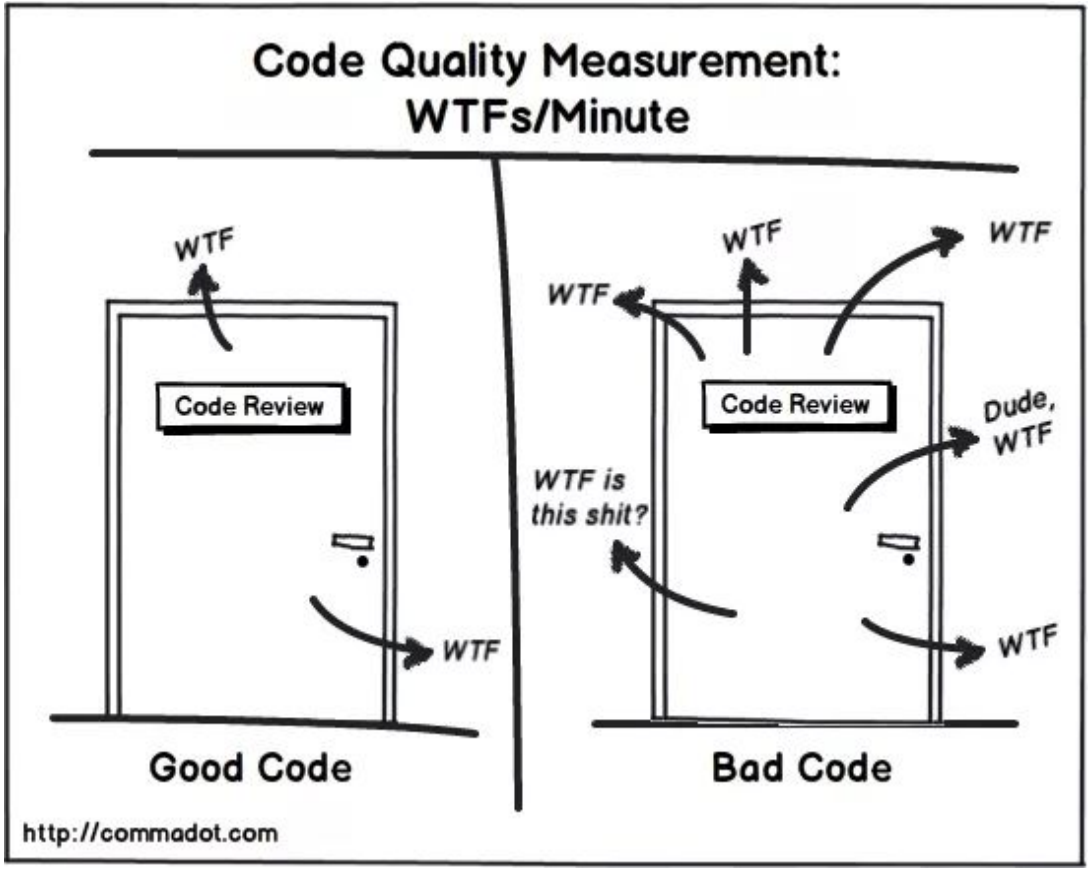
Refactoring

Yet another thing you shouldn’t trust your IDE to do

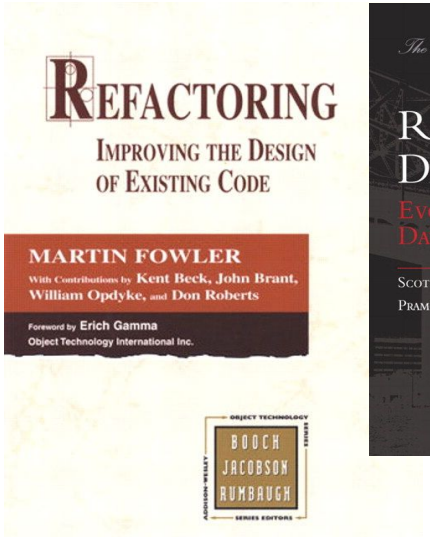
Refactoring

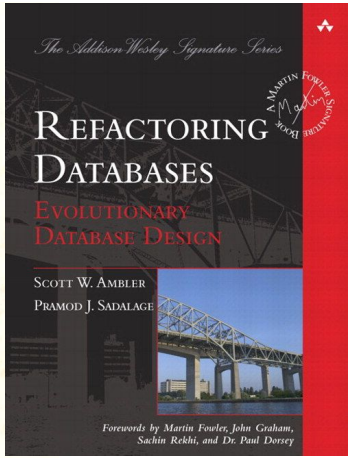
- What is it?
- How to do it?
- What can go wrong?
- How to implement a refactoring tool

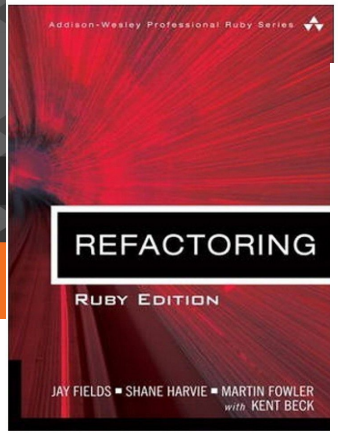

Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
WTF
Code Review
Dude,
WTF
WTF is
this shit?
WTF
Bad Code
http://commadot.com

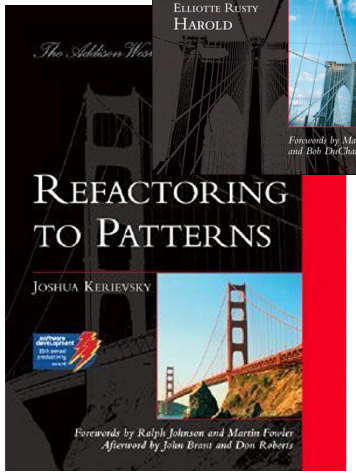
Refactoring: what is it?

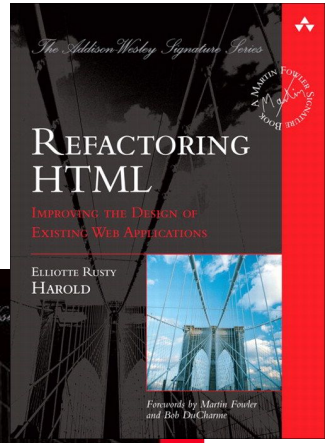
The process of restructuring existing computer code without changing its external behavior. (https://en.wikipedia.org/wiki/Code_refactoring, Martin Fowler, Opdyke)











Refactoring: what is it? - examples

Rename Local Variable

Rename Method

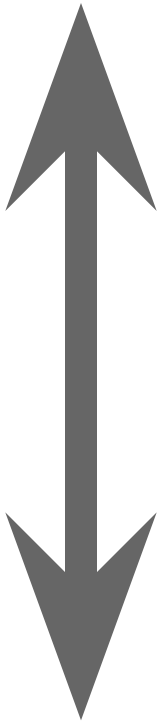
Rename Class

Extract Variable

Extract Method

Move Method

Replace inheritance with delegation



Low level

High level

Refactoring: how to do it?

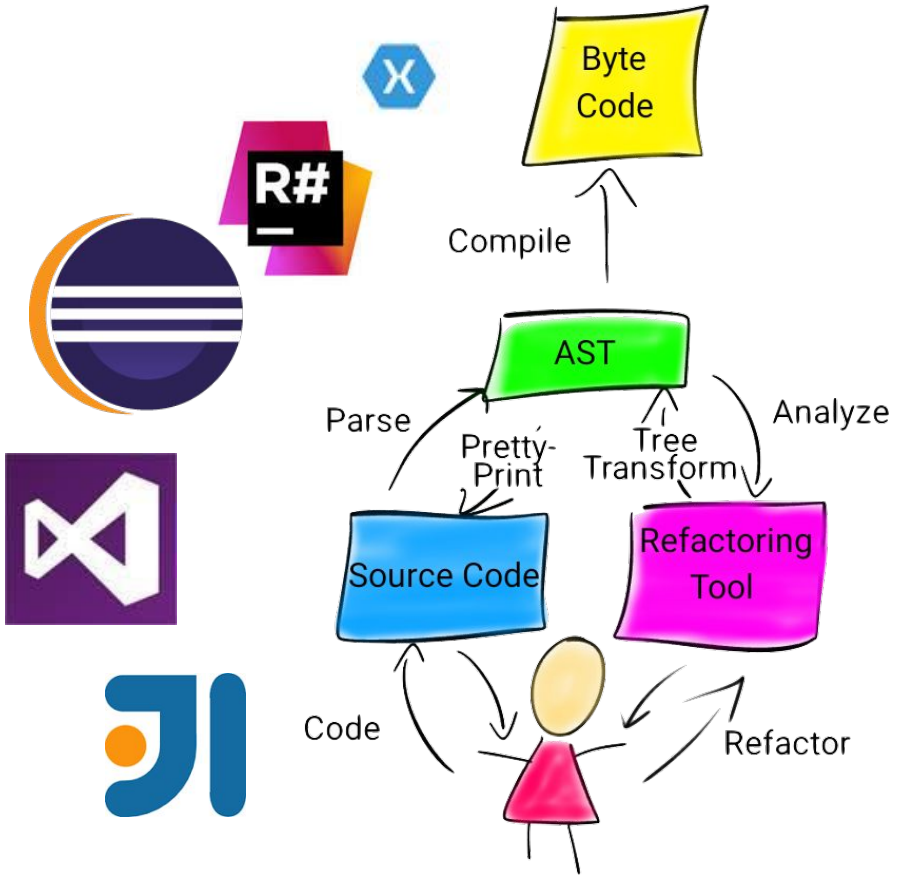
Manual refactoring

- How?
- Pros/cons?

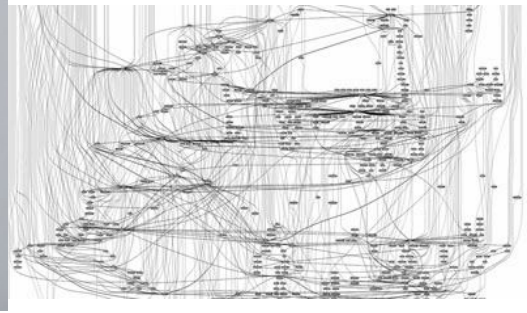
Automated tools

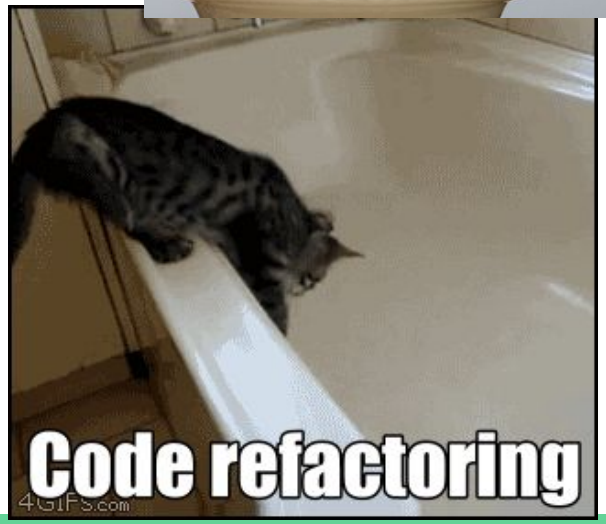
- How?
- Other pros/cons?

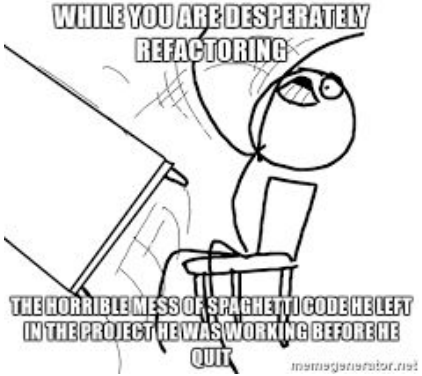
Ideally: *correct* automation (as with everything..)



Refactoring: ~~how to do it?~~ Why does everyone hate it?







Refactoring: how to do it? - Extract Local Variable

Before

```
1  public void f() {
2    a.b.c.d.m();
3    a.b.c.d.n();
4    a.b.foo(a.b.c.d);
5    a.b.bar();
6    a.b.c.d.m();
7    }
```

After

```
public void f() {
  D temp = a.b.c.d;
  temp.m();
  temp.n();
  a.b.foo(temp);
  a.b.bar();
  temp.m();
}
```

Refactoring: how to do it? - Extract Local Variable

input : e – an expression of non-void type E
: S – a selection, as a list of consecutive statements
: *context* – the outermost, non-type scope containing S
output: *context* with e extracted to a local variable in S

1 $v \leftarrow$ fresh variable name;
2 **for** $s \in S$ **do**
3 | in s replace all occurrences of e with v;
4 **end**
5 add a new variable declaration $E\ v = e$ *context* just before S;

Algorithm 1: `Extract Local Variable` algorithm: add a new variable declaration initialized to the target argument in the beginning of the selection, then replace all occurrences of target with a reference to the variable.

Refactoring: how to do it? - Extract Local Variable

Preconditions. These preconditions ensure that the resulting code is well-formed and behaves the same as before:

- The selected expression is of a non-void type.
- The selected expression has no side effects.
- The selected expression or its aliases are not assigned to within the code that is reachable from the selection.
- The selection is not the outermost type declaration in a compilation unit.
- The program is well-formed, i.e. syntactically correct and type-checks (compiles)

How to implement a refactoring tool

The Eclipse project JDK core refactoring API plugin!

➜ org.eclipse.jdt.core.refactoring ➜

Which relies on

- Eclipse Platform
  - Language Tool Kit, UI, ...
- org.eclipse.jdt.core
- org.eclipse.jdt.core.dom
- org.eclipse.jdt.core.dom.rewrite
- org.eclipse.jdt.core.util

...a tumble into the Eclipse java language model...

The Java Model (org.eclipse.jdt.core)

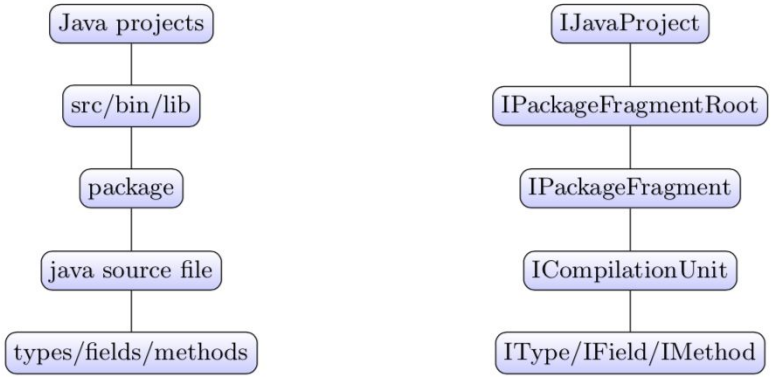


Figure 3.1: Project elements on the left, Java Model element to the right. Each node on the right represents the types that nodes on that level can have in the Java model, with the corresponding project elements to the left.

...a tumble into the Eclipse java language model...

The Java Model (org.eclipse.jdt.core)

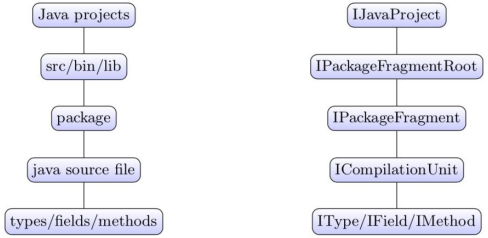


Figure 3.1: Project elements on the left, Java Model element to the right. Each node on the right represents the types that nodes on that level can have in the Java model, with the corresponding project elements to the left.

The DOM/AST (org.eclipse.jdt.core.dom)

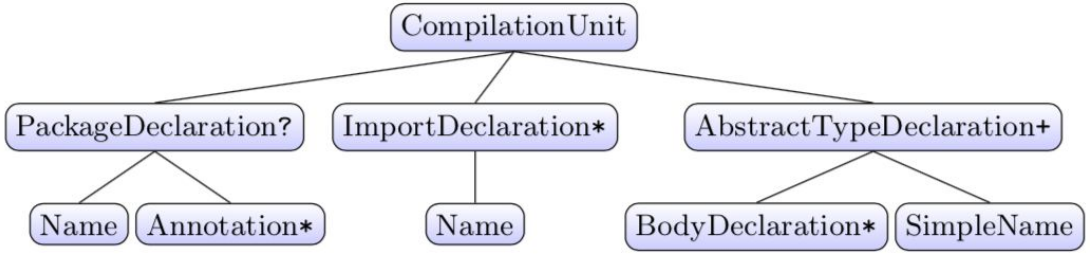
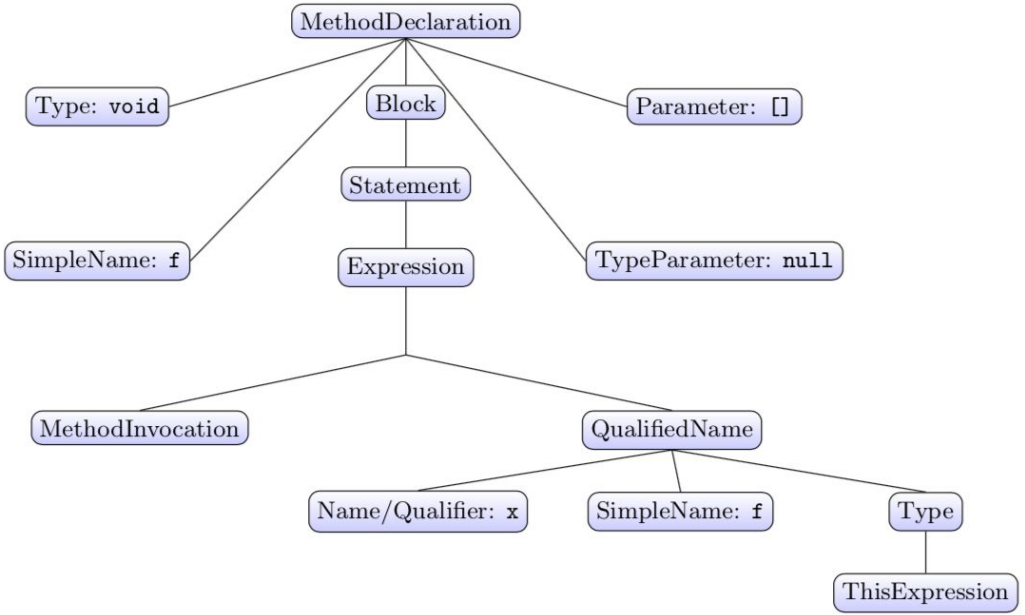


Figure 3.2: Simplified Eclipse AST representation of a Java Project

...a tumble into the Eclipse java language model...

The DOM/AST (org.eclipse.jdt.core.dom)

```
1  public class C {
2          public X x = new X()
3          public void f(X x) {
4                  x.m(this);
5          }
6  }
```



...a tumble into the Eclipse java language model...

```
1  ASTParser parser = ASTParser.newParser(AST.JLS8);
2  parser.setKind(ASTParser.K_COMPILATION_UNIT);
3  parser.setSource(document.get().toCharArray());
4  parser.setResolveBindings(true);
5  parser.setProject(PROJECT_NAME));
6  parser.setUnitName(UNIT_NAME);
7  CompilationUnit cu = (CompilationUnit) parser.createAST(PROGRESS_MONITOR);
```

...a tumble into the Eclipse java language model...

The general lifecycle of a refactoring in Eclipse is as follows:

1. The refactoring is launched by a user or a script. An initial check is performed to determine whether the refactoring is applicable at all in the context desired by the user (`checkInitialConditions()`).
2. Configuration details is supplied by the user or script if necessary.
3. After all necessary information has been provided, an in-depth check is invoked (`checkFinalConditions()`) and the individual changes in the source text are calculated (`createChange()`).
4. The preview dialogue displays the changes; the user confirms and the LTK applies them to the workspace.

A last step can include adding an undo change to the IDE's undo history.

...a tumble into the Eclipse java language model...

ExtractTempRefactoring.java

(running and debugging Eclipse in Eclipse.. Linked to the single class representing the refactoring)