

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/220298907>

Refactoring to Patterns.

Article in *Journal of Object Technology* · January 2005

DOI: 10.5381/jot.2005.4.4.r2 · Source: DBLP

CITATIONS

0

READS

7,232

2 authors, including:



Thomas Kannampallil

Washington University in St. Louis

107 PUBLICATIONS 1,296 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Cognitive Complexity and Error in Critical Care [View project](#)



Medical Reasoning [View project](#)

Book Review

Refactoring to Patterns

By Joshua Kerievsky, Addison-Wesley Professional, Reading, MA, 2004. 367 pp., \$49.99 (hardbound). ISBN 0-321-21335-1.

Reviewed by John M. Daughtry III and Thomas George Kannampallil
School of Information Sciences & Technology, The Pennsylvania State University

Refactoring to Patterns is an interesting merger of two of the most important concepts in software engineering that have arisen in the last ten years. However, it does not belong on every software engineer's bookshelf. For those who would expect it to be a comprehensive reference tool, the text will be disappointing. However, for those with the ability, time, and desire to extend the ideas presented, it will be a very useful read. The existing pattern literature provides us with the what, the when, the why, and the where of a pattern's use, but not the how. The major contribution of Refactoring to Patterns is that it provides a solid explanatory framework for describing how one can introduce and remove patterns from code. Just as patterns and refactorings were introduced in their respective texts, Refactoring to Patterns does not purport to be a complete volume, but a starting point. To Kerievsky's credit, he does acknowledge that the book is a work in progress that is being published now because of its usefulness, as opposed to its completeness. While some books fulfill their mission as a reference, Refactoring to Patterns should be read through before serving as a tool. It is, in many ways, a series of case studies, as opposed to a catalog of refactorings.

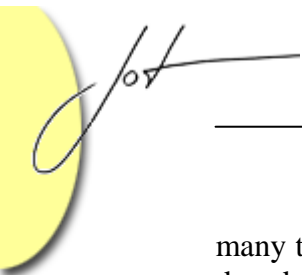
1 BACKGROUND - 1999, 1994, & 2004

Martin Fowler wrote Refactoring [2], which has since grown to become an integral part of industrial practice. In this work, Fowler succeeded in formalizing the art of just-in-time redesign. In essence, a refactoring is a formalized change in code, such as extracting a code fragment into its own method for clarity and reuse. With the advent of agile development methodologies, where just-in-time redesign is commonplace, refactoring has taken on an increasing importance in software engineering.

Four years prior to the publication of Refactoring, the Gang of Four (GoF) published Design Patterns [3]. It took the software engineering world by storm and is argued by

Cite this article as follows: John M. Daughtry III, Thomas George Kannampallil: review "Refactoring to Patterns", in Journal of Object Technology, vol. 4, no. 4, May-June 2005, pp 193-196.

<http://www.jot.fm/books/review17>



many to be the single most important work in the field of software engineering in the last decade. In Design Patterns, the GoF described common solutions to common problems in object-oriented software design by adopting the pattern paradigm from the field of Architecture [1].

While Refactoring is a book on the practice of design, Design Patterns is a book on the design itself. Until now, no direct connection has been made between the two except for the fact that the two communities overlap significantly. Kerievsky wrote Refactoring to Patterns with the intention of its being the link between the two areas. A professional programmer since 1987, he is an active member of both communities.

2 FRAMING THE CONTENT

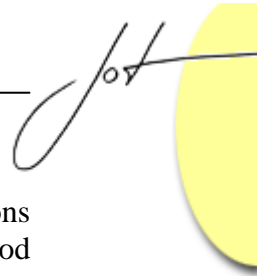
Kerievsky makes five arguments in Chapter One that are extremely important to keep in mind while reading, as they help the reader understand the intentions of the author and frame the concepts. First, software engineers are inclined to over-architect code just in case the system has to grow to meet other needs. Second, patterns tend to be over-used when simpler solutions would work just as well. Third, under-engineered code is far too common an occurrence because of various external forces. Fourth, test-driven development, along with continuous refactoring, allows for under-architected code to be altered with confidence. And finally, studying the evolution of a design or system provides more knowledge and insights than studying the final artifacts alone. If these core concepts are not understood, Refactoring to Patterns cannot be appreciated fully.

3 ORGANIZATION

The majority of the text consists of refactoring examples where specific code examples are refactored into or out of patterns. There are six sections, grouping the refactoring by motivation:

1. Creation – six refactorings that address the creation of objects
2. Simplification – six refactorings that simplify code
3. Generalization – seven refactorings that aid in abstracting code
4. Protection – three refactorings that improve the protection of existing code
5. Accumulation – two refactorings dealing with code that accumulates information
6. Utilities – three low-level refactorings that support other refactorings in the text

As indicated above, within each section there is a handful of refactorings. These refactorings are named and summarized. In addition, the benefits and liabilities of using each refactoring are enumerated, and the step-by-step instructions are provided. Examples are given for each refactoring that conveniently use the same notation found in Refactoring. These examples are mostly from one of three projects, namely, an XML



Builder, HTML Parser, or a Loan Risk Calculator. Some of the refactoring descriptions also contain variations. The code examples are easy to follow and backed up by good UML diagrams. Most exciting, however, is that they are real-world examples with some level of complexity.

4 PRAISE AND PROBLEMS

Overall, Kerievsky did an excellent job in presenting the material. The organization by motivation is useful, and the content of each refactoring description is complete. Particularly surprising, however, was the quality of the earlier parts of the book that do not relate to the refactorings themselves. The forwards introduce the importance of the text and the credibility of the author with authority. The preface and first chapter explain in detail the views and background of the author and his motivation for writing the book. Chapters Two and Four give an outstanding abbreviated overview of refactoring and code smells for those who need a refresher.

Two problems exist with the text. First, there are very few comments in the example code. Granted, the examples are simple; however, there are some cases where more comments would have been appropriate, particularly in a book that will no doubt be read by young programmers from a wide range of cultures who may or may not be familiar with loan risk or blackjack.

Second, the work is incomplete. It is a convincing work, but there is not enough content to justify its being published as a full book in its current state. In fact, the prior incarnation of this book is a technical report Kerievsky wrote in 2001 that bears the same name. The book could have been published online as a more complete version of the original technical report without detracting from its usefulness, and publication could have been deferred until it was more complete. Kerievsky argues that refactoring into, towards, and away from patterns is a natural occurrence. However, instead of offering 57 refactorings (three each for the 19 patterns), he offers 27. Interestingly, there is no single pattern that offers each type of refactoring (into, towards, and away from the pattern), even though the claim is that they are natural occurrences.

5 CONCLUSION

The Martin Fowler Signature Series emblem on the cover means that Martin Fowler hand-selected it as a book he wished he could have written, which is no surprise. Refactoring to Patterns is on a subject that has tremendous potential, and this book is an outstanding beginning that should be read as a starting point rather than being used as a reference. Even the forwards provide useful insights for the software engineer, which is not very common. Although it can be read on its own, reading and understanding both Design Patterns and Refactoring is requisite to get a mature view of this new text.



REFERENCES

- [1] Christopher Alexander: *A Timeless Way of Building*, Oxford University Press, 1979.
- [2] Martin Fowler: *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 1999.
- [3] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.