Software design pattern

From Wikipedia, the free encyclopedia

In software engineering, a **design pattern** is a general reusable solution to a commonly occurring problem within a given context in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Many patterns imply object-orientation or more generally mutable state, and so may not be as applicable in functional programming languages, in which data is immutable or treated as such.

Design patterns reside in the domain of modules and interconnections. At a higher level there are architectural patterns that are larger in scope, usually describing an overall pattern followed by an entire system.[1]

There are many types of design patterns, like

- **Algorithm strategy patterns** addressing concerns related to high-level strategies describing how to exploit application characteristic on a computing platform.
- **Computational design patterns** addressing concerns related to key computation identification.
- **Execution patterns** that address concerns related to supporting application execution, including strategies in executing streams of tasks and building blocks to support task synchronization.
- **Implementation strategy patterns** addressing concerns related to implementing source code to support
    1. program organization, and
    2. the common data structures specific to parallel programming.
- **Structural design patterns** addressing concerns related to high-level structures of applications being developed.

Contents

History

Patterns originated as an architectural concept by Christopher Alexander (1977/79). In 1987, Kent Beck and Ward Cunningham began experimenting with the idea of applying patterns to programming and presented their results at the OOPSLA conference that year.[2][3] In the following years, Beck, Cunningham and others followed up on this work.

Design patterns gained popularity in computer science after the book *Design Patterns: Elements of Reusable Object-Oriented Software* was published in 1994 by the so-called "Gang of Four" (Gamma et al.). That same year, the first Pattern Languages of Programming Conference was held and the following year, the Portland Pattern Repository was set up for documentation of design patterns. The scope of the term remains a matter of dispute. Notable books in the design pattern genre include:

- Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN 0-201-63361-2.
- Buschmann, Frank; Regine Meunier, Hans Rohnert, Peter Sommerlad (1996). *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons. ISBN 0-471-95869-7.
- Schmidt, Douglas C.; Michael Stal, Hans Rohnert, Frank Buschmann (2000). *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. John Wiley & Sons. ISBN 0-471-60695-2.
- Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley. ISBN 978-0321127426.
- Hohpe, Gregor; Bobby Woolf (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley. ISBN 0-321-20068-3.
- Freeman, Eric T; Elisabeth Robson, Bert Bates, Kathy Sierra (2004). *Head First Design Patterns*. O'Reilly Media. ISBN 0-596-00712-4.

Although design patterns have been applied practically for a long time, formalization of the concept of design patterns languished for several years.[4]

In 2009 over 30 contributors collaborated with Thomas Erl on his book, *SOA Design Patterns*.[5] The goal of this book was to establish a de facto catalog of design patterns for SOA and service-orientation.[6] (Over 200+ IT professionals participated world-wide in reviewing Erl's book and patterns.) These patterns are also published and discussed on the community research site soapatterns.org (http://www.soapatterns.org)

Practice

Design patterns can speed up the development process by providing tested, proven development paradigms.[*citation needed*] Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems, and it also improves code readability for coders and architects who are familiar with the patterns.

In order to achieve flexibility, design patterns usually introduce additional levels of indirection, which in some cases may complicate the resulting designs and hurt application performance.

By definition, a pattern must be programmed anew into each application that uses it. Since some authors see this as a step backward from software reuse as provided by components, researchers have worked to turn patterns into components. Meyer and Arnout were able to provide full or partial componentization of two-thirds of the patterns they attempted.[7]

Often, people only understand how to apply certain software design techniques to certain problems[*citation needed*]. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that does not require specifics tied to a particular problem.

Structure

Design patterns are composed of several sections (see Documentation below). Of particular interest are the Structure, Participants, and Collaboration sections. These sections describe a *design motif*: a prototypical *micro-architecture* that developers copy and adapt to their particular designs to solve the recurrent problem described by the design pattern. A micro-architecture is a set of program constituents (e.g., classes, methods...) and their relationships. Developers use the design pattern by introducing in their designs this prototypical micro-architecture, which means that micro-architectures in their designs will have structure and organization similar to the chosen design motif.

In addition to this, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than *ad-hoc* designs.

Domain-specific patterns

Efforts have also been made to codify design patterns in particular domains, including use of existing design patterns as well as domain specific design patterns. Examples include user interface design patterns,[8] information visualization,[9] secure design,[10] "secure usability",[11] Web design [12] and business model design.[13]

The annual Pattern Languages of Programming Conference proceedings [14] include many examples of domain specific patterns.

Classification and list

Design patterns were originally grouped into the categories: creational patterns, structural patterns, and behavioral patterns, and described using the concepts of delegation, aggregation, and consultation. For further background on object-oriented design, see coupling and cohesion, inheritance, interface, and polymorphism. Another classification has also introduced the notion of architectural design pattern that may be applied at the architecture level of the software such as the Model–View–Controller pattern.

Name	Description	In *Design Patterns*	In *Code Complete*[15]	Other

Creational patterns				
Abstract factory	Provide an interface for creating families of related or dependent objects without specifying their concrete classes.	Yes	Yes	N/A
Builder	Separate the construction of a complex object from its representation allowing the same construction process to create various representations.	Yes	No	N/A
Factory method	Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses (dependency injection[16]).	Yes	Yes	N/A
Lazy initialization	Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed.	No	No	*PoEAA*[17]
Multiton	Ensure a class has only named instances, and provide global point of access to them.	No	No	N/A
Object pool	Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of connection pool and thread pool patterns.	No	No	N/A
Prototype	Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.	Yes	No	N/A
Resource acquisition is initialization	Ensure that resources are properly released by tying them to the lifespan of suitable objects.	No	No	N/A
Singleton	Ensure a class has only one instance, and provide a global point of access to it.	Yes	Yes	N/A
Structural patterns				
Adapter or Wrapper	Convert the interface of a class into another interface clients expect. An	Yes	Yes	N/A

	adapter lets classes work together that could not otherwise because of incompatible interfaces.			
Bridge	Decouple an abstraction from its implementation allowing the two to vary independently.	Yes	Yes	N/A
Composite	Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.	Yes	Yes	N/A
Decorator	Attach additional responsibilities to an object dynamically keeping the same interface. Decorators provide a flexible alternative to subclassing for extending functionality.	Yes	Yes	N/A
Facade	Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.	Yes	Yes	N/A
Front Controller	The pattern relates to the design of web applications. It provides a centralized entry point for handling requests.	No	Yes	N/A
Flyweight	Use sharing to support large numbers of similar objects efficiently.	Yes	No	N/A
Proxy	Provide a surrogate or placeholder for another object to control access to it.	Yes	No	N/A
Behavioral patterns				
Blackboard	Generalized observer, which allows multiple readers and writers. Communicates information system-wide.	No	No	N/A
Chain of responsibility	Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.	Yes	No	N/A
Command	Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable	Yes	No	N/A

	operations.			
Interpreter	Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.	Yes	No	N/A
Iterator	Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.	Yes	Yes	N/A
Mediator	Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.	Yes	No	N/A
Memento	Without violating encapsulation, capture and externalize an object's internal state allowing the object to be restored to this state later.	Yes	No	N/A
Null object	Avoid null references by providing a default object.	No	No	N/A
Observer or Publish/subscribe	Define a one-to-many dependency between objects where a state change in one object results with all its dependents being notified and updated automatically.	Yes	Yes	N/A
Servant	Define common functionality for a group of classes	No	No	N/A
Specification	Recombinable business logic in a Boolean fashion	No	No	N/A
State	Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.	Yes	No	N/A
Strategy	Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.	Yes	Yes	N/A
Template method	Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template method lets subclasses redefine certain steps of an	Yes	Yes	N/A

	algorithm without changing the algorithm's structure.			
Visitor	Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.	Yes	No	N/A

Name	Description	In *POSA2*[18]	Other
Concurrency patterns			
Active Object	Decouples method execution from method invocation that reside in their own thread of control. The goal is to introduce concurrency, by using asynchronous method invocation and a scheduler for handling requests.	Yes	N/A
Balking	Only execute an action on an object when the object is in a particular state.	No	N/A
Binding properties	Combining multiple observers to force properties in different objects to be synchronized or coordinated in some way.[19]	No	N/A
Messaging design pattern (MDP)	Allows the interchange of information (i.e. messages) between components and applications.	No	N/A
Double-checked locking	Reduce the overhead of acquiring a lock by first testing the locking criterion (the 'lock hint') in an unsafe manner; only if that succeeds does the actual lock proceed. Can be unsafe when implemented in some language/hardware combinations. It can therefore sometimes be considered an anti-pattern.	Yes	N/A
Event-based asynchronous	Addresses problems with the asynchronous pattern that occur in multithreaded programs.[20]	No	N/A
Guarded suspension	Manages operations that require both a lock to be acquired and a precondition to be satisfied before the operation can be executed.	No	N/A
Lock	One thread puts a "lock" on a resource, preventing other threads from accessing or modifying it.[21]	No	*PoEAA*[17]

Monitor object	An object whose methods are subject to mutual exclusion, thus preventing multiple objects from erroneously trying to use it at the same time.	Yes	N/A
Reactor	A reactor object provides an asynchronous interface to resources that must be handled synchronously.	Yes	N/A
Read-write lock	Allows concurrent read access to an object, but requires exclusive access for write operations.	No	N/A
Scheduler	Explicitly control when threads may execute single-threaded code.	No	N/A
Thread pool	A number of threads are created to perform a number of tasks, which are usually organized in a queue. Typically, there are many more tasks than threads. Can be considered a special case of the object pool pattern.	No	N/A
Thread-specific storage	Static or "global" memory local to a thread.	Yes	N/A

Documentation

The documentation for a design pattern describes the context in which the pattern is used, the forces within the context that the pattern seeks to resolve, and the suggested solution.[22] There is no single, standard format for documenting design patterns. Rather, a variety of different formats have been used by different pattern authors. However, according to Martin Fowler, certain pattern forms have become more well-known than others, and consequently become common starting points for new pattern-writing efforts.[23] One example of a commonly used documentation format is the one used by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (collectively known as the "Gang of Four", or GoF for short) in their book *Design Patterns*. It contains the following sections:

- **Pattern Name and Classification:** A descriptive and unique name that helps in identifying and referring to the pattern.
- **Intent:** A description of the goal behind the pattern and the reason for using it.
- **Also Known As:** Other names for the pattern.
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** Situations in which this pattern is usable; the context for the pattern.
- **Structure:** A graphical representation of the pattern. Class diagrams and Interaction diagrams may be used for this purpose.
- **Participants:** A listing of the classes and objects used in the pattern and their roles in the design.
- **Collaboration:** A description of how classes and objects used in the pattern interact with each other.
- **Consequences:** A description of the results, side effects, and trade offs caused by using the pattern.
- **Implementation:** A description of an implementation of the pattern; the solution part of

the pattern.
- **Sample Code:** An illustration of how the pattern can be used in a programming language.
- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

Criticism

The concept of design patterns has been criticized in several ways.

The design patterns may just be a sign of some missing features of a given programming language (Java or C++ for instance). Peter Norvig demonstrates that 16 out of the 23 patterns in the Design Patterns book (which is primarily focused on C++) are simplified or eliminated (via direct language support) in Lisp or Dylan.[24] See also Paul Graham's essay Revenge of the Nerds.[25]

The idea may not be as new as suggested by the authors: for instance the Model-View-Controller paradigm is an example of a "pattern" which predates the concept of "design patterns" by several years.

Moreover, shifting the code too far forcing it to look like a standard pattern unnecessarily increases complexity.[26]

See also

- Abstraction principle
- Algorithmic skeleton
- Anti-pattern
- Architectural pattern
- Business pattern
- Distributed design patterns
- Double-chance function
- Enterprise Architecture framework
- GRASP (object-oriented design)
- Interaction design pattern
- List of software development philosophies
- List of software engineering topics
- Pattern language
- Pattern theory
- Pedagogical patterns
- Portland Pattern Repository
- Refactoring
- Software development methodology

References

1. ^ Martin, Robert C.. "Design Principles and Design Patterns" (http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf) . http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf. Retrieved 2000.
2. ^ Smith, Reid (October 1987). "Panel on design methodology". *OOPSLA '87 Addendum to the Proceedings*. OOPSLA '87. doi:10.1145/62138.62151 (http://dx.doi.org/10.1145%2F62138.62151) ., *"Ward cautioned against requiring too much programming at, what he termed, 'the high level of wizards.' He pointed out that a written 'pattern language' can significantly improve the selection and application of abstractions. He proposed a 'radical shift in the burden of design and implementation' basing the new methodology on an adaptation of Christopher Alexander's work in pattern languages and that programming-oriented pattern languages developed at Tektronix has significantly aided their software development efforts."*
3. ^ Beck, Kent; Ward Cunningham (September 1987). "Using Pattern Languages for Object-Oriented Program" (http://c2.com/doc/oopsla87.html) . *OOPSLA '87 workshop on* Specification and Design for

Object-Oriented Programming. OOPSLA '87. http://c2.com/doc/oopsla87.html. Retrieved 2006-05-26.
4. ^ Baroni, Aline Lúcia; Yann-Gaël Guéhéneuc and Hervé Albin-Amiot (June 2003). "Design Patterns Formalization" (http://www.iro.umontreal.ca/~ptidej/Publications/Documents/Research+report+Metamodeling+June03.doc.pdf) (PDF). Nantes: École Nationale Supérieure des Techniques Industrielles et des Mines de Nantes. http://www.iro.umontreal.ca/~ptidej/Publications/Documents/Research+report+Metamodeling+June03.doc.pdf. Retrieved 2007-12-29.
5. ^ Erl, Thomas (2009). *SOA Design Patterns*. New York: Prentice Hall/PearsonPTR. p. 864. ISBN 0-13-613516-1.
6. ^ http://soa.sys-con.com/node/809800
7. ^ Meyer, Bertrand; Karine Arnout (July 2006). "Componentization: The Visitor Example" (http://se.ethz.ch/~meyer/publications/computer/visitor.pdf) . *IEEE Computer* (IEEE) **39** (7): 23–30. http://se.ethz.ch/~meyer/publications/computer/visitor.pdf.
8. ^ Laakso, Sari A. (2003-09-16). "Collection of User Interface Design Patterns" (http://www.cs.helsinki.fi/u/salaakso/patterns/index.html) . University of Helsinki, Dept. of Computer Science. http://www.cs.helsinki.fi/u/salaakso/patterns/index.html. Retrieved 2008-01-31.
9. ^ Heer, J.; M. Agrawala (2006). "Software Design Patterns for Information Visualization" (http://vis.berkeley.edu/papers/infovis_design_patterns/) . *IEEE Transactions on Visualization and Computer Graphics* **12** (5): 853. doi:10.1109/TVCG.2006.178 (http://dx.doi.org/10.1109%2FTVCG.2006.178) . PMID 17080809 (http://www.ncbi.nlm.nih.gov/pubmed/17080809) . http://vis.berkeley.edu/papers/infovis_design_patterns/.
10. ^ Chad Dougherty et al (2009). *Secure Design Patterns* (http://www.cert.org/archive/pdf/09tr010.pdf) . http://www.cert.org/archive/pdf/09tr010.pdf.
11. ^ Simson L. Garfinkel (2005). *Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable* (http://www.simson.net/thesis/) . http://www.simson.net/thesis/.
12. ^ "Yahoo! Design Pattern Library" (http://developer.yahoo.com/ypatterns/) . http://developer.yahoo.com/ypatterns/. Retrieved 2008-01-31.
13. ^ "How to design your Business Model as a Lean Startup?" (http://torgronsund.wordpress.com/2010/01/06/lean-startup-business-model-pattern/) . http://torgronsund.wordpress.com/2010/01/06/lean-startup-business-model-pattern/. Retrieved 2010-01-06.
14. ^ Pattern Languages of Programming, Conference proceedings (annual, 1994—) [1] (http://hillside.net/plop/pastconferences.html)
15. ^ McConnell, Steve (June 2004). "Design in Construction". *Code Complete* (2nd ed.). Microsoft Press. p. 104. ISBN 978-0735619678. "Table 5.1 Popular Design Patterns"
16. ^ "Design Patterns: Dependency injection" (http://msdn.microsoft.com/de-de/magazine/cc163739(en-us).aspx) . http://msdn.microsoft.com/de-de/magazine/cc163739(en-us).aspx. Retrieved 2011-04-13. "The use of a factory class is one common way to implement DI."
17. ^ ***a*** ***b*** Fowler, Martin (2002). *Patterns of Enterprise Application Architecture* (http://martinfowler.com/books.html#eaa) . Addison-Wesley. ISBN 978-0321127426. http://martinfowler.com/books.html#eaa.
18. ^ Schmidt, Douglas C.; Michael Stal, Hans Rohnert, Frank Buschmann (2000). *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. John Wiley & Sons. ISBN 0-471-60695-2.
19. ^ http://c2.com/cgi/wiki?BindingProperties
20. ^ Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, and Morgan Skinner (2008). "Event-based Asynchronous Pattern". *Professional C# 2008*. Wiley. pp. 570–571. ISBN 0470191376.
21. ^ http://c2.com/cgi/wiki?LockPattern
22. ^ Gabriel, Dick. "A Pattern Definition" (http://web.archive.org/web/20070209224120/http://hillside.net/patterns/definition.html) . Archived from the original (http://hillside.net/patterns/definition.html) on 2007-02-09. http://web.archive.org/web/20070209224120/http://hillside.net/patterns/definition.html. Retrieved 2007-03-06.
23. ^ Fowler, Martin (2006-08-01). "Writing Software Patterns" (http://www.martinfowler.com/articles/writingPatterns.html) . http://www.martinfowler.com/articles/writingPatterns.html. Retrieved 2007-03-06.
24. ^ Norvig, Peter (1998). "Design Patterns in Dynamic Languages" (http://www.norvig.com/design-patterns/) . http://www.norvig.com/design-patterns/.
25. ^ Graham, Paul (2002). "Revenge of the Nerds" (http://www.paulgraham.com/icad.html) . http://www.paulgraham.com/icad.html.
26. ^ McConnell, Steve (2004). *Code Complete: A Practical Handbook of Software Construction, 2nd Edition*. p. 105.

Further reading

- Alexander, Christopher; Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, Shlomo Angel (1977). *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press. ISBN 978-0195019193.
- Alur, Deepak; John Crupi, Dan Malks (May 2003). *Core J2EE Patterns: Best Practices and Design Strategies (2nd Edition)*. Prentice Hall. ISBN 0131422464.
- Beck, Kent (October 2007). *Implementation Patterns*. Addison-Wesley. ISBN 978-0321413093.
- Beck, Kent; R. Crocker, G. Meszaros, J.O. Coplien, L. Dominick, F. Paulisch, and J. Vlissides (March 1996). *Proceedings of the 18th International Conference on Software Engineering*. pp. 25–30.
- Borchers, Jan (2001). *A Pattern Approach to Interaction Design*. John Wiley & Sons. ISBN 0-471-49828-9.
- Coplien, James O.; Douglas C. Schmidt (1995). *Pattern Languages of Program Design*. Addison-Wesley. ISBN 0-201-60734-4.
- Coplien, James O.; John M. Vlissides, and Norman L. Kerth (1996). *Pattern Languages of Program Design 2*. Addison-Wesley. ISBN 0-201-89527-7.
- Fowler, Martin (1997). *Analysis Patterns: Reusable Object Models*. Addison-Wesley. ISBN 0-201-89542-0.
- Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley. ISBN 978-0321127426.
- Freeman, Eric; Elisabeth Freeman, Kathy Sierra, and Bert Bates (2004). *Head First Design Patterns*. O'Reilly Media. ISBN 0-596-00712-4.
- Hohmann, Luke; Martin Fowler and Guy Kawasaki (2003). *Beyond Software Architecture*. Addison-Wesley. ISBN 0-201-77594-8.
- Alur, Deepak; Elisabeth Freeman, Kathy Sierra, and Bert Bates (2004). *Head First Design Patterns*. O'Reilly Media. ISBN 0-596-00712-4.
- Gabriel, Richard (1996) (PDF). *Patterns of Software: Tales From The Software Community* (http://www.dreamsongs.com/NewFiles/PatternsOfSoftware.pdf) . Oxford University Press. p. 235. ISBN 0-19-512123-6. http://www.dreamsongs.com/NewFiles/PatternsOfSoftware.pdf.
- Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN 0-201-63361-2.
- Hohpe, Gregor; Bobby Woolf (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley. ISBN 0-321-20068-3.
- Holub, Allen (2004). *Holub on Patterns*. Apress. ISBN 1-59059-388-X.
- Kircher, Michael; Markus Völter and Uwe Zdun (2005). *Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware*. John Wiley & Sons. ISBN 0-470-85662-9.
- Larman, Craig (2005). *Applying UML and Patterns*. Prentice Hall. ISBN 0-13-148906-2.
- Liskov, Barbara; John Guttag (2000). *Program Development in Java: Abstraction, Specification, and Object-Oriented Design*.. Addison-Wesley. ISBN 0-201-65768-6.
- Manolescu, Dragos; Markus Voelter and James Noble (2006). *Pattern Languages of Program Design 5*. Addison-Wesley. ISBN 0-321-32194-4.
- Marinescu, Floyd (2002). *EJB Design Patterns: Advanced Patterns, Processes and Idioms*. John Wiley & Sons. ISBN 0-471-20831-0.
- Martin, Robert Cecil; Dirk Riehle and Frank Buschmann (1997). *Pattern Languages of Program Design 3*. Addison-Wesley. ISBN 0-201-31011-2.
- Mattson, Timothy G; Beverly A. Sanders and Berna L. Massingill (2005). *Patterns for*

Parallel Programming. Addison-Wesley. ISBN 0-321-22811-1.
- Shalloway, Alan; James R. Trott (2001). *Design Patterns Explained, Second Edition: A New Perspective on Object-Oriented Design*. Addison-Wesley. ISBN 0-321-24714-0.
- Vlissides, John M. (1998). *Pattern Hatching: Design Patterns Applied*. Addison-Wesley. ISBN 0-201-43293-5.
- Weir, Charles; James Noble (2000). *Small Memory Software: Patterns for systems with limited memory* (http://www.cix.co.uk/~smallmemory/) . Addison-Wesley. ISBN 0201596075. http://www.cix.co.uk/~smallmemory/.

External links

- 101 Design Patterns & Tips for Developers (http://sourcemaking.com/design-patterns-and-tips)
- Design Patterns Reference (http://www.oodesign.com/) at oodesign.com
- Directory of websites that provide pattern catalogs (http://hillside.net/patterns/onlinepatterncatalog.htm) at hillside.net.
- Jt (http://jt.dev.java.net) J2EE Pattern Oriented Framework
- Lean Startup Business Model Pattern (http://torgronsund.wordpress.com/2010/01/06/lean-startup-business-model-pattern/) Example of design pattern thinking applied to business models
- Messaging Design Pattern (http://jt.dev.java.net/files/documents/5553/150311/designPatterns.pdf) Published in the 17th conference on Pattern Languages of Programs (PLoP 2010).
- On Patterns and Pattern Languages (http://media.wiley.com/product_data/excerpt/28/04700590/0470059028.pdf) by Buschmann, Henney, and Schmidt
- Patterns and Anti-Patterns (http://www.dmoz.org/Computers/Programming/Methodologies/Patterns_and_Anti-Patterns//) at the Open Directory Project
- Patterns for Scripted Applications (http://www.doc.ic.ac.uk/~np2/patterns/scripting/)
- PerfectJPattern Open Source Project (http://perfectjpattern.sourceforge.net) Design Patterns library that aims to provide full or partial componentized version of all known Patterns in Java.
- JPattern (http://jpatterns.org) JPatterns is a collection of annotations for Design Patterns.
- Printable Design Patterns Quick Reference Cards (http://www.mcdonaldland.info/2007/11/28/40/)
- Are Design Patterns Missing Language Features? at the Portland Pattern Repository
- History of Patterns at the Portland Pattern Repository
- Show Trial of the Gang of Four at the Portland Pattern Repository
- Category: Pattern at the Portland Pattern Repository
- "Design Patterns in Modern Day Software Factories (WCSF)" (http://www.xosoftware.co.uk/Articles/WCSFDesignPatterns/) . XO Software, Ltd. http://www.xosoftware.co.uk/Articles/WCSFDesignPatterns/.
- "Core J2EE Patterns: Patterns index page" (http://java.sun.com/blueprints/corej2eepatterns/Patterns/) . Sun Microsystems, Inc.. http://java.sun.com/blueprints/corej2eepatterns/Patterns/.

Categories: Software design patterns