Is copy-and-paste coding ever acceptable?

Asked 15 years, 11 months ago Modified 10 years, 8 months ago

Viewed 4k times

14

It's generally accepted that copy and paste programming is a bad idea, but what is the best way to handle a situation where you have two functions or blocks of code that really **do** need to be different in just a few ways make generalizing them extremely messy?

What if the code is substantially the same, except for a few minor variations, but those few minor variations aren't in things that are easy to factor out through either adding a parameter, template methods, or something like that?

More generally, have you ever encountered a situation where you would admit that a little copy-and-paste coding was truly justified.

design-patterns **copy-paste**

Share

Improve this question

Follow

edited Apr 2, 2014 at 17:50

Kara

6,206 ● 16 ● 53 ● 58

asked Dec 30, 2008 at 23:40

dsimcha
68.6k ● 55 ● 219 ● 340

it should be COPY and paste anyway ;-P – Ilmbus Dec 30, 2008 at 23:55

He he Good point Ilmbus. Is it cut'n'paste or copy/paste? – OscarRyz Dec 31, 2008 at 0:08

cut+paste code is the same as 'moving code'. If the discussion is about several chunks of almost-the-same code, then it should be a copy+paste discussion :) – gbjbaanb Jan 1, 2009 at 15:32

Related question - stackoverflow.com/questions/2490884/… – Oded Jul 25, 2010 at 17:46

14 Answers

Sorted by: Highest score (default)

29

Ask this question about your functions

"if this small requirement changes, will I have to change both functions in order to satisfy it?"

Share Improve this answer

Follow

answered Dec 30, 2008 at 23:44

Tim Jarvis
18.8k ● 10 ● 59 ● 95

+1. This is the most important aspect. The problem comes when developers just don't care and they say "Hey, here's what I'm looking for" Ctr+C, Ctrl+V. When the same piece is used in two different apps, there's no harm. But when it is used in the same app (worst of all in the same file) Ouch!
– OscarRyz Dec 31, 2008 at 0:13

This is a nice way to look at the issue and helps avoid getting stuck with a frozen habit one way or the other.
– Mark Brittingham Jan 8, 2009 at 16:10

14

Of course it's sometimes acceptable. That's why people keep snippet files. But if you're cutting and pasting code very often, or with more than a few lines, you should think about making it a subroutine. Why? because odds on you'll have to change something, and this way, you only need to change it once.

Middle case is to use a macro if you have such available.

Share Improve this answer

Follow

edited Dec 31, 2008 at 3:05

Andreas Grech
108k ● 101 ● 303 ● 362

answered Dec 30, 2008 at 23:44

Charlie Martin
112k ● 26 ● 196 ● 264

Yeap. I agree, but snippet files are one thing, but copy/pasting inside one single app is wrong. Mhhh of course you can't avoid repeat "print" for instance. Or for(int i = 0 ; i < arr.length ; i++) but I think these fall in a different category :-S +1 for a good point – OscarRyz Dec 31, 2008 at 0:00

12

I've heard people say the will copy and paste once (limiting duplication of code to at most two instances), as abstractions don't pay off unless you use the code in three places or more. () Myself, I try to make it a good habit of refactoring as soon as I see the need.

Share Improve this answer

Follow

answered Dec 31, 2008 at 0:59

Øyvind Skaar

1,840 ● 14 ● 22

3 Yes, the canonical book on Refactoring (by Martin Fowler) says "Three strikes and you refactor." If you're using something twice, it *might* be okay, but if you see something a third time, you *should* refactor. – ShreevatsaR Dec 31, 2008 at 3:14

I haven't read that yet, but I believe I picked it up from "Rapid Development" by McConnel. – Øyvind Skaar Dec 31, 2008 at 13:46

Then how do you keep track of what has been copied? – Ola Eldøy Jan 1, 2009 at 13:39

6

Yes, and it's exactly as you say; minor but hard-to-factor variations. Don't flagellate yourself if it's really what the situation calls for.

Share Improve this answer

Follow

answered Dec 30, 2008 at 23:43

chaos

124k ● 34 ● 306 ● 310

> Wow, I see this is a highly religious topic. I'm not trying to validate incompetent coders' failure to factor or write for reuse, people. I'm saying that sometimes, on vanishingly rare occasions, copy-paste-modify is what you need. – chaos Dec 31, 2008 at 0:05

5

Re Is cut-and-past ever acceptable:

Yes. When the segment is slightly different and you're doing disposable systems (systems that are there for a very short amount of time and won't need maintenance). Otherwise, its usually better to extract the commonalities out.

Re segments that look a like but not exactly alike:

If the difference is in the data, refactor by extracting the function and using the difference in data as parameters (If there are too many to data to pass as a parameter,

consider grouping them into an object or structure). If the difference is in some process in the function, refactor by using command pattern or abstract template. If it's still hard to refactor even with these design patterns, then your function might be trying to handle to many responsibilities on its own.

For example, if you have a code segment that differs in two segments - diff#1, and diff#2. And in diff#1, you can have diff1A, or diff1B, and for diff#2 you can have diff2A and diff2B.

If diff1A & diff2A are always together, and diff1B & diff2B are always together, then diff1A & diff2A can be contained in one command class or one abstract template implementation, and diff1B & diff2B in another.

However, if there are several combination (i.e. diff1A & diff2A, diff1A & diff2B, diff1B & diff2A, diff1B & diff2B), then you may want to rethink your function because it may be trying to handle too many responsibilities on its own.

Re SQL statements:

Using logics (if-else, loops) to build your SQL dynamically sacrifices readability. But creating all SQL variations would be hard to maintain. So meet half-way and use SQL Segments. Extract commonalities out as SQL segments and create all SQL variations with those SQL segments as constants.

For example:

```
private static final String EMPLOYEE_COLUMNS = "
id, fName, lName, status";

private static final String EMPLOYEE_TABLE = "
employee";

private static final String
EMPLOYEE_HAS_ACTIVE_STATUS = " employee";

private static final String GET_EMPLOYEE_BY_STATUS
=
  " select" + EMPLOYEE_COLUMNS + " from" +
EMPLOYEE_TABLE + " where" +
EMPLOYEE_HAS_ACTIVE_STATUS;

private static final String
GET_EMPLOYEE_BY_SOMETHING_ELSE =
  " select" + EMPLOYEE_COLUMNS + " from" +
EMPLOYEE_TABLE + " where" + SOMETHING_ELSE;
```

Share Improve this answer

Follow

answered Dec 31, 2008 at 0:55

Franz See

3,372 ● 5 ● 43 ● 49

Nice answer - well thought out with a lot of detail
– Mark Brittingham Jan 8, 2009 at 16:07

I like this, just because two things sorta look alike do not necessarily make them so. – Nathan Feger Jan 8, 2009 at 16:23

As Martin Fowler suggests,

do it once, fine.

4

do it twice, starts to smell.

do it thrice, time to refactor.

EDIT: in answer to the comment, the origin of the advice is Don Roberts:

Three strikes and you refactor.

Martin Fowler describes that in **Refactoring** chapter 2, section The Rule of Three (page 58).

Share Improve this answer Follow

edited Jan 1, 2009 at 15:20

answered Dec 31, 2008 at 2:52

icelava

9,857 ● 7 ● 53 ● 75

I think is Kent Beck on the "Refactoring" book. I don't remember. – OscarRyz Dec 31, 2008 at 16:09

Do it twice and it stinks already IMO. I've seen lots of code that started out as two small copies and then gradually involved into two monsters. (It's interesting looking back at the first version in source control. The copy-and-paste code often looks so small and innocent at the beginning.) Once they become monsters they are hard to refactor because they have so differences between them. Nip them in the bud I say. – dan-gph Feb 14, 2012 at 2:44

In my company's code base, we have a series of about 10 or so big hairy SQL statements that have a high degree of commonality. All of the statements have a common core, or at least a core that only differs by a word or two. Then, you could group the 10 statements into 3 or 4 groupings that add common appendages to the core, again with maybe one or two words different in each appendage. At any rate, think of the 10 SQL statements as sets in a Venn diagram with significant overlap.

We chose to code these statements in such as way as to avoid any duplication. So, there is a function (technically, a Java method) to build the statement. It takes some parameters that account for the word or two of difference in the common core. Then, it takes a functor for building out the appendages, which of course is also parameterized with more parameters for minor differences and more functors for more appendages, and so on.

The code is clever in that none of the SQL is ever repeated. If you ever need to modify a clause in the SQL, you modify it in just one place and all 10 SQL statements are modified accordingly.

But man is the code hard to read. About the only way to figure out what SQL is going to be executed for a given case is to step through with a debugger and print out the SQL after it has been completely assembled. And figuring

out how a particular function that generates a clause fits into the bigger picture is nasty.

Since writing this, I've often wondered if we would have been better off just cutting-and-pasting the SQL query 10 times. Of course, if we did this, any change to the SQL might have to occur in 10 places, but comments could help point us to the 10 places to update.

The benefit of having the SQL understandable and all in one place would probably outweigh the disadvantages of cutting-and-pasting the SQL.

Share Improve this answer

Follow

answered Dec 31, 2008 at 0:03

Clint Miller

15.4k ● 4 ● 39 ● 39

Having suffered similarly, I'd steer clear of the multiple SQLs - 10 statements means 10 chances to make a mistake. If the construction method is tough, maybe consider it an opportunity for further refactoring> – Mike Woodhouse Dec 31, 2008 at 13:58

In this case, refactoring would have meant restructuring our core tables. Given infinite time and money, that would be a great idea. But, it's just not practical. – Clint Miller Dec 31, 2008 at 14:43

1 +1 for the "but comments could help us", comments fix everything. – gbjbaanb Dec 31, 2008 at 15:38

How about making a view to hide some of the complexity from the application? – WW. Mar 15, 2009 at 21:37

ABSOLUTELY NEEEVER..

2

:)

You could post the code in question and see that it is easier than what it looks like

Share Improve this answer

Follow

answered Dec 30, 2008 at 23:44

OscarRyz

199k ● 119 ● 396 ● 573

> Subjectively speaking of course. I had had the worst times debugging copy/pasted code from others years ago. That and this: stackoverflow.com/questions/235474/#235523 are the worst thing had happened to me ever. :) – OscarRyz Dec 30, 2008 at 23:47

2

If it is the only way to do it, then go for it. Often times (depending on the language), you can satisfy minor changes to the same function with an optional argument.

Recently, I had an add() function and an edit() function in a PHP script. They both did virtually the same thing, but the edit() function performed an UPDATE query instead of an INSERT query. I just did something like

```
function add($title, $content, $edit = false)
{
    # ...
    $sql = edit ? "UPDATE ..." : "INSERT ...";
```

```
    mysql_unbuffered_query($sql);
}
```

Worked out great -- but there are other times when copy/paste is necessary. Don't use some weird, complicated path to prevent it.

Share Improve this answer

Follow

answered Dec 30, 2008 at 23:53

Logan Serman
29.8k • 27 • 105 • 144

2

1. Good code is reusable code.
2. Don't reinvent the wheel.
3. Examples exist for a reason: to help you learn, and ideally code better.

Should you copy and paste? Who cares! What is important is *why* you're copy and pasting. I'm not trying to get philosophical on anyone here, but let's think about this practically:

Is it out of laziness? "Blah blah, I've done this before... I'm only changing a few variable names.. done."

Not a problem if it was already good code before you copied and pasted it. Otherwise, you're perpetuating crappy code out of laziness which will bite your ass down the road.

Is it because you don't understand? "Damn.. I don't understand how that function works, but I wonder if it'll

work in my code.." It might! This may save you time in the immediate moment when you're stressed that you have a deadline at 9 a.m. and you're staring red eyed at a clock around 4 a.m.

Will you understand this code when you return to it? Even if you comment it? No really - after thousands of lines of code, if you don't understand what the code is doing as you write it how will you understand coming back to it weeks, months later? Attempt to learn it, despite all temptation otherwise. Type it out, this will help commit it to memory. Each line you type, ask yourself what that line is doing and how it contributes to the overall purpose of that function. Even if you don't learn it inside out, you might have a chance at recognizing it at the very least when you return to it later on.

So - copying and pasting code? Fine if you're conscious of the implications of what you're doing. Otherwise? Don't do it. Also, make sure you have a copy of the license of any 3rd party code you copy and paste. Seems common sense, but you'd be surprised how many people don't.

Share Improve this answer Follow

answered Jan 2, 2009 at 12:07

user50858

1

I avoid cut and paste like the plague. It's even worse than its cousin clone and modify. If faced with a situation like yours I'm always ready to use a macro processor or other

script to generate the different variations. In my experience a *single point of truth* is hugely important.

Unfortunately the C macro processor is not very good for this purpose because of the annoying quoting requirements for newlines, expressions, statements, and arguments. I hate writing

```
#define RETPOS(E) do { if ((E) > 0) then return; }
while(0)
```

but that quoting is a necessity. I will often use the C preprocessor despite its deficiencies because it doesn't add another item to the toolchain and so doesn't require changing the build process or Makefiles.

Share Improve this answer Follow

edited Jan 1, 2009 at 0:26

answered Dec 30, 2008 at 23:49

Norman Ramsey
202k ● 62 ● 371 ● 541

I don't know why you bold the text of your answers at seemingly random points but it makes it difficult to read.
– Robert Gamble Dec 30, 2008 at 23:56

For the record I didn't downvote you either but I might upvote some of your answers if they were easier to read ;)
– Robert Gamble Dec 31, 2008 at 0:29

There's an upvote because I really don't see why this should have a -2 vote. There is valuable information in this answer.

Readability is not THAT bad, granted it isn't great, but still ... – hmcclungiii Dec 31, 2008 at 1:04

@Robert, amdfan: I'm following Jakob Neilsen's advice to use bold text on the web for points of emphaiss. Maybe I should stop? (alertbox.com). – Norman Ramsey Dec 31, 2008 at 23:08

Yes, please stop. When 25%+ of your answer is bolded, it is very distracting and significantly affects readability. If you have a long answer and are worried about readers losing interest, put a 1-2 sentence summary at the top of the answer (not bolded). – Robert Gamble Dec 31, 2008 at 23:57

0

I'm glad this one is tagged as subjective because it certainly is! This is an overly vague example, but I would imagine that if you have enough code that is duplicated that you could abstract those sections out and keep the different parts different. The point of not copy-pasting is so you don't end up having code that is hard to maintain and fragile.

Share Improve this answer

Follow

answered Dec 30, 2008 at 23:46

sammich

335 ● 5 ● 13 ● 19

0

The best way (besides convert into common functions or use macros) is to put comments in. If you comment where the code is copied from and to, and what the commonality is, and the differences, and the reason for doing it... then you'll be ok.

Share Improve this answer

Follow

answered Dec 30, 2008 at 23:56

gbjbaanb

52.6k ● 12 ● 110 ● 154

0

If you find that you have functions which are mostly the same, but in different scenarios require slight tweaks, it is your design that is the problem. Use polymorphism and composition instead of flags or copy-paste.

Share Improve this answer

Follow

answered Dec 30, 2008 at 23:57

Ed Swangren

125k ● 24 ● 188 ● 268

1 Yeah, but if you throw tons of OO tricks at a simple problem, your API ends up being way too complex/bulky, and you end up with an overengineered POS. – dsimcha Dec 31, 2008 at 3:33

It's not a simple problem if you are copying and pasting code all around. how hard is it to create an abstract base class? – Ed Swangren Dec 31, 2008 at 8:28