WIKIPEDIA

Abstraction principle (computer programming)

In software engineering and programming language theory, the **abstraction principle** (or the **principle of abstraction**) is a basic dictum that aims to reduce duplication of information in a program (usually with emphasis on code duplication) whenever practical by making use of abstractions provided by the programming language or software libraries. The principle is sometimes stated as a recommendation to the programmer, but sometimes stated as a requirement of the programming language, assuming it is self-understood why abstractions are desirable to use. The origins of the principle are uncertain; it has been reinvented a number of times, sometimes under a different name, with slight variations.

When read as recommendation to the programmer, the abstraction principle can be generalized as the "don't repeat yourself" principle, which recommends avoiding the duplication of information in general, and also avoiding the duplication of human effort involved in the software development process.

Contents

The principle

As a recommendation to the programmer, in its formulation by Benjamin C. Pierce in *Types and Programming Languages* (2002), the abstraction principle reads (emphasis in original):[1]

> “ Each significant piece of functionality in a program should be implemented in just one place in the source code. Where similar functions are carried out by distinct pieces of code, it is generally beneficial to combine them into one by *abstracting out* the varying parts. ”

As a requirement of the programming language, in its formulation by David A. Schmidt in *The structure of typed programming languages* (1994), the abstraction principle reads:.[2]

> “ The phrases of any semantically meaningful syntactic class may be named. ”

History and variations

Under this very name, the abstraction principle appears into a long list of books. Here we give a necessarily incomplete list, together with the formulation if it is succinct:

- Alfred John Cole, Ronald Morrison (1982) *An introduction to programming with S-algol*: "[Abstraction] when applied to language design is to define all the semantically meaningful syntactic categories in the language and allow an abstraction over them".[3]
- Bruce J. MacLennan (1983) *Principles of programming languages: design, evaluation, and implementation*: "Avoid requiring something to be stated more than once; factor out the recurring pattern".[4]
- Jon Pearce (1998) *Programming and Meta-Programming in Scheme*: "Structure and function should be independent".[5]

The principle plays a central role in design patterns in object-oriented programming, although most writings on that topic do not give a name to the principle. The influential book by the Gang of Four, states: "The focus here is *encapsulating the concept that varies*, a theme of many design patterns." This statement has been rephrased by other authors as "Find what varies and encapsulate it."[6]

In this century, the principle has been reinvented in extreme programming under the slogan "Once and Only Once". The definition of this principle was rather succinct in its first appearance: "no duplicate code".[7] It has later been elaborated as applicable to other issues in software development: "Automate every process that's worth automating. If you find yourself performing a task many times, script it."[8]

Implications

The abstraction principle is often stated in the context of some mechanism intended to facilitate abstraction. The basic mechanism of control abstraction is a function or subroutine. Data abstractions include various forms of type polymorphism. More elaborate mechanisms that may combine data and control abstractions include: abstract data types, including classes, polytypism etc. The quest for richer abstractions that allow less duplication in complex scenarios is one of the driving forces in programming language research and design.

Inexperienced programmers may be tempted to introduce too much abstraction in their program—abstraction that won't be used more than once. A complementary principle that emphasize this issue is "You Ain't Gonna Need It" and, more generally, the KISS principle.

Since code is usually subject to revisions, following the abstraction principle may entail refactoring of code. The effort of rewriting a piece of code generically needs to be amortized against the estimated future benefits of an abstraction. A rule of thumb governing this was devised by Martin Fowler, and popularized as the rule of three. It states that if a piece of code is copied more than twice, i.e. it would end up having three or more copies, then it needs to be abstracted out.

Generalizations

"Don't repeat yourself", or the "DRY principle", is a generalization developed in the context of multi-tier architectures, where related code is by necessity duplicated to some extent across tiers, usually in different languages. In practical terms, the recommendation here is to rely on automated tools, like code generators and data transformations to avoid repetition.

Hardware programming interfaces

In addition to optimizing code, a hierarchical/recursive meaning of Abstraction level in programming also refers to the interfaces between hardware communication layers, also called "abstraction levels" and "abstraction layers." In this case, level of abstraction often is synonymous with interface. For example, in examining shellcode and the interface between higher and lower level languages, the level of abstraction changes from operating system commands (for example, in C) to register and circuit level calls and commands (for example, in assembly and binary). In the case of that example, the boundary or interface between the abstraction levels is the stack.[9]

References

1. Pierce, Benjamin (2002). *Types and Programming Languages*. MIT Press. p. 339. ISBN 0-262-16209-1.
2. David A. Schmidt, *The structure of typed programming languages*, MIT Press, 1994, ISBN 0-262-19349-3, p. 32
3. Alfred John Cole, Ronald Morrison, *An introduction to programming with S-algol*, CUP Archive, 1982, ISBN 0-521-25001-3, p. 150
4. Bruce J. MacLennan, *Principles of programming languages: design, evaluation, and implementation*, Holt, Rinehart, and Winston, 1983, p. 53
5. Jon Pearce, *Programming and meta-programming in scheme*, Birkhäuser, 1998, ISBN 0-387-98320-1, p. 40
6. Alan Shalloway, James Trott, *Design patterns explained: a new perspective on object-oriented design*, Addison-Wesley, 2002, ISBN 0-201-71594-5, p. 115
7. Kent Beck, *Extreme programming explained: embrace change*, 2nd edition, Addison-Wesley, 2000, ISBN 0-201-61641-6, p. 61
8. Chromatic, *Extreme programming pocket guide*, O'Reilly, 2003, ISBN 0-596-00485-0

9. Koziol, *The Shellcoders Handbook"*, Wiley, 2004, p. 10, ISBN 0-7645-4468-3