WIKIPEDIA

Aspect-oriented programming

In computing, **aspect-oriented programming** (**AOP**) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does so by adding additional behavior to existing code (an advice) *without* modifying the code itself, instead separately specifying which code is modified via a "pointcut" specification, such as "log all function calls when the function's name begins with 'set'". This allows behaviors that are not central to the business logic (such as logging) to be added to a program without cluttering the code, core to the functionality. AOP forms a basis for aspect-oriented software development.

AOP includes programming methods and tools that support the modularization of concerns at the level of the source code, while "aspect-oriented software development" refers to a whole engineering discipline.

Aspect-oriented programming entails breaking down program logic into distinct parts (so-called *concerns*, cohesive areas of functionality). Nearly all programming paradigms support some level of grouping and encapsulation of concerns into separate, independent entities by providing abstractions (e.g., functions, procedures, modules, classes, methods) that can be used for implementing, abstracting and composing these concerns. Some concerns "cut across" multiple abstractions in a program, and defy these forms of implementation. These concerns are called *cross-cutting concerns* or horizontal concerns.

Logging exemplifies a crosscutting concern because a logging strategy necessarily affects every logged part of the system. Logging thereby *crosscuts* all logged classes and methods.

All AOP implementations have some crosscutting expressions that encapsulate each concern in one place. The difference between implementations lies in the power, safety, and usability of the constructs provided. For example, interceptors that specify the methods to express a limited form of crosscutting, without much support for type-safety or debugging. AspectJ has a number of such expressions and encapsulates them in a special class, an aspect. For example, an aspect can alter the behavior of the base code (the non-aspect part of a program) by applying advice (additional behavior) at various join points (points in a program) specified in a quantification or query called a pointcut (that detects whether a given join point matches). An aspect can also make binary-compatible structural changes to other classes, like adding members or parents.

Contents

History

AOP has several direct antecedents A1 and A2:[1] reflection and metaobject protocols, subject-oriented programming, Composition Filters and Adaptive Programming.[2]

Gregor Kiczales and colleagues at Xerox PARC developed the explicit concept of AOP, and followed this with the AspectJ AOP extension to Java. IBM's research team pursued a tool approach over a language design approach and in 2001 proposed Hyper/J and the Concern Manipulation Environment, which have not seen wide usage.

The examples in this article use AspectJ.

The Microsoft Transaction Server is considered to be the first major application of AOP followed by Enterprise JavaBeans.[3][4]

Motivation and basic concepts

Typically, an aspect is *scattered* or *tangled* as code, making it harder to understand and maintain. It is scattered by virtue of the function (such as logging) being spread over a number of unrelated functions that might use *its* function, possibly in entirely unrelated systems, different source languages, etc. That means to change logging can require modifying all affected modules. Aspects become tangled not only with the mainline function of the systems in which they are expressed but also with each other. That means changing one concern entails understanding all the tangled concerns or having some means by which the effect of changes can be inferred.

For example, consider a banking application with a conceptually very simple method for transferring an amount from one account to another:[5]

```
void transfer(Account fromAcc, Account toAcc, int amount) throws Exception {
  if (fromAcc.getBalance() < amount)
      throw new InsufficientFundsException();

  fromAcc.withdraw(amount);
  toAcc.deposit(amount);
}
```

However, this transfer method overlooks certain considerations that a deployed application would require: it lacks security checks to verify that the current user has the authorization to perform this operation; a database transaction should encapsulate the operation in order to prevent accidental data loss; for diagnostics, the operation should be logged to the system log, etc.

A version with all those new concerns, for the sake of example, could look somewhat like this:

```
void transfer(Account fromAcc, Account toAcc, int amount, User user,
    Logger logger, Database database) throws Exception {
  logger.info("Transferring money...");

  if (!isUserAuthorised(user, fromAcc)) {
    logger.info("User has no permission.");
    throw new UnauthorisedUserException();
  }

  if (fromAcc.getBalance() < amount) {
    logger.info("Insufficient funds.");
    throw new InsufficientFundsException();
  }

  fromAcc.withdraw(amount);
  toAcc.deposit(amount);

  database.commitChanges();  // Atomic operation.

  logger.info("Transaction successful.");
}
```

In this example other interests have become *tangled* with the basic functionality (sometimes called the *business logic concern*). Transactions, security, and logging all exemplify *cross-cutting concerns*.

Now consider what happens if we suddenly need to change (for example) the security considerations for the application. In the program's current version, security-related operations appear *scattered* across numerous methods, and such a change would require a major effort.

AOP attempts to solve this problem by allowing the programmer to express cross-cutting concerns in stand-alone modules called *aspects*. Aspects can contain *advice* (code joined to specified points in the program) and *inter-type declarations* (structural members added to other classes). For example, a security module can include advice that performs a security check before accessing a bank account. The pointcut defines the times (join points) when one can access a bank account, and the code in the advice body defines how the security check is implemented. That way, both the check and the places can be maintained in one place. Further, a good pointcut can anticipate later program changes, so if another developer creates a new method to access the bank account, the advice will apply to the new method when it executes.

So for the example above implementing logging in an aspect:

```
aspect Logger {
  void Bank.transfer(Account fromAcc, Account toAcc, int amount, User user, Logger logger)  {
    logger.info("Transferring money...");
  }

  void Bank.getMoneyBack(User user, int transactionId, Logger logger)  {
    logger.info("User requested money back.");
  }

  // Other crosscutting code.
}
```

One can think of AOP as a debugging tool or as a user-level tool. Advice should be reserved for the cases where you cannot get the function changed (user level)[6] or do not want to change the function in production code (debugging).

Join point models

The advice-related component of an aspect-oriented language defines a join point model (JPM). A JPM defines three things:

1. When the advice can run. These are called *join points* because they are points in a running program where additional behavior can be usefully joined. A join point needs to be addressable and understandable by an ordinary programmer to be useful. It should also be stable across inconsequential program changes in order for an aspect to be stable across such changes. Many AOP implementations support method executions and field references as join points.
2. A way to specify (or *quantify*) join points, called *pointcuts*. Pointcuts determine whether a given join point matches. Most useful pointcut languages use a syntax like the base language (for example, AspectJ uses Java signatures) and allow reuse through naming and combination.
3. A means of specifying code to run at a join point. AspectJ calls this *advice*, and can run it before, after, and around join points. Some implementations also support things like defining a method in an aspect on another class.

Join-point models can be compared based on the join points exposed, how join points are specified, the operations permitted at the join points, and the structural enhancements that can be expressed.

AspectJ's join-point model

- The join points in AspectJ include method or constructor call or execution, the initialization of a class or object, field read and write access, exception handlers, etc. They do not include loops, super calls, throws clauses, multiple statements, etc.
- Pointcuts are specified by combinations of *primitive pointcut designators* (PCDs).

"Kinded" PCDs match a particular kind of join point (e.g., method execution) and tend to take as input a Java-like signature. One such pointcut looks like this:

```
execution(* set*(*))
```

This pointcut matches a method-execution join point, if the method name starts with "`set`" and there is exactly one argument of any type.

"Dynamic" PCDs check runtime types and bind variables. For example,

```
 this(Point)
```

This pointcut matches when the currently executing object is an instance of class `Point`. Note that the unqualified name of a class can be used via Java's normal type lookup.

"Scope" PCDs limit the lexical scope of the join point. For example:

```
within(com.company.*)
```

This pointcut matches any join point in any type in the `com.company` package. The `*` is one form of the wildcards that can be used to match many things with one signature.

Pointcuts can be composed and named for reuse. For example:

```
pointcut set() : execution(* set*(*) ) && this(Point) && within(com.company.*);
```

This pointcut matches a method-execution join point, if the method name starts with "`set`" and `this` is an instance of type `Point` in the `com.company` package. It can be referred to using the name "`set()`".

- Advice specifies to run at (before, after, or around) a join point (specified with a pointcut) certain code (specified like code in a method). The AOP runtime invokes Advice automatically when the pointcut matches the join point. For example:

```
after() : set() {
   Display.update();
}
```

This effectively specifies: "if the *`set()`* pointcut matches the join point, run the code `Display.update()` after the join point completes."

Other potential join point models

There are other kinds of JPMs. All advice languages can be defined in terms of their JPM. For example, a hypothetical aspect language for UML may have the following JPM:

- Join points are all model elements.
- Pointcuts are some boolean expression combining the model elements.
- The means of affect at these points are a visualization of all the matched join points.

Inter-type declarations

Inter-type declarations provide a way to express crosscutting concerns affecting the structure of modules. Also known as *open classes* and *extension methods*, this enables programmers to declare in one place members or parents of another class, typically in order to combine all the code related to a concern in one aspect. For example, if a programmer implemented the crosscutting display-update concern using visitors instead, an inter-type declaration using the visitor pattern might look like this in AspectJ:

```
aspect DisplayUpdate {
  void Point.acceptVisitor(Visitor v) {
    v.visit(this);
  }
  // other crosscutting code...
}
```

This code snippet adds the `acceptVisitor` method to the `Point` class.

It is a requirement that any structural additions be compatible with the original class, so that clients of the existing class continue to operate, unless the AOP implementation can expect to control all clients at all times.

Implementation

AOP programs can affect other programs in two different ways, depending on the underlying languages and environments:

1. a combined program is produced, valid in the original language and indistinguishable from an ordinary program to the ultimate interpreter
2. the ultimate interpreter or environment is updated to understand and implement AOP features.

The difficulty of changing environments means most implementations produce compatible combination programs through a process known as *weaving* - a special case of program transformation. An aspect weaver reads the aspect-oriented code and generates appropriate object-oriented code with the aspects integrated. The same AOP language can be implemented through a variety of weaving methods, so the semantics of a language should never be understood in terms of the weaving implementation. Only the speed of an implementation and its ease of deployment are affected by which method of combination is used.

Systems can implement source-level weaving using preprocessors (as C++ was implemented originally in CFront) that require access to program source files. However, Java's well-defined binary form enables bytecode weavers to work with any Java program in .class-file form. Bytecode weavers can be deployed during the build process or, if the weave model is per-class, during class loading. AspectJ started with source-level weaving in 2001, delivered a per-class bytecode weaver in 2002, and offered advanced load-time support after the integration of AspectWerkz in 2005.

Any solution that combines programs at runtime has to provide views that segregate them properly to maintain the programmer's segregated model. Java's bytecode support for multiple source files enables any debugger to step through a properly woven .class file in a source editor. However, some third-party decompilers cannot process woven code because they expect code produced by Javac rather than all supported bytecode forms (see also "Criticism", below).

Deploy-time weaving offers another approach.[7] This basically implies post-processing, but rather than patching the generated code, this weaving approach *subclasses* existing classes so that the modifications are introduced by method-overriding. The existing classes remain untouched, even at runtime, and all existing tools (debuggers, profilers, etc.) can be used during development. A similar approach has already proven itself in the implementation of many Java EE application servers, such as IBM's WebSphere.

Terminology

Standard terminology used in Aspect-oriented programming may include:

Cross-cutting concerns
Even though most classes in an OO model will perform a single, specific function, they often share common, secondary requirements with other classes. For example, we may want to add logging to classes within the data-access layer and also to classes in the UI layer whenever a thread enters or exits a method. Further concerns can be related to security such as access control [8] or information flow control.[9] Even though each class has a very different primary functionality, the code needed to perform the secondary functionality is often identical.

Advice
This is the additional code that you want to apply to your existing model. In our example, this is the logging code that we want to apply whenever the thread enters or exits a method.

Pointcut
This is the term given to the point of execution in the application at which cross-cutting concern needs to be applied. In our example, a pointcut is reached when the thread enters a method, and another pointcut is reached when the thread exits the method.

Aspect
The combination of the pointcut and the advice is termed an aspect. In the example above, we add a logging aspect to our application by defining a pointcut and giving the correct advice.

Comparison to other programming paradigms

Aspects emerged from object-oriented programming and computational reflection. AOP languages have functionality similar to, but more restricted than metaobject protocols. Aspects relate closely to programming concepts like subjects, mixins, and delegation. Other ways to use aspect-oriented programming paradigms include Composition Filters and the hyperslices approach. Since at least the 1970s, developers have been using forms of interception and dispatch-patching that resemble some of the implementation methods for AOP, but these never had the semantics that the crosscutting specifications provide written in one place.

Designers have considered alternative ways to achieve separation of code, such as C#'s partial types, but such approaches lack a quantification mechanism that allows reaching several join points of the code with one declarative statement.

Though it may seem unrelated, in testing, the use of mocks or stubs requires the use of AOP techniques, like around advice, and so forth. Here the collaborating objects are for the purpose of the test, a cross cutting concern. Thus the various Mock Object frameworks provide these features. For example, a process invokes a service to get a balance amount. In the test of the process, where the amount comes from is unimportant, only that the process uses the balance according to the requirements.

Adoption issues

Programmers need to be able to read code and understand what is happening in order to prevent errors.[10] Even with proper education, understanding crosscutting concerns can be difficult without proper support for visualizing both static structure and the dynamic flow of a program.[11] Beginning in 2002, AspectJ began to provide IDE plug-ins to support the visualizing of crosscutting concerns. Those features, as well as aspect code assist and refactoring are now common.

Given the power of AOP, if a programmer makes a logical mistake in expressing crosscutting, it can lead to widespread program failure. Conversely, another programmer may change the join points in a program – e.g., by renaming or moving methods – in ways that the aspect writer did not anticipate, with unforeseen consequences. One advantage of modularizing crosscutting concerns is enabling one programmer to affect the entire system easily; as a result, such problems present as a conflict over responsibility between two or more developers for a given failure. However, the solution for these problems can be much easier in the presence of AOP, since only the aspect needs to be changed, whereas the corresponding problems without AOP can be much more spread out.

Criticism

The most basic criticism of the effect of AOP is that control flow is obscured, and that it is not only worse than the much-maligned GOTO, but is in fact closely analogous to the joke COME FROM statement. The *obliviousness of application*, which is fundamental to many definitions of AOP (the code in question has no indication that an advice will be applied, which is specified instead in the pointcut), means that the advice is not visible, in contrast to an explicit method call.[12][13] For example, compare the COME FROM program:[12]

```
 5 INPUT X
10 PRINT 'Result is :'
15 PRINT X
```

```
20 COME FROM 10
25      X = X * X
30 RETURN
```

with an AOP fragment with analogous semantics:

```
main() {
    input x
    print(result(x))
}
input result(int x) { return x }
around(int x): call(result(int)) && args(x) {
    int temp = proceed(x)
    return temp * temp
}
```

Indeed, the pointcut may depend on runtime condition and thus not be statically deterministic. This can be mitigated but not solved by static analysis and IDE support showing which advices *potentially* match.

General criticisms are that AOP purports to improve "both modularity and the structure of code", but some counter that it instead undermines these goals and impedes "independent development and understandability of programs".[14] Specifically, quantification by pointcuts breaks modularity: "one must, in general, have whole-program knowledge to reason about the dynamic execution of an aspect-oriented program."[15] Further, while its goals (modularizing cross-cutting concerns) are well understood, its actual definition is unclear and not clearly distinguished from other well-established techniques.[14] Cross-cutting concerns potentially cross-cut each other, requiring some resolution mechanism, such as ordering.[14] Indeed, aspects can apply to themselves, leading to problems such as the liar paradox.[16]

Technical criticisms include that the quantification of pointcuts (defining where advices are executed) is "extremely sensitive to changes in the program", which is known as the *fragile pointcut problem*.[14] The problems with pointcuts are deemed intractable: if one replaces the quantification of pointcuts with explicit annotations, one obtains attribute-oriented programming instead, which is simply an explicit subroutine call and suffers the identical problem of scattering that AOP was designed to solve.[14]

Implementations

The following programming languages have implemented AOP, within the language, or as an external library:

- .NET Framework languages (C# / VB.NET)[17]
  - PostSharp (https://www.postsharp.net/) is a commercial AOP implementation with a free but limited edition.
  - Unity, It provides an API to facilitate proven practices in core areas of programming including data access, security, logging, exception handling and others.
- ActionScript[18]
- Ada[19]

- AutoHotkey[20]
- C / C++[21]
- COBOL[22]
- The Cocoa Objective-C frameworks[23]
- ColdFusion[24]
- Common Lisp[25]
- Delphi[26][27][28]
- Delphi Prism[29]
- e (IEEE 1647)
- Emacs Lisp[30]
- Groovy
- Haskell[31]
- Java[32]
  - AspectJ
- JavaScript[33]
- Logtalk[34]
- Lua[35]
- make[36]
- Matlab[37]
- ML[38]
- Perl[39]
- PHP[40]
- Prolog[41]
- Python[42]
- Racket[43]
- Ruby[44][45][46]
- Squeak Smalltalk[47][48]
- UML 2.0[49]
- XML[50]

See also

- Aspect-oriented software development
- Distributed AOP
- Attribute grammar, a formalism that can be used for aspect-oriented programming on top of functional programming languages

- Programming paradigms
- Subject-oriented programming, an alternative to Aspect-oriented programming
- Role-oriented programming, an alternative to Aspect-oriented programming
- Predicate dispatch, an older alternative to Aspect-oriented programming
- Executable UML
- Decorator pattern
- Domain-driven design

Notes and references

1. Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J. M.; Irwin, J. (1997). *Aspect-oriented programming* (http://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP1997-AOP.pdf) (PDF). ECOOP'97. *Proceedings of the 11th European Conference on Object-Oriented Programming*. LNCS. **1241**. pp. 220–242. CiteSeerX 10.1.1.115.8660 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.115.8660). doi:10.1007/BFb0053381 (https://doi.org/10.1007%2FBFb0053381). ISBN 3-540-63089-9. Archived (https://web.archive.org/web/20160112141810/http://www.cs.ubc.ca/%7Egregor/papers/kiczales-ECOOP1997-AOP.pdf) (PDF) from the original on 2016-01-12.
2. "Adaptive Object Oriented Programming: The Demeter Approach with Propagation Patterns" *Karl Liebherr* 1996 ISBN 0-534-94602-X presents a well-worked version of essentially the same thing (Lieberherr subsequently recognized this and reframed his approach).
3. Don Box; Chris Sells (4 November 2002). *Essential.NET: The common language runtime* (https://books.google.com/books?id=Kl1DVZ8wTqcC&pg=PA206). Addison-Wesley Professional. p. 206. ISBN 978-0-201-73411-9. Retrieved 4 October 2011.
4. Roman, Ed; Sriganesh, Rima Patel; Brose, Gerald (1 January 2005). *Mastering Enterprise JavaBeans* (https://books.google.com/books?id=60oym_-uu3EC&pg=PA285). John Wiley and Sons. p. 285. ISBN 978-0-7645-8492-3. Retrieved 4 October 2011.
5. Note: The examples in this article appear in a syntax that resembles that of the Java language.
6. "gnu.org" (https://www.gnu.org/software/emacs/manual/html_node/elisp/Advising-Functions.html). *www.gnu.org*. Archived (https://web.archive.org/web/20171224053656/http://www.gnu.org/software/emacs/manual/html_node/elisp/Advising-Functions.html) from the original on 24 December 2017. Retrieved 5 May 2018.
7. "Archived copy" (https://web.archive.org/web/20051008065854/http://www.forum2.org/tal/AspectJ2EE.pdf) (PDF). Archived from the original (http://www.forum2.org/tal/AspectJ2EE.pdf) (PDF) on 2005-10-08. Retrieved 2005-06-19.
8. B. De Win, B. Vanhaute and B. De Decker. Security through aspect-oriented programming. In Advances in Network and Distributed Systems Security 2002.
9. T. Pasquier, J. Bacon and B. Shand. FlowR: Aspect Oriented Programming for Information Flow Control in Ruby. In ACM Proceedings of the 13th international conference on Modularity (Aspect Oriented Software Development) 2014.
10. Edsger Dijkstra, *Notes on Structured Programming* (http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF) Archived (https://web.archive.org/web/20061012020239/http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF) 2006-10-12 at the Wayback Machine, pg. 1-2
11. "*AOP Considered Harmful*" (http://pp.info.uni-karlsruhe.de/uploads/publikationen/constantinides04eiwas.pdf) (PDF). *uni-karlsruhe.de*. Archived (https://web.archive.org/web/20160323061458/https://pp.info.uni-karlsruhe.de/uploads/publikationen/constantinides04eiwas.pdf) (PDF) from the original on 23 March 2016. Retrieved 5 May 2018.

12. "AOP Considered Harmful (https://pp.info.uni-karlsruhe.de/uploads/publikationen/constantinides04eiwas.pdf) Archived (https://web.archive.org/web/20160304055622/http://pp.info.uni-karlsruhe.de/uploads/publikationen/constantinides04eiwas.pdf) 2016-03-04 at the Wayback Machine", Constantinos Constantinides, Therapon Skotiniotis, Maximilian Störzer, *European Interactive Workshop on Aspects in Software* (EIWAS), Berlin, Germany, September 2004.
13. C2:ComeFrom
14. Steimann, F. (2006). "The paradoxical success of aspect-oriented programming". *ACM SIGPLAN Notices*. **41** (10): 481. CiteSeerX 10.1.1.457.2210 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.457.2210). doi:10.1145/1167515.1167514 (https://doi.org/10.1145%2F1167515.1167514)., (slides (http://people.dsv.su.se/~johano/ioor/succ_aop.pdf) Archived (https://web.archive.org/web/20160304060007/http://people.dsv.su.se/~johano/ioor/succ_aop.pdf) 2016-03-04 at the Wayback Machine,slides 2 (http://www.eecs.ucf.edu/~leavens/modular-aop/Discussion.pdf) Archived (https://web.archive.org/web/20150923234021/http://www.eecs.ucf.edu/~leavens/modular-aop/Discussion.pdf) 2015-09-23 at the Wayback Machine, abstract (http://www.oopsla.org/2006/submission/essays/the_paradoxical_success_of_aspect-oriented_programming.html) Archived (https://web.archive.org/web/20150924060711/http://www.oopsla.org/2006/submission/essays/the_paradoxical_success_of_aspect-oriented_programming.html) 2015-09-24 at the Wayback Machine), Friedrich Steimann, Gary T. Leavens, OOPSLA 2006
15. "More Modular Reasoning for Aspect-Oriented Programs" (http://www.eecs.ucf.edu/~leavens/modular-aop/). Archived (https://web.archive.org/web/20150812045258/http://www.eecs.ucf.edu/~leavens/modular-aop/) from the original on 12 August 2015. Retrieved 11 August 2015.
16. "AOP and the Antinomy of the Liar" (http://www.fernuni-hagen.de/ps/pubs/FOAL2006.pdf) (PDF). *fernuni-hagen.de*. Archived (https://web.archive.org/web/20170809201001/http://www.fernuni-hagen.de/ps/pubs/FOAL2006.pdf) (PDF) from the original on 9 August 2017. Retrieved 5 May 2018.
17. Numerous: Afterthought (https://github.com/vc3/Afterthought) Archived (https://web.archive.org/web/20160315162029/https://github.com/vc3/Afterthought) 2016-03-15 at the Wayback Machine, LOOM.NET (http://www.rapier-loom.net/) Archived (https://web.archive.org/web/20080827215106/http://www.rapier-loom.net/) 2008-08-27 at the Wayback Machine, Enterprise Library 3.0 Policy Injection Application Block (http://www.codeplex.com/entlib) Archived (https://web.archive.org/web/20070119154829/http://www.codeplex.com/entlib) 2007-01-19 at the Wayback Machine, AspectDNG (http://sourceforge.net/projects/aspectdng/) Archived (https://web.archive.org/web/20040929053513/http://sourceforge.net/projects/aspectdng) 2004-09-29 at the Wayback Machine, DynamicProxy (http://www.castleproject.org/projects/dynamicproxy/) Archived (https://web.archive.org/web/20151205001755/http://www.castleproject.org/projects/dynamicproxy/) 2015-12-05 at the Wayback Machine, Compose* (http://composestar.sourceforge.net/) Archived (http://archive.wikiwix.com/cache/20050821193804/http://composestar.sourceforge.net/) 2005-08-21 at Wikiwix, PostSharp (http://www.postsharp.net/) Archived (https://web.archive.org/web/20160503095409/https://www.postsharp.net/) 2016-05-03 at the Wayback Machine, Seasar.NET (http://www.seasar.org/en/dotnet/) Archived (https://web.archive.org/web/20060725015434/http://www.seasar.org/en/dotnet/) 2006-07-25 at the Wayback Machine, DotSpect (.SPECT) (http://dotspect.tigris.org/) Archived (https://web.archive.org/web/20060331071126/http://dotspect.tigris.org/) 2006-03-31 at the Wayback Machine, Spring.NET (http://www.springframework.net/) Archived (https://web.archive.org/web/20060402205922/http://springframework.net/) 2006-04-02 at the Wayback Machine (as part of its functionality), Wicca and Phx.Morph (http://www.cs.columbia.edu/~eaddy/wicca) Archived (https://web.archive.org/web/20061207110849/http://www1.cs.columbia.edu/~eaddy/wicca/) 2006-12-07 at the Wayback Machine, SetPoint (http://setpoint.codehaus.org/) Archived (https://web.archive.org/web/20081007002006/http://setpoint.codehaus.org/) 2008-10-07 at the Wayback Machine
18. "Welcome to as3-commons-bytecode" (http://www.as3commons.org/as3-commons-bytecode). *as3commons.org*. Archived (https://web.archive.org/web/20141003100345/http://www.as3commons.org/as3-commons-bytecode/) from the original on 3 October 2014. Retrieved 5 May 2018.
19. "Ada2012 Rationale" (http://www.adacore.com/uploads/technical-papers/Ada2012_Rational_Introducion.pdf) (PDF). *adacore.com*. Archived (https://web.archive.org/web/20160418132340/http://www.adacore.com/uploads/technical-papers/Ada2012_Rational_Introducion.pdf) (PDF) from the original on 18 April 2016. Retrieved 5 May 2018.
20. "Function Hooks" (https://archive.is/20130117125117/http://www.autohotkey.com/forum/viewtopic.php?p=243426). *autohotkey.com*. Archived from the original (http://www.autohotkey.com/forum/viewtopic.php?p=243426) on 17 January 2013. Retrieved 5 May 2018.

21. Several: AspectC++, FeatureC++ (http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/fop/featurec/), AspectC (http://www.cs.ubc.ca/labs/spl/projects/aspectc.html) Archived (https://web.archive.org/web/20060821190630/http://www.cs.ubc.ca/labs/spl/projects/aspectc.html) 2006-08-21 at the Wayback Machine, AspeCt-oriented C (http://www.aspectc.net/) Archived (https://web.archive.org/web/20081120144608/http://www.aspectc.net/) 2008-11-20 at the Wayback Machine, Aspicere (https://archive.is/20120721212648/http://www.bramadams.org/aspicere/index.html)
22. "Cobble" (http://homepages.vub.ac.be/~kdeschut/cobble/). *vub.ac.be*. Retrieved 5 May 2018.
23. "AspectCocoa" (https://web.archive.org/web/20071026022525/http://www.ood.neu.edu/aspectcocoa/). *neu.edu*. Archived from the original (http://www.ood.neu.edu/aspectcocoa/) on 26 October 2007. Retrieved 5 May 2018.
24. "ColdSpring Framework: Welcome" (https://web.archive.org/web/20051105014513/http://coldspringframework.org/). 5 November 2005. Archived from the original on 5 November 2005. Retrieved 5 May 2018.
25. "Closer Project: AspectL" (http://common-lisp.net/project/closer/aspectl.html). Archived (http://archive.wikiwix.com/cache/20110223172923/http://common-lisp.net/project/closer/aspectl.html) from the original on 23 February 2011. Retrieved 11 August 2015.
26. "infra - Frameworks Integrados para Delphi - Google Project Hosting" (https://code.google.com/p/infra/). Archived (https://web.archive.org/web/20150909070130/http://code.google.com/p/infra/) from the original on 9 September 2015. Retrieved 11 August 2015.
27. "meaop - MeSDK: MeObjects, MeRTTI, MeAOP - Delphi AOP(Aspect Oriented Programming), MeRemote, MeService... - Google Project Hosting" (https://code.google.com/p/meaop/). Archived (https://web.archive.org/web/20150910210536/http://code.google.com/p/meaop/) from the original on 10 September 2015. Retrieved 11 August 2015.
28. "Google Project Hosting" (https://code.google.com/p/delphisorcery/). Archived (https://web.archive.org/web/20141225080131/https://code.google.com/p/delphisorcery/) from the original on 25 December 2014. Retrieved 11 August 2015.
29. "RemObjects Cirrus" (https://web.archive.org/web/20120123094027/http://prismwiki.codegear.com/en/Cirrus). *codegear.com*. Archived from the original (http://prismwiki.codegear.com/en/Cirrus) on 23 January 2012. Retrieved 5 May 2018.
30. "Emacs Advice Functions" (https://www.gnu.org/software/emacs/elisp/html_node/Advising-Functions.html). *gnu.org*. Archived (https://web.archive.org/web/20111024211408/http://www.gnu.org/software/emacs/elisp/html_node/Advising-Functions.html) from the original on 24 October 2011. Retrieved 5 May 2018.
31. monad (functional programming) ("Monads As a theoretical basis for AOP". CiteSeerX 10.1.1.25.8262 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.8262).) and Aspect-oriented programming with type classes. (http://portal.acm.org/citation.cfm?id=1233842) A Typed Monadic Embedding of Aspects (http://dl.acm.org/citation.cfm?id=2451457)

32. Numerous others: CaesarJ (http://www.caesarj.org/) Archived (https://web.archive.org/web/20081219181529/http://caesarj.org/) 2008-12-19 at the Wayback Machine, Compose* (http://composestar.sourceforge.net/) Archived (http://archive.wikiwix.com/cache/20050821193804/http://composestar.sourceforge.net/) 2005-08-21 at Wikiwix, Dynaop (http://dynaop.dev.java.net/) Archived (https://web.archive.org/web/20070724061030/https://dynaop.dev.java.net/) 2007-07-24 at the Wayback Machine, JAC (http://jac.objectweb.org/) Archived (https://web.archive.org/web/20040619135517/http://jac.objectweb.org/) 2004-06-19 at the Wayback Machine, Google Guice (as part of its functionality), Javassist (http://www.csg.is.titech.ac.jp/~chiba/javassist/) Archived (https://web.archive.org/web/20040901095342/http://www.csg.is.titech.ac.jp/~chiba/javassist/) 2004-09-01 at the Wayback Machine, JAsCo (and AWED) (http://ssel.vub.ac.be/jasco/) Archived (https://web.archive.org/web/20050411010213/http://ssel.vub.ac.be/jasco/) 2005-04-11 at the Wayback Machine, JAML (http://www.ics.uci.edu/~trungcn/jaml/) Archived (https://web.archive.org/web/20050415091244/http://www.ics.uci.edu/~trungcn/jaml/) 2005-04-15 at the Wayback Machine, JBoss AOP (http://labs.jboss.com/portal/jbossaop) Archived (https://web.archive.org/web/20061017211354/http://labs.jboss.com/portal/jbossaop/) 2006-10-17 at the Wayback Machine, LogicAJ (http://roots.iai.uni-bonn.de/research/logicaj) Archived (https://web.archive.org/web/20060504154852/http://roots.iai.uni-bonn.de/research/logicaj/) 2006-05-04 at the Wayback Machine, Object Teams (http://www.objectteams.org/) Archived (https://web.archive.org/web/20050831200910/http://objectteams.org/) 2005-08-31 at the Wayback Machine, PROSE (http://prose.ethz.ch/) Archived (https://web.archive.org/web/20070124094344/http://prose.ethz.ch/) 2007-01-24 at the Wayback Machine, The AspectBench Compiler for AspectJ (abc) (http://www.aspectbench.org/) Archived (https://web.archive.org/web/20141216200424/http://aspectbench.org/) 2014-12-16 at the Wayback Machine, Spring framework (as part of its functionality), Seasar, The JMangler Project (http://roots.iai.uni-bonn.de/research/jmangler/) Archived (https://web.archive.org/web/20051028191025/http://roots.iai.uni-bonn.de/research/jmangler/) 2005-10-28 at the Wayback Machine, InjectJ (http://injectj.sourceforge.net/) Archived (https://web.archive.org/web/20050405080539/http://injectj.sourceforge.net/) 2005-04-05 at the Wayback Machine, GluonJ (http://www.csg.is.titech.ac.jp/projects/gluonj/) Archived (https://web.archive.org/web/20070206010219/http://www.csg.is.titech.ac.jp/projects/gluonj/) 2007-02-06 at the Wayback Machine, Steamloom (http://www.st.informatik.tu-darmstadt.de/static/pages/projects/AORTA/Steamloom.jsp) Archived (https://web.archive.org/web/20070818100432/http://www.st.informatik.tu-darmstadt.de/static/pages/projects/AORTA/Steamloom.jsp) 2007-08-18 at the Wayback Machine
33. Many: Advisable (http://i.gotfresh.info/2007/12/7/advised-methods-for-javascript-with-prototype/) Archived (https://web.archive.org/web/20080704052200/http://i.gotfresh.info/2007/12/7/advised-methods-for-javascript-with-prototype) 2008-07-04 at the Wayback Machine, Ajaxpect (https://code.google.com/p/ajaxpect/) Archived (https://web.archive.org/web/20160709203939/https://code.google.com/p/ajaxpect/) 2016-07-09 at the Wayback Machine, jQuery AOP Plugin (http://plugins.jquery.com/project/AOP) Archived (https://web.archive.org/web/20080113184156/http://plugins.jquery.com/project/AOP) 2008-01-13 at the Wayback Machine, Aspectes (http://aspectes.tigris.org/) Archived (http://archive.wikiwix.com/cache/20060508035836/http://aspectes.tigris.org/) 2006-05-08 at Wikiwix, AspectJS (http://www.aspectjs.com/) Archived (https://web.archive.org/web/20081216010832/http://www.aspectjs.com/) 2008-12-16 at the Wayback Machine, Cerny.js (http://www.cerny-online.com/cerny.js/) Archived (https://web.archive.org/web/20070627024906/http://www.cerny-online.com/cerny.js/) 2007-06-27 at the Wayback Machine, Dojo Toolkit (http://dojotoolkit.org/) Archived (https://web.archive.org/web/20060221211958/http://www.dojotoolkit.org/) 2006-02-21 at the Wayback Machine, Humax Web Framework (http://humax.sourceforge.net/) Archived (https://web.archive.org/web/20081209103012/http://humax.sourceforge.net/) 2008-12-09 at the Wayback Machine, Joose (https://code.google.com/p/joose-js/) Archived (https://web.archive.org/web/20150318193601/http://code.google.com/p/joose-js/) 2015-03-18 at the Wayback Machine, Prototype - Prototype Function#wrap (http://www.prototypejs.org/api/function/wrap) Archived (https://web.archive.org/web/20090505233620/http://www.prototypejs.org/api/function/wrap) 2009-05-05 at the Wayback Machine, YUI 3 (Y.Do) (http://developer.yahoo.com/yui/3/api/Do.html) Archived (https://web.archive.org/web/20110125115930/https://developer.yahoo.com/yui/3/api/Do.html) 2011-01-25 at the Wayback Machine
34. Using built-in support for categories (which allows the encapsulation of aspect code) and event-driven programming (which allows the definition of *before* and after *event* handlers).
35. "AspectLua" (http://luaforge.net/projects/aspectlua/). Archived (https://web.archive.org/web/20150717094121/http://luaforge.net/projects/aspectlua/) from the original on 17 July 2015. Retrieved 11 August 2015.
36. "MAKAO, re(verse)-engineering build systems" (https://archive.is/20120724151524/http://www.bramadams.org/makao/). Archived from the original (http://www.bramadams.org/makao/) on 24 July 2012. Retrieved 11 August 2015.

37. "McLab" (http://www.sable.mcgill.ca/mclab/aspectmatlab/index.html). Archived (https://web.archive.org/web/20150924093214/http://www.sable.mcgill.ca/mclab/aspectmatlab/index.html) from the original on 24 September 2015. Retrieved 11 August 2015.
38. "AspectML - Aspect-oriented Functional Programming Language Research" (http://www.cs.princeton.edu/sip/projects/aspectml/). Archived (https://web.archive.org/web/20101205005108/http://www.cs.princeton.edu/sip/projects/aspectml/) from the original on 5 December 2010. Retrieved 11 August 2015.
39. Adam Kennedy. "Aspect - Aspect-Oriented Programming (AOP) for Perl - metacpan.org" (https://metacpan.org/module/Aspect). Archived (https://web.archive.org/web/20130831064935/https://metacpan.org/module/Aspect) from the original on 31 August 2013. Retrieved 11 August 2015.
40. Several: PHP-AOP (AOP.io) (http://aop.io) Archived (http://archive.wikiwix.com/cache/20140818050736/http://aop.io/) 2014-08-18 at Wikiwix, Go! AOP framework (http://go.aopphp.com) Archived (https://web.archive.org/web/20130301043014/http://go.aopphp.com/) 2013-03-01 at the Wayback Machine, PHPaspect (https://code.google.com/p/phpaspect/) Archived (https://web.archive.org/web/20160822234503/https://code.google.com/p/phpaspect/) 2016-08-22 at the Wayback Machine, Seasar.PHP (http://www.seasar.org/en/php5/index.html) Archived (https://web.archive.org/web/20051226040309/http://www.seasar.org/en/php5/index.html) 2005-12-26 at the Wayback Machine, PHP-AOP (https://archive.is/20120712081326/http://php-aop.googlecode.com/), Flow (https://flow.neos.io/) Archived (https://web.archive.org/web/20180104132543/https://flow.neos.io/) 2018-01-04 at the Wayback Machine, AOP PECL Extension (https://github.com/AOP-PHP/AOP) Archived (https://web.archive.org/web/20170411031809/https://github.com/AOP-PHP/AOP) 2017-04-11 at the Wayback Machine
41. "bigzaphod.org is coming soon" (http://www.bigzaphod.org/whirl/dma/docs/aspects/aspects-man.html). *www.bigzaphod.org*. Archived (https://web.archive.org/web/20160420181837/http://www.bigzaphod.org/whirl/dma/docs/aspects/aspects-man.html) from the original on 20 April 2016. Retrieved 5 May 2018.
42. Several: PEAK (http://peak.telecommunity.com/) Archived (https://web.archive.org/web/20050409082546/http://peak.telecommunity.com/) 2005-04-09 at the Wayback Machine, Aspyct AOP (https://web.archive.org/web/20110609153559/http://old.aspyct.org/), Lightweight Python AOP (http://www.cs.tut.fi/~ask/aspects/aspects.html) Archived (https://web.archive.org/web/20041009194711/http://www.cs.tut.fi/~ask/aspects/aspects.html) 2004-10-09 at the Wayback Machine, Logilab's aspect module (http://www.logilab.org/projects/aspects) Archived (https://web.archive.org/web/20050309034259/http://www.logilab.org/projects/aspects) 2005-03-09 at the Wayback Machine, Pythius (http://pythius.sourceforge.net/) Archived (https://web.archive.org/web/20050408072457/http://pythius.sourceforge.net/) 2005-04-08 at the Wayback Machine, Spring Python's AOP module (http://springpython.webfactional.com/1.1.x/reference/html/aop.html) Archived (https://web.archive.org/web/20160304055741/http://springpython.webfactional.com/1.1.x/reference/html/aop.html) 2016-03-04 at the Wayback Machine, Pytilities' AOP module (http://pytilities.sourceforge.net/doc/1.1.0/guide/aop/) Archived (https://web.archive.org/web/20110825101213/http://pytilities.sourceforge.net/doc/1.1.0/guide/aop/) 2011-08-25 at the Wayback Machine, aspectlib (http://python-aspectlib.readthedocs.org/en/latest/) Archived (https://web.archive.org/web/20141105061010/http://python-aspectlib.readthedocs.org/en/latest/) 2014-11-05 at the Wayback Machine
43. "PLaneT Package Repository : PLaneT > dutchyn > aspectscheme.plt" (http://planet.racket-lang.org/display.ss?package=aspectscheme.plt&owner=dutchyn). Archived (https://web.archive.org/web/20150905062740/http://planet.racket-lang.org/display.ss?package=aspectscheme.plt&owner=dutchyn) from the original on 5 September 2015. Retrieved 11 August 2015.
44. "AspectR - Simple aspect-oriented programming in Ruby" (http://aspectr-fork.sourceforge.net/). Archived (http://archive.wikiwix.com/cache/20150812003432/http://aspectr-fork.sourceforge.net/) from the original on 12 August 2015. Retrieved 11 August 2015.
45. Dean Wampler. "Home" (https://web.archive.org/web/20071026055811/http://aquarium.rubyforge.org/). Archived from the original (http://aquarium.rubyforge.org/) on 26 October 2007. Retrieved 11 August 2015.
46. "gcao/aspector" (https://github.com/gcao/aspector). *GitHub*. Archived (https://web.archive.org/web/20150104180534/https://github.com/gcao/aspector) from the original on 4 January 2015. Retrieved 11 August 2015.
47. "AspectS" (https://web.archive.org/web/20060106112030/http://www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/). *tu-ilmenau.de*. Archived from the original (http://www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/) on 6 January 2006. Retrieved 5 May 2018.

48. "MetaclassTalk: Reflection and Meta-Programming in Smalltalk" (http://csl.ensm-douai.fr/MetaclassTalk). Archived (https://web.archive.org/web/20150729062351/http://csl.ensm-douai.fr/MetaclassTalk) from the original on 29 July 2015. Retrieved 11 August 2015.
49. "WEAVR" (http://www.iit.edu/~concur/weavr). *iit.edu*. Archived (https://web.archive.org/web/20081212200221/http://www.iit.edu/~concur/weavr/) from the original on 12 December 2008. Retrieved 5 May 2018.
50. "aspectxml - An Aspect-Oriented XML Weaving Engine (AXLE) - Google Project Hosting" (https://code.google.com/p/aspectxml/). Archived (https://web.archive.org/web/20150912161613/http://code.google.com/p/aspectxml/) from the original on 12 September 2015. Retrieved 11 August 2015.

Further reading

- Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C.; Loingtier, J. M.; Irwin, J. (1997). *Aspect-oriented programming* (http://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP1997-AOP.pdf) (PDF). ECOOP'97. *Proceedings of the 11th European Conference on Object-Oriented Programming*. LNCS. **1241**. pp. 220–242. CiteSeerX 10.1.1.115.8660 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.115.8660). doi:10.1007/BFb0053381 (https://doi.org/10.1007%2FBFb0053381). ISBN 3-540-63089-9. The paper generally considered to be the authoritative reference for AOP.
- Andreas Holzinger; M. Brugger; W. Slany (2011). *Applying Aspect Oriented Programming (AOP) in Usability Engineering processes: On the example of Tracking Usage Information for Remote Usability Testing*. *Proceedings of the 8th International Conference on electronic Business and Telecommunications. Sevilla*. D. A. Marca, B. Shishkov and M. v. Sinderen (editors). pp. 53–56.
- Robert E. Filman; Tzilla Elrad; Siobhán Clarke; Mehmet Aksit (2004). *Aspect-Oriented Software Development*. ISBN 978-0-321-21976-3.
- Renaud Pawlak, Lionel Seinturier & Jean-Philippe Retaillé (2005). *Foundations of AOP for J2EE Development*. ISBN 978-1-59059-507-7.
- Laddad, Ramnivas (2003). *AspectJ in Action: Practical Aspect-Oriented Programming*. ISBN 978-1-930110-93-9.
- Jacobson, Ivar; Pan-Wei Ng (2005). *Aspect-Oriented Software Development with Use Cases*. ISBN 978-0-321-26888-4.
- Aspect-oriented Software Development and PHP, Dmitry Sheiko, 2006 (http://www.cmsdevelopment.com/en/articles/aosdinphp/)
- Siobhán Clarke & Elisa Baniassad (2005). *Aspect-Oriented Analysis and Design: The Theme Approach*. ISBN 978-0-321-24674-5.
- Raghu Yedduladoddi (2009). *Aspect Oriented Software Development: An Approach to Composing UML Design Models*. ISBN 978-3-639-12084-4.
- "Adaptive Object-Oriented Programming Using Graph-Based Customization" – Lieberherr, Silva-Lepe, *et al.* - 1994
- Zambrano Polo y La Borda, Arturo Federico (5 June 2013). "Addressing aspect interactions in an industrial setting: experiences, problems and solutions" (http://sedici.unlp.edu.ar/handle/10915/35861): 159. Retrieved 30 May 2014.
- Wijesuriya, Viraj Brian (2016-08-30) *Aspect Oriented Development, Lecture Notes, University of Colombo School of Computing, Sri Lanka (http://www.slideshare.net/tyrantbrian/aspect-oriented-development)*
- Groves, Matthew D. (2013). *AOP in .NET*. ISBN 9781617291142.

External links

- Eric Bodden's list of AOP tools (http://www.sable.mcgill.ca/aop.net/) in .net framework
- Programming Styles: Procedural, OOP, and AOP (http://www.dreamincode.net/forums/blog/gabehabe/index.php?showentry=1016)
- Programming Forum: Procedural, OOP, and AOP (http://forum.codecall.net/java-help/26210-object-reading-writing.html)
- Aspect-Oriented Software Development (https://web.archive.org/web/20030821074213/http://aosd.net/conference/), annual conference on AOP
- AOSD Wiki (http://aosd.net/wiki), Wiki on aspect-oriented software development
- AspectJ Programming Guide (http://www.eclipse.org/aspectj/doc/released/progguide/index.html)
- The AspectBench Compiler for AspectJ (https://web.archive.org/web/20141216200424/http://aspectbench.org/), another Java implementation

- Series of IBM developerWorks articles on AOP (http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=AOP@work:)
- A detailed series of articles on basics of aspect-oriented programming and AspectJ (http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect.html)
- What is Aspect-Oriented Programming? (https://web.archive.org/web/20090821185951/http://codefez.com/what-is-aspect-oriented-programming/), introduction with RemObjects Taco
- Constraint-Specification Aspect Weaver (http://www.cis.uab.edu/gray/Research/C-SAW/)
- Aspect- vs. Object-Oriented Programming: Which Technique, When? (http://www.devx.com/Java/Article/28422)
- Gregor Kiczales, Professor of Computer Science, explaining AOP (http://video.google.com/videoplay?docid=8566923311315412414&q=engEDU), video 57 min.
- Aspect Oriented Programming in COBOL (http://database.ittoolbox.com/documents/academic-articles/what-does-aspectoriented-programming-mean-to-cobol-4570)
- Aspect-Oriented Programming in Java with Spring Framework (http://static.springframework.org/spring/docs/2.0.x/reference/aop.html)
- Wiki dedicated to AOP methods on.NET (http://www.sharpcrafters.com/aop.net)
- Early Aspects for Business Process Modeling (An Aspect Oriented Language for BPMN) (https://web.archive.org/web/20090124183215/http://dssg.cs.umb.edu/wiki/index.php/Early_Aspects_for_Business_Process_Modeling)
- Spring AOP and AspectJ Introduction (http://java2novice.com/spring/aop-and-aspectj-introduction/)
- AOSD Graduate Course at Bilkent University (http://www.cs.bilkent.edu.tr/~bedir/CS586-AOSD)
- Introduction to AOP - Software Engineering Radio Podcast Episode 106 (http://www.se-radio.net/podcast/2008-08/episode-106-introduction-aop)
- An Objective-C implementation of AOP by Szilveszter Molnar (https://web.archive.org/web/20120801133941/http://innoli.com/en/Innoli-EN/OpenSource.html)
- Aspect-Oriented programming for iOS and OS X by Manuel Gebele (https://github.com/forensix/MGAOP)
- DevExpress MVVM Framework. Introduction to POCO ViewModels (https://community.devexpress.com/blogs/wpf/archive/2013/12/04/devexpress-mvvm-framework-introduction-to-poco-viewmodels.aspx)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Aspect-oriented_programming&oldid=887730084"