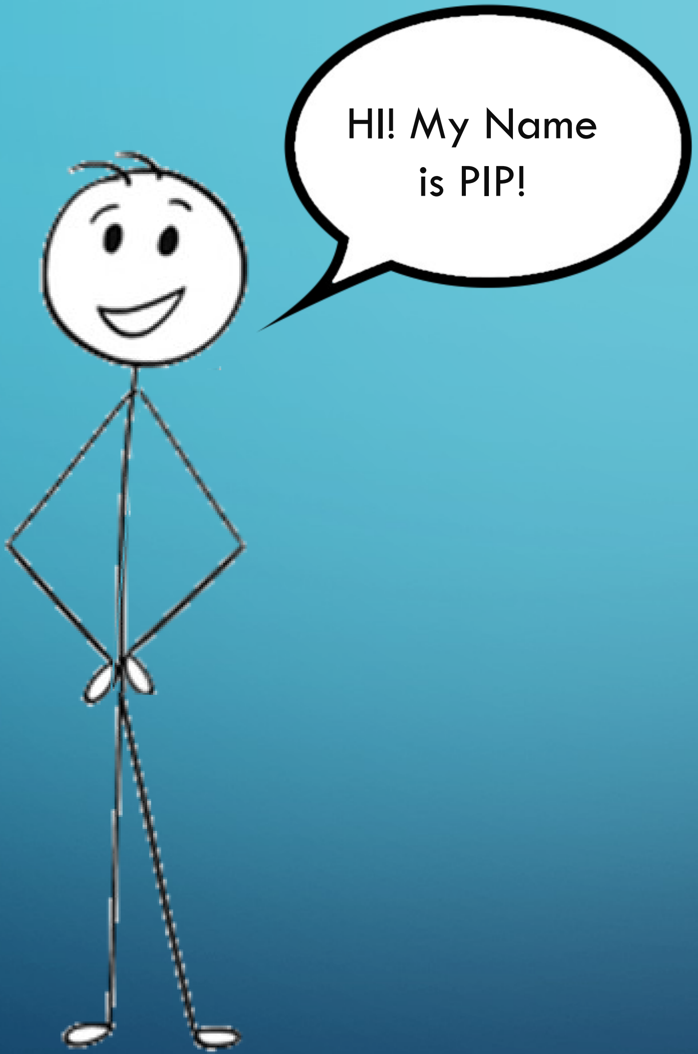
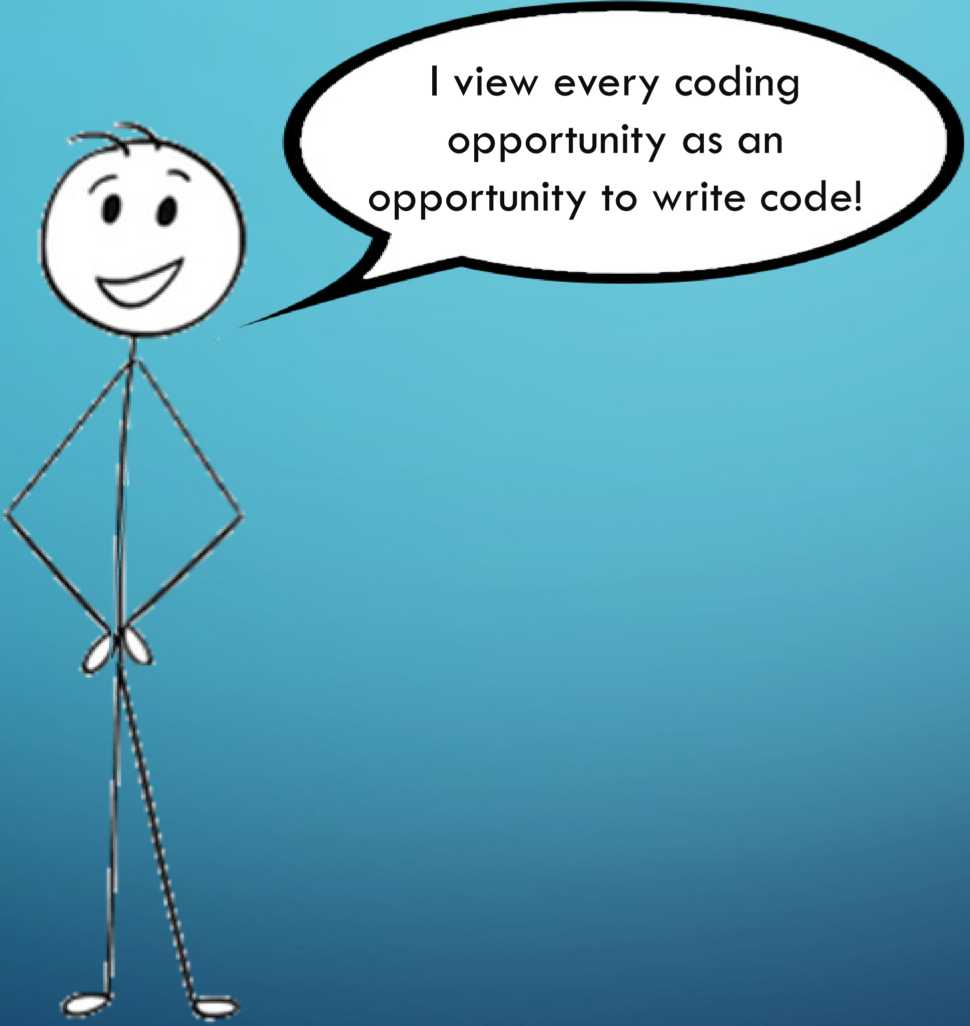
PIP'S TALE

A PROGRAMMER'S JOURNEY

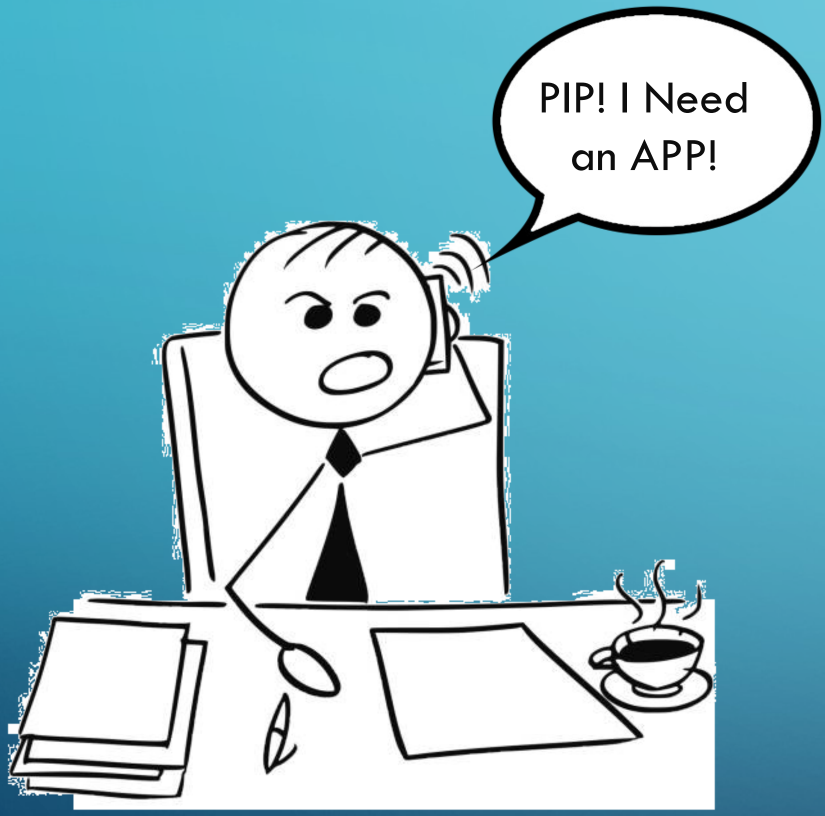
Once upon a time, there was a programmer named Pip

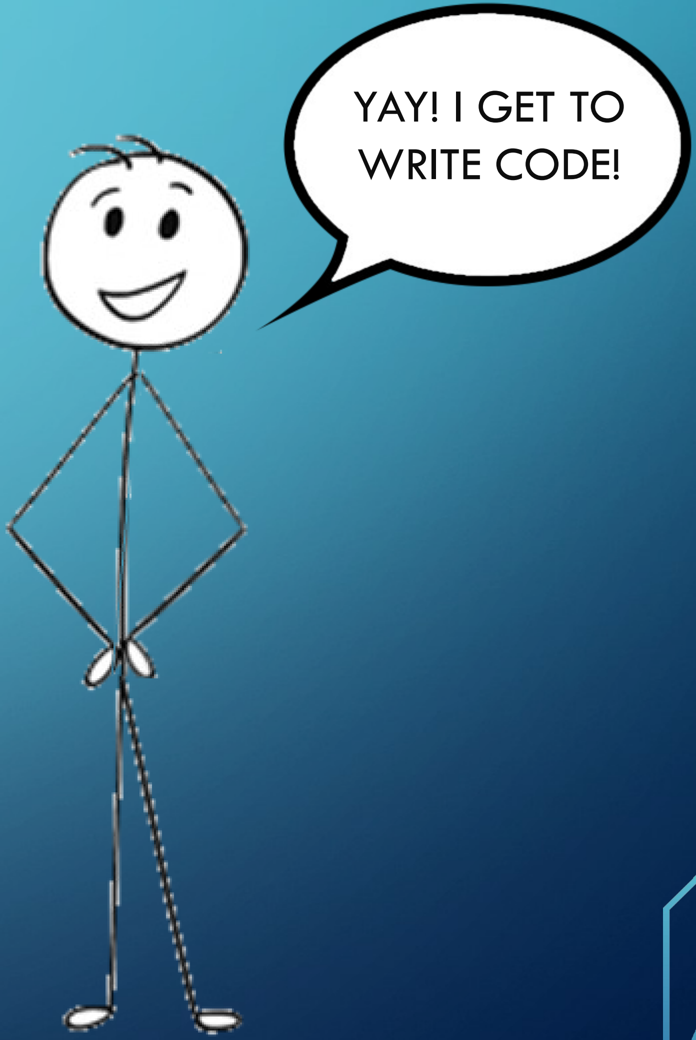


Pip was a very enthusiastic and hardworking programmer…


I view every coding opportunity as an opportunity to write code!

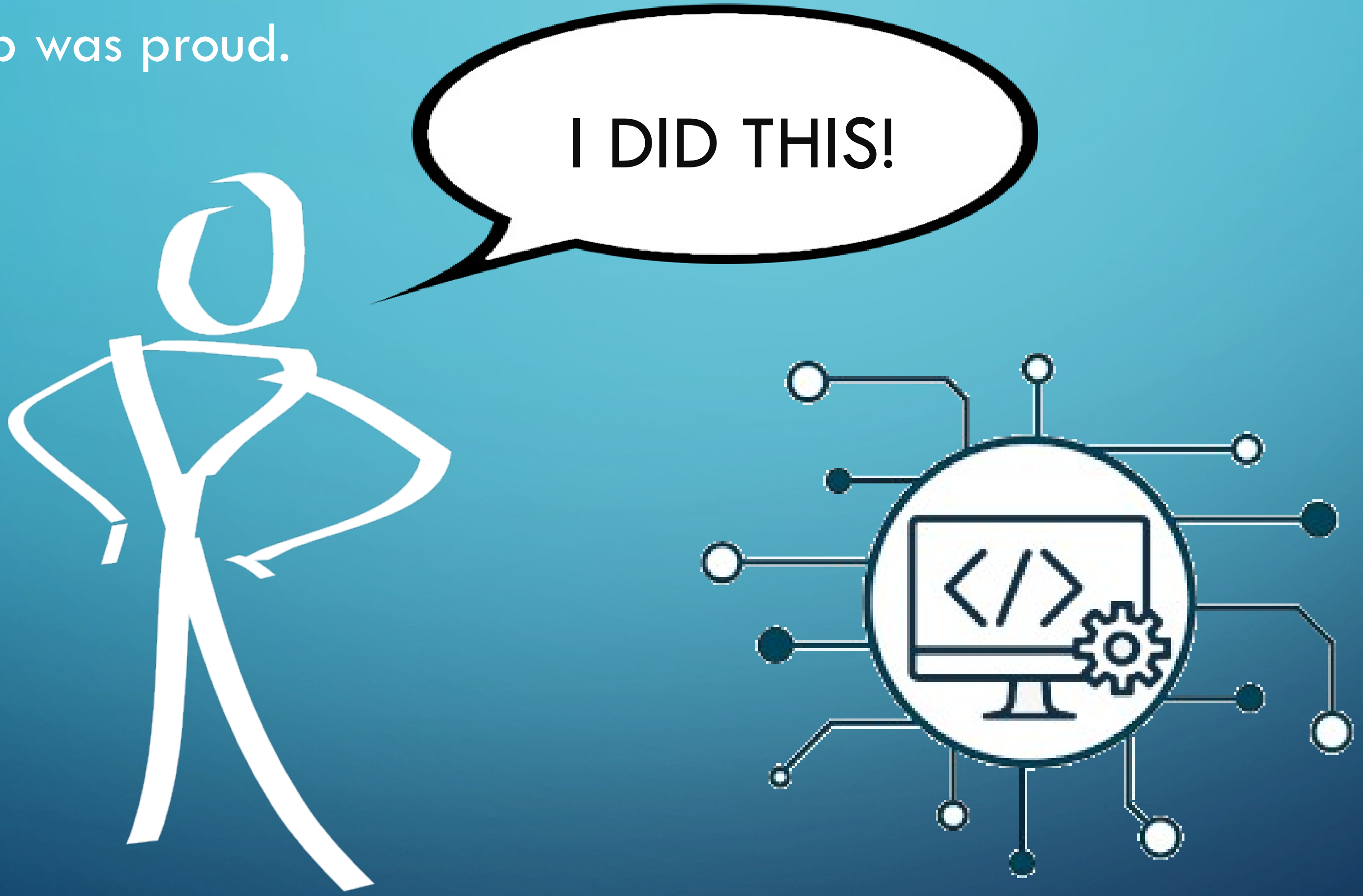
Pip got a job where his boss asked him to build a software app to solve a problem!



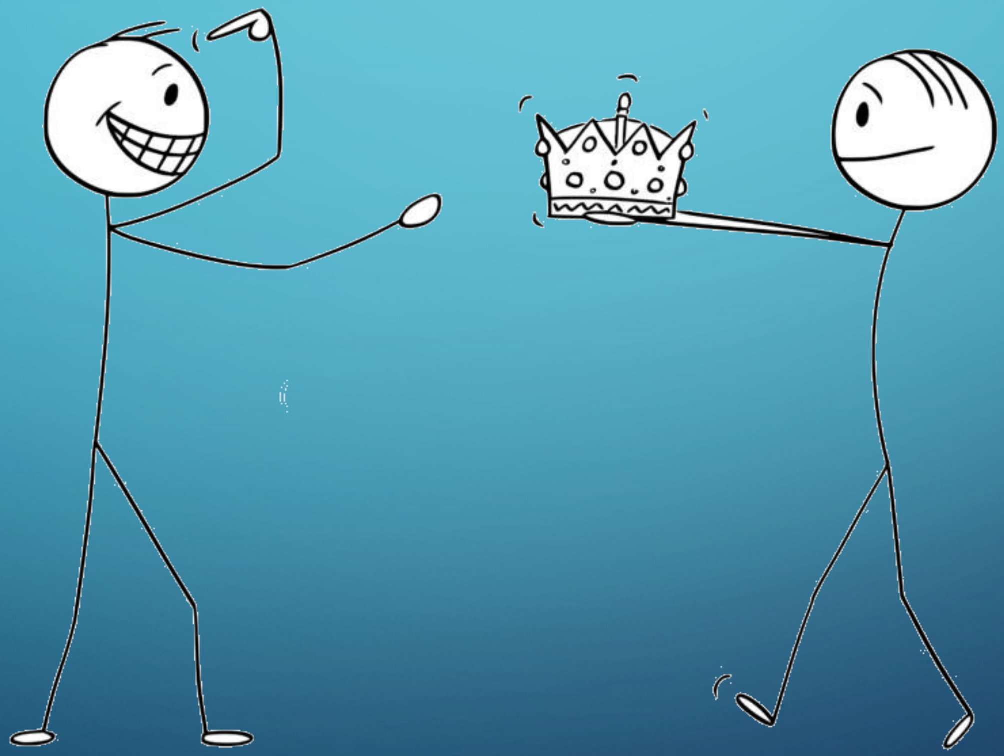


So Pip got hard to work. He chose PHP to build the app, because it was the language he was most familiar with, and he let loose with his enthusiastic gusto to write code

After long weeks and much sweat, tears, and overtime, Pip had his Magnum-Opus, and Pip was proud.



Pip basked in the gratitude of his boss and his customers, and that gratitude and adulation made pip realize one thing…



He had arrived…

But all was not well in the Kingdom, and the passage of time was not kind to Pip

A certain "smell" has arisen from the app…

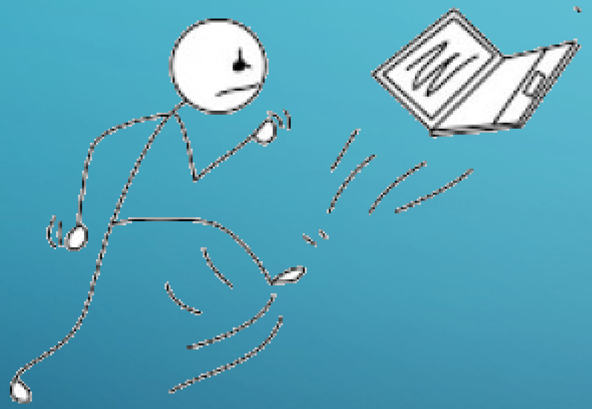
As changes are introduced in one part of the system, they break other seemingly not related parts

The code's brittleness causes many errors in production, which leads to much overtime, and those 3:00 am crisis calls

It takes longer and longer to push out releases, and the customers are getting frustrated

Pip's quality of life is decreasing, along with the quality of the code base

Pip starts wondering what went wrong, and he starts looking for answers…



His magnum-opus is now reviled by its users

His boss is looking for a replacement, and may outsource it

But Pip worked so hard on that app

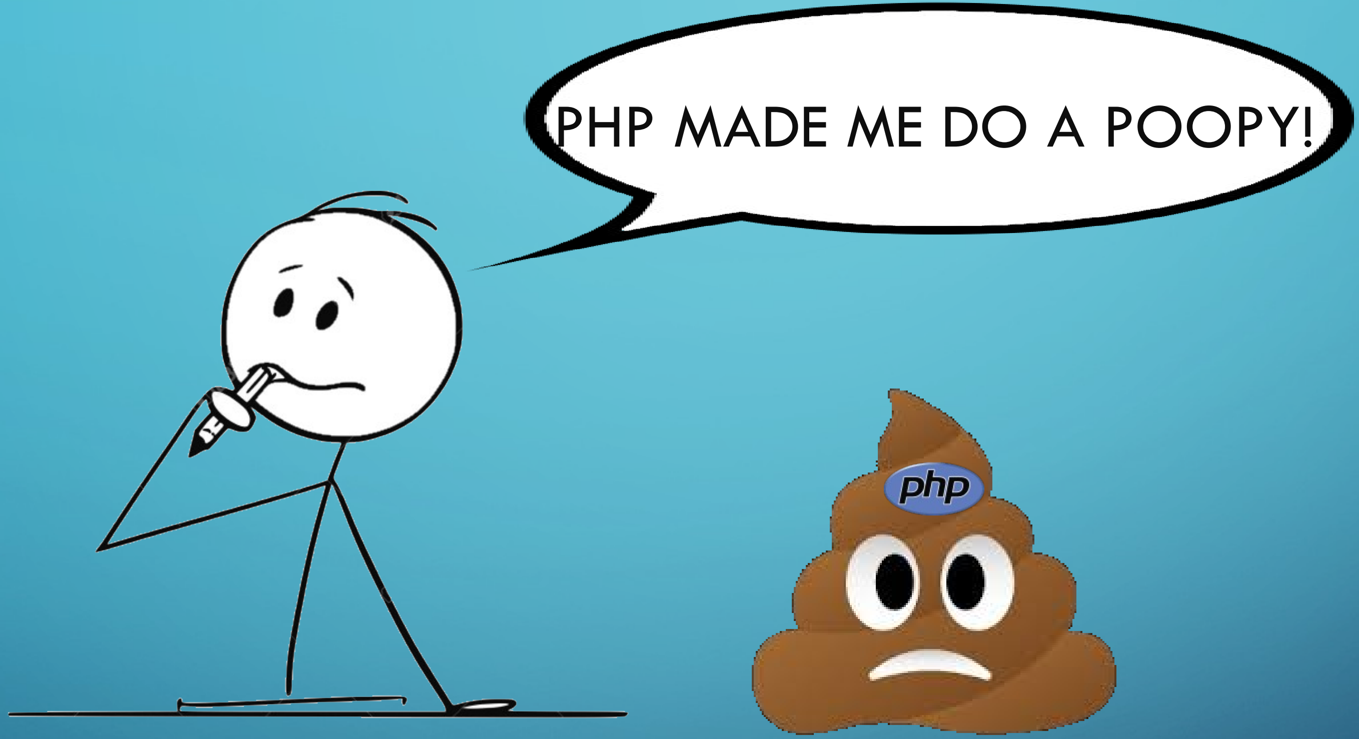
He implemented the requirements the way they were written…

He listened to his end-users…

And he wrote so… much… code…

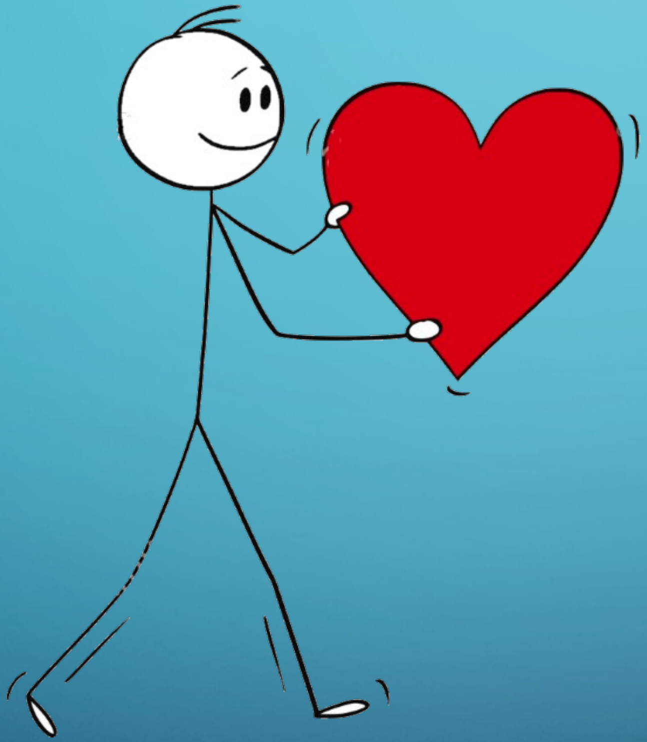
Which then lead him to the following, and the only possible logical conclusion…

It was the language's fault…



And since it was the language's fault, there was only one possible solution to the problem…

It was time to switch languages…



Because that’s the only thing that could have been the cause of his failures (IKR).

10 YEARS GO BY…

Older, but maybe not yet wiser, Pip has some painful reminiscing to do…

Pip has been involved with many projects that seem successful at first, but became cumbersome and unwieldy after time, and this pattern was repeated regardless of language and platform

Many projects… same outcomes. First comes success at the initial release of the code, but then failure in the maintenance and enhancement cycle

Pip has switched languages several times, and started using frameworks, but at best, it only delays the inevitable negative outcome

Pip can only come to one conclusion… the problem isn't with the language… the problem is Pip!!!

Humbled by the realization that HE is the problem, Pip resolves to improve…

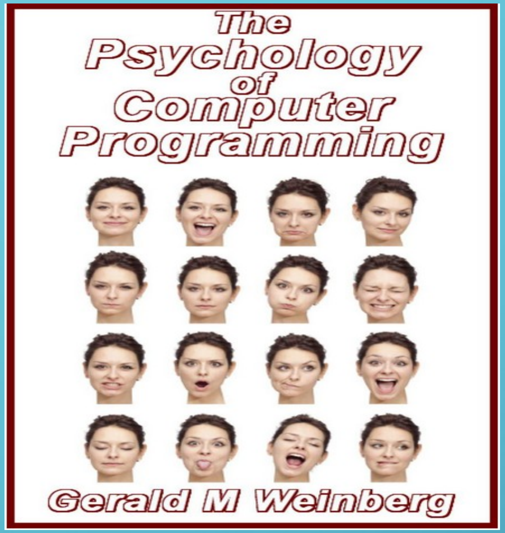
Pip Embraces “The Three Virtues” of a Great Programmer

1. **Laziness**: The quality that makes you go to great effort to reduce overall energy expenditure. It makes you write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.

2. **Impatience**: The anger you feel when the computer is being lazy. This makes you write programs that don't just react to your needs, but actually anticipate them. Or at least pretend to.

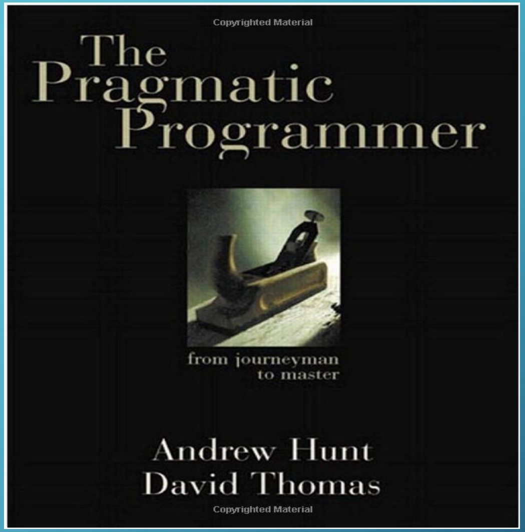
3. **Hubris**: The quality that makes you write (and maintain) programs that other people won't want to say bad things about.

And PIP reads some books…



The Psychology of Computer Programming

Gerald M Weinberg



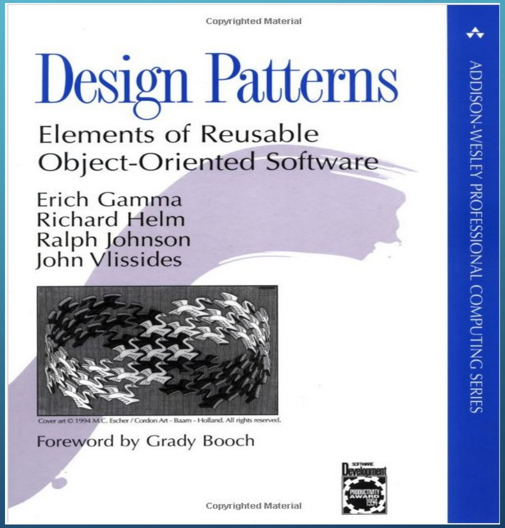
Copyrighted Material

The Pragmatic Programmer

from journeyman to master

Andrew Hunt
David Thomas

Copyrighted Material



Copyrighted Material

Design Patterns
Elements of Reusable Object-Oriented Software

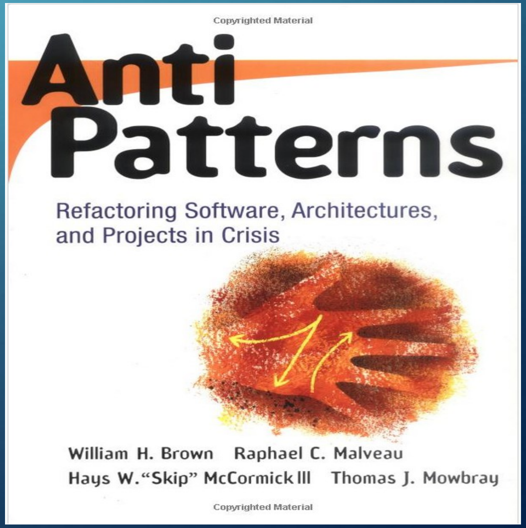
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

Copyrighted Material

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



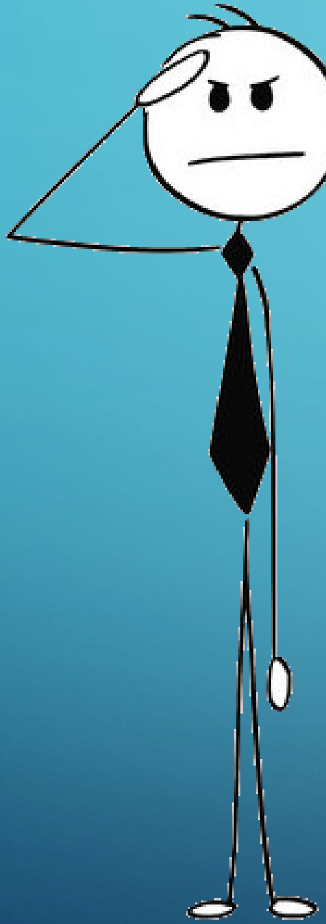
Copyrighted Material

AntiPatterns
Refactoring Software, Architectures, and Projects in Crisis

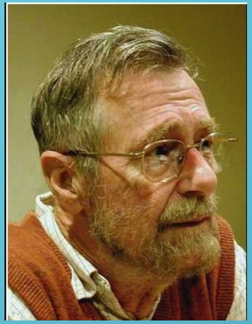
William H. Brown Raphael C. Malveau
Hays W. "Skip" McCormick III Thomas J. Mowbray

Copyrighted Material

And Pip looks to recognized experts for their guidance and insights…



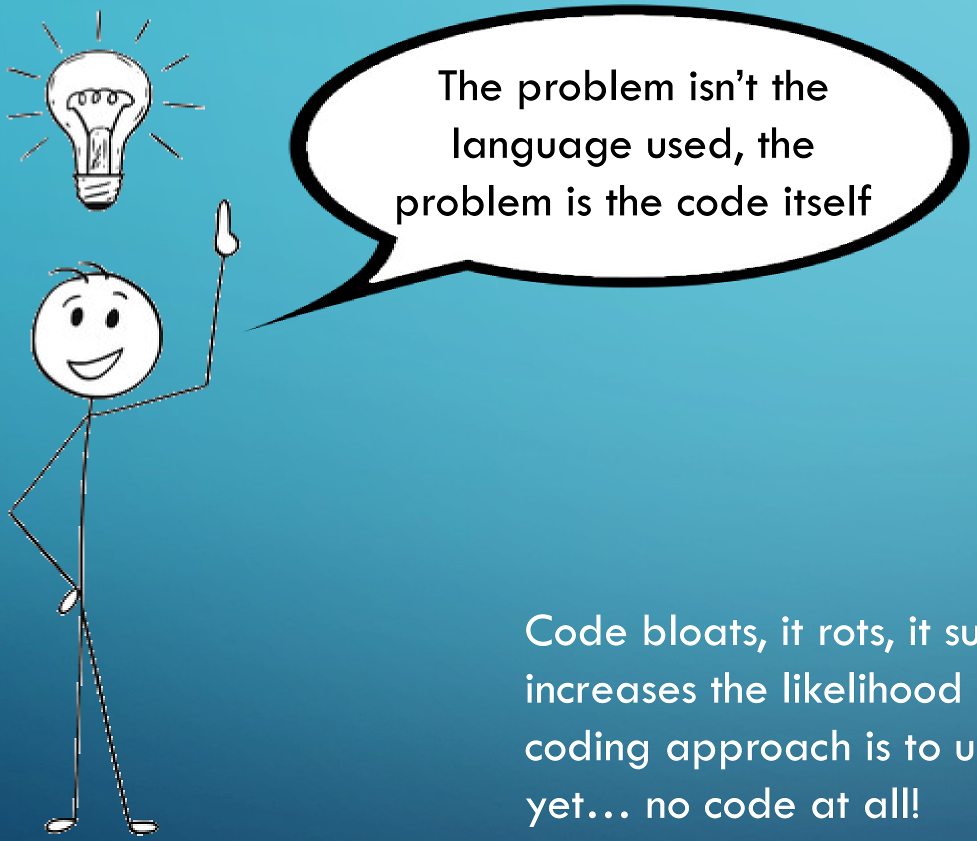
Grace Hopper


Edsgar Djikstra

Martin Fowler

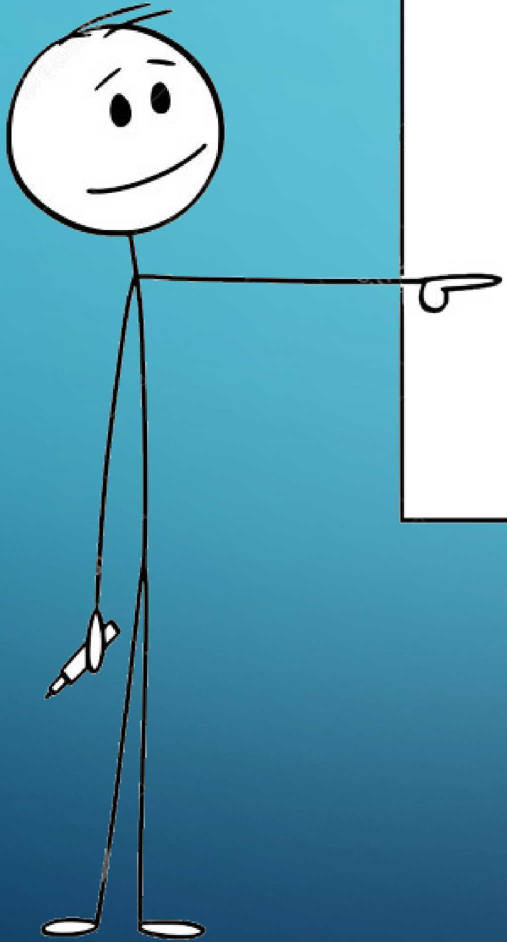
Doug Crockford

With all this new knowledge, Pip finally realized what the problem REALLY was…


The problem isn't the language used, the problem is the code itself

Code bloats, it rots, it suffers from Software Entropy, and every line of code increases the likelihood of bugs. Not all code is created equally, but the best coding approach is to use the least amount of code to solve a problem, or better yet… no code at all!

Pip learned to not try to solve every problem with code, but when code is needed, remember these tips and techniques…

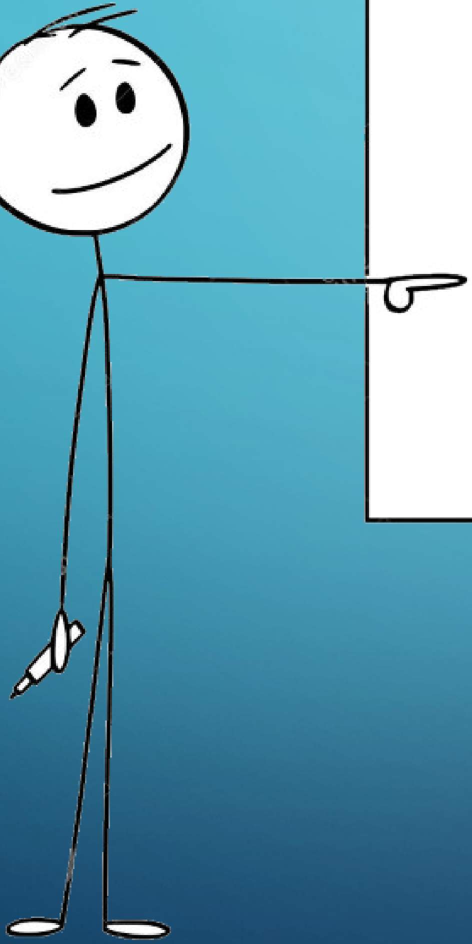


Design patterns are your friend

Know your Anti-Patterns as well and if you find yourself possibly implementing one, stop and find a different solution

Avoid re-inventing the wheel, but if you have to, make sure it is a round one!

If you code, use these techniques…

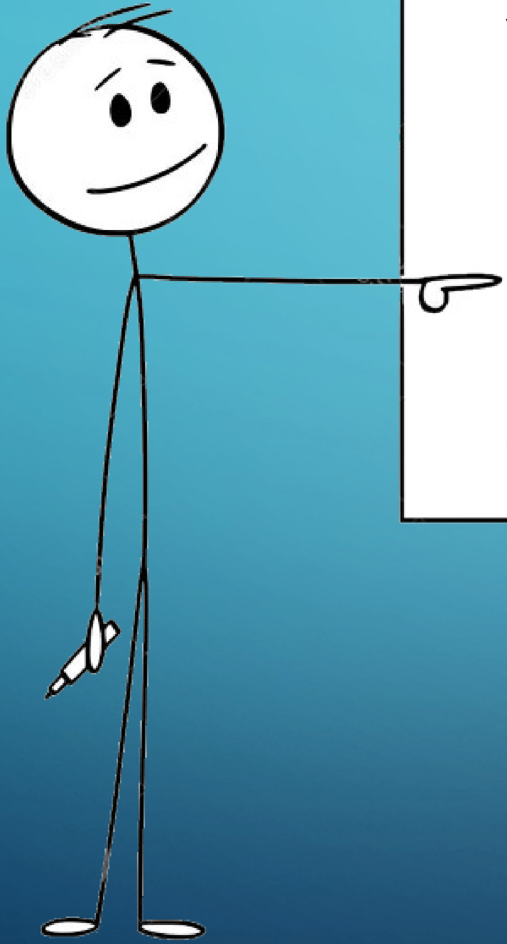


Well defined Conventions Over Configuration

Separation Of Concerns can help you with managing complex code bases

Favor Composition Over Inheritance when you need to build classes

Know your enemies!

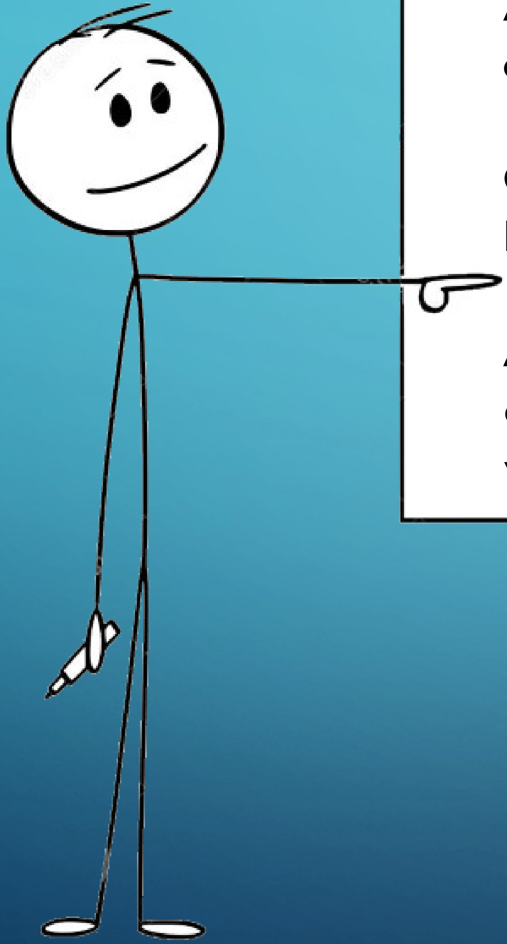


Watch out for Software Entropy, and remember that refactoring is your friend!

Know what Technical Debt is, and how to mitigate and avoid it

Be on the lookout for Code Smells, and deal with them early and often!

And most importantly… Abstract as much as you can!

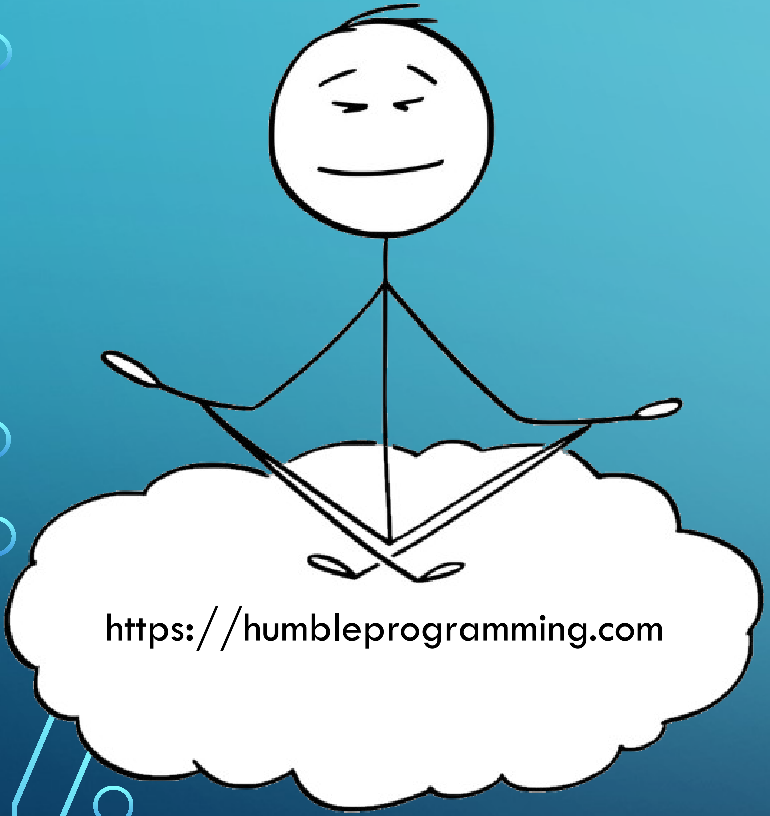


Abstractions reduce complexities by creating a simplified version of something

Code generators can reduce maintenance by automating common tasks

Applied together, they greatly reduce the amount of code you have to support or where bugs might be hiding

It turns out being a Great Programmer is a Journey, and not a destination.



Pip has learned his lesson, and has traded his enthusiasm for coding with a healthy dose of skepticism

Code is written only when necessary, is well documented and understood, and the knowledge is shared with his peers

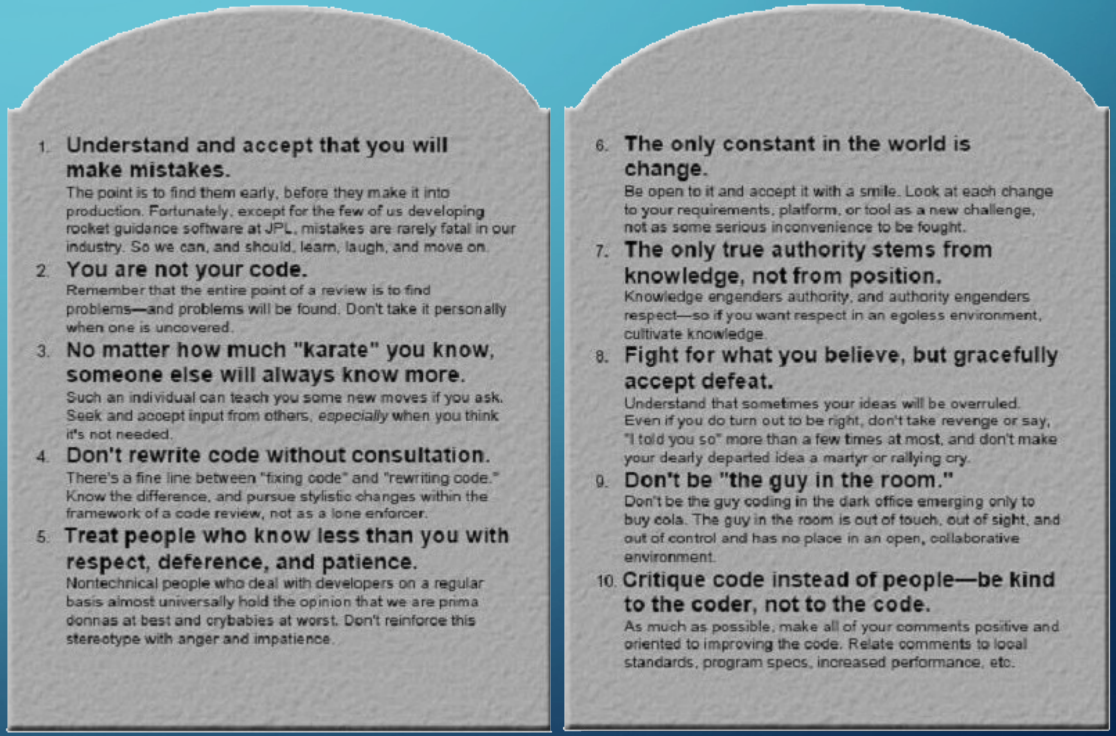
Rather than solving individual problems, systems are built to solve recurring problems, and the system is adapted for new problems

The simplest approach to complex problems are pursued, using known and established techniques. The days of Cowboy-Coding are over

Pip's biggest reward is sense of achievement, accomplishment, and a much higher quality of life

Pip wishes you all well in your Journey and to take this last bit of wisdom with you…

The 10 Commandments of Egoless Programming



1. **Understand and accept that you will make mistakes.**
   The point is to find them early, before they make it into production. Fortunately, except for the few of us developing rocket guidance software at JPL, mistakes are rarely fatal in our industry. So we can, and should, learn, laugh, and move on.
2. **You are not your code.**
   Remember that the entire point of a review is to find problems—and problems will be found. Don't take it personally when one is uncovered.
3. **No matter how much "karate" you know, someone else will always know more.**
   Such an individual can teach you some new moves if you ask. Seek and accept input from others, *especially* when you think it's not needed.
4. **Don't rewrite code without consultation.**
   There's a fine line between "fixing code" and "rewriting code." Know the difference, and pursue stylistic changes within the framework of a code review, not as a lone enforcer.
5. **Treat people who know less than you with respect, deference, and patience.**
   Nontechnical people who deal with developers on a regular basis almost universally hold the opinion that we are prima donnas at best and crybabies at worst. Don't reinforce this stereotype with anger and impatience.
6. **The only constant in the world is change.**
   Be open to it and accept it with a smile. Look at each change to your requirements, platform, or tool as a new challenge, not as some serious inconvenience to be fought.
7. **The only true authority stems from knowledge, not from position.**
   Knowledge engenders authority, and authority engenders respect—so if you want respect in an egoless environment, cultivate knowledge.
8. **Fight for what you believe, but gracefully accept defeat.**
   Understand that sometimes your ideas will be overruled. Even if you do turn out to be right, don't take revenge or say, "I told you so" more than a few times at most, and don't make your dearly departed idea a martyr or rallying cry.
9. **Don't be "the guy in the room."**
   Don't be the guy coding in the dark office emerging only to buy cola. The guy in the room is out of touch, out of sight, and out of control and has no place in an open, collaborative environment.
10. **Critique code instead of people—be kind to the coder, not to the code.**
    As much as possible, make all of your comments positive and oriented to improving the code. Relate comments to local standards, program specs, increased performance, etc.

Further learning via Google:

- Convention over Configuration
- Separation of Concerns
- Inversion of Control
- Design Patterns
- Anti-Patterns
- Abstraction
- Software Entropy
- Technical Debt
- Code Smells
- Refactoring
- Composition Over Inheritance
- Low-Code/No-Code
- The Best Code Is No Code At All

- Egoless Programming
- Post Heroic Leadership
- East Coast vs West Coast Management
- SEI/CMMi