Learn How Products Get Started

Stories of how tech founders setup their products and the lessons they learned

John McDowall

REVOLUTION LAKE

Learn How Products Get Started

Stories of how tech founders setup their products and the lessons they learned

John McDowall

This book is for sale at
http://leanpub.com/learn-how-products-get-started

This version was published on 2014-06-24

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help John McDowall by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just purchased 'Learn How Products Get Started' on Leanpub.com!
https://leanpub.com/learn-how-products-get-started/

The suggested hashtag for this book is #lhpgs.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#lhpgs

Dedicated to Momoko Price. A-bah.

Contents

Introduction

It was 1995 and I was at High School in Scotland. The Computing Science teacher had collected some magazine that were published weekly in the UK, *PC Plus*, *BYTE* and so on. It was towards the golden era of British computer magazine publishing, where the articles were as much 'Write your own Word Processor', complete with code listings that had to be types in by hand, as they were reviews of current software packages.

I'd had a Commodore 64 for around four or five years by this point, and had been learning to program by visiting the town library, searching for programming books in the Library index cards, ordering them from the central repository or another library, and waiting a couple of weeks for the phone call to arrive that the book was waiting for me at the library. Now I had discovered a treasure trove of programming information in these magazines. My fate was sealed.

In one of the older magazine from 1987 or so, they had serialized a book called Programmers at Work[1] by Susan Lammers. In the book, Susan had interviewed nineteen of the big programming names of that decade: Bill Gates, Gary Kildall, John Warnock and so on. With permission I clipped those articles and I remember thinking : *'This is what it's like to be a programmer!'* I fell in love.

Programmers at Work was already out of print by the time I discovered it in 1995, and even if it hadn't been it would have

[1] http://www.amazon.com/Programmers-at-Work-Susan-Lammers/dp/0914845713

been so rare a request in the area of Scotland I grew up in that it would probably have to have been ordered from Glasgow or Edinburgh if requested in a local book shop. In the following years I occasionally searched for the book on eBay, but to no avail. So I treasured my clippings.

Soon after I graduated with a Bachelor of Science (Honours) in Computing Science from the University of Glasgow in 2003, I was browsing some books in a second hand store in Glasgow, a habit that I perform in every City I ever go to, and - *by the gods!* - here was a perfect copy of *Programmers at Work*. Deliriously I handed over a £10 note for the £2.50 book and told them to keep the change. Now I could read all of the interviews that had been hidden from me for all these years. I ate it up, and I still have that very book to this day.

Silicon Valley has changed a lot since then. Today, it's easier than ever to reach people, and a thought was growing in my mind as I read all of these success stories around Startups. All focus is usually on crafting the Startup's 'story', no one ever talks about the early days, when they were just people working on an idea. I couldn't find any interviews with people who were small, and had launched something.

The High Scalability website often gives very detailed breakdowns of the architectures behind successful startups that had large traction, but what about those little guys, or people in the early days?

As time went on and I started to work on my own projects and products, I sometimes found myself in an analysis paralysis, over-engineering solutions that no one was using yet, or worrying about details that were essentially premature optimizations. I had no one to turn to with real experience to say 'Don't worry about that'.

So I set out to create the book you are now reading. It's exactly what I wanted to read - a collection of Q&A style interviews with the technical founders of very small startups, or small startups that made it very big.

I hope it can inspire young programmers as *Programmers at Work* inspired me, and teach then that they can start building their ideas and getting people to use it with less than they think.

For everyone else, I hope it also teaches them that they too can get started with less than they think.

John McDowall

June 2014, Vancouver

P.S. I'm always interested in interviewing new people. If you've started something, and you've got some users, please email john@revolutionlake.co and let's set up an interview.

P.P.S. You can read the interviews from Susan Lammer's Programmers at Work here[2]

Who Should Read This Book

If you're thinking about starting up and beginning your journey of bringing a software product to the world, maybe as a side project, or a full quit-your-job single focus startup, there's much you can learn in each of these interviews.

Each interview contains nuggets of learning that will help you avoid mistakes that you'd otherwise make. Nuggets that will help you get there faster, and more cost effectively. We focus

[2] http://programmersatwork.wordpress.com/

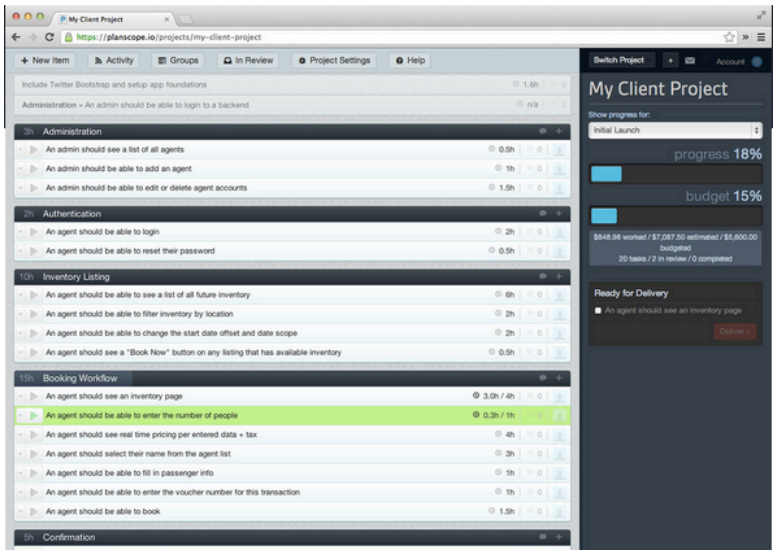
on the technical choices and mistakes that happened, and how each Founder overcame those obstacles.

Interviews contain lots of technical information, so you should have familiarity with programming and the way the web works. A lot of the interviewees used Rails to get launched, but you do not need to be familiar with Rails at all.

Ultimately, this book is about avoiding *analysis paralysis*, because there are so many technical choices and options at the beginning that it's easy to fall into a tar pit.

If you think you could benefit from these points, this book is for you :)

Planscope.io - Brennan Dunn



Planscope.io

Brennan Dunn is founder of Planscope.io[3] - a product that helps you and your clients stay on top of the budget & scope of your projects. He also offers a range of training courses for consultants with his books Double Your Freelancing Rate[4], The Blueprint[5], and his Consultancy Masterclass[6].

[3] http://planscope.io

[4] http://doubleyourfreelancingrate.com/

[5] http://doubleyourfreelancingrate.com/the-blueprint

[6] http://doubleyourfreelancingrate.com/build-a-consultancy

Tell Us About the origins of Planscope.

Well I guess the first thing to say is *"Why another Project Management app?"*. I created Planscope because I realized that there was a pretty big disconnect between budget and what actually got done. For a lot of price sensitive clients, or people who aren't booking us for a year at a certain weekly rate, say, getting non-technical clients to realize how they were spending their money and what they were getting in return wasn't always easy to do. You'd be recording work in a project management app, and then you'd send them an invoice through Harvest[7] or Freckle[8], and there'd be a pretty big disconnect between *'log six hour time block'* in Freckle, say, and it's very hard to correlate that with time spent on actual feature development. When things go bad, and the client asks *'Where'd all this money go?'*, you want to be able to have data to support you. Planscope makes that relationship between money and the time worked clear to both parties, and also does a little forecasting to make it obvious if there's budget to get everything else done. And if not, raise the budget, or reduce the scope.

I wrote the first line of code in November 2011, and I launched it publicly in February 2012, so it's been public for a little over a year now, and I do everything for the product: support, marketing, development and design.

[7] http://www.getharvest.com/

[8] http://letsfreckle.com/

At the point you opened the doors, what was the server architecture?

When I launched I was using Heroku[9] for probably the first two, or three months. Heroku went through a lot of issues about a year ago (2011) where it was down a lot, and since most of my audience were developers, I could reply to them with *'Well, Heroku is down.'*, and as it happened a lot of their favourite websites would also be down at the same time, since they used the same Amazon EC2 infrastructure that Heroku does, so it wasn't that big of a deal. Since April of 2012, I've moved to Rackspace[10] and I use a very small, lightweight VPS, and it's been the same VPS that I've used since, and it's never gone down, except maybe for my own idiocracy when I've botched a deploy. I have everything in place to monitor the server and I have backups being made but, knock on wood, I've never actually had to use them yet.

The server runs the web application, hosts Postgres, and I also run Wordpress and MySQL for the Planscope blog, and I know this is heresy to a lot of people.

What operating system did you pick and why?

It's Ubuntu 10.0.4. I'd been very familiar with Ubuntu from previous projects and client work that I did.

[9] https://www.heroku.com/

[10] http://www.rackspace.com/

What database did you pick and why?

I picked Postgres, even though my experience previously had only been in MySQL, because the Oracle acquisition of MySQL and a general community shift away from MySQL made it feel like the right choice to make.

What web framework did you pick and why?

Ruby on Rails. I've been doing Rails projects for the last few years, so I used what I knew. Generally speaking there's very little Ruby code in Planscope. The majority of the code is Coffeescript for the front-end, with the back-end being really basic responding to JSON requests and sending out emails.

What third party services did you rely on?

I use KissMetrics[11] for general revenue and analytics graphs. I use Stripe[12] for payment and charging people on subscriptions. I know that there's things like Recurly[13] that I could be putting in front of Stripe to handle recurring payments, but I'm just talking to Stripe directly. I'm using Customer.io[14] for lifecycle emails, and SendGrid[15] for transactional email. I'm also a pretty big fan of Intercom[16] for support and generally

[11] https://www.kissmetrics.com/
[12] https://stripe.com
[13] http://recurly.com/
[14] http://customer.io/
[15] http://sendgrid.com/
[16] https://www.intercom.io/

having a Dashboard in a box. I used to be very big into constantly refreshing Intercom, because I obsessed over metrics at the start of the project, but now I primarily use Intercom over email, and occasionally log into Intercom. Lastly, there's Pusher[17] for web socket stuff.

How much bandwidth do you consume every month? Is the cost included in your plan or do you go over?

Just whatever is included with my Rackspace VPS, it's not something I really have to think about.

What was the burn rate monthly cost?

My web hosting is around $50 a month, Pusher is $50, I'm grandfathered in on KissMetrics at around $30 per month. Customer.io is $29, Stripe is obviously payment on transaction fees. SendGrid is $80 a month for outgoing mail. I had to get the higher end package, even though I don't send that much mail in order to intercept sent email with SendGrid. So in total, it's probably around $250 - $300 a month.

What did you forget to account for at launch?

If I could go back and change something about how I did the launch, I would have done a bit more when it came to doing different on-boarding emails, and really optimizing

[17] http://pusher.com/

that. Granted, until you have a churn rate and you know why users are cancelling and have that data, you can't make a lot of these judgement calls.

I would also have collected credit cards immediately up front, because one of the mistakes I made was I offered guided tours over Skype. It was good for getting raw information about how people used the product, but I ended up spending a lot of time doing those Skype tours. I think if I had taken credit cards up front it would have done a better job of qualifying people by virtue of the fact that they have entered in their credit card details, even if they're maybe going to cancel in the trial.

When you want to cancel your account, you now have to enter into a text box why you are cancelling. Being able to track those cancellation reason, and correlate that with new feature development has been great. If I can squash all the things that are causing people to churn then it plugs those leaks in the Planscope funnel. If I can cut churn, which is a very easy thing to do compared to acquiring a new customer, I'm going to do it. And you also find out things like: users are cancelling because they didn't know how a feature worked, or even that a feature was actually already in the product!

Do you use any server config management tools (Chef, Puppet)

I hate administrating things. That's one of the reasons I went with Heroku in the first place; I don't want to have to set up nginx and Postgres and all those things. And I usually do it the wrong way, anyway! So I'm using a Rails plugin called

Moonshine[18], which I believe is just Capistrano with Puppet. It's been great. It gives me the benefit of Heroku, in that if I need to scale out horizontally and need to add another server, or I bring on someone else and I need a staging server, I could literally just do `cap:deploy` it's literally going to setup that server. Moonshine will set it all up, keep it all in check, apply security patches for me. It's very nice. It takes me maybe 10 minutes to set up a new server.

When you got your first real user, was the architecture the same?

I invited maybe four or five people before the launch in February to help me test Planscope out, and I had also previously built up a mailing list which had built up to about 300 people before I even started coding. I'd populated that mailing list via a few things on Hacker News and Twitter, and I had a landing page which collected email addresses. It was a full blown sales letter in the style of Amy Hoy's Charm sales letter[19]. I had been emailing the list off and on, so I sent them a message saying *'If you're ok with experimenting and you'd like to be a tester, get in touch'*, which is where I got those four or five initial people using Planscope. In February I then emailed the entire list and had them all come on board. Out of the 300 on the list, I probably got around 100 of them starting a trial, and maybe 20 or 30 of them converted into paying users. There was lots of dialog happening between that initial email capture and launch.

It's been that cheap Heroku account, followed by a cheap Rackspace VPS the whole time. At the point of getting the

[18] https://github.com/railsmachine/moonshine

[19] http://web.archive.org/web/20111229010249/http://charmhq.com/

mailing list on board, it became clear that Heroku wasn't going to be financially viable, and people tout the virtues of Heroku, but realistically I don't think `git push heroku` is that much different from `cap deploy`.

How many users do you have now? How has the architecture changed?

We just crossed about 200 paid users. I use Stripe to determine how many active account I have since everyone in the system always has active credit card details on file, and some percentage of them will be in trial mode. Kissmterics also lets me see how many users have been charged each month which is how I determine active users.

The server architecture hasn't changed at all since I moved to Rackspace. Having said that, I'm lucky to have maybe 200 people log in during a day, and most of the interaction happens on the front end. The server is not over taxed. I could probably hit over 1000 paying accounts with what I have now.

The current server is a 2Gb Rackspace VPS instance, with 80Gb disk attached.

Where do you see the architecture needing to change as your customers grow?

I've been No. 1 on Hacker News before, and that's hitting the VPS and the Wordpress backed blog and it's held up fine. What I would most likely do is set up new front end instances and throw a load balancer in there in front of them.

Did you run into any problems? What were they? How did you overcome them?

I've never really had any server issues. We haven't gone down. There hasn't been any attempt to compromise the server, which is all locked down via SSH keys. I haven't have any technical hurdles which made me doubt what I was doing or made me give up. I have backups in place that are hourly, so in the event of data loss I might lose an hour's worth of data.

I guess my biggest fear is waking up one morning and finding that I'm suddenly over 1000 active accounts, because I'm just not geared up to handle that level of support right now.

What's your favourite voodoo server/database/web server optimization trick?

I don't have any! I'm definitely a stock-setup kind of guy, because as I mentioned, I hate administrating things. Whatever Moonshine does, I follow along with. There's maybe one thing which is the tweak I have to route requests to `/blog` to Wordpress and everything else to Passenger.

Any parting advice or anecdotes?

Originality is overrated. You don't see a lot of original ideas. It's not like we're creating masterpieces to put in a museum.

If you're going to have a blog, don't do it on a subdomain.

There's a really good video of Steve Jobs being insulted by some technical guy back in 1997[20], and he says you have to concentrate on the customer experience first, not the technology. As engineers we get excited about all the technology, the Mongo, the Postgres and MySQL and all this stuff, but to the end user it probably doesn't matter at all. Obviously keeping the customer's data safe and keeping the App snappy is important, but it's less important than the 'what data store should we use' question. We tend to think about scalability long before we tend to think of 'Will this scale in terms of people wanting to pay for it'?. Obviously if you're doing a SaaS where you're expecting a lot of traffic from the start it's different, but for most wild-garden SaaS apps, you won't need to worry about that for a long time.

TL;DR

- Originality is fine, but solving a customer's pain point is awesome.
- Don't get trapped by premature discussions on scalability.
- Don't put your blog on a subdomain of your product site.
- You can get pretty far with basic Rails tools before you have to consider Chef or Puppet.
- You an run a lot on just one Rackspace VPS.

Further Reading and Tools

- Planscope.io[21]

[20] http://www.youtube.com/watch?v=FF-tKLISfPE

[21] http://planscope.io

- The Steve Jobs Tech Guy Insult[22]

[22] http://www.youtube.com/watch?v=FF-tKLISfPE