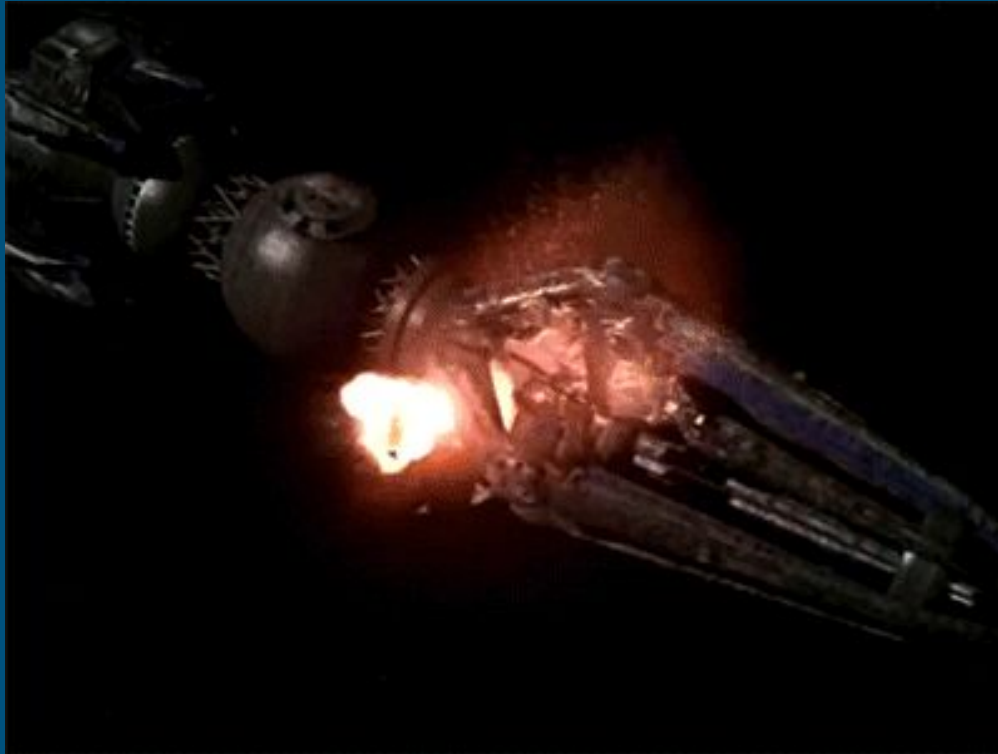




Live Long and Refactor 🖐️



Star Trek Voyager Season 4 “Year of Hell”



Gif of a ship in open space exploding

What is a refactor?

Martin Fowler, author of “Refactoring - Improving the Design of Existing Code”, defines a refactor as:

“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". However the cumulative effect of each of these transformations is quite significant”

When and why is a refactor
better than a complete
rewrite?

Why refactor and not just rebuild from scratch?

- Substantial risk with rewriting every feature, especially without a feature freeze
- Drastic changes could result in bad experience for paying users
- Cost is too high, especially for smaller teams

When should we do a large-scale refactor?

- When an application is built quickly in order to get to market, but now you have the time to make the codebase more maintainable
- When the current application is profitable and you can't risk losing paying users
- When users like the product but developers are scared to make changes and deploy

So how do I start?

First, stabilize the application

- **Reproducibility Checklist**

- Could you deploy your application across different environments?
- Is everything in version control?
- Are your secret keys and credentials in environment variables?
- Do you have database backups regularly generated?

Next, some basic tests

- Basic test scaffolding can give you some protection against unintended downtime
- These tests ensure your application is still up but not that individual features work



*Image of the Capitol Building
with construction scaffolding*

Third, identify high risk, high value elements

- Find the high risk, high value elements by tracing the bugs as far back as possible to the root cause
 - Don't focus on how the issues manifest
 - If it scares you, you're probably on the right track
- **Resource Tip:** The Mikado Method by Daniel Brolund and Ola Ellnestam
 - Great read on how to trace issues back to their root causes

Refactor Cycle Overview

1. Refactor - Write the least amount of code to add user value
2. Deploy - Ship new code
3. Validate - Human confirmation everything still works
4. Repeat

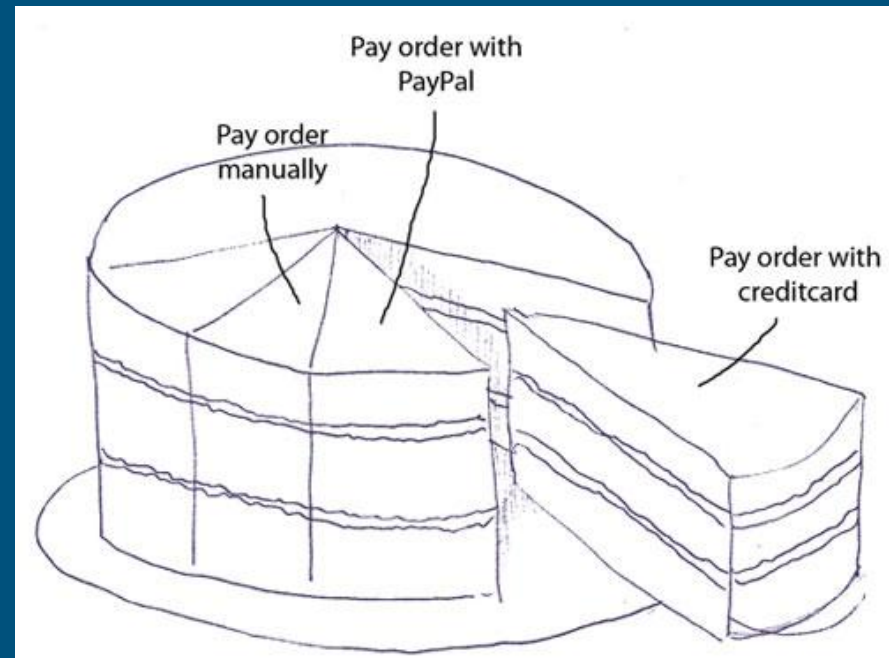




Image of a cake with slices titled, "Pay order with credit card", "Pay order with PayPal", and "Pay order manually" from the Agilistic Blog





Image of Captain Kirk and Spock smiling and eating cake

...even Spock can't say no to cake!


An example of vertical stories as tickets in Github



Complete 29  

 **Editorial Permissions: User with daily permission can only access daily** 

#2166


priority — high





 **Editorial Permissions: user with magazine permission can only access magazine** 

#2164


priority — high



 **Editorial Permissions: user with archive permission can only access archive content** 

#2163

priority — high



Diving into the Refactor phase

- Understand the feature and acceptance criteria (much easier with institutional knowledge)
- Write tests for the old system
- Write tests for the new system
- Write code for the new system
- **Resource tip** - Strangler Application pattern on Martin Fowler's blog



When your old code doesn't want to get strangled

Image of Captain Kirk fighting the Gorn

Writing clean, maintainable code

- Don't try to DRY up your code immediately
 - Drying up code can lead to tight coupling and difficulty testing
- Slow down and try to solve the problem at hand
 - This is the place we emphasize maintainability when we couldn't before
 - Find hidden classes and interfaces
- Hold the line on scope
 - It's easy to want to fix everything while you're at it but keep your eyes on the prize

Writing clean, maintainable code is hard

Resource tips:

- “Clean Code” by Bob Martin
- “Working with Legacy Code” by Michael Feathers
- “Refactoring - Improving the Design of Existing Code” by Martin Fowler



*“Uncle” Bob Martin
as Spock in Clean Code*

Recap

- If your application is profitable, has users, but is difficult for developers to maintain and work on, refactoring is better than rewriting
- Start the refactor by ensuring your application is reproducible and has some basic test scaffolding
- Identify the root cause of problems in the application

Recap

- Follow the refactor, deploy, and validate cycle to mitigate risk and and refactor to the user feature, not system
- Take your time and bring maintainability and quality to the code
- Maintain two versions of the functionality while you strangle the old code



Gif of Lieutenant Commander Uhura doing the Vulcan Salute