What does AOP address?

Answer:

Aspect-oriented programming is a new programming technique that make possible to clearly express programs involving some design decisions have been so difficult to cleanly capture in actual code aspects, including appropriate isolation, composition and reuse of the aspect codes.

AOP based implementation is easy to understand and efficient both. We can say that it addresses 1) Basic Functionality and 2) Optimizing Memory Usage and Cross-Cutting

(Cross-Cutting :- There is certain generic functionality that is required in numerous locations in any application. However, this feature has nothing to do with the application's business logic. Assume you do a role-based security check before each business method in your app. Security is a cross-cutting issue here. It is required for any program, but it is not required from a business standpoint; it is a simple general capability that we must implement in multiple places throughout the application.)

How is it done?

Answer:

A property that can be neatly captured in a generalized technique is referred to as a component. Aspect is a crosscutting issue that can't be neatly captured in a standard process. Aspects are attributes that affect a component's performance or semantics. Abstraction and composition are the two key mechanisms that allow AOP to function. In terms of connecting points, abstraction is just encapsulating crosscutting issues using an aspect description language. Composition is the process of utilizing an aspect weaver to combine components that have been run or compiled using a traditional language with aspects.

1) Basic Functionality

→ Achieving the first goal- “easy to understand” is relatively easy. Good old-fashioned procedural programming can be used to implement the system clearly, concisely, and in good alignment with the domain model. A set of primitive procedures would implement the basic filters, and higher level filters would be defined in terms of the primitive ones.

2) Optimizing Memory Usage and Cross-Cutting.

→ To achieve optimizing memory usage take a more global perspective of the program, map out what intermediate results end up being inputs to what other filters, and then code up a version of the program that fuses loops appropriately to implement the original functionality while creating as few intermediates possible.

In short, revising the code to make more efficient use of memory has destroyed the original clean component structure.

→ Whenever two properties being programmed must compose differently and yet be coordinated, we say that they cross-cut each other.

List the elements of AOP with a brief definition/description of each.

Answer:

The AOP-based implementation of an application consists of: --

(i.a) a component language with which to program the components: -

With only slight differences, the component language is similar to the procedural language used previously. For starters, filters are no longer treated as operations. Second, primitive loops are designed in such a way that their loop structure is as obvious as feasible.

(i.b) one or more aspect languages with which to program the aspects: -

The aspect language is a simple procedural language that allows you to perform simple operations on dataflow graph nodes. The aspect software can then simply look for loops that need to be fused and perform the necessary fusion. The fusion situation is handled by the following code snippet, which is part of the core of that aspect program.

(ii) an aspect weaver for the combined languages: -

The weaver is not a "smart" compiler, which might be difficult to design and build. This is an important characteristic of this system. Using AOP, have prepared for the programmer to supply all the major implementation strategy decisions—all the actual smarts—using the relevant aspect languages. Rather than being inspired, the weaver's task is to integrate.

(iii.a) A component program, that implements the components using the component language: -

The pixelwise construct is an iterator that, in this case, moves in lockstep through pictures a and b, binding aa and bb to the pixel values and producing a result image. The different scenarios of aggregation, distribution, shifting, and combining of pixel values that are needed in this system are provided by four identical constructions. The

introduction of these high-level looping structures is a significant advance that makes it much easier for aspect languages to discover, analyze, and fuse loops.

(iii.b) one or more aspect programs that implement the aspects using the aspect languages: -

Indicating a dozen identical sentences about when and how to fuse are required to describe the composition rules and fusion structure for the five types of loops in the real system. This is one of the reasons why this system could not be handled by relying on an optimizing compiler to perform the necessary fusion—the amount of program analysis and understanding required is so large that compilers cannot be relied upon to do it reliably.