Patterns of Refactoring

CODESAI

Patterns of refactoring

Meta refactoring techniques

- Techniques that allow you to refactor in small steps while remaining always in green.
- Present in the mechanics of most refactorings.
- Essential for long refactorings.

Honourable Retreat

- If things go wrong, don’t stay too long looking for the error. Just go back to the last commit in green.
- Prerequisite: Commit often, and only in green.

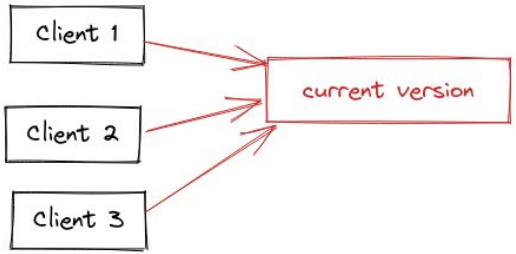
Narrowed Change

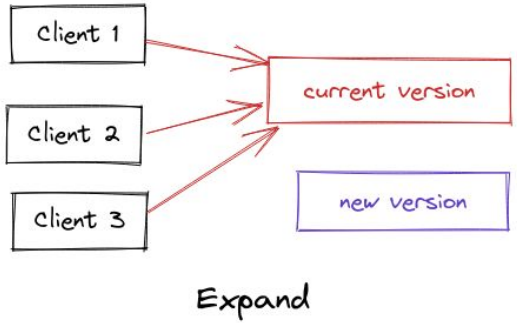
- Narrow the change down to a small number of change points.
- When you need to do a breaking change that will affect many clients (a change affecting a wide code surface).
  1. Extract methods to remove duplication and isolate the changes in only a few places (hopefully only one).
  2. Then you'll be able to make the changes quickly, minimizing the time in red.
- It's also useful for not breaking changes.

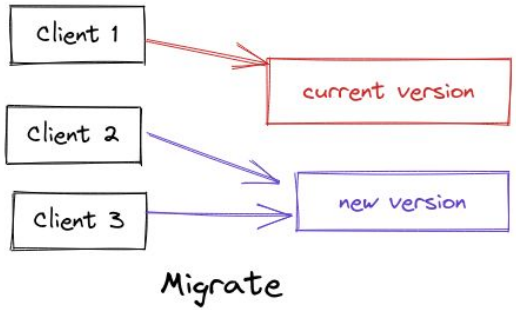
Parallel Change

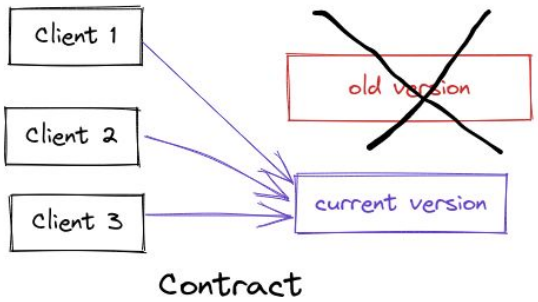
- A pattern to implement backward-incompatible changes to an interface in a safe manner, by breaking the change into three distinct phases:
  1. **Expand**: augment the code to support the old and new versions.
  2. **Migrate**: update all clients using the old version to use the new version.
  3. **Contract**: once the migration ends, remove the old version of the code. Don’t forget this phase!!

Parallel Change









Migrate phase details

- During the *migrate* phase, the old version and the new one will coexist for some time. Some parts of the code may be using the old design and other parts may be using the refactored. This is the longest phase.
- A variant implementation: implement the old method in terms of the new API and use inline method to update all usages at once.
  - A way to break the *migrate* phase into smaller and safer steps, allowing to change the internal implementation before changing the API that clients see.
  - Useful to avoid maintaining two separate implementations when the migrate phase is long.

Long Refactorings & CD

- Present in the mechanics of most refactorings.
- **Essential for long term refactorings**.
  - You can ship to production having only partially refactored the code.
  - This type of refactoring can be done without affecting the release schedule of other development work.
  - Ideal for continuous delivery.

Applications beyond refactoring

- ***Refactoring***: when changing a method or function signature, especially when doing a long term refactoring or when changing a published interface.
- ***Database refactoring***: this is a key component to evolutionary database design.
- ***Deployments***: deployment techniques such as canary releases and blue green deployment are applications of the parallel change pattern.
- ***Remote API evolution***: can be used to evolve a remote API when you can't make the change in a backwards compatible manner. This is an alternative to using an explicit version in the exposed API.

Parallel Change: to learn more

- 🎥 The Limited Red Society
- About The Limited Red Society
- Parallel Change
- Branch by Abstraction
- 🎥 Modificando el software de manera progresiva (slides)