary as, "The modification or control of a process or system by its results or effects".
- Loop is defined as, "A structure, series, or process, the end of which is connected to the beginning."

When we put these two terms together, we understand the meaning of a feedback loop to be a system where the outputs are put back into the inputs to improve the process or product while

increasing or decreasing the effects produced later. Feedback loops are basically an internal review of how teams, systems, and users function.

Yes, this still seems kind of confusing. So, let's dive deeper into how feedback loops work in business and technology. As many experts agree,

*"Feedback loops enforce priorities and project goals so that the freedom and fast pace in development doesn't lead it astray."*

The objective of this system is to create a circuit between the two above-mentioned DevOps business units. Essentially starting a flow where when a change happens in one unit, it causes a change in the other unit, eventually leading to a change in the first unit. This allows a company the agility to perform necessary corrections continually in the right direction. Using a feedback loop to collect IT data and create a constant flow of information simply means valuable growth at an enormous scale.

And, one of the best reasons to use DevOps feedback loops is to bridge the gap between software function and customers' expectations. This is organized, optimized change!

# A Simple DevOps Feedback Loop

A developer (input) writes some code for a new program. He then sends the newly written code to the designer to build it. After that, an OS runs the built-out code (output). While running it, the developer observes the code. After observation, he determines what to do next (input). He sends the next steps back to the designer and again observes it on the OS (output).

Can you see the loop?

The purpose of the feedback loop here is to streamline the way the code is written, delivered, built, processed, and the time it takes to input modifications. If one of the steps in information delivery is off, the entire loop will be off.

# Two types of feedback loops

## Reinforcing (positive)

Also called an accelerating loop or change, this type of loop happens when a group sees a positive increase resulting in a second group seeing a positive increase, which then causes a positive increase in the first group. Please note that it can go the other way with a negative influence as well. Take a moment to note how the above-mentioned example is a positive feedback loop.

## Balancing (negative)

This type of loop is the opposite of accelerating. Balancing happens when a first group sees a positive increase resulting in a second group seeing a decreased value, which then negatively decreases the value of the first group. The system will eventually stabilize at a place where no more change can occur.

As each type of feedback loop embraces varying factors--less leading to less or more leading to more--the basic function remains the same. As the loop continues, each unit individually changes while affecting others.

# Common mistakes to avoid

Focusing on feedback, ignoring the loop

Yes, feedback is always great. But, the point of the loop is to take action. Maybe you have gotten the first steps all in place--integration, building, testing, and deployment. Then things start failing and commitments increase. Your teams are lost in what to do because they fail to find the root cause and communicate. Perhaps they saw the feedback but didn't continue the loop to solve the problem.

## Feedback loops are positive, ignoring the negative

Going back to the types of loops, you must, without any compromise, define what type you are dealing with before implementing any changes. If you are focusing on one outcome but the loop creates a different outcome, balances when not expected, or increases, you will end up with slower, less effective systems and operations. This process will lead you to issues that are never fixed.

## Failure on training teams to provide feedback

Confirming that your entire team can give reliable, usable feedback as well as implementing any feedback received is a key part of having successful feedback loops. To achieve this, you can use already existing feedback loops and discuss what is working within the system. Also, be sure to make all notifications actionable, ensuring follow-through and effectiveness.

## Fostering the idea of single loops, not a system

While there is value in focusing on just one loop, if you fail to see the entire system of loops, your single-loop will fail. For example, your system is unstable due to network failure and full storage space. To fix the problem, you apply a balancing loop, issuing notifications to employees when an issue occurs. This stabilizes the system. But, you can't stop there. Once stabilized, you need to implement an accelerating loop where you can plan for future system modifications allowing you to expand your network and storage. Without the system of loops working together, you will hit a change stalemate.

## Feedback loops as notification systems

Speaking of notifications, it is important to always remember that alerts and notifications are just pieces of information. These are not loops in themselves. You need to focus on humans relating the information from the output into the input rather than relying solely on automated system alerts. Continually check your automation and improve your loops.

## Forgetting to define specific problems

An assessment of your current situation will save you a big headache in the future. As each problem is defined before you start your loops, you can avoid duplicates, unclosed loops, and system problems. On top of that, when your problems are defined, you can easily define your type of loop, building from the ground up.

## Facilitating reviews without closing the loop

From small to large operations, closing the loop and ensuring information feedback actively arrives to the other end is essential. With machine to person, person to person, machine to machine, and person to machine communications to consider, you need to make sure every outlet is communicating correctly with the other outlets. If one is off, they are all going to be off.

# A better understanding of feedback loops

Analysis and optimization of system and software delivery depend on your ability to properly set up feedback loops. Without them, your business and its structure will fail due to a lack of solved problems and information being rapidly shared.

# Related reading

- BMC DevOps Blog
- DevOps Guide, a series of articles
- Shift Left Testing: What, Why & How To Shift Left
- Testing Automation Explained: Why & How To Automate Testing
- Testing Frameworks: Unit Tests, Functional Tests, TDD & BDD Explained
- Deploying vs Releasing Software: What's The Difference?