Software Engineering Best Practices for Busy Experts

Software Engineering Best Practices for Busy Experts is a course designed to help scientists, engineers and other domain experts to learn the most important software engineering best practices to help them write correct code more efficiently.

The course consists of 4 seminars:

Seminar A: Automated Tests

Seminar B: Good Code

Seminar C: Code Review

Seminar D: Continuous Integration

These seminars take the form of a 45 minute presentation plus plenty of time for questions. Each of the seminars will feature plenty of practical examples to see how these techniques can be applied.

Seminar A: Automated Tests

In this seminar I will be discussing automated tests, including why they are important. It will also contain a detailed section with examples of code that is tricky to test, and how one would go about testing them.

Title	Contents
What tests are.	An introduction into testing in the context of software engineering, and terms for different kinds of tests
Why tests are valuable.	Covers some of the advantages of writing tests, including: ● Catching bugs ● Quicker iteration time on development ● Refactoring code safely and quickly ● Documenting expected behavior
How to write tests.	A primer on what a very simple test looks like, then a deeper dive into the common reasons why writing tests can be tricky: ● Dependencies of external systems like databases or machines ● Dependencies on other complex code systems ● Global states ● Hard to observe outputs These would be tailored to examples that have been seen at the company and ideally people could provide examples of code that has proven difficult to test. For each of these examples, I would explain how to use modern testing practices to test the code, with minimal modification.
When to write tests.	Deciding an appropriate level of testing. How to use tests to speed up writing new code.
Running tests	Running the tests locally.

Seminar B: Good Code

In this seminar I would provide some guidance on how to write code that is “good”. This module is expected to be dependent on the code issues identified in the investigation, but I’ve produced a rough syllabus based on typical software engineering problems.

Title	Contents
A working definition of good code.	Motivation for caring about good code over code that works. Discuss the qualities of good code: ● Readable ● Testable ● Maintainable
How to make code readable.	Look at some common ways code can be hard to read, and how to rewrite the code to be more straightforward. This would include things like naming, complex code structure, patterns.
How to make code testable.	Look at some common code patterns that make code hard to test, and how to rewrite the code to avoid these patterns. This would include dependencies on external resources, functions that depend on global state and functions that have hard to observe outcomes.
How to make code maintainable.	Look at ways in which code can become hard to maintain, and how to write code that avoids that. This would include tightly coupled systems, and poorly understood code.

Seminar C: Code Review

In this seminar I will explain how to use a code review process to effectively improve code quality and share best practices. I will also cover how to make code easily reviewable.

Title	**Contents**
What a code review process is.	Explain what a code review process is. Explain the advantages a code review process can bring.
What to look for in a code review.	Covers what a code reviewer should (and should not) be looking for when reviewing code.
How to make your code easy to review.	Covers how to make code changes as easy to review as possible.

Seminar D: Continuous Integration

In this seminar I will explain what Continuous Integration (CI) means, why it is useful and some practical considerations,

Title	Contents
What Continuous Integration (CI) is and does.	Explain what is meant by Continuous Integration (and related terms). Cover the main benefits for having a CI system. Cover the main steps a good CI would do and why they are useful.
How to set up a basic CI on Gitlab.	Cover what prerequisites there are for setting up a CI. Cover how to set up a CI on Gitlab.
How to use a CI system.	Cover how the developers will interact with a CI system once it has been set up. In particular, what to do when it fails.
Maximizing the value of a CI system.	Cover best practices and common pitfalls when setting up a CI system.