Programming Paradigms

Terry Yin

November 1, 2014

Just like there are many different painting styles that artists can use to represent their arts, there are many different fundamental programming styles programmers can use to code their program. These fundamental styles in programming are called **programming paradigms**, (Colburn & Shute 2007, p.244). Just like painting styles are often related to the painting materials that artists use, programming paradigms are often related to programming languages.

One early programming paradigm is **procedural programming**. It has a couple of related paradigms, including **imperative programming** and **structured programming**. Examples of programming languages that support procedural programming paradigm are C, C++, Java and Python, Wikipedia (2014). Procedural programming instructs the computer what to do in a step by step style. These steps can be organized in a structured way, e.g. into functions (or procedures), and modules. Data and data types that are operated by the functions or modules are represented separately. But data in procedural programming could have different **scopes** within the program. Some data could have global scope, which means they could be accessed anywhere in the program. Some have local scope, which means they could only be accessed within the scope where it's defined, (Colburn & Shute 2007, p.260). Procedural programming paradigm introduced the modularity into programming and made large program easier to manage.

But having only **scope** is not enough to represent the cohesion between a certain type of data and the operation on them. Around 1974, Barbara Liskov and Stephen N. Zilles first introduce the idea of **Abstract Data Type**, which logically combine the data type and its operation together, (Liskov & Zilles 1974, ADT). ADT can be implemented with a procedural language like C. It could make the program structure more clear and easier to maintain.

The **ADT** model was then further extended into the **object-oriented programming** paradigm, with a lot of other elements people believed that were good ideas. And OO programming paradigm can be found in most of the new programming languages, like Java, C++, Python and Ruby. C++ is very often considered an OO language. The new elements in OOP other than abstract data type include inheritance, encapsulation, polymorphism, message passing, etc, etc. There are quite a lot of "good" things. And it's not easy to understand what does these hard words really mean, Meyerovich & Rabkin (2013) (unfortunately, the referenced paper uses Sourceforge as the main data source. It would be more convincing if it was Github).

It was generally believed that **inheritance** is a great way of software reuse. In year 1995, a great book on OO design was published, and it changed this believing of a lot of programmers. This great book is called *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma et al. (1994), by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, who were later referred as the Gang Of Four, (not to be confused with the Gang Of Four in the contemporary history of China). They said: "Favor 'object composition' over 'class inheritance'." But this didn't prevent many programmers to misuse inheritance all the time. OO language has dominate the software industry for about two decades, but there's hardly any evidence that it improved anything at all, Cardelli (1996).

One of the quite arguable **design patterns** in OO programming is the **Singleton Pattern**, (Gamma et al. 1994, p.138). A Singleton is an only instance of a type of data in the whole system. In a procedural programming paradigm, a singleton is nothing but a global variable. It brings the global state to the system, which increase the overall complexity of the system. As there are more and more need for parallel processing, programs that designed in procedural or OO programming paradigm often depends heavily on the singleton pattern to communicate between the processes or threads.

Functional programming is a programming paradigm that avoiding changing the global state or having any **side-effect**. It treats function as the "first class citizen", which means you can use a function as the parameter to the other function. Examples are Erlang, Haskell, JavaScript and Python. Because functional programming avoids changing global state, the need for mutual exclusion between processes is eliminated. So functional programming has a great opportunity in the era of parallel processing.

In the past, the hot topic of major programming paradigm evolved first from needing high level language, next to procedure programming, then to object-oriented programming, and last to functional programming. Each topic dominated for about 20 years, Tate (n.d.). Besides the major paradigm, there is another often used programming paradigm named the **declarative programming**. declarative programming style builds the program by describing the problem without specifying the detailed steps instructing the computer. Functional programming is also one type of declarative programming. There are other examples like SQL, regular expression and CSS.

What? You don't think CSS is a programming language? If it's not a programming language, why do I need to write unit test for it, Shore (2014)? "Quixote" is a new open source unit test framework for CSS set up by James Shore last month.

References

Cardelli, L. (1996), 'Bad engineering properties of object-orient languages', *ACM Computing Surveys (CSUR)* **28**(4es), 150.

Colburn, T. & Shute, G. (2007), 'Abstraction in computer science', *Minds and Machines* **17**(2), 169–184.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994), *Design patterns: elements of reusable object-oriented software*, Pearson Education.

Liskov, B. & Zilles, S. (1974), Programming with abstract data types, *in* 'ACM Sigplan Notices', Vol. 9, ACM, pp. 50–59.

Meyerovich, L. A. & Rabkin, A. S. (2013), Empirical analysis of programming language adoption, *in* 'Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications', ACM, pp. 1–18.

Shore, J. (2014), 'Quixote, a unit testing framework for css', https://github.com/jamesshore/quixote.

Tate, B. (n.d.), 'Fear. the drivers of language evolution', https://www.youtube.com/watch?v=l-zL_RLGBq4. Accessed: 2014-11-1.

Wikipedia (2014), 'Comparison of programming paradigms — wikipedia, the free encyclopedia'. [Online; accessed 1-November-2014].
URL: *http://en.wikipedia.org/w/index.php?title=Comparison_of_programming_paradigms&oldid=620477381*