Scaling with Legacy Code
About LearnUpon

- Founded in 2012
- SaaS/Cloud-based LMS (Learning Management System)
- Headquartered in Dublin, Ireland with offices in Belgrade, Sydney, and Philadelphia
- Awarded Technology Ireland Company of the Year 2018
- 5,000,000 (and growing) users worldwide, over 20,000,000 course enrollments
- 142,062,346 questions answered, 209,471,619 API requests
- Ruby/Rails/Angular based application

About me

- Des Anderson - Co-Founder & CTO of LearnUpon
- 15+ years experience developing enterprise applications
What is Legacy Code?

- Mike Feathers - code that is not tested?
- Perhaps code that we cannot easily change?
- Very old code?
- Valuable code that we are afraid to change?

Legacy code. The phrase strikes disgust in the hearts of programmers. It conjures images of slogging through a murky swamp of tangled undergrowth...

Although our first joy of programming may have been intense, the misery of dealing with legacy code is often sufficient to extinguish that flame.

~ Robert Martin
Tip 1

When a dev sees legacy code, we know because they ask “Why did they do that?” and the answer is generally “don’t ask!”
What’s the source of Legacy Code?

- Create a startup (all companies started up!)
- Build it (ignoring Bertrand Meyer, open to extension, closed to modification) and they (customers) will come.
- Be successful and they (product health police) will come.
- IMO: Ultimately, bad design patterns and code smells (A framework like Rails doesn’t help! User.where?).

Why Care?

- Software maintenance consumes more than 50% of your team’s effort.
- Which equates to 40 to 90% of your costs.
- **Important**: Tech Debt/Legacy Code is a term that not everyone understands.
Tip 2

Don’t regret early engineering decisions?
You overengineered.
Technical “Death”?

- Call it Technical Health, it sounds more positive.
- What does it mean to you? Remember it’s just a metaphor (a bad one!).

Is it Technical Debt?

- Don’t do this: “Compared to ideals, how far are we from them?” Instead: Discover its meaning to you.
- Don’t do this: “Lots of compiler warnings? We have lots of technical health issues!” Instead: Know your indicators.
- Don’t do this: “The code is 7 years old... That has to go. REWRITE”... Instead: Think of it like COBOL.
Tip 3

It’s anything that prevents you from doing your job (effectively).
Rewrites are rarely worth it, overly complex, 2 versions supported influx etc.
What does it mean to you?

- Can new team members get stuck into code easily? (High Readability).
- Can you make changes/add features to your code easily? (Highly Extensible).
- Do you have lots of library/version dependencies? (Low Dependencies).
Tip 5

Tests are a tool. They are not your silver bullet.
It’s not easy...

- Github: Rails 5 upgrade took 1.5 years.
- Shopify: 100 devs for 1 year to get to Rails 5 (with automated tests a go-go).

@LearnUpon...

- Replaced JQuery with AngularJS all in one go (mostly).
- Replaced Rails 3.0 with 3.2, while rewriting many gems and moving service providers - Painful!
- We upgraded to Rails 5.2 recently - Less Painful!
- Now are looking into moving from AngularJS to Angular, Monolith to MicroServices.

Sounds utopian, but did it work?

- The first statement of the upgrade guides for Rails... “make sure you have tests”. Remember, it’s not all about tests.
- Split up preparation and refactoring/upgrading code.
- A nice side effect here is that you promote team discussion
- Ensure your downtime is kept to 0 (ship boxes of small t-shirts).
Tip 6

It’s a never-ending story, know this, and you’ll be at ease.
We learned, huddled, talked, teamed, everything and more...

So, “Conway’s Law” is your guide.
How to plan an attack?

1. **Measure the following:**
   - File Churn, Delivery Time, Dev Speed, #Devs, Function size, Readability (the forgotten gem).

   On the other side, **do not measure:**
   - Dev Speed, Completed Tasks, etc. and other such variable metrics.

   * Yes, I mentioned Dev Speed in both categories, why?

2. **Know your firewall:** Product/Management and most of all procrastination.
Tip 8

Find your garden path to the shed, not the BBQ.
The solution is scientific right? Not always...

- Developers source solutions from non-human sources like Stack Overflow.
- To embrace this... set guard rails.
  - 30-minute rules
  - Encourage buddies/rubber ducking
- Peer code reviews and coding.
- Control expectations to management (new team members vs. veterans vs. seniors).
- Agile/Scrum are extroverted tools for introverted personas which doesn’t help.
Tip 9

Stuck in progress (=) == lacking info, it’s not that you are inadequate.
Plan created, now who executes it?!

- Know who are your stables and your volatiles.
- Stables love process, they love to plan. Volatiles don’t.
- YOU NEED BOTH. Embrace diversity.
- Know your “rock stars” - they’re your stables, not your volatiles.
Surely I can do something NOW!?

- Coding linters/smell detection really help (SonarQube, Rubocop, Reek).
- Start removing code dependency, e.g. Devise, rails gem, Paperclip, etc.
- DelayedJob does not scale. Go for alternatives like Sidekiq/Kafka.... Or write your own.
- Raw SQL, avoid the niceties of AR, it doesn’t perform and breaks on upgrades.
Doing it all again would make it better, right? No.

- To start with, we’d probably not have a company to talk about right now.
- Perfectionism holds you back, be “nearly perfect”. Remember that engineering factoid from earlier. *Not regretting engineering issues, implies you overengineered it.*
- Attention to detail, Clean Code (Uncle Bob), good behaviours are key - because predicting change is impossible.
Tip 10

It’s only going to get worse. Because tomorrow, there WILL be Rails 7.0, SomethingJS 8.5, YetAnother++ ...

Be brave, dive in, own it. Keep scaling.
Bored? A TLDR/L slide just for you!

- If this talk feels wasted on you just know this...
- The enemies of scale surrounding your code are selling to management, procrastination and not making it a continued topic.
- If all else fails: Remember, not all code is bad. Measure your ability to add a feature non-invasively, with low file churn.
Questions?

Thank you for listening!
Except for you at the back browsing Instagram.