# MODERN DESIGN PATTERNS


TODAY: PROMISE

# Software design patterns

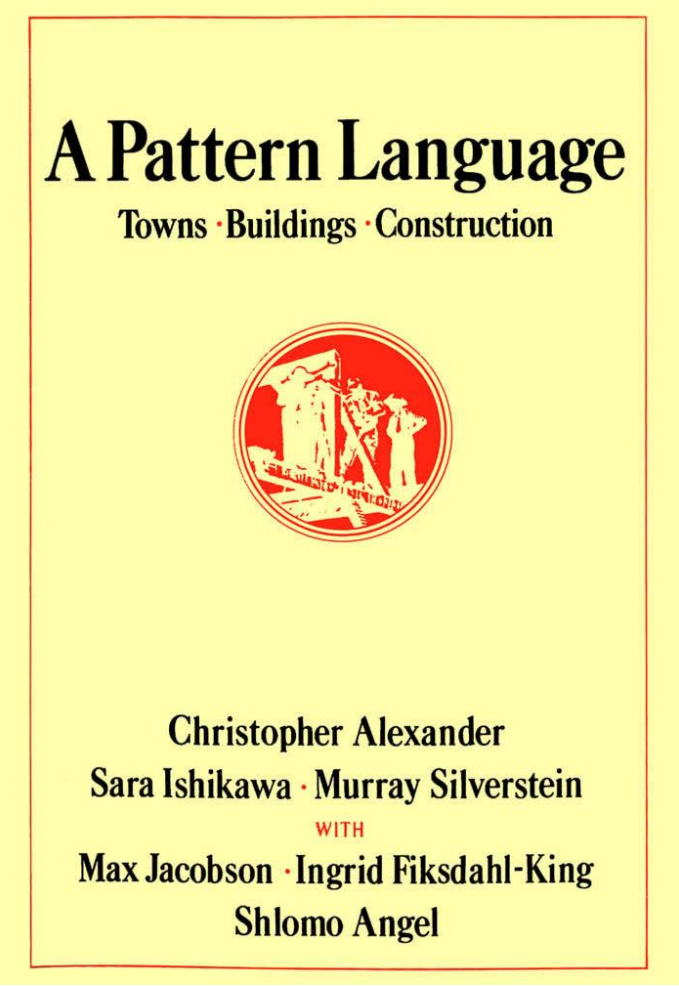
Introduction

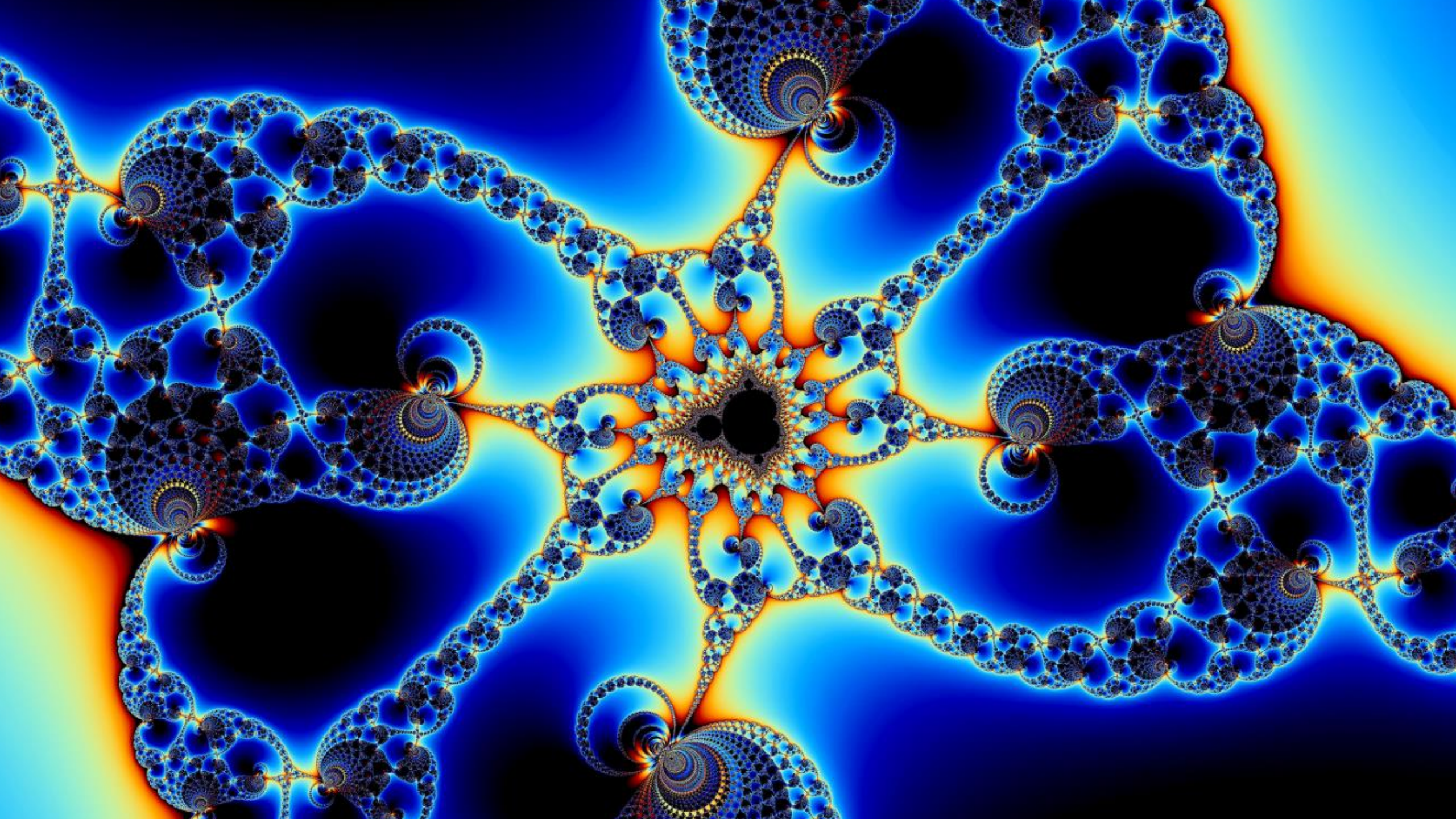
version: 1.0.1

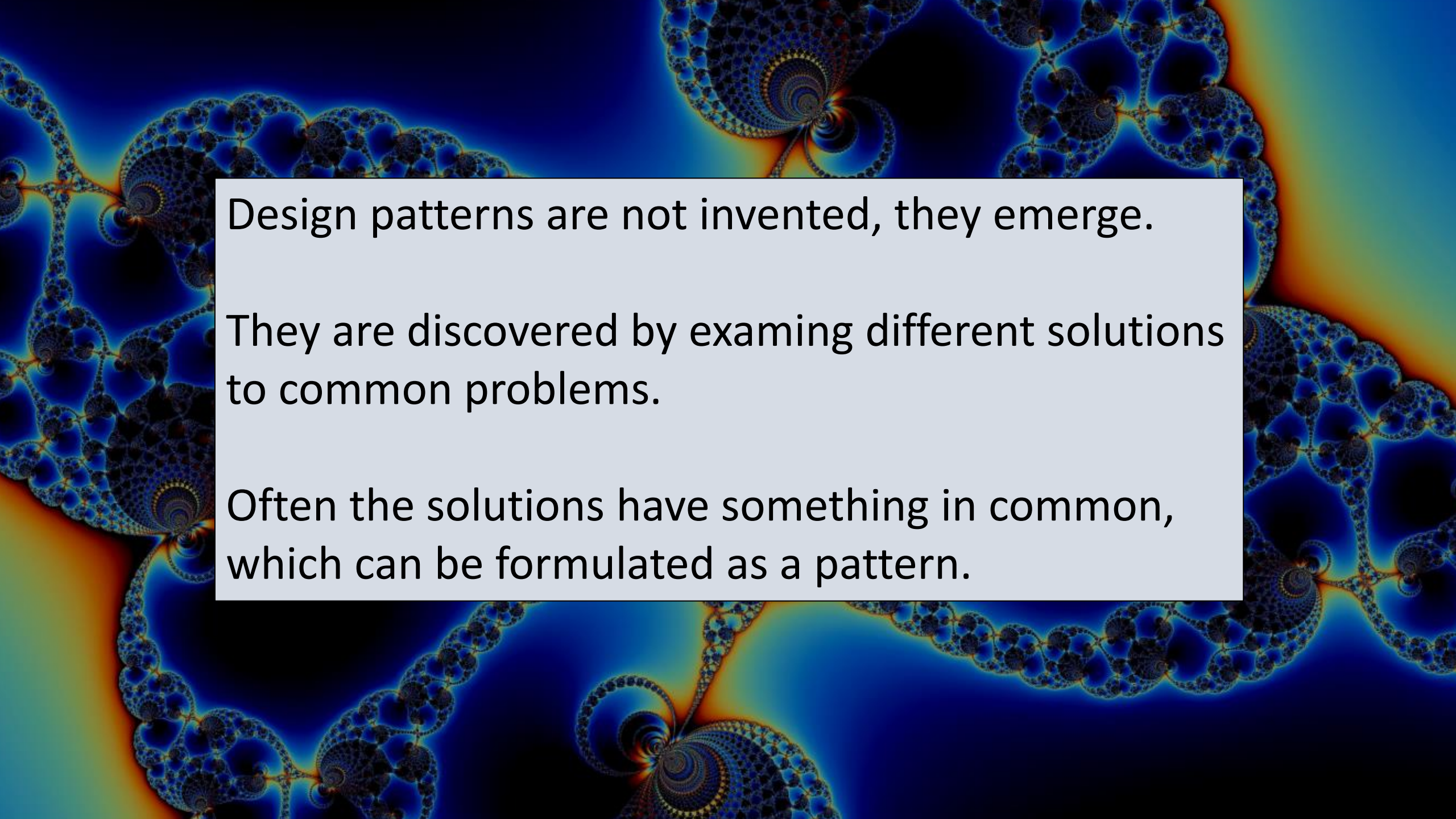
# What is a design pattern?

*"(…) Each pattern describes **a problem** that occurs over and over again in our environment, and then describes the **core of the solution to that problem**, in such a way that you can use this solution a million times over, **without ever doing it the same way twice**."*

*Christopher Alexander, architect*







Design patterns are not invented, they emerge.

They are discovered by examing different solutions to common problems.

Often the solutions have something in common, which can be formulated as a pattern.

# Software Design Patterns

Best practices for recurring problems

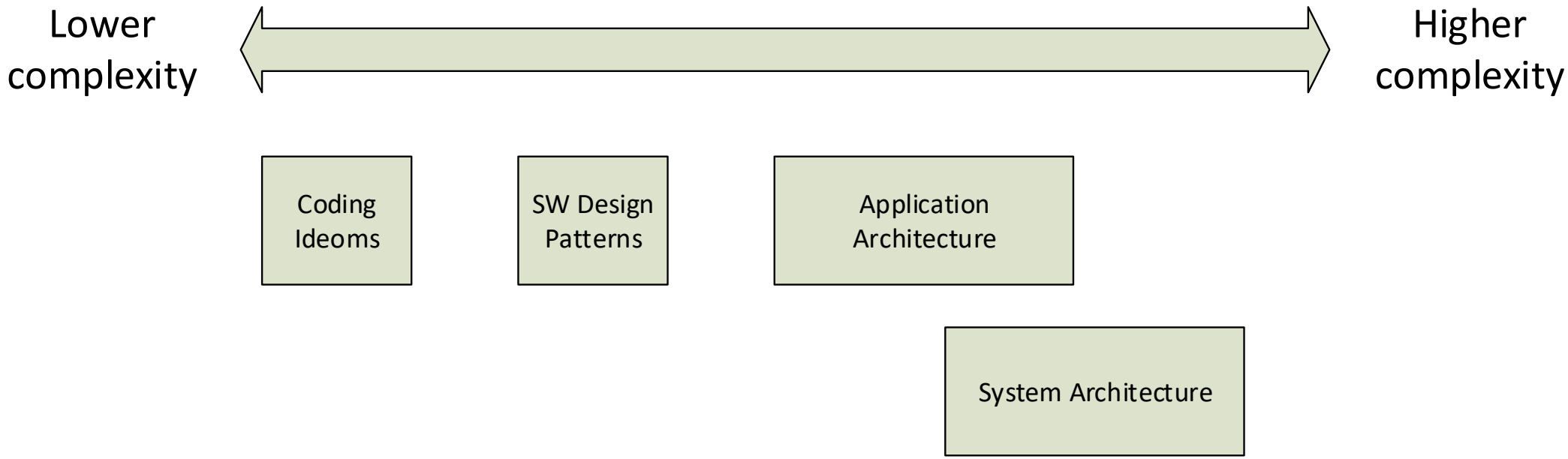
- General designs
- Must be customized to context

Terminology and *pattern language*

Learning opportunity

Maintainable software

# Design or architecture?



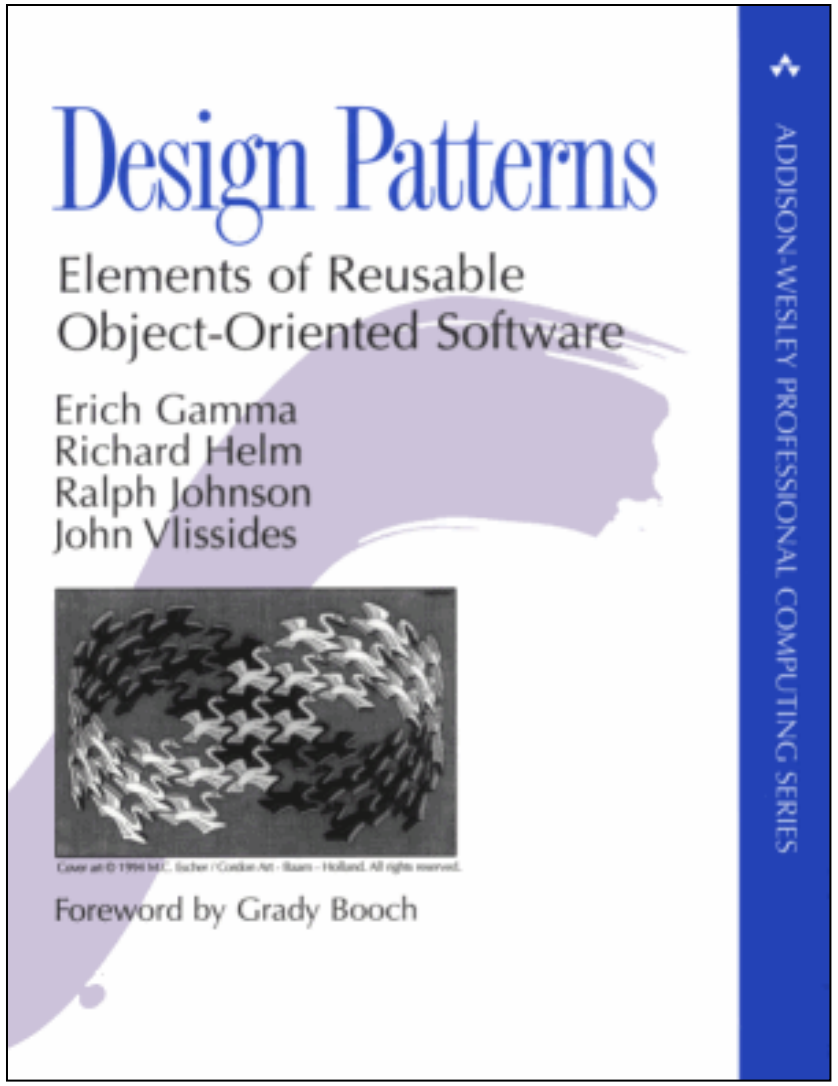
# Design patterns give us a vocabulary

Couldn't we just use *Strategy* and *Iterator* to solve that?

Apply *State* and *Command* together, and we're home free!

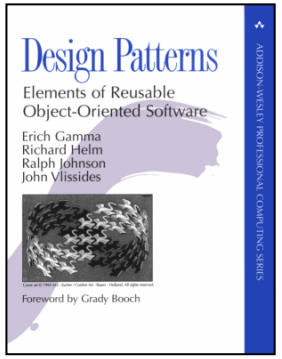
Should we build the server as a *Reactor*?

# Gang-of-Four Design Patterns



# Gang-of-Four Design Patterns

- *Creational* patterns
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- *Structural* patterns
  - Adaptive
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- *Behavioral* patterns
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template
  - Visitor




http://www.dofactory.com/Patterns/Patterns.aspx

# Why, and When to, use Design Patterns?

**S**ingle Responsibility Principle

**O**pen – Closed Principle

**L**isskov's Substitution Principle

**I**nterface Segregation Principle

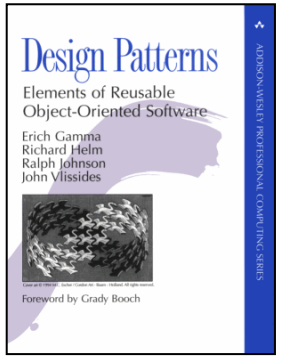
**D**ependency Inversion Principle

# Gang-of-Four Design Patterns

- *Creational* patterns
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton

- *Structural* patterns
  - Adaptive
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy

- *Behavioral* patterns
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template
  - Visitor




http://www.dofactory.com/Patterns/Patterns.aspx

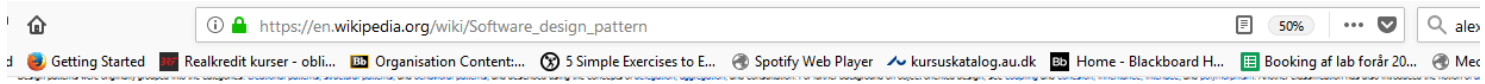
# GoF Pattern Description Template

| Name | Observer |
| --- | --- |
| Intent | |
| Also Known As | |
| Motivation | |
| Applicability | |
| Structure | |
| Participants | |
| Collaborations | |
| Consequences | |
| Implementation | |
| Sample code | |
| Known Uses | |
| Related Patterns | |

# Other patterns?

**Creational patterns**

| Name | Description | In Design Patterns | In Code Complete[15] | Other |
|---|---|---|---|---|
| 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 |
| Dependency Injection | A class accepts the objects it requires from an injector instead of creating the objects directly. | No | No | N/A |
| Factory method | Define an interface for creating a single object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses. | 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. This pattern appears in the GoF catalog as "virtual proxy", an implementation strategy for the Proxy pattern. | No | No | PoEAA[16] |
| Multiton | Ensure a class has only named instances, and provide a 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 from the 'skeleton' of an existing object, thus boosting performance and keeping memory footprints to a minimum. | Yes | No | N/A |
| Resource acquisition is initialization (RAII) | 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 |




### Creational patterns [edit]

| Name | Description | In *Design Patterns* | In *Code Complete*[12] | Other |
|---|---|---|---|---|
| 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 |
| Dependency injection | A class accepts the objects it requires from an injector instead of creating the objects directly. | No | No | N/A |
| Factory method | Define an interface for creating a single object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses. | 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. This pattern appears in the GoF catalog as "virtual proxy", an implementation strategy for the Proxy pattern. | Yes | No | PoEAA[14] |
| Multiton | Ensure a class has only named instances, and provide a 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 from the 'skeleton' of an existing object, thus boosting performance and keeping memory footprints to a minimum. | Yes | No | N/A |
| Resource acquisition is initialization (RAII) | 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 [edit]

| Name | Description | In *Design Patterns* | In *Code Complete*[12] | Other |
|---|---|---|---|---|
| Adapter, Wrapper, or Translator | Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces. The enterprise integration pattern equivalent is the translator. | Yes | Yes | N/A |
| 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 |
| Extension object | Adding functionality to a hierarchy without changing the hierarchy. | No | No | Agile Software Development, Principles, Patterns, and Practices[13] |
| 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 |
| Flyweight | Use sharing to support large numbers of similar objects efficiently. | Yes | No | N/A |
| Front controller | The pattern relates to the design of Web applications. It provides a centralized entry point for handling requests. | No | No | J2EE Patterns[15] PoEAA[17] |
| Marker | Empty interface to associate metadata with a class. | No | No | Effective Java[16] |
| Module | Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity. | No | No | N/A |
| Proxy | Provide a surrogate or placeholder for another object to control access to it. | Yes | No | N/A |
| Twin[18] | Twin allows modeling of multiple inheritance in programming languages that do not support this feature. | No | No | N/A |

### Behavioral patterns [edit]

| Name | Description | In *Design Patterns* | In *Code Complete*[12] | Other |
|---|---|---|---|---|
| Blackboard | Artificial intelligence pattern for combining disparate sources of data (see blackboard system) | 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 allowing for the parameterization of clients with different requests, and the queuing or logging of requests. It also allows for the support of undoable operations. | Yes | No | N/A |
| 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 allows their interaction to vary 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 in 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 algorithm without changing the algorithm's structure. | Yes | Yes | N/A |
| Visitor | Represent an operation to be performed on the elements of an object structure. Visitor lets a new operation be defined without changing the classes of the elements on which it operates. | Yes | No | N/A |

### Concurrency patterns [edit]

| Name | Description | In *POSA2*[20] | Other |
|---|---|---|---|
| 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[21] | No | N/A |
| Blockchain | Decentralized way to store data and agree on ways of processing it in a Merkle tree, optionally using digital signature for any individual contributions. | No | N/A |
| Compute kernel | The same calculation many times in parallel, differing by integer parameters used with non-branching pointer math into shared arrays, such as GPU-optimized Matrix multiplication or Convolutional neural network. | 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 locking logic proceed.<br>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.[22] | 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 |
| Join | Join-pattern provides a way to write concurrent, parallel and distributed programs by message passing. Compared to the use of threads and locks, this is a high-level programming model. | No | N/A |
| Lock | One thread puts a "lock" on a resource, preventing other threads from accessing or modifying it.[23] | No | PoEAA[14] |
| Messaging design pattern (MDP) | Allows the interchange of information (i.e. messages) between components and applications. | No | N/A |
| 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 |

https://en.wikipedia.org/wiki/Software_design_pattern

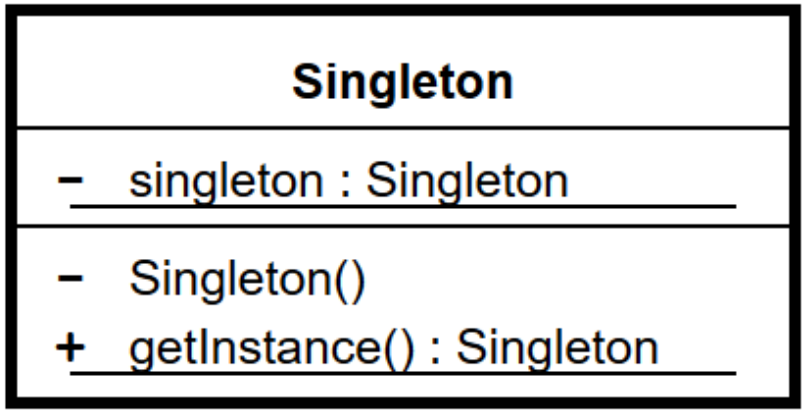
# Anti-patterns

**DevelopmentAntiPattern** (see also DevelopmentAntiPatternRoadMap and CategoryDevelopmentAntiPattern)

- AccidentalComplexity
- AccidentalInclusion
- AddingEpicycles
- AlcoholFueledDevelopment
- AmbiguousViewpoint
- AsynchronousUnitTesting
- BearTrap
- BigBallOfMud
- BoatAnchor
- CascadingDialogBoxesAntiPattern
- ContinuousObsolescence
- ControlFreak
- CrciCards
- CreepingFeaturitis
- CopyAndPasteProgramming
- DbClass
- DeadEnd

http://wiki.c2.com/?AntiPatternsCatalog

# Pattern or anti-pattern?




```
public sealed class Singleton
{
    private static volatile Singleton? _instance;
    private static readonly object _lock = new object();

    private Singleton() { }

    public static Singleton Instance
    {
        get
        {
            if (_instance != null)
            {
                return _instance;
            }

            lock (_lock)
            {
                if (_instance == null)
                    _instance = new Singleton();
            }

            return _instance;
        }
    }
}
```

AARHUS UNIVERSITY

# References and image sources

Mandelbrot image: https://github.com/palle-k/Mandelbrot