

# Refactoring to Patterns

par

Date de publication : 16/05/2006

Dernière mise à jour : 28/08/2006

Critique du livre [Refactoring to Patterns](#) de *Joshua Kerievsky*

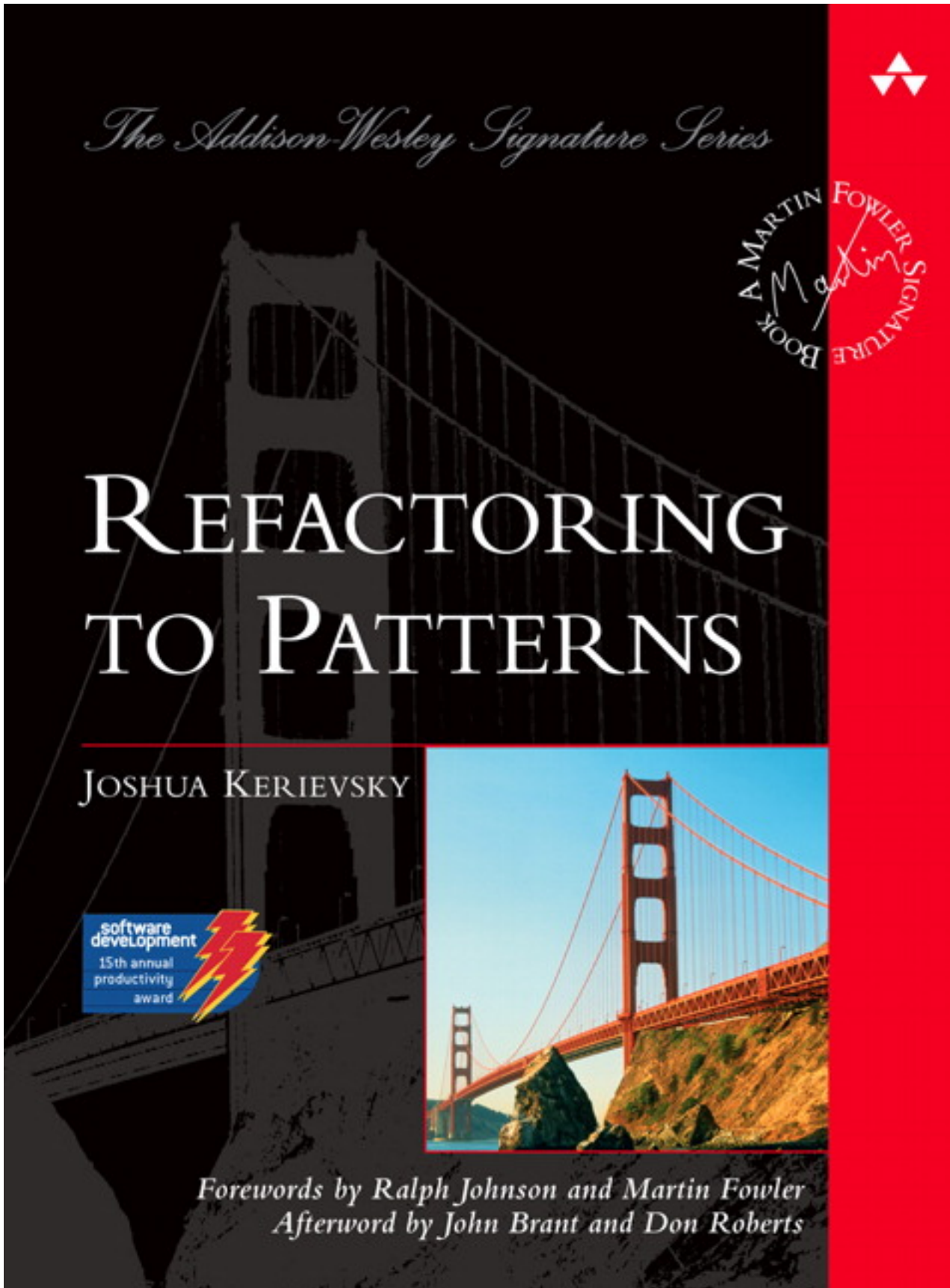
I - Description

II - Table des matières

III - Critique : A appliquer d'urgence partout !

IV - Liens annexes

## I - Description



Ce livre traite du mariage du refactoring - le processus d'amélioration du design du code existant - avec des patterns, les solutions classiques aux problèmes récurrents de design. Refactoring to Patterns suggère que l'utilisation des patterns pour améliorer un design existant est meilleur que l'utilisation des patterns tôt dans un nouveau design. Ceci est vrai, que le code soit jeune ou vieux. Nous améliorons les designs avec des patterns en appliquant des séquences de transformations de design de bas niveau, aussi connu sous le nom de refactorings.

## II - Table des matières

- 1. Why I Wrote This Book.
  - Over-Engineering.
  - The Patterns Panacea.
  - Under-Engineering.
  - Test-Driven Development and Continuous Refactoring.
  - Refactoring and Patterns.
  - Evolutionary Design.
- 2. Refactoring.
  - What Is Refactoring?
  - What Motivates Us to Refactor?
  - Many Eyes.
  - Human-Readable Code.
  - Keeping It Clean.
  - Small Steps.
  - Design Debt.
  - Evolving a New Architecture.
  - Composite and Test-Driven Refactorings.
  - The Benefits of Composite Refactorings.
  - Refactoring Tools.
- 3. Patterns.
  - What Is a Pattern?
  - Patterns Happy.
  - There Are Many Ways to Implement a Pattern.
  - Refactoring to, towards, and away from Patterns.
  - Do Patterns Make Code More Complex?
  - Pattern Knowledge.
  - Up-Front Design with Patterns.
- 4. Code Smells.
  - Duplicated Code.
  - Long Method.
  - Conditional Complexity.
  - Primitive Obsession.
  - Indecent Exposure.
  - Solution Sprawl.
  - Alternative Classes with Different Interfaces.
  - Lazy Class.
  - Large Class.
  - Switch Statements.
  - Combinatorial Explosion.
  - Oddball Solution.

- 5. A Catalog of Refactorings to Patterns.
  - Format of the Refactorings.
  - Projects Referenced in This Catalog.
  - A Starting Point.
  - A Study Sequence.
- 6. Creation.
  - Replace Constructors with Creation Methods.
  - Move Creation Knowledge to Factory.
  - Encapsulate Classes with Factory.
  - Introduce Polymorphic Creation with Factory Method.
  - Encapsulate Composite with Builder.
  - Inline Singleton.
- 7. Simplification.
  - Compose Method.
  - Replace Conditional Logic with Strategy.
  - Move Embellishment to Decorator.
  - Replace State-Altering Conditionals with State 166
  - Replace Implicit Tree with Composite.
  - Replace Conditional Dispatcher with Command.
- 8. Generalization.
  - Form Template Method.
  - Extract Composite.
  - Replace One/Many Distinctions with Composite.
  - Replace Hard-Coded Notifications with Observer.
  - Unify Interfaces with Adapter.
  - Extract Adapter.
  - Replace Implicit Language with Interpreter.
- 9. Protection.
  - Replace Type Code with Class.
  - Limit Instantiation with Singleton.
- 10. Accumulation.
  - Move Accumulation to Collecting Parameter.
  - Move Accumulation to Visitor.
- 11. Utilities.
  - Chain Constructors.
  - Unify Interfaces.
  - Extract Parameter.

### III - Critique : A appliquer d'urgence partout !

On parle Java dans tout le livre, mais chaque conseil est pour tous les langages. De plus, le framework a été porté pour fonctionner avec presque tous les langages. L'approche de *Kent Beck* est agréable et didactique. On construit vraiment au fur et à mesure, petit pas par petit pas. Le premier exemple est simple, gérer de l'argent dans différentes devises, et ce petit exemple nous mène déjà loin.

En fait, les exemples servent juste à bien nous asséner le principe de *Beck*, tester en construisant progressivement. Si on a compris qu'on doit toujours tout tester et qu'aucun code ne doit être buggé quand on arrête le travail, on a compris le livre. Ensuite, l'auteur nous donne des astuces, des *patterns* pour avancer, comme écrire des tests isolés, ne dépendant pas de toute l'architecture afin de ne pas déprimer - important... -, faire des raisonnements inductifs pour généraliser une réponse, ...

Depuis la lecture de ce livre, je me suis mis aux tests. J'avoue qu'avant, c'était la dernière de mes préoccupations. Maintenant, mon code est plus propre - on dirait une pub de lessive, là -, et surtout bardé de tests au cas où j'aurai une modification au niveau de l'architecture à faire - refactoring, et ça arrive souvent -. En même temps, je n'applique pas sa solution à 100%, c'est tout de même très difficile, ça demande du temps d'écrire le test avant d'avoir la solution qui va avec, mais on voit aussi comment on avance, c'est bien ;)



## IV - Liens annexes

 ***Critique sur la page de livres Conception***

 ***Achat sur Amazon.fr***

 ***Lien vers le site de l'éditeur***

