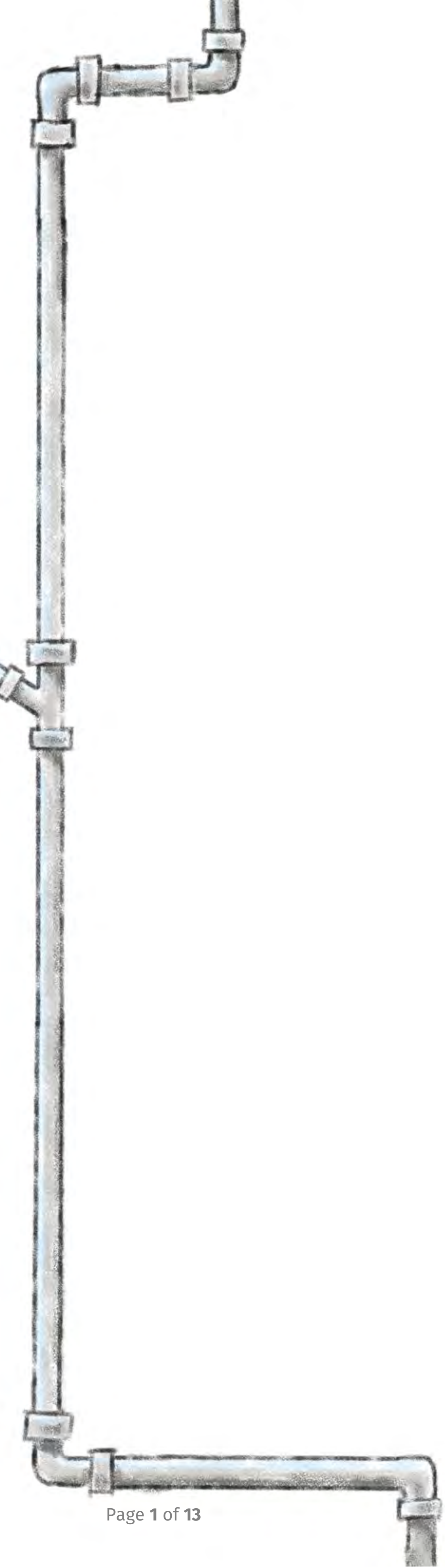
Continuous Integration

Continuous Delivery

Continuous Headache?

Learn about practical CI/CD implementation strategies with effective real-world benefits.

inedo



CI/CD/CH?

Continuous Integration / Continuous Delivery / Continuous Headache?

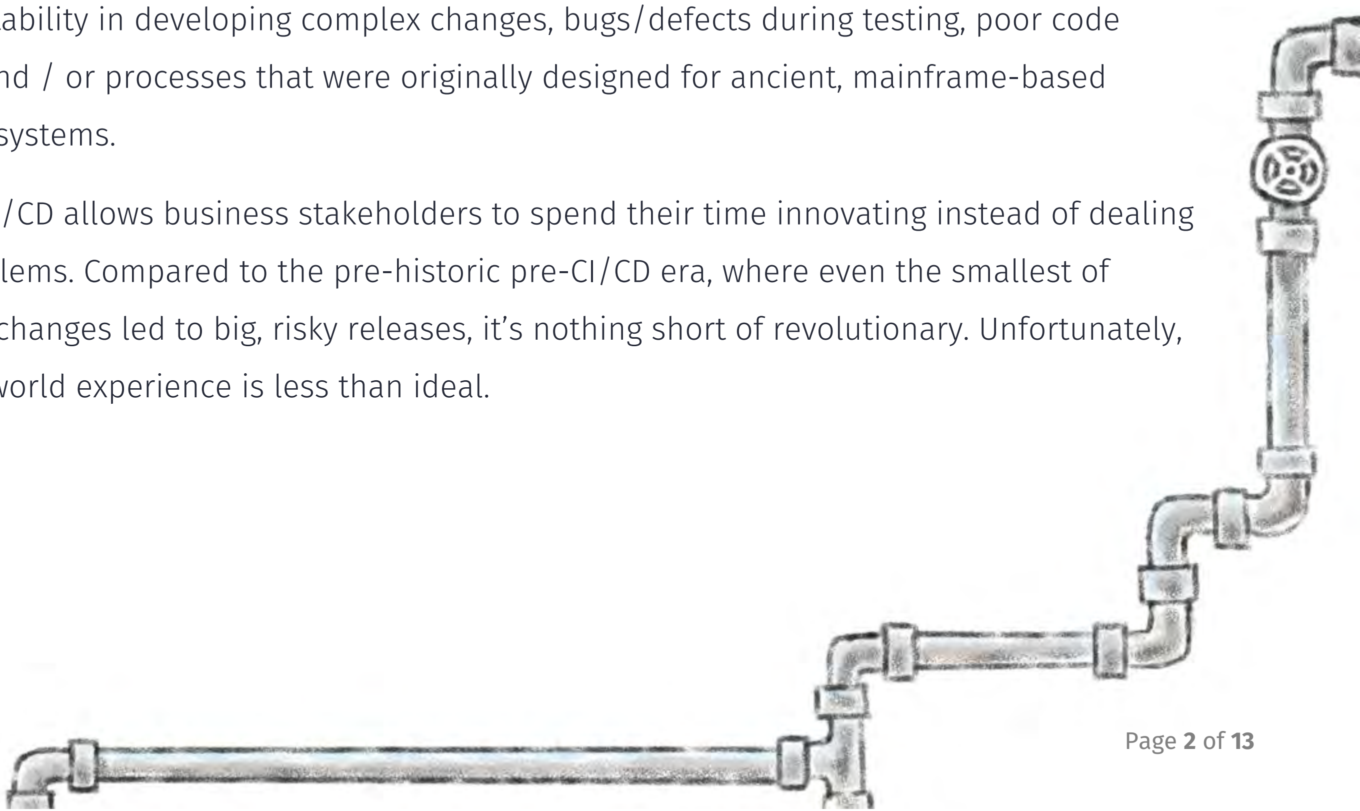
The Promise

CI/CD was a dream come true. Automated builds! Automated testing! A brief verification and then, *Click!* Instant deployment of beautiful, new, and validated code that sailed straight to production. It can and should work like this because when it's done correctly, it delivers:

- Better quality of software because fewer defects make it to production.
- Faster delivery of business ideas to market by enabling faster release cycles that allow software to be changed in days or weeks rather than months or quarters.
- Cheaper implementation across the entire lifecycle, including less time spent coding, deploying, and testing software changes.

The true value of CI/CD is that organizations can adapt to market changes and innovate just as fast (if not faster) than their competitors. This is accomplished by changing the software that powers the business, and then releasing those applications as fast as developers can write new code and testers can verify it. Prior to CI/CD, software releases were often held up by unpredictability in developing complex changes, bugs/defects during testing, poor code quality, and / or processes that were originally designed for ancient, mainframe-based software systems.

Ideally, CI/CD allows business stakeholders to spend their time innovating instead of dealing with problems. Compared to the pre-historic pre-CI/CD era, where even the smallest of software changes led to big, risky releases, it's nothing short of revolutionary. Unfortunately, the real-world experience is less than ideal.



The Headache

The dream of CI/CD has not materialized for most organizations. In fact, things are actually worse than before. There is plenty of automation, but no one seems to know how it all ties together. Critical releases are moving slower and have more defects than before because there are too many tools with overlapping duties and each tool has its own set of idiosyncrasies. Problems lead to finger pointing. Senior decision makers wonder why they approved purchasing these tools. Managers worry about their team morale and getting releases finished bug-free. Engineers can't access the detailed execution logs they need and a slow approval process stymies up-and-coming developers.

Like all unholy messes, a CI/CD headache isn't built overnight. It emerges after bolting tool on top of tool in various parts of the software delivery process over time. Each of these tools may make sense but when you bring everything together – including the hundreds of interdependent applications that rely on these tools – you end up with a Frankentool that no single person can fully understand, let alone properly use or improve. It's become a monster.

You may have already created a Frankentool and not even know it yet. Answer the following questions and then keep reading.

1. Do only a handful of engineers actually understand how your software delivery process works?
2. Have you experienced an information bottleneck because only 1 or 2 people had the information integral to a build or release?
3. Do less than 5 people interface with the CI/CD process?
4. Does it take hours to track down basic information – like which changes are in which build artifact, which environment those builds were deployed to, or what exactly was tested?
5. Have business stakeholders built their own "metasystems" using a series of issue tracking tools, spreadsheets, meetings, sticky notes, and emails just to figure out what application is in what state?

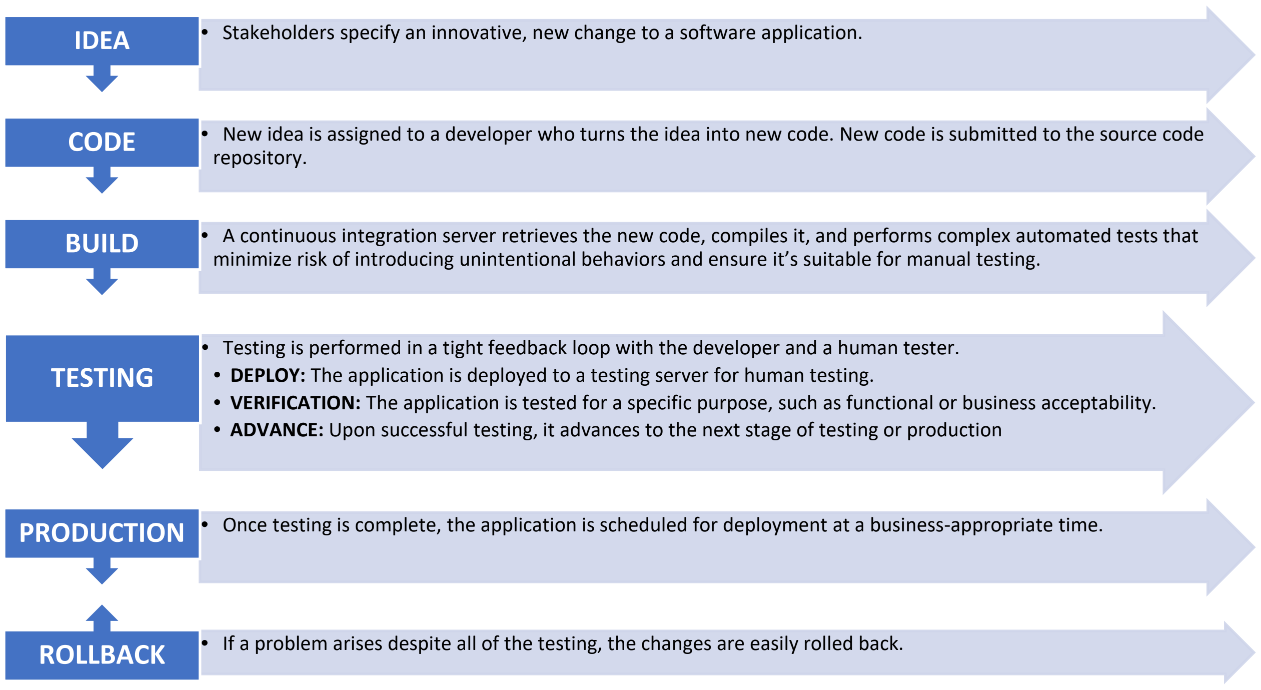
6. Can your organization deliver software only as fast as the handful of people who understand the complex systems can deliver information and decisions?

If you answered "yes" to any of the previous questions, you likely have a CI/CD headache on your hands. Don't panic. Fortunately, there's a way out of this mess. Inedo's BuildMaster was specifically designed to help you navigate the complex and technical landscape to realize the dream of CI/CD without the headache.

A Little About Software Delivery with CI/CD Pipelines

Real world CI/CD relies on a balance between automated building, automated testing, and manual verification to quickly deliver ideas to production… and roll those ideas back quickly if they don't work out.

Figure 1: The CI/CD Pipeline



IDEA
- Stakeholders specify an innovative, new change to a software application.

CODE
- New idea is assigned to a developer who turns the idea into new code. New code is submitted to the source code repository.

BUILD
- A continuous integration server retrieves the new code, compiles it, and performs complex automated tests that minimize risk of introducing unintentional behaviors and ensure it's suitable for manual testing.

TESTING
- Testing is performed in a tight feedback loop with the developer and a human tester.
  - **DEPLOY:** The application is deployed to a testing server for human testing.
  - **VERIFICATION:** The application is tested for a specific purpose, such as functional or business acceptability.
  - **ADVANCE:** Upon successful testing, it advances to the next stage of testing or production

PRODUCTION
- Once testing is complete, the application is scheduled for deployment at a business-appropriate time.

ROLLBACK
- If a problem arises despite all of the testing, the changes are easily rolled back.

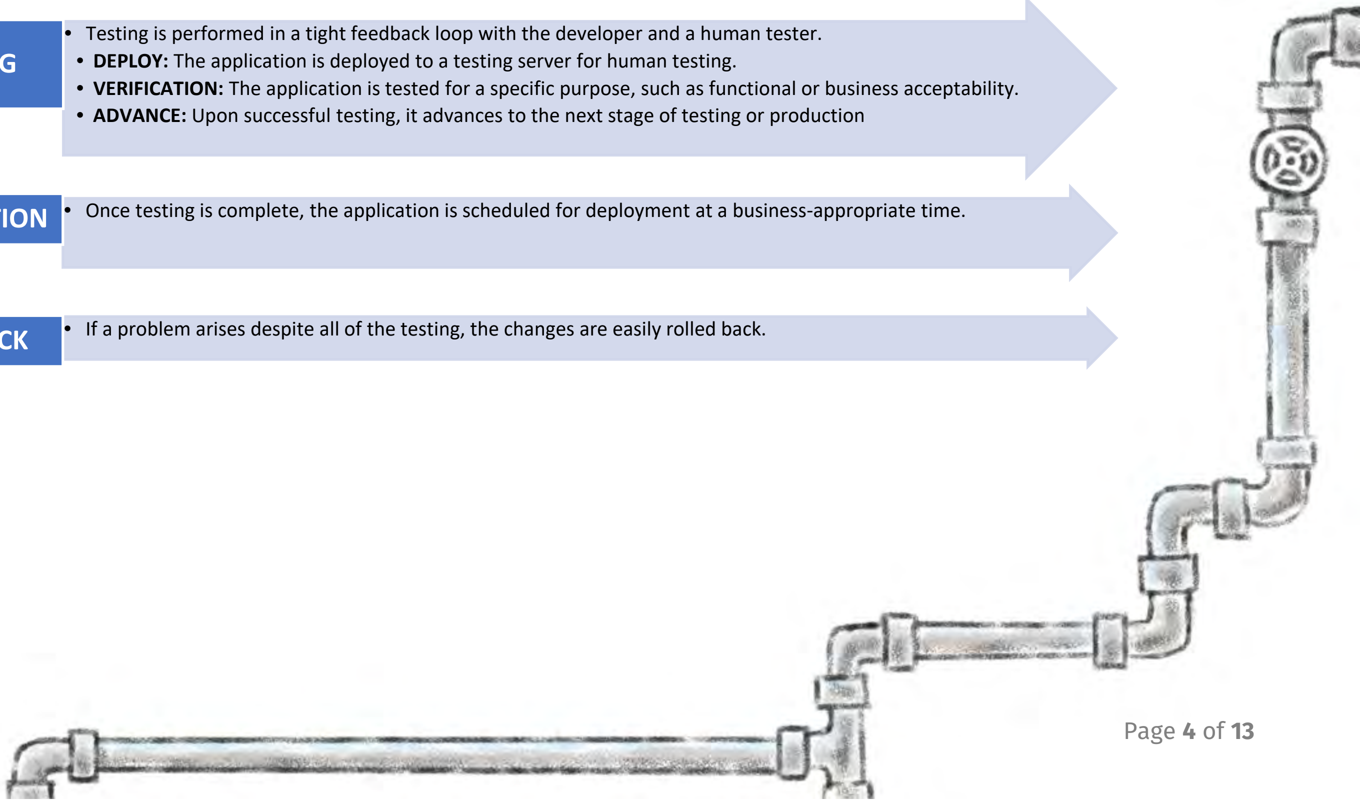
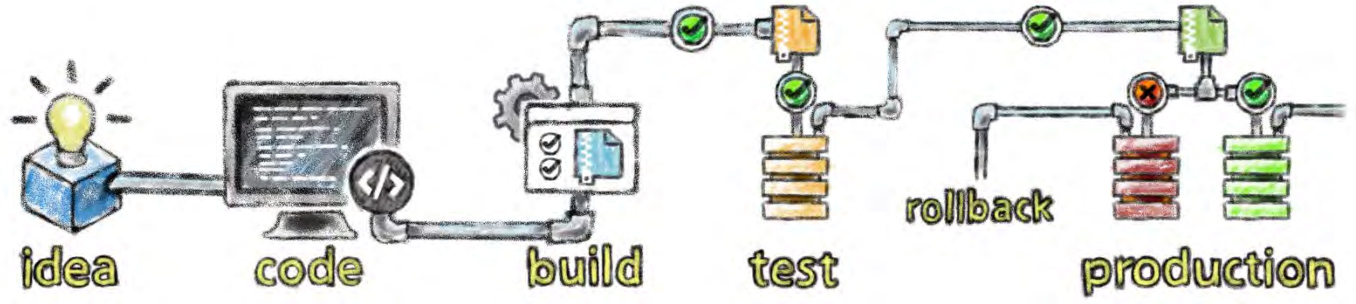


Figure 2: CI/CD Pipeline Idealized



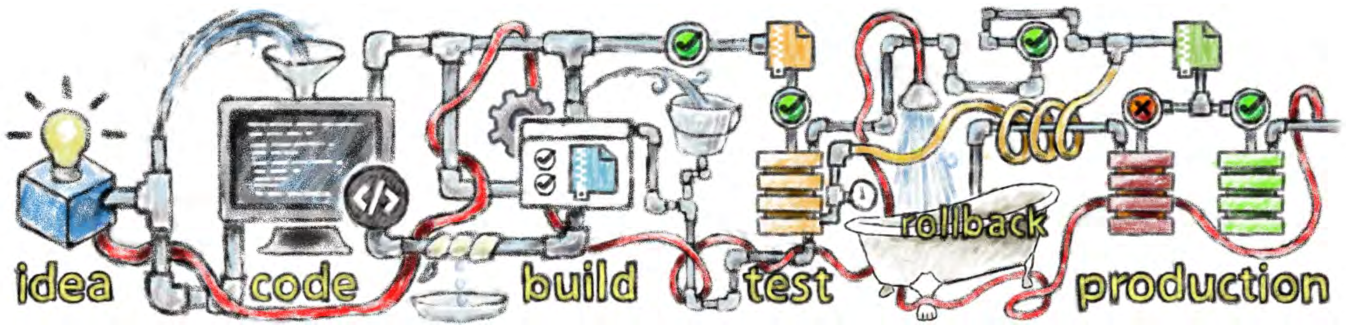
Nearly all organizations have a less-than-ideal software delivery process, which leads to less-than-ideal releases. A poor process can only result in delays, bugs, and/or a product that does not match the vision. In today's marketplace, the pressure to deliver releases faster and faster puts a strain on the software delivery process. Therefore, the process must be seamless. At Inedo, we have pinpointed multiple points of potential process breakdowns.

Breakdown 1: Brittle Toolchain Links

Software delivery processes and tools are linked together through a combination of people-driven conventions and software-driven integrations. For example, when a developer enters an Issue ID as a source control commit note, a script will post a comment to the issue tracking system with the Commit ID.

As the pace of software delivery increases in the real world, not only do these brittle links become more and more important, but so does establishing even more links between the disparate tools and systems. But, each new link added is just as brittle as the existing links and is often implemented with some hastily written script that's tied to an arcane feature of one of the tools, or an entirely new tool to do something different.

Figure 3: CI/CD Pipeline Failure



At the same time, the pressure to speed up software delivery increases, and this creates new opportunities for human error and script failure. These failures can be costly: if a developer neglected to properly link an Issue ID, or the script that processed it failed, imagine the challenge of trying to identify which one of dozens of commits were related to which one of dozens of issues. These brittle links compound, and only get worse as more tools and complexity is added.

Breakdown 2: Browning Greenfields

A greenfield is a brand-new application that is built from scratch to solve an organization's problematic software delivery process. No patching. No maintaining old software. It's a fresh start! But, changing the whole system is costly and inevitably, the new software becomes old software after two to three years.

A brownfield is an old greenfield. It's the legacy software systems and processes that are not the ideal way of doing things and are seemingly more complicated to maintain.

Many organizations have realized the challenge with overhauling their software delivery processes and have instead invested in a "greenfield" process for new, "greenfield" applications. The hope is that, over time, the new applications will gradually replace the existing applications and underlying software delivery processes.

Of course, this neglects the actual, larger problem (i.e. effectively delivering the existing/legacy applications that are actually creating business value today), but it also establishes a new set of tools and processes that will unavoidably become a brownfield.

Then, the “new” processes become too risky to overhaul, and someone else will decide to solve *that* problem by creating yet another “greenfield” to use. It’s a vicious, unwinnable cycle.

Breakdown 3: Crisscrossing Dependencies

Organizations are comprised of many different business units, each with their own sets of applications that follow their own delivery processes.

For example, the mortgage division at a bank would use totally different systems than the same bank’s wealth management division. Historically, this hasn’t been a problem; a bank customer would realize that these are, in fact, two different business divisions and thus, would have two different relationships with the bank.

However, as organizations evolve, consumers seek simpler and more integrated relationships, which mean different divisions must become aligned. For example, a unified banking portal that allows customers to see their mortgage and investment accounts all in one place.

This sort of integration is technically challenging in and of itself, but it also must be coordinated across a mass of greenfield and brownfield processes, each with their own set of brittle toolchain links.

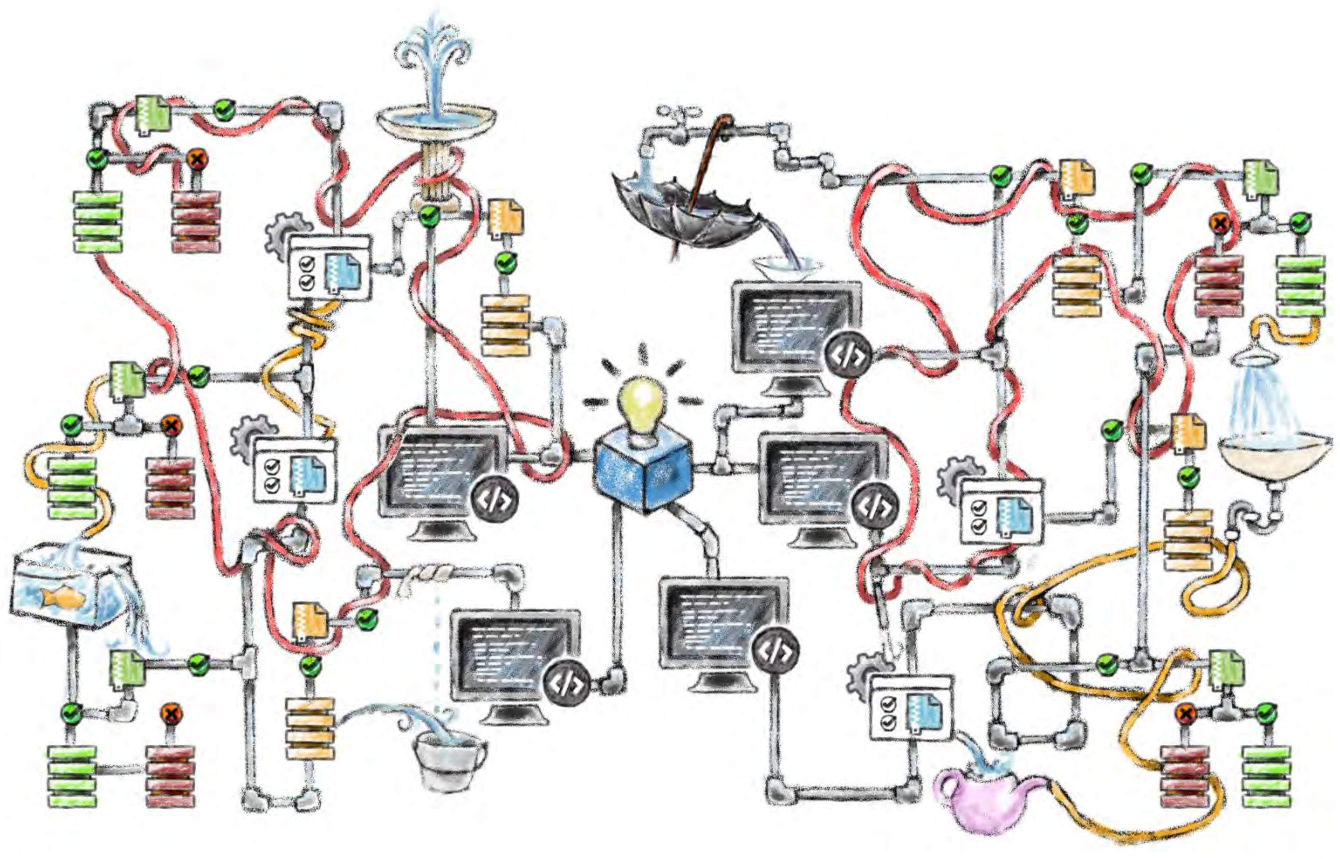
Breakdown 4: Bottlenecks

The few engineers that oversee these tools and processes quickly become information bottlenecks, and a sort of human interface to the CI/CD process. Tracking down basic information – like what changes happened in what build artifact and in which environment the build deployed to, and what exactly was tested – can take hours because only a few very busy people can track down the answer. By the time those answers are provided, it’s already outdated.

As a result, business stakeholders build their own "metasystems" using a mess of issue tracking tools, spreadsheets, meetings, sticky notes, and emails just to figure out what application is in what state, and when they can release new changes. Then, the stakeholders quickly become decision bottlenecks as they emerge as the human interface to these equally incomprehensible metasystems.

Ultimately, the organization can deliver software only as fast as the handful of people who understand the complex systems can deliver information and decisions. As more complexity and automation gets added, fewer and fewer people can manage them. The process only serves failure.

Figure 4: The Metasystem Mess



The Solution: BuildMaster

Inedo's BuildMaster delivers CI/CD for the real world. It was designed from the ground up to manage and automate the software delivery process as a whole. The application strengthens the brittle links between the tools that you're already using and oversees software delivery for all of your applications – legacy or otherwise – in a way that everyone in your organization can understand. Here's how we do it.

1. BuildMaster Embraces Complexity

Organizations fighting the Greenfield / Brownfield cycle can relax with BuildMaster because the application does not make you choose between new and legacy systems. The application does not force a single way because in the real world, there will always be a mix of legacy and new systems.

BuildMaster allows you to use the old systems, develop new systems, and use the two together at the same time in a process that is seamless. The application lets you adapt to a changing environment and embrace reality.

Inedo has designed BuildMaster to be constantly reusable and adaptable to your changing business needs. Our aim is to address the complex nature of IT and make it work *for* you. It's time to let go of the vicious flip flopping between an old software delivery process that works but is less than ideal and a new, shiny process that is expensive and unproven. You no longer have to choose.

2. BuildMaster Works at Your Speed

Whether your organization is aiming for 5 deployments a day or 1 per year, BuildMaster's flexibility works at any speed. BuildMaster was designed, first and foremost, to manage software delivery processes and allow organizations to define and change those processes at the speed they require.

Because software powers so many processes, the software must be able to adapt at an ever-increasing rate. Currently, your organization may have a mismatch between the ability of separate teams to deliver according to the release cycle. Though one application team can deliver releases on a reliable two-week cycle, the organization can only change as fast as its slowest application. With BuildMaster, each team is integrated and working at the optimal speed.

3. BuildMaster Decentralizes CI/CD

Attempting to force a uniform CI/CD process throughout the organization will quickly result in noncompliance, and ultimately, an even worse mess of undocumented or even rough processes. Instead, we provide groups with tooling that lets them define and take responsibility for their own software delivery processes.

Groups are ultimately responsible for meeting their own goals and they must become proficient at changing their own software-driven processes. Just like there's no one-size-fits-all software application, there's no one-size-fits-all software delivery process. Different groups must have the flexibility to define and change their own software delivery processes. BuildMaster was designed to be a self-service platform and has fine-grained permissions that allow different groups to not only build and deploy their own applications but define their delivery processes to whichever environment is configured.

But, at the same time, oversight is important. While each group is ultimately responsible for their own software delivery process, it still must be aligned with IT governance, which means auditors will need centralized access.

4. BuildMaster Is Scalable

BuildMaster allows you to start simple and then scale to servers and the cloud. It was designed for heterogeneous environments and works well with a mix of development languages (.NET, Java, NodeJS) and platforms (Windows, Linux, cloud). It's also language and technology agnostic and supports delivering virtually every type of software application and component.

When engineers hear the world “scalable”, they tend to think of things like distributed systems and load-balancing. While these are important considerations for CI/CD tooling, it’s equally important to have tools that can be adopted by all of the different groups throughout the organization.

BuildMaster is intuitive to learn and use. But companies do not have to take our word for it. Inedo now offers multiple training options including hands-on Classroom style to self-guided training. We understand that training is an important consideration, as different groups have different backgrounds and skill levels. Inedo is committed to making sure that training is not an adoption barrier to BuildMaster.

5. BuildMaster Focuses on Business Stakeholders

BuildMaster’s design is friendly to every level of your organization. Everyone can use the program easily, whether they are Executives, Managers, Engineers, or Developers.

Managers have 24/7 access to the dashboard which shows the information they need at a glance making it easier and simpler to report up. The dashboard shows in real time the status of every process in the production cycle and allows Managers to give approval and sign offs on the dashboard to deploy the next environment. The days of delay when Managers had to wade through a series of emails to check on status before giving approval are over. It’s all on the dashboard. No email needed. In addition, Managers can set up automated approvals that are triggered by the successful completion of milestones.

Inedo understands that Engineers live and die by the details. On BuildMaster, Engineers have access to very granular information including comprehensive execution logs that show exactly what happened on which server, when, and by whom. Developers can maximize their efficiency because they no longer have to wait for approvals.

Next Steps

Step #1: The High-Level Interview

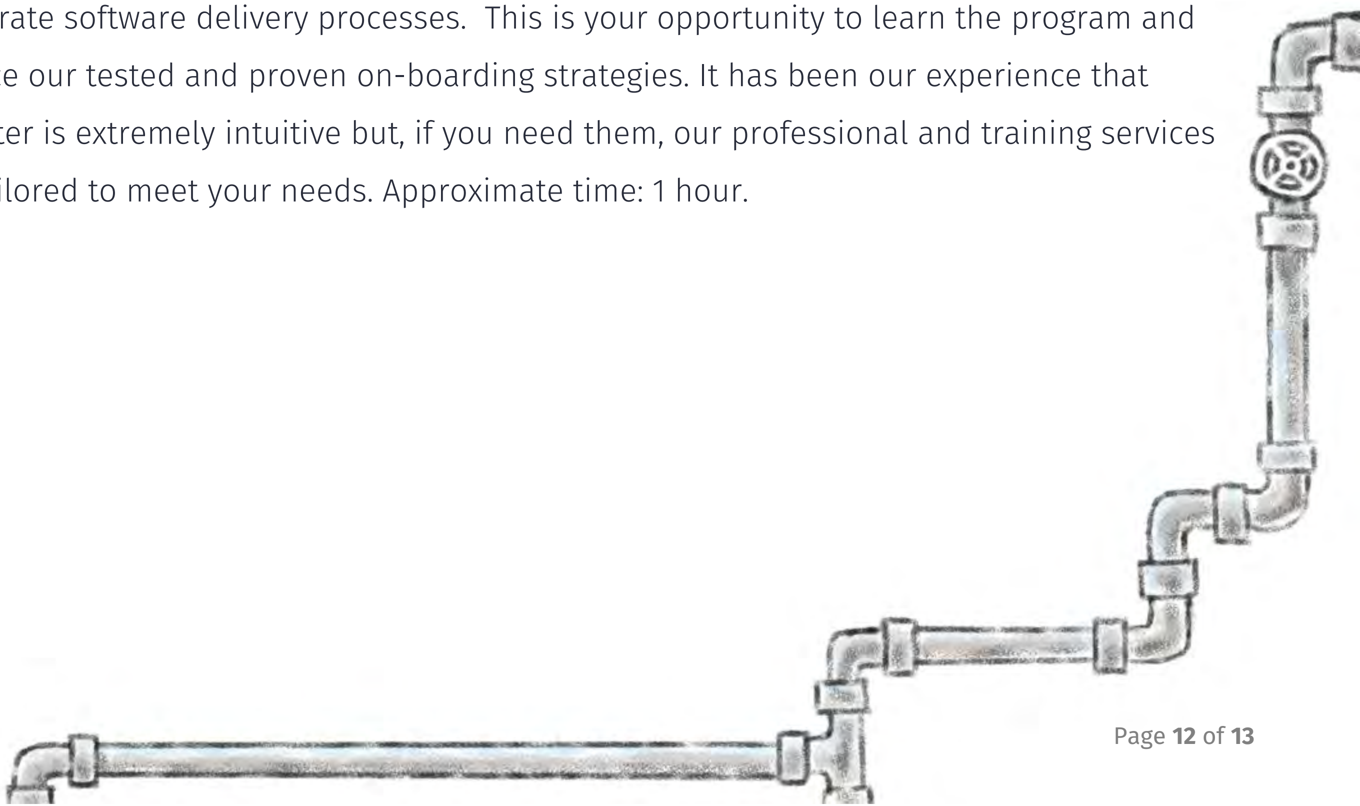
It’s free, easy, and happens at the Management level. We understand the stress behind delivering a great software product that is free of bugs and delivering it quickly. We want to talk to you, so we can hear your vision and learn your goals. We want to know what your current problems are and what keeps you up at night. Our Executive team is good at listening. Approximate time: 1 hour.

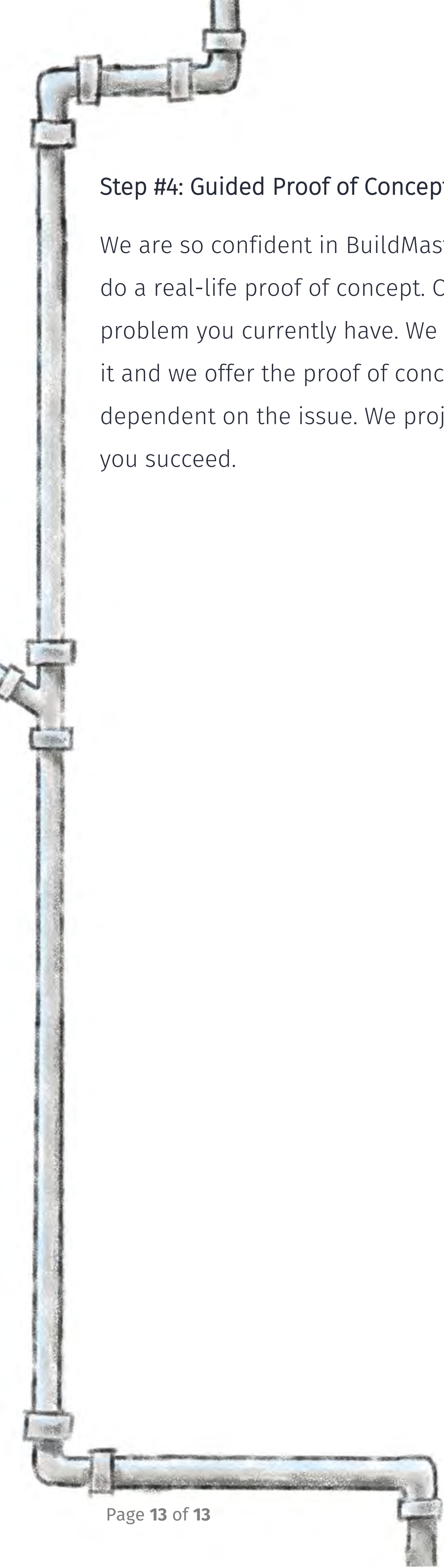
Step #2: Into the Weeds

Once you decide BuildMaster is worth more investigation, we get our technical and engineering team on the phone with your technical team. This conversation gets way down into the weeds. This is important because BuildMaster should be a good fit for every level of your organization and completely in line with your organization’s vision and existing and future technology goals. We spend time talking with you to ensure that no one wastes their time and resources. Approximate time: 1 hour.

Step #3: Hands on Demonstration

Inedo’s best engineers will demonstrate exactly how BuildMaster can automate, accelerate, and integrate software delivery processes. This is your opportunity to learn the program and experience our tested and proven on-boarding strategies. It has been our experience that BuildMaster is extremely intuitive but, if you need them, our professional and training services can be tailored to meet your needs. Approximate time: 1 hour.





Step #4: Guided Proof of Concept

We are so confident in BuildMaster that we will prove it works. Our team will work with you to do a real-life proof of concept. Our engineers will collaborate with your team and solve a real problem you currently have. We know the value of testing a new program before implementing it and we offer the proof of concept in recognition of this. Approximate time: It is highly dependent on the issue. We project a minimum of 4 hours to prove how BuildMaster will help you succeed.