Acquia

EXPERIENCE DIGITAL FREEDOM

# STANDARDIZING YOUR DEVELOPMENT WORKFLOW FOR DRUPAL 8 AND BEYOND

# CONTENTS



When it comes to Drupal development workflows, too many organizations lack a standard, established process that can be easily adopted and implemented. Development workflows tend to evolve organically, and sometimes erratically. Their specific structure — or lack thereof — is frequently influenced by starting conditions (people tend to follow whatever process was in place when they got there, regardless of how it originated), business decisions that happen outside the development process, and inherent biases of leadership, not to mention mergers and acquisitions.

While the workflow your organization currently uses may "work" most of the time, the haphazard nature of workflow processes can lead to real problems, from a chaotic, "wild west" approach to code submission and review, to the inability to determine where things have broken and why, when things inevitably break.

In this e-book, we'll describe, at a high level, the most common development workflow process that's currently being used in countless organizations and development shops today. We will then recommend a superior, standardized approach that teams can implement to drive better results as well as higher morale.

# SECTION 1

## The Common Path

# The Common Path

Here is how code typically goes from being a ticket in a system such as Jira all the way to being a feature that is available on the production website.

## 1. Pulling a ticket and submitting a pull request

Each developer works on their own local computer and is responsible for maintaining their local development environment or AMP stack.

All of the developers are working on the same codebase and developing the same websites, and they're using Git to move their code between their computers. Since every developer is responsible for their local environment—all of their tools, applications and configurations outside of Drupal—everyone on your team may be running completely different configurations and setups.

Next, the developer pulls a ticket from the project management system and does whatever the ticket asks. The developer then begins a GitFlow workflow process and pushes their work up to a hosted Git service (such as GitHub or GitLab). Then, the developer submits a pull request.

## 2. Confirming and approving the pull request

Another developer will see there's a new ticket in code review. This developer goes into GitHub, sees the pull request, reads the code, and makes sure it meets the stated requirements of the original ticket. In this process, the second developer should download the code to their own machine or another environment in order to confirm that it actually works.

Once confirmed, this developer either approves the pull request and moves it along to the next stage, or finds an issue with it, sends it back, and the process starts all over again.

## 3. Merging pull requests into primary branches of code

Pull requests get merged into the primary branches of code during the course of the sprint (or whatever project management methodology you are using). It might be the develop branch, the master branch or something else entirely. Whatever its next stop may be, code needs to be merged into a branch that will be worked on by developers, assessed by user acceptance testing or deployed into production (GitFlow facilitates this branching model).

Before deploying into the hosting environment, you need to back up databases (a process that may be automated with custom scripts), perform final QA and testing, follow launch checklist best practices and so on.

## 4. Deploying into the host environment

After being merged, the code now needs to be deployed to the hosting environment. Whether this environment is Acquia, AWS, a dramble of Raspberry Pis or something homegrown, someone will need to push the code manually into that server environment and then manually deploy it. If you are using best practices, you will deploy using a specific Git tag.

Now that deployment is complete, you will probably use some configuration management strategy to import configurations. You might also run some Drush commands to clean up your environment after a code deploy.

**Whether this environment is Acquia, AWS, a dramble of Raspberry Pis or something homegrown, someone will need to push the code manually into that server environment and then manually deploy it.**

# SECTION 2

## Problems with the Common Path

# Problems with the Common Path

Even though the workflow outlined above is used by countless developers and development teams, it has some inherent problems that create suboptimal conditions and outcomes.

## 1. Manual processes

Manual processes always bring with them the potential for human error, whether this comes in the form of someone forgetting to do something, someone making a mistake while they're doing something or everyone doing the same thing in different ways, introducing untraceable variations into your process.

## 2. No continuous integration (CI) or continuous delivery (CD)

**Continuous integration** ensures consistency of testing, primarily via automation. Developers have multiple testing frameworks at their disposal. Indeed, there are testing frameworks built right into Drupal core. There are also behavioral testing frameworks, such as Behat, which is built into Acquia's Build and Launch Tool BLT and allows you to write your ticket requirements as tests. Since the test is written as part of the ticket, in the course of deploying your code, developers can confirm that the code actually works.

**Continuous delivery** ensures that your package compiles consistently and without any errors across all of your platforms. It also ensures that your code is going to automatically deploy into the appropriate environments every time the build succeeds. This reduces the amount of manual review to be done and completely eliminates the manual push of your code to Acquia Cloud (or wherever it is going).

Continuous delivery reduces breaking overall, as well as makes things more secure. Indeed, it's a system that is not kind to breaking, and will let you know very quickly if anything went wrong. One benefit of this is that it ensures your developers only do code reviews when the code is ready. Finally, it also ensures that, when things are merged together, conflicts aren't created that then break your systems.

**Developers have multiple testing frameworks at their disposal. Indeed, there are testing frameworks built right into Drupal core.**

# SECTION 3

## There Is a Better Way

# There Is a Better Way

Fortunately, there are a number of best practices that, taken together, represent a more efficient and consistent development workflow. Here's what that looks like.

## 1. Create a standardized workflow

You have to actually think about and standardize your workflow. If you just cobble something together without planning or doing your due diligence, it will show in your results. Luckily, if you're using GitFlow, you're already on the right track.

Although we will recommend a host of standard practices here, we are not trying to be overly prescriptive. Some details will necessarily change based on the composition of your organization. **This means that, after reading through this, you need to ask yourself, "What's going to work best for me?"**

## 2. Have project management in place

You need to understand and acknowledge that project management truly matters before you start writing code. Accurately defining requirements and technical implementations and grooming your backlog to set up reasonable expectations for each sprint will mean the difference between success and frustration—if not outright failure.

It is also important to have an agile development process in place, rather than a traditional Waterfall process. Waterfall is a linear/sequential model where each phase must be completed before the next phase can begin, with no overlapping in the phases. But Waterfall was designed in a world with far less unpredictability, and it lacks the flexibility and agility required for modern workflow development.

Agile development methodology is based on iterative and dynamic development, allowing teams to be flexible and more responsive to customer requirements. Development is done in small iterations, usually bi-weekly sprints where use cases are created, developed and tested. At the end of each of these sprints, there is a deployable product available. You can also incorporate changes dynamically on the go and deliver the same at the next sprint drop. This cannot happen in Waterfall until the initial design and changes are fully implemented and deployed. Different agile development processes include Scrum, XP, and Kanban.

**Agile development methodology is based on iterative and dynamic development, allowing teams to be flexible and more responsive to customer requirements.**

## 3. Standardize the local development environment

It is imperative that you standardize across all systems involved in the development process; tools, machines, your continuous integration service, as well as all of your servers must be standardized. Local development environments must have configuration parity with production environments.

You also need to make sure that all of your environments are running the same version of PHP, the same version of Linux, all the same patches and so on. Configuration must be identical across all environments before you even get to the Drupal layer, so you can be sure that everything will work when you deploy it. (Resource management may differ, depending on the circumstances.)

### Other considerations

You might consider Git commit message enforcement. It may also be prudent to have code sniffing on every single commit to ensure code quality before your code gets into a system somewhere.

## 4. Use Composer

Previously, developers were responsible for maintaining all their local tools – their own version of Drush, their own version of NPM, their own version of PHP, etc. But this is no longer necessary. Composer can do this for you.

### Security

Because Composer lets you create explicit development requirements for your projects, you can include code that will only ever be shipped on local development environments. As a result, when your CI/CD service runs builds to deploy code into your production environments, it will not include anything that is marked “required development.” This means that you don’t ship any development-specific tools with your code and everything stays secure.

### Continuous Integration

Essentially, Composer makes it possible for you to create a second repository for code free of any development requirements. So, how do you balance two different repositories of code, one running Composer install containing all these development requirements, onerunning Composer install without them?

The only way to do this is to create two completely distinct code repositories, and have an automated process in place to manage the migration between them. This is where continuous integration comes in. Not only will it be responsible for running multiple tests on your codebase every single time you submit pull requests, but it will also be responsible for running the scripts that you ship with your codebase in order to create a production artifact that will eventually be deployed.

Remember, you are not just deploying the Git repo that all of the developers have on their computers. Instead, you are deploying a modified repo that is specifically designed for running in a production environment. And that is a radical change to the development workflow. Until now, no one has solved this problem of managing these two separate code repositories. There are homegrown solutions that try to do this, but they bring with them all the usual problems and headaches that come with homegrown solutions (no documentation, no standards, reliance on tribal knowledge to maintain, etc.)

Previously, developers were responsible for maintaining all their local tools – their own version of Drush, their own version of NPM, their own version of PHP, etc. **But this is no longer necessary. Composer can do this for you.**

## 5. Use continuous integration and continuous delivery

One great thing about continuous integration is that it's highly configurable. It can be used to automate testing of your entire application and run all of your predefined tests. At the same time, continuous delivery can be used to automate the deployment process to push code into another environment. (Of course, it will not automatically deploy code in your environments – we have mechanisms in place to prevent that. So, you do need a manual process here. Still, at the end of the day, continuous delivery effectively gets code into the right environment.)

# SECTION 4

## Additional Tools to Help Optimize Your Development Workflow

# Additional Tools to Help Optimize Your Development Workflow

There are a number of other tools available that can help standardize and secure your development workflow:

## Cloud Hooks

Cloud Hooks are scripts in your code repository that Acquia Cloud executes on your behalf when a triggering action occurs. While the Acquia Cloud user interface automates the most common tasks involved in developing a Drupal application, Cloud Hooks can automate additional tasks, including:

- Performing Drupal database updates each time you develop new code.
- Scrubbing your production database each time you copy the database to development or staging by removing customer emails or disabling production-only modules.
- Running your test suite or an application performance test each time you deploy new code.

## Cloud API

Cloud API is available to all Acquia Cloud customers. Acquia offers public APIs for many of its products to enable developers to build powerful tools, automate repetitive tasks and create custom development and testing workflows for websites. The Acquia Cloud API allows developers to extend, enhance and customize Acquia Cloud, and we introduce and expand our APIs regularly.

## Acquia BLT for Drupal

Acquia BLT, available on GitHub, provides an automation layer for testing, building and launching Drupal 8 applications. BLT provides both a common suite of tools and standardized structure to help developers reduce incidents of duplicated work, speed up project configuration and onboard new developers faster.

**Using Acquia BLT for your Drupal projects will help you with the following during your development cycles:**

- Provide a standard project template for Drupal-based projects.
- Provide tools automating the configuration and maintenance work for projects.
- Document and enforce Drupal standards and best practices through default configuration, automated testing and continuous integration.

## Factory Hooks

As you develop your Acquia Cloud Site Factory websites, you may have functions to perform during runtime, such as using Drupal core's Fast 404 capabilities when your website begins to load. You may also want to perform actions after certain events, such as deploying a theme or installing a website.

You can achieve these goals by triggering code execution through Acquia Cloud Site Factory Hooks. Factory Hooks simplify code deployments and trigger the execution of custom code at multiple points during website installation, deployment and runtime.

## Acquia Developer Studio

Acquia Developer Studio integrates many of the tools developers need to build better digital experiences and be more productive. With Acquia Developer Studio, you can focus on building the best Drupal websites using the tools you already know.

Acquia Developer Studio provides a command-line interface (CLI) that exposes Drupal development workflows that are tailored to Drupal developers who work in modern, virtual environments. CLI can be used in lieu of BLT to simplify some workflows.

**Drupal development goals a user should be able to accomplish using the Acquia Developer Studio CLI include:**

- Creating a new Drupal project
- Deploying to a remote site
- Refreshing local files and deploy between remote sites
- (Optional) Cloning an existing Drupal project, if applicable

Acquia Developer Studio Remote IDE is a browser-based integrated development environment (IDE), source code editor and Drupal development stack running in Acquia Cloud. It provides a local development environment in the cloud with all of your development tools built in. Although Remote IDE requires Acquia Developer Studio CLI to be previously installed, you can access Remote IDE without installing any additional software on your computer.

# SECTION 5

## Bring Your Development Workflow to the Next Level

# Bring Your Development Workflow to the Next Level

In many situations in life, it may be wise to live in the moment, not be overly concerned about the future and just see what happens. Drupal development, however, is not one of them.

Having the right plan, the right tools and the ability to detect and resolve problems as soon as possible will make your development workflow significantly more efficient and productive. You will also be leaps and bounds ahead of your peers, as most organizations still operate with an extemporaneous system that reactively fixes problems rather than proactively prevents them.

We are very excited to help organizations maximize their development workflows to create more products at a higher quality. To see how Acquia can be your trusted partner in bringing your development workflow to the next level, contact us.

## ABOUT ACQUIA

Acquia is the open source digital experience company. We provide the world's most ambitious brands with technology that allows them to embrace innovation and create customer moments that matter. At Acquia, we believe in the power of community — giving our customers the freedom to build tomorrow on their terms.

acquia.com

Acquia