

# Refactoring at Scale



By Maude Lemaire

## Refactoring

Restructure existing code  
**WITHOUT** changing its external behavior



## At Scale

- One that affects a substantial surface area of your systems
- Involves typically large codebases

## Benefits

- Increase developer productivity
- Greater ease identifying bugs

## Risks

- Serious Regressions
- Unearthing Dormant Bugs
- Scope Creep



Shift in Product Requirements

Performance issues

Using a new Technology



Code Complexity  
Hinders Development

Small Scope

For Fun or Out of Boredom

Because You Happened  
to Be Passing By



When NOT ?

When You Don't Have Time

To Make Code More Extendable

## PLANNING



## MEASURE OUR STARTING STATE



### Measure Code Complexity

- Halstead metrics
- Cyclomatic Complexity
- NPath Complexity



### Test Coverage Metrics

- Quantitatively : proportion of code under test
- Qualitatively : suitable test quality has been attained



### Documentation

- Formal : everything you most likely think of as documentation
- Informal : Chat / email transcripts, Bug Tracking system, ...



### Version Control

- Commit messages : keywords for given code
- Commits in Agg : change frequencies, authorship



### Reputation

- Low-effort means of collecting reputation data
- Interview fellow developers



### Build a Complete Picture

Pick one metric from every category

## DRAFT A PLAN



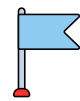
### Define your end state

Outline all starting metrics and target end metrics



### Map the shortest distance

- Open a blank document technique
- OR Gather a few coworkers



### Identify Strategic Intermediate Milestones

- 1) Does this step feel attainable in a reasonable period?
- 2) Is this step valuable on its own?
- 3) If something comes up, could we stop at this step and pick it back up easily later?

### Dark Mode / Light Mode

- Compare pre-refactor and post-refactor behavior :
- Both implementations are called
  - The results are compared

Dark

The results from the OLD implementation are RETURNED

Light

The results from the NEW implementation are RETURNED

### Choose a Rollout Strategy



How To ?

- Put in place an abstraction
- Enable dark mode
  - Monitor any differences between the 2 result sets
  - Track down and fix any potential bugs in the new implementation
- Enabling dark mode to broader groups of users
  - Continue logging any differences in the result sets
- Opt groups of users into light mode
  - Until everyone is successfully processing results from the new implementation
- Disable execution of both code paths
- Remove the old logic



### Clean Up Artifacts

- Feature Flags
- Dead Code
- Comments (TODOs)



### Reference Metrics

Include definitive progress metrics



### Share your plan

- Provide Transparency
- Gather perspective to strengthen it

*"No refactor is complete unless all remaining transitional artifacts are properly cleaned up"*

## GET BUY-IN

Always remember

Aren't Coding



See the Risk

Need to Coordinate

Managers

### Persuade Them

(some techniques)

Using Conversational Devices

Build an Alignment Sandwich



Rely on Evidence

Play Hardball

## BUILD THE RIGHT TEAM

### 2 Ways to Enlist Someone



#### Active Contributor

- Heavily involved from day one
- Actively contributing to the effort by writing code
- Consulted for input on the execution plan



#### Subject matter experts (SMEs)

- Agreed to be available to talk through solutions with you
- Answer questions
- Can do some code review



*"To execute on a large refactoring effort successfully, we need our own Ocean's 11 [...] a team just the right size with just the right skills"*

## EXECUTION



## COMMUNICATION

### Stand-Ups

Everyone aligned at regular intervals



### Weekly Syncs

- 1st part : accomplishments
- 2nd part : discuss any important topics

### Retrospectives

Reflect on the latest iteration cycle

Within Your Team

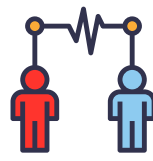
### When Kicking Off

#### Single Source of Truth

Choose a platform to collect all documentation

#### Set Expectations

Draft a communication plan



### During Project Execution

#### Announce Progress

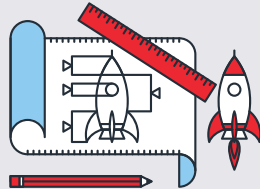
#### Execution Plan

Living Version

Outside Your Team

*"Policy of no laptops and minimal phone usage during meetings"*

## PROGRAM PRODUCTIVELY



### Prototype

#### Early and often

Help move faster

#### Know your solution won't be perfect

Not spend too much time perfecting the details

#### Be willing to throw code away



### Keep Things Small

- Commit small, incremental changes
- Makes it much easier to author great code



### Test, Test, Test

- Confirm everything has remained unaffected
- Or pinpoint the precise moment at which the behavior diverged



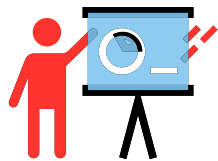
### Asking the "Stupid" Question

- Prioritize clarity
- Over maintaining an illusion of omniscience

## MAKE THE REFACTOR STICK

Foster Adoption through education

Integrate **Improvement** into the Culture



Active

Planning / leading workshops



Passive

- Step-by-step tutorials
- Online courses, ...



### To maintain a healthy codebase

- Continuous small refactoring
- Incrementally improve areas of the codebase

### Hold design reviews

- Early in the feature development process

### Encourage design conversations



### Case Studies @Slack

- Redundant Database Schemas
- Migrating to a New Database

