ERICH KADERKA @ PRIMEHAMMER

REFACTORING

WHAT IS REFACTORING?

- Martin Fowler: Refactoring is a controlled technique for improving the design of an existing code base. Its essence is applying a series of small behavior-preserving transformations, each of which "too small to be worth doing".
- Joshua Kerievsky: By **continuously** improving the design of code, we make it easier and easier to work with. This is in sharp contrast to what typically happens: little refactoring and a great deal of attention paid to expediently adding new features. If you get into the hygienic habit of refactoring continuously, you'll find that it is easier to extend and maintain code.
- Most required prerequisite for refactoring is having tests. If you don't have, write them before you start to refactor.

TOOLS

- Rubycritic - https://github.com/whitesmith/rubycritic
  Rubycritic wraps around static analysis gems such as Reek, Flay and Flog to provide a quality report of your Ruby code.
- Pronto - https://github.com/mmozuras/pronto
  Pronto runs analysis quickly by checking only the relevant changes. Created to be used on GitHub pull requests, but also works locally and integrates with GitLab and Bitbucket. Perfect if want to find out quickly if branch introduces changes that conform to your style guide, are DRY, don't introduce security holes and more.

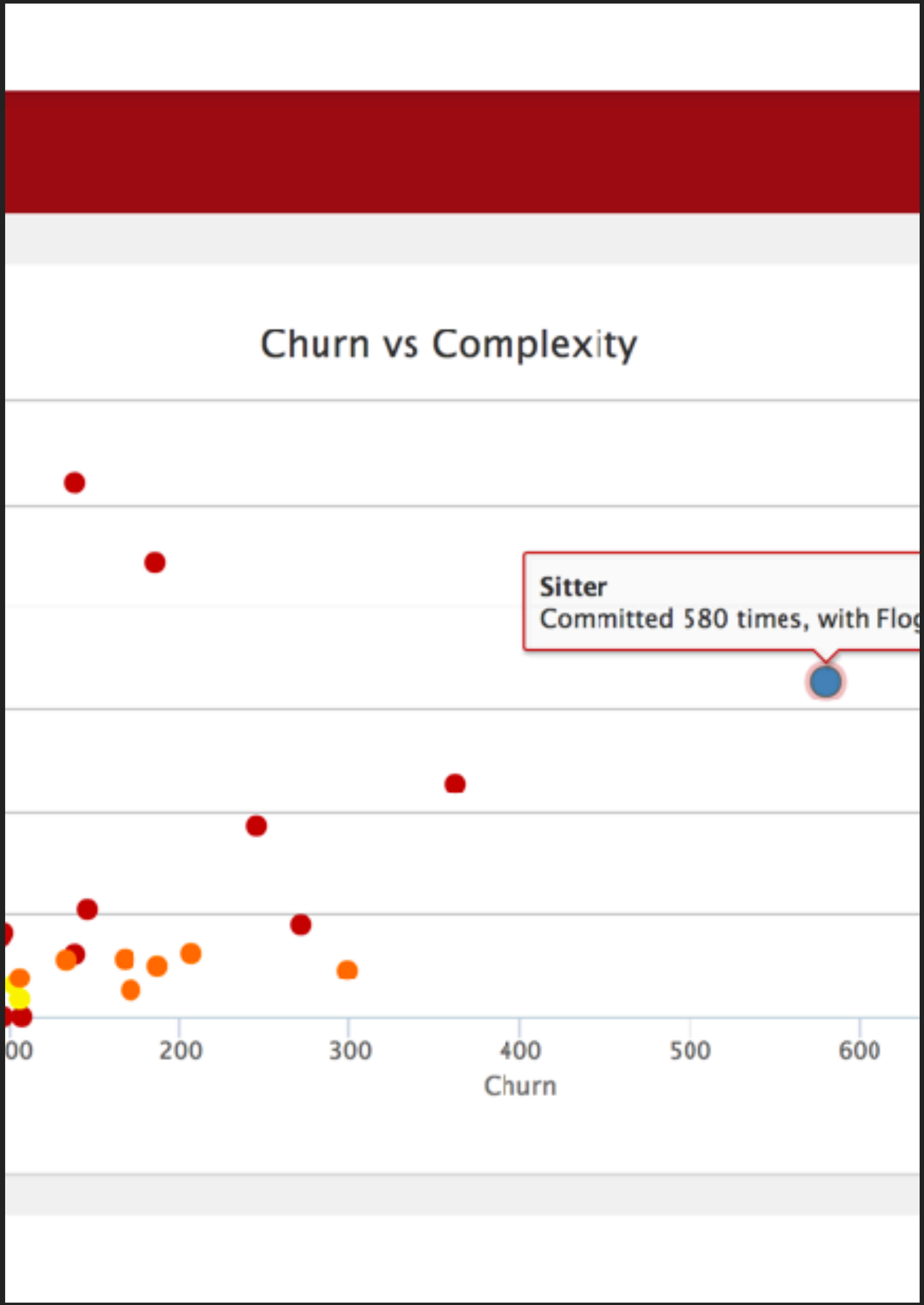
HOW TO USE THEM

- I recommend to use Pronto on daily basis. It has to be configured to get rid off of some annoying checks. But it's highly adaptive. From time to time it's not a bad idea to run rubycritic (especially after finishing a biggish feature) to see the churn table

- Churn chart is really important for one reason. It doesn't make sense to refactor code, that nobody touches. If it works and there are no changes, leave it alone :).

EXAMPLE

- Project hlidacky.cz
- Model Sitter
- More than 1800 lines
- High Complexity & Duplication
- Nearly 580 commits

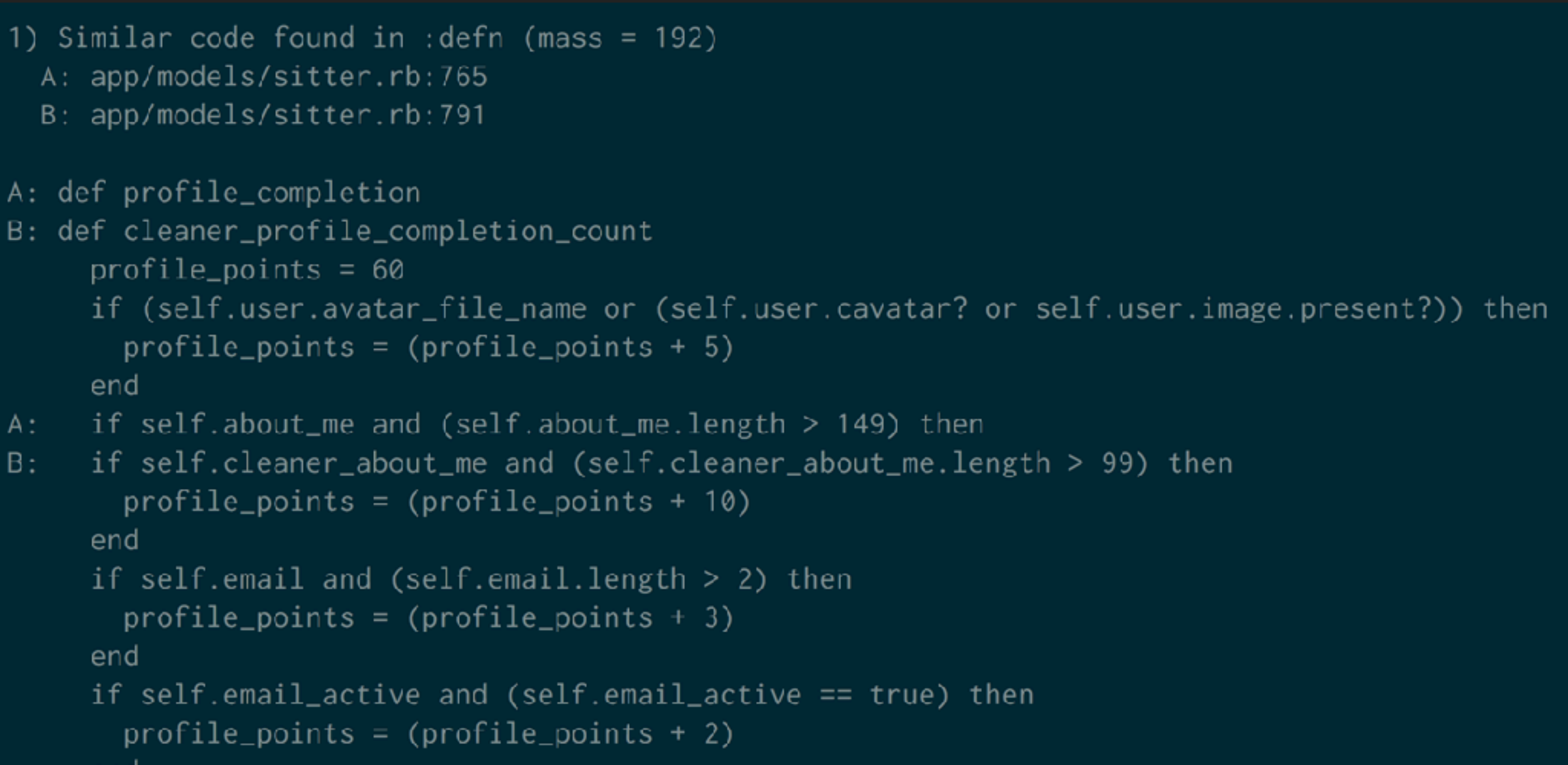


REEK

- Reek is a tool that examines Ruby classes, modules and methods and reports any Code Smells it finds. The list of smells isn't very long and there some great examples.
- e.g. UnusedPrivateMethod or UncommunicativeMethodName
- Personally I don’t agree with PrimaDonnaMethod. Btw it doesn't hurt to run this for test folder.

FLAY

- analyzes code for structural similarities (duplication)



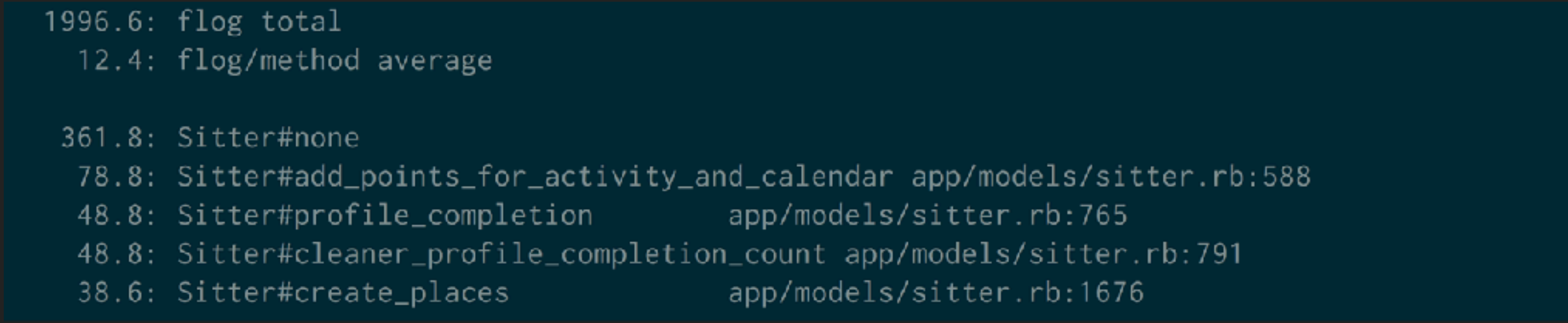
```
1) Similar code found in :defn (mass = 192)
  A: app/models/sitter.rb:765
  B: app/models/sitter.rb:791

A: def profile_completion
B: def cleaner_profile_completion_count
     profile_points = 60
     if (self.user.avatar_file_name or (self.user.cavatar? or self.user.image.present?)) then
       profile_points = (profile_points + 5)
     end
A:   if self.about_me and (self.about_me.length > 149) then
B:   if self.cleaner_about_me and (self.cleaner_about_me.length > 99) then
       profile_points = (profile_points + 10)
     end
     if self.email and (self.email.length > 2) then
       profile_points = (profile_points + 3)
     end
     if self.email_active and (self.email_active == true) then
       profile_points = (profile_points + 2)
```

FLOG

- deals with complexity. Reports the most tortured code in an easy to read pain report. The higher the score, the more pain the code is in. Flog parses it and builds up a structure of the code internally using RubyParser. Then it goes through every class and method until everything is scored. What you end up with is a breakdown of how each class and method scored.

FLOG OUTPUT



```
1996.6: flog total
  12.4: flog/method average

 361.8: Sitter#none
  78.8: Sitter#add_points_for_activity_and_calendar app/models/sitter.rb:588
  48.8: Sitter#profile_completion         app/models/sitter.rb:765
  48.8: Sitter#cleaner_profile_completion_count app/models/sitter.rb:791
  38.6: Sitter#create_places              app/models/sitter.rb:1676
```

- The solution was to extract points calculations to its own class. It wasn't too hard, you could see duplication and complexity in isolation so it was easier eliminate.

SHORT SUMMARY

- Main reason for this presentation - to learn more about gems which are used in rubycritic, pronto, code climate etc
- Flog - measures the complexity
- Flay - reports duplication
- Reek - code smell detector

WHEN TO REFACTOR? STABLE

- S - Smell your code. Study code smells and other don't's, so you can identify things you could fix. You want to start noticing and naming the problems in the code you look at. Even though you won't be fixing them all yet.
- T - Tiny problems first. In messy code, there will be lots of smells. The best way to get started is to pick a really small problem that you know how to solve in a very concrete way and just fix that. For example, rename variable. Your goal is to see the large problems better, by clearing away the small problems obscuring them.
- A: Augment your tests. You will almost certainly have to do this to be able to refactor large classes. You need integration tests one level higher than the class you are working on. If you want to refactor a big controller method, you need a feature spec. If you want to refactor a big model, you need controller-level tests. In general test behavior, not implementation. You need tests that describe the behavior you want to keep.

STABLE

- B: Back up. When the code has an abstraction in it that is no longer serving you well, sometimes the most useful thing to do is to ‘rewind’ the code into more procedural code, put all the duplication back, and start again. It is much more easier to move from procedure to the right set of objects than it is to move from the wrong set to the right set.
- L: Leave it better than you found it. During any one expedition into the code to add feature or fix a bug you won’t be able to fix all the problems you see. Sometimes the only thing you have time to do alongside your stated goal is a rename a method. The next person to come though this code will be able to understand it a little bit more easily. Maybe the big abstraction will suddenly be obvious.
- E: Expect good reasons. Assume past developers had good reasons to write the code they did. Some code looks so horrible that we think: “what idiot wrote this?” And then you run git blame and find out it was you. Start thinking about the social pressures that affect your codebase, and many things will make a lot more sense.

RESOURCES

- Sarah Mei - "Is Your Code Too SOLID?" - https://www.youtube.com/watch?v=WtpQD1cR3mA
- Bryan Helmkamp - Code Quality Lessons Learned - https://www.youtube.com/watch?v=vcH0RBe4Eew
- Github & Ruby toolbox