

Basic of Refactoring

Refactoring is a systematic process of improving code without creating new functionality that can transform a mess into clean code and simple design

Checklist

- The code should become cleaner
- New functionality shouldn't be created during refactoring
- All existing tests must pass (Write BDD style tests)

Automatic unit tests should be set up before refactoring to ensure methods still behave as expected. (Also check BDD testing). Refactoring is then an iterative cycle of making a small program transformation, testing it to ensure correctness and making another small transformation

Techniques (see Martin Fowler refactoring book)

For more understanding what is going on:

- Program Dependence Graph - explicit representation of data and control dependencies
- System Dependence Graph - representation of procedure calls
- Software intelligence - reverse engineers the initial state to understand existing
- UML from source code

For more abstraction

- Encapsulate field
- Generalize type
- Replace type-checking code with state/strategy (Replace type-checking code with State/Strategy)
- replace conditional with polymorphism (Replace conditional with polymorphism)

For breaking code apart

- Componentization breaks code
- Extract class moves part of the code from an existing class into a new class
- Extract method, to turn part of a larger method into a new method.

Improve naming

Automatic clone detection softwares (Duplicate code: https://en.wikipedia.org/wiki/Duplicate_code)

Although refactoring code has been done informally for decades, William Griswold's 1991 Ph.D. dissertation[23] is one of the first major academic works on refactoring functional and procedural programs, followed by William Opdyke's 1992 dissertation[24] on the refactoring of object-oriented programs,[25] although all the theory and machinery have long been available as program transformation systems

Books: <http://cseweb.ucsd.edu/~wgg/Abstracts/gristhesis.pdf>

<https://web.archive.org/web/20191216212919/https://dl.acm.org/citation.cfm?id=169783>

<https://dl.acm.org/doi/book/10.5555/169783>

Martin Fowler's book *Refactoring: Improving the Design of Existing Code*

Many software editors and IDEs have automated refactoring support. It is possible to refactor application code as well as test code.^[29] Here is a list of a few of these editors, or so-called refactoring browsers.

- [DMS Software Reengineering Toolkit](#) (Implements large-scale refactoring for C, C++, C#, COBOL, Java, PHP and other languages)
- Eclipse based:
 - [Eclipse](#) (for Java, and to a lesser extent, C++, PHP, Ruby and JavaScript)
 - [PyDev](#) (for Python)
 - [Photran](#) (a Fortran plugin for the Eclipse IDE)
- [Embarcadero Delphi](#)
- IntelliJ based:
 - [Resharper](#) (for C#)
 - [AppCode](#) (for Objective-C, C and C++)
 - [IntelliJ IDEA](#) (for Java)
 - [PyCharm](#) (for Python)
 - [WebStorm](#) (for JavaScript)
 - [Android Studio](#) (for Java)
- [JDeveloper](#) (for Java)
- [NetBeans](#) (for Java)
- [Smalltalk](#): Most dialects include powerful refactoring tools. Many use the original refactoring browser produced in the early '90s by [Ralph Johnson](#).
- Visual Studio based:
 - [Visual Studio](#) (for .NET and C++)
 - [CodeRush](#) (addon for Visual Studio)
 - [Visual Assist](#) (addon for Visual Studio with refactoring support for C# and C++)
- [Wing IDE](#) (for Python)
- [Xcode](#) (for C, Objective-C, and Swift)^[30]
- [Qt Creator](#) (for C++, Objective-C and QML)^[31]

Refactoring browsers (Automated refactoring support)

List of tools for static code analysis

https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

Code smells

Bloaters are code, methods and classes that have increased to such gargantuan proportions that they are hard to work with.

- Long Method
- Large Class
- Primitve Obsession
- Long parameter list
- Data clumps (different parts of the code contain identical groups of variables)

Object Orientation Abusers

- Alternative Classes with different Interface (Two classes perform identical functions but have different method name)
- Refused Bequest (If a subclass uses only some of the methods and properties inherited from its parents, the hierarchy is off-kilter)
- Switch Statements (If a switch is based on type code, such as when the program's runtime

mode is switched, use Replace Type Code with Subclasses or Replace Type Code with State/Strategy.)

- Temporary Field (Temporary fields and all code operating on them can be put in a separate class via Extract Class, introduce Null Object)

Change Preventers (if you need to change something in one place in your code, you have to make many changes in other places too)

- Divergent Change (You find yourself having to change many unrelated methods when you make changes to a class, introduce new class)
- Parallel Inheritance Hierarchies
- Shotgun Surgery (Making any modifications requires that you make many small changes to many different classes, move methods, files, introduce new class)

Dispensables (A dispensable is something pointless and unneeded whose absence would make the code cleaner, more efficient and easier to understand)

- Comments
- Duplicate Code
- Dead Code (if code is not longer used)
- Lazy Class (too small class or only specific functionality used in one place, consider inline class)
- Speculative Generality (programming for future, you end up with class and methods that you do not use)

Couplers

- Feature Envy (method accesses that of another obj more than its own data, move method)
- Inappropriate Intimacy (class uses private fields and methods of another class)
- Incomplete Library Class (use existing library instead of creating your own)
- Message Chains (`$a->b()->c()->d()` , use Hide Delegate)
- Middle Man (If a class performs only one action, delegating work to another class, why does it exist at all)

also

https://en.wikipedia.org/wiki/Data_deduplication

https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

https://en.wikipedia.org/wiki/Redundant_code

[https://en.wikipedia.org/wiki/Rule_of_three_\(computer_programming\)](https://en.wikipedia.org/wiki/Rule_of_three_(computer_programming))

Program_transformation

TODO https://en.wikipedia.org/wiki/Program_transformation

Anti paterns

<https://en.wikipedia.org/wiki/Anti-pattern>

From

-> <https://refactoring.guru/refactoring>

-> https://en.wikipedia.org/wiki/Code_refactoring

-> <https://refactoring.com/> STAO, knjige procitaj

-> <https://refactoring.guru/store> check this hmm

-> <https://sourcemaking.com>

-> <https://www.sciencedirect.com/topics/computer-science/refactoring>

[https://www.agilealliance.org/glossary/refactoring/#q=~\(infinite~false~filters~\(postType~\(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video\)~tags~\(~'refactoring\)\)~searchTerm~'~sort~false~sortDirection~'asc~page~1\)](https://www.agilealliance.org/glossary/refactoring/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'refactoring))~searchTerm~'~sort~false~sortDirection~'asc~page~1))

-> <https://wiki.c2.com/?WhatIsRefactoring>