

Design pattern (computer science)

From Wikipedia, the free encyclopedia

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem 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. Algorithms are not thought of as design patterns, since they solve computational problems rather than design problems.

See also: Anti-pattern

Contents

- 1 History
- 2 Uses
- 3 Classification
- 4 Documentation
- 5 Criticism
 - 5.1 Targets the wrong problem
 - 5.2 Lacks formal foundations
 - 5.3 Unlike components, does not provide reuse
 - 5.4 Leads to inefficient solutions
 - 5.5 Does not differ significantly from other abstractions
- 6 See also
- 7 Notes
- 8 References
- 9 External links

History

Patterns originated as an architectural concept by Christopher Alexander. 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^{[1][2]}. 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 (Gamma et al). That same year, the first Pattern Languages of Programs conference was held and the following year, the Portland Pattern Repository was set up for documentation of design patterns. The scope of the term remained a matter of dispute into the next decade.

Uses

Design patterns can speed up the development process by providing tested, proven development paradigms. 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 improves code readability for coders and architects familiar with the patterns.

Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.

Design patterns are composed of several sections (see Documentation). 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, 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.

Classification

Design patterns can be classified in terms of the underlying problem they solve. Examples of problem-based pattern classifications include:

- Fundamental patterns
- Creational patterns
- Structural patterns
- Behavioral patterns
- Concurrency patterns
- Architectural patterns

Documentation

The documentation for a design pattern should contain enough information about the problem that the pattern addresses, the context in which it is used, and the suggested solution. Nonetheless, authors use their own layouts to document design patterns, and these layouts usually resemble the essential parts. The authors usually include additional sections to provide more information, and organize the essential parts in different sections, possibly with different names.

A commonly used format is the one used by the Gang of Four. It contains the following sections:

- **Pattern Name and Classification:** Every pattern should have a descriptive and unique name that helps in identifying and referring to it. Additionally, the pattern should be classified according to a classification such as the one described earlier. This classification helps in identifying the use of the pattern.

- **Intent:** This section should describe the goal behind the pattern and the reason for using it. It resembles the problem part of the pattern.
- **Also Known As:** A pattern could have more than one name. These names should be documented in this section.
- **Motivation(Forces):** This section provides a scenario consisting of a problem and a context in which this pattern can be used. By relating the problem and the context, this section shows when this pattern is used.
- **Applicability:** This section includes situations in which this pattern is usable. It represents the context part of the pattern.
- **Structure:** A graphical representation of the pattern. Class diagrams and Interaction diagrams can be used for this purpose.
- **Participants:** A listing of the classes and objects used in this pattern and their roles in the design.
- **Collaboration:** Describes how classes and objects used in the pattern interact with each other.
- **Consequences:** This section describes the results, side effects, and trade offs caused by using this pattern.
- **Implementation:** This section describes the implementation of the pattern, and represents the solution part of the pattern. It provides the techniques used in implementing this pattern, and suggests ways for this implementation.
- **Sample Code:** An illustration of how this pattern can be used in a programming language
- **Known Uses:** This section includes examples of real usages of this pattern.
- **Related Patterns:** This section includes other patterns that have some relation with this pattern, so that they can be used along with this pattern, or instead of this pattern. It also includes the differences this pattern has with similar patterns.

Criticism

The concept of design patterns has been criticized by some in the field of computer science.

Targets the wrong problem

The need for patterns results from using computer languages or techniques with insufficient abstraction ability. Under ideal *factoring*, a concept should not be copied, but merely referenced. But if something is referenced instead of copied, then there is no "pattern" to label and catalog. Paul Graham writes in the essay *Revenge of the Nerds*^[3].

This practice is not only common, but institutionalized. For example, in the OO world you hear a good deal about "patterns". I wonder if these patterns are not sometimes evidence of case (c), the human compiler, at work. When I see patterns in my programs, I consider it a sign of trouble. The shape of a program should reflect only the problem it needs to solve. Any other regularity in the code is a sign, to me at least, that I'm using abstractions that aren't powerful enough -- often that I'm generating by hand the expansions of some macro that I need to write.

Peter Norvig provides a similar argument. He 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^[4].

Further arguments along this line are discussed on Portland Pattern Repository's wiki.^{[5][6]}

Lacks formal foundations

Unlike components, does not provide reuse

A pattern must be programmed anew into each application that uses it. Some authors see this as a step backward from software reuse as provided by components. This observation has led to work on "componentization": turning patterns into components, in particular by Meyer and Arnout, who claim a 2/3rds success rate in componentizing the best-known patterns. ^[9]

Leads to inefficient solutions

The idea of a design pattern is an attempt to standardize what are already accepted best practices. In principle this might appear to be beneficial, but in practice it often results in the unnecessary duplication of code. It is almost always a more efficient solution to use a well-factored implementation rather than a "just barely good enough" design pattern.

Does not differ significantly from other abstractions

Some authors allege that design patterns don't differ significantly from other forms of abstraction^[10], and that the use of new terminology (borrowed from the architecture community) to describe existing phenomena in the field of programming is unnecessary. The Model-View-Controller paradigm is touted as an example of a "pattern" which predates the concept of "design patterns" by several years.^[11] It is further argued by some that the primary contribution of the Design Patterns community (and the Gang of Four book) was the use of Alexander's pattern language as a form of documentation; a practice which is often ignored in the literature.

See also

- Design Patterns for the list of patterns in the book by Gamma et al.
- Anti-pattern
- Interaction design pattern
- Pedagogical patterns
- Portland Pattern Repository
- Programming practice
- Refactoring
- Software engineering and List of software engineering topics
- Business Process Improvement Pattern
- List of software development philosophies

Notes

- ^{^ a b} Smith, R. (October 1987). "Panel on design methodology (<http://doi.acm.org/10.1145/62138.62151>)". *OOPSLA '87 Addendum to the Proceedings.*, *"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."

2. ^a ^b Beck, K.; Ward Cunningham (September 1987). Using Pattern Languages for Object-Oriented Program (<http://c2.com/doc/oopsla87.html>). Retrieved on 2006-05-26. Submitted to the OOPSLA '87 workshop on the *Specification and Design for Object-Oriented Programming*
3. ^a ^b Paul Graham (May 2002). Revenge of the Nerds (<http://www.paulgraham.com/icad.html>). Retrieved on 2006-01-20.
4. ^a ^b Peter Norvig (1998-03-17). Design Patterns in Dynamic Programming (<http://norvig.com/design-patterns/>). Retrieved on 2006-01-20.
5. ^a ^b Are Design Patterns Missing Language Features? (<http://www.c2.com/cgi/wiki?AreDesignPatternsMissingLanguageFeatures>). Cunningham & Cunningham, Inc.. Retrieved on 2006-01-20.
6. ^a ^b Design Patterns in Dynamic Programming (<http://www.c2.com/cgi/wiki?DesignPatternsInDynamicProgramming>). Cunningham & Cunningham, Inc.. Retrieved on 2006-01-20.
7. ^a Show Trial of the Gang of Four (<http://www.c2.com/cgi/wiki?ShowTrialOfTheGangOfFour>). Cunningham & Cunningham, Inc.. Retrieved on 2006-01-20.
8. ^a Show Trial Verdict (<http://www.c2.com/cgi/wiki?ShowTrialVerdict>). Cunningham & Cunningham, Inc.. Retrieved on 2006-01-20.
9. ^a Meyer, B. and Arnout, K.: Componentization: The Visitor Example, *IEEE Computer*, vol. 39, issue 7, pages 23-30, July 2006, pre-publication version available online (<http://se.ethz.ch/~meyer/publications/computer/visitor.pdf>)
10. ^a <http://www.lukew.com/ff/entry.asp?348>
11. ^a <http://rd13doc.cern.ch/Notes/004/Note004-7.html>

References

- Alexander, Christopher, et al (1977). *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press.
- Beck, K., 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*, 25-30.
- Borchers, Jan (2001). *A Pattern Approach to Interaction Design*. John Wiley & Sons. ISBN 0-471-49828-9.
- Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal (1996). *Pattern-oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons. ISBN 0-471-95869-7.
- Cooper, James W. (October 1998). *The Design Patterns Java Companion*. Addison-Wesley.
- 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.
- den Burger, Mathijs (March 2002). *Design Patterns for Networking Applications in Java*.
- Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley. ISBN 0-321-

12742-0.

- Fowler, Martin. *Patterns [software patterns] Software*, IEEE, Volume: 20, Issue: 2, March-April 2003. Pages: 56 – 57.
- Freeman, Eric, Elisabeth Freeman, Kathy Sierra, and Bert Bates (2004). *Head First Design Patterns*. O'Reilly Media, 637. ISBN 0-596-00712-4.
- Gabriel, Richard (1996). *Patterns of Software: Tales From The Software Community* (<http://www.dreamsongs.com/NewFiles/PatternsOfSoftware.pdf>). Oxford University Press, 235. ISBN 0-19-512123-6.
- Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*, hardcover, 395 pages, Addison-Wesley. ISBN 0-201-63361-2.
- Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides (1997). *Design Patterns CD*. ISBN 0-201-63498-8.
- Harrison, Neil, Brian Foote, and Hans Rohnert (1999). *Pattern Languages of Program Design 4*. Addison-Wesley. ISBN 0-201-43304-4.
- Hohpe, Gregor, Bobby Woolf (2004). *Enterprise Integration Patterns*. Addison-Wesley. ISBN 0-321-20068-3.
- Holub, Allen (2004). *Holub on Patterns*. Apress Publishers. ISBN 1-59059-388-X.
- Kaplan, Jonathan, William C. R. Crawford (2003). *J2EE Design Patterns*. O'Reilly. ISBN 0-596-00427-3.
- Kerievsky, Joshua (2004). *Refactoring to Patterns*. Addison-Wesley. ISBN 0-321-21335-1.
- Kircher, Michael, Prashant Jain (2004). *Pattern-oriented Software Architecture. Volume 3: Patterns for Resource Management*. John Wiley & Sons. ISBN 0-470-84525-2.
- 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, Third Edition*. Prentice Hall. ISBN 0-13-148906-2.
- 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.
- Metsker, Steven, William C. Wake (2006). *Design Patterns in Java™*. Addison-Wesley. ISBN 0-321-33302-0.
- Nock, Clifton (2003). *Data Access Patterns: Database Interactions in Object-Oriented Applications*. Addison-Wesley. ISBN 0-13-140157-2.
- Schmidt, Douglas C., Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-oriented Software Architecture. Volume 2: Patterns for Concurrent and Networked Objects*. John Wiley & Sons. ISBN 0-471-60695-2.
- Schmidt, Douglas C., Stephen D. Huston (2002). *C++ Network Programming: Mastering Complexity Using ACE and Patterns*. Addison-Wesley. ISBN 0-201-60464-7.
- Shalloway, Alan, James R. Trott (2001). *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison-Wesley. ISBN 0-201-71594-5.
- Vlissides, John M. (1998). *Pattern Hatching: Design Patterns Applied*. Addison-Wesley. ISBN 0-201-43293-5.
- History of Patterns (<http://c2.com/cgi-bin/wiki?HistoryOfPatterns>). *Portland Pattern Repository*. Retrieved on 2005-07-28.

External links

- The PatternShare community (<http://patternshare.org/>) - a community site for sharing patterns.
- Directory of websites that provide pattern catalogs (<http://hillside.net/patterns/onlinepatterncatalog.htm>) at hillside.net.
- Ward Cunningham's *Portland Pattern Repository* (<http://c2.com/cgi/wiki?CategoryPattern>).
- Patterns and Anti-Patterns (http://dmoz.org/Computers/Programming/Methodologies/Patterns_and_Anti-Patterns/) at the Open Directory Project
- Design Patterns Resources (<http://www.javawhat.com/showCategory.do?id=454>)
- Microsoft patterns & practices Home (<http://msdn.microsoft.com/practices>)en:Design Pattern

Retrieved from "http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29"

Categories: Articles with unsourced statements | Software design patterns

-
- This page was last modified 09:46, 30 October 2006.
 - All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.)
Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc.