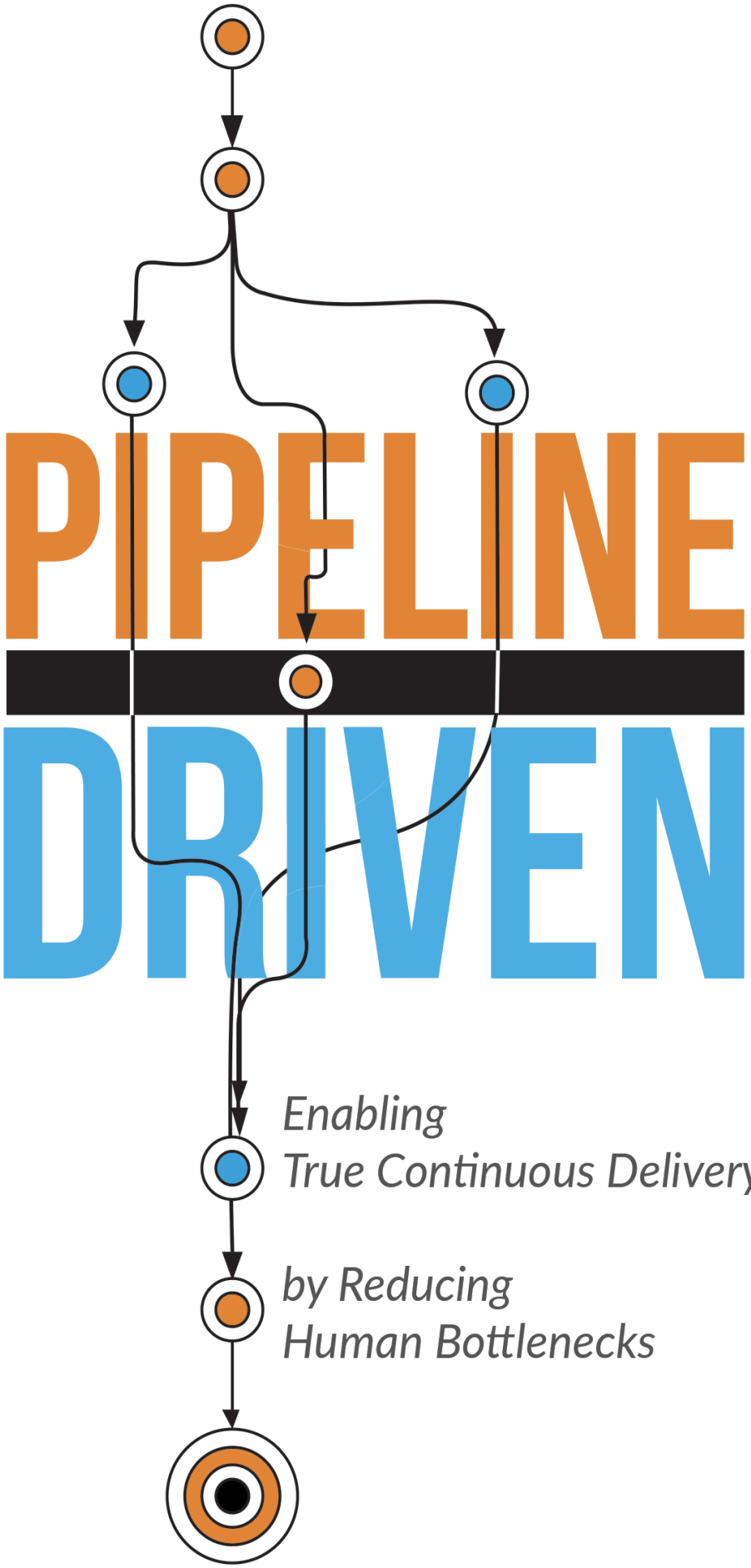
ROY OSHEROVE



PIPELINE DRIVEN

Enabling
True Continuous Delivery

by Reducing
Human Bottlenecks

Pipeline Driven

Enabling True Continuous Delivery by Reducing Human Bottlenecks

Roy Osherove

This book is for sale at http://leanpub.com/pipelinedriven

This version was published on 2019-06-10

ISBN 978-82-999332-1-6

Published by Team Agile Publishing. Yet another Roy Osherove “Project”. More info at osherove.com

Tweet This Book!

Please help Roy Osherove by spreading the word about this book on Twitter!

The suggested tweet for this book is:

AUTOMATE ALL THE THINGS. Just got #beautifulbuilds book. http://beautifulbuilds.com

The suggested hashtag for this book is #beautifulbuilds.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#beautifulbuilds

Contents

About the Author

Roy Osherove is the author of “The Art Of Unit Testing” and “Elastic Leadership“ and has been in leadership roles for most of his professional life, acting as team lead, CTO and architect in many places.

Roy currently works as Senior Staff Software Engineer at Siemens Healthcare Diagnostics in New York.

- Roy’s online courses (including on the topic of this book) can be found at http://courses.osherove.com
- You can read his blog at 5whys.com and osherove.com
- You can reach roy at roy@osherove.com

1. Pipeline Troubles

1.1 The Holy Grail of Continuous Delivery

What does it take to work at the speed and agility of a company like Netflix, Amazon and Google, deploying changes to production code many times a day?

Why do so many medium and large companies fail to do it? What does it take to enable *true* continuous delivery?

Why is it so common to hear the phrase "That'll never work at my job"? What's the glue, the magic ingredient, the measurable difference that turns an organizations with a few Jenkins jobs lying around to an organization that goes *fast*?

The difference is, of course, as with all software problems, *people*. More specifically, the way people work with the pipeline. The expectations we set for the pipeline determine how much it enables true continuous delivery.

A pipeline-driven culture requires both *pipelines* and *culture* so let's start with the *Pipeline*.

1.2 Pipelines

A *pipeline* is a set of one or more *automated* tasks or "*jobs*" that run in a certain order to produce a result. This result is usually made up of two things:

- A *judgement result* : The pipeline passed or failed or an inconclusive result (Always use Failed/passed, never inconclusive. I'll cover why later in this book).
- Any related *artifacts* : deployed application, binaries, log files or anything else we care about.

Tasks

At the heart of it, a task is a command line (shell) script or program that runs automatically on some machine (locally or in the cloud), executes some tasks (sometimes many tasks that

are related to each other, like compiling, running tests etc), and returns a special command line "exit code".

It is common to have an exit code of *zero* at the command line signifies *success*. Any *non-zero* exit code signifies *non-success*. The program or script decides what the exit code will be.

CI/CD Servers

Tools like *Jenkins* (Colloquially called "Continuous Integration Servers" or "CI/CD servers" for "Continuous Delivery") act as an abstraction layer on top of these tasks. CI Servers allow us to easily run these tasks (based special code commit triggers or schedules), and listen to the task's exit codes, as well as its outputs (logs).

They would then parse out the logs, and give us a nice UI on showing the results, the error logs and any artifacts.

CI/CD Servers allow us to manage our tasks in a single location and monitor their running progress, run tasks in parallel and run them on different machines (sometimes called 'build agents').

1.3 Human Bottlenecks

Let's turn our attention to the other side of the pipeline equation: the culture, and people who use the pipeline.

In many companies I consulted for, there isn't just one single pipeline. There are several for a given product. They are usually separated by the special invisible "walls" — usully related to expertise in the organization that that specific pipeline is supporting.

There might be a "dev" pipeline, a "test/QA" pipeline, a Security pipeline, a Deployment pipeline , etc..

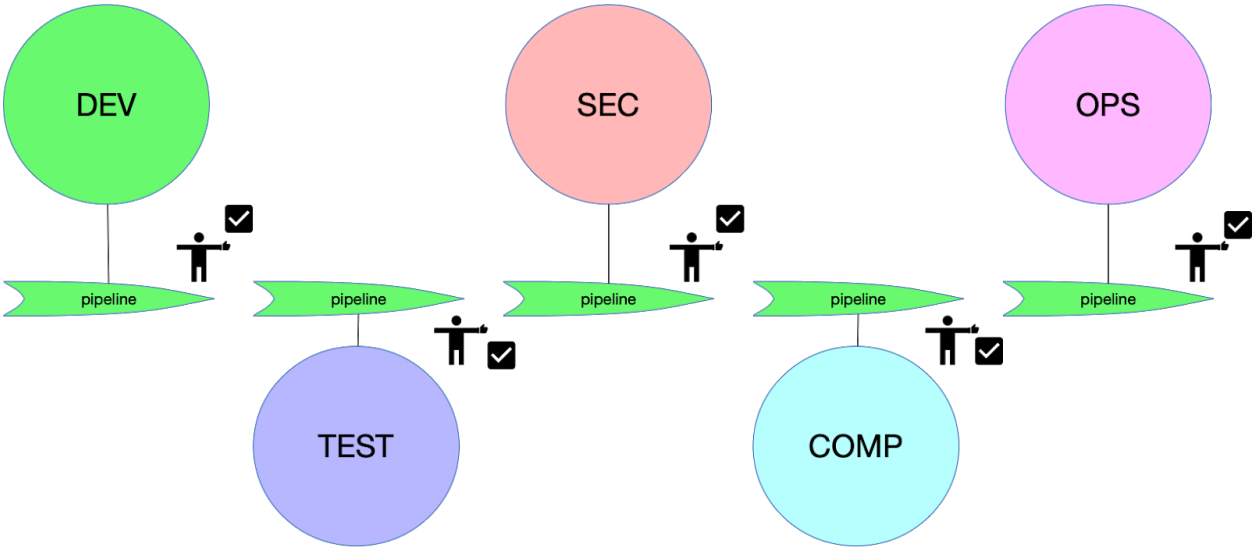


Figure 1.1: Human Bottlenecks

In the figure above you can see the long process code usually has to flow in medium to large organizations, where there are knowledge silos and special permissions needed for each expertise.

Moving on Slowly

The code we write has to slowly move from pipeline to pipeline, slowly getting to its final Goal - *Production*.

To get there, an active pipeline (usually a development related one) "asks" or "notifies" its owners the results of its latest run. If everything is OK (tests passed, compilation passed for example) the a human is notified ,and the pipeline stops.

Different Folks, Different Strokes

Not all pipelines stop at the same place - each product is different and requires different tasks in the pipeline. And each organization is different and has a different structure to the knowledge silos inside it.

In some places the initial "dev" pipeline might stop shy of deploying the code to some "dev" environment. In other organizations, the dev pipeline might go all the way to deploying code to staging. In others it might not even get to the master branch, as there might be a pipeline-per-branch scenario. (I'll touch on these anti patterns and others in the second part of this book that deals with common bottleneck patterns and how to solve them.

Hot Potato Relay Race

The point is, that *people* represent the biggest time gaps in the way of the code getting as far into production as it can. In many organizations, humans look at the result of one pipeline, then notify other humans that "it might be OK" to run the next pipeline so that the code "progresses" to a more mature state, and eventually production.

It is a relay race, people pass the "stick" (i.e "code" or application binaries) from one to the other, but unfortunately it contains many factors that cause it to be very slow.

In some cases it cab be more of a hot-potato game: "I'm not sure if it's OK but as long as *your* pipeline passes it'snot *my* fault!"

in other cases (and of course in the case of the *last* pipeline and the human attached to it in the hierarchy) people are afraid of making a decision and approving or rejecting the result of a pipeline run. And that's where we can start to see the crux of the problem.

1.4 Fear, Uncertainty, Doubt

Human-enabled pipelines might carry a large mental cost.

A company I was consulting for had the following process before doing their monthly product release:

- They would hold an all-hands leadership meeting
- Invite the Test-Lead in and question them about wether they believe the version is "ready for prime time".
- The poor test lead would have to *sign* off (at some point it escalated to *literally with their name on paper*) that the release was OK on all aspects: Quality , regression issues, performance, security, compliance.

Let's consider several factors on the human mind in this situation:

- How much stress would you be in if the *top brass* would meet with you monthly and you had to swear on your job that this *version* won't be like the last one, or that this version is just as good as the last one?
- What type of verification would you conduct if you, and you alone, were held accountable for any issues that arose?

- Would you trust a pipeline that told you "everything is OK!" Blindly so that you would come into the meeting without checking things for yourself (with some other folks helping you), regardless of what the pipeline says or does?
- If instead of at the end of the month, the high brass would come in in the *middle* of the month and ask for a high priority hot-fix in production that needs to go out immediately. Would you have signed off on that ? Would you have a special "we don't know what will happen so let's pray we won't find out" ?
- When you need to personally sign off on something in front of your boss, would you do it quickly, or would you take your sweet time, making sure you're not putting yourself in any danger?

1.5 What's the Worst that could Happen?

Here are some things that might result from those types of pressure:

- Burnout, health risks and job switching
- Very long and manual verifications *after* a pipeline has run, or ignoring the results of a pipeline completely.
- Approves are less likely to approve changes fast enough, or passing any blame for fast approval up the chain.
- Change become slower and slower
- More people might be added to the "sign off" list if more issues are found, cause more slow down
- More people working on manual verification in a waterfall fashion (when it's tie to sign off on a release)

Now, multiply these factors by the amount of people who have to "pass the buck" along to the next pipeline, and you can see that the further we get away fro the development pipeline, the more pressure and stress there is on a single person at each turn, to sign off on pushing things to the next pipeline.

Is it any wonder that deployments to production (especially in mid-large companies) are in a category of their own when it comes to bottlenecks? — They usually happen in specific times, you have to plan them days, weeks (sometimes months) in advance, they involve a chain of people , each signing off, they can be very manual, and mostly they are brought with fear, uncertainty and doubt.

1.6 Where the buck stops

Imagine you're the last person running the last pipeline before a release. Maybe you're in a large organiation and you're part of the "finish line" team - and you need to deploy to production and check that everything went well.

Maybe you're a security admin and you're verifying the environment is properly secured.

Imagine all of the possible colossal failures hiding in all the steps that came before you in the process, accumulated up until now. They *could* suddenly come into fruition here, at this stage, exploding in the face of the last person that can do anything about them, or even be aware of them. You didn't find any. But they could still be there. You're signing off on this. After this, there's nobody else.

Would you have signed off quickly and quietly? Or would you take the often taken road of "CYA" (Cover your ass)? And if anything went wrong - would you assume responsibility? Would you feel accountable? Would you feel helpless, frustrated? How would your actions change the *next release* if this one didn't go well?

1.7 Summary

I believe that tasking humans with the sign off on something that is complex enough to have many hidden issues can lead not only to an unhealthy amount of stress and eventually burnout, it also leads to very , very slow sign-off times, which means slow handoff times between pipelines. Which eventually means a simple thing:

No *continuous delivery* for you if you do this. Intermittent at best.

Next we'll look at the benefits of switching from just having pipelines, to being *pipeline-driven.*