

Clean Code Fundamentals

Function Structure

Pre-work

- Video: <https://cleancoders.com/episode/clean-code-episode-4>
- Exam: <https://cleancoders.com/episode/clean-code-episode-4/exam>

Chapters

Chapter	Time
Overview	00:00:55
Fusion	00:04:29
Arguments	00:09:09
Three Arguments Max	00:10:10
No Boolean Arguments Ever	00:12:19
Innies not Outies	00:14:00
The Null Defense	00:15:27
The Stepdown Rule	00:17:22
Switches and Cases	00:28:24
Paradigms	00:40:51
Functional Programming	00:41:31
Side Effects	00:43:28
Command Query Separation	00:47:28

Chapter	Time
Tell, Don't Ask	00:51:35
Structured Programming	00:56:32
Early Returns	01:00:13
Error Handling	01:02:55
Errors First	01:06:30
Prefer Exceptions	01:08:04
Exceptions are for Callers	01:05:49
Use Unchecked Exceptions	01:09:49
Special Cases	01:15:17
Null is not an Error	01:19:59
Null is a Value	01:24:23
Trying is One Thing	01:27:09
Conclusion	01:28:00

Timetable

Activity	Time
Warmup	5 min
Exercise 1	20 min
Exercise 2	20 min
Exercise 3	20 min
Wrap up	5 min

Warmup

- In your practice, what do you find the most useful technique to organize code within a function or a class?
 - Type in the meeting chat

Exercise 1

- Prompt
 - Collaborate to build the list of principles and techniques you learned from the video episode.
 - You must have at least 10 principles and techniques.
- Time limit: 10 minutes

Possible answer

- Function signature should be small - 3 or less arguments
- Avoid “output” arguments
- Avoid passing boolean values and null
- Limit the use of switch statements to top-level factory functions
- Limit the interdependencies by using the principle of the least knowledge
- “Pass a block” to solve the temporal coupling problem
- Use early returns to reduce the nesting level
- Avoid breaks/returns in a middle of a loop
- Prefer exceptions to error codes
- Separate commands and queries
- Tell, don't ask

Design patterns

- Definition
 - A general reusable solution to a commonly occurring problem within a given context in software design.
 - A template for how to solve a problem that can be used in many different situations.
- Examples
 - Null Object
 - Factory
- Categories
 - Creational
 - Structural
 - Behavioral
 - Concurrency
- Catalog
 - Design Patterns
 - Software design pattern

Exercise 2

- Prompt
 - Introduce categories to split items from Exercise 1 into
 - Make sure to create at least 3 categories
 - Make sure to create an effective list!
- Time limit: 10 minutes

Possible answer

- Simplify function signature
 - Function signature should be small - 3 or less arguments
 - Avoid “output” arguments
 - Avoid passing boolean values and null
- Reduce coupling
 - Limit the use of switch statements to top-level factory functions
 - Limit the interdependencies by using the principle of the least knowledge
 - “Pass a block” to solve the temporal coupling problem
- Clarify control flow
 - Use early returns to reduce the nesting level
 - Avoid breaks/returns in a middle of a loop
 - Prefer exceptions to error codes
- Clarify state management
 - Separate commands and queries
 - Tell, don't ask

Exercise 3

- Prompt
 - Select top 3 principles and techniques from Exercise 1 by the highest ROI
 - High return, low time effort cost
 - Refer to your experience, if applies
 - Explain and justify your choice
- Time limit: 10 minutes

Possible answer

1. User early returns
 - Low effort
 - Clarifies the control flow
2. “Pass a block” to solve the temporal coupling problem
 - Medium effort
 - Helps to avoid critical bugs in resource management
3. Avoid “output” arguments
 - Medium effort
 - Make code more readable and less error-prone

Summary

- Simplify function signature
 - Function signature should be small - 3 or less arguments
 - Avoid “output” arguments
 - Avoid passing boolean values and null
- Reduce coupling
 - Limit the use of switch statements to top-level factory functions
 - Limit the interdependencies by using the principle of the least knowledge
 - “Pass a block” to solve the temporal coupling problem
- Clarify control flow
 - Use early returns to reduce the nesting level
 - Avoid breaks/returns in a middle of a loop
 - Prefer exceptions to error codes
- Clarify state management
 - Separate commands and queries
 - Tell, don't ask

Wrap-up

Call to action!

Next 7 days focus on using the techniques from this episode in your day-to-day work.

What is next?

- Expect an e-mail with instructions for upcoming coding dojo

Final words

Always leave the code better than you found it.
– *The Software Craftsmanship Rule*