

# Refactoring

## DIKU — Software Development

`<oleks@oleks.info>`

May 17, 2016

# Factoring

The breaking up of a complex task into simple tasks  
which are easier to grasp.

- ▶ Also called decomposition, or “divide and conquer”;
- ▶ A natural part of software development;
- ▶ Which we often get wrong on first try.

# Refactoring

Recalibration of the decomposition  
to improve software quality.

# When?

- ▶ While programming.
- ▶ Once you get something working.
- ▶ When you find a bug.
- ▶ When adding features becomes a hurdle.

But prioritize automated testing.

# Purpose

- ▶ Reflect new domain understanding.
- ▶ Reduce technical debt:
  - ▶ cost of software maintenance,
  - ▶ cost of adding new features.
- ▶ Increase reusability.

All this without changing program behaviour.

# Comments

*Comment your source code so it is easy to understand.*

— OSM 2016, 2015, 2014, 2013, ...

# Comments

*Comment your source code so it is easy to understand.*

— OSM 2016, 2015, 2014, 2013, ...

*Comment non-trivial parts of your code.*

— Advanced Programming 2015, 2014, 2013, ...

- ▶ A good programmer comments non-obvious code.

# Comments

*Comment your source code so it is easy to understand.*

— OSM 2016, 2015, 2014, 2013, ...

*Comment non-trivial parts of your code.*

— Advanced Programming 2015, 2014, 2013, ...

- ▶ A good programmer comments non-obvious code.

👍 Refactor to avoid the need for comments. 👍

What a variable, method, class, module, etc. does should be immediately clear from its *name*, *parameters*, and *context*.



# Cryptic Names

👍 (Re)name well. 👍

- ▶ Bad names: ...temp..., ...helper..., ...aux...
- ▶ Good names: toFloat, name, id, sum.
- ▶ May be good names: x, xs, n, fst, snd, ndx.
- ▶ When in Rome, do as the Romans do.
  - ▶ Don't use uncommon abbreviations.
  - ▶ Embrace the common ones.

# Do One Thing Well

A method should do one thing well.

A class should have one reason to change.

# Deep Levels of Indentation

*The answer to that is that if you need more than 3 levels of indentation, you're screwed anyway, and should fix your program.*

— Linux kernel coding style

👍 Keep the indentation level low. 👍

# Long Parameter Lists

- ▶ Indicates that there is too much going on in a method.
- ▶ It has a complicated API that is easy to get wrong.

👍 Group parameters into objects/structs. 👍

# Long Methods

- ▶ Long methods are hard to wrap your head around.
- ▶ A method longer than 10–20 lines is considered long.
- ▶ This might have to do with limits of working memory<sup>1</sup>.

👍 Split a long method into several. 👍

---

<sup>1</sup>Miller, G. A. (1956). *The magical number seven, plus or minus two: Some limits on our capacity for processing information*. Psychological Review 63 (2), pp. 81–97.

# Reduce Context

- ▶ A global variable is modifiable by every method.
- ▶ An instance variable is modifiable by every instance method.
- ▶ A public instance variable is even worse.

👍 Reduce the number of global and instance variables. 👍

👍 Program functional. 👍

# Takeaway

What a variable, method, class, module, etc. does should be immediately clear from its *name*, *parameters*, and *context*.

- ▶ Pick good names,
- ▶ have few parameters, and
- ▶ keep a small context.

The method algorithm should be clear at a glance.

- ▶ Keep the methods short (algorithmic), and
- ▶ keep the indentation level low.

# Tool Support

- ▶ Visual Studio

- ▶ ReSharper [https://www.jetbrains.com/resharper/features/code\\_refactoring.html](https://www.jetbrains.com/resharper/features/code_refactoring.html).
- ▶ Visual F# Power Tools <https://fsprojects.github.io/VisualFSharpPowerTools/>

- ▶ Eclipse

- ▶ [http://help.eclipse.org/mars/topic/org.eclipse.jdt.doc.user/concepts/concept-refactoring.htm?cp=1\\_4\\_4](http://help.eclipse.org/mars/topic/org.eclipse.jdt.doc.user/concepts/concept-refactoring.htm?cp=1_4_4)

- ▶ XCode

- ▶ Edit → Refactor

- ▶ Unix-like programming environment

- ▶ grep, perl, sed, vim, emacs, ...



# Reading Material

## Academic:

- ▶ Tom Mens and Tom Tourwé. 2004. *A Survey of Software Refactoring*. IEEE Transactions on Software Engineering 30 (2), 2004, 126–139.

## Web:

- ▶ Shvets Group, et al. <https://refactoring.guru>
  - ▶ <https://refactoring.guru/catalog/>.
- ▶ Martin Fowler, <http://refactoring.com/>
  - ▶ <http://refactoring.com/catalog/>.

## Light reading:

- ▶ oleks & br0ns. *Unix-Like Data Processing Utilities*. 2015.  
<http://atu15.onlineta.org/unix-like-data-processing.pdf>

# Video Material

- ▶ Martin Fowler. *Workflows of Refactoring*. OOP2014. <https://youtu.be/vqEg37e4Mkw>.
- ▶ Ben Orenstein. *Refactoring from Good to Great*. Aloha Ruby Conf 2012. <https://youtu.be/DC-pQPq0acs>
- ▶ Joshua Bloch. *How To Design A Good API and Why It Matters*. Google Tech Talks 2007. <https://youtu.be/aAb7hSCtvGw>.
- ▶ Robert C. Martin. *SOLID Principles of Object Oriented & Agile Design*. Yale, 2014. <https://youtu.be/QHnLmvDxGTY>.

# Summer Reading

- ▶ Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.