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?
  - Is 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 perhaps a bit differently each time).

...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.

Once mastered, patterns are forgotten

“Learn the patterns, then...

forget ’em.”

Charlie Parker
Jazz musician

Patterns in the Real World: **Adapter**

- An **adapter** takes input in one form and outputs it in another form.
- It doesn't transform the input fundamentally
- An obvious example is an electrical adapter, but...

Adapters abound!

Alexander and Software

- Alexander has some following in the architecture community.
- However, his work was lionised by the software community in the 1990s.
  - Several international pattern conferences sprang up
  - Alexander himself gave an invited keynote address at the world premiere OO conference (OOPSLA) in 1996.
- 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 blindly – 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 (the Quality Without A Name)
  - 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

- This is easy: what problem does the pattern solve?
  - What obstacle is in the way of producing an efficient/elegant/ powerful design?
  - Should help in determining where to apply a pattern

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 solve them.
  - For example, try optimising code for speed, memory usage and maintainability (3 forces in conflict).

Pattern Form: Solution

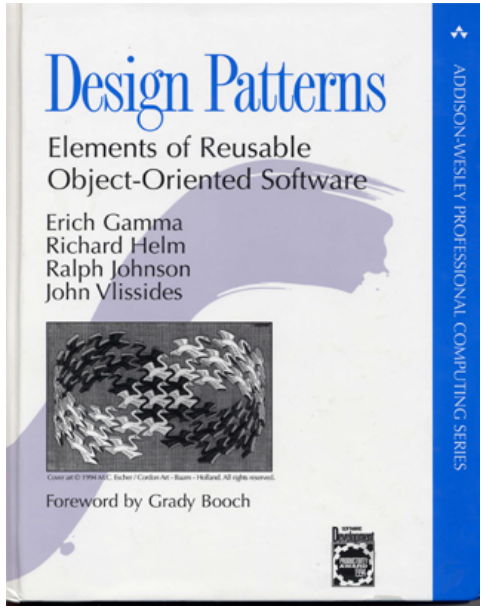
- This is a description of the elements of the solution design, their responsibilities, relationships and collaborations.
- It is not a concrete design or implementation, but a general, tailorable solution.
- The solution should resolve the forces described in the Pattern Context in a “satisfying” way.

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 design pattern application is vital.
  - This comes close to the essence of design

“Gang of Four” Patterns

- In the early 1990s software developers wondered if the same 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.
- Was best appreciated by experienced programmers.



Quantifying the effect of patterns in software

- Einstein's most famous paper: **17K** citations.
- Gosling's (creator of Java) most famous paper: **8K** citations.
- "Design Patterns" by the Gang of Four: **39K** citations.

- *P.S. Des Higgins of UCD Bioinformatics 'Clustal W' paper (most highly cited CS paper):* ***58K citations***

Source: Google Scholar, April 26th, 2018.

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.
- Earlier in the course we looked at two specific GoF design patterns: Observer and Strategy.
- After massive hype in the 1990s, patterns are now an established part of software development.