I stumbled over *The Ten Commandments of Egoless Programming* which supposedly hails from the book *The Psychology of Computer Programming*, and found them fascinating. I haven’t read the book but loved the idea so much that I thought I’d attempt to write down some of my thoughts. As an experiment I’ve taken the “commandments” and written my interpretations. These may or may not align with the original author’s interpretation.

**The Ten Commandments of Egoless Programming**

1. Understand and accept that you will make mistakes.
2. You are not your code.
3. No matter how much “karate” you know, someone else will always know more.
4. Don’t re-write code without consultation.
5. Treat people who know less than you with respect, deference, and patience.
6. The only constant in the world is change.
7. The only true authority stems from knowledge, not position.
8. Fight for what you believe, but gracefully accept defeat.
9. Don’t be the “coder in his corner”.
10. Critique code instead of people – be kind to the coder, not the code.

1. **Understand and accept that you will make mistakes**

Mistake – noun. An error or fault resulting from defective judgment, deficient knowledge, or carelessness.

*Synonyms:* error, fault, inaccuracy, omission, slip, blunder, miscalculation, fail

I found myself naturally drawn to the word failure when reading mistake. The words are usually used in similar contexts, but failure has been gaining traction in recent times. It’s branded across the Internet for all entrepreneurs, startups and everyone else to see: “Embrace failure”, “Those who don’t fail, can’t truly succeed”, “Failure is the last step before success”… and the list goes on.

**The value of mistakes**

Some environments are toxic when it comes to mistakes. Mistakes are frowned upon and fear leads to negative behavior. Sometimes the origin of such environments is cultural, company or personal experience.

I believe that one must first be or strive to be in an environment where making mistakes isn’t looked upon in a negative way that it will reflect poorly on you and limit your personal or professional path forward.

From our earliest years, we learn the most by pushing boundaries and making mistakes. Our first steps end with a fall, the first time we climb to the top of a tree may often end flat on our backs, learning new sports may lead to some awkward experiences… and the list goes on throughout our lives. We rarely think of our mistakes as failures when growing up, but rather practice or learning.

When we start to fear mistakes due to whatever personal experiences we have had, then we seek to avoid them. They get shrouded with negativity and life or careers or businesses becomes about avoiding mistakes. Looking past the mistake and reaching for the learning experience is where the value lies. A mistake is only a failure if you don’t learn from it.

2. **You are not your code**

Code is the result of human interaction. It is a production of our creative minds, a representation of our thoughts and reflections on a given problem and how we decided to best solve that problem. It’s no wonder that developers naturally feel defensive when others find flaws in their code.
But this is also where many developers fail to differentiate the creative thought-process of the problem at hand and the suggested solution in code. On the one hand there are a lot of valuable thoughts & ideas. On the other side is the technical implementation that may or may not adequately solve the problem. When the situation arises where the code isn’t acceptable or has bugs, then embracing the first commandment to learn from the mistakes you have made will lead to better quality code in the future.

3. No matter how much “karate” you know, someone else will always know more.
Not sure about the martial arts reference here, but there is no lack of technical fields of expertise to indulge yourself in. As you master more aspects of the software development, there may come a time where you feel empowered or feel you know all the answers. Your knowledge is a valuable asset, but you should know that there are limitations. Every developer has followed a different path, giving them unique insights. They may have experiences that you haven’t and thus see potential in approaches you don’t. Or they just know more “karate” and you need to realize that your yellow belt-worth of experience means you should heed their words instead of blindly ignoring them.

By doing so, you can bring your unique perspective and “sand on their shoulders” to achieve more than any of you could alone.

4. Don’t re-write code without consultation.
The boy-scout rule is a good rule to go by when delving in any code base. The core principle is to leave the code in better condition than before you were there. At times though this may lead to the “squeaky closet” syndrome, where one improvement leads to another, and another and before you know it you’ve refactored an entire module of an application.

When looking at this in an isolated manner, this may not be such bad thing. When working in a team on the other hand this may cause other developers to “lose their way” when the code changes to your personal style often. An approach here to spread knowledge and also to avoid forcing an issue where another developer feels bad about their code being ripped out is to ask the original developer or possibly even the team to do a review of your intended changes.

This invites to an open dialog and forces you to think upon the changes you are proposing.

5. Treat people who know less than you with respect, deference, and patience.
We have all been there; either being the new developer on the team with no idea about the teams coding style / preferences or the problem domain. Being open and inclusive is the best way to allow this developer to get in the groove of things and become productive.

Sadly though some developers feel the need to assert their authority. This leads to insecurity, negativity conflict and lack of productivity.

One of the best ways to learn or re-learn a topic is by teaching. Embrace this as a mutual relationship where you can further strengthen your knowledge while providing value to another and creating a safe environment for them.

6. The only constant in the world is change.
Change is a given in our field of expertise and it’s natural to feel overwhelmed at times. But recognizing this is as important as recognizing that change is constant. If we are always seeking to ride with the bleeding edge of change, then mastery may be hard to attain. On the other hand, being at the opposite side will soon lead from being a master of your field to being outdated and no longer attractive on the job-market.

Find your cadence, your sweet spot for how much you need to learn to keep up-to-date for your daily work and still be relevant as a software developer. But do this deliberately.
7. The only true authority stems from knowledge, not position.
Instead of seeking ways to gain authority through office politics, bad-mouthing or other actions that directly stem from your ego, attempt to lead by example. Focus on your circle of influence, which may purely be the code you create, and you will naturally progress to being the go-to developers when seeking aid. By improving yourself and helping others along the way your informal authority rises and your peers will value your input.

Commenting on any topic based on the latest trends without displaying true knowledge on the subject will diminish your authority and decrease your circle of influence. As well as make you unpopular among your peers.

8. Fight for what you believe, but gracefully accept defeat.
There will be times when your opinion, ideas or approach will be overruled. This may be for any number of reasons; not good enough, not enough support from your peers, seniority overruling or maybe even management plotting a new course. Do whatever you can to stand up for your opinion and position when there is room for it. Be clear and communicate with the knowledge your peers respect you for.

When a decision is made, do your part in getting behind it and following it through. This will yet again enforce you as being a developer that isn’t driven by personal ego-trips and a valuable asset that is willing to voice their opinions yet also prove to be a valuable team player.

9. Don’t be the “coder in the corner”.
I touched on this in a previous post on Pitfalls of writing software alone. Contradictory to the traditional view of a programmer, software development is very social in nature. Spending time to build relationships with your peers will increase your and their happiness. At the same time you will avoid isolating yourself from potential career possibilities.

Walling yourself in will also deprive you of the potential learning possibilities from co-workers and community members. On the one side you may be creating great things, but what fun is it if there’s no-one to share it with?

10. Critique code instead of people – be kind to the coder, not the code.
I touched on this with commandment 3 (You are not your code). By caring for the next person and truly showing empathy, you can slice and dice your codebase without fear of insulting each other. Code is the result of a shared collaborative process that involves many people. Sometimes your interpretation ends up being wrong, and it’s completely fine to alter it.
I’ve previously written a blog post on code-reviews and how focus on shared values and working towards the same end goals greatly improves the dynamics of a team.

Rounding up
I quite enjoyed these “commandments” and breaking them down. They shed light on several aspects that often turn out sour when working as a software developer. Mixing of team-members without a shared set of values may often lead to conflicts. These conflicts overshadow the important end result; creating valuable software that solves the users needs.

There will always be ego involved in anything we do, but our will to embrace empathy and have shared experiences that enrich our lives should drown out any of the crap that ego attempts to throw at us.

https://codingwithempathy.com/2016/03/01/reflections-on-the-ten-commandments-of-egoless-programming/