How often should you refactor?

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

Viewed 13k times

63

I had a discussion a few weeks back with some co-workers on refactoring, and I seem to be in a minority that believes "Refactor early, refactor often" is a good approach that keeps code from getting messy and unmaintainable. A number of other people thought that it just belongs in the maintenance phases of a project.

If you have an opinion, please defend it.

refactoring

Share

Improve this question

Follow

asked Sep 26, 2008 at 16:59

Bob King

25.8k ● 7 ● 57 ● 66

8 The Boy Scouts of America have a simple rule *Leave the campground cleaner than you found it*. So `Always check a module in cleaner than when you checked it out`. Robert Martin. – Andrii Abramov Sep 29, 2016 at 20:49

25 Answers Sorted by: Highest score (default)

83

Just like you said: refactor early, refactor often.

Refactoring early means the necessary changes are still fresh on my mind. Refactoring often means the changes tend to be smaller.

Delaying refactoring only ends up making a big mess which further makes it harder to refactor. Cleaning up as soon as I notice the mess prevents it from building up and becoming a problem later.

Share Improve this answer Follow

edited Sep 26, 2008 at 17:07

answered Sep 26, 2008 at 17:00

jop

84k ● 10 ● 57 ● 52

37

I refactor code as soon as it's functional (all the tests pass). This way I clean it up while it's still fresh in my mind, and before anyone else sees how ugly the first version was.

After the initial check-in I typically refactor every time I touch a piece of code. Refactoring isn't something you should set aside separate time for. It should be something you just do as you go.

Share Improve this answer Follow

edited Sep 26, 2008 at 17:15

community wiki
2 revs
Bill the Lizard

2 Agreed, that's also the motive with one representation of TDD: Red, Green, Refactor! – Jim Deville Sep 26, 2008 at 17:03

I think that this is a late stage to refactor, as you're stuck with a significant codebase by that stage. – Marcin Sep 26, 2008 at 17:27

By "code" I don't mean "application", I mean module, class, method, or some other small increment of code.
– Bill the Lizard Sep 29, 2008 at 17:06

I would think you'd want your refactored code tested as well? – Aaron Bush Feb 2, 2010 at 22:03

2 @Aaron: Testing is a *part* of refactoring. You can't say you're done until the code is tested and passes. Testing is a part of the larger process, so it's redundant to explicitly state that you're going to test the code. – Bill the Lizard Feb 3, 2010 at 15:57

28

You write code with two hats on. The *just-get-the-thing-working* hat and the *I-need-to-understand-this-tomorrow* hat. Obviously the second hat is the refactoring one. So you refactor every time you have made something work but have (inevitably) introduced smells like duplicated code, long methods, fragile error handling, bad variable names etc...

Refactoring whilst trying to get something working (i.e. wearing both hats) isn't practical for non-trivial tasks. But postponing refactoring till the next day/week/iteration is very bad because the context of the problem will be gone from your head. So switch between hats as often as possible but never combine them.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:13

Garth Gilmour

11.3k ● 5 ● 28 ● 36

> what a great point! i'm gonna use it in my lecture to persuade my colleagues to refactor their shitty code/ – MartianMartian Mar 30, 2018 at 8:36

19

I refactor every chance I get because it lets me hone my code into the best it can be. I do this even when actively developing to prevent creating unmaintainable code in the first place. It also oftentimes lets me straighten out a poor design decision before it becomes unfixable.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:01

Bob King

25.8k ● 7 ● 57 ● 66

15

Three good reasons to refactor:

- Your original design (perhaps in a very small area, but design nonetheless) was wrong. This includes



where you discover a common operation and want to share code.

- You are designing iteratively.
- The code is so bad that it needs major refurbishment.

Three good reasons not to refactor:

- "This looks a little bit messy".
- "I don't entirely agree with the way the last guy did this".
- "It might be more efficient". (The problem there is 'might').

"Messy" is controversial - there is a valid argument variously called "fixing broken windows", or "code hygiene", which suggests that if you let small things slide, then you will start to let large things slide too. That's fine, and is a good thing to bear in mind, but remember that it's an analogy. It doesn't excuse shunting stuff around interminably, in search of the cleanest possible solution.

How often you refactor should depend on how often the good reasons occur, and how confident you are that your test process protects you from introducing bugs.

Refactoring is never a goal in itself. But if something doesn't work, it has to be fixed, and that's as true in initial development as it is in maintenance. For non-trivial changes it's almost always better to refactor, and

incorporate the new concepts cleanly, than to patch a single place with great lumps of junk in order to avoid any change elsewhere.

For what it's worth, I think nothing of changing an interface provided that I have a handle on what uses it, and that the scope of the resulting change is manageable.

Share Improve this answer

Follow

edited Sep 26, 2008 at 17:18

answered Sep 26, 2008 at 17:12

Steve Jessop

279k ● 40 ● 469 ● 709

7

As the book says, You refactor when

- you add some code... a new feature
- when you fix a bug / defect
- when you do a code-review with multiple people
- when you find yourself duplicating something for the third time.. 3 strikes rule

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:03

Gishu

137k ● 47 ● 226 ● 311

3 strikes is highly debatable. see c2.com/cgi/wiki?OnceAndOnlyOnce vs c2.com/cgi/wiki?

ThreeStrikesAndYouRefactor I have lived through and heard too many anecdotes about projects being adversely affected because someone said "Wait until we write that one more time". – Mike A Sep 2, 2009 at 7:53

1 lol, we need more one comments to start a disagreement on this one. – MartianMartian Mar 30, 2018 at 8:41

7

I try to go by this motto: leave all the code you touch better than it was.

When I make a fix or add a feature I usually use that opportunity to do limited refactoring on the impacted code. Often this makes it easier to make my intended change, so it actually doesn't cost anything.

Otherwise, you should budget dedicated time for refactoring, if you can't because you are always fighting fires (I wonder why) then you should force yourself to refactor when you find making changes becomes much harder than it should and when "code smells" are just unbearable.

Share Improve this answer

Follow

answered Sep 26, 2008 at 20:18

Wedge

19.8k ● 7 ● 50 ● 71

6

A lot of times when I'm flushing out ideas my code starts out very tightly coupled and messy. As I start polishing the idea more the logical separations start becomming more and more clear and I begin refactoring. It's a constant

process and as everyone suggests should be done 'Early and Often'.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:03

Micah

116k ● 87 ● 237 ● 331

4

I refactor when:

I'm modifying code and I'm confused by it. If it takes me a while to sift it out, it needs refactoring.

I'm creating new code and after I've got it "working". Often times I'll get things working and as I'm coding I realize "Hey, I need to redo what I did 20 lines up, only with a few changes". At that point I refactor and continue.

The only thing that in my opinion should stop you from doing this is time constraints. Like it or not, sometimes you just don't have the time to do it.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:09

Leanan

732 ● 7 ● 13

> I see what you mean with "confused by it", but sometimes that makes it even worse for the next maintenance guy. He *did* understand the old code, and he's confronted with a diff that could have been a one-liner to handle a special case, but instead touches every line in the file... – Steve Jessop Sep 26, 2008 at 17:16

@onebyone if that's the case then you refactored incorrectly. Refactoring should always improve the maintainability of the code. – Wedge Sep 26, 2008 at 20:20

1 Right. But even if you increased code comprehensibility by 50% (in some hypothetical metric), someone who *already understands* the previous code, because they've seen it before, will not find the new code more comprehensible next time it's their turn to look at it. – Steve Jessop Sep 27, 2008 at 22:32

1 Furthermore, comprehensibility is important for diffs as well as for revisions. Refactors shouldn't be combined with functional changes if you can avoid it. – Steve Jessop Sep 27, 2008 at 22:37

4

It's like the National Parks -- Always leave it a little better than you found it.

To me, that means any time I open code, and have to scratch my head to figure out what's going on, I should refactor something. My primary goal is for readability and understanding. Usually it's just renaming a variable for clarity. Sometimes it's extracting a method -

For example (trivial), If I came across

```
temp = array[i];
array[i] = array[j];
array[j] = temp;
```

I would probably replace that with a swap(i,j) method.

The compiler will likely inline it anyways, and a swap() tells everyone semantically what's going on.

That being said, with my own code (starting from scratch), I tend to refactor for design. I often find it easier to work in a concrete class. When its done and debugged, then I'll pull the old Extract Interface trick.

I'll leave it to a co-worker to refactor for readability, as I'm too close to the code to notice the holes. After all, I know what I meant.

Share Improve this answer Follow

edited Sep 26, 2008 at 17:48

answered Sep 26, 2008 at 17:21

Chris Cudmore

30.1k ● 12 ● 59 ● 95

4

Refactor opportunistically! Do it whenever it's *easy*.

If refactoring is difficult, then you're doing it at the wrong time (when the code doesn't need it) or on the wrong part of the code (where there are better efficiencies to be gained elswhere). (Or you're not that good at refactoring yet.)

Saving refactoring for "maintenance" is a tautology. Refactoring *is* maintenance.

Share Improve this answer

answered Sep 26, 2008 at 17:53

Follow

catfood
4,331 ● 5 ● 32 ● 57

4

I refactor every time I **read** anything and can make it **more readable**. Not a major restructuring. But if I think to myself "what does this `List` contain? Oh, `Integer`s!" then I'll change it to `List<Integer>`. Also, I often extract methods in the IDE to put a good name of a few lines of code.

Share Improve this answer

Follow

answered Sep 29, 2008 at 20:12

Craig P. Motlin
26.7k ● 18 ● 103 ● 127

3

The answer is always, but more specifically:

- Assuming you branch for each task, then on each new branch, before it goes to QA.
- If you develop all in the trunk, then before each commit.
- When maintaining old code, use the above for new tasks, and for old code do refactoring on major releases that will obtain extra QA.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:01

Brian R. Bondy
347k ● 126 ● 602 ● 640

Everytime you encounter a need. At least when you're going to change a piece of code that needs refactoring.

2

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:01

VVS

19.6k ● 5 ● 49 ● 66

I localize refactoring to code related to my current task. I try to do my refactoring up front. I commit these changes separately since from a functional standpoint, it is unrelated to the actual task. This way the code is cleaner to work with and the revision history is also cleaner.

2

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:36

s_t_e_v_e

2,546 ● 3 ● 31 ● 35

Continuously, within reason. You should always be looking for ways to improve your software, but you have to be careful to avoid situations where you're refactoring for the sake of refactoring (Refactorbation).

2

If you can make a case that a refactoring will make a piece of code faster, easier to read, easier to maintain or easier or provide some other value to the business I say go for it!

Share Improve this answer

Follow

answered Sep 26, 2008 at 20:54

community wiki
James Bender

2

"Refactor early, refactor often" is a productive guideline. Though that kind of assumes that you really know the code. The older a system gets, the more dangerous refactoring becomes, and the more deliberation is required. In some cases refactoring needs to be managed tasks, with effort level and time estimates, etc.

Share Improve this answer

Follow

edited Sep 29, 2008 at 1:41

answered Sep 26, 2008 at 17:50

Chris Noe
37.1k • 22 • 74 • 92

2

If you have a refactoring tool that will make the changes safely, then you should refactor **whenever the code will compile**, if it will make the code clearer.

If you do not have such a tool, you should refactor **whenever the tests are green**, if it will make the code clearer.

Make small changes -- rename a method to make what it does clearer. Extract a class to make a group of related variables be clearly related. Refactoring is not about making large changes, but about making things cleaner minute by minute. Refactoring is clearing your dishes after each meal, instead of waiting until every surface is covered in dirty plates.

Share Improve this answer

Follow

answered Jun 23, 2011 at 17:31

Sean McMillan

10.1k ● 6 ● 56 ● 65

1

Absolutely as soon as it seems expedient. If you don't the pain builds up.

Since switching to Squeak (which I now seem to mention every post) I've realised that lots of design questions during prototyping fall away because refactoring is really easy in that environment. To be honest, if you don't have an environment where refactoring is basically painless, I recommend that you try squeak just to know what it can be like.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:02

Marcin

49.8k ● 18 ● 132 ● 206

Refactoring often can often save the day, or at least some time. There was a project I was working on and we refactored all of our code after we hit some milestone. It

1

was a great way because if we needed to rip code out that was no longer useful it made it easier to patch in whatever new thing we needed.

Share Improve this answer

Follow

answered Sep 26, 2008 at 21:51

Zee JollyRoger

399 ● 4 ● 15

1

We're having a discussion at work on this right now. We more or less agree that "write it so it works, then fix it". But we differ on the time perspective. I am more "fix it right away", my coworker is more "fix it in the next iteration".

Some quotes that back him up:

Douglas Crockford, Senior Javascript Architect Yahoo:

> refactor every 7th sprint.

Ken Thompson (unix man):

> Code by itself almost rots and it's gonna be rewritten. Even when nothing has changed, for some reason it rots.

I would like that once done with a task the code submitted is something you can come back to in 2 months and think "yes, I did well here". I do not believe that it is easy to find

time to come back later and fix it. believing this is somewhat naive from my point of view.

Edit: spelling error

Share Improve this answer

Follow

edited Dec 7, 2010 at 8:26

answered Nov 11, 2010 at 7:44

sonstabo

1,025 ● 1 ● 11 ● 25

0

I think you should refactor something when you're currently working on a part of it. Means if you have to enhance function A, then you should refactor it before (and afterwards?). If you don't do anything with this function, then leave it as it is, as long as you have something else to do.

Do not refactor a working part of the system, unless you already have to change it.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:03

ComSubVie

1,609 ● 1 ● 16 ● 20

0

There are many views on this topic, some linked to a particular methodology or approach to development. When using TDD, refactor early and often is, as you say, a favoured approach.

In other situations you may refactor as and when needed. For example, when you spot repetitious code.

When following more traditional methods with detailed up-front design, the refactoring may be less often. However, I would recommend not leaving refactoring until the end of a project. Not only will you be likely to introduce problems, potentially post-UAT, it is often also the case that refactoring gets progressively more difficult. For this reason, time constraints on the classic project cause refactoring and extra testing to be dropped and when maintenance kicks in you may have already created a spaghetti-code monster.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:04

BlackWasp

4,961 ● 2 ● 33 ● 42

0

If it ain't broke, don't refactor it.

I'd say the time to refactor belongs in the initial coding stage, and ou can do it as often and as many times as you like. Once its in the hands of a customer, then it becomes a different matter. You do not want to make work for yourself 'tidying' up code only to find that it gets shipped and breaks something.

The time after initial delivery to refactor is when you say you'll do it. When the code gets a bit too smelly, then have a dedicated release that contains refactorings and

probably a few more important fixes. That way, if you do happen to break something, you know where it went wrong, you can much more easily fix it. If you refactor all the time, you will break things, you will not know that its broken until it gets QAd, and then you'll have a hard time trying to figure out whether the bugfix/feature code changes caused the problem, or some refactoring you performed ages ago.

Checking for cbreaking changes is a lot easier when the code looks roughly like it used to. Refactor a lot of code structure and you can make it next to impossible, so only refactor when you seriously mean to. Treat it like you would any other product code change and you should be ok.

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:08

community wiki
gbjbaanb

0

I think this Top 100 Wordpress blog post may have some good advice. http://blog.accurev.com/2008/09/17/dr-strangecode-or-how-i-learned-to-stop-worrying-and-love-old-code/

Share Improve this answer

Follow

answered Sep 26, 2008 at 17:48

AlexForbes

link is now accurev.com/blog/2008/09/17/… – Sean McMillan Jun 23, 2011 at 17:23