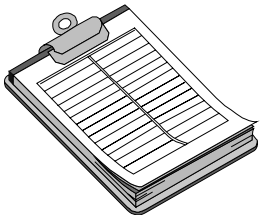
Beyond the Gang of Four

Kevlin Henney
Curbralan Limited
kevlin@curbralan.com

Frank Buschmann
Siemens AG
frank.buschmann@siemens.com

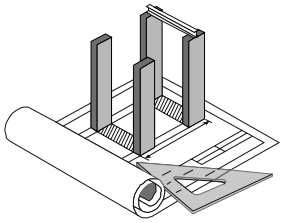
Agenda

- Objectives
  - Consider the pattern concept beyond the Gang of Four
- Contents
  - Introduction
  - Missing ingredients
  - Alternative solutions
  - Scope and granularity
  - Outroduction



Introduction

- Intent
  - Outline our goal in this talk
- Content
  - The Gang of Four
  - Misconceptions about the Gang of Four
  - Limitations in the Gang of Four



The Gang of Four

- *Design Patterns* has had a profound effect on what is considered to be design
  - Highly influential and a useful starting point
- But it has also restricted the expectations and design vocabulary of many programmers

> ***The enormous success of design patterns is a testimonial to the commonality seen by object programmers. The success of the book* Design Patterns*, however, has stifled any diversity in expressing these patterns.***
>
> **Kent Beck**

Misconceptions about GoF

- Misconceptions are about individual patterns as well as about the catalogue as a whole
  - The motivating example is the pattern
  - The diagram is the pattern
  - The source code is the pattern
  - Copy-and-paste solutions
  - There are only 23 patterns

> ***23 patterns you can cut and paste into your own design documents.***
> **Blurb on back cover of *Design Patterns CD***

Limitations in GoF

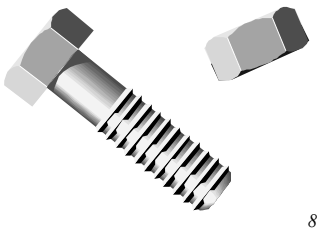
- Pattern structure
  - The role of applicability is unclear
  - Sometimes the example carries the weight of the pattern
- Catalogue structure
  - Everything related to a particular kind of problem or a solution structure is one pattern
  - The meaning and explicitness of connections
- The effect of time
  - The 23 patterns identified are not the ones we would necessarily consider the most important today

Into the Gang of Four

- Our aim is to pin down...
  - Missing ingredients in some existing GoF patterns, which makes the patterns richer and more complete
  - Alternative solutions that complement GoF patterns, that make our design vocabulary richer and more complete
  - The proper scope and granularity for many of the patterns, so that they do one thing well

Missing Ingredients

- Intent
  - Discover what is missing from some common patterns and complete them
- Content
  - History and hindsight
  - Factory Method and Disposal Method
  - Completing factories



History and Hindsight

- Experience is often responsible for letting us spot certain omissions
  - Whether omissions in individual patterns or from related groups of patterns
- An eye for cohesion let's us spot other omissions
  - What do we repeatedly include when applying a pattern, i.e. what additional patterns are there?

History rarely happens in the right order or at the right time, but the job of a historian is to make it appear as if it did.
James Burke

The Factory Method Pattern

- Factory Method is itself balanced and contained with respect to its goal
  - This is true in both the specific GoF view of Factory Method and also more generalised views of the pattern
- The question of completeness arises when a Factory Method is used as part of another pattern

Factory Method
Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

The Abstract Factory Pattern

- There is a meaningful and generative relationship between Abstract Factory and Factory Method
  - Scaling from the creation of individual objects to the creation of whole families of objects
- A concrete factory implements a number of related Factory Methods according to an interface

Abstract Factory
Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

The Disposal Method Pattern

- Factories can leave unresolved lifetime and resource management issues
  - Who cleans up factory products after use?
  - How can you reduce dynamic memory usage?
- A Disposal Method can provide the answer

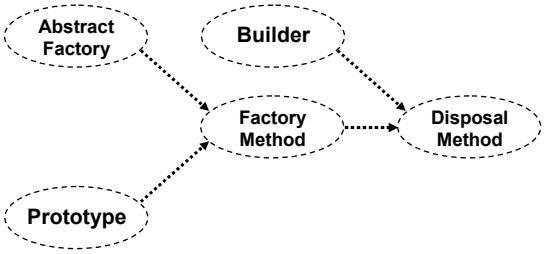
Disposal Method
Encapsulate the concrete details of object disposal by providing an explicit method for clean up instead of letting object users either abandon objects to the tender mercies of the garbage collector or terminate them with extreme prejudice and delete.

Linking Factory and Disposal

- Factory Method and Disposal Method are complementary with respect to symmetry
  - One completes the other by addressing unresolved questions in the lifecycle of an object
- Together they form a generative phrase
  - Where one pattern strongly suggests the application of the next as part of its structure

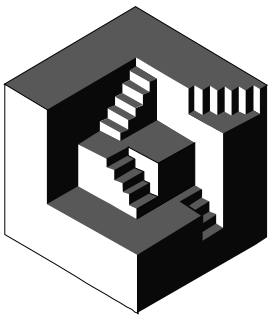
Completing Other Factories

- Consideration of disposal becomes a natural consequence of considering creation
  - Any creational pattern becomes more complete and more useful by raising the question of disposal



Alternative Solutions

- Intent
  - Identify the complements of some common patterns
- Content
  - A one-way system?
  - Iterator
  - Batch Method
  - Enumeration Method



A One-Way System?

- With the exception of the creational patterns, each problem type has only one GoF pattern against it
  - Some people see this as a strength, reducing their design options rather than creating a design dialogue
  - Gives the impression that there is only one solution per general problem, and the one in the book is that one
- Good design involves making informed choices
  - Patterns should communicate those choices
  - Arbitrarily reducing the available degrees of freedom at the expense of suitability is not a sound approach

The Iterator Pattern

- Classically defined in terms of separating the responsibility for iteration from its target
  - The knowledge for iteration is encapsulated in a separate object
  - Rendered idiomatically in different languages, e.g. STL in C++ and (twice) in Java's core *java.util* package

Iterator
Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

The Batch Method Pattern

- Iterated simple methods use up bandwidth
  - Spending far more time in communication than in computation
- Provide the repetition in a data structure
  - More appropriate granularity that reduces communication and synchronisation costs

Batch Method
Group multiple collection accesses together to reduce the cost of multiple individual accesses in a distributed environment.

The Enumeration Method Pattern

- An inversion of the Iterator design
  - Iteration is encapsulated within the collection
  - Collection stateless with respect to iteration
- A collection method receives a Command that it applies to its elements
  - Idiomatic iteration in Smalltalk based on block objects

Enumeration Method
Support encapsulated iteration over a collection by placing responsibility for iteration in a method on the collection. The method takes a Command object that is applied to the elements of the collection.

Scope and Granularity

- Intent
  - Nail down the proper scope and granularity for some common patterns, so that they do one thing well
- Content
  - Overworked and underpaid
  - Bridge
  - Many problems and solutions
  - Iterator reiterated

Overworked and Underpaid

- Some patterns just seem to be too busy solving everyone's problems to be cohesive
  - Adapter is expressed as two fundamentally different solutions... so isn't that two patterns?
  - Bridge seems to span more water than the Øresund, addressing multiple problems with multiple solutions
  - Must Iterator necessarily be the solution to all iteration designs?
  - Objects for States covers intent but is short on detail
  - And what is the Flyweight pattern?

Problem–Solution Multiplicity

- Some common patterns seem to identify themselves with the solution structure
  - And therefore do not have clear problem requirements
- Other common patterns seem to identify with the general theme of the problem
  - And therefore do not have distinct solutions
- Many common patterns appear to hide or act as the gateway to a small community of patterns
  - Inspection reveals another level of design consideration

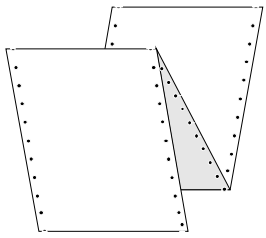
The Bridge Pattern

- The metaphor of a bridge between abstraction and internal implementation is compelling
  - But the motivation seems to be disconnected from many of the uses and the discussion of consequences
- Bridge appears to be many things to many people
  - As with other things in life, this is not a good thing

Bridge
Decouple an abstraction from its implementation so that the two can vary independently.

The Problems Listed

- The original list plus recent additions...
  - Permanence of implementation binding
  - Variation of implementation and abstraction
  - Reducing compilation ripple effect
  - Data hiding in C++
  - Elimination of multiple inheritance
  - Reduction of stack footprint
  - Representation sharing
  - Exception safety in C++



The Solutions Offered

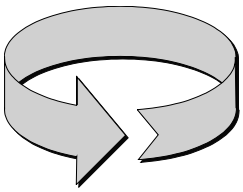
- The stable theme is that there is a handle–body separation introduced into the structure
  - But handle–body on its own is structure without intent
- The variations, however, are less easily fixed...
  - No class hierarchies, abstraction class hierarchy, implementation class hierarchy, or hierarchies for both
  - The abstraction is either a handle or a proper entity
  - No sharing of representation or sharing, optionally with reference counting
  - Optionally hide the representation class definition

Different Problem or Solution?

- This seems like a number of different patterns that are largely unrelated
  - Their only common feature is a handle–body separation... which is true of most GoF patterns
- Diversity in pattern expression is good... but there is such a thing as too much of a good thing...
  - The pattern becomes all things to all people, each person with their own different (incompatible) view
  - Dilutes the notion of patterns as design vocabulary

Iterator Reiterated

- GoF tries to accommodate both C++-like and Smalltalk-style iteration in one pattern
  - The external and internal variants of Iterator
  - However, most developers identify Iterator with External Iterator
- The problem solved, the solution structure and the consequences are totally different
  - I.e. two different patterns: Iterator and Enumeration Method



Outroduction

- Intent
  - Review our position, draw some conclusions and then, in the spirit of the talk title, move on
- Content
  - Twenty-three patterns in review
  - Pattern communities
  - Pattern languages

Twenty-Three Patterns in Review

- Yes, there is good design thinking in Gang of Four
  - But it gives insufficient coverage of the practices that are involved in successful real-world OO design
- Yes, the naming of individual practices is useful
  - But scope and applicability is often unclear or too broad
- Yes, the catalogue was a good start
  - But a starting point is just that: it is only one point on the whole line of development

Pattern Communities

- Patterns can be used in isolation with some degree of success
  - Represent foci for discussion or point solutions
- However, patterns are in truth gregarious
  - They are rather fond of the company of other patterns
  - To make practical sense as a design idea, patterns inevitably enlist other patterns for expression and variation

Tales of Software Design

- A pattern is a study in design and a fragment of design vocabulary
- It tells a "successful software engineering story"
  - Albeit a short one
- A pattern language is a framework that expands the range and scope of the stories than can be told

In Closing

- Patterns distil successful design experience
  - This is the message; it should not be confused with or restricted to the medium
- There is more to both OO design and patterns than is achieved or intended by the Gang of Four
  - It was a good start but it is a historical subset of what we need to know

> ***There are more things in heaven and earth, Horatio, than are dreamt of in your philosophy.***
>
> **William Shakespeare**