Entrepreneurship 101

Yishai Beeri

TOC

Product

How to build it

- Don't write it
- Well, write some
- Coding Processes
- APIs

What to build

- MVP
- Focus
- Measure Everything
- the Beta

Business

- Starting out
- Teaming up
- Talk with customers
- Fail Fast
- Funding
- NDA
- Patents

Product

How to build it

Don't Write It

If I have seen further it is by standing on the shoulders of giants (Isaac Newton)

- There's already a library or service that does it
- … better
- … and cheaper

Shoulders of Giants

Inside your code:

- Plenty of existing services and libraries to choose from
- Choose the stack that's right for you:
  - Programming Language (Python)
  - Framework (Django)
- Sometimes modifying costs more than writing from scratch
  - Maintaining a fork
  - Dealing with upstream updates
  - Monolithic vs. modular

more Giants

Around your code

- RDBMS (Postgresql)
- Key/Value Store (Memcached, Redis)
- Web server ecosystem (Nginx, Apache, Varnish)

For your code (and servers)

- [D]VCS (Git[hub])
- Tickets (Github, Redmine, Trac, JIRA)
- Continuous Integration (Travis, Jenkins)
- Monitoring (Pingdom, PagerDuty, New Relic)
- Measuring (ClickTale, Google Analytics)
- Load testing (Blitz.io)
- Deployment (Salt Stack, Chef, Puppet)

Giants in the Sky

Infrastructure

- AWS, Google AppEngine, Azure
- IAAS / PAAS (Heroku);
- General computing (PiCloud)

Hosted Services

- Video Encoding (Zencoder, Encoding.com)
- Sending emails (mailchimp, sendgrid)
- Push notifications (Urban Airship)
- Voice over IP (Twilio)
- Image processing, billing (Stripe),
- … and many more

Well, write *some* code

- Stuff that sets you apart
- Glue between 3rd party components
- The “core” of what you do

Code you write anyway

- Write for code readers
- Keep your code DRY
- Break down into modules
- Tests (don't overdo it), coverage
- Don't waste time on long spec docs
  - These grow very old, very fast

Coding Processes

Source Control (just use Github)

- Commit (very) frequently
- Always give useful comment
- Use branching generously

Bug database / Issue Tracker

- (ideally) Never write code without a ticket
- Connect tickets and commits
- Use for backlog / future / ideas

Repeatability

- Automated build
- Isolation
- Dealing with Data

"The Joel Test"

http://www.joelonsoftware.com/articles/fog0000000043.html

Automated Build

Can't be overemphasized

- Clean machine to running product
- Clean machine to local dev
- Take care of all dependencies
- Take care of initial data (fixtures) too

Isolation

Isolate development environments

- Underlying dependencies, versions
  - Avoiding "DLL Hell"
- Code tree
- 3rd party services
  - Different api keys for dev / production

Virtual Machines are a great help

Virtual Machines

- Cheap to set up (VirtualBox, Vagrant)
- (almost) perfect Isolation
- Work with various target OS
- Simulate clients / servers

Get more RAM!

Dealing with Data

Real bugs only happen with real data

Automate data setup for local dev

- Fixtures with sample data
- Dynamic generation of sample data
- Use backup from production data
- Sanitization…

APIs

Think about your APIs from the start:

- as a tool for modularization
- as a tool for documentation
- for better testing
- as a business model

- The UI (Web, Mobile) is just another API client

REST over HTTP is usually the right choice:

- Easy to implement (for clients too)
- Ubiquitous access

Product

What to Build

Focus

The single most important thing

Drop features if:

- You don't need them **now**
- They're not the core of what you do
- Are not being paid for by real customers

MVP

We'll talk later about "Fail Fast"

- What's the **minimum** viable product?
- MVP should allow you to prove your point
  - Enough to get customers
  - Enough to prove the core demand is real
  - Add other stuff later

The MVP also usually turns out to be the technological "heart" of things, and where your real technological value lies

Measure Everything

If you're dealing with consumer space (web, mobile), make sure you measure everything you can about how users interact with your products

- You don't know in advance what to measure
- Only measuring will tell you what works
- Prepare for a/b testing
- Take the time to study the data

Beta

- What goals do you want to achieve?
  - Weed out bugs? Market validation? Fine tune business model?
  - Define tangible, measurable success criteria
- How long? Who gets in?
  - Having users has real costs
  - Beta plan should support the goals
- Target specific geography? Demography? Use case?
  - Easier to control, grow and even dominate
  - Not always portable to new domain
  - Local launch: easier to get started, tougher on dev (BIDI), weaker market validation

All the above applies to full launch too

Business

Building your Startup

Starting out

- Why are you doing this?
- Are you cut out for this?
- Startup vs. Lifestyle Company
- Are you "on the bus"?
- Who else is coming along for the ride?

Teaming Up

Startups are hard

Going it alone is even harder

No one excels in everything

Team interaction is key to success (and even more to failure)

Equity distribution should align interests

Talk with your Customers

No one knows better than your customers.

You certainly don't.

If you hit the right pain, they will **love** to tell you all about it

Fail Fast

Working on the wrong startup is very risky

You're risking your most valuable asset: Time

Better to fail fast

- Launch quickly with smaller product
- Don't subsidize early customers
- Aim for real revenues early on
- Ask for feedback (and listen to it)
- Reiterate

Funding

- The traditional VC model is broken
  - Startups need less capital, smaller exits
  - Plenty of alternatives (angels, micro-funds, bootstrap)
- How much to raise? When?
  - "as much as you can, whenever you can"
  - "start raising the next round a day after you closed the previous one"
  - Risk: not failing fast
- Choosing your investors
  - Do they know what they're doing?
  - Do they know what you're doing?
  - How will they help besides money?

Valuation

- Shouldn't be the most important issue
- Nor the 2nd, 3rd or 5th
- Expect to give away 20-40% of the company in each round
- Avoid down rounds
  - Don't raise early at unrealistic high valuation…

Funding Mechanics

Equity financing

- Investor gives company $X, company issues Y shares
- Use pre-money valuation and price-per-share to simplify math

Debt financing

- Investor loans $X to company, paid back with interest
- Loan can convert to shares according to future events
  - Equity financing round of at least $Z
  - W months pass without an equity round
- Investor gets discount on future round
- Good way to avoid deciding valuation now

http://blakemasters.com/post/21742864570/peter-thiels-cs183-startup-class-6-notes-essay

Funding Mechanics – cont.

Share Classes

- Common Stock – what founders have (sometimes early investors too)
- Preferred Shares
  - Better protection of investor's money
  - Participating / non participating

Other Rights

- Board Seat (valuable if investor is smart and willing to help)
- Preemptive rights – protects investor from shareholder behavior
- Veto rights - protects investor from management behavior
- Anti-dilution - protects investor from bad bet

Funding Mechanics – cont.

Options

- Investors usually want option pool set aside
- Can be done before / after the round
- Between 5% and 15% is reasonable
- Vesting over ~3 years, 1 year cliff

Founder Equity

- Reverse Vesting (not all investors ask for this)
- Investors want founders to have substantial equity

NDA

Don't.

http://www.forbes.com/2004/01/27/0127artofstartmidas04.html

http://jonathantower.wordpress.com/tag/guy-kawasaki/

Patents

Don’t.

About Me

Co-Founder of Platonix Technologies (1996): www.platonix.com

- Boutique consulting / R&D shop
- Early stage investments in "work for equity" model
- Founded and managed 2 startups inside Platonix

- Computer Science B.Sc. from HUJI

yishai@platonix.com

Good Reading

- Paul Graham http://www.paulgraham.com
- Joel Spolsky http://www.joelonsoftware.com/index.html
- 37 Signals (blog and books): http://37signals.com/svn