The evolution of programming techniques

Abstract

This article points out the benefits of structured programming (SP), object-oriented programming (OOP), and Design by Contract (DbC).

Structured programming

Procedural programming is based on the concepts of discrete, reusable code entities and data scope. A procedural program comprises multiple such entities, which may be written by the programmer or taken from a code library. Each entity is composed of one or more procedures - pieces of software that perform particular tasks. Procedures can be termed methods, functions, routines, subroutines, or procedures.

A procedural program can have multiple scopes - levels at which particular variables can be seen. For example, a variable can have global scope - it can be seen by all procedures. Another variable's scope may encompass only a particular procedure, or it may encompass the procedure and all procedures defined within it.

Procedural programming languages enable you to practice structured programming (SP) - an approach to the detailed coding of software entities that is based on the concept of one-in, one-out control constructs. The execution of such a construct can begin at just one point, and end at just one point.

A properly designed structured program is human-readable from start to finish - it is simple and direct in the way it executes, and its execution does not involve jumps from one point to another within the program.

Structured programming involves three basic types of construct that - used singly or in combination - you can use to program any conceivable control flow:

- sequences
- selections
- iterations

A sequence is a number of consecutive lines of code that execute in the order in which they are written. For example, two method invocations positioned one after the other in a program constitute a sequence.

A selection is a control structure containing a condition that determines which statements belonging to the structure will be executed. There are two main types of selection - the if-then-else and switch statements.

An iteration is a control structure that causes one or more statements to be executed repeatedly a number of times until a condition ends the cycle and the code's execution moves on. Typical examples of iterations include for loops and while loops.

Structured programming constitutes a set of restrictions on what you can do as a programmer. These restrictions have advantages and disadvantages:

Advantages

- the code structure matches the logic of the solution
- factoring and modularity are built into permitted constructs
it encourages clarity and simplicity
its divide-and-conquer approach to problem-solving becomes ingrained in the minds of programmers

Disadvantages

- it keeps data separate from the process
- it emphasizes procedures over data
- it doesn't have clearly defined interfaces and contracts
- it doesn't facilitate reuse of routines
- it becomes progressively harder to maintain as it grows in size and complexity

Object-oriented programming (OOP)

Object-oriented programming (OOP) is a paradigm in which a program is composed of discrete units called objects, each of which functions to some extent like a program in its own right. Each object is a unit containing data and functionality in a way that comprises a logical entity that can be used and reused. Each object is designed as a model of a particular abstract functional entity - an actor - that can perform tasks, report its state, change its state, and exchange information with other objects.

OOP is not a departure from SP techniques, but an evolution. It differs from structured programming in a number of important ways, including an emphasis on data over action and a separation of data from procedures. However, it retains many of the features of structured programming, such as:

- functions and routines (whose OOP equivalents are called methods)
- structured flow control
- data typing (including the concept of classes as data types)
- naming conventions
- packaging into modules
- strict compile-time checking

OOP formalizes a number of concepts, including:

- modularity
- abstraction
- encapsulation
- inheritance
- polymorphism

Modularity is made possible in OOP languages by objects. This is a more advanced version of the modularity characteristic of SP. Methods in OOP are equivalent to SP's functions and routines.

Abstraction is the ability of a program to deal with objects and methods as discrete entities without being concerned with what actually occurs within them. In effect, abstraction enables the program to focus on the essential elements of the information it is working with.

Encapsulation, which is also termed data hiding, permits only methods internal to an object to change the object's state. Other objects can interact with the object only through the interface exposed by the object for that purpose. The interface restricts the types of interactions that can occur. Conceptually, this is an advance on SP's focus on one-in, one-out control constructs.

Inheritance enables you to reuse code, so you don't have to re-implement functionality that already exists. In OOP, you can define a new object that inherits functionality from an existing object and also has its own behavior.
Polymorphism is an OOP technique that enables you to implement multiple versions of a particular operation in a way that enables the operation to accept a number of different inputs. The operation's behavior may be specific to the type of input passed to it.

The formalization of these features enables you, as a programmer, to focus more on building solutions. The compiler and the environment handle much of the activity required for memory management, data safety, passing data between methods, and syntax checking.

Benefits of OOP

OOP, when it is implemented properly, is a more flexible and powerful way of programming than just SP on its own. Because it builds on all of the features of SP, it is an advance on SP, rather than just an alternative. In addition, OOP-based programs tend to be easier to develop and maintain. OOP's modularity enables program designers to mirror aspects of the problem that the program is meant to solve.

However, OOP can be far more complex than simple SP, or any older programming paradigm, and because of this it is more difficult to attain proficiency in OOP.

Design by Contract (DbC)

Design by Contract (DbC) is based on the design of contracts - sets of conditions that modular code components must fulfill. DbC is an enhancement of OOP (trademarked by Bertrand Meyer and implemented in his Eiffel language as assertions) in which a software system is viewed as a set of components whose interactions are determined by contracts. DbC constitutes an effective framework for debugging, testing, and - more generally - quality assurance. It enhances OOP by requiring contracts for all classes and methods before these are actually coded.

A contract is a set of preconditions, postconditions, and - less commonly - performance guarantees that specify or assert the changes in state caused by the component.

Preconditions are criteria that the caller must fulfill in order to be able to access the relevant component. These include

- acceptable and unacceptable input values
- invariants
- all possible error and exception conditions
- return values and their meanings
- changes of state associated with the component other than its return value (side-effects)

Postconditions are criteria that the component guarantees to meet when it completes its tasks, such as the return of particular parameters.

The benefits of DbC include

- promoting a better understanding of OOP principles and, more generally, of software construction
- promoting a better understanding and control of the inheritance mechanism

In addition, DbC provides

- a systematic approach to building bug-free object-oriented systems
- a method of documenting software components
- safe and effective language constructs for exception handling

Summary
Structured programming is an approach to the detailed coding of software entities that is based on the concept of one-in, one-out control constructs. It constitutes a set of restrictions on what you can do as a programmer. These restrictions have drawbacks as well as advantages, but generally enhance your effectiveness as a programmer.

Object-oriented programming (OOP) builds on the benefits of structured programming, sharing a number of its characteristics. It is based on the principles of abstraction, modularity, code reuse, polymorphism, and encapsulation. Modularity is enforced through objects. In OOP, methods are the equivalent of functions and procedures.

Design by Contract (DbC) is an approach to high-level software design that enhances OOP. It is based on formulating contracts - sets of conditions that modular code components must fulfill.