WASP | WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM

# Software Engineering & Cloud Computing VT 2019

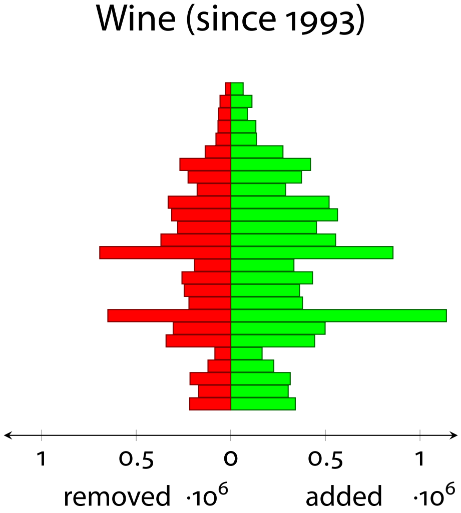
## REFACTORING

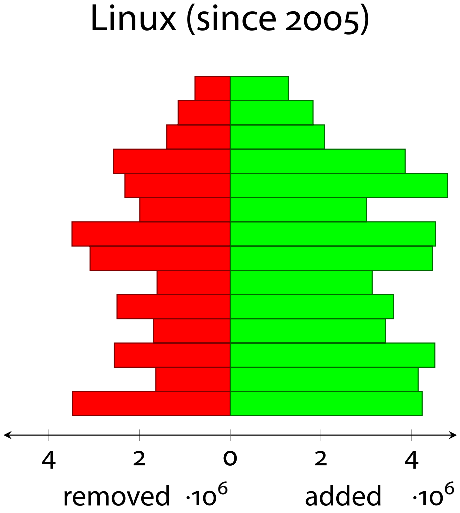
Christoph Reichenbach

CHALMERS UNIVERSITY OF TECHNOLOGY · KTH VETENSKAP OCH KONST · LINKÖPINGS UNIVERSITET · LUND UNIVERSITY · UMEÅ UNIVERSITET

## Never Change a Running System?

# Code Evolution





## Changes

Why change a running system *without altering its output*?

## Changes

Why change a running system *without altering its output*?
**Qualitative improvements:**

- Readability
- Maintainability
- Extensibility
- Safety
  Behaviour when we're running outside of the 'official' spec
- Resource utilisation

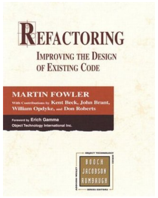
  …

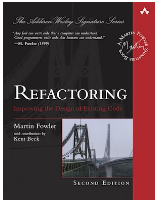
**Changes that don't affect *observable* behaviour**

## Refactoring

*"Refactoring is the process of chaging a software system in such a way tha it does not alter the external behavior of the code yet improves its internal structure."*

*— Martin Fowler, 'Refactoring: Improving the Design of Existing Code'*

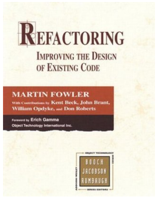

REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
BOOCH JACOBSON RUMBAUGH

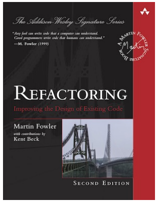

The Addison Wesley Signature Series
"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
—M. Fowler (1999)
A Martin Fowler Signature Book
REFACTORING
Improving the Design of Existing Code
Martin Fowler
with contributions by Kent Beck
SECOND EDITION

## Refactoring

*"Refactoring is the process of chaging a software system in such a way tha it does not alter the external behavior of the code yet improves its internal structure."*

*— Martin Fowler, 'Refactoring: Improving the Design of Existing Code'*

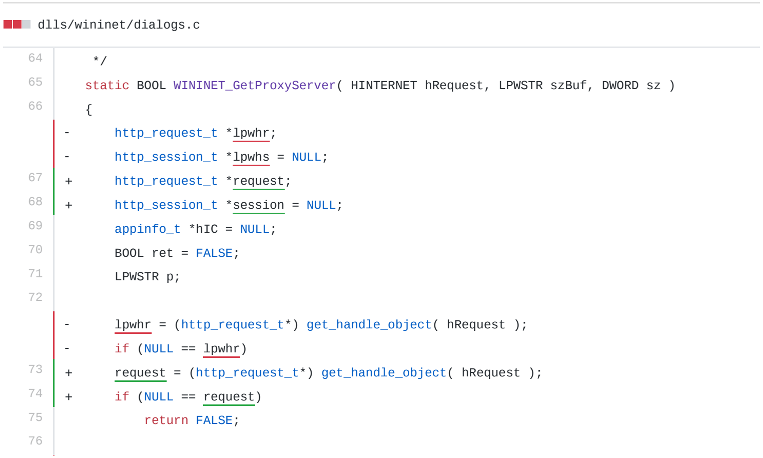




***noun***: *a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior*
***verb***: *to restructure software by applying a series of refactorings without changing its observable behavior.*

*— Martin Fowler, 19 years later*

## Readability



```diff
dlls/wininet/dialogs.c

  */
 static BOOL WININET_GetProxyServer( HINTERNET hRequest, LPWSTR szBuf, DWORD sz )
 {
-    http_request_t *lpwhr;
-    http_session_t *lpwhs = NULL;
+    http_request_t *request;
+    http_session_t *session = NULL;
     appinfo_t *hIC = NULL;
     BOOL ret = FALSE;
     LPWSTR p;

-    lpwhr = (http_request_t*) get_handle_object( hRequest );
-    if (NULL == lpwhr)
+    request = (http_request_t*) get_handle_object( hRequest );
+    if (NULL == request)
         return FALSE;

```

## The ‘Rename’ refactoring

- **Problem:** The name of a function / method / variable / …either:
  - does not reflect its meaning, or
  - violates naming conventions, or
  - is inconsistent with naming elsewhere in the code base
- **Remedy:**
  - Rename the function / method / variable / …
  - Adjust all references to the renamed entity

```
procedure allelts() {  ...  }
                ⇓
procedure size()    {  ...  }
```

## The ‘Rename’ refactoring

- **Problem:** The name of a function / method / variable / …either:
    - does not reflect its meaning, or
    - violates naming conventions, or
    - is inconsistent with naming elsewhere in the code base
- **Remedy:**
    - Rename the function / method / variable / …
    - Adjust all references to the renamed entity
    - *Validate that behaviour hasn't changed* (e.g. unit tests)

```
procedure allelts() {  ...  }
               ⇓
procedure size()    {  ...  }
```

## Renaming is easy...?

**Name capture**

```
var a = 10;
function f(b) {
    return a * b;
}
```

$$\Downarrow$$

```
var a = 10;
function f(a) {
    return a * a;
}
```

## Renaming is easy...?

### Name capture

```
var a = 10;
function f(b) {
    return a * b;
}
```

$\Downarrow$

```
var a = 10;
function f(a) {
    return a * a;
}
```

### Hierarchical Dependency

```
class A {
  method f() {...}
}
```

```
class B extends A {
  override
  method f() {...}
}
```

# Renaming is easy...?

## Name capture

```
var a = 10;
function f(b) {
    return a * b;
}
```

$\Downarrow$

```
var a = 10;
function f(a) {
    return a * a;
}
```

## Hierarchical Dependency

```
class A {
  method f() {...}
}
```

```
class B extends A {
  override
  method f() {...}
}
```

## External APIs

- Project **P1**, org. C1:
  - f → g
- Project **P2**, org. C2:
  - Calls operation f

## The ‘Rename’ refactoring

`https://refactoring.com/catalog/renameVariable.html`

### Rename Variable

```
let a = height * width;
```

```
let area = height * width;
```

## When to Refactor?

## Refactoring: ‘Replace Magic Literal’

### Replace Magic Literal

```
2 * 3.14 * radius
```

$\uparrow$

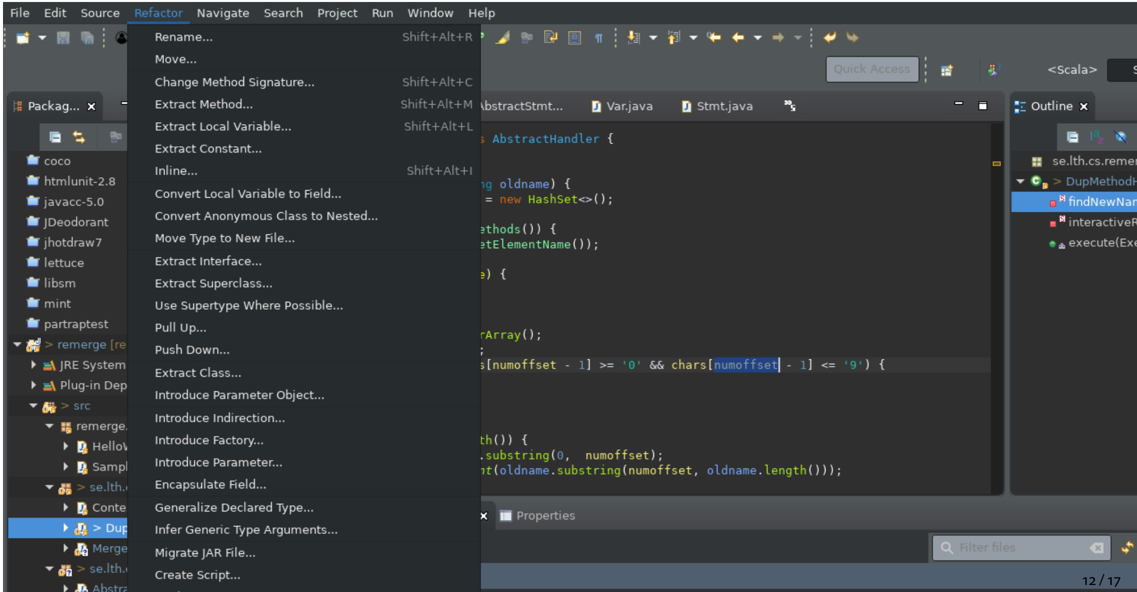
$\pi$

```
function potentialEnergy(mass, height) {
  return mass * 9.81 * height;
}
```

$$\Downarrow$$

```
const STANDARD_GRAVITY = 9.81;
function potentialEnergy(mass, height) {
  return mass * STANDARD_GRAVITY * height;
}
```

# Renaming IDE support



```
s AbstractHandler {

ng oldname) {
= new HashSet<>();

ethods()) {
etElementName());

e) {


rArray();
;
s[numoffset - 1] >= '0' && chars[numoffset - 1] <= '9') {



th()) {
.substring(0,  numoffset);
nt(oldname.substring(numoffset, oldname.length()));
```

## Refactoring Tools

| | Eclipse | IntelliJ | NetBeans | Visual Studio | Wing IDE |
|---|---|---|---|---|---|
| **Java** | yes | yes | yes | | |
| **C** | some | AppCode | | yes | |
| **C++** | some | AppCode | | yes | |
| **Python** | PyDev | PyCharm | | | yes |
| **JavaScript** | | WebStorm | | | |
| **Fortran** | Photran | | | | |

## Some Common Refactorings

- Extract Variable
- Extract Function / Extract Method
- Move
- Convert Function $\leftrightarrow$ Method
- Split Temporary
- Change Function Declaration
- Remove Dead Code

## Most Popular Refactorings

'*A Comparative Study of Manual and Automated Refactorings*', by Stas Negara, Nicholas Chen, Mohsen Vakilian, Ralph E. Johnson, and Danny Dig (ECOOP 2013)

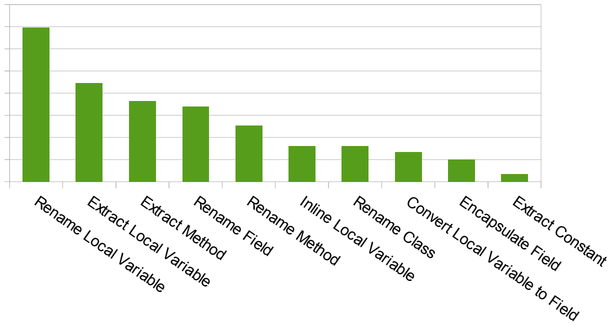


**Fig. 4.** Popularity of refactorings.

## Object Orientation

- Programming strategy
- Often with language support
- Idea:
  - Group operations with most relevant data record
  - Hierarchically structure data records (e.g., Employee, Manager, SW Engineer)
  - Specialise operations for each variant (e.g., `computeBonus()`)
- Requirements:
  - *Inheritance*
  - *Dynamic Dispatch*
  - *Subtyping*
- Usually combined with *Encapsulation*
- `http://ee402.eeng.dcu.ie/introduction/chapter-1---introduction-to-object-oriented-programming`

## Grading Criteria

- Make 16 points’ worth of refactoring commits
  - Simple refactoring (rename etc.): 1 point
  - Complex refactoring: 3 points
  - At least 2 complex refactorings
  - Jesper Öqvist maintains list of refactorings, categories on Canvas
  - Jesper needs repository access
  - You submit commit IDs to Canvas
- Stored as:
  - A *single commit*
  - Names the refactoring that you used
    - Use names from Fowler’s catalogue whenever possible

```
Moved Example1.circleSize into Circle class

Applied the "Move Method" refactoring in preparation for
generalising our size computation to other shapes.
```