DJANGO**CODERS.COM**

THE **PROCESS**

This 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

Core strength built on healthy process

Our Operating Process

Iterations and Hours

We work on client projects in weekly iterations, five days per week.
We typically work work Monday through Friday (barring no major holidays or staff illness). We work during normal business hours, but usually 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 invoice every Friday or at the end of the month.
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 wire transfer.

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 Planning 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. We use Google Hangouts for the Planning Session and the recap meetings.

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 cloud shared folders.

In addition, we document our progress using git and project management tools (JIRA, Phabricator, Trello, Asana, etc). 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.

Project Management

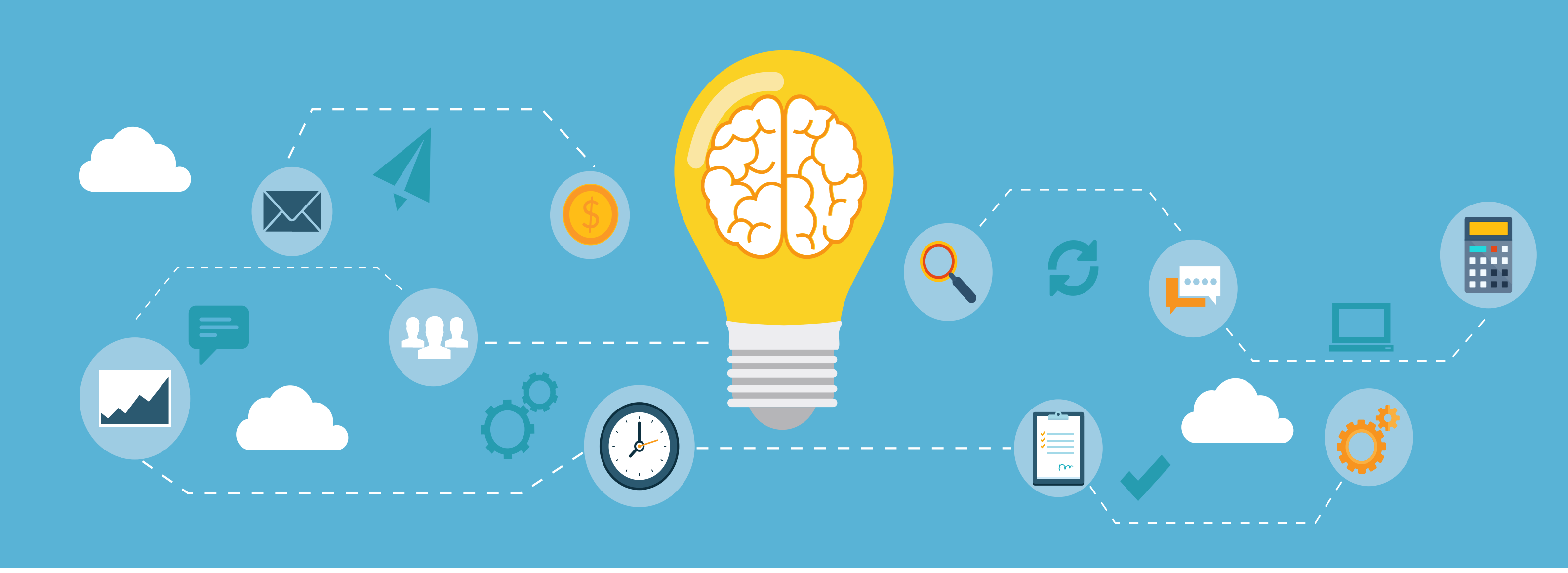
Our preferred project management software is Phabricator.
Phabricator is a collection of open source web applications developed by Facebook that help software companies build better software. It does ticketing with agile/kanban style boards, has good integration with source version control and continuous integration systems, a code review app that's very nicely integrated into our git flow, and lots of other goodies.
Compared to others, we found Phabricator to be light-weight, easier to administer and more productive. 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.
Some of our clients use Atlassian tools, we can work with them as wel

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 Planning 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 Planning Session

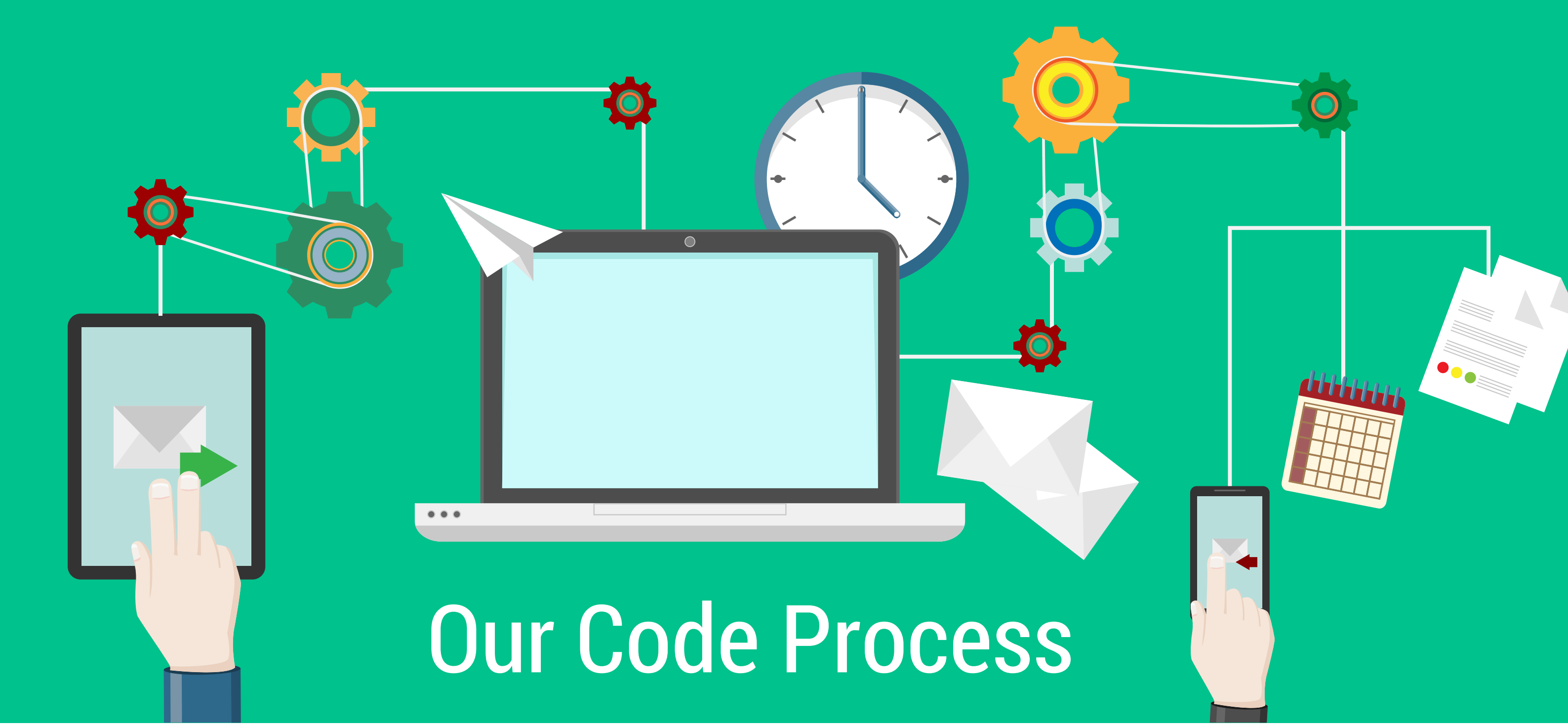
Before we begin working on a new project, we conduct a Planning Session. During this meeting, we discuss the project and its relationship to the client's overall business.
We also talk about use-cases, product validation, and the project's overall goals. This conversation enables us to produce a Roadmap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

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 Git to manage the source code all of our projects. We prefer to use hosted solutions, Github or Bitbucket but can use client's custom hosted repository if required.

Frameworks

We use ember.js for the front-end of our apps and Python based frameworks (Django or Flask) for the back-end code and API.

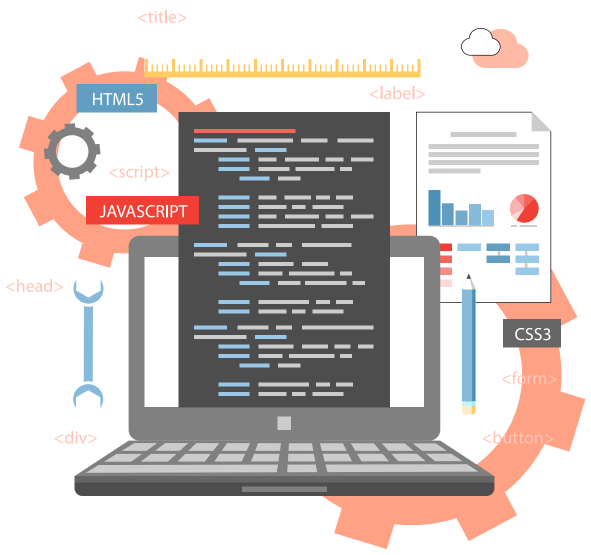
We have years of experience working with these framework and know the bits required to make them work really well together on large scale applications. However, occasionally, we use Django/ Flask for the front-end in cases where we can't use Ember.

Development Workflow

We follow a standard branching workflow during development:

- A developer creates a branch from master that describes the feature. The branch is prefixed with the id of the feature/task being worked on. Example: 'SU-506-allow-users-to-sign-in-using-facebook'

- During development, the branch is periodically updated with the latest changes from master, to avoid large conflicts. 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.
- The developer creates a Pull Request. The team is notified and we require that all Pull Requests are reviewed and approved by at least one other team member. This marking indicates that the changes are ready to merge to master.
- 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.
- If a Pull Request is approved and passes CI, it is merged to master by the developer who originally opened it.
- 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.
- 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 (Jenkins, Travis CI, Bamboo, etc) 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. Once merged, the CI server auto-deploys to staging. When running the test suites we also monitor the code coverage.

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, our database of choice, is a proven, reliable, and modern database suitable for a wide range of use cases. We also also have experience working with NoSQL databases (MongoDB, CouchDB, Neo4j)

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 tools like Pylint, JSHint, ESlint 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 Python code, we follow the guidelines promoted by PEP8, the only rule that we've not enforced is the number of characters per line. We were hitting the 80 characters limit too often, so we decided to use 120.
For JavaScript we follow the style guide published by AirBnb (https://github.com/airbnb/javascript). We use Ember-cli and follow their conventions and best practices for Ember apps that we develop.

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.
vv
For Ember apps we develop and write acceptance tests against client-side fixtures (using Ember-cli-mirage), so we can test the frontend independently from the backend. For integration testing we start by using the acceptance tests against the Python backend instead of the client-side fixtures and write additional tests for functionality that depended on the Python backend and was not included in the acceptance testing.

Monitoring

It's important to track your application's performance metrics. We use kibana to visualize metrics from different places and tools. We use tools like New Relic or Ruxit to detect and improve slow actions and queries.

Bug Tracking

We recommend Sentry to track bugs. We find Sentry works best for environments with different types of codebases or separate projects. Other services will work fine too (bugsnag, airbrake, etc) . The important thing is that there is a system in place.

We can't wait to see what we get to build with you.
http://djangocoders.com