CS 4530: Fundamentals of Software Engineering

Lesson 11 Code Smells, Refactoring and Technical Debt

Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand

Khoury College of Computer Sciences

Outline of this lesson

1. Some common code “smells” (anti-patterns).
2. “Refactoring”: restructuring of code to improve structure.
3. “Technical Debt”: generalization covering all internal problems in a code-base.

Learning Objectives for this Lesson

- By the end of this lesson, you should be able to:
  - Review several classes of code smells;
  - Describe several kinds of refactoring;
  - Identify the “technical debt” metaphor;
  - Indicate when and where technical debt is appropriate to accrue versus retire.

“Code Smells” are Anti-Patterns

- Cases of **poor** code:
  - Likely to harbor faults;
  - Difficult to use;
  - Expensive to maintain.
- Common and Known:
  - Each code smell has a name,
  - … and a recommended fix.
- Example catalog:

https://refactoring.guru/refactoring/smells

(figure courtesy of Refactoring Guru)

Code Smell Example (1 of 3)

```
class Product {
    private _id : string;
    private _desc : string;
    private _weight : number;

    public get id() { return                this._id      ; }
    public set id(          newID ) {
        this._id        =  newID ;
    }
    public get desc() {
        return      this._desc      ;
    }
    // set desc
    // get weight, set weight
}
```

- DATA CLASS
- A class has public properties (or public getters and setters) and few if any methods.
- How to fix:
  - Determine what is being done with class properties;
  - Make some properties immutable;
  - Define methods to perform tasks;
  - Reduce getters/setters.

Code Smell Example (2 of 3)

```
if (     this.width         >  lineSize      ) {
     warn('at beginning, too big');
     this.width         - = OVERFULL;
}

// more code

if (     this.width         >  lineSize      ) {
     warn('before return, too big');
     this.width         - = OVERFULL;
}
```

- DUPLICATED CODE
- The same (or very similar code) occurs more than once.
  - Multiplies maintenance work.
- How to Fix:
  - Extract the common code in a method;
  - Use that method where code was.

Code Smell Example (3 of 3)

```
setUpPage
    ( USLetter.width       ,
      USLetter.height      ,
      recipe.getTitle      (),
      recipe.getContents        (),
      defaultFont      ,
      2, /* number of columns
      true, /* number pages? */
      false, /* balance? */
      1.4, /* PDF level */
      outputFile      );
```

- TOOMANYPARAMETERS
- A method has a long list of parameters; difficult for clients to keep order and number straight.
- How to Fix:
  - Package up groups of related parameters in objects, or
  - Separate method into parts with fewer arguments.

Refactoring is Code Restructuring

- Code is reorganized:
  - No (executable) code is added or removed;
  - Code's behavior is preserved;
    - (not for fixing bugs!)
  - Change is reversible;
- Metaphor: topology-preserving transformations:

Refactoring Can Improve Code

- Refactoring can remove “smells”:
  - Bring together similar responsibilities;
  - Separate disjoint responsibilities.
- Refactoring can improve code flexibility:
  - It can add generality/abstraction;
  - This prepares for changes to come later.
- Refactoring can break code, if done wrong:
  - IDEs provide (usually) safe refactorings;
  - Use regression tests to double-check.

Refactoring Example (1 of 3)

- EXTRACT LOCAL
- Pull an expression out into a named local variable.
- (In this case, preparing for next step so that duplicates can become identical.)

```
if (     this.width         >  lineSize      ) {
    warn('at begin, too big');
    this.width         - = OVERFULL;
}
```

```
const msg =                    ;
if (     this.width         >  lineSize      ) {
    warn(  'at begin, too big'
                                    msg);
    this.width         - = OVERFULL;
}
```

Refactoring Example (2 of 3)

- EXTRACT METHOD
- Pull out code with locals becoming formal parameters.

```
checkWidth    ( lineSize:number      ,
                msg:string     ) {




}
```

```
const msg = 'at begin, too big';
if (     this.width         >  lineSize     ) {
    warn(msg);
    this.width         - = OVERFULL;
}
```

```
const msg = 'at begin, too big';
if (     this.width         >  lineSize     ) {
    warn(msg);
    this.width         - = OVERFULL;
}
```

Refactoring Example (3 of 3)

- INLINE LOCAL
- Replace name with value.
- Inverse of EXTRACT LOCAL.

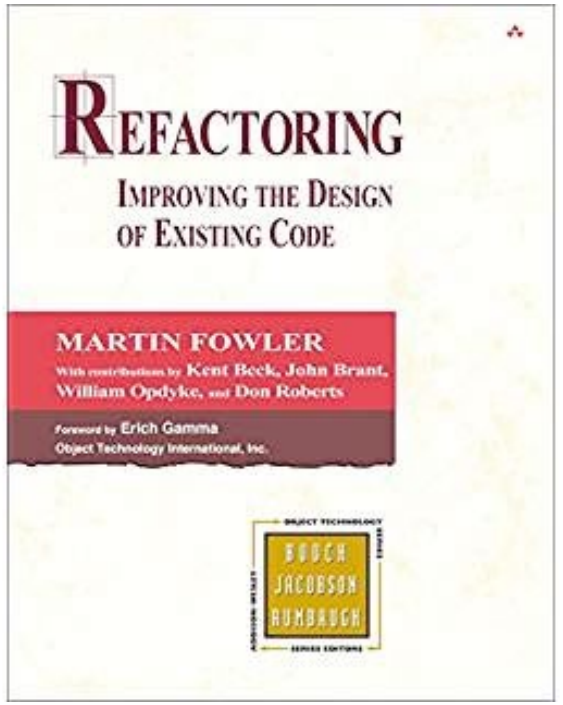
To avoid hard-coding, the next task would be to EXTRACT CONSTANT.

```
const msg = 'at begin, too big';
this.checkWidth      ( lineSize     , msg);
```

```
const msg =      'at begin,; too big'
this.checkWidth      ( lineSize     ,   msg);
```

More Refactoring

- Extract Interface / Extract Abstract Class
- Introduce Parameter
  - Take out special case from function into new argument.
- Make Static / Make Instance
- Move Method (to new class)
- […]



Technical Debt is Sum of Internal Problems in Project Codebase

- **Internal** because they don't show as user-visible failures.
- Examples:
  - Code Smells;
  - Missing tests;
  - Missing documentation;
  - Dependency on old versions of third-party systems;
  - Inefficient and/or non-scalable algorithms.

Not just code!

Technical Debt Exacts Interest During Maintenance (Usually)

Example of Debt

- Code Smells;
- Missing tests;
- Missing documentation;
- Dependency on old versions of third-party systems;
- Inefficient and/or non-scalable algorithms.

Example of Cost

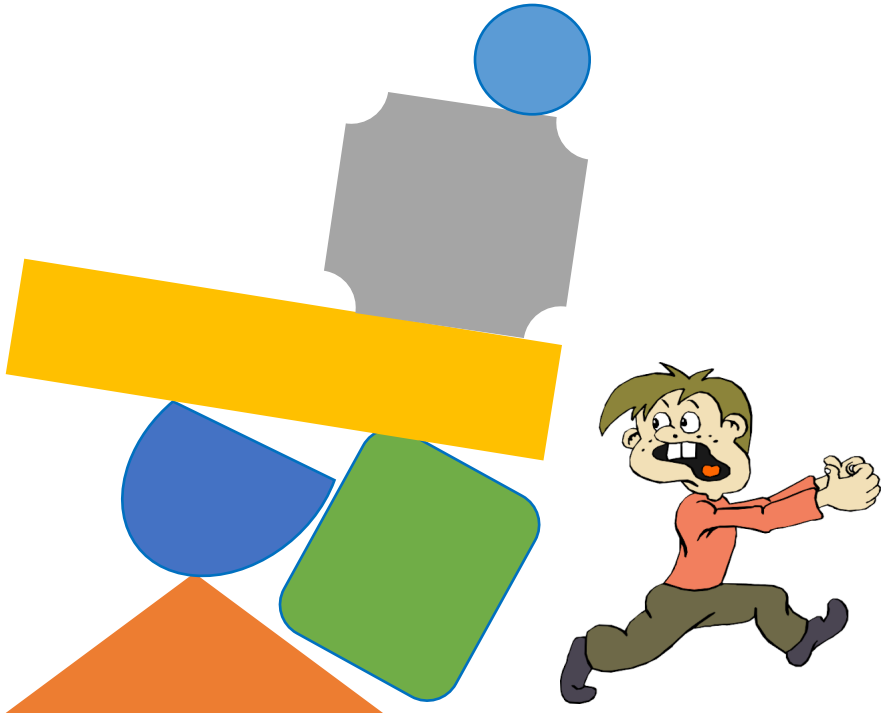
- “Smelly” code is less flexible;
- Need to revert breaking change;
- Can’t figure out how to use;
- May have take over maintenance of old system;
- Lose potential customers.

Good Reasons to Go Into Technical Debt

- Prototyping:
  - If code will be discarded, or drastically rewritten, don't waste time perfecting it.
- Getting a product out the door:
  - Time is often crucial in a competitive environment.
- Fixing a critical failure:
  - People are waiting.
- Maybe a simple algorithm is good enough:
  - "Premature optimization is the root of all evil"
    - Tony Hoare, Donald Knuth

Retire Technical Debt at Leisure

- Set aside time to pay off technical debt:
  - Google has (had?) “20%-time” for tasks such as this.
- A new initiative can take on some technical debt:
  - Refactoring at the start of a project.
- Don’t keep on putting off!
  - When a crisis hits, it’s too late;
  - Hasty fixes to unmaintainable code multiplies problems;
  - Eventually mounting technical debt can bury the team.



Review: Learning Objectives for this Lesson

- You should now be able to:
  - Review several classes of code smells;
  - Describe several kinds of refactoring;
  - Identify the “technical debt” metaphor;
  - Indicate when and where technical debt is appropriate to accrue versus retire.