Unblocking Your Hackathon

Get past common hurdles so you can actually build something!

Clara Bennett, Allie Meng, Aditya Natraj

Overview

- Start with a manageable scope
- Push your boundaries, but not too far
- Don't reinvent the wheel
- Get a little help from your friends
- Put your project out there
- Our prototype, in all its glory

Scope

Start small, so you can start building

A story

Once, in the long long ago, I went to a one-day hackathon, joined a project, and hacked for several hours.

Idea: Let's do something like WhoWritesFor, but for sources in articles instead of authors on the front page!

Result: A Hackpad with ideas and links to resources, a "hello-world" Chrome extension, a "hello-world" Django app, an unused Google group, and some sandbox code on my laptop

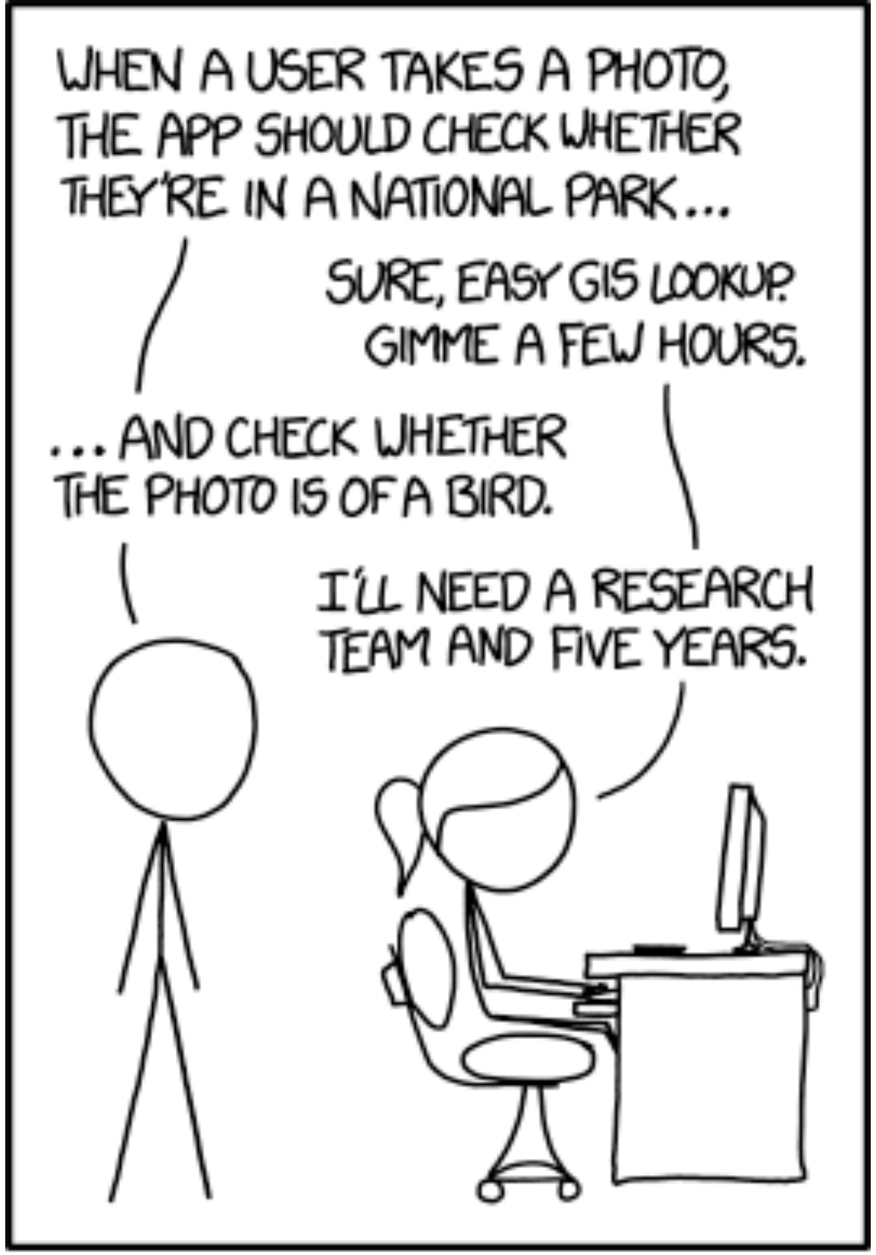
But why!?

The Feature List

This was for a one-day hackathon. That ended at 5pm.

- Identify gender of all sources in an article
- Recognize which names refer to someone who was mentioned earlier
- In-browser, show statistics about gender breakdown of sources
- Make sure that we can properly identify the gender of single-name people like Madonna
- In-browser, highlight names of sources in the article
- Allows users to interact with the highlights and identify incorrectly-highlighted sources or missed sources

😟😣😓😩😭😟😣😓😩😭😟😣😓😩😭😟😣😓😩😭



How to cut it down to size

- **What is the smallest piece[1] of your idea that you can build that is interesting and deployable?**
  Build that first, top-to-bottom: think thin vertical slices
- **How long do you think it will take to do that small slice?**
  If `T_hackathon >= 2 * T_slice`, try cutting it down smaller. Be ruthless!
- **How many independent pieces need to fit together for it to work?**
  If it's more than 2 or 3, try to cut it down or figure out if you can fake out one of the pieces. Hardcode something (e.g. the database) and come back to it.

Remember: If you finish early, you can always build on another thin, vertical slice.

---

[1] A good "small slice" depends in part on you and what you already know. What would be the least amount of work *given your team*, but still is interesting or useful?

But what would that even look like?

We built the project that my hackathon team never produced. Here's how we scoped it down:

- Identify gender of all ~~sources~~names in an article
- ~~Recognize which names refer to someone who was mentioned earlier~~
- ~~In-browser, show statistics about gender breakdown of sources~~
- ~~Make sure that we can properly identify the gender of single-name people like Madonna~~
- In-browser, highlight names ~~of sources~~ in the article
- ~~Allows users to interact with the highlights and identify incorrectly-highlighted sources or missed sources~~

Note that while name extraction and gender recognition are not easy problems, they are problems that many others have solved before. How do we know that? Google!

Learn something new!

But maybe not all the new things

What tools should I use?

- Hackathons are a great place to try out new tools and technologies!
- Too many unfamiliar tools can be a hinderance: focus on a few[2], and stick to known quantities[3] for the rest
- Try teaming up with people who have experience with the tools you don't, or work together to learn a new one
- Get to the new stuff early

---

[2] This is a pretty hand-wavy quantity, but it really depends on how weighty the tools are. A fully-featured MVC web framework (e.g. Django, Ruby on Rails) counts for a lot more than a library that solely translates unicode into ASCII.

[3] Don't let anyone talk you out of using a tool that you know just because another is "better suited" to the job. workable >>> optimal at a hackathon

Go beyond the tutorial

- Tutorials are everywhere: you should be able to find some good ones with some diligent Googling
- Good place to start, but highly simplified
- Maybe try a few, to get coverage of the basics from different angles
- But move on to building things yourself, so you find out what the tutorials glossed over

What tools did we use?

- Clara, for the API:
  - **Familiar:** Python, Flask
  - **Used Once Before:** natural language toolkit (nltk)
  - **New:** Heroku, various gender-prediction tools
- Allie & Aditya, for the Chrome extension:
  - **Familiar:** Javascript, AJAX
  - **New:** Chrome extensions

A highly incomplete list of tools to consider

- Web frameworks:
  - **Python:** Django (fully-featured), Flask (micro)
  - **Ruby:** Rails (fully-featured), Sinatra (micro)
  - **Javascript:** Angular.js, Ember.js, Meteor, React.js, Node.js (backend)
- APIs:
  - Twitter, Twilio, the Google API universe
  - Easily make requests against APIs with Postman on Chrome

Reuse & Adapt

Embrace your lazy side

Make use of existing solutions

- Most of the problems you need to solve today have already been solved many times by other people
- Probably at least one of those people has put their solution on the internet
- You can reuse or adapt those solutions! This can:
  - save time & energy for novel problems
  - leave more space to learn new tools
  - get you to a prototype quickly, on which you can then improve

What do I look for?

Think small! The narrower the scope, the easier to reuse or adapt

- Code snippets on StackOverflow or Github Gists
- Small, single-purpose or narrow-scope libraries
- Single-purpose projects and applications on Github

How we re-used code

Chrome extension

- Began with this keyword-highlighting Chrome extension
- Send page text to backend API on load
- Modified highlighted word selection to key off of API response, rather than user input
- Color-coded highlighting by predicted gender in API response

How we re-used code

API

- Pulled name phrases from text using this Github gist for named-entity extraction with nltk
- Return only PERSON-type named entities (excluded ORGANIZATION and PLACE)
- Used this pip-installable gender-prediction library[4] to predict the gender of *first* names
- Added some glue to predict gender on full name phrases (e.g. "Senator Charles Grassley, Hillary Clinton"), without knowing how to reliably isolate the *first* name
- Wrapped everything in a REST API

---

[4] We also tried using genderize.io's gender-prediction API, but all those API requests were too slow.

Working together

Two brains are better than one, but only if they communicate

Dividing the work

- A team of N people will be tempted to divide the work N ways: aim for less, unless the team is very small
  - Working alone can be slow and frustrating
  - Work will be finished at different times
  - The more independently-built pieces that need stitching together, the harder it will be
- Try instead to do some of the work in pairs, e.g. with one typer and one driver, or with one coder and one researcher
- Switch up the roles and pairs, to keep people fresh and occupied

Communicating

- When coding, the cost of interruption can be high, slowing you down in a time-sensitive situation
- Try using an asynchronous system like Google Chat to ask non-urgent questions and keep your teammates updated on your progress
- A coding partner can discuss and problem-solve with you while your teammates hack on their pieces
- Share code and assets using git & Github, or Dropbox or Google Drive, if "version control" isn't on your list of things to learn today

Pulling things together

- If possible, fake out the dependencies that haven't yet been built, so that you aren't blocked on building
- Start with a plan for how the pieces will fit together, but also be prepared for the plan to need changing
- Expect and budget time for something to be missed: there is *always* something
- Keep your teammates up-to-date on how your progress impacts interface decisions
- Integrate pieces as soon as is feasible, to catch errors early

How we collaborated

- Clara focused on the API, Allie & Aditya on the Chrome extension
- The first version of the API on Heroku returned names that all had `null` gender
  - Would have been better to release a version that gave a hard-coded response, regardless of the request content
- Initially, Allie faked out the entire API response in the Chrome extension
- Later, assigned randomized genders to replace the all-`null` genders in API response
- We integrated the two as soon as the API was on Heroku, discovering small bugs in both the API and Chrome extension in the process
- First gender-prediction solution was too slow when tested with the Chrome extension, so we implemented a backup
- Whole team shifted focus to Chrome extension once API was functional

Release Early and Often

Heroku[5] or it didn't happen

[5] For example

When to do the first release

- If you want to have a public internet application at the end of the hackathon, start releasing early, if you can[6]
- It will be difficult to debug release issues if you have to sort out platform, dependency, and code issues all at the same time

[6] Not before you test it locally, though. You can catch code-related bugs locally and only have to deal with release bugs when you deploy.

What tool should I use?

Heroku

- Probably best all-around recommendation
- Quick setup
- Free-to-use single-application instance
- Can deploy code from a git repository or from code stored in Dropbox
- Supports deployment of projects in Node.js, Ruby, Java, PHP, Python, Scala, Go, and Clojure
- Simple dependency management

Some other options

- **AWS Elastic Beanstalk:**
  - Free to use, just pay AWS usage (which has a free tier)
  - Quick setup
  - Handles infrastructure-management and scaling for you
  - Might be a good option if you plan to grow your application post-hackathon
- **Github pages:**
  - Completely free, no restrictions
  - Serves static files only: HTML, CSS, Javascript
  - Based on Jekyll static server, so you'd need to learn some about Jekyll
  - Deploy code just by pushing to a Github repo
  - Easy setup, but some potential for entering Ruby dependency hell

Demo

Questions?