Incremental Development

Maintaining and improving long-lived sites

The WHAT.

Long-lived

- Common for web applications and products (themes)
- Legacy code abounds!

Short-lived

- Microsites
- Marketing sites (movies, promotions)
- Whoops, you don’t exist anymore!

Our challenges.

Code from 4+ years ago.

Legacy code is everywhere.

How much are you living with?

It’s all around us...

- Third-party libraries that are defunct
- Cryptic code written by a technical wiz
- Custom In-house frameworks
- Languages that have a dwindling supply of developers
- Code you wrote last month in a rush

Technical debt is unavoidable.

Does your team pay off their credit card every month?

Be kind.

Don’t spend energy blaming previous developers.

	Reckless	**Prudent**
Deliberate	No one has time to review it, so we'll just commit to master.	We know this bug exists, but we have to focus on this feature instead.
Inadvertent	What are Javascript design patterns?	In hindsight, it's clear that we should have taken this approach instead.

- Martin Fowler

The WHY.

“

Legacy code. The phrase strikes disgust in the hearts of programmers. It conjures images of slogging through a murky swamp of tangled undergrowth with leeches beneath and stinging flies above. It conjures odors of murk, slime, stagnancy, and offal. Although our first joy of programming may have been intense, the misery of dealing with legacy code is often sufficient to extinguish that flame.

- Michael Feathers, Working Effectively with Legacy Code

Whoa.

That doesn’t sound fun.

Reaching the Tipping Point

- Developers are scared to change anything
- Worried about the “Butterfly Effect”
- No one want to work on that codebase anymore
- Architecture can’t be scaled/impacts future planning
- High turnover on the team

Be kind to your future self

- Be ready to support new features down the line
- Not all “features” are optional (eg. a11y)
- Browsers change rapidly and new APIs appear that you will want to use
- There is no “set it and forget it”

Maintenance is often overlooked

- It’s not hip, cool or sexy
- Not working with code that is the latest and greatest
- BUT being able to properly understand (and respect) legacy code is a skill
- It is critical and important

Preaching to the choir.

You already feeling the pain..

The WHERE.

I see dead parentheses.

Identifying your issues.

Questions to ask

- Is it currently broken? Like really broken?
- Can a hotfix be applied in the short-term?
- When is the last time the code was touched?
- Are you refactoring just to be fancy?
- Does *anyone* understand what the code does?
- Does the old code affect new code?

Make friends with your codebase.

Leave that bitterness at the door.

Spend time with your code

- Practice reading other people’s code
- Work your way through the event flows (Characterization tests)
- Get acquainted before you make edits
- Identify easy fixes like removing old polyfills and unnecessary vendor prefixes (tiny wins, but still tidies things up)

Resist the urge to tear it all down

- Time investment is massive - most companies can't afford this (t = $)
- BUT small iterative improvements can have a big impact (especially in the front end)
- Team is familiar with current stack and introducing new one would cause major friction

Or maybe… tear it all down

- Higher-ups are on board, willing to invest
- Architecture is needlessly complicated for something that could be built with a modern framework
- MUST have clear spec to make sure there is not massive loss of features

The HOW.

(Mileage may vary)

A caveat...

- Focusing on the front end
- Suggestions are from personal experience and case studies from other companies
- Unfortunately every codebase is different

Documentation.

(collective groan)

Code style guides.

Get everyone on the same page.

Design patterns.

Rinse and repeat..

Tooling.

Linters, compilers, task-runners - make your life easy-ers.

Test Driven Development.

Can take a while to implement, but time savings in the long term.

Feature flags.

Only visible to a select few.

Version control.

For your sanity.

Releases.

And a changelog!

Some examples...

Just a couple - there are lots!

Case studies

- Github and removing jQuery from front end
- Dropbox moving from Underscore to Lodash
- Slack having to tweet about legacy code :(
- Dependencies - left-pad
- CloudZoom vs newer Zoom library

The WHO.

Warm and fuzzies.

It is TOUGH to work on legacy code all day every day.

“ *Fall in Love with the Problem, Not the Solution*

- Unknown

We’re all in this together

- Refactor your own code! Nothing is too precious to be reworked and improved
- Provide opportunities to work on new features
- Get team to take personal ownership of code that they are commiting

Tenacity and optimism

- Don’t become complacent with the code you are committing
- Know your contribution affects the long-term prospects of the business and the site

Get buy-in.

Use data to back up importance of tackling technical debt with those holding the purse-strings.

Know when to ask for help

- Employees aren't always going to have specific skill set you need
- Is it cheaper to hire someone vs. do it yourself?
- An expert can almost always accelerate the process
- Don't develop in a vacuum - different perspectives can be invaluable

The WHEN.

Is it worth it?

- Can be difficult to convince people in charge of budgets to go back and update “working” code
- Speak to fragility of existing code
- Spaghetti code impedes velocity of new feature implementation
- Performance!

Don’t bet the farm - wait out trends

- Difficult to hire for outdated tech
- Technology changes at incredible pace - think back 2, 5, 10, 15 years
- Popularity = more resources
- WordPress vs SilverStripe vs MODx
- React vs Angular vs Vue vs NewHotness

Start on the right foot

- Learn from past mistakes
- Make sure you have someone on your team who understands architecture
- Think ahead to future features
- Understand the mvp
- Build time for maintenance into your developers' schedule

Resources

- Working Effectively with Legacy Code - Michael Feathers (OOP)
- Life-changing magic of deleting code
- Human cost of technical debt
- How to conquer legacy code
- Getting to know a legacy codebase
- So you've inherited a legacy codebase
- Top 4 symptoms of bad code
- 10 Things you will eventually learn about JS projects

“

It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change.

- Charles Darwin

Thank you!

@AlfalfaAnne
@outofthesandbox