10 Best Practices for Software Developers

Basil Udoudoh

CONTENTS OF THIS TEMPLATE

Here's what you'll find in this Slidesgo template:

1. A slide structure based on a newsletter, which you can easily adapt to your needs. For more info on how to edit the template, please visit **Slidesgo School** or read our **FAQs**.
2. A **thanks** slide, which you must keep so that proper credits for our design are given.
3. A **resources** slide, where you'll find links to all the elements used in the template.
4. **Instructions for use**.
5. Final slides with:
    - The **fonts and colors** used in the template.
    - More **infographic resources**, whose size and color can be edited.
    - Sets of **customizable icons** of the following themes: general, business, avatar, creative process, education, help & support, medical, nature, performing arts, SEO & marketing, and teamwork.

You can delete this slide when you're done editing the presentation.

TABLE OF CONTENTS

INTRODUCTION

This is me, Basil

I'm a Software Architect/Lead from Slalom. I've been building software on multiple platforms (web, mobile, desktop, cloud) for 15 years

I was born and raised right here in Atlanta

I'm still waiting for that superbowl parade…….

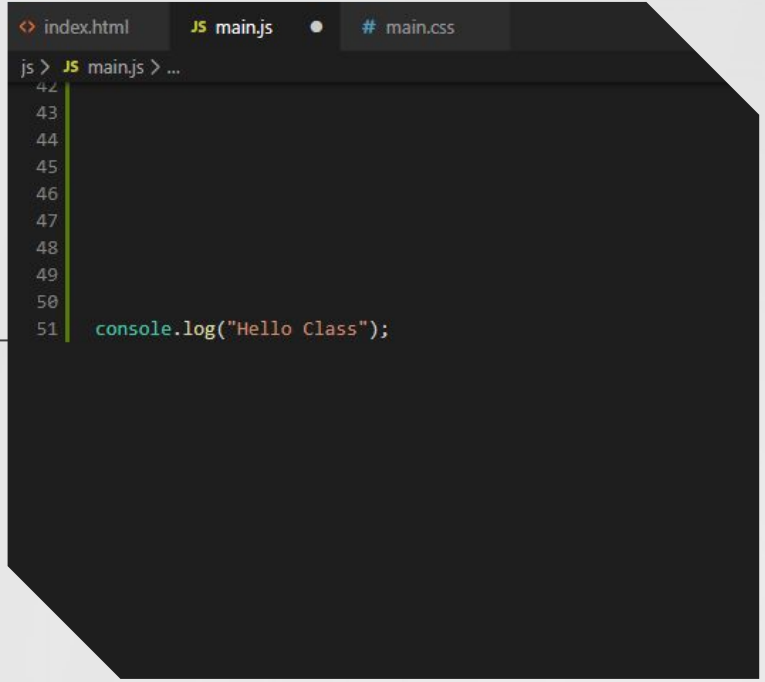
TELL ME ABOUT YOU

This is you, class

Professional Background

Where are you from originally?

What are you looking to get out of this class?



COURSE RULES

PARTICIPATE!

You will all get so much more from this experience if you participate. Try to contribute to all of the discussions.

ASK QUESTIONS

Though it might sound like it when I talk, I don't just want to listen to my voice all day. Ask questions and share experiences whenever you want.

NO CODING NEEDED

We all had to start somewhere and for some of us, it might be here. You don't need to have coding experience to get something out of this course

SAY YES TO JS

All Coding Examples are in JavaScript, though the principles apply to all programming languages

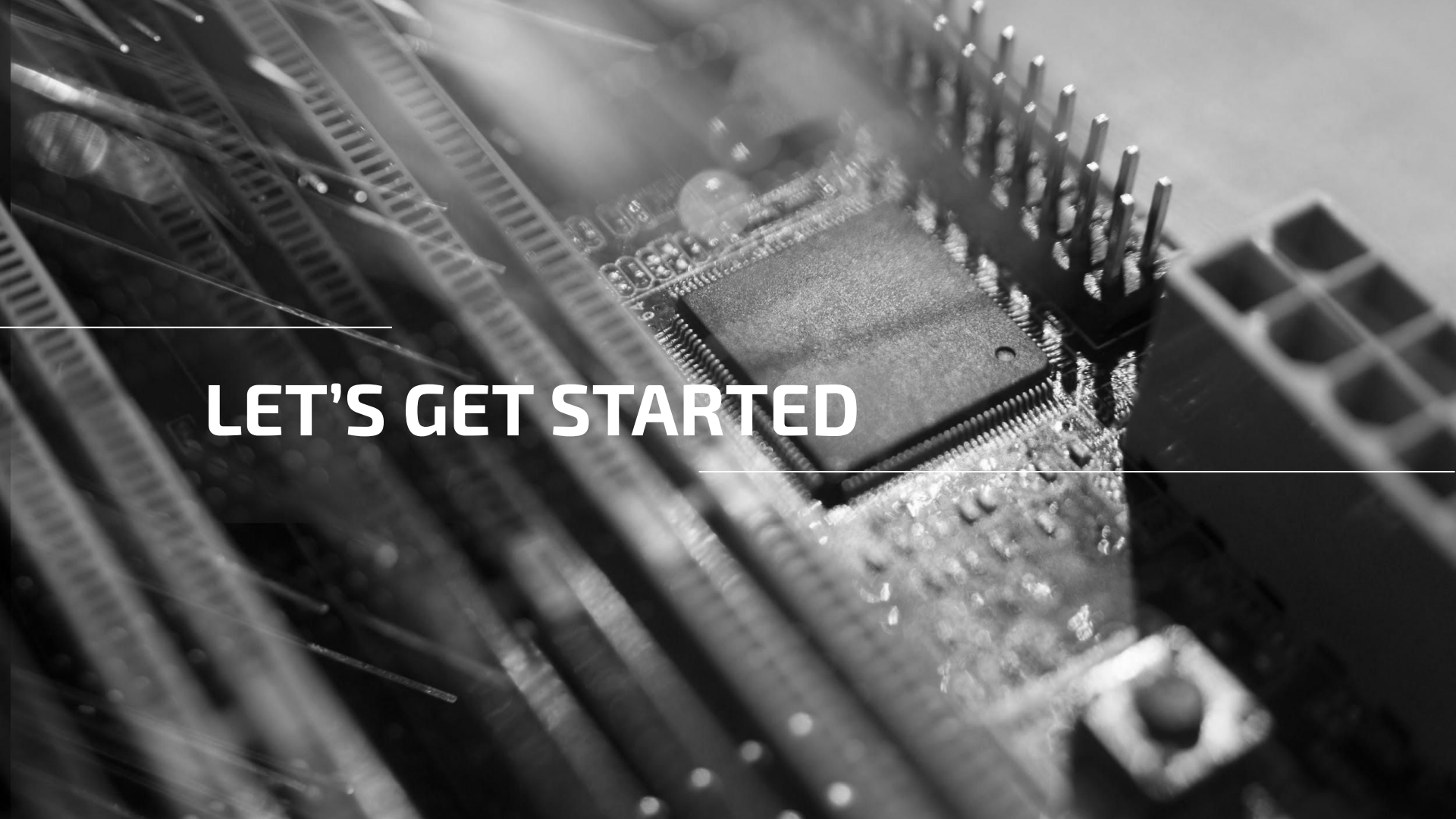
GETTING STARTED WITH REPL.IT

https://repl.it/join/hzffvjrv-basiludoudoh

All of the coding examples for later on in our course can be found here. We'll be working together on repl.it.

If you don't already have an account, go ahead and sign up. It's free and a great tool to use to work on code collaboratively.

If you would rather follow along and not use repl.it, that works too!


LET'S GET STARTED

BEST PRACTICE CATEGORIES

DEVELOPMENT SCOPING/PLANNING

A key part of building anything that you don't want to fail spectacularly, include software, is the planning process. These best practices will help you start your projects off on the right footing

Every trade has its rules. These are your rules for development. Follow them. All of them.

CODING PRACTICES

TESTING PRACTICES

How do you know that something works if you haven't tested it? You don't know, that's how. Follow these practices to ensure you produce good tests, so that you know your code works.


DEVELOPMENT SCOPING/PLANNING

DEVELOPMENT SCOPING/PLANNING - BEST PRACTICES

01. Think/Plan before writing code

02. Develop Iteratively

03. Prioritize working code

01

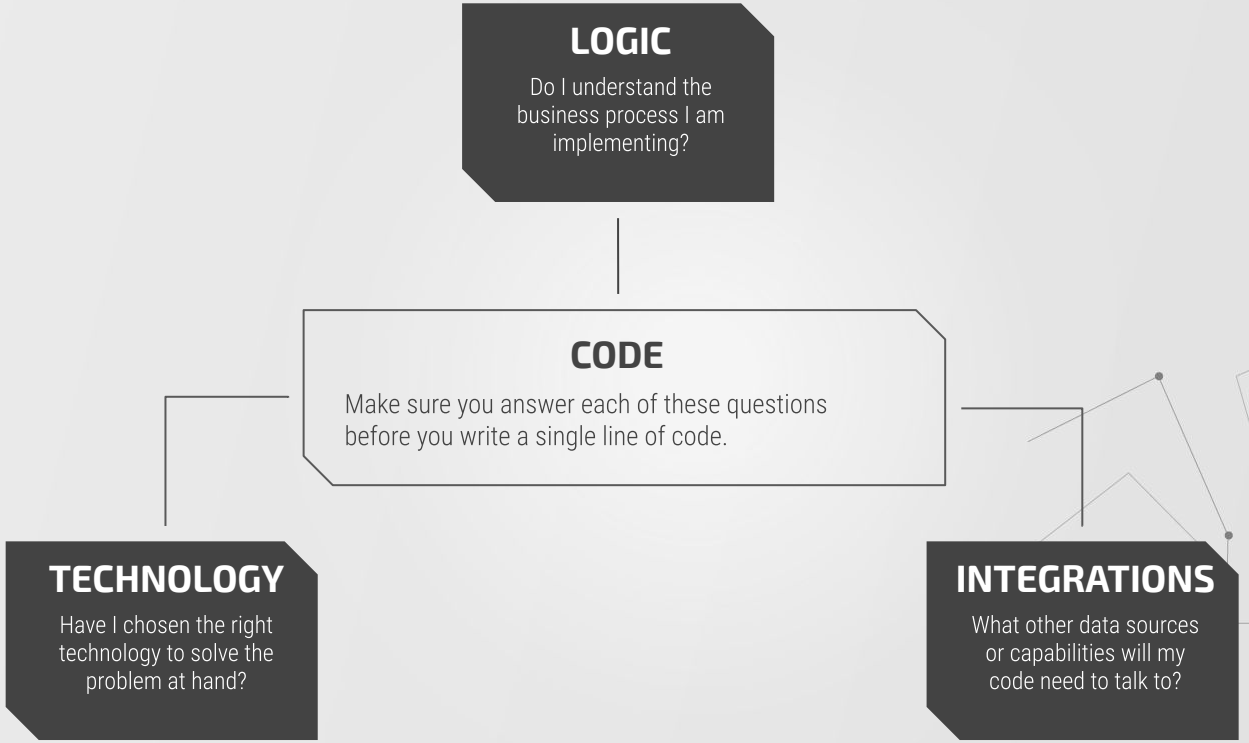
Think/Plan before writing code

Don’t underthink it!

DEVELOPMENT “COOKBOOK”

Writing code is like following a recipe

DEVELOPMENT PLAN- DON'T UNDERTHINK IT!



EXERCISE - LET'S ASK SOME QUESTIONS

TOC Needs an App, and you can help!

Marcellus has had enough of all of the social media and meetup apps that are out there, so he's decided to take matters into his own hands and build a separate mobile app for Technologist of Color. He's looking for our help to join the team and help with this effort, but we have to prove ourselves with a small feature development task. He wants us to build a function in the API that will perform the initial registration of a user. What questions should we be asking to make sure we have the right plan before we start working on this function?

02

Develop Iteratively

The real “MVP”

SPEED TO VALUE

Modern companies want to go from zero to release in as little time as possible

THE REAL “MVP” - MINIMUM VIABLE PRODUCT

Minimum Viable Product is defined as the version of a product that has the least amount of features and still allows the user to get value from it

MOVE FAST AND MAKE THINGS

With an agile methodology-based iterative development and release cycle, companies are able to get their MVPs in the hands of their customers continuously.

Developers have to familiarize themselves with concepts like “Continuous Delivery”, which describes code release capabilities and practices that allow teams to frequently deploy new features to their customer-facing application version (Production)

EXERCISE - LET'S DEFINE OUR MVP

Where should TOC's app start?

Marcellus has big hopes and dreams for the TOC app. He expects it to be a game-changing app that brings technologists from all over the world together to share their hopes, dreams, ambitions, and code. He wants people to be able to communicate with each other whenever they want, however they want, through this app. It will truly shift the paradigm of developers of color coming together in all forms. But for now, he'll settle for just letting people know about the meetups. What features are necessary for this application's MVP version?

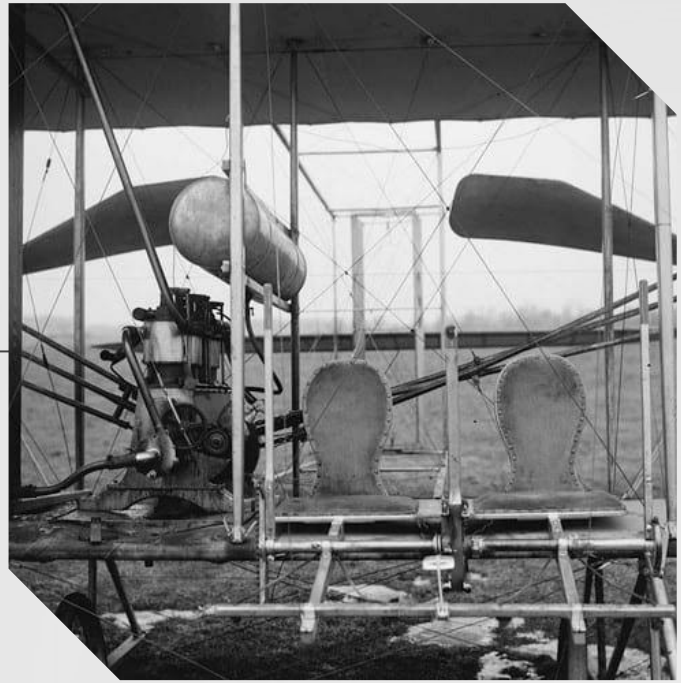
03

Prioritize working code

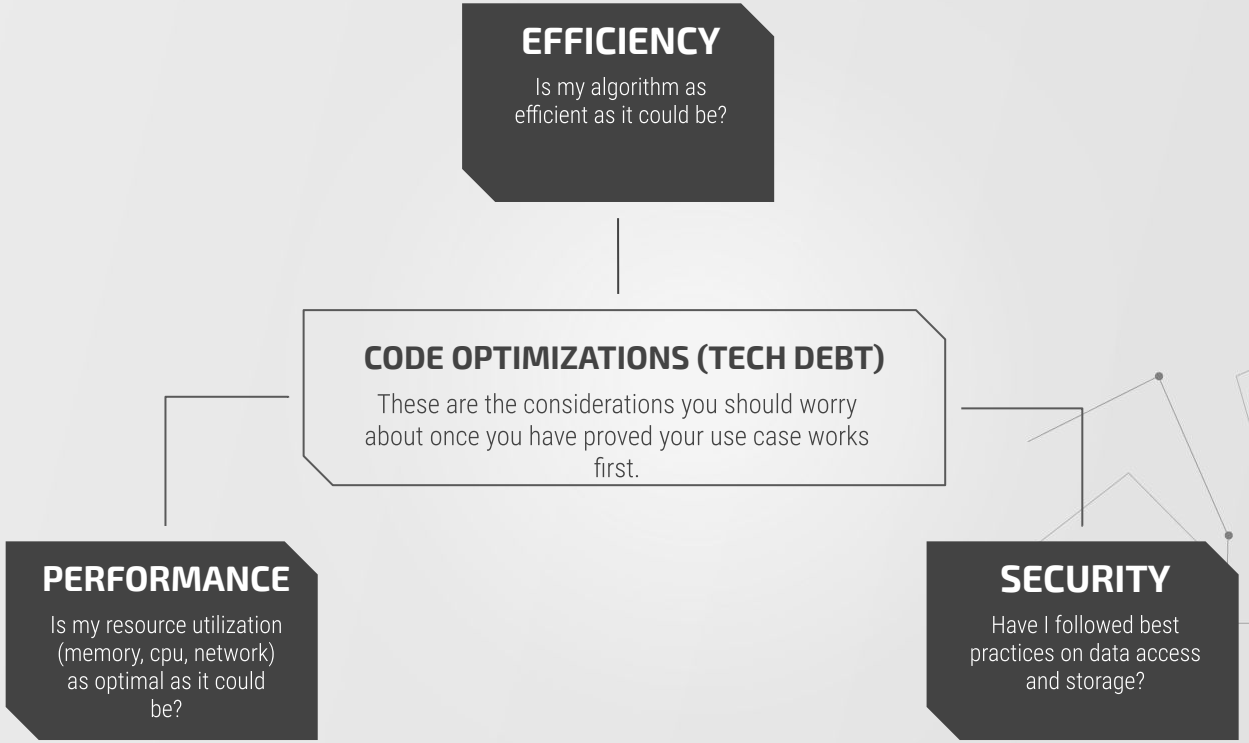
Make it work, then make it work well

Crawl, Walk, Run, Fly….

Does anyone recognize what's in this picture?



MAKE IT WORK, THEN MAKE IT WORK WELL



DEVELOPMENT SCOPING/PLANNING - SUMMARY

01. Think/Plan before writing code

Make sure you plan out all of the logic and components of your project/code before you start creating, just like in a recipe

02. Develop Iteratively

Modern companies want to be able to release features to their users as frequently as possible. Determining your MVP and developing iteratively helps you get there.

03. Prioritize working code

Our first objective should always be to solve the problem we're trying to solve. Once we're done, then we can make our solution more efficient and secure.

CODING PRACTICES

CODING - BEST PRACTICES

04. Write easy to read code

05. Use logical/consistent function and variable names

06. D.R.Y.

07. Zen and the art of commenting

08. Plan for failure

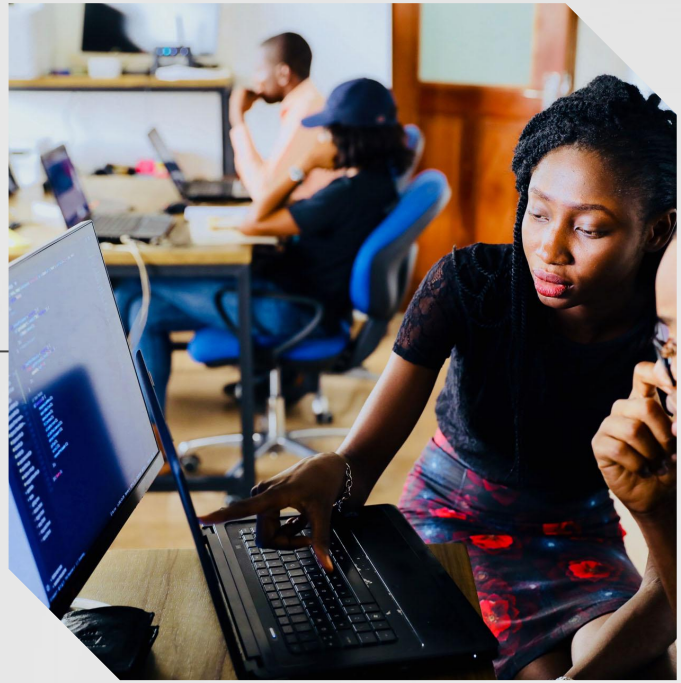
04

Write easy to read code

Keeping it simple, keeping it real

SOFTWARE DEVELOPMENT: THE ULTIMATE TEAM SPORT

There are very few software projects out there that have one contributor. You have to write your code as if someone else will read it.



KEEPING IT SIMPLE, KEEPING IT REAL

- Always remember K.I.S.S - Keep It Simple Smarty
- Since we're all smarties, let's add another Smarty term to that description: **Cognitive Complexity**
  - C.C. is a measure of how difficult a unit of code is to read and understand
  - The higher the cognitive complexity, the harder the code is to read.
- There are 4 things can we can all do as developers to help reduce our code's cognitive complexity
  - Consistent Indentation
  - Limit line length
  - Avoid deep nesting
  - Consistent variable names (we'll cover this one later……)

EXERCISE - COGNITIVE COMPLEXITY

Let's take a look at a code snippet from our API

Can you identify all of the issues with this snippet, based on the examples we just discussed around cognitive complexity?

05

Logical/Consistent function and variable names

A rose by any other name wouldn't make sense!

WHAT'S IN A NAME

Where do you all think this, lush scenic landscape is located?

FUNCTION/VARIABLE NAMES SHOULD BE FUNCTIONAL

WHAT I DO

Function names should describe the logical operation the function is performing. If your function calculates the age difference between two Students, call it “studentAgeDifference”

Your functions should describe the data they act on. If your functions job is to fetch an Student record from a database, call it “getStudent”

WHAT I ACT ON

WHAT I REPRESENT

Your variable names should describe what they represent. If your variable holds a student record from the database, call it “Student”

Above all else, don’t name your variables and functions after funny jokes or secret messages. That’s what Slack is for!

EXERCISE - A ROSE BY ANY OTHER NAME DOESN'T MAKE SENSE

Let's take a look at a code snippet from our API

Can you identify the issues with the naming conventions used in this code? What issues does it cause in understanding it's purpose

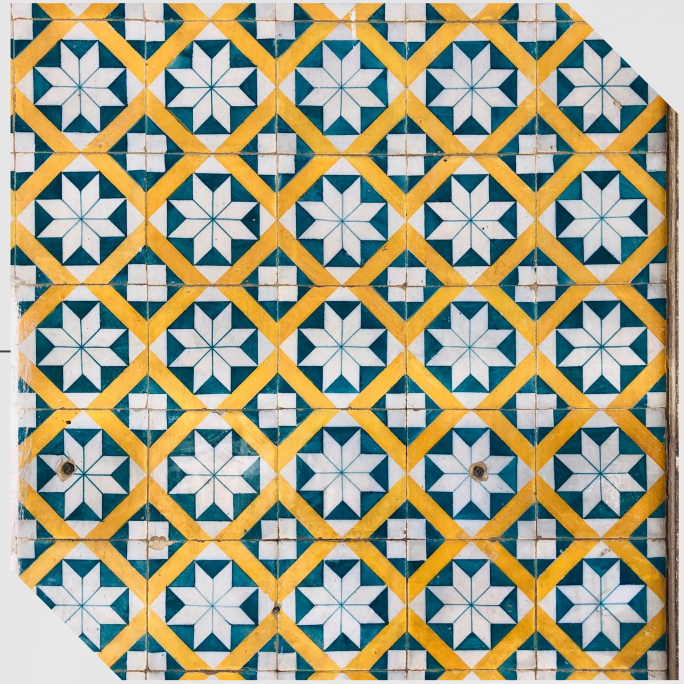
06

D.R.Y.

For brevity, that's W.H.Y.

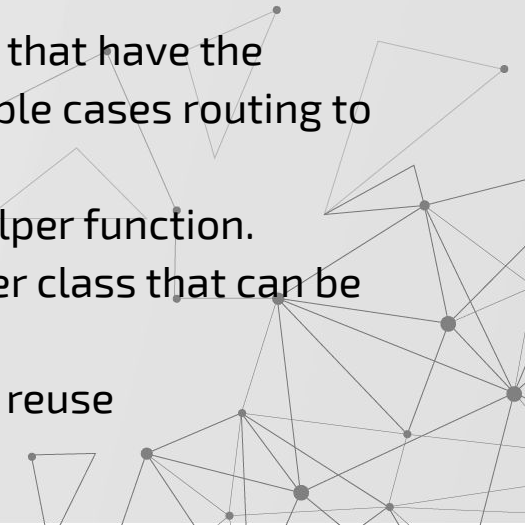
SO NICE, I DID IT TWICE

Patterns are a central hallmark of art. Not of code.



BREVITY IS THE W.H.Y

- Always remember D.R.Y - Don't Repeat yourself
- If you find yourself typing the same thing twice in your code, you're doing something wrong!
  - It makes your code harder to read
  - Makes it more difficult to maintain
- Multiple ways you can achieve code reuse:
  - At the function level, condense your logically branches that have the same outcome into switch statements that have multiple cases routing to the same logic.
  - At the class/file level, make your reused code into a helper function.
  - At the project level, make your reused code into a helper class that can be imported.
- Great developers are always looking for ways to avoid code reuse



EXERCISE - BREVITY IS THE W.H.Y.

Let's take a look at a code snippet from our API

Can you see any areas in this code that you could streamline. Any repeated codes? Any areas that could be better spelled out in shorthand?

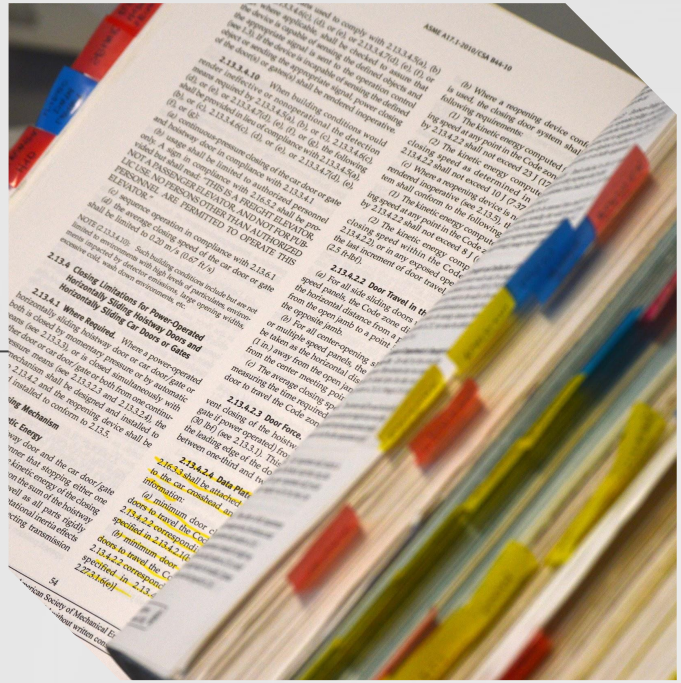
07

Zen and the art of commenting

No comment....

WHY DID MY TV COME WITH A NOVEL?

Some things are so confusing, the manual needs a manual. Some things are so simple that they don't need a manual at all. Your code should be more like that latter than the former.



NO COMMENT

- First things first - **GOOD CODE COMMENTS ITSELF**
- Different types of comments:
  - **Documentation Comments** - Provide API documentation for your code to be used by other developers that leverage your code as a dependency to theirs. Ex. JavaDocs, JSDocs, etc. **Use extensively,** especially if you know your code will be a dependency to another application.
  - **Logical comments** - Single line comments, usually used to comment on business logic or application logic. **Use sparingly.** If you find yourself using these, ask yourself if you can rename functions and variables around the comment to tell a better story.
  - **Process comments** - Single line comments, usually used to leave breadcrumbs for other developers on the team to understand where and why changes are being made. **Usage is up to the team.** There are a lot of tools that can help you document these things outside of the code. If it works for you, then it works. Just make sure your compiler/transpiler/runner removes these comments in the build.
- To put a pin in this issue - **GOOD CODE COMMENTS ITSELF**

08

Plan for failure

Users cannot be trusted

NOTHING ALWAYS WORKS AS INTENDED

Things fail, and even still, things are used in ways they weren't intended to.

USERS CAN'T BE TRUSTED

TYPE SAFETY

Are you using a type safe language? If not, are you checking each value?

POTENTIAL FAILURES THAT MUST BE ANTICIPATED

Once your code is in another person's hands, you can't control how they use it. You have to anticipate potential failures and mitigate them.

PARSING SAFETY

When you're parsing a value, are you checking to make sure it's structured the way you expect?

NULL SAFETY

Are you checking all values passed into your function to make sure they exist?

EXERCISE - USERS CAN'T BE TRUSTED

Let's take a look at a code snippet from our API

Can you identify any areas of code that could use some validation logic? What potential issues are we introducing if those validations aren't added?

CODING - SUMMARY

04. Write easy to read code

Cognitive Complexity is everything. Write code that is easy to read and understand so that your team members and your future self will thank you.

05. Use logical/consistent function and variable names

Function/variable names aren't the time to be cute or funny. Functions do something. The name should tell you what it does.

06. D.R.Y.

Write it once, use it everywhere. If you've used the same logic twice, you've used it one too many times.

07. Zen and the art of commenting

Good code comments itself (see above rules). Good comments help you not have to read the good code.

08. Plan for failure

The best developers have a tragic imagination. Always think of the worst case scenario for your code and plan for it!

TESTING PRACTICES

DEVELOPMENT SCOPING/PLANNING - SUMMARY

09. Write small, tightly scoped tests

10. Decouple your functionality

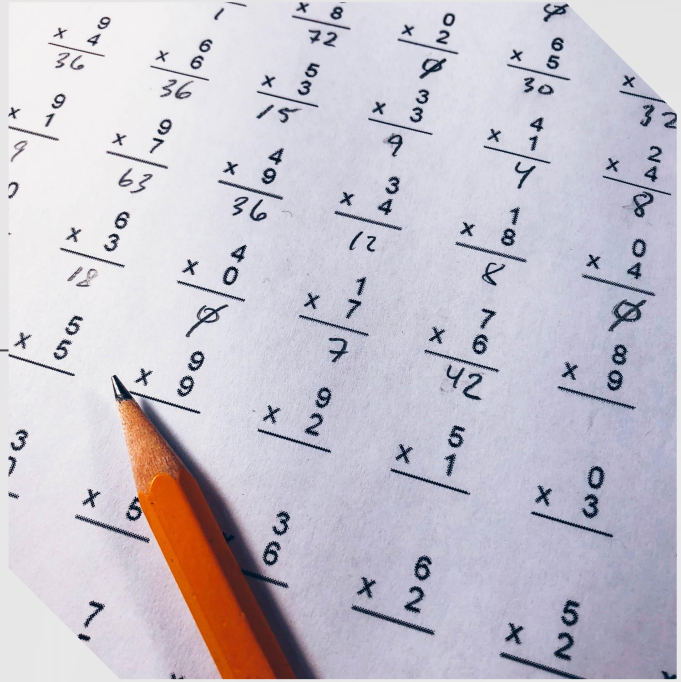
09

Write small, tightly scoped tests

Assert yourself! But only once per test.

ONE QUESTION, ONE ANSWER

On every test we've ever taken, each question we answer only has one answer.



ASSERT YOURSELF! BUT ONLY ONCE PER TEST

- Software Unit Tests are meant to test the logical functionality of the code we create
  - The tests should validate that the code “understood” the business process we’re implementing
    - If you’re creating a calculator that can perform multiplication, your unit test should assert that “6x4 = 24” is true.
- Software Unit Tests have two primary properties
  - Small, tight scope - Unit tests should test a single logical branch of your code. Just like our example, one question.
  - Single Assertion - Unit tests should only test one assertion for that logical branch of code. Just like out example, one answer.
- This tight scope allows developers to be confident in their test results.
  - If your test fails, you know exactly where the failure occurred - in the logic that test was meant to test.
  - If you get one question wrong on an exam, you know exactly what you need to study for the next exam.
- These tests are instrumental to developing a stable, effective system.

EXERCISE - ASSERT YOURSELF! BUT ONLY ONCE PER TEST

Let's take a look at a code snippet from our API

Can you identify any tests that are doing too much? How could we make those tests more tightly scoped?

10

Decouple your functionality

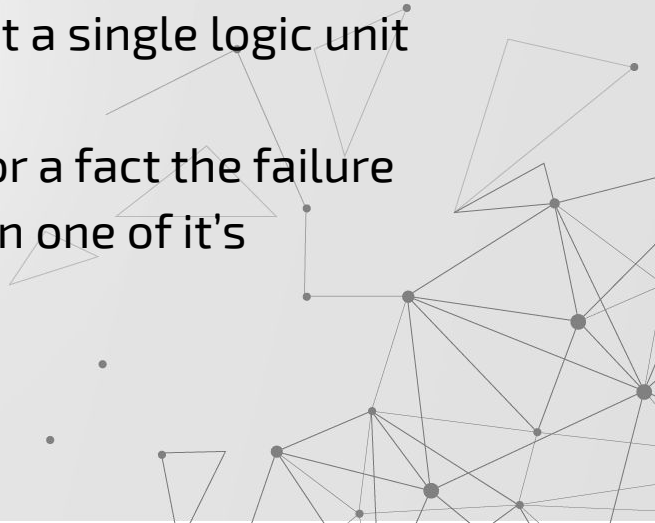
One is functioning, two is a crowd

TROUBLESHOOTING IS AN ART

How many of us have ever been in a situation where our phone wasn't charging on our regular charger?

ONE IS FUNCTIONING, TWO IS A CROWD

- To make our code as effective as possible for unit testing, it is important that our logic be **decoupled.**
  - This means that each of the logical functions are separated from each other and can be run independent of the other logical functions
- Decoupling allows your unit tests to run against a single logic unit without any other **dependencies**
  - If there is a failure in your test, you know for a fact the failure happened in that single logic unit and not in one of it's dependencies.



EXERCISE - ONE IS FUNCTIONING, TWO IS A CROWD

Let's take a look at a code snippet from our API

Can you identify any code that has tightly coupled functionality? How can we separate this functionality to make these tests more tightly scoped?

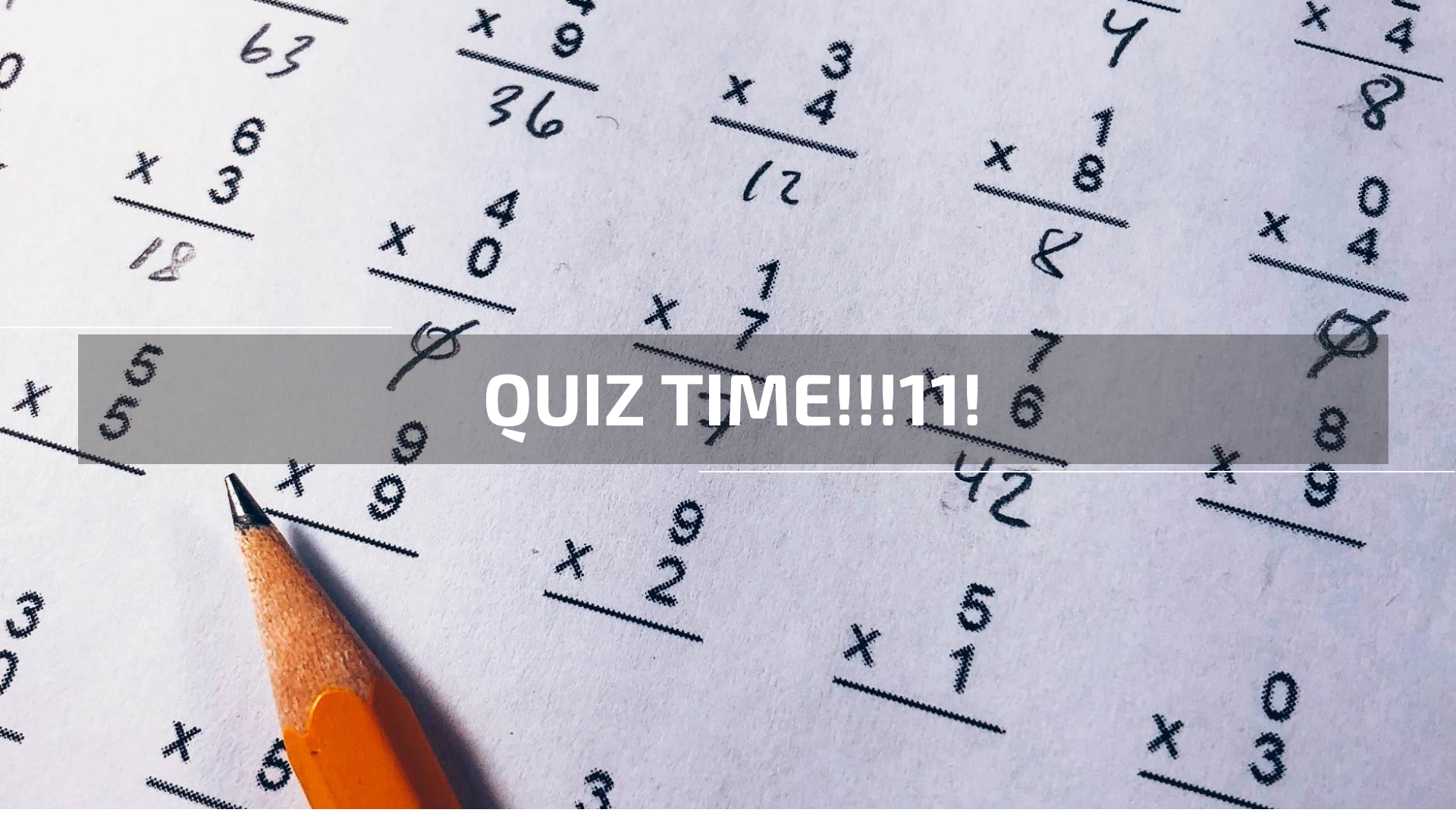
DEVELOPMENT SCOPING/PLANNING - SUMMARY

09. Write small, tightly scoped tests

Each test should only test one unit of logic or integration. No more. No less. If only school were that easy……

10. Decouple your functionality

Your functions should perform logic or integration, but not both. Tightly coupled functions mean loosely scoped tests. Not even school was that hard……..


QUIZ TIME!!!11!

DEVELOPMENT SCOPING/PLANNING - BEST PRACTICES

01. Think/Plan before writing code

02. Develop Iteratively

03. Prioritize working code

CODING - BEST PRACTICES

04. Write easy to read code

05. Use logical/consistent function and variable names

06. D.R.Y.

07. Zen and the art of commenting

08. Plan for failure

DEVELOPMENT SCOPING/PLANNING - SUMMARY

09. Write small, tightly scoped tests

10. Decouple your functionality

THANKS

All today's code and this presentation are available on github @
https://github.com/budoudoh/coding-best-practices