How do you refactor?

Asked 16 years, 2 months ago Modified 13 years, 6 months ago

Viewed 2k times

20

I was wondering how other developers begin refactoring. What is your first step? How this process (refactoring) differ if you refactor code which is not yours? Do you write tests while refactoring?

unit-testing refactoring

Share Follow

edited Oct 15, 2008 at 16:35

harriyott

10.7k ● 10 ● 68 ● 104

asked Oct 15, 2008 at 16:31

rafek

5,484 ● 13 ● 60 ● 72

1 See stackoverflow.com/questions/20262/…. See stackoverflow.com/questions/48817/…. None of these helped? stackoverflow.com/questions/tagged/refactoring – S.Lott Oct 20, 2008 at 12:00

14 Answers

Sorted by: Highest score (default)

25

1. do not refactor anything non-trivial that does not already have unit tests
2. write unit tests, then refactor
3. refactor small pieces and re-run the tests frequently
4. stop refactoring when the code is DRY* clean

* DRY = Don't Repeat Yourself

Share Follow

edited Oct 17, 2008 at 15:55

answered Oct 15, 2008 at 16:34

Steven A. Lowe
61.1k ● 19 ● 135 ● 204

DRY = Don't Repeat Yourself, I didn't know until I looked it up, so I thought I would give others a heads up. – James McMahon Oct 17, 2008 at 12:07

1 @[Mike Burton]: hence item (2) – Steven A. Lowe Dec 11, 2008 at 16:57

1 Have you tried to write unit tests for large pre-existing code bases? – Mike Burton Dec 11, 2008 at 22:53

1 @[Mike Burton]: yes. No one said it would be easy, but refactoring without them is asking for tragic surprises ;-) – Steven A. Lowe Dec 12, 2008 at 2:49

What is your first step?

8

The first step is to run the unit tests to make sure they all pass. Indeed, you can waste a large amount of time looking for which of your changes broke the a test if it was already broken *before* your modify the code.

> How this process differ if you refactor code which is not yours ?

I'm certainly doing smaller steps when refactoring code I didn't write (or code I wrote a long time ago). I may also verify the test coverage before to proceed, to avoid relying on unit tests that always pass ... but that do no test the area I'm working on.

> Do you write tests while refactoring ?

I usually don't, but I may add new tests in the following circumstances (list not exhaustive) :

- idea of a new test sparkles in my mind (*"what happen if ... ?"* - write a test to know)
- discover hole in the test coverage

it also depends on the refactoring being performed. When extracting a function, I may create a new test if it can be invoked a different way as it was before.

Here are some general advice:

First thing is to maintain a list of the code smells noticed while working on the code. This allows freeing one's mind from the burden of remembering what was seen in the code. Also,

The golden rule is never refactor when the the unit tests do not pass completely.

Refactor when the code is stable, before adding something you know will be impacted by a future refactoring, before to integrate and above all **before to say your done**.

In the absence of unit tests, you'll have to put the part of code you want to refactor under test. If unit tests are too hard to retrofit, as it usually is the case, then you can create characterization tests, as recommended per Michael Feathers in *Working Effectively with Legacy Code*. In short they are end-to-end tests that allow you to pin down the current behavior of the code (which is not assumed to be working perfectly all the time).

Don't be afraid to do *baby steps*. Do not do two things at the same time. If you remark something requiring refactoring, note it down, do not fix it right now, even if it seem very easy.

Check in very often, when the tests pass. So that you can revert a bad refactoring without losing what was done before.

Keep on mind refactoring does not add value to your customer (this can be discussed), but the customer does not pay you to refactor. One rule of thumb is to refactor prior making changes or adding new capabilities to the code.

Share Follow

edited May 23, 2017 at 12:13

Community Bot

1 ● 1

answered Oct 17, 2008 at 11:48

philant

35.7k ● 11 ● 73 ● 113

4

Read Martin Fowler's book "Refactoring"

BTW - that's Martin Fowler's own Amazon exec link, if you're wondering :)

Share Follow

edited Jan 23, 2009 at 17:13

answered Oct 15, 2008 at 16:42

MarkJ

30.4k ● 5 ● 71 ● 113

4

I take crap and make it less crappy. :-)

Seriously. I don't refactor to create new functionality. Refactoring occurs before new stuff. If there are no tests I

write tests to make sure that I'm not breaking anything with my refactoring. If there are tests, I use those. If the tests are insufficient, I might write more tests but I would consider this separate from the refactoring and do it first.

The first step for me is to notice that I can abstract something and make it more general (and useful in other places that need the functionality now), or I notice that something is bad and could be better (subjective). I don't refactor to generality without a reason. YAGNI principle applies.

We have the concept of shared ownership so the code is always mine -- I may not have written it, but I don't consider that when refactoring. I may seek to understand things before I decide it needs refactoring if the purpose isn't clear -- although that's almost always a reason to refactor in and of itself.

Share Follow

edited May 17, 2011 at 5:27

yegor256

105k ● 130 ● 460 ● 620

answered Oct 15, 2008 at 16:39

tvanfosson

532k ● 102 ● 699 ● 798

3

Depends very much on my goals. As has been said, you need unit tests to make sure your refactoring hasn't broken anything, and if it has you have to commit time to fixing it. For many situations, I test the existing solution,

and if it works, wrap it rather than refactoring it, as this minimises the possibility of a break.

If I have to refactor, for example I recently had to port a bunch of ASCII based C++ to UNICODE, I tend to make sure I have good regression tests that work at end-user as well as unit level. Again, I try to use tools rather than manually refactoring, as this is less prone to error, and the errors you do get are systematic rather than random.

Share Follow

answered Oct 15, 2008 at 16:48

SmacL

22.9k ● 15 ● 99 ● 151

2

For me first thing is make sure the code hits all of the best practices of our office. For example, using strict, warnings and taint for our Perl scripts.

If there are efficienry or speed troubles focus on them. Things like find a better algorithm, or find a better way to to do what the quadruply nested for loop is doing.

And lastly see if there is a way to make the code more readable. This is normally accomplished by by turning 5 little scripts that do similar things into 1 module(class).

Share Follow

answered Oct 15, 2008 at 16:35

J.J.

4,892 ● 1 ● 26 ● 29

2

I refactor whilst writing new code, using unit tests. I'll also refactor old code, either mine or someone else's, if methods are too long, or variables named badly, or I spot duplication etc.

Share Follow

answered Oct 15, 2008 at 16:35

harriyott

10.7k ● 10 ● 68 ● 104

2

Start with getting unit tests, and then use automated refactoring tools. If the refactoring can't be automated then it's not truly a mechanical transformation of the code and so isn't a refactoring. The unit tests are to make sure that you are truly just performing mechanical transformations from one codebase to an equivalent one.

Share Follow

answered Oct 15, 2008 at 16:37

workmad3

25.6k ● 4 ● 37 ● 56

2

Refactoring without **Unit Test** is dangerous. Always have unit test. If you change something without having good testing you might be safe for some portion of the code but something elsewhere might not have the same behavior. With Unit Testing you protect any change.

Refactoring **other code** is fine too but the extreme is not. It's normal that someone else do not program like you do.

It's not "kind" to change stuff because you would have did it in the other way. Just refactoring if it's really necessary.

Share Follow

answered Oct 15, 2008 at 16:39

Patrick Desjardins

141k ● 89 ● 294 ● 346

2

I remove duplication, which unifies the thought patterns inherent in the code. Refactoring needs to achieve those two things. If you have code that does the same thing twice, refactor it to a common location, unifying the abstraction. If you have the same literal in three places, put it in a constant, unifying the purpose. If you have the same group of arguments, ensure they're always used in the same order or, better yet, put them in a common structure, unifying information groups.

Share Follow

answered Oct 15, 2008 at 16:40

Mike Burton

3,040 ● 25 ● 33

2

I'm a lot more reluctant to refactor code written by others than to refactor my own.

If it's been written by one of my predecessors, I generally refactor only within a function. E.g. I might replace an if statement with a switch. Anything much bigger than that is usually out of scope and not within budget.

For my own code, I usually refactor as I'm writing, whenever something looks ugly or starts to smell. It's a lot easier to fix it now instead of waiting for it to cause problems down the road.

Share Follow

answered Oct 16, 2008 at 20:05

Bruce Alderman

2,294 ● 2 ● 27 ● 38

1

I agree with the other posters when you're refactoring code you wrote.

If it's code you didn't write, and especially if there's lots of it, I would start by using tools like fxCop, Visual Studio's Code Analysis, DevPartner -- I'm sure there are other good ones. They would give you ideas about where to start and what the most common coding issues are. I would also do stress testing to see where the bottlenecks are, therefore the greatest return on your effort at improving the code.

I love to refactor my code, but it is possible to overdo it. If you aren't really improving the app's performance, or seriously improving code readability, you should probably stop. There is always the possibility of introducing new bugs when you refactor, especially if you're working without unit tests.

Share Follow

answered Oct 15, 2008 at 16:43

DOK

32.8k ● 8 ● 63 ● 93

0

First step: Identify a code smell.

Second step: Consider alternative implementations and what the trade offs are and which do I accept in terms of which is "better."

Third step: Implement better solution.

This doesn't differ if the code is mine or not as sometimes I may go back over code I wrote months or years ago and it'll look like code from someone else. I may write tests if I'm making new methods or there aren't adequate tests to the code, IMO.

Share Follow

answered Jan 23, 2009 at 17:18

JB King

11.9k ● 4 ● 40 ● 49

0

General approach

I may start out by peeking at the software (component) from bird-view. Dependency analysis and graphing tools are a great help here (see later sections). I look for a circle in either the package-level or the class-level dependencies, and also alternatively for classes with too

many dependencies. These are good candidates for refactoring.

Tools of the trade

Dependency analysis tools:

- Python
- Java

Share Follow

edited May 23, 2017 at 12:06

Community Bot

1 ● 1

answered Jun 15, 2011 at 12:18

ron

9,364 ● 5 ● 44 ● 73