O'REILLY®

Becoming a Better Programmer

A HANDBOOK FOR PEOPLE WHO CARE ABOUT CODE

Credits:

Pete Goodliffe.

Recommended for training purpose - Aashish

Becoming Better Programmer.

Part - 1

Care about the code.

"From caring comes courage"

– Lao Tzu

- Bad programmers produce elephantine monstrosities which others have to clean.

- “Attitude” - The only difference between good and mediocre programmer.

- Avoid fixing problem with “hacks.”

- Write intent revealing, maintainable, correct code.

- Leave code better than you found it.

There is nothing wrong with an emotional response to code. Being proud of your great work, or disgusted at bad code is healthy.

Keeping up appearances.

"Appearances are deceptive."

– Aesop

- No one likes to work with messy code.

- Stop bickering about which editor is best, Tabs vs spaces, Brace positioning, empty lines, column per lines, capitalisation.

- Stop doing dysfunctional code review about layout.

- Don’t feel proud by adding 10 comments on brace positioning in code review.

- Good code is clear, reveals intent and layout is almost invisible.

- Good code presentation reveals your code's intent. It is not an artistic endeavour.

- It is all about communication.

What to check in appearance?

- Code structure - functions length , order etc.
- Consistency.
- Naming convention
- Redundancy?
- Accuracy.

Never alter presentation and behaviour at same time. Make different branches.

Write less code.

"A well-used minimum suffices for everything."

– Jules Verne

- Less code can mean mean more software.

- Unnecessary code is nefarious. Can be unused component, dead code, pointless comments, unnecessary verbosity.

- Express code clearly and succinctly. Avoid unnecessarily long-winded statements.

- Duplication - Never.

- Do not copy code sections. Factor them into common functions.

- Remove dead code.
- Consider comments as your inability to express yourself in code.
- Comment should only explain “why” but only if it is not clear.
- Do not remove code by commenting out. Use version control for that.
- Get comfortable with Ternary operator. (?:)

- Over the time code changes, so may become redundant, dead code. Remove that.

- Pigs live in their own filth. Programmers needn't. Clean up after yourself.

- Every day, leave your code little better than it was. Remove redundancy and duplication as you find it.

Improve code by removing

We ascribe beauty to that which is simple;
which has no superfluous parts, which exactly
answers its end..”

– Ralph Waldo Emerson

- You can improve a code by adding new code. You can also improve a system by removing code.
- Do not write extra code if you do not need it now.
- Do not fall into the trap “It is just small thing.”
- Do no invent features yourself.
- Remove dead code whenever possible. It gets in the way and slows you down.
- Code clean up in separate commits.

Ghost of codebase past

“I will live in the past, present, future.
The Spirit of all three shall strive in me.
I will not shut out the lessons they teach.”

– Charles Dickens

- Looking back at your code will inform you about improvement (or otherwise) in your coding skills.
- learn by doing. Reading is not enough.

- Study a small piece of code. Critique it. Determine weak spots. Refactor it. Mercilessly.

Wallowing in Filth

“As a dog returns to its vomit, so fools repeat their folly.”

– Psalms

- Be prepared to encounter bad code. Fill your toolbox with sharp tools to deal with it.

- Silence the feeling of revulsion when you encounter “bad” code. Instead, look for ways to practically improve it.

- Pick your battles.Consider carefully whether you should invest time and effort in tidying-up the code. It may be pragmatic to leave it alone right now.

- Boy scout rule. Leave the code better than you found.

- Make code changes slowly and carefully. One change at a time.

Do not ignore that error.

“All you need is ignorance and confidence and success is sure.”

– Mark Twain

- Treat warnings as errors.
- But error handling mechanism in place right from beginning.
- Use exceptions well, with discipline.
- Use threads carefully.
- Consider all possible things that can go wrong.
- Well why do we ignore error handling in first place?

Bug Hunting.

"If debugging is the process of remove bugs from system,
then programming must be the process of putting them In"

– Edsger Dijkstra

- Bug hunting - Cost is too high.
- An ounce of prevention is worth a pound of cure.
- Avoid injecting bugs into your code by employing sound engineering practices.
- “Debugging is twice as hard as writing the code in first place.” - Brian Kernighan

How to debug?

- Reduce it to simplest reproduction steps possible.
- Tackle one issue at one time.
- Determine how repeatable the problem is.
- Lay traps.
- Don’t litter the code.
- Use Binary Chop strategy.

- Software Archaeology - Historical records in version control.
- Untested code is breeding ground for bugs. Test your bleach.
- Learn how to use debugger but use it at right times.
- Fix bugs as soon as you find them. Do not let them pile up.
- Don’t waste too much time on one bug. Take a break. Come back with fresh perspective.
- Don’t rush away after fixing the issue. Check is similar issue exists or not.


I'M NOT LOOKING FOR BUGS
IT'S FAR WORSE THAN THAT
THIS CODE IS INFESTED WITH CRABS

Testing Times.

“Quality is free but only to those who are willing to pay for it.”

– Timothy Lister

- Test Driven Development (TDD) - Nice Idea? Waste of effort? Hmm..Lets find out.

- Testing on developer end shortens feedback loop.

- Manual testing is time consuming.

- Write tests as you code. Do not postpone test writings, or your test will not be as effective.

- Unit Test, Integration test, System test.

- You might inject tests directly into build process. So if any test fails project will not combine.

- Bad tests can be liability. They can impede effective development.
- Characteristics of good tests -
- Short, Clear Name.
- Maintainable.
- Runs quickly.
- Up-to-date.
- Does not depends on any other tests
- Should run in production too.

- Bad tests -
- Tests that sometimes run sometimes fails.
- Tests that look awful and are hard to read.
- Tests that are too large.
- Tests for third party codes that you don’t write.
- Test that actually do not cover main functionality.
- Test that cover pointless functionality with excruciating details. (Testing getters and setters?)
- Test that work only on one machine.

- The Structure of tests -
- Covers all important functionality.
- Consider failure cases.
- Consider boundary values.
- No Duplicates.
- Check behaviour of code; not every single functions.
- Maintain your test suits and listen to it when it talks to you.
- Isolated test.

A Tale of two system.

“Architecture is an Art of how to waste space.”

– Philip Johnson

Just Read the chapter from book.

TIME IS NOT A GREAT HEALER

EXAMPLE #1:
PICKING UP SOME OLD SOURCE CODE