WIKIPEDIA

Separation of concerns

In computer science, **separation of concerns** (**SoC**) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program. A concern can be as general as the details of the hardware the code is being optimized for, or as specific as the name of a class to instantiate. A program that embodies SoC well is called a modular[1] program. Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well-defined interface. Encapsulation is a means of information hiding.[2] Layered designs in information systems are another embodiment of separation of concerns (e.g., presentation layer, business logic layer, data access layer, persistence layer).[3]

Separation of concerns results in higher degrees of freedom for some aspect of the program's design, deployment, or usage. Common among these is higher degrees of freedom for simplification and maintenance of code. When concerns are well-separated, there are higher degrees of freedom for module reuse as well as independent development and upgrade. Because modules hide the details of their concerns behind interfaces, increased freedom results to later improve or modify a single concern's section of code without having to know the details of other sections, and without having to make corresponding changes to those sections. Modules can also expose different versions of an interface, which increases the freedom to upgrade a complex system in piecemeal fashion without interim loss of functionality.

Separation of concerns is a form of abstraction. As with most abstractions, interfaces must be added and there is generally more net code to be executed. So despite the many benefits of well separated concerns, there is often an associated execution penalty.

Contents

Implementation

The mechanisms for modular or object-oriented programming that are provided by a programming language are mechanisms that allow developers to provide SoC.[4] For example, object-oriented programming languages such as C#, C++, Delphi, and Java can separate concerns into objects, and architectural design patterns like MVC or MVP can separate content from presentation and the data-processing (model) from content. Service-oriented design can separate concerns into services. Procedural programming languages such as C and Pascal can separate concerns into procedures or functions. Aspect-oriented programming languages can separate concerns into aspects and objects.

Separation of concerns is an important design principle in many other areas as well, such as urban planning, architecture and information design.[5] The goal is to more effectively understand, design, and manage complex interdependent systems, so that functions can be reused, optimized independently of other functions, and insulated from the potential failure of other functions.

Common examples include separating a space into rooms, so that activity in one room does not affect people in other rooms, and keeping the stove on one circuit and the lights on another, so that overload by the stove does not turn the lights off. The example with rooms shows encapsulation, where information inside one room, such as how messy it is, is not available to the other rooms, except through the interface, which is the door. The example with circuits demonstrates that activity inside one module, which is a circuit with consumers of electricity attached, does not affect activity in a different module, so each module is not concerned with what happens in the other.

Origin

The term *separation of concerns* was probably coined by Edsger W. Dijkstra in his 1974 paper "On the role of scientific thought".[6]

> Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects. We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. But nothing is gained —on the contrary!— by tackling these various aspects simultaneously. It is what I sometimes have called **"the separation of concerns"**, which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focusing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously.

Fifteen years later, it was evident the term **Separation of Concerns** was becoming an accepted idea. In 1989, Chris Reade wrote a book titled "Elements of Functional Programming"[7] that describes separation of concerns:

> The programmer is having to do several things at the same time, namely,
>
> 1. describe what is to be computed;
> 2. organise the computation sequencing into small steps;
> 3. organise memory management during the computation.

Reade continues to say,

> Ideally, the programmer should be able to concentrate on the first of the three tasks (describing what is to be computed) without being distracted by the other two, more administrative, tasks. Clearly, administration is important, but by separating it from the main task we are likely to get more reliable results and we can ease the programming problem by automating much of the administration.
>
> The **separation of concerns** has other advantages as well. For example, program proving becomes much more feasible when details of sequencing and memory management are absent from the program. Furthermore, descriptions of what is to be computed should be free of such detailed step-by-step descriptions of how to do it, if they are to be evaluated with different machine architectures. Sequences of small changes to a data object held in a store may be an inappropriate description of how to compute something when a highly parallel machine is being used with thousands of processors distributed throughout the machine and local rather than global storage facilities.
>
> Automating the administrative aspects means that the language implementor has to deal with them, but he/she has far more opportunity to make use of very different computation mechanisms with different machine architectures.

Examples

Internet protocol stack

Separation of concerns is crucial to the design of the Internet. In the Internet Protocol Suite, great efforts have been made to separate concerns into well-defined layers. This allows protocol designers to focus on the concerns in one layer, and ignore the other layers. The Application Layer protocol SMTP, for example, is concerned about all the details of conducting an email session over a reliable transport service (usually TCP), but not in the least concerned about how the transport service makes that service reliable. Similarly, TCP is not concerned about the routing of data packets, which is handled at the Internet Layer.

HTML, CSS, JavaScript

HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript (JS) are complementary languages used in the development of webpages and websites. HTML is mainly used for organization of webpage content, CSS is used for definition of content presentation style, and JS defines how the content interacts and behaves with the user. Historically, this was not the case: prior to the introduction of CSS, HTML performed both duties of defining semantics and style.

Subject-oriented programming

Subject-oriented programming allows separate concerns to be addressed as separate software constructs, each on an equal footing with the others. Each concern provides its own class-structure into which the objects in common are organized, and contributes state and methods to the composite result where they cut across one another. Correspondence rules describe how the classes and methods in the various concerns are related to each other at points where they interact, allowing composite behavior for a method to be derived from several concerns. Multi-dimensional Separation of Concerns allows the analysis and composition of concerns to be manipulated as a multi-dimensional "matrix" in which each concern provides a dimension in which different points of choice are enumerated, with the cells of the matrix occupied by the appropriate software artifacts.

Aspect-oriented programming

Aspect-oriented programming allows cross-cutting concerns to be addressed as primary concerns. For example, most programs require some form of security and logging. Security and logging are often secondary concerns, whereas the primary concern is often on accomplishing business goals. However, when designing a program, its security must be built into the design from the beginning instead of being treated as a secondary concern. Applying security afterwards often results in an insufficient security model that leaves too many gaps for future attacks. This may be solved with aspect-oriented programming. For example, an aspect may be written to enforce that calls to a certain API are always logged, or that errors are always logged when an exception is thrown, regardless of whether the program's procedural code handles the exception or propagates it.[8]

Software build automation

Most project organization tasks are seen as secondary tasks. For example, build automation is an approach to automating the process of compiling source code into binary code. The primary goals in build automation are reducing the risk of human error and saving time.

Levels of analysis in artificial intelligence

In cognitive science and artificial intelligence, it is common to refer to David Marr's levels of analysis. At any given time, a researcher may be focusing on (1) what some aspect of intelligence needs to compute, (2) what algorithm it employs, or (3) how that algorithm is implemented in hardware. This separation of concerns is similar to the interface/implementation distinction in software and hardware engineering.

Normalized Systems

In Normalized Systems separation of concerns is one of the four guiding principles. Adhering to this principle is one of the tools that helps reduce the combinatorial effects that, over time, get introduced in software that is being maintained. In Normalized Systems separation of concerns is actively supported by the tools.

SoC via partial classes

Separation of concerns can be implemented and enforced via partial classes.[9]

SoC via partial classes in Ruby

bear_hunting.rb

```
class Bear
  def self.hunt
    # TODO: return some food
  end
end
```

bear_eating.rb

```
 class Bear
   def self.eat( food )
     raise "#{food} is not edible!" unless food.respond_to? :nutrition_value
     food.nutrition_value
   end
 end
```

bear_hunger.rb

```
class Bear
  attr_accessor :hunger
  def monitor_hunger
    if @hunger > 50
      @hunger -= self.eat( self.hunt )
    end
  end
end
```

See also

- Abstraction principle (programming)
- Aspect-oriented software development
- Concern (computer science)
- Core concern
- Coupling (computer science)
- Cross-cutting concern
- Holism
- Modular design
- Modular programming
- Orthogonality § Computer science
- Separation of presentation and content
- Single responsibility principle

References

1. Laplante, Phillip (2007). *What Every Engineer Should Know About Software Engineering* (https://books.google.com/books?id=pFHYk0KWAEgC&pg=PA85&dq=%22separation+of+concerns%22&hl=en&sa=X&ei=WQ_aUNn5DYjNiwLS54GADQ&ved=0CEwQ6AEwAw#v=onepage&q=%22separation%20of%20concerns%22&f=false). CRC Press. ISBN 0849372283.
2. Mitchell, Dr. R. J. (1990). *Managing Complexity in Software Engineering* (https://books.google.com/books?id=uXtHeZt8ZowC&pg=PA5&dq=%22separation+of+concerns%22&hl=en&sa=X&ei=WQ_aUNn5DYjNiwLS54GADQ&ved=0CFEQ6AEwBA#v=onepage&q=%22separation%20of%20concerns%22&f=false). IEE. p. 5. ISBN 0863411711.
3. *Microsoft Application Architecture Guide* (https://books.google.com/books?id=D9on897Ep7AC&pg=PT54&dq=%22separation+of+concerns%22+%22layered+design%22&hl=en&sa=X&ei=0xPaUKKJMITmiwKhiYDYCA&ved=0CDgQ6AEwAQ). Microsoft Press. 2009. ISBN 0-7356-2710-X.

4. Painter, Robert Richard. "Software Plans: Multi-Dimensional Fine-Grained Separation of Concerns". Penn State. CiteSeerX 10.1.1.110.9227 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.110.9227).
5. Garofalo, Raffaele (2011). *Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModel Pattern* (https://books.google.com/books?id=6WJ5oQT0dcUC&pg=PA18&dq=%22separation+of+concerns%22+%22urban+planning%22&hl=en&sa=X&ei=CxjaUK-hN-yVjAKz84D4Cg&ved=0CDQQ6AEwAA#v=onepage&q=%22separation%20of%20concerns%22%20%22urban%20planning%22&f=false). Microsoft Press. p. 18. ISBN 0735650926.
6. Dijkstra, Edsger W (1982). "On the role of scientific thought" (http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html). *Selected writings on Computing: A Personal Perspective*. New York, NY, USA: Springer-Verlag. pp. 60–66. ISBN 0-387-90652-5.
7. Reade, Chris (1989). *Elements of Functional Programming*. Boston, MA, USA: Addison-Wesley Longman. ISBN 0-201-12915-9.
8. Jess Nielsen (June 2006). "Building Secure Applications" (http://jess.heidrun.dk/cv/BuildingSecureApp.pdf) (PDF). Retrieved 2012-02-08.
9. Tiago Dias (October 2006). "Hyper/Net: MDSoC Support for .NET" (http://ptsoft.net/tdd/papers/TDIAS06_HyperNet__MDSOC_support_for_NET.pdf) (PDF). *DSOA 2006*. Retrieved 2007-09-25.

External links

- Multi-Dimensional Separation of Concerns (http://domino.watson.ibm.com/library/cyberdig.nsf/1e4115aea78b6e7c85256b360066f0d4/5af350e4286e003985256766004ebe71?OpenDocument)
- TAOSAD (http://trese.cs.utwente.nl/taosad/separation_of_concerns.htm)
- Tutorial and Workshop on Aspect-Oriented Programming and Separation of Concerns (http://www.comp.lancs.ac.uk/computing/users/marash/aopws2001/)