Last updated: Aug 25, 2023

The Curse of Copy-and-Paste Code

Or, how you as a non-technical person can influence how software is developed, improve code quality, and get to know your software team a little bit better.

To: Product managers, Project managers, Directors, Vice Presidents, CEOs, and all non-technical stakeholders. Clients of outsourced consulting firms. Anyone with a vested interest to see their software project, large or small, succeed.

Cc: Developers, CTOs, Scrum Masters, Tech Leads, IT, Software Vendors, Consulting Firms.

Subject: Cursed Software Applications

If you're involved with software, chances are, your software — yes, *your* software; it's yours as long as you're even remotely a stakeholder — is cursed. Somewhere, deep in the codebase you're expecting to work, is a tiny, insidious flaw.

Who am I kidding? Your code is probably *rife* with flaws, big and small. One small problem can remain unnoticed for ages, until a corner case or a breaking change occurs to reveal it and then, hopefully, it's quickly resolved. But really, that's par for the course. All software has some issues in it; it's the degree to which those issues impact the overall product or business that is of interest to you, dear stakeholder.

No system is perfect. Software development (not unlike life) is a series of compromises that have to be made. Code choices are made based on the same factors that drive any project: how much can we do with the time, talent, and technologies we have?

That means all software has been written with less-than-desired concessions — as far as the developers are concerned. But for the rest of us? Such concessions probably don't matter. You don't see the pine framing of your house, hidden behind the drywall and paint after all, you trust that the building contractor knows what is proper; similarly there's rarely a need to care about how the software was actually made, trusting in your development team(s).

Except, sometimes, you do need to care. Or at least you need assurances. This is a story of one place where you should get such assurances.

The Problem

There is one particular type of software flaw that is dead easy to spot (if you have access to the codebase), seems inconsequential, and yet is truly insidious, a curse on your application waiting to arise when least expected.

Last updated: Aug 25, 2023

☠️ *Copy-and-paste code* ☠️

It's exactly what you imagine it to be.

Source code is just text. Anyone can read it (whether you can interpret it is a different story), and anyone can copy it.

The copy-and-paste curse starts when a developer copies code from source file X (or another project, or the Internet, or Chat GPT results) and pastes it into destination file Y (or another place in file X, even). They then tweak it, save it, run tests (if they wrote such beasts to begin with), submit for merging, and then move onto the next chunk of code.

Now there are two near copies of the same code, meaning you now have two places where the same thing can go wrong. The client wants to change the text of that sentence? Someone has to change it in two (or more) places. Customer found a bug on a specific webpage? Well, it may only be fixed in one file and not in the other file. Oops!

Changing a line or two of text is one thing, but scale these code copies from two to dozens, or even hundreds (which I have seen before), and say those copies are complex functions not just marketing text… then you have a potential disaster waiting to happen.

✂️ *Copy-and-paste in real life*

To give you a more concrete example, I worked on a project where there were four separate sections in the application, each very similar in behavior to one another. The user interface looked almost the same, plus or minus a feature here or there, and underlying logic that did the processing on each of the pages was very similar, plus or minus a function call to fetch from a URL or API.

The pages were close enough in appearance and functionality that one could write a single page and then make exceptions (as software lets you do) to that page as necessary by passing in various configuration parameters, all without duplicating anything.

Understand my frustration as a project manager then. Each week, I'd present a demo to the client, and each week, the client might have noticed a bug we'd have to fix. Fine, as we can fix a single bug here and there… but not so fine when in the next week the client notices the *same* bug on a *different* page.

Bugs (or design updates) were being updated in *one* place but not *all* places. Worse, sometimes bugs would be found and fixed, only to show up again in another place at another time, adding to the confusion. It was not a good look for us.

I'm a software developer as well as a project manager, so I personally took a look into the code. I couldn't have this continue on and show up again to the client next week to play whack-a-change. To

the horror of my very experienced eye, the code was awash with copy-and-paste. That was literally the main source of our problems (there were other issues, but this was the big one!).

In many cases, copy-and-paste code is *incredibly* obvious. Sections of code that are word-for-word the same as elsewhere. But sometimes you have to look beyond the names of variables and look at the "shape" of the code. It's like putting glasses on Superman and taking the cape away to make him into Clark Kent. New look, but same eyes, chin, and muscles.

In this situation, the variables had been renamed and some of the internal logic and display text was also changed. By all accounts though, it was basically the same code. Looking at it section by section, I could see the same data structures, the same for-loops, the styling code, the same overall layout, even a few shared misspellings.

It took me a week of intense work to refactor the code so that duplication was removed, sections were based on a single shared unit, all of the inherent problems were solved, and made it so that only the differences made a difference.

Imagine if this had been even a slightly larger application. Say a website with dozens, if not hundreds of pages? The amount of effort to correct would be astronomical; the impact of leaving it in a buggy state of over-duplication would be equally problematic.

Why You Should Care

For most people, software can come across as arcane and obtuse. It has jargon that you don't use in everyday speech, like "for-loops,", "data structures," "APIs", "closures," "encapsulation," or "idempotent."

But copy-and-paste is something everyone can understand. You understand from school that copying someone else's work and posting it as your own is plagiarism. Copy-and-paste code doesn't (usually) reach that level of legal liability, but it does introduce considerable risk to the project depending on how extensive it is (and the source of what one may have copied).

If you've ever wondered why when you've asked the development team to make a "small" change it seems to take way longer than expected, there's a decent chance it's because they had to make one change multiple times, then check those changes in, test, and hope that nothing got missed. The problem gets progressively worse over time because small changes to make an exception on one page (making it different from the original and other copies) add up.

Most copy-and-paste code is done blindly, with no consideration of the source (which could be incorrect to start), nor the consequences of blind pasting. The very nature of copy-and-paste means that there wasn't a *system* in place to begin with that defined how code is supposed to be added.

No system means no plan; in such a case developers are adding features and fixing problems piecemeal, not holistically. They're unable to test reliably (or they need to keep writing a whole lot of necessary tests to begin with) and the parts of the software start to become overly tangled with one another.

Regressions can also more readily occur. Yes, the coders fixed that one particular bug in *all* the places, but then two weeks later, someone undoes that change in just one of the copied locations (for whatever reasons, perhaps to add a new feature). Suddenly, an old bug you were sure was fixed rears its ugly head once again, right when you're showing the demo to an investor.

You should care because copy-and-paste bugs are very understandable and they are trivial to avoid by *simply not copy-and-pasting code.* That's all it takes. Developers just[1] need to *not* cultivate this bad habit; doing so means an entire category of bugs will never come to light. The cost savings alone can be enormous if measured by risk mitigation, not to mention the piece of mind that the code is at least semi-well written (hopefully better than that!).

Sources and Prevelance

Chances are, if you're the stakeholder of an open source project, things are okay, at least as far as copy-and-paste is concerned. Public scrutiny does wonders to make sure the most obvious of flaws don't creep in or are rapidly corrected in the code.

If you're on a growing and fast moving team, the more likely it is that copy-and-paste will show up, because developers take shortcuts when they're under pressure and copying the code you have right in front of you is very tempting compared to creating a new file, renaming some parameters, and thinking a little more deeply about the architecture (which is what the developers *should* be doing).

User interfaces are often full of copy-and-paste code, especially when it comes to styling[2]. "I need six green buttons." Create and copy once, paste five times, deed is done. The question is, what if you need to make those green buttons blue? What if you want to change the color scheme of the entire app?

Backend software, because there isn't the obvious visual repetition seen with UIs, is slightly less prone to copy-and-pasting, but that doesn't mean it's not a risk, especially when it comes to data structures. Sometimes identical functions will be copied from one file to another instead of creating a common utility file. This means that a change in one function won't happen in its twin. If it's a feature upgrade, you'll only see it in place. If it's a bug fix… you'll only see it fixed in one place.

---

[1] This is not an onerous request, developers. Your job is to get things as close to correct the first time; not run the risk of duplicating mistakes through the code to start with. Or, for that matter, make someone change something multiple times when it could have been done only once.

[2] Inline styling systems like Tailwind or Bootstrap are awesome if used correctly, except that's rarely the case. I see styles copied all over when inline like that. Styling is *still* code. Don't treat it differently, developers.

Every common layer of coding, from the user interface to the instructure has room for the copy-and-paste curse to be injected and cause problems.

🤖*Our robot overlords*

I haven't talked much about the source of copy-and-paste code, i.e, what is actually being copied to begin with. This is because the source of what is being copied isn't really important in this conversation; it's the act of *repeatedly pasting* that is the actual problem. Without duplicate code increasing the amount of risk, then copying code once from an external source to within the your software's codebase is just plain ol' development (and usually not a problem).

However, that doesn't mean that we should ignore where developers are getting inspired for code ideas and solutions. Most code snippets are innocuous (be it from Stackoverflow, Chat GPT, or elsewhere on the Internet), assuming there's a discerning eye doing the copying. But recognize that there is *some* risk that copied and unvetted code has inherent flaws in it; especially if 1) the copier is inexperienced or 2) if the code being copied is large or complex in nature, or 3) the copied code is already full of its own duplicates.

We're trying hard to not replicate other people's problems; if the copying developer already has a habit of copy-and-pasting, then that exacerbates the overall problem we're discussing[3].

🐘 *Elephant in the source code*

Okay, developers, I'm getting to you. If you're working on the minority of projects that *actually* wrote automated tests and *actually* runs them, say as part of pre-commit or CI/CD, kudos to you. If you have that much diligence in place, there's a decent chance that you're doing pretty well on avoiding the copy-and-paste curse. But not entirely.

You know what else can be copy and pasted beyond "traditional" code? Test code. Infrastructure code. Config files. Environmental variables. API keys and other secrets. Entire folder structures can be copied over, be tweaked, and then made to pass.

If you have one copy-and-paster on your team and nobody corrected them yet then there's a chance you have *more* than one, especially in a high stress environment. It's just too easy when you're freaking about making a deadline, with the project manager who doesn't know any better asking you to work a long weekend.

---

[3] Thankfully, obvious and highly replicated errors will quickly come to light, usually well before the code is submitted for merging and deployment (assuming your team has proper CI/CD systems in place). However, it's the more subtle issues that might be propagated that we should worry about, which is why it's worth the development team talking about good code hygiene.

When it comes to fast moving projects, something always gets dropped and most of the time, it's testing, which just exacerbates the problem of trying to fix copy-and-paste code.

Copy-and-paste can occur anywhere. Yes, there are decent reasons for copy-and-pasting, but they are few and far between and usually relegated to toy projects (and maybe microservices, but that's intentional duplication, not copy-and-pasting).

If a project, no matter the size, has even a *remote* chance of being taken seriously, developers should never be too lazy to make use of componentizing or dependency injection or any number of good design patterns. Copy-and-paste will *always* come back to bite; it's just a matter of when.

… and now back to everyone else.

What to Do

Unlike with developers, you — the non-technical stakeholder — can't easily comprehend nor have the time to casually peruse the software source code. Yet, there are things you can do about the dark pattern of copy-and-paste.

👀 *Pay attention*

First is to pay attention. You don't have to be technical to notice that small issues keep popping seemingly at random, or that bugs are regressing, or designs are complete in one place, but not another, stylistically similar, place.

Usually, it's a project or product manager/owner who is closer to the developers that might notice this first. They're hands on with the product and should be seeing these things and recording them in whatever issue tracking/project management software the project is using.

📈 *Get metrics*

Get those metrics. How many issues have regressed? In which part of the code (you should be able to ask this to be surfaced). How often? How long does a "simple" change take to make?

If a change is small (some text, a particular style), but seems to take a long time to adjust, the lead developer should be able to give a good reason as to why; and that reason shouldn't amount to a sheepish, "because we have to make the same change in half a dozen places."

Keep in mind that not all "small" changes are equal; sometimes what seems small to you isn't necessarily small to the developer and for good reasons. It all depends on where the code is located and how things are connected.

If the code is fine in terms of copy-and-paste levels, then the metrics might point to another problem so gathering such metrics is hardly a waste of time, but purposeful and unnecessary code duplication is a good suspicion to start with[4].

💬 *Ask to see the code*

Finally, just ask. If you're working on an internal project, ask to get a tour of how the code works. Be upfront and use this document as an excuse; "I heard that some codebases are full of copy-and-paste code. How does ours look?"

The earlier you can ask this question, the better. If you're not code-savvy, be prepared to be at least a little overwhelmed, but don't be afraid to ask "how does this apply to the copy-and-paste question?" a few dozen times.

If you've hired an outside consultancy, then *definitely* ask to see the code. You're paying for it after all. A good consultancy will make their source code accessible immediately at the start of the project via access to a repository (like GitHub). If you're not technical yourself, and if your vendor doesn't seem forthcoming, bring in someone else to take a look at the code (and probably find another vendor who is a bit more open to sharing).

🪨A segue into monoliths

So… you're having the conversation about copy-and-paste code and your lead developer is showing you around. Take stock of how long files are. You'll often see line numbers running down the left hand side of a file.

One side effect of copy-and-paste code is that files can become *extremely* long, well over 500 lines or more. These are typically called monolithic files. If you see a lot of monolithic code files, under most languages and file types for hand-written source code, that's a sign of an unhealthy codebase.

For a language like C, C++, Java, JavaScript or TypeScript, the source code of a file is often well under 200 lines (but not always!). It's usually better to have a lot of small, well-organized files, rather than a few monolithic files. Changes to monolithic files are at high risk if the developer doesn't fully understand how it's written.

Don't panic — There are a lot of reasons why files will be over-sized. Ask what's going on first.

Generated files, for example, concatenate a lot of information, grossly exaggerating the size of the output. Files that deal with complex algorithms may be long, as well as library and type files. But generally speaking, the code that the developers touch on a day to day basis should be somewhat restrained in size.

[4] To be fair, *all* software has some duplicated code in it, if only by coincidence, so things can still slip past. That's not the same as the copy-and-paste curse, though.

Last updated: Aug 25, 2023

Make the Change

This curse can happen to *any* company, large or small and even under the guise of modern software practices. Agile won't protect you; test driven development can be sunk past; CI/CD might be able to catch something… but probably not. All it takes is one developer to slip up and make the copy happen, perpetuating the problem.

⚙️*Automated Code Analysis*

While there are static code analyzers out there that will check your codebase for copy-and-paste problems, I don't have a lot of personal experience with them; I think they'd be great for checking code that such as from an external vendor where your project doesn't have the time or possibly the domain knowledge to check the codebase.

🤝 *The Code Agreement*

Within your organization I highly recommend sitting down and talking to the leadership of your software teams. In any software company, department, or team, there should be a common shared manifesto between the developers on how to write code that is written down, agreed upon, and followed.

It doesn't have to be long — in fact, it *shouldn't* be very long. Most of the actual "agreements" are built into the tools and processes the developers are using with automated checks and balances (such as code formatters, pre-commit checks, test driven development, and CI/CD).

It should, though, cover the basics of how the team works such as (but not limited to): how they communicate, how code is reviewed, how features are documented and bugs tracked, to what degree warnings are accepted (conventionally, that's never), and anything that is specific to the "personality" of the group as a whole.

If you look at (or create) enough of these sorts of docs, you'll find that they often have very similar items in them, but each is still customized to some degree, because each member, each team, each company, and the circumstances of development can of course be different.

One of the most common things I've seen in code manifestos, especially if written by experienced developers, is, "Code must be DRY" — where DRY means "don't repeat yourself," i.e., don't copy-and-paste code or cause code to be unnecessarily duplicated.

It's a phrase prevalent in these sorts of code agreements because the problem of copy-and-paste is so prevalent (and so easy to indulge) in codebases. DRY is a foundational best practice, yet even written down, some developers forget (or refuse, as the case may be) to follow it.

🏯 *A Cultural Shift*

If your developers don't have such a document, encourage them to create one. The very act of thinking through a team's best practices, and then later updating it as they gain new experiences, is invaluable to the process of creating quality code.

If your leads are reporting to you that someone isn't following through on DRY (or any part of the agreed upon manifesto) and they keep submitting poor code, it may be time for an intervention. Usually the issue is subtle; they simply don't see that they're repeating themselves and it takes a more experienced developer to show them the error of their ways. A bit of mentoring can go a long way.

But I've worked on a few projects where copy-and-paste was an issue of outright recalcitrance. It seems so silly at first blush, but what I've discovered is that some developers are simply not confident enough in their own coding skills to write better code, or they feel under such time pressure to perform, they revert to what is easiest, and that is often copy-and-pasting.

In either case, not only does there probably need to be some time for education, but there also might need to be a look at how projects are managed in general. If under pressure junior developers keep slipping up on simple things, despite being told otherwise, what pressure are the more experienced developers going through? It's always good to take a step back and have a breather, maybe even before the normal planned retrospectives.

The boons for creating a best practices document are fantastic. It helps the team(s) self-organize, and if followed, not only will it improve overall code quality, it also greatly limits the insidious curse that is at the heart of this conversation, copy-and-paste.

Conclusion and Banishment

The premise of this document is that copy-and-paste code is a very poor, negatively consequential, development pattern that should be avoided.

One goal of this document was to show the nature of those consequences, such as:

- Unnecessary code complexity.
- Risk of missing features or code improvements.
- Risk of defect regressions.
- Incomplete bug corrections.
- Indications of other code-related deficiencies.

The copy-and-paste curse is an easily solvable problem; there is a distinct and noticeable pattern that anyone, even non-technical/non-developers can see. If you've copied text from one document

and pasted to another, you know exactly what's happening. Developers can easily avoid the problems related to the copy-and-paste curse by simply not doing copy-and-pasting to begin with.

We also explored that there are reasons that perhaps a developer might engage this behavior; often it relates to the fact that developers are under a lot of pressure to deliver and will revert to the simplest and quickest solution in the moment (a lot of technical debt occurs this way).

It's good to keep open communications between the project manager and developers so that pressures like that can be alleviated. Ultimately, the cost of correcting a bug that came out of an under-pressure, poor-judgment situation is much more costly than letting the developer take the time to generate the correct solution in the first place.

For the non-technical audience, much of software development happens behind closed doors, so the other goal of this document was to demonstrate that there are things you can understand about how coding works and that there are places where you can apply positive influence, even if you can't get into the finer grained details.

The copy-and-paste curse is very much an avoidable problem. It's usually the result of poor habits or undue pressures, both of which are easily corrected through education and improved communications and processes.

Make the change. Banish the curse. No more copy-and-pasting!

🙏 Thank you

Thank you so much for taking the time to read this far. Whenever I write a long-form paper, I'm reminded that we live in a short-form world. Bullet points, TL;DR, tweets. It's easy to think that no one really reads beyond the fold or the first paragraph anymore, but you've shown that simply isn't true and for your efforts, I'm truly grateful.

I hope that this article has likewise been useful to you and that you can make use of the concepts within your organization. My goal has always been to make the struggles of running even the most complex and sophisticated business easier. My approach is always to view the details of the daily grind and see how tiny fixes might be amplified into greater, more holistic solutions for the wider organization. Much of my experience in business has taught me that positive corrections to the human elements have the greatest impact on the technical and business elements, more so than just adjusting proverbial knobs and seeing what happens.

Last updated: Aug 25, 2023

I'm equally available for conversation or consultation and definitely on software and operational topics far beyond fixing coding issues. Feel free to reach out to me at any of the following channels:

- https://www.linkedin.com/in/adamkecskes/
- adam@kecskes.net
- +1 512-662-2969 (text me first!)

Thanks again for your time and energy. Best of luck on growing your business to new heights!