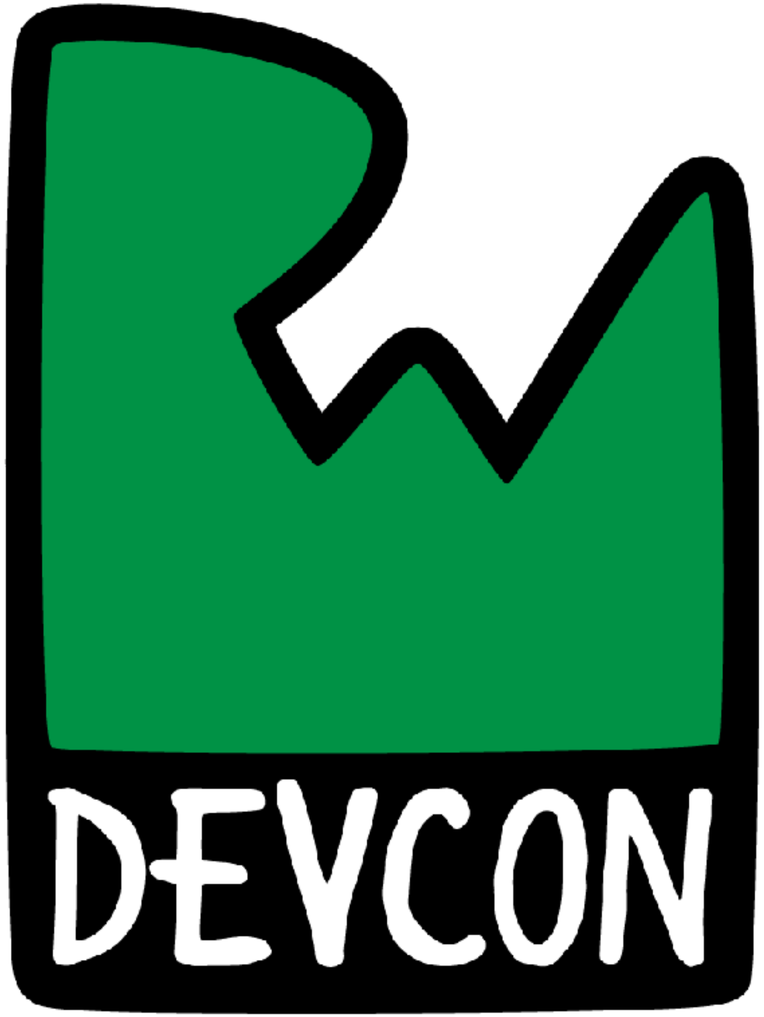
Session 9:
Spring Cleaning Your App



REFACTORING

What we will cover

- How to identify bad code
- The downsides of poorly structured code
- The benefits of structuring your code
- Techniques to refactor code safely
- Testing your code
- Preventing degradation of structure

What Does bad code look like?

- Large classes (Massive View Controller?)
- Code smells 
- Files which frequently change
- Code with many branches 

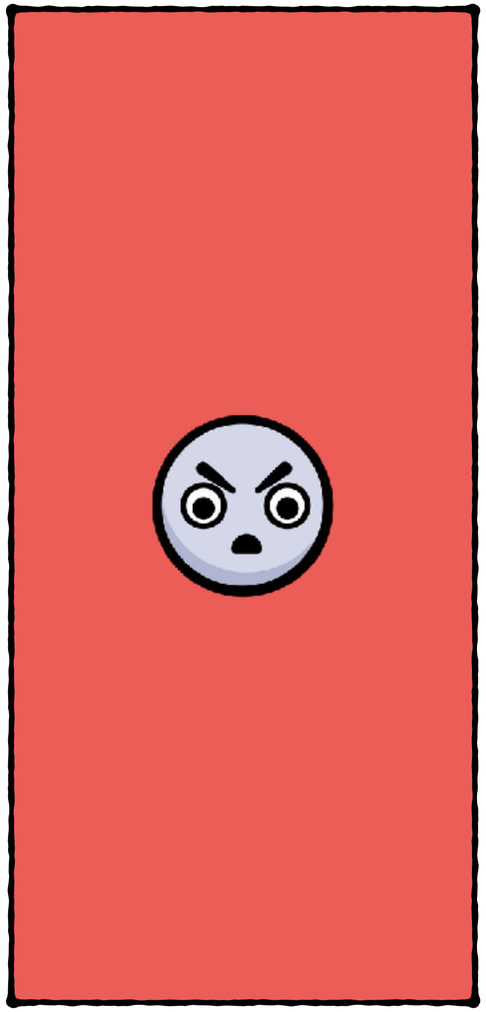
What Issues do bad code cause?

- Random bugs and crashes
- Hard to add a new features
- Overwhelming to new developers

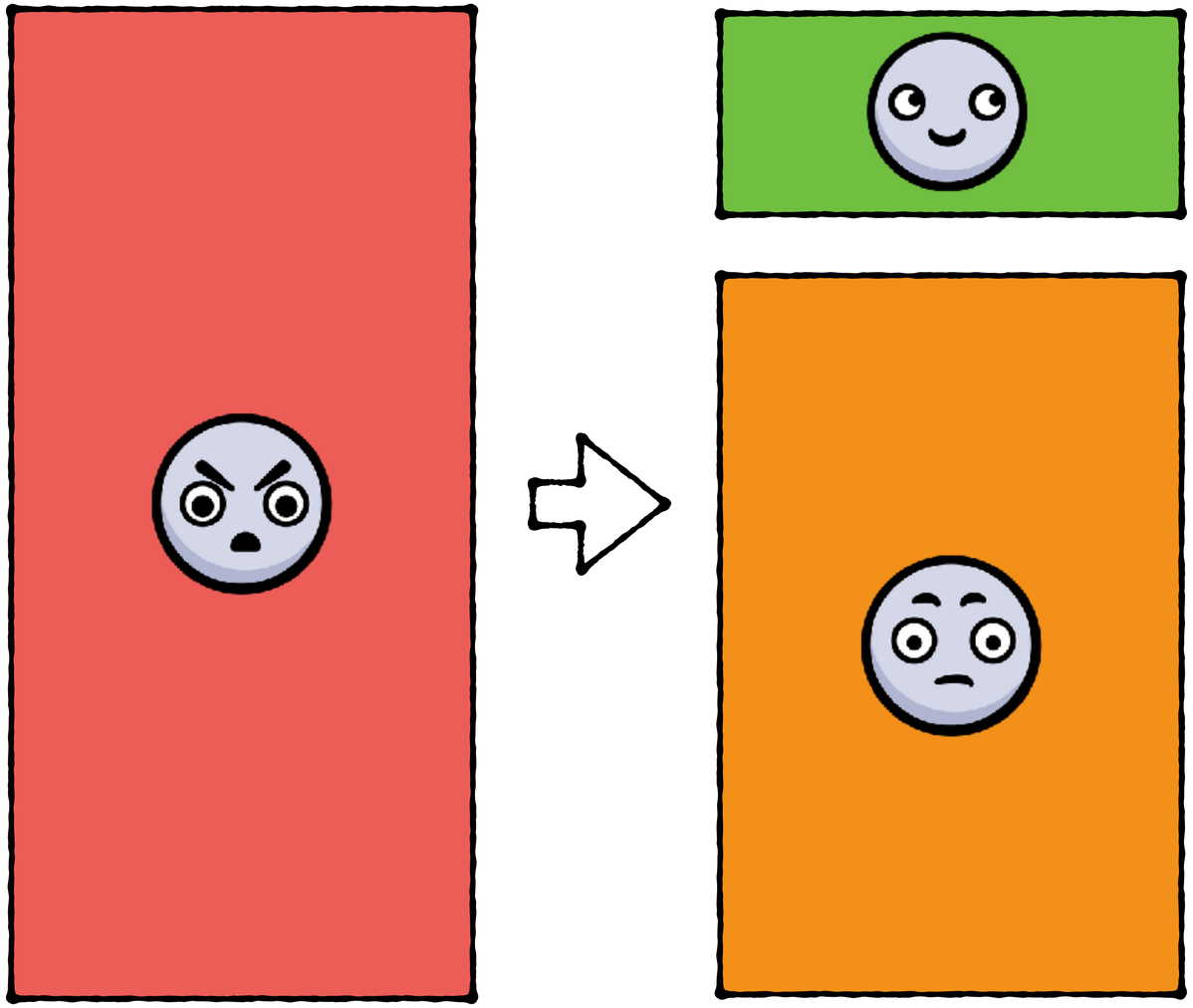
Refactoring legacy code

- Get to the quickest possible point you can start testing
- Avoid changing behaviour
- Commit frequently, build frequently
- Don’t be scared to roll back!

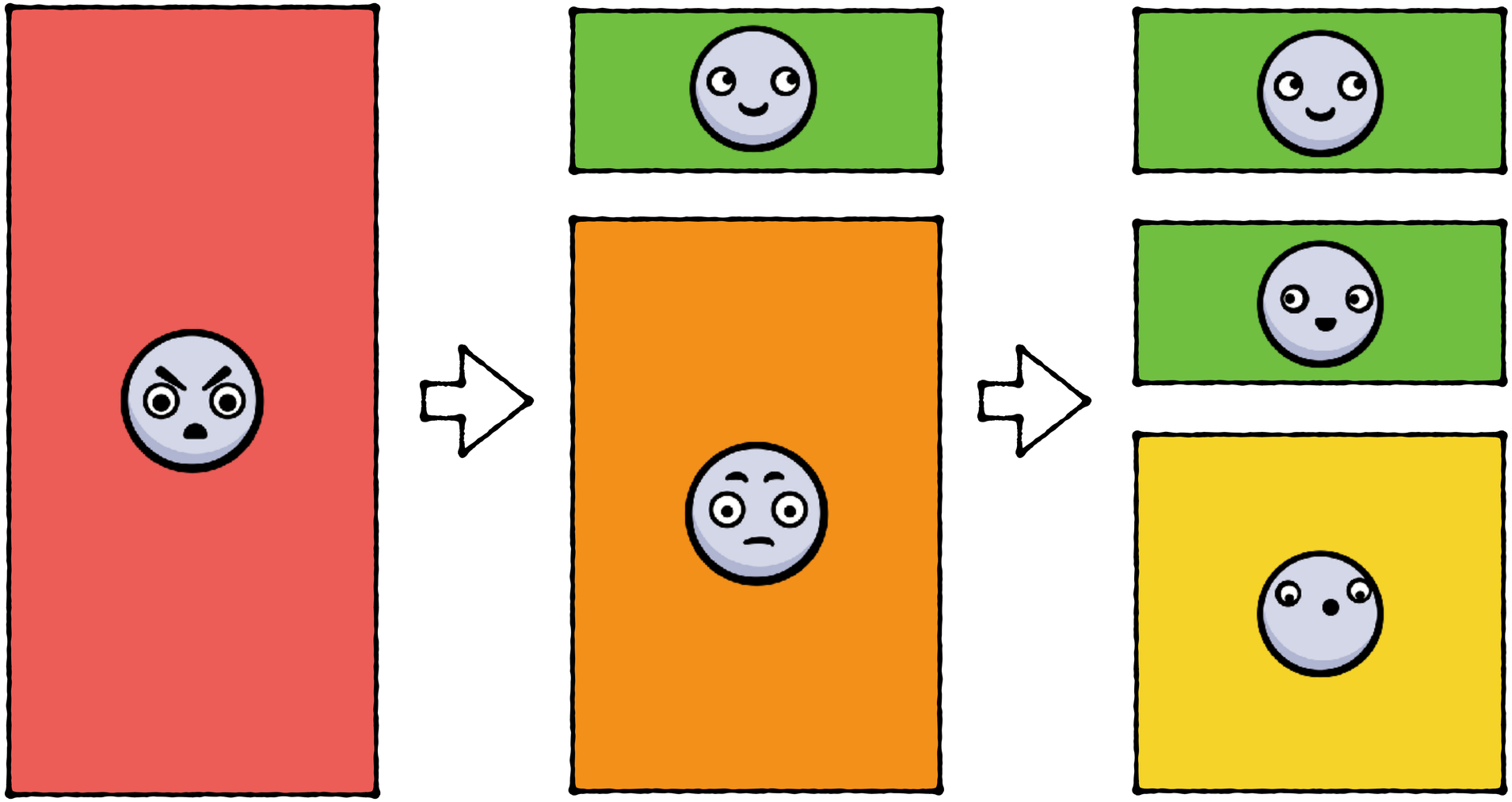
Refactoring a massive view controller



Refactoring a massive view controller



Refactoring a massive view controller

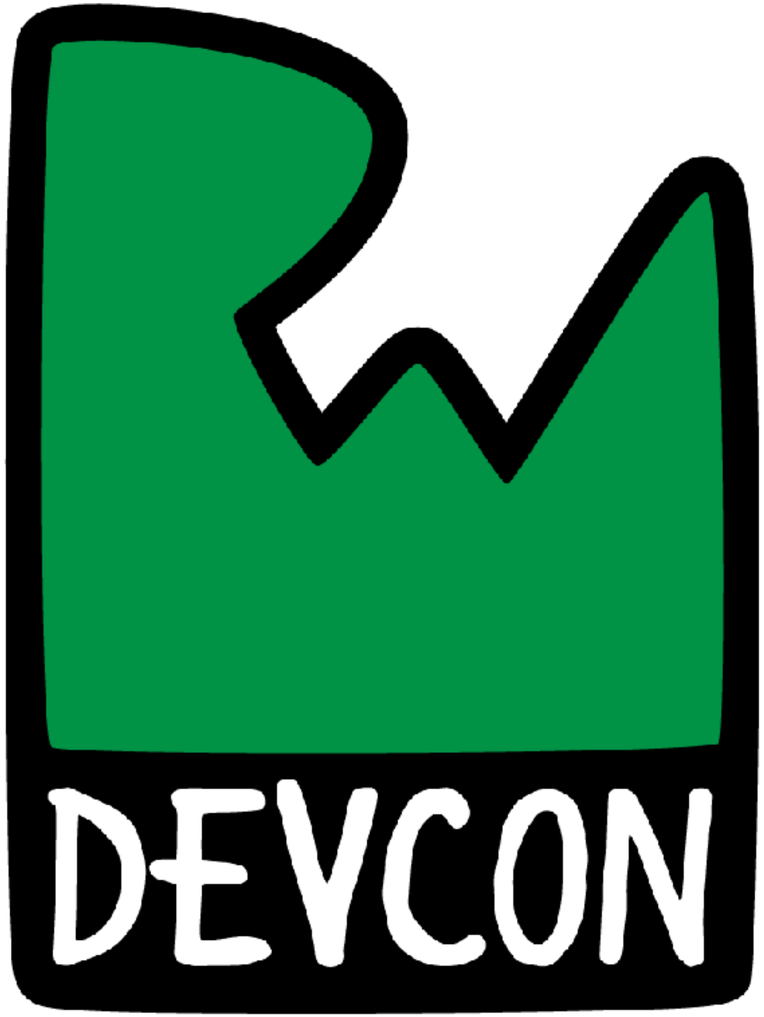


Refactoring

- Refactor methods out in order to work out different roles
- Naming is key to a good method
- Less moving around required when creating new objects later

Demo 1

Session 9:
Spring Cleaning Your App



TESTING

Preparing easy-to-test code

- Separate logic from view lifecycle
- Don’t have to “pretend” to be a view controller in tests!
- Silly names work until you find a better one

Why do we write tests?

- Allow us to be sure that our code is doing what it should be doing
- Confidence in future we won't break things 😎
- Earlier and more reliable feedback

Writing tests is hard

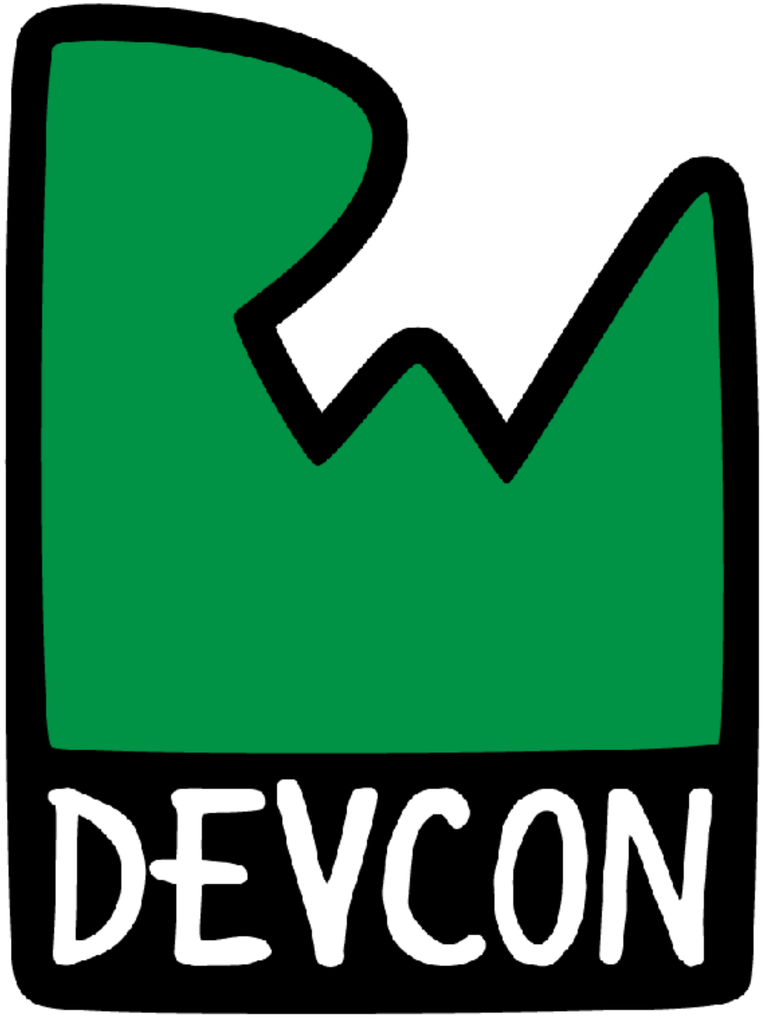
- Poorly structured code is very hard to test
- Code relying on external factors is unreliable to test (e.g. network, filesystem)
- Unreliable tests get ignored, are useless
- Good test names are important

Writing large tests

- When your legacy code isn't in units, it's hard to write a unit test :]
- Start with "when a user does X, they should see Y"
- Splitting apart the view means we can check things easier

Demo 2

Session 9:
Spring Cleaning Your App



Hardening Your Code

Making it harder to go wrong

- Sad fact that human error is the biggest issue in developing apps :]
- Testing makes it easier to find out when you've made a mistake
- Stricter coding makes it harder to make a mistake

Offensive coding

- Defensive coding means handling many eventualities
- We want to minimise the amount of these
- The best defence is a good offence!
- Make your code explicit about what it needs and how it works

Strong types

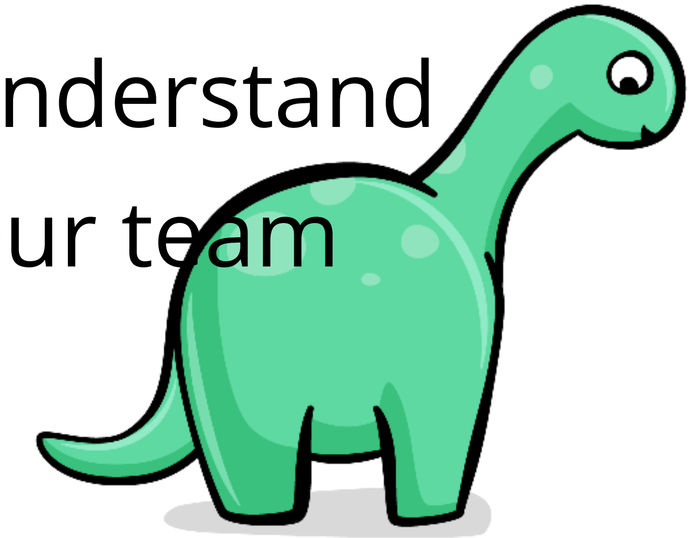
- Strong types prevent errors
- Transforms runtime crash into compile-time error
- Compare: URL vs String
- Add additional behaviour these “models”
- Offensively telling what validation is needed

Improving modeling

- Modelling in your code prevents excessive error handling, or guesswork ;]
- A model where every property is optional — when is it valid?
- A model that represents an API response — why is it mutable?
- The less code that can’t happen, the better
- Offensively displaying earlier validation

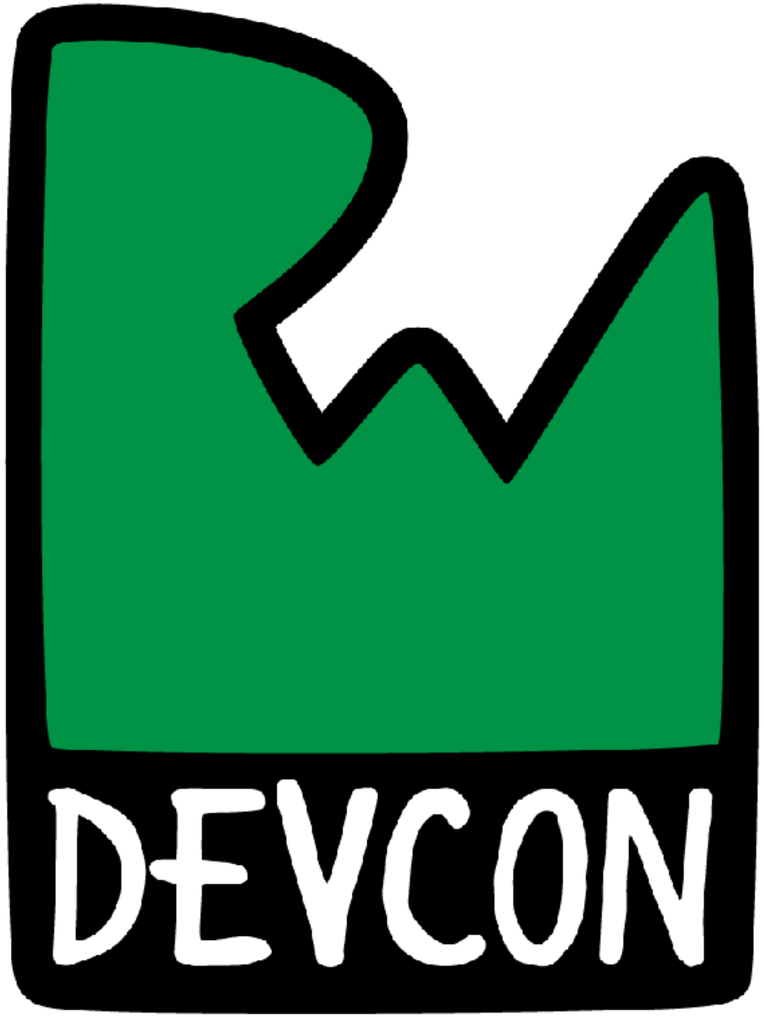
Removing tech debt

- Improving code is great, but remember to go over and clean up
- Technical debt or legacy code is code which doesn’t work, is inefficient, or invalid
- Tidier code means less to have to understand
- Offensively making less work for your team



Demo 3

Session 9:
Spring Cleaning Your App



Conclusion

What You Learned

- **Demo 1**: Refactoring to get to a better state
- **Demo 2**: Testing our code to give us safety
- **Demo 3**: Strengthening constraints in our code

What You Learned

- **Demo 1**: Refactoring to get to a better state
  - Extracting methods splits actions & results
  - Pushing logic into the Presenter
  - Make ViewController implement the View protocol
  - *Any architecture here is good!*

What You Learned

- **Demo 2**: Testing our code to give us safety
  - Be careful of unintended test actions!
  - Think of how strong you want your test to be

What You Learned

- **Demo 3**: Strengthening constraints in our code
  - The most flexible code is not always the best
  - Optionals can show “greedy objects”
  - Use optionals to your advantage

Where To Go From Here?

- Growing Object-Oriented Software — Steve Freeman and Nat Price
- Clean Code — Bob Martin
- Source Making: https://sourcemaking.com
- Twitter: @amlcurran