# C++ Software Engineering

*for engineers of other disciplines*

Module 8

*"Software Engineering"*

*5th Lecture: Anti-Pattern*

ALTEN

*Spring 2022*

*Gothenburg, Sweden*

# Anti-pattern



ALTEN

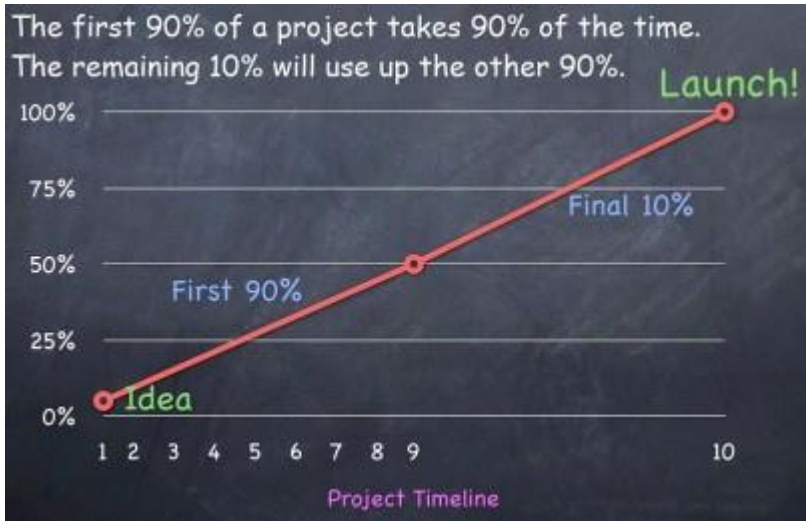
- Anti-pattern is a common response to a recurring in effective and highly counterproductive problem.
- The usage has been extended beyond software design:
  - Organizational
  - Project Management
  - Programming
  - Methodological
  - Configuration Management

> *"Two key elements present to formally distinguish an actual anti-pattern from a simple bad habit, bad practice, or bad idea (are):*
>
> *1. A commonly used process, structure, or pattern of action that despite initially appearing to be an appropriate and effective response to a problem, **has more bad consequences than good ones**.*
>
> *2. **Another solution exists** that is documented, repeatable, and proven to be effective."* https://en.wikipedia.org/wiki/Anti-pattern#Definition

# Anti-pattern -- Planning



- ***Cart before the horse***: prioritizing tasks irrationally
- **Ninety-ninety rule**: underestimating the amount of time to complete a project when it is "nearly done"



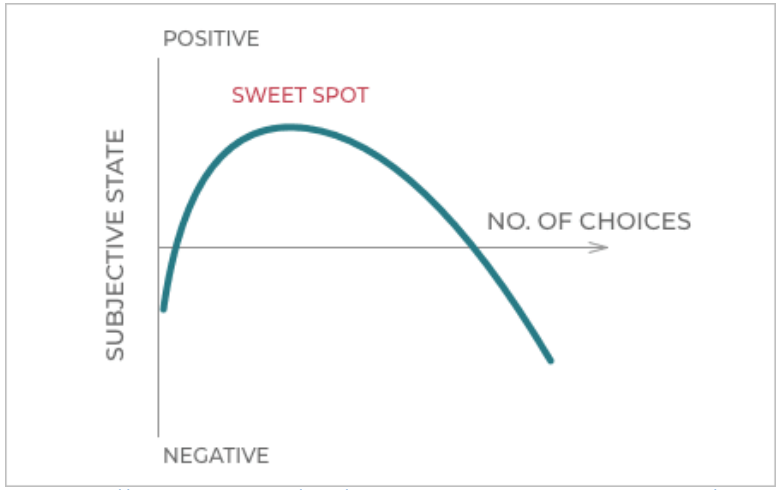

http://alvares.in/mario/20101217/the-ninety-ninety-rule-in-projects/

https://medium.com/@socialtrendspot/cart-before-the-horse-marketing-strategy-vs-tactics-a26f360d35c3

# Analysis Paralysis



- A scenario in which *overthinking* “paralyzes” decision making.
- Iterative, incremental, and agile have came to rescue!
  - If the iteration plans have no *meaningful* connection?
- To overcome:
  - Undrestand the *main* objectives.
  - Start early, prioritize *tasks* ***appropriately*** and set a deadline!
  - Limit the inputs, intentionaly.
  - Decompose problem and iterate on solutions!
  - Do not worry about being wrong.
  - Seek help.




https://www.typetalk.com/blog/8-steps-stopping-analysis-paralysis-tracks/

- Overthinking kills performance, causes stress which lowers creativity, and can cause paradox of choice shown above.
- Common reasons resulting Analysis Paralysis include:
  - *Fear of deciding*
  - *Lack of experience or extensive experience*

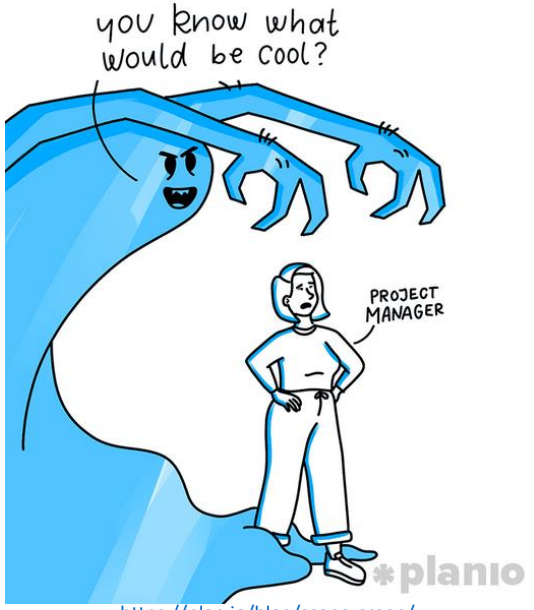
# Anti-pattern -- Design



- ***Circle–ellipse problem***
- ***Overengineering***: making a project more robust and complex than is needed

https://morioh.com/p/03b62ca64435

- ***Scope/Feature/Requirement creep***: adding new features to the project after the original requirements have been drafted and accepted
- ***Action at a distance***: unexpected interaction between widely separated parts of a system



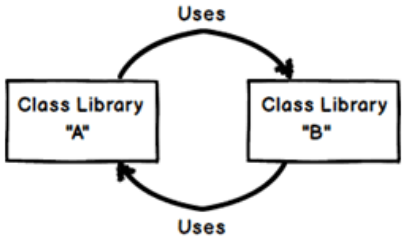

https://plan.io/blog/scope-creep/

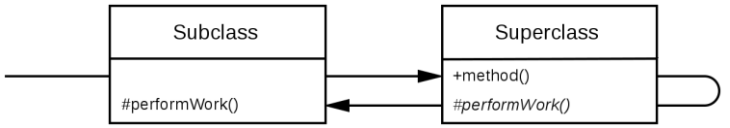
- Circle–ellipse problem is sometimes used to criticize object-oriented programming. It also exposes the difficulties of adapting hierarchical taxonomies universally, advocating *situational classification systems.*
- *Action at a distance* usually happens when using global variables.

# Anti-pattern – OO Design



- ***Call super***: requiring subclasses to call a superclass's overridden method
- ***Circular dependency***: introducing unnecessary direct or indirect mutual dependencies between objects or software modules
- ***Constant interface***: using interfaces to define constants
- ***God object***: too many functions in a single part of the design (class)
- ***Object orgy***: failing to properly encapsulate objects permitting unrestricted access to their internals




- *Call super* on constructor happens in C++ for every inheritance.

# Anti-pattern – Group Work



- ***Design by committee***: many contributors with no unifying vision.
- ***Escalation of commitment***: failing to revoke a decision when it proves wrong.
- ***Moral hazard***: Insulating a decision-maker from the consequences of their decision
- ***Groupthink:*** the desire for harmony or conformity in the group results in an irrational or dysfunctional decision-making outcome.



Bill Mauldin. *The Sun-Times* (Chicago), 1966.
Copyright 1966–Chicago Sun Times. Reproduced by courtesy of Wil-Jo Associates, Inc., and Bill Mauldin.

# Anti-pattern -- QoS





- ***Input kludge***: Failing to specify and implement the handling of possibly invalid input
- ***Race hazard***: Failing to see the consequences of events that can sometimes interfere with each other
- ***Premature Optimization***: Coding early-on for perceived efficiency, sacrificing good design, maintainability, and sometimes even real-world efficiency

# Anti-pattern -- Coding



- ***Magic numbers*** & ***Magic strings***
- ***Hard code***: Embedding assumptions about the environment of a system in its implementation
- ***Loop-switch sequence***: Encoding a set of sequential steps using a switch within a loop statement


```
// parse a key and value
String key = stream.parse();
String value = stream.parse();

// parse 3 parameters
List<String> params = new LinkedList<String>();
for (int i = 0; i < 3; i++) {
    params.add(stream.parse());
}
```

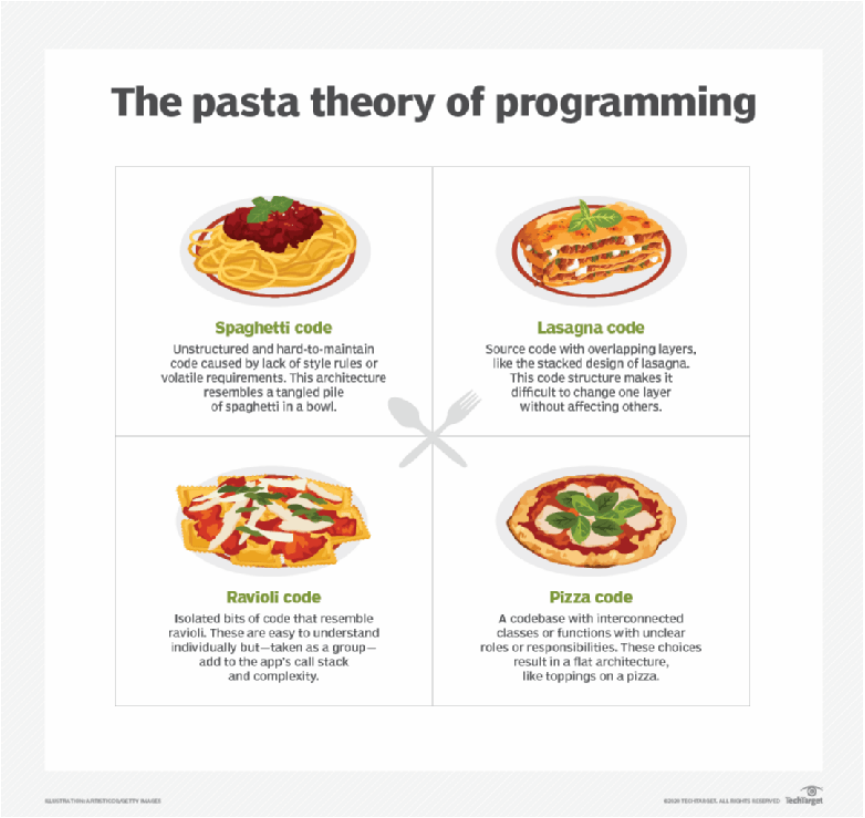
```
// parse a key, a value, then three parameters
String key = null;
String value = null;
List<String> params = new LinkedList<String>();

for (int i = 0; i < 5; i++) {
    switch (i) {
        case 0:
            key = stream.parse();
            break;
        case 1:
            value = stream.parse();
            break;
        default:
            params.add(stream.parse());
            break;
    }
}
```

# Anti-pattern – Code Structure



- ***Ravioli Code***: well-structured classes that are easy to understand in isolation, but difficult to understand as a whole.
- ***Spaghetti Code***: unstructured and difficult-to-maintain source code
- ***Lasagna Code***: too many layers of inheritance
- ***Pizza Code***: unclear reason




https://searchsoftwarequality.techtarget.com/tip/Fix-spaghetti-code-and-other-pasta-theory-antipatterns

# Code Smell – Class Level



ALTEN

- ***Large class***: See God object.
- ***Lazy class / freeloader***: a class that does too little.

- ***Feature envy***: a class that uses methods of another class excessively.
- ***Inappropriate intimacy***: a class that has dependencies on implementation details of another class. See Object orgy.

- ***Orphan variable or constant class***: a collection of constants which belong elsewhere
- ***Data clump***: a group of variables which should  compose an object are passed around together in various parts of the program

# Code Smell – Method Level



ALTEN

- ***Excessively long identifiers***
- ***Excessively short identifiers***

- ***Too many parameters***: the purpose of the function is ill-conceived and that the code should be refactored
- ***Cyclomatic complexity***: too many branches or loops – broken into more *simpler* functions
- ***Long method***: a method, function, or procedure that has grown too large

- ***Excessive return of data***: a function or method that returns more than what each of its callers needs.
- ***Excessively long line of code*** (or ***God Line***)