29: Race to Running Software

Get something real up and running quickly

Running software is the best way to build momentum, rally your team, and flush out ideas that don't work. It should be your number one priority from day one.

It's ok to do less, skip details, and take shortcuts in your process if it'll lead to running software faster. Once you're there, you'll be rewarded with a significantly more accurate perspective on how to proceed. Stories, wireframes, even html mockups, are just approximations. Running software is real.

With real, running software everyone gets closer to true understanding and agreement. You avoid heated arguments over sketches and paragraphs that wind up turning out not to matter anyway. You realize that parts you thought were trivial are actually quite crucial.

Real things lead to real reactions. And that's how you get to the truth.

The Real Thing Leads to Agreement

When a group of different people set out to try and find out what is harmonious...their opinions about it will tend to converge if they are mocking up full-scale, real stuff. Of course, if they're making sketches or throwing out ideas, they won't agree. But, if you start making the real thing, one tends to reach agreement.

—Christopher Alexander, Professor of Architecture (from Contrasting Concepts of Harmony in Architecture)

Get It Working ASAP

> *I do not think I've ever been involved with a software project — large or small — that was successful in terms of schedule, cost, or functionality that started with a long period of planning and discussion and no concurrent development. It is simply too easy, and sometimes fun, to waste valuable time inventing features that turn out to be unnecessary or unimplementable.*
>
> *This applies at all levels of development and "get something real up and running" is a fractal mantra. It doesn't just apply to the project as a whole, it is at least equally applicable to the smaller-scale development of components from which the application is built.*
>
> *When there is a working implementation of a key component available, developers want to understand how it will or won't work with their piece of the application and will generally try to*

> *use it as soon as they can. Even if the implementation isn't perfect or complete at first, this early collaboration usually leads to well-defined interfaces and features that do exactly what they need to.*

—Matt Hamer, developer and product manager, Kinja

30: Rinse and Repeat

Work in iterations

Don't expect to get it right the first time. Let the app grow and speak to you. Let it morph and evolve. With web-based software there's no need to ship perfection. Design screens, use them, analyze them, and then start over again.

Instead of banking on getting everything right upfront, the iterative process lets you continue to make informed decisions as you go along. Plus, you'll get an active app up and running quicker since you're not striving for perfection right out the gate. The result is real feedback and real guidance on what requires your attention.

Iterations lead to liberation

You don't need to aim for perfection on the first try if you know it's just going to be done again later anyway. Knowing that you're going to revisit issues is a great motivator to just get ideas out there to see if they'll fly.

Maybe you're smarter than me

> *Maybe you're a LOT smarter than me.*
>
> *It's entirely possible. In fact, it's likely. However, if you're like most people, then like me, you have trouble imagining what you can't see and feel and touch.*
>
> *Human beings are extremely good at responding to things in the environment. We know how to panic when a tiger enters the room, and how to clean up after a devastating flood. Unfortunately, we're terrible at planning ahead, at understanding the ramifications of our actions and in prioritizing the stuff that really matters.*
>
> *Perhaps you are one of the few individuals who can keep it all in your head. It doesn't really matter.*
>
> *Web 2.0, the world where we start by assuming that everyone already uses the web, allows smart developers to put this human frailty to work for them. How? By allowing your users to tell you what they think while there's still time to do something about it.*
>
> *And that last sentence explains why you should develop this way and how you might want to promote/launch.*

> *Get your story straight. Make sure the pieces work. Then launch and revise. No one is as smart as all of us.*

—Seth Godin, author/entrepreneur

31: From Idea to Implementation

Go from brainstorm to sketches to HTML to coding

Here's the process we use to Get Real:

Brainstorm

Come up with ideas. What is this product going to do? For Basecamp, we looked at our own needs. We wanted to post project updates. We wanted clients to participate. We knew that projects had milestones. We wanted to centralize archives so people could easily review old stuff. We wanted to have a big-picture, bird's-eye view of what's going on with all our projects. Together, those assumptions, and a few others, served as our foundation.

This stage is not about nitty gritty details. This is about big questions. What does the app need to do? How will we know when it's useful? What exactly are we going to make? This is about high level ideas, not pixel-level discussions. At this stage, those kinds of details just aren't meaningful.

Paper sketches

Sketches are quick, dirty, and cheap and that's exactly how you want to start out. Draw stuff. Scrawl stuff. Boxes, circles, lines. Get your ideas out of your head and onto paper. The goal at this point should be to convert concepts into rough interface designs. This step is all about experimentation. There are no wrong answers.

Create HTML screens

Make an html version of that feature (or section or flow, if it's more appropriate). Get something real posted so everyone can see what it looks like on screen.

For Basecamp, we first did the "post a message" screen, then the "edit a message" screen, and it went on from there.

Don't write any programming code yet. Just build a mock-up in html and css. Implementation comes later.

Code it

When the mock-up looks good and demonstrates enough of the necessary functionality, go ahead and plug in the programming code.

During this whole process remember to stay flexible and expect multiple iterations. You should feel free to throw away the deliverable of any particular step and start again if it turns out crappy. It’s natural to go through this cycle multiple times.

32: Avoid Preferences

Decide the little details so your customers don't have to

You're faced with a tough decision: how many messages do we include on each page? Your first inclination may be to say, "Let's just make it a preference where people can choose 25, 50, or 100." That's the easy way out though. Just make a decision.

Preferences are a way to avoid making tough decisions

Instead of using your expertise to choose the best path, you're leaving it in the hands of customers. It may seem like you're doing them a favor but you're just making busy work for them (and it's likely they're busy enough). For customers, preference screens with an endless amount of options are a headache, not a blessing. Customers shouldn't have to think about every nitty gritty detail — don't put that burden on them when it should be your responsibility.

Preferences are also evil because they create more software. More options require more code. And there's all the extra testing and designing you need to do too. You'll also wind up with preference permutations and interface screens that you never even see. That means bugs that you don't know about: broken layouts, busted tables, strange pagination issues, etc.

Make the call

Make simple decisions on behalf of your customers. That's what we did in Basecamp. The number of messages per page is 25. On the overview page, the last 25 items are shown. Messages are sorted in reverse chronological order. The five most recent projects are shown in the dashboard. There aren't any options. That's just the way it is.

Yes, you might make a bad call. But so what. If you do, people will complain and tell you about it. As always, you can adjust. Getting Real is all about being able to change on the fly.

Preferences Have a Cost

> *It turns out that preferences have a cost. Of course, some preferences also have important benefits — and can be crucial interface features. But each one has a price, and you have to carefully consider its value. Many users and developers don't understand this, and end up with a lot of cost and little value for their preferences dollar...I find that if you're hard-core disciplined*

> *about having good defaults that Just Work instead of lazily adding preferences, that naturally leads the overall ui in the right direction.*

—Havoc Pennington, tech lead, Red Hat (from Free software and good user interfaces)

33: “Done!”

Decisions are temporary so make the call and move on

Done. Start to think of it as a magical word. When you get to done it means something’s been accomplished. A decision has been made and you can move on. Done means you’re building momentum.

But wait, what if you screw up and make the wrong call? It’s ok. **This isn’t brain surgery, it’s a web app.** As we keep saying, you’ll likely have to revisit features and ideas multiple times during the process anyway. No matter how much you plan you’re likely to get half wrong anyway. So don’t do the “paralyis through analysis” thing. That only slows progress and saps morale.

Instead, value the importance of moving on and moving forward. Get in the rhythm of making decisions. Make a quick, simple call and then go back and change that decision if it doesn’t work out.

Accept that decisions are temporary. Accept that mistakes will happen and realize it’s no big deal as long as you can correct them quickly. Execute, build momentum, and move on.

Be An Executioner

> *It’s so funny when I hear people being so protective of ideas. (People who want me to sign an nda to tell me the simplest idea.)*
>
> *To me, ideas are worth nothing unless executed. They are just a multiplier. Execution is worth millions.*
>
> *Explanation:*
>
> - *Awful idea = -1*
> - *Weak idea = 1*
> - *So-so idea = 5*
> - *Good idea = 10*
> - *Great idea = 15*
>
> - *Brilliant idea = 20*
> - *No execution = $1*
> - *Weak execution = $1000*
> - *So-so execution = $10,000*

> - *Good execution = $100,000*
> - *Great execution = $1,000,000*
> - *Brilliant execution = $10,000,000*
>
> *To make a business, you need to multiply the two.*
>
> *The most brilliant idea, with no execution, is worth $20. The most brilliant idea takes great execution to be worth $20,000,000.*
>
> *That's why I don't want to hear people's ideas. I'm not interested until I see their execution.*

—Derek Sivers, president and programmer, CD Baby and HostBaby

34: Test in the Wild

Test your app via real world usage

There's no substitute for real people using your app in real ways. Get real data. Get real feedback. Then improve based on that info.

Formal usability testing is too stiff. Lab settings don't reflect reality. If you stand over someone's shoulder, you'll get some idea of what's working or not but people generally don't perform well in front of a camera. When someone else is watching, people are especially careful not to make mistakes — yet mistakes are exactly what you're looking for.

Instead, release beta features to a select few inside the real application itself. Have them use the beta features alongside the released features. This will expose these features to people's real data and real workflow. And that's where you'll get real results.

Further, don't have a release version and a beta version. They should always be the same thing. A separate beta version will only get a superficial walk through. The real version, with some beta features sprinkled in, will get the full workout.

The Beta Book

> *If developers are nervous releasing code, then publishers and authors are terrified of releasing books. Once a book gets committed to paper, it's seen as a big hairy deal to change it. (It really isn't, but perception and memories of problems with old technologies still linger in the industry.) So, publishers go to a lot of trouble (and expense) to try to make books "right" before they're released.*
>
> *When I wrote the book Agile Web Development With Rails, there was a lot of pent up demand among developers: give us the book now — we want to learn about Rails. But I'd fallen into the mindset of a publisher. "It isn't ready yet," I'd say. But pressure from the community and some egging on from David Heinemeier Hansson changed my mind. We released the book in pdf form about 2 months before it was complete. The results were spectacular. Not only did we sell a lot of books, but we got feedback — a lot of feedback. I set up an automated system to capture readers' comments, and in the end got almost 850 reports or typos, technical errors, and suggestions for new content. Almost all made their way into the final book.*
>
> *It was a win-win: I got to deliver a much improved paper book, and the community got early access to something they wanted. And if you're in a competitive race, getting something out earlier helps folks commit to you and not your competition.*

—Dave Thomas, The Pragmatic Programmers

Do it quick

> *1. Decide if it's worth doing, and if so:*
> *2. Do it quick — not perfect. just do it.*
> *3. Save it. upload it. publish it*
> *4. See what people think*
>
> *Though I'm always reluctant to add new features to things, once I have that "yeah!" moment of deciding something is worth doing, it's usually up on the website a few hours later, flawed but launched, letting feedback guide future refinement of it.*

—Derek Sivers, president and programmer, CD Baby and HostBaby

35: Shrink Your Time

Break it down

Estimates that stretch into weeks or months are fantasies. The truth is you just don't know what's going to happen that far in advance.

So shrink your time. Keep breaking down timeframes into smaller chunks. Instead of a 12 week project, think of it as 12 weeklong projects. Instead of guesstimating at tasks that take 30+ hours, break them down into more realistic 6-10 hour chunks. Then proceed one step at a time.

The same theory applies to other problems too. Are you facing an issue that's too big to wrap your mind around? Break it down. Keep dividing problems into smaller and smaller pieces until you're able to digest them.

Smaller Tasks and Smaller Timelines

> *Software developers are a special breed of optimist: when presented with a programming task, they think, "That'll be easy! Won't take much time at all."*
>
> *So, give a programmer three weeks to complete a large task, and she'll spend two and a half procrastinating, and then one programming. The off-schedule result will probably meet the wrong requirements, because the task turned out to be more complex than it seemed. Plus, who can remember what the team agreed upon three weeks ago?*
>
> *Give a programmer an afternoon to code a small, specific module and she'll crank it out, ready to move onto the next one.*
>
> *Smaller tasks and smaller timelines are more manageable, hide fewer possible requirement misunderstandings, and cost less to change your mind about or redo. Smaller timelines keep developers engaged and give them more opportunities to enjoy a sense of accomplishment and less reason to think, "Oh I've got plenty of time to do that. For now, let me finish rating songs in my iTunes library."*

—Gina Trapani, web developer and editor of Lifehacker, the productivity and software guide

True Factors

> *Next time someone tries to pin you down for an exact answer to an unknowable question — whether it's for a deadline date, a final project cost, or the volume of milk that would fit in the Grand Canyon — just start by taking the air out of the room: say "I don't know."*

> *Far from damaging your credibility, this demonstrates the care you bring to your decision-making. You're not going to just say words to sound smart. It also levels the playing field by reframing the question as a collaborative conversation. By learning how exact your estimate needs to be (and why), you can work together to develop a shared understanding about the true factors behind the numbers.*

—Merlin Mann, creator and editor of 43folders.com

Solve The One Problem Staring You in the Face

> *My absolute favorite thing to happen on the web in recent memory is the release and adoption of the "nofollow" attribute. Nobody talked about it beforehand. There were no conferences or committees where a bunch of yahoos could debate its semantic or grammatical nature. No rfc that could turn a simple idea into a 20-line xml snippet I'd have to read up on just to figure out how to use, and then not use because I wasn't sure if I was formatting for version .3 or 3.3b.*
>
> *It's simple, it's effective, it provided an option for people who wanted an option — and that is far more important when dealing with a population of the web that doesn't care about specifications or deference.*
>
> *Sometimes solving the next twenty problems is not as useful or as prudent as solving the one staring us right in the face. It wasn't just a small victory against spam (all victories against spam are small), but a victory for those of us who enjoy the simple and swift results that being a web developer is all about.*

—Andre Torrez, programmer and VP of Engineering at Federated Media Publishing