School of Tech

part of accenture
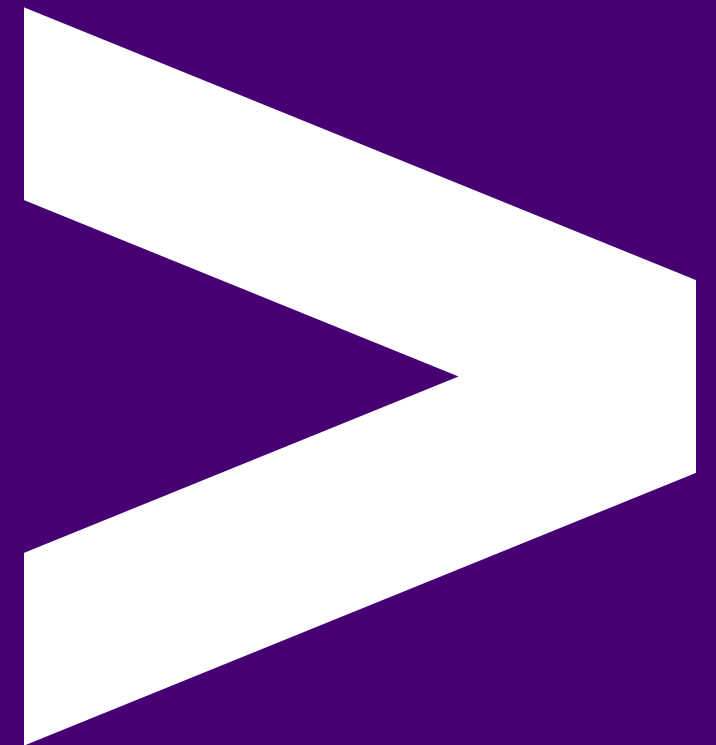
Programming Practices

School of Tech
part of accenture >

Overview

- Introduction to Programming Practices
- The Importance of Documentation
- Tech Debt and How to Reduce it

Learning Objectives

- Discover the features of clean, well written code
- Identify why documentation is important
- Explain what refactoring is
- Define what tech debt is and learn how to reduce it
- Discover the importance of code reviews

School of Tech
part of accenture >

Writing code like a pro

> Programs must be written for people to read, and only incidentally for machines to execute. -- Harold Abelson

and

> You know you are working on clean code when each routine you read turns out to be pretty much what you expected. -- Ward Cunningham

School of Tech

part of accenture >

Clean code is easier to...

- Understand
- Reuse
- Test
- Refactor
- Extend

Key Ideas

Functionality: does it work?

Testability: are you sure it works?

Readability: can everyone understand how it works?

Bad code tries to do too much; it is often muddled and ambiguous.

Clean code is focused. Each function, class, and module does a single, clear, obvious thing.

The Acronyms of Clean Code

- KISS
- DRY
- YAGNI
- RTFM

Programming Practices

Much like the humble piano player, it takes time to refine our abilities to practice clean code practices.

School of Tech
part of accenture

Code Layout

- Indentation is essential
- Whitespace is free
- Keep your lines short
- Keep related elements together
- Remove dead code aggressively

School of Tech

part of accenture >

Is this good code?

Open the example code snippets provided in `handouts/bad-code.py`.

> Discuss: What do we think about these examples?

Is this good code?

Open the example code snippets provided in `handouts/good-code.py`.

> Discuss: Are these better? If so, why?

Emoji Check:

Has looking at the bad and then good examples helped? Say so if not!

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😃 Yes, enough to start working on it collaboratively

Variables

- Avoid global variables
- Use intention-revealing names
- Longer names are not bad if they're meaningful (but don't overdo it)
- If you feel the need to have a really long variable name, your variable is probably trying to do too much.
- Immutable variables make for applications which are easier to follow and test

Discussion - 5m

Say we have some code running a `for` loop over a list of animals.

Can we think of some bad variable names?

How about some good ones?

Functions

- Keep them small
- Functions should only do one thing
- Functions should have no side effects
- Function names should be verbs
- Should not affect the state of the application directly

Discussion - 5m

Say we have a function to add up two numbers, or multiply them together.

Can we think of some bad function names? How bad can you make them?

How about some good ones?

Emoji Check:

Do you feel you have some idea about naming variables and functions now? Say so if not!

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😃 Yes, enough to start working on it collaboratively

Function Side-Effects

A function has "side effects" if it makes an observable change to the application state other than just providing its return value.

Side effects tend to muddy the waters when it comes to ease of testing and ease of understanding, so they're usually avoided as far as possible.

Necessary Side-Effects

Sometimes, side effects are necessary.

Can we think of any?

- Accepting user input
- Rendering to the screen (or terminal)
- Reading and writing Files
- Database access
- API Calls

Emoji Check:

Do you feel you know what "side effects" are and why we try and avoid them?

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

Documentation

- Include what adds value for understanding
- Code comments are good, but also lean towards self-documenting code
- Include a README. Tell people how to install, set up, use, and (where applicable) contribute code to your application

Self-Documenting Code

Is this self-documenting?

```
g = 9.81
t = 5
d = 0.5*g*t**2
```

Self-Documenting Code

Is this better? If so, how?

```python
gravitational_acceleration = 9.81
time = 5
distance = 0.5 * gravitational_acceleration * (time**2)
```

Self-Documenting Code

Is this even better? If so, how?

```python
def calculate_free_fall_distance(time):
    gravitational_acceleration = 9.81
    distance_meters = 0.5 * gravitational_acceleration * (time
    return distance_meters
```

Quiz Time! 🤓

Which of these statements about variables is false?

1. Global variables should be avoided
2. Variables should have intention-revealing names
3. Longer variable names are fine if they're meaningful
4. Mutable variables make for applications which are easier to follow and test

Answer: 4

Which of these statements about functions is false?

1. Functions should be small

2. Functions should only do one thing

3. Functions should have side effects

4. Function names should be verbs

Answer: 3

Refactoring

When we start working on a new feature and identify issues in the code we're going to be working on, it's a good idea to refactor the code first and develop the feature after.

Good test coverage is essential to refactoring confidently and effectively. Otherwise, welcome to regression hell...

We refactor code when we improve the way our code is written without affecting the end result.

Refactoring is a gradual process. Do it as required.

School of Tech

part of accenture

A Short List Of Dangerous Things to Do In Life:

1. Cliff jumping

2. Shark diving

3. Refactoring without good test coverage

Technical Debt (aka Tech Debt)

Deadlines and priorities will sometimes force us to take shortcuts and implement quick-and-dirty solutions for our problems.

Tech debt reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer.

Every time we take one of these shortcuts, we increase our tech debt.

Why Is Tech Debt Bad?

- Introduces uncertainty
- Balloons time and effort estimates
- Slows down development
- Increases likelihood of regressions

Much like regular debt, tech debt accrues "interest" in the form of slowing down other development.

Tech debt should be recorded as and when it's noticed - unknown or untracked tech debt is even worse for causing unexpected delays.

Causes of Tech Debt

- Lack of documentation

- Delayed refactoring

- Business pressures causing rush

- Lack of technical understanding

- Lack of collaboration - knowledge isn't being shared around

- Many many more!
  https://en.wikipedia.org/wiki/Technical_debt#Causes

Ensuring Code Quality

Coding Standards

How do you enforce coding standards?

- Team consensus
- Document them (or follow a public style guide)
- Code reviews
- Linters
- Consistency, consistency, CONSISTENCY!

Code Reviews

- Reviewing some code written by a peer to ensure it meets the team's quality standards
- Concentrate on the important stuff, let the tools pick up on the minutiae
- Does the code appear to do what it's supposed to do? Code structure? Is the code readable? Are the tests good? Is the test coverage good?

Quiz Time!

What is refactoring?

1. Improving the way our code is written which affects the end result.

2. Improving the way our code is written without affecting the end result.

3. Improving our ability to ensure tech debt is kept to a minimum.

4. The implied cost of additional rework.

Answer: 2

Which comment is best?

```python
# comment to go here
date_config = get_date_config(event)
for target in targets:
    target['date'] = date_config
```

1. Update targets.

2. Attach the date configuration to each target.

3. Here we are getting the date config for a particular event. As we iterate through each target, we assign a key-value pair, with `date` being the key, and the date config being the value.

4. None. It's self-documenting.

Answer: 2

School of Tech

part of accenture >

Terms and Definitions - recap

Documentation: Any communicable material that is used to describe, explain or instruct.

Tech Debt: Reflects the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer.

Terms and Definitions - recap

Linting: Analysis of your code to find inconsistencies.

Code Review: Reviewing some code written by a peer to ensure it meets the team's quality standards.

Further Reading

School of Tech
part of accenture >
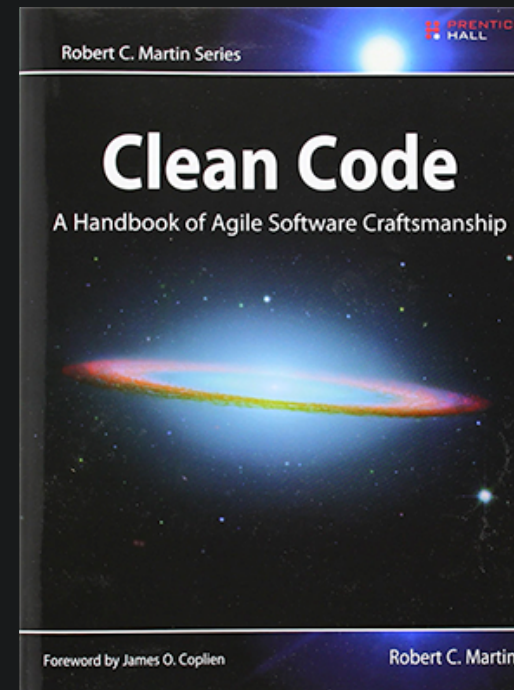
Overview - recap

- Introduction to Programming Practices
- The Importance of Documentation
- Tech Debt and How to Reduce it

School of Tech
part of accenture >

Learning Objectives - recap

- Discover the features of clean, well written code
- Identify why documentation is important
- Explain what refactoring is
- Define what tech debt is and learn how to reduce it
- Discover the importance of code reviews

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 🙁 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively