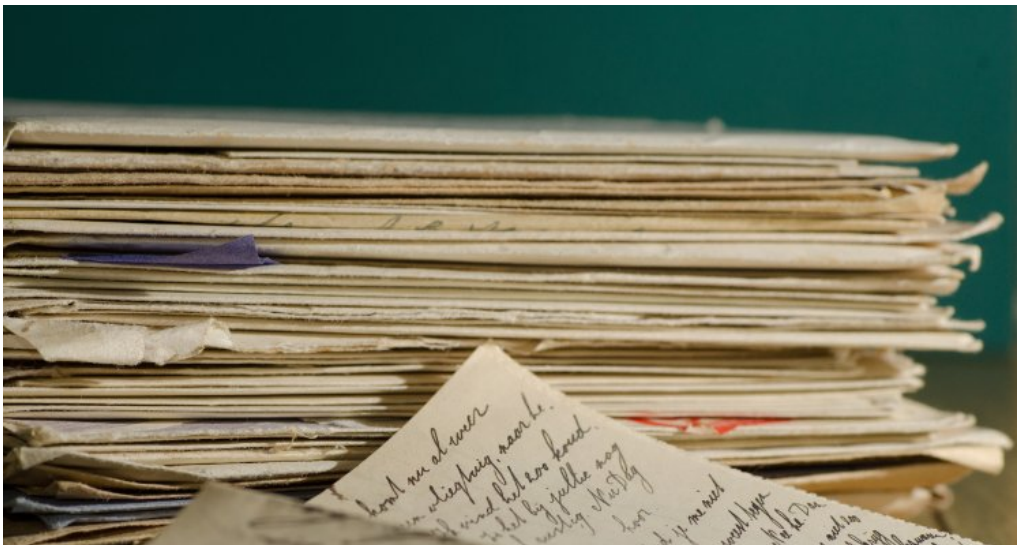

Avoid these 35 habits that lead to unmaintainable code

Published on May 11, 2017 | Featured in: Software Engineering


Christian Maioli Mackeprang | Follow
Senior Web Developer

Bad habits are hard to break and even harder if you don't realize that what you're doing is undermining your work. If you know but don't care—that would be the worst. But you're here, aren't you?

As a programmer, I've seen a lot of poor practices, not just around code, but also around teamwork skills. I've been guilty of practicing many of these bad habits myself. **Here are my top 35 bad programming habits, organized into four categories: code organization, teamwork, writing code, and testing and maintenance.**

Code organization

1. Saying "I'll fix it later", and never doing it

The habit of postponing code fixes is not merely a problem of priorities. Organizing your issue tracker might generate some progress, but you also need to have a way of tracking smaller issues that come up. Adding "TODO" comments is a quick way of making sure you don't miss anything.

2. Insisting on a one-liner solution

Being obsessive about writing efficient, elegant pieces of code is a common trait of programmers. It's like solving a puzzle—you find a combination of functions and

regular expressions that turn 20 code lines into 2 or 3. Unfortunately, it doesn't always result in readable code, and that's generally the far more important outcome. Make your code accessible first, then clever.

3. Making pointless optimizations

Another place where we often misplace our efforts is optimizations. It sounds great to reduce the size of your website a few bytes, but won't gzip make up for it anyway? And aren't requests more important? Address optimizations at the end of a project, because more often than not, requirements will change, and your time will have been wasted.

> *"Premature optimization is the root of all evil."*
> *—Donald Knuth*

4. Convincing yourself that styling issues are not that important

If I've learned anything over years of looking at other people's code, it's that dealing with coding style issues is the thing that developers are most likely to postpone. Maybe it's hard for inexperienced coders to see what good will come out of addressing styling issues, but over time it will become evident that once code quality derails, a snowball effect will turn any project into a complete mess. Be strict about best practices even if they seem negligible. Set up code checking and linting tools to give yourself space to worry about the more important things.

5. Sweeping things under the rug

Either by catching and ignoring exceptions, or by using libraries that don't report errors (such as jQuery), there are many ways to sweep things under the rug. But when one of those errors becomes a priority, the challenge of fixing it will be many times greater, considering that you won't have a clue where to begin. An easy way to avert this is by logging those ignored errors so you can study them later.

6. Using names that don't add information

Naming is hard, but there's an easy way to make sure your variable and function names are at least of decent quality. So long as the names add some kind of information that the rest of the code doesn't convey, other developers will have an easier time reading your code. The reason that naming is so important is that names can give a general idea of what the code does. It takes more time if you need to dig into the calculations to figure out what piece of code does, but a good name can help you understand what the code does in seconds.

7. Ignoring proven best practices

Code reviews, test-driven development, quality assurance, deployment automation—

these practices, and several others, have proved their value in countless projects, which is why developers blog about them constantly. A great reference for these best practices is the book Making Software: What Really Works, and Why We Believe It. Take the time to learn how to do them properly, and your development process will improve in all of your projects in ways that will surprise you.

Teamwork

8. Abandoning plans too early

A sure-fire way for making your system inscrutable is to not commit to a plan. You can always say, whenever your code is criticized, that the plan isn't complete. However, having half-done modules will lead to tightly coupled code as soon as you try to make those unfinished modules work with each other. This kind of complication also comes up when a project's leadership roles change and the new leads decide that having it their way is more important than architectural consistency.

9. Insisting on a plan that has little chance of working

Just as abandoning your plans can cause problems, so can sticking to a plan that doesn't work. That's why you should share your ideas with your team to get feedback and advice when things get tricky. Sometimes a different perspective can make all the difference.

10. Working on your own all the time

You should strive to share your progress and ideas with the team. Sometimes you think you're building something the right way, but you're not, so constant communication is very valuable. It's also beneficial for other people when you work with them. Their work often improves when you discuss ideas with them and mentor the less experienced members of your team, who are more likely to get stuck.

11. Refusing to write bad code

There comes a time in every developer's life when deadlines will force you to write terrible code, and that's okay. You've tried warning your client or manager about the consequences, but they insist on sticking to the deadline, so now it's time to code. Or perhaps there's an urgent bug that can't wait for you to come up with a clean solution. That's why it's important to be versatile as a programmer and to be able to write poor code very quickly as well as good code. Hopefully, you can revisit the code and pay back the technical debt.

12. Blaming others

It's no secret that arrogance is an all-too-common trait among developers and other technical professionals. Taking responsibility for your mistakes is a virtue that will

make you shine among your peers. Don't be afraid to admit that you've made a mistake. Once you're okay with that, you will be free to focus on learning why you made that mistake and how to avoid it. If you don't own up to it, learning becomes impossible.

13. Not sharing with your team what you've learned

Your value as a developer is not only placed on the code you write, but also on what you learn when writing it. Share your experiences, write comments about it, let others know why things are the way they are, and help them learn new things about the project and

One of the most valuable character traits of any craftsman lies in making sure that everyone is on the same page about the work, as much as possible. The reason for this is not so that your manager call fill spreadsheets. It's for your own gain as well: You will have fewer insecurities and reduce uncertainty about the lifetime and future of the project.

15. Not using Google enough

The best way of solving a complex problem quickly is not having to solve it at all. When in doubt, Google it. Of course, you can bother the engineer next to you instead, but rarely will he be able to give a response as detailed as Stack Overflow, not to mention that you'll be interrupting his work as well.

16. Overvaluing your personal style

Always aim to coordinate your working style and environment setup with your team. Ideally, everyone on your team should be working under similar conditions and following the same coding style. Doing things your way can be more fun, but co-workers might not be used to your coding style, and if it's unusual, it will be harder for the next developer to work on what you've built.

17. Having a personal attachment to your code

When someone comments on your code, don't take it personally. Your code should stand on solid ground; that is, you should be able to explain why you wrote it that way. If it needs improvement, that's only a reflection of the code's correctness, not of yourself.

Writing code

18. Not knowing how to optimize

A good optimization strategy takes some experience to get right. It takes exploration, analysis, and knowing every system involved in a process. Inform yourself about these

things. Learn about algorithmic complexity, database query evaluation, protocols, and how to measure performance in general.

19. Using the wrong tool for the job

You can only know so much, but the reason why you have to keep learning is that each new problem brings a different context and requires a different tool—one more applicable to the task at hand. Be open to new libraries and languages. Don't make decisions based strictly on what you know.

20. Not bothering with mastering your tools and IDE

Each new hotkey, shortcut, or parameter you learn while using the tools you work with every day will have a more positive effect on your coding speed than you realize. It's not about saving a few seconds by using a hotkey; it's about reducing the context switching. The more time you spend on each small action, the less time you'll have available to think about why you're doing it and about what comes next. Mastering shortcuts will free your mind.

21. Ignoring error messages

Don't assume that you know what's wrong with your code without even reading an error message, or that you'll figure it out quickly enough. Having more information about a problem is always better, and taking the time to gather that information will save *more* time in the long run.

22. Romanticizing your developer toolkit

Sometimes your preferred editor or command line tool isn't the the best tool for the job at hand. Visual Studio is great for writing IDEs, Sublime is great for dynamic languages, Eclipse is great for Java, and so on. You might love vim or emacs, but that doesn't mean that it's the right tool for every job.

23. Hardcoding values instead of making them configurable

Always be thinking about what changes might come and how to deal with them. Technical debt will grow at a monstrous rate if you don't separate the moving pieces from the rest of your work. Use constants and configuration files where appropriate.

24. Reinventing the wheel all the time

Don't write code you don't need to. Perhaps someone else has spent a good deal of time on your problem already, and he or she might have a well-tested solution that you can reuse. Save yourself some trouble.

25. Blindly copy/pasting code

Understand code before you reuse it. Sometimes you don't immediately notice everything the code is doing on first glance. You will also learn more about a problem when you take the time to read the code in detail.

26. Not taking the time to learn how things really work

Always take the opportunity to expand your knowledge by thinking about how things work and reading about their underlying issues. You might save time by not bothering right now, but what you learn on a project will be more important in the long term than actually getting it done.

27. Having excessive confidence in your own code

It's dangerous to assume that just because you wrote something, it must be great. You learn more about programming as you work on new things and gain experience, so take a look at your old code from time to time and reflect on how you've progressed.

28. Not thinking about the trade-offs of each design, solution, or library

Every product has its fine points that you'll only learn about by using and analyzing it. Seeing a few usage examples for a library will not make you a master of it, nor does it mean that it's the perfect fit for every situation that will come up in your project. Be continually critical of everything you use.

29. Not getting help when you're stuck

Keeping a short feedback loop will always be less painful for you. Asking for help doesn't mean that you're incompetent. The right people will see your effort and admission of ignorance as a drive to learn, and that's a great virtue to have.

Testing and maintenance

30. Writing tests to pass

Writing tests that you know will pass is necessary. They will make refactoring and reorganizing a project much safer. On the other hand, you also have to write tests that you know won't pass. They are necessary to move the project forward and keep track of issues.

31. Disregarding performance testing for critical cases

Prepare an automated performance testing setup at about the middle point of a project's development process so you can make sure you don't have escalating performance problems.

32. Not checking that your build works

It's rare when a build passes but doesn't really work, but it can happen, and it might be troublesome to fix the problem the longer you wait to look into it. Quickly testing every build is an important habit to have.

33. Pushing large changes late, or leaving after making a large push

This is where overconfidence will get you, and it can take getting burned multiple times to learn why you shouldn't do this, so take my advice now and make sure you are always there when your build eventually breaks.

34. Disowning code you wrote

Be willing to support code you wrote. You are the most suitable person for helping others understand it. You should strive to make your code remain readable to yourself and others many years from now.

35. Ignoring the nonfunctional requirements

When you're trying to deliver something, it can be easy to forget about some important areas such as performance and security. Keep a checklist for those. You don't want them ruining your party because you drew up your deadlines without thinking about these nonfunctional concerns.

What are your worst programming habits?

As it's often said, we are creatures of habit. Improving the way you work through habits is a great way to avoid having to think too much about every single situation. Once you've assimilated a good way of doing something, it becomes effortless.

Note: this post was originally featured on TechBeacon.

Want to read more of my articles? Sign up to my newsletter!

Christian Maioli Mackeprang
Senior Web Developer
7 articles

Josh H. ··· 9m

Let's build a faster Web

@fixme Number 8 out of tune with the 4 core values of the Agile Manifesto

Like Reply

Larry Mayer ··· 1d

Quality Assurance Engineer at Serengeti Law

I'm not a developer - but I know where I work our Developers are always pressed to meet impossible deadlines - and they often succeed! how? By writing an "an inelegant but effective solution to a computing problem" - FOR NOW. The Product Owner / Manger never asks "how is the architecture of your code?" The question is always "when are you going to be done? People do… See more

Like Reply | 1

There are 56 other comments. Show more.

Don't miss more articles by Christian Maioli Mackeprang

The most neglected programming constructs

Christian Maioli Mackeprang on LinkedIn

How terrible code gets written by perfectly sane people

Christian Maioli Mackeprang on LinkedIn

Most project managers ignore a huge aspect of task estimation

Christian Maioli Mackeprang on LinkedIn

Looking for more of the latest headlines on LinkedIn?

Discover more stories