refactoring

Andrew J. Ko
question of the day...

you want to change what your software can do

how do you do this without sacrificing all of the other work you’ve done?
the primitive approach

just go in and change what you need to change until it works!

if you do this, you risk breaking functionality that worked before, causing regressions, data loss, and other issues
the principled approach

1. specify what you want the new version of your software to do

2. analyze what aspects of your current version conflict with this desired change in behavior

3. determine the difference between the current and desired version in terms of some set of code transformations

4. devise a plan for implementing the code transformations

5. execute!
refactoring

changing a computer program’s code without modifying its external behavior in order to improve some non-functional quality

improving readability
reducing complexity
facilitating the implementation of new features
improving testability
etc.

etymology

from mathematics and algebra, where the structure of expressions are transformed while maintaining equality

\[ x^2 + x \rightarrow x(x + 1) \]

in the above example, \( x \) is factored out of the original expression

development, but applied to the structure of code
what can you refactor?

the **structure** of your code

its form, not its function

you can **generalize** code, making the structure more amenable to other behaviors

this may help you add new features, customize functionality, make code more readable, improve the separation of concerns, or hide information

you can **specify** code, making the structure less amenable to other behaviors

this may improve performance, remove features, make code simpler, or eliminate security issues
rename

change all references to a name from one name to another

```javascript
var dialog = $('#dialog');
dialog.css('color', 'red');
dialog.append('Sorry.');

var errorDialog = $('#dialog');
errorDialog.css('color', 'red');
errorDialog.append('Sorry.');
```
move

moving some encapsulation from one container to another

```javascript
var note = {
    delete: function() { ... }
... 
}
note1.delete();

var note = {
    delete: function() { ... }
... 
}
var NoteOperations {
    delete: function(note) { ... }
... 
} 
NoteOperations.delete(note1);
```
change method signature

adding, removing, or retyping parameters

```javascript
function addNote(text, category) {
    ...
    addNote("Hello", 0);
}
```

```javascript
function addNote(text, category, richtext) {
    ...
    addNote("Hello", 0, false);
}
```
extract local

some expression computes a value you’d like to reuse

\[
\text{find}(\max(0, \text{latestTime}(\text{notes})))
\]

\[
\text{var } \text{latestTime} = \max(0, \text{latestTime}(\text{notes}));
\text{find}(\text{latestTime});
\]
extract constant

you’d like to make it easier to change a literal that you’ve hard coded

```javascript
return new Note("default text");
```

```javascript
var Constants {
    DEFAULT_TEXT : "default text"
}
...
return new Note(Constants.DEFAULT_TEXT);
```
inline

the opposite of extracting a constant or local, eliminating a variable

```javascript
var Constants {
    DEFAULT_TEXT : "default text"
}
...
return new Note(Constants.DEFAULT_TEXT);
```

```javascript
return new Note("default text");
```
local variable to field

the opposite of extracting a constant or local, eliminating a variable

```javascript
var object = {
   doStuff: function() {
      var thing = 5;
      otherStuff(thing);
    
   }
}

var object = {
   thing : 5,
   doStuff: function() {
      otherStuff(this.thing);
    
   }
}```
and so on...

refactoring can be applied to any structure in a programming language

therefore, you can imagine there being refactoring tools for many specific situations

the limitation is that refactoring tools can only be applied automatically when the tool can identify **all possible dependencies** on the subject of a refactoring

if it cannot, then it cannot be 100% confident that the refactoring applied will preserve the system’s external behavior
manual refactoring

most refactoring can’t be done automatically especially in a dynamic language like JavaScript, where there is little to no type information

therefore, you have to identify all dependencies on the subject of a refactoring manually if you don’t, you’ll break something, and you’ll have to rely on your unit and regression tests to catch cases that you overlooked

you did write unit and regression tests, right? :)

Friday, February 24, 12
refactoring enables expression

ultimately, refactoring is about changing a program’s structure to enable you to express ideas that its current structure makes difficult or impossible

therefore, before you engage in it, you should have a very clear idea about what you’re trying to express

and what impact your refactorings will have on other ideas you may want to express in the future
activity while you do this, I’ll be coming around to grade deliverable 7 with you

based on your 2.0 plan, what parts of our application do you have to refactor?

make a **refactoring plan**

what specific, concrete changes do you need to make? what order will you make them?

what code depends on those changes?

what testing will you do to verify that, after each change, you didn’t change (or break) any of the behavior that worked before?

when will you do the refactoring? will it block other people from contributing work?