Published in Level Up Coding

Shalitha Suranga Follow
Apr 19 · 7 min read ★ · Listen

How to Program Simply by Not Using Object-Oriented Programming

OOP is great, but it silently makes your simple programs complex

Photo by Joshua Aragon on Unsplash, edited with Canva

Programmers use various programming paradigms for developing software systems.

example, we can use the procedural paradigm by arranging statements into several procedures (functions), and we can use the DRY and YAGNI principles to improve the source code further.

Modern programmers often tend to use OOP (Object-Oriented Programming) without even considering the project domain, requirements, size, and scaling needs. OOP is undoubtedly a good paradigm to organize codebases well, but it can make your simple programs complex silently.

In this story, I will explain an alternative procedural-based programming paradigm that you can use to achieve simplicity in your software development projects. This alternative programming paradigm borrows several concepts from existing paradigms except for OOP.

How OOP Complicates Simple Projects

CPU is a computational device that follows the Turing machine concept. In other words, the CPU can theoretically execute endless series of linear Assembly instructions of a program. Initially, programmers wrote software programs with raw Assembly instructions, but later they introduced human-friendly programming languages for better productivity and code portability. I explained these low-level programming concepts in the following story:

5 Computer Hardware Concepts That Every Programmer Should Know

Your computer executed the program you wrote thanks to these concepts

levelup.gitconnected.com

Nowadays, we can use human-friendly programming languages with various paradigms — thanks to English-like syntax and modern language features. The OOP


with the `Customer` business entity with `TransactionBase` parent class:

```
28
29  class Customer(TransactionBase):
30      def get_feed(self):
31          return self.customer_name
32
33      def onload(self):
34          """Load address and contacts in `__onload`"""
35          load_address_and_contact(self)
36          self.load_dashboard_info()
37
38      def load_dashboard_info(self):
39          info = get_dashboard_info(self.doctype, self.name, self.loyalty_program)
40          self.set_onload("dashboard_info", info)
```

The ERPNext platform's Customer class implementation, screenshot by the author

Programming with the basic object-oriented approach doesn't make programs complex, but object-oriented (OO)-related concepts like inheritance, polymorphism, and OO-based design patterns typically do. On the other hand, we can't make better practical OOP-based source codes without such concepts. Therefore, OOP-based software projects can often become over-engineered with OOP principles and design patterns. As a result, your software program's simple logical flow can get complicated with multiple objects and scattered states. Modern programming languages like Golang don't offer complex OOP concepts support due to this reason.

> *Also, the lack of a type hierarchy makes "objects" in Go feel much more lightweight than in languages such as C++ or Java. — Go FAQ*

Sometimes, you have to even create classes and objects for entities that don't need multiple instances (i.e., Application, Server, Configuration, etc.) to adhere to OO-style. This situation often happens with non-business-oriented projects.

The Procedural Programming Paradigm for Software Projects

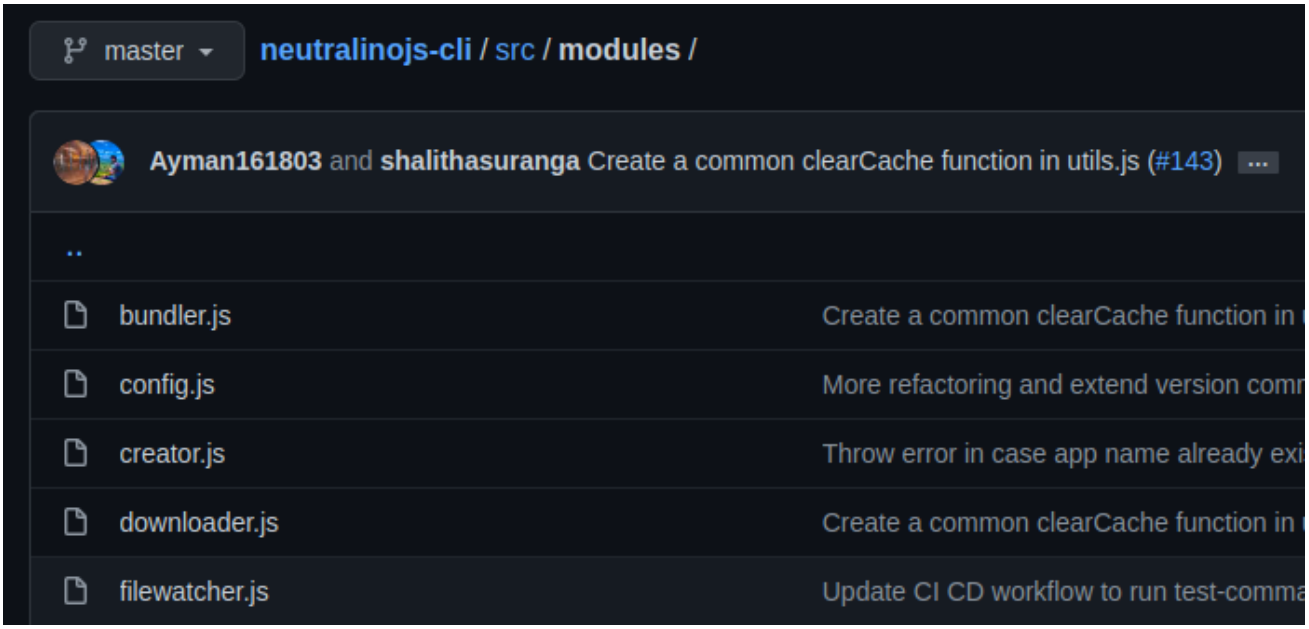
If OOP often complicates projects, which alternative paradigm is suitable for

manipulation, and mathematical calculations.

When and how did you learn computer algorithms first? — you probably started learning algorithms with flow charts by learning generic control structures. Later, you probably studied structured programming by using those control statements. Developing software with structured programming isn't complex since it has zero abstraction from theoretical computer science algorithms and less abstraction from the CPU's native language (ISA Assembly). We can combine concepts from structured, modular, procedural, and functional programming paradigms to make a simple alternative for OOP.

Here is a simple alternative for OOP:

We can use modules instead of classes, procedures (functions) instead of class methods (or classes), records instead of stateful objects, custom code styles instead of access modifiers, and module-level variables instead of persistent class states. For example, see how I decomposed a CLI program code into several CommonJS modules according to modular programming fundamentals:



An example of modular programming, a screenshot by the author

In OOP, we often distribute logic across class methods, then we have to inspect

the following example:

```
int main(int argc, char ** argv)
#endif
                                {
    json args;
    for (int i = 0; i < ARG_C; i++) {
        args.push_back(ARG_V[i]);
    }
    __initFramework(args);
    __startServerAsync();
    __configureLogger();
    __initExtra();
    __startApp();
```

Decomposing large functions into smaller private functions, a screenshot by the author

Managing the program state is easy and natural with OOP, but inheritance-like concepts can complicate the state handling flow. There are two approaches for managing the program state in procedural programming: with global variables or passing state as an argument — pick one according to your preference and requirement. The argument-based state handling produces well-testable and clean code. However, global variables-based state handling is not bad — it also produces good code as long as we don't alter the program state in many places.

Some non-looping programs (i.e., CLI programs, automation scripts, utilities, etc) typically don't need a persistent state, so we can write all functions according to the functional paradigm's pure functions concept without side effects.

This procedural programming alternative looks great, but how can we manage stateful object pools? For example, how can we implement a process list without an OOP-based `Process` class? In the procedural programming world, an object becomes a record. Look at the following example:

```
61
62  vector<string> getLoaded() {
63      return loadedExtensions;
64  }
65
66  bool isLoaded(const string &extensionId) {
67      return find(loadedExtensions.begin(), loadedExtensions.end(), extensionId)
68              != loadedExtensions.end();
69  }
```

Extension management module of the Neutralinojs framework, a screenshot by the author

The above C++ code snippet is a part of an extensions management module. It has a module-level `loadedExtensions` variable and several procedures related to it. If we re-rewrite this with the OOP paradigm, we probably need to create two classes: `Extension` and `ExtensionManager`, but now, everything is simple with a single minimal module. Here we used a string vector to hold extension information, but we can use a struct vector if there are many fields in the extension record (not object).

Using the Procedural Programming Paradigm

We can indeed use the procedural programming approach with C since C is a procedural language. For example, check the file handling functions from the C standard library, you will notice that you need to pass the `FILE` pointer into each file handling function. Also, we can see the procedural programming pattern by inspecting the Linux kernel source code, as shown below:

```
2004
2005  int __cpuhp_state_add_instance(enum cpuhp_state state, struct hlist_node *node,
2006                                 bool invoke)
2007  {
2008          int ret;
2009
2010          cpus_read_lock();
2011          ret = __cpuhp_state_add_instance_cpuslocked(state, node, invoke);
2012          cpus_read_unlock();
2013          return ret;
```

Using a pure procedural paradigm is undoubtedly impossible with popular programming languages like C++, JavaScript, and Python. The reason is that those languages' standard library APIs offer OO classes for programmers. For example, see how the JavaScript standard library offers an OO interface for creating sets:

```
let numbers = new Set(); // constructor
numbers.add(10); // class method
console.log(numbers.size); // class property
```

On the other hand, C++ also offers a set of classes to interact with the C++ standard library. However, all modern stable programming languages are multi-paradigm languages, so those languages support functional programming with lambda functions, callbacks, recursion, and anonymous functions. Therefore, we can use this alternative procedural paradigm in any programming language by wrapping OOP-based standard library interfaces with functions. We can use the OOP-based standard library classes inside our procedures by not turning the entire project's paradigm into OOP.

Conclusion

OOP is a natural programming paradigm for solving almost all real-world problems, but it's artificial from the CPU's perspective. CPU doesn't treat a particular program's instruction set as objects and references, but it treats the entire program as a set of procedures and parameters. Therefore, from the technical perspective, the natural programming paradigm is the procedural paradigm for developing software systems.

However, from the business-oriented perspective, OOP maps the business logic and the codebase well — that's why most business-oriented software systems tend to use OOP instead of other paradigms. Also, the procedural programming approach may produce code smells in business-oriented software projects (i.e., e-commerce systems, payroll software, etc).

principles. As programmers, we always need to solve problems with simple, minimal, and efficient solutions rather than adding unwanted complexities. Therefore, try a procedural, functional, and modular-mixed approach for your next awesome project before drawing class diagrams.

This story is not to criticize OOP — it indeed showed you an alternative paradigm to improve your future software projects' simplicity by eliminating the over-engineering effect. The following story explains the advantages of maintaining simplicity in software development:

5 Programming Principles that Help You to Write Better Code

Impress both the compiler and your [illegible]ting better code with these programming prin[illegible]

levelup.gitconnected.com