Introduction to Design Patterns

Mel Ó Cinnéide
School of Computer Science
University College Dublin

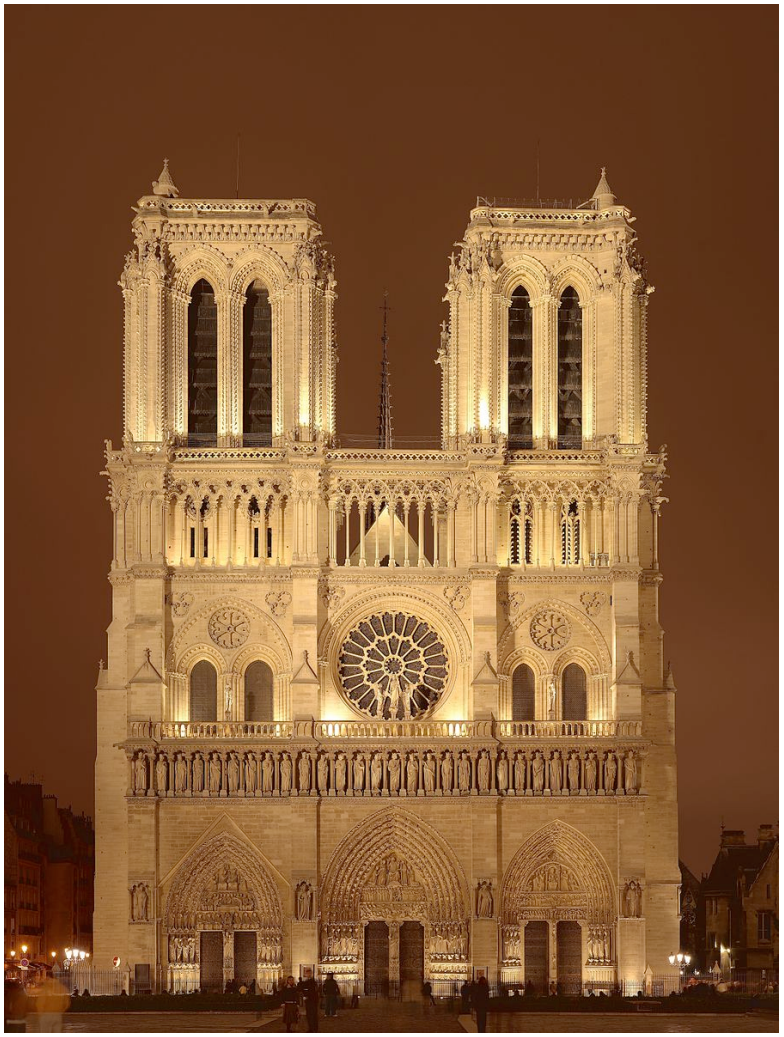
Roadmap of this Section

- Christopher Alexander's Pattern work
- Patterns in Software
- Pattern Form
- Intro to the "Gang of Four" Patterns
- Summary

Where Patterns Started

- In the 1960s, an architect called Christopher Alexander revisited some age-old questions:
  - What is good architecture?
  - Is beauty objective?
  - Or it simply subjective?

Beauty, you decide!



Where Patterns Started

- Alexander studied master solutions to the same architectural problems.
- The similarities he found, he termed **patterns**.
- Examples: Main entrance, alcove, ... corner store, ...

So, what's a pattern?

- A pattern is a **proven solution** to a **recurring problem** in a **certain context**.
- Or, in Alexander's own words:
  - "Each pattern describes a problem which 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."

A pattern is a **proven solution**...

- The solution isn't new, it's already been proven to be valid.
  - Usually many examples already exist
- A novel, brilliant idea is not a pattern.
- To someone who has mastered the field, a pattern should be nothing new.

E.g. the **Loop** pattern will contain little new for an experienced programmer

...to a **recurring problem**...

- The problem isn't a one-off.
- It's a problem that you encounter again and again (but a bit differently each time).

E.g. instances of the **Roundabout** pattern are similar, yet never quite the same.

...in a **certain context**.

- A pattern doesn’t solve a problem in a vacuum.
- The pattern has to fit into some **context**.
- The context varies, and this changes how the pattern is applied.

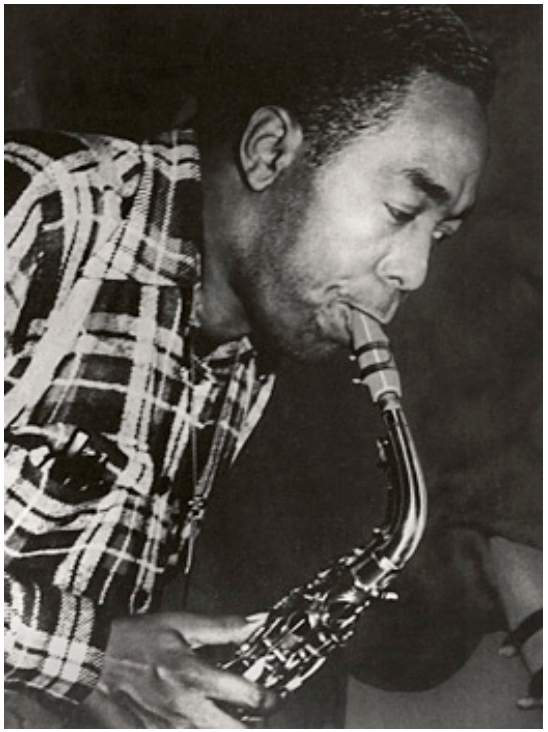
This "tailoring to context" is one of the challenges in using patterns.

Once mastered, patterns are forgotten

“Learn the patterns, then...

forget 'em.”

(“forget 'em” really means to *embody* them.)



Charlie Parker
Jazz musician

Alexander and Software

- Alexander had some following in the architecture community.
- However, his work was lionised by the software community in the 1990s.
- Patterns became, and have remained, an established part of software development.

Key properties of patterns

- Sometimes it is not clear what is and isn't a pattern. Some guidelines:
  - More than just a clever trick.
  - Patterns can't be applied automatically – they need to be tailored to fit their context. They are "**half-baked**".
  - A pattern should be THE solution, and provide you with a warm, fuzzy feeling (what patterns people call the Quality Without A Name, or QWAN)
  - Patterns are not invented: a pattern shouldn't look surprising to an expert in the domain.

Pattern Form

- There are various accepted forms with which to describe a pattern, but five essential parts of any form are:
  - Name
  - Problem
  - Context
  - Solution
  - Consequences
- We examine these in the following slides.

Pattern Form: Name

- Pattern name may seem like a trivial issue. However it has a significant impact:
  - Increases our design vocabulary
  - Enables discussion at higher level of abstraction
- Comment from an experienced programmer on reading the GoF book:
  - “The exciting part of patterns is the names”

Pattern Form: Problem

- Describes when to apply the pattern
  - Explains the type of problem the pattern solves.
- Could be a design problem
  - E.g. how to represent an algorithm as an object?
- Or an inflexible design
  - E.g. how can subclass explosion be ameliorated?

Pattern Form: Context

- No problem exists in a vacuum — there is always a surrounding context
- Context helps you determine where to use the pattern, and provides evidence that it is of general application
  - Also called the constraints or **forces**
- Usually there are some forces in conflict; that is why a pattern is required to resolve them.
  - E.g. optimising for performance and maintainability is usually impossible — these two forces tend to be in conflict.

Pattern Form: Solution

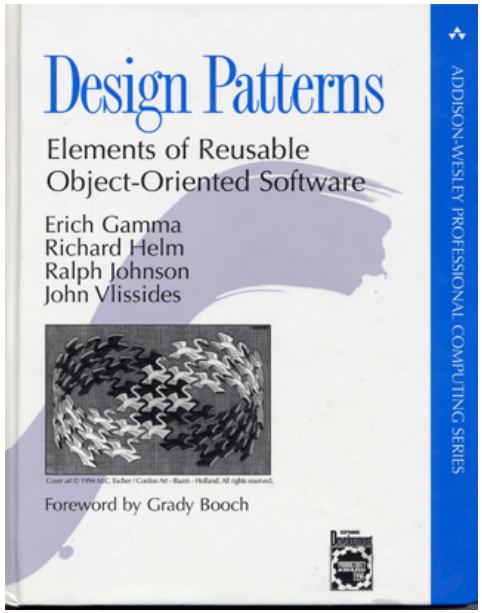
- This is a description of the elements of the solution design, their responsibilities, relationships and collaborations.
- It is not a specific implementation, but a general, tailorable solution.
- The solution should resolve the forces described in the Pattern Context in a “satisfying” way.

Pattern Form: Consequences

- These are the results and trade-offs of applying the pattern.
- Include the pros and cons of applying the pattern.
  - Design patterns usually increase flexibility
  - Complexity usually increases
  - Performance may decrease
- Understanding the consequences of using a design pattern is vital.
  - Comes close to the essence of software design

“Gang of Four” Patterns

- In the early 1990s software developers wondered if the patterns approach could be used in our discipline
  - Are there problems that occur over and over again that could be solved in essentially the same way?
- The so-called “Gang Of Four”, Gamma, Helm, Vlissides and Johnson produced one of the best selling books ever in software.



GoF Patterns

	Purpose		
	Creational	**Structural**	**Behavioral**
	Factory Method	Adapter (class)	Interpreter Template Method
	Abstract Factory Builder Prototype Singleton	Adapter(obj) Bridge Composite Decorator Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Flyweight Observer State Strategy Visitor

GoF Patterns

- Although the GoF design patterns are described separately, they are interrelated.
- They are not however intended to be complete – it is not a **pattern language**.

Summary

- We looked at various aspects of patterns and the patterns movement from Alexander to software.
- We will now look further at a few GoF design patterns.
- After excessive hype in the 1990s, patterns have simply become an established part of software development.