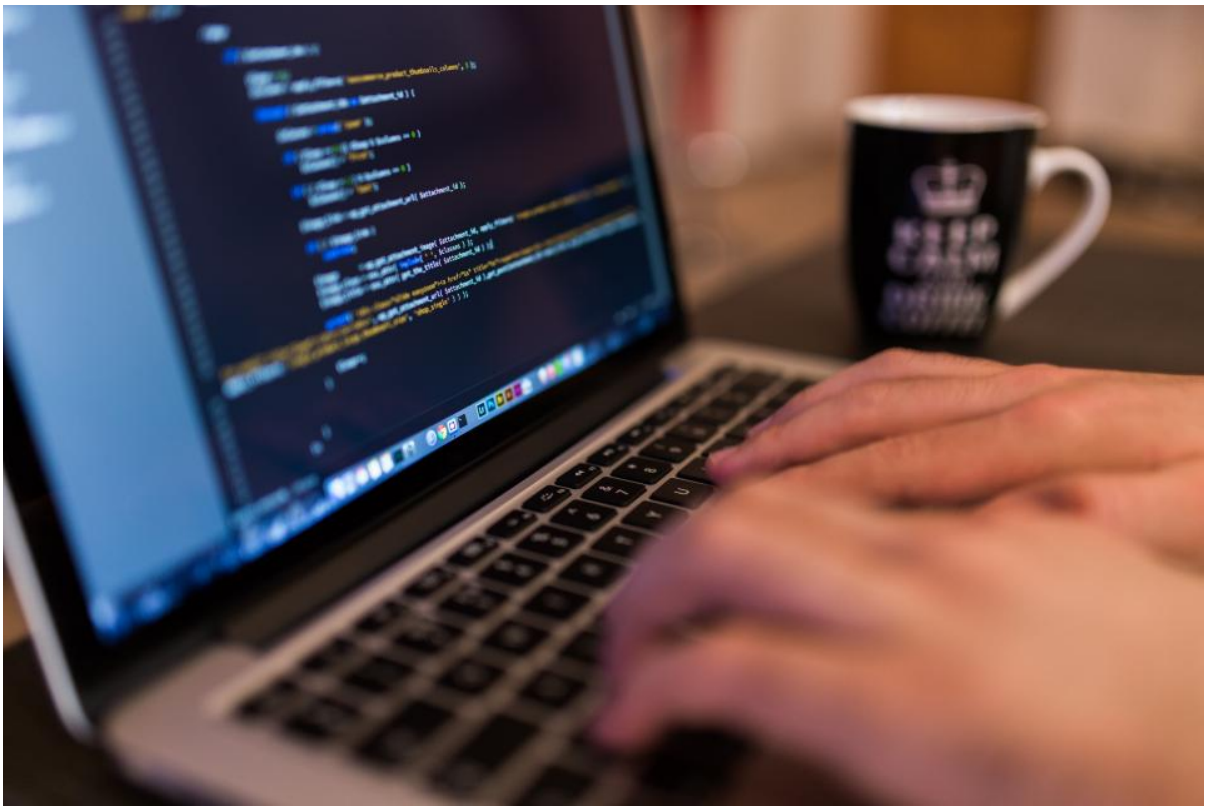
codementor.io

Are You a Bad Developer? Take This Quiz to Find Out! | Codementor

Codementor



There's a saying that "a great programmer can be as 10 times as good as a mediocre one".

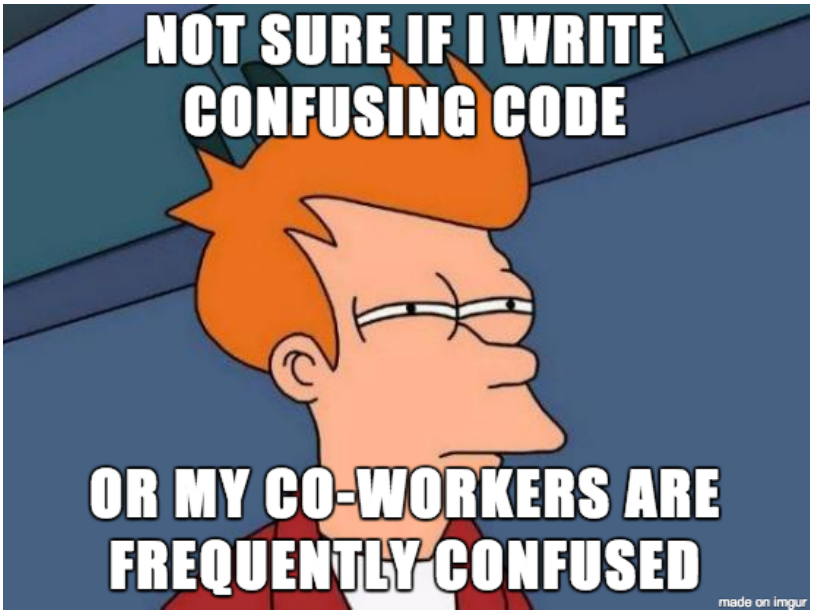
No one really wants to be labeled as a bad developer, but the sad reality is that a lot of developers aren't even aware that they're bad.

So, the question is: Are you a bad developer? Take this quiz to find

out!

Ok. Regardless of what results you got, positive or negative, don't get too happy yet as there are some key determining factors not included in the quiz.

Bad Developers



chances are, you're writing confusing code

If you're still a beginner to coding and you're worried if you're producing bad code, in terms of ability you are naturally not good right now. However, don't feel discouraged as there is one major characteristic that makes a developer bad, and as long as you don't fall into that trap, you have room for improvement (i.e. You're green, not bad).

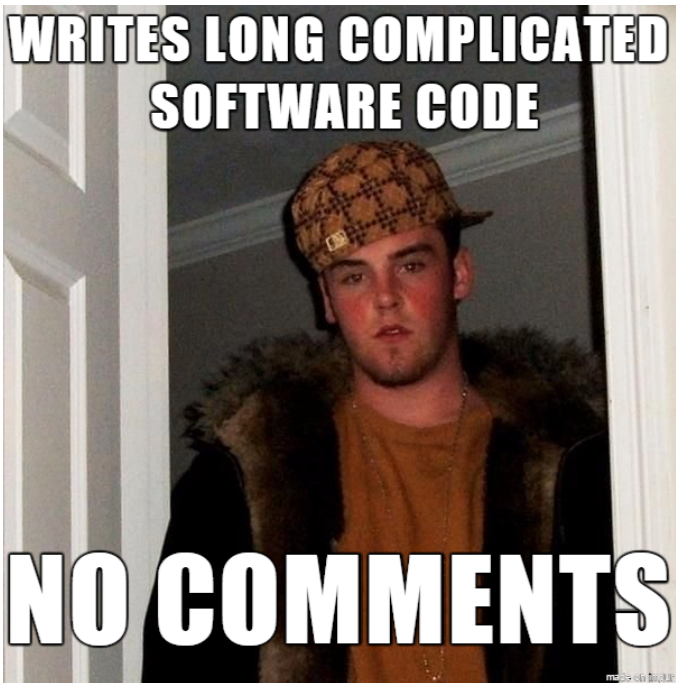
That said, let's first understand what are the two main types of bad developers:

- The Cowboy/girl Coder (for the sake of reading ease let's just use "cowboy" when referring to this type)

- The Mediocre Dev

At the core they are the same, but they usually exhibit different behaviors.

Cowboy Coders



Cowboy coders would destroy a team and they work best by themselves and on a single project with a short life-span.

Self-taught coders who never received any guidance with how to write usable code are usually in danger of becoming one of these, and many good, experienced programmers have likely been a cowboy at the beginning of their coding careers. So what are key attributes of a cowboy coder?

1. Codes Very Quickly

Usually these types of bad devs can churn out new features far more quickly than the average dev, and unfortunately people who don't know code would think these speedy coders are awesome (which only further bolsters the cowboy's ego). These devs work best alone and for clients with extremely tight deadlines and who are only looking to get feature out as soon as possible.

Cowboy coders code very fast because they usually code on the fly – meaning, they code without any planning for future maintainability. Which leads to…

2. Messy, Unreadable Code

The code design of quickly built projects would be a complete mess (or rather, code design is non-existent). This sort of messy code is often referred to as "Spaghetti Code", which is not at all as tasty as it sounds.

it's called Spaghetti because everything is jumbled together and impossible

to separate

Spaghetti code is difficult to understand and is usually unnecessarily large and complex to the point where others will find it difficult to understand what the programmer does, and thus it is usually a nightmare to maintain. This means decreased productivity for the whole team if anyone is unfortunate enough to have to work with a cowboy coder.

The result of messy code is…

3. Bugs. Bugs, Everywhere

If a company's software grows larger and more complex and their code is still a pile of spaghetti, then it's just a ticking bomb waiting to explode. At the worst it would cause problems as severe as Toyota's unintended acceleration. Everyone can agree that the

Toyota car recall was a disaster.

It's also never pleasant if your software happens to enter the hall of shame at the Daily WTF.

As the Joke goes: "99 little bugs in the code 99 little bugs in the code Take one down, patch it around 117 little bugs in the code" (source)

What's more, spaghetti code is not extensible. This means that adding new features to a Spaghetti code is like walking in a minefield that will explode, no matter how large or small the step and what direction you take. This is usually because a cowboy coder jumbled every functionality together, so any change would break the software altogether. This could be prevented with better code design and/or unit tests, but of course, cowboys don't give a care for whether their code is usable and also don't care for writing tests (something that takes time). Not to mention, with the way their code is structured thanks to bad design decisions, it's hardly going to be testable or even debuggable anyway. What usually happens with a cowboy coder is they quickly "fix" some bug, only to create more bugs. They'd likely feel like busy, heroic firefighters who never actually put out the source of the fire.

All in all, every bug and error created by a bad developer would cause negative productivity. At first it seems that this cowboy is

being super productive by always meeting deadlines other developers won't dare to promise, but this is at the cost of loads of "unexpected" errors that could have been prevented by well-designed and clean code programmed by a good dev.

If you're spending more than 80% of your development time debugging your own code and if your code is a nightmare to debug (i.e. you end up creating another bug), this usually means the codebase is not good and you may need help with improving your code.

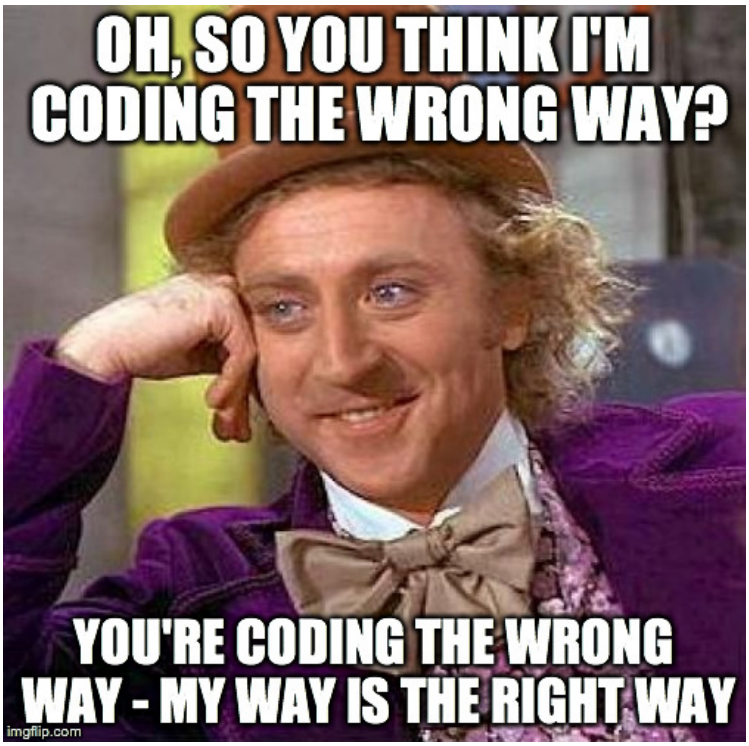
Arrogance

Sometimes cowboy programmers are not bad because they want to be, and they're simply cooking up spaghetti code because management/clients had impossible deadlines (though as the saying goes, any developer who takes pride in their code would GTFO those companies or decline such clients). Many beginners and junior devs go through the mistake of coding without planning and producing a bunch of buggy code, sometimes because they have less experience with these problems so they make bad decisions.

These beginners can be easily straightened up by receiving mentorship from experienced devs who take pride in building quality code, as a lot of times beginners are cowboys because they didn't know better. However, if they're surrounded by equally bad or mediocre developers, then they would be in trouble of falling into the delusion that they're good.

As long as you are willing to take responsibility for your mistakes and as long as you are learning from your mistakes, you're not a bad developer

The most important attribute that makes these programmers bad is **arrogance**.



the typical attitude of an arrogant programmer

Bad programmers think their code is perfect and would blame customers for being stupid and for crashing their program rather than reflect on why their software crashed. Cowboy coders are usually selfish devs who don't have a shred of empathy for others who have to clean up after all the problems they've created.

What's more, these arrogant programmers also think others are beneath them in terms of intelligence. They'd usually assume people who need comments and who don't understand their code

are too dumb to work with them, but never try to think about why people don't understand their code. As a result of always thinking they're right and always thinking other people are inferior, they are uncommunicative when they build features, which can cause a lot of problems for a team. Some may think they're so good that they would sometimes shun "best practices" or "standards" as they assume their own code is better (without good reason).

Worst of all, bad programmers are unwilling to listen or learn from mistakes because they don't acknowledge that they've made mistakes—as mentioned before, they usually play the blaming game instead.

Do note that this doesn't mean cowboy coders are difficult people or jackasses in real life—they could be the nicest person you've met—but this arrogance and unwillingness to take responsibility for mistakes is usually ingrained in the mental attitude they take whenever faced with criticism.

Any decent developer who has had the misfortune of working with a bad programmer likely has a load of horror stories to tell about these types of programmers.

The Mediocre Dev

No, I'm not talking about the "mediocre developer" in terms of what was referenced in Jacob Kaplan-Moss's speech at PyCon 2015 on "the Programming Talent Myth".

The mediocre we're referring to here is the "barely adequate" mediocre.

In some ways, the mediocre dev is worse than the cowboy coder because they know they're not great, but they are usually content with staying at the bottom of the ladder in terms of skill.

Unlike cowboys, mediocre devs usually lack in an interest in programming altogether and thus have difficulties with understanding programming concepts. They take a long time to build something, but the code they produce is still subpar and filled with problems. They usually have no passion/interest in coding at all, and they are slow to learn new technologies or they're practically untrainable.

Maybe mediocre devs aren't as destructive as cowboys because they will play in a team, but they're definitely not bringing anything to the table and their solutions will always be worse than good developers (they'd usually create a lot of buggy/inefficient code as well due to many bad decisions).

There's not much more to say about mediocre devs. At worse they

might also be pasta chefs who drag down the entire team, and at best they're just barely making it to the finish line.

The Core of the Problem

At the core of what makes a developer bad is the lack of the desire to become a better programmer. Bad programmers are satisfied and comfortable with the way things currently are. Worse, both cowboys and mediocre coders usually think they know what they actually don't know.

you've all likely seen this meme. While this happens a lot, there's a problem if you don't then proceed to figure out "why"

What's more, a bad programmer is someone who is not interested

in learning what they don't know, and thus not interested in improving themselves.

This is also why you'd usually find copious amounts of copy & pasting in a bad programmer's code, as they make zero effort in figuring out why something works or doesn't work—they just want the fix. Copy & pasting isn't inherently bad, but only under the following circumstances:

- You know what you're doing (though many bad developers would think they know what they're doing)
- You're sure that the code you're copying & pasting will work
- It's only for testing/trialing

Bad developers would usually copy & paste StackOverflow code without understanding it or tweaking the solutions to fit their own code.

This lack of curiosity of how a code works will cause bad developers to have a superficial understanding of the language/tools/libraries they use. Incorporating libraries/packages /what-have-you without reading the source code is also kind of similar to copying and pasting. As Jeff Atwood, the co-founder of StackOverflow says, "Read the Source Code, Luke". How can you understand how your code works if you don't even understand the tools you use? Of course, if you're a beginner to programming, it may seem daunting at first to absorb all the high-level information

about **why** things work this well, but simply accepting things as they are and knowing how to use syntaxes is the wrong attitude to take when learning how to code.

On that note, **people who always insist on following "best practices" without understanding why those practices are considered "best" can also be categorized as bad programmers**.

All in all, these guys are also called "cargo cult programmers ", or "Programmers who understand what the code does, but not how it does it." Perhaps you don't need to know every detail of how a large, complex framework works. However, you should at least figure out how the part **you're** using works.

In addition, bad programmers don't ever seem to learn from their mistakes, either because they don't acknowledge they've made a mistake or because they have a lack of desire to learn, or the combination of the two.

It's ok to make mistakes and create bugs, because everyone makes mistakes. However, if you continue to repeat your mistakes, this means you're not learning and that makes you a bad developer.

Want to speed up your learning process? Check out our Codementor Live Classes

Good Developers

After rambling about bad developers for long enough, you probably already have an idea of what makes a good developer. Good developers should make up the bulk of the development workforce, and they usually have the following characteristics:

Awareness that there is always going to be a better developer Humbleness and a willingness to take responsibility for mistakes as well as to learn from mistakes Writes readable, structured code Solid code design that can be debugged easily Strives to understand how things work Communicates/cooperates well with others in a team Open to criticism and different approaches Able to keep up with learning new technologies Likes solving problems

Ok… quality code is very hard to measure (which is why it couldn't be included in the quiz, but this is an important aspect of what makes a developer "good").

How do you know whether the code you wrote is good? Well, this comic illustrates things perfectly:

P.S. If you're a beginner developer or freelancer, you can get code reviews from our expert developers at Codementor.

Good devs are humble, responsible good guys who will get things done and make sure everything works as it should, but they lack the curiosity and passion for coding that will make them "great". That's ok, however. Most employers only need good devs and they don't really have a need for "great" developers. You don't have to feel pressured to be a "great" developer, as you can't force yourself

to be who you aren't.

The Really Good Developers

You don't have to feel pressured to be a "great" developer, because those who are really good don't need any hints—they're probably already doing these things because their passion and love of coding compels them to.

There are two types of developers that would really help a team:

- the MVP
- the Helpful Dev

The MVP

MVP-type developers don't want to simply just solve problems, but as they know there are many ways to do things, they strive to find the best method to solve a problem. They thrive on challenge and thus always work best on difficult tasks—this is what makes MVPs

far more productive than most developers as they are able to achieve what normal developers can't. Thanks to this love of challenge, however, employers may have difficulty in keeping them if the work assigned to them is too easy or mundane, because MVPs might leave if they get bored.

MVP devs usually take great pride in their work and thus they are sticklers for quality and performance. In fact, they would consider many of the edge cases and account for them before it happened. In some cases, they're their own QA engineer and aim to break their code before users do it. They're not the type who do TDD blindly because it's a "best practice", but they can design a program to greatly reduce the debugging time. As such, an MVP programmer can be at least 10x more productive than a bad one.

MVP devs have a strong sense of curiosity and would stop at nothing to find out "why" something works or doesn't work. They thus spend much time reading about programming as well just to keep up with all the technologies or learn about new things, but they don't jump on any bandwagons because they are more interested in figuring out things themselves. They are so passionate about coding, they usually program during their spare time as well, either on side projects or simply trying out new technologies, tools, and languages.

Finally, MVP devs are confident but humble as they always keep in mind that there will always be someone better than them, and rather than feeling threatened they would love to work with someone better simply because they want to learn from the better

developer.

The Developer Your Team Deserves

These developers exhibit similar traits as an MVP developer with their love of coding and their curiosity in constantly learning outside of work, but they're not necessarily as productive (though they're usually in the top tier in terms of productivity as well.) They usually leave just the right comment others need, and take the initiative in documenting things that need to be written down. In general what they do helps everyone else on the team because it's the documentation that will help everyone on the team be more productive. You can have a team of MVPs but this developer is the one who will make them all work at peak efficiency.

What's more, these kind hearted souls are the knowledgeable good guys who take great pleasure in helping inexperienced programmers become better programmers. They have deep knowledge of the tools they use and are patient in explaining to beginners why things work, and they strive to make sure that others

will be able to understand and learn to improve.

Most, if not all, experts on Codementor are these types. They aren't interested in just solving that pesky bug for you, but they want to help you learn how to be a better programmer. If you want to understand "why" things work and are having a hard time doing so through googling or fishing for answers at StackOverflow, then these are the right people to ask.

If you like helping other developers become better at solving problems, feel free to sign up to become a Codementor now!

Conclusion

In the end, the most important factor in becoming a good, or even great, developer lies within yourself. Perhaps it takes talent and a true innate passion to become a phenomenal top 1% programmer, but anyone with an interest in programming and solving problems can be a "good" programmer. If you don't want to become a good programmer, then no one, not even a great mentor, can help you. You are your greatest enemy, and you should always aim to be a better programmer than you are now.

Want to be a better programmer? Get a 7-day free trial with our Codementor Monthly Plan, where you can work with a dedicated mentor as you build your own project and learn why things are done in a certain way!