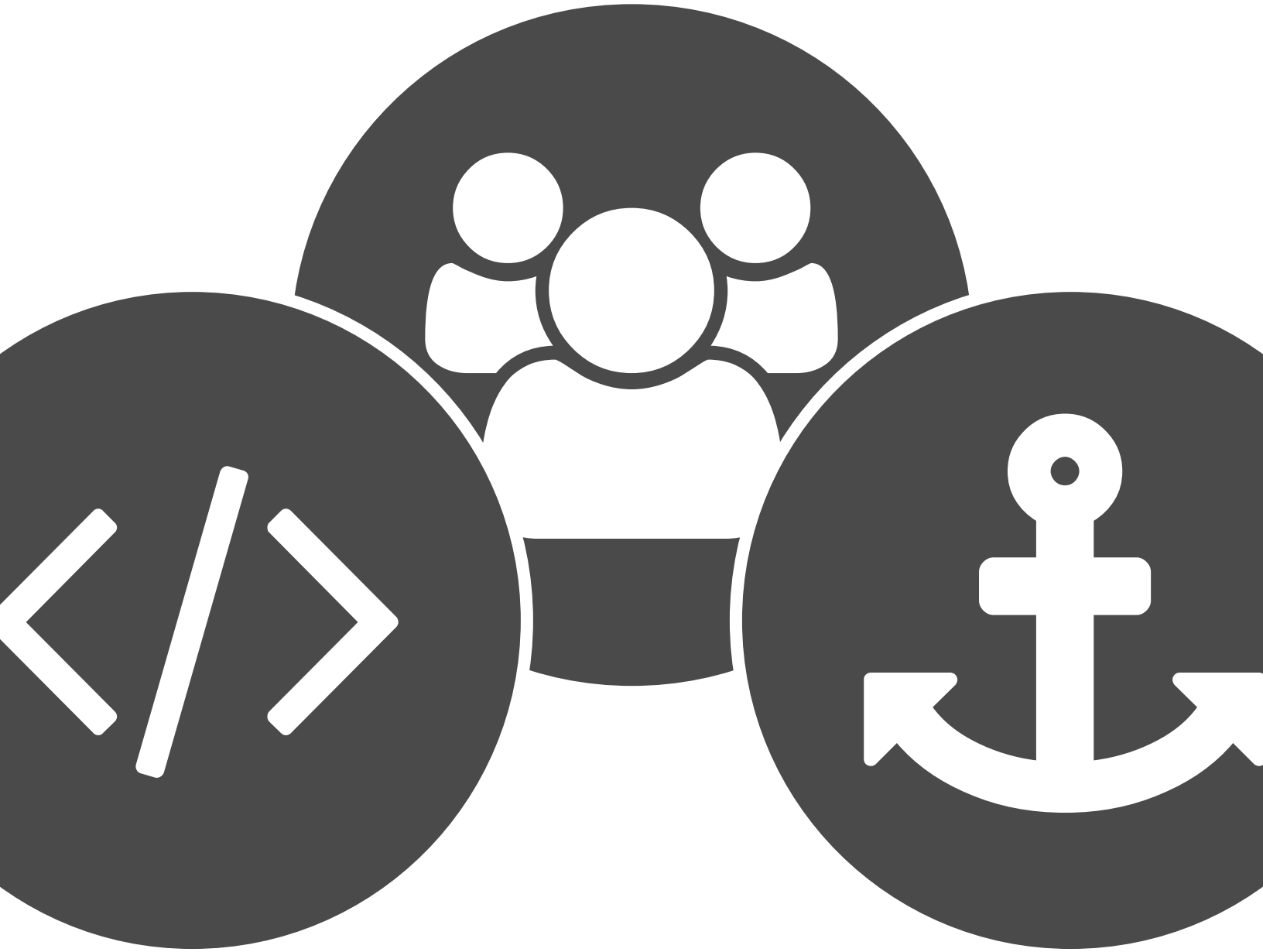
THE PLAYBOOK

THIS IS THE WAY WE WORK.



echobind

The Playbook is a guide that outlines our operating procedures and coding processes. These practices help us to create the best possible software products while ensuring a successful working relationship with our clients.

This is how we work.

Our Operating Process

Iterations and Hours

We work on client projects in weekly iterations, four days per week. We typically work work Monday through Thursday (barring no major holidays or staff illness). Fridays are normally reserved for echobind's internal business tasks. We work during normal business hours, but do not track the exact number of hours worked. All work that is completed during the week is clearly documented via source control, project management systems, chat and weekly recap meetings.

Invoicing and Payment

We require one week's advance payment to begin a project. In successive working weeks, we invoice every Friday. Because we bill by the week, we do not itemize our invoices. We keep things simple for us and our clients and charge one amount per developer, per week.

Payment is due within 15 days of the invoice date. Invoices can be paid by check, wire transfer, or credit card.

Communication

Slack is our preferred chat system. Each project receives a private chat room that brings together our developers and project stakeholders. This chat room is used to deliver all of our communications. We use Slack integrations extensively to notify us when important events occur.

Example events include: - staging/production deployment notifications - bug notifications - source control commits - project management updates

If you have files, documents, or images, you can drop them into the Slack chat room for group sharing. This practice eliminates cumbersome, back-and-forth processes such as sending multiple emails to several people or adding individual accounts to Dropbox shared folders.

In addition, we document our progress via GitHub (our preferred source control system) and Waffle (our preferred project management system). We put multiple processes in place so our clients never need to ask, "How is everything going?". We try to use chat as much as possible so we can communicate frequently and informally.

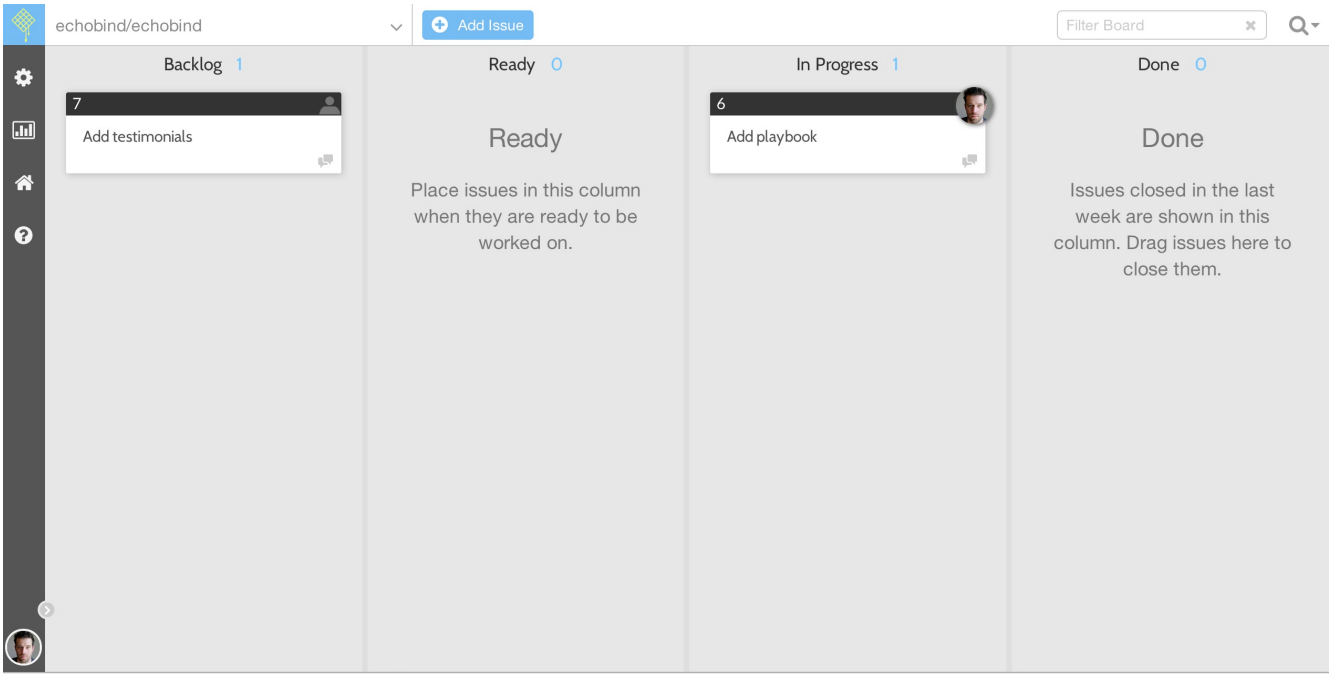
Meetings

We don't have a lot of meetings. Our client's budgets are far better served when we're actively working on a project. However, we do begin each project with a Roadmapping Session, which lasts between two hours and a full day. At the end of each week, we hold a quick recap meeting with our clients to discuss the iteration that was just completed. We limit the recap meeting to 30 minutes. Since we're a virtual office, we use Google Hangouts for the Roadmapping Session and the recap meetings.

Project Management

Our preferred project management software is Waffle. Waffle is a Kanban board tool that integrates directly with Github Issues, which we use extensively during projects. These tools are easy to grasp and client friendly.

We usually set up our boards in the following way:



As illustrated, features are first placed into the "Backlog" column. The client is expected to help write these features and review and suggest any necessary changes to them. Then, each week, we estimate which features can be completed and move them into the "Ready" column. While we do our absolute best to complete the "Ready" features within the week's iteration, we cannot guarantee that all work will be finished. Incomplete items from an iteration get moved to the next one. Once development or design work begins on "Ready" features, they are moved to "In Progress." Once completed, "In Progress" features are reviewed, and once they are accepted, items are marked as "Done." For more details about this process, see Our Code Process.

The First Week

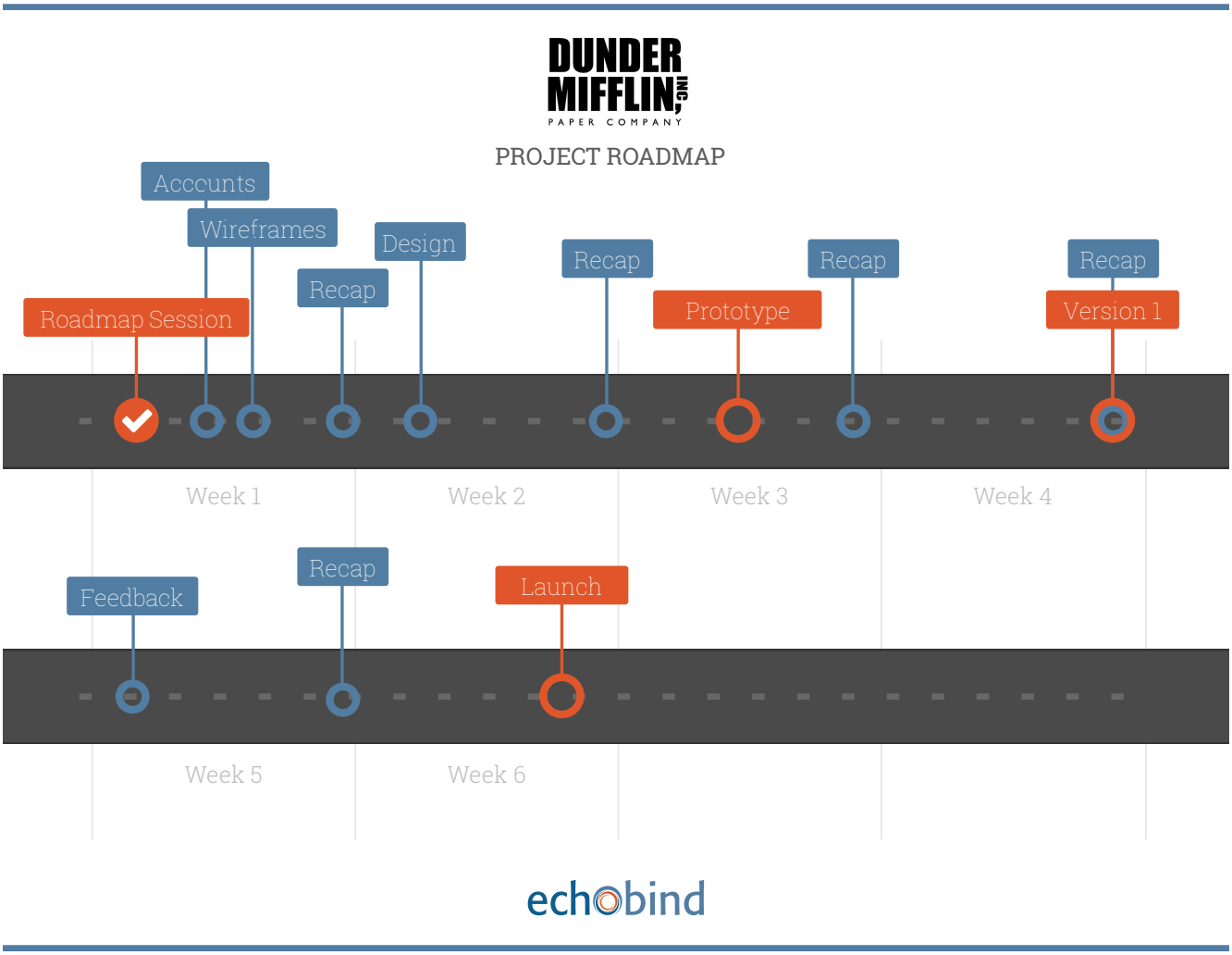
Our work in the first week can vary depending on the project's scope. But here are some of the things that we typically do:

- Conduct the Roadmapping Session with the client
- Set up any required accounts (hosting, source control, etc)
- Set up continuous integration (CI)
- Specify development process / workflows
- Audit existing code
- Write tests to cover untested functionality

The Roadmapping Session

Before we begin working on a new project, we conduct a Roadmapping Session. During this meeting, we discuss the project and its relationship to the client's overall business. We also talk about use-cases, product validation, and the project's overall goals. This conversation enables us to produce a Roadmap, which is a 1 to 2 page PDF that details the project's timeline and helps ensure its success.

A hypothetical Roadmap looks like this:



Maintenance Agreements

After completing a project, we can transition to a maintenance agreement. Maintenance agreements are billed as a monthly retainer and typically include:

- security updates and patches
- advice on best practices
- help architecting new features

- ongoing code review
- bug fixes

Maintenance agreements require a minimum three-month commitment and include up to 32 hours per month.

Our Code Process

Source Control

We use GitHub to host the source code for all of our projects. Github provides workflow features that are a core part of our development and design process. If the client requires that source code be managed in-house, GitHub offers an Enterprise Edition.

Frameworks

We use ember.js for the front-end of our apps and Rails for the back-end code and API. Both of these frameworks promote convention over configuration. They operate in a similar way and work really well together. However, occasionally, we use Rails for the front-end in cases where we can't use Ember.

Development Workflow

We follow a standard branching workflow during development:

1. A developer **creates a branch** from master that describes the feature. The branch is prefixed with the developer initials. Example: 'cb-allow-users-to-order-takeout'
2. During development, the branch should be periodically pushed to origin. When the code is complete with **tests written to back up the change**, the developer looks through the commit messages in the branch and makes sure that they are readable and understandable. Any commit messages that need to be updated are fixed using the interactive rebase feature of git.
3. The developer **creates a Pull Request** on GitHub. Slack notifies the team. We require all Pull Requests to be reviewed and marked with a +1 by another team member. This marking indicates that the changes are ready to merge to master.
4. When the Pull Request is opened, **a CI (Continuous Integration) server runs the test suite**, ensuring the entire test suite still passes. The CI server will automatically update the Pull Request with the pass/fail status of the test suite.
5. If a Pull Request has a +1 and passes CI, it is **merged to master** by the developer who originally opened it.
6. The CI server will run the test suite a final time, and **automatically deploy to a staging server** where it can be reviewed from the browser.
7. After a merge, developers should spot check the feature on the staging server so they can be sure everything works as intended in a production-like environment.

More on Continuous Integration

Testing Automation from a CI server is vital to our process. Testing Automation ensures our developers write proper tests, and it prevents changes to the app from "breaking" the test suite. The CI server is connected to Slack and notifies us if a Pull Request is ready to merge. Once merged,

the CI server auto-deploys to staging. Codeship is our preferred CI server because it's easy to use, supports multiple languages and test suites, and is reasonably priced.

Releasing to Staging

We auto-deploy to a staging server so that it always reflects master. This deploy serves as the final checkpoint before code is released to production.

Releasing to Production

We release code to production by either: (1) manually deploying through a script or git push; or (2) using the promote feature available on certain hosts. All of our developers can deploy to production, and they typically do so multiple times per day.

Databases

Postgres is our database of choice. Postgres is proven, reliable, and modern.

We recommend that our clients migrate to Postgres if they use Mongo to store their data. We can complete this migration for you. In our experience, Mongo is often used to store data in a relational manner, a purpose for which it was not designed. As a result, it performs poorly. We've even seen cases where Mongo has caused our clients to lose data. Mongo does not support transactions or rollback features. For these reasons, we have instituted a "No Long-term Mongo" policy on all new projects.

Code Quality

We value code quality. Code that is easily readable and cleanly written is easier to maintain and update in the future.

We use Code Climate along with code reviews in Pull Requests to help us maintain code quality.

We aim for about 80% test coverage on projects. We've found that striving for a very high percentage offers diminishing returns, and can cause developers to write bad or unnecessary tests. Instead, we rely on our developers to write enough tests to be confident that the project is sufficiently covered.

Code Style

For Ruby code, we follow a community style guide with a few minor additions:

- We allow omission of parenthesis when calling methods, e.g.: `User.eat 'taco'` and `User.eat('taco')` are both acceptable.

We are working on a style guide for our Ember applications.

Test Driven Development (TDD)

We practice TDD as much as possible. We write extensive unit tests and acceptance tests to ensure the integrity of each feature. In some situations, we write code first and write tests afterwards. But, usually we write the tests first. Either way, tests are always written as they are critical to ensure that future features and iterations do not break existing functionality or business rules.

Hosting

We host static sites and ember apps with Divshot and Ruby and Rails apps with Heroku. We are not sysadmins, and find our time is better spent making applications awesome rather than configuring servers. If clients need a recommendation for sysadmin work, we know some really great people that we can refer them to.

Browser Support

We design and develop for modern browsers. We don't support IE versions lower than 9. If users visit your app from an old browser, they will see a message that asks them to upgrade to a newer one.

Monitoring

It's important to track your application's performance metrics. We use the metrics feature of the Heroku dashboard to look at web response times, CPU, and memory usage. We use Skylight to detect and improve slow actions and queries. We are also experienced using New Relic.

Bug Tracking

We recommend Bugsnag or Honeybadger to track bugs. We find Bugsnag works best for environments with different types of codebases or separate projects. Honeybadger works best for Rails environments. Other services will probably work fine too. The important thing is that there is a system in place.

Thanks for reading our playbook.
We can’t wait to see what we get to build with you.