**Bedeutung**

<https://de.wikipedia.org/wiki/Entwurfsmuster>

**Wer hat’s erfunden??**

That’s a good, but not a very accurate, question. Design patterns aren’t obscure, sophisticated concepts—quite the opposite. Patterns are typical solutions to common problems in object-oriented design. When a solution gets repeated over and over in various projects, someone eventually puts a name to it and describes the solution in detail. That’s basically how a pattern gets discovered.

The concept of patterns was first described by Christopher Alexander in A Pattern Language: Towns, Buildings, Construction. The book describes a “language” for designing the urban environment. The units of this language are patterns. They may describe how high windows should be, how many levels a building should have, how large green areas in a neighborhood are supposed to be, and so on.

The idea was picked up by four authors: Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm. In 1995, they published Design Patterns: Elements of Reusable Object-Oriented Software, in which they applied the concept of design patterns to programming. The book featured 23 patterns solving various problems of object-oriented design and became a best-seller very quickly. Due to its lengthy name, people started to call it “the book by the gang of four” which was soon shortened to simply “the GOF book”.

Since then, dozens of other object-oriented patterns have been discovered. The “pattern approach” became very popular in other programming fields, so lots of other patterns now exist outside of object-oriented design as well.

Design Pattern sind Ratschläge wie man bestimmte Probleme seines Codes in einer besseren Art und Weise lösen könnte.

**Why should I learn patterns?**

The truth is that you might manage to work as a programmer for many years without knowing about a single pattern. A lot of people do just that. Even in that case, though, you might be implementing some patterns without even knowing it. So why would you spend time learning them?

Design patterns are a toolkit of tried and tested solutions to common problems in software design. Even if you never encounter these problems, knowing patterns is still useful because it teaches you how to solve all sorts of problems using principles of object-oriented design.

Design patterns define a common language that you and your teammates can use to communicate more efficiently. You can say, “Oh, just use a Singleton for that,” and everyone will understand the idea behind your suggestion. No need to explain what a singleton is if you know the pattern and its name.

**Criticism of patterns**

It seems like only a lazy person haven’t criticized design patterns yet. Let’s take a look at the most typical arguments against using patterns. Kludges for a weak programming language

* Usually the need for patterns arises when people choose a programming language or a technology that lacks the necessary level of abstraction. In this case, patterns become a kludge that gives the language much-needed super-abilities.

For example, the Strategy pattern can be implemented with a simple anonymous (lambda) function in most modern programming languages.

* Inefficient solutions

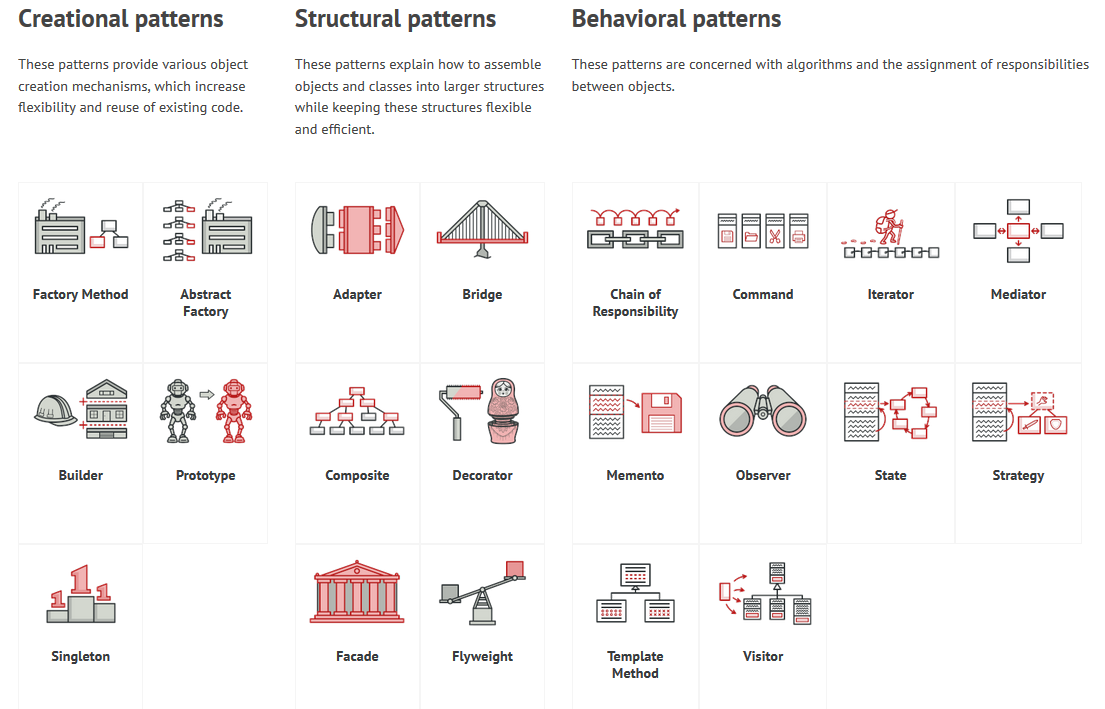
Patterns try to systematize approaches that are already widely used. This unification is viewed by many as a dogma and they implement patterns “to the point”, without adapting them to the context of their project.

* Unjustified use

If all you have is a hammer, everything looks like a nail.

This is the problem that haunts many novices who have just familiarized themselves with patterns. Having learned about patterns, they try to apply them everywhere, even in situations where simpler code would do just fine.

Man unterteilt sie in drei große Kategorien (Erzeugung, Verhalten, Struktur).



**Facade** **Abstrakte Fabrik** **Decorator** **State** **Strategie** **Iterator** **Command** **SOLID**

**Aufgabe Gruppenarbeit**

* Bilden Sie Gruppen zu jeweils … Personen
* Behandeln Sie eine der oben dargestellten DesignPattern anhand folgendem Schema.
  + **Definition**
  + **Beispiel Problem**
  + **Beispiel Lösung**
  + **Pro/Cons**
  + **Allgemeine Struktur**
    - Klassendiagramm + Erläuterung
  + **Real World Example in PHP**
    - keine Hunde, Katzen, Enten
    - evtl. Hinweise auf deren Verwendung in bekannten Frameworks
    - Quellcode und Darstellung in einem Klassendiagramm
    - kein designpattern.readthedocs.com
    - kein philliphauer.com
* **Präsentation der Ergebnisse**
  + Powerpoint oder Ähnliches
  + Quellcode