Make it real

Ideas are cheap. Make a prototype, sketch a CLI session, draw a wireframe. Discuss around concrete examples, not hand-waving abstractions. Don't say you did something, provide a URL that proves it.

Ship it

Nothing is real until it's being used by a real user. This doesn't mean you make a prototype in the morning and blog about it in the evening. It means you find one person you believe your product will help and try to get them to use it.

Do it with style

Just because we're building bad-ass infrastructure and tools doesn't mean it can't be cool, stylish, and fun. Aesthetic matters. See The Substance of Style

Slick and fun meets powerful and serious.

Before Heroku (and a few others, like Github and Atlassian), developer-facing products were almost always stodgy, ugly, and completely lacking in style or fun.

We're part of the consumerization of IT.

Intuition-driven balances data-driven

Hunches guide you to places to create new value in the product. Users don't really know what they want. Creating products people loves requires treating product development as an art, not a science; but products have to solve real user problems. Understanding impact of product changes to existing product is best done by mining the data. When you have a mature product and many users you have lots of data on how they are using it. Use that data to make evidence-based decisions.

See: Inspired: Created Products People Love

Divide and conqueor

Big, hard problems become easy if you cut them into small pieces. How do you eat the elephant? One bite at a time. If a problems seems hard, think about how you can cut it into two smaller, easier problems. If one of those problems is still too hard, cut it in half again.

Wiggins' Law: If it's hard, cut scope.

Timing matters

If you're building something and just can't seem to get it right, maybe now isn't the right time. You learned something in the attempt, set it down for a while. Maybe in a few weeks or a few months you (or someone else) will pick it up again and find that the world has changed in a way that makes it the right time to build the thing.

Throw things away

It's not the code that is valuable, it's the understanding you've gained from building it. See James' startup school talk.

Never be afraid to throw something away and do it again, it will almost always be faster to build and much better the second (or third, or Nth) time around.

Machete design

Create a single, general-purpose tool which is simple to understand but can be applied to many problems. It's like the product version of occam's razor.

The value of a product is the number of problems it can solve divided by the amount of complexity the user needs to keep in their head to use it. Consider an iPhone vs a standard TV remove: an iPhone touchscreen can be used for countless different functions, but there's very little to remember about how it works (tap, drag, swipe, pinch). With a TV remove you have to remember what every button does; the more things you can use the remote for, the more buttons it has. We want to create iPhones, not TV remotes.

Small sharp tools

Composability. Simple tools which do one thing well and can be composed with other tools to create a nearly infinite number of results. For example, the unix methodology (stdin/stdout and pipes), see The Art of Unix Programming. Heroku examples include the add-ons API, logging/logplex, and procfile/the process model.

Small is beautiful. This isn't just tools, it's also teams. Several small, autonomous, focused teams working in concert almost always beat a single monolithic team.

Put it in the cloud

I don't want to run software, ever. Given a choice between a great app that runs locally and a mediocre app that runs in the cloud, i'll always take the latter. (e.g. excel vs google spreadsheet, 1password vs lastpass, Things vs a textfile todo list on Dropbox) Services, not software.

Results, not politics

You "get ahead" in your heroku career by delivering real value to customers and to the company, not by impressing your boss or with big talk.

Decision-making via ownership, not consensus or authority

Every product, feature, software component, web page, business deal, blog post, and so on should have a single owner. Many people may collaborate on it, but the owner is "the buck stops here" and makes the final call on what happens with the owned thing.

The owner can and should collect feedback from others, but feedback is just that: input that the owner might or might not choose to incorporate into their work. If something doesn't

have an owner, no one should be working on it or trying to make decisions about it. Before those things can happen, it has to be owned.

Ownership can't be given, only taken. Ownership can't be declared, only demonstrated. Ownership begins with whoever creates the thing first. Later the owner may hand it off to someone else. If an item gets dropped for some reason (for example, the current owner switching teams or leaving the company), it's fair game for anyone else to pick up.

Apple's term for an owner is "directly responsible individual," or DRI.

Do-ocracy / intrapreneurship

Ask forgiveness, not permission.

- Do-ocracy
- Intrapreneur

Everything is an experiment

Anything we do -- a product, a feature, a standing meeting, an email campaign -- is always subject to change. That includes discontinuing or shutting down whatever the thing is. Ending an experiment isn't a failure, since we often learn the most from experiments that don't produce the results we wanted.

Own up to failure

Did you make a mistake by posting to the blog at the wrong time? By failing to document the feature before you shipped it? By screwing up a customer's app? By not respecting someone's ownership, or hurting someone's feelings?

Own it. Admit your mistake, say you're sorry (when applicable), and feel the failure to make sure you learned from it. Then, get back to work.

Gradual rollouts

Ease into everything. Use feature flags to activate people slowly into changes, then let it bake for a bit. Test out the message for a public launch by first sending it around internally, and later writing the private beta announcement. Collect feedback and adjust. By the time you're ready to take it public to a wide audience, you'll be fairly certain to have worked out all the kinks.

See: Crossing the Chasm

Design everything

Be intentional.

See:

- Less is More
- The Design of Everyday Things

Do less

Do we really need that feature? Can we delete that code? Do we really need that command? Can we outsource to or partner with another company so that we don't have the build and maintain something?

See: Ephemeralization

Question everything

The status quo is never good enough.

See:

- The Innovator's Dilema
- Blue Ocean Strategy

Interfaces matter

Everything has an interface. A platform has an API. A computer has a keyboard, a mouse, and a GUI operating system.

Teams have interfaces too. How do you file a bug or make a request? Where and when does the team collaborate with any other team?

The two critical components of a good interface are that it be narrow and well-defined.

For example, the add-ons API is extremely narrow. Add-on providers only need implement two calls: one to provision a resource, and one to consume it.

The add-ons API is also well-defined, with an API spec and the kensa tool which runs a live test to verify the correctness of your implementation.

A poor interface is one that is wide and poorly-defined. For example, the way that apps interact with the operating system in traditional server-based hosting is poor. The number of ways the app can interact with the operating system -- system calls, libc, the entire filesystem, executing binaries in subshells -- is essentially infinite and impossible to specify.

See: Explicit Contracts

Names matter

Think careful about how something is named. Pick exactly one name for each concept the user needs to track, and use it consistently. For example, add-on providers are always called providers, never "vendor" or "partner" or anything else. Writing a glossary can be a good way to design the vocabulary around something.

Maniacal focus on simplicity

There is no step 1.

CLI 4 LIFE

Web UIs are great for many things, but command-line interfaces are the heart of developer workflows.

Ignore the competition (except to borrow good ideas)

Tim O'Reilly said it best.

Write well

Good writing is a powerful tool for communication. Clear writing is clear thinking.

See:

- The Elements of Style
- On Writing Well
- The Book on Writing

Strong opinions, weakly held

Have a strong opinion and argue passionately for it. But when you encounter new information, be willing to change your mind.

See: Strong Opinions, Weakly Held

Candor

Be blunt, honest, and truthful. Constructive criticism is the best kind. Avoid keeping quiet with your criticism about someone or something for the sake of politeness. Don't say something about someone to a third party that you wouldn't say to their face.

See: Winning

Programming literacy for all

Software is eating the world. Everyone can and should be able to write software in order to have a stake in the future.

See: End-user computing