Arcadia

ENGINEERING STRATEGY AND VALUES

A document that represents our values as an engineering organization and our processes for building software. This covers our philosophy towards coding and architecture, as well as an understanding of how we make decisions about our work. This document also serves to detail how we treat each other and expect to be treated, and the core pillars of our engineering culture. This is who we always strive to be.

TABLE OF CONTENTS

WHO WE ARE

We are egalitarian and diverse

Taking on climate change — one of the world's greatest challenges — requires diverse perspectives. All team members (regardless of seniority) feel comfortable voicing their opinion and feel that their opinions are appropriately and equitably considered. We believe a diverse set of voices makes a stronger team. We recognize that every person on the team makes an impact. We want to improve the status quo, both for our members and as a technology team.

We are collaborative and supportive

We learn from each other and foster an educational working environment. We support each other's growth through pair programming, code review, lunch and learns, and collaborating on difficult problems. We are constructive with peer feedback and respectful of each other. Engineers who are part of this team should always be able to find peers happy and willing to work through a problem.

We make time to learn new tools and features, ensuring that Arcadia is a place where engineers can grow their skills and expand their comfort zone. We value this constant improvement and share responsibility as a team to help others grow.

We don't feign surprise.

… you shouldn't act surprised when people say they don't know something. This applies to both technical things ("What?! I can't believe you don't know what the stack is!") and non-technical things ("You don't know who RMS is?!"). Feigning surprise has absolutely no social or educational benefit: When people feign surprise, it's usually to make them feel better about themselves and others feel worse.

—RECURSE CENTER MANUAL (H/T JULIA EVANS)

We distribute ownership and knowledge

We encourage people to take initiative and challenge themselves with exciting technical problems. We hire smart people, and we trust them to make good decisions. We have autonomy to use new tools and features when we believe they are the best solution to the problem. We recognize that the best way to engage team members is to give them ownership of a project, but we strive to avoid situations where any one person is a single point of failure or domain knowledge.

We move fast

We're taking on an industry that is filled with massive monopolies that have no incentive to care about their customers and a global problem that needed to be solved yesterday. We believe that moving fast is a competitive advantage and strive for this in our software development. We embrace this adventure and value frequent delivery and incremental improvement.

We take care of ourselves and each other

We're a startup engineering team, but we strive to balance work and life. If we see teammates struggling or on an island, we try to reach out and offer help or guidance. If we start feeling burned out, we talk about it and take time off. We're solving big problems, and we need to be healthy to come up with the best solutions.

We strive for excellence but embrace pragmatism

We always strive to author excellent, elegant solutions to problems, but we embrace shamelessness. We emphasize simplicity and straight-forward logic over cleverness. Sometimes you just need to solve the problem and move on, and as a team we understand that. Effective software engineering is all about compromise and constraints, and we embrace those, while keeping our eyes on our technical debt and strategically paying it off. We care deeply about the quality of our products.

We are kind and respectful

This value is represented in all of the others, but it's worthy of its own spot. Being kind, empathetic and respectful to your peers is a key requirement for being a part of this team.

HOW WE BUILD PRODUCTS

We care about our members and our colleagues

The software we build solves problems for our members and our team. We find satisfaction in making a better, more efficient experience for our internal teams. We know that the energy industry is complicated and can be difficult to understand, so we work hard to build software that helps our members navigate their options and gain a stronger understanding of their choices.

We use the right tool for the job, while understanding that sometimes the right tool is the one you already have

We are a polyglot organization. We believe that some tools are better than others for certain problems, and we enable the use of those tools.

We recognize that the best way to build an engaged, high performing team is to give team members ownership of problems or projects and empower them to make decisions about technology. That said, we know the perils of too much disparity in a stack, so we favor technologies we already have in order to maintain consistency and tooling, as well as flexibility within our team.

We reduce risk and increase maintainability by building small, meaningful isolated services

We believe that smaller, easier to understand codebases make for a better developer experience and a more maintainable architecture. Where possible, we try to isolate business logic into a dedicated service. We do this strategically, avoiding duplication of logic and data wherever possible.

We are transitioning from a monolithic architecture to an extensible, API-driven service architecture. We are doing this through a measured, deliberate approach.

The technologies currently in our stack

We use Ruby on Rails as our default server-side framework. We test our Ruby code with RSpec and ensure consistent style with Rubocop. Our primary databases are PostgreSQL.

Our front end framework is React. We test with our front ends with Jest, Enzyme and Puppeteer. We lint our JavaScript with ESLint and a custom variant of AirBNB's linting config. We use Apollo Server and Apollo Client to expose and interface with our data through GraphQL. We use Python for specific data science, analytics, and data collection tasks. Our data warehouse is Redshift. Our default infrastructure is AWS-based, configured with Cloudformation authored through Stax.

HOW WE AUTHOR AND CONTRIBUTE CODE

We PR every line of code that goes into a production branch

Code review is critical for growth (for both the reviewer and author) and knowledge sharing. We ensure everyone has the opportunity to ask questions or suggest changes to solutions, while ensuring that the author gains confidence in the code.

We take on tech debt strategically by tying it into the product roadmap and explicitly planning for it

We know that tech debt is never gone so we address it as an ongoing part of our product roadmap. This means we sometimes add tech debt in order to deliver business value, but we also sometimes need to move slower on product priorities in order to pay tech debt.

We flatten abstraction layers when they don't add value, and create abstractions when their value has been proven

We prefer duplication over the wrong abstraction. We strive to recognize the sunk cost fallacy in existing code and understand that abstractions that may have been right yesterday may not be right today.

When the abstraction is wrong, the fastest way forward is back. This is not retreat, it's advance in a better direction. Do it. You'll improve your own life, and the lives of all who follow.

—SANDI METZ

We let computers nitpick style and formatting

We use static code analysis tools like Rubocop and ESLint to nitpick style in code. If we find ourselves frequently asking code style or formatting questions in code review, we seek to find a way to automate this check.

We blamelessly deal with bugs and issues

We understand that code and decisions are made under constraints and compromises are the nature of engineering. For this reason, when issues or bugs manifest in the code, we deal with them without casting blame on our colleagues. We are empathetic to the compromises and work to move forward, not look back.

HOW WE OPERATE OUR SOFTWARE

We provision infrastructure through code

We believe in infrastructure that is easy to test, maintain, and recreate - and this means storing our configuration in code.

We practice continuous integration, and favor automatic deploys

Not all of our applications are automatically deployed, but where possible, we prefer to automatically deliver the production or master branch. We work towards having high confidence in our code through our CI pipeline. Where possible, we test end-to-end user flows through browser automation with Puppeteer.

We make DevOps a first class citizen within our organization

We try to involve ops early and often. We value their expertise throughout the software development lifecycle. They provide valuable insight into how to better structure our applications and architecture to be more resilient and secure.

HOW WE TEST OUR SOFTWARE

We believe in writing valuable tests

We strive to author tests that assert the logical parts of our software are working properly. We avoid spending extra effort testing already well-tested frameworks or libraries that we use.

We monitor test coverage and seek to always improve it

We monitor our test coverage and limit large, untested code spikes. We strive for a high percentage of test coverage, while understanding that 100% of lines covered is not a reasonable or valuable goal.

We require tests to pass before deploying

Applications must pass CI before being deployed to production.

Wherever possible, we test the full stack

We understand that end-to-end tests can be brittle and infrastructurally complex. However, when feasible, we prefer to test critical user flows across the entire stack.

HOW WE ORGANIZE OURSELVES AND OUR WORK

We wear lots of hats, but prioritize focus

When working as an engineer in a startup environment, people frequently wear several hats. It's not possible to have predefined lanes for everyone to work in as the software rapidly evolves and the business grows. With that said, we recognize the extreme cost of context switching and lack of focus, and work to minimize this as much as possible.

We favor small, quick stand ups. Each person has an impact on their team.

We organize into small, cross-functional, domain-oriented teams. The intent is not to silo or pigeonhole team members, but rather to ensure that teams understand where their time is best spent without having to worry about everything going on across the organization. To counterbalance siloing, we frequently collaborate across teams, especially on new features that may require different viewpoints.

We minimize meetings, but aren't afraid to have them when it makes sense

We work best with long, uninterrupted blocks of time to write and read code. That said, we know a 15 minute meeting can be a lot faster and more humane than a multi-hour, asynchronous back and forth on a pull request.