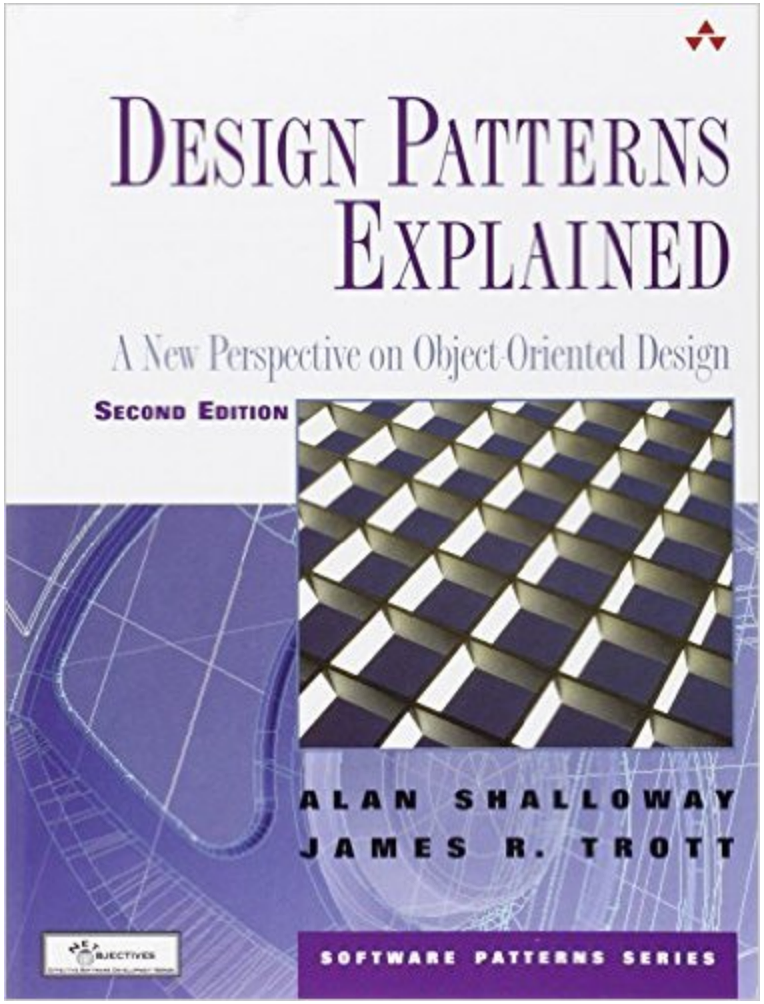
Software Design and Architecture

Pattern Focus Design



DESIGN PATTERNS EXPLAINED

A New Perspective on Object-Oriented Design

SECOND EDITION

ALAN SHALLOWAY

JAMES R. TROTT

SOFTWARE PATTERNS SERIES

Class Focus Design

Design is to build by fitting things together: “build from pieces”

- Functional decomposition
  - decompose the problem into small pieces and then build up from there

Pattern Focus Approach

- **Design** is often thought of as **a process of synthesis**, a process of putting together things, a process of combination.
- According to this view, a whole is created by putting together parts.
  - The parts come first: and the form of the whole comes second.

Design for Everyone

The interesting thing is that design is something that can be learned by anyone

- A design that follows well-established patterns will produce good, solid results.

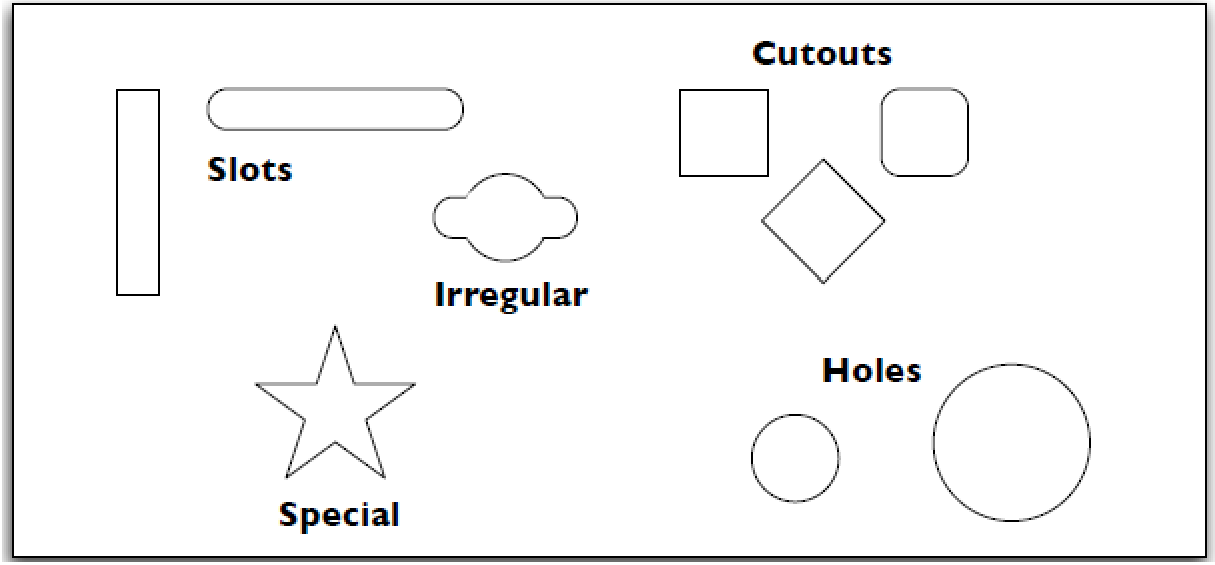
- Quality solutions for similar problems appear very much alike
  - For instance, following Model-View-Controller brings benefits to even the most novice designers…

Example

Design a system that aids a geologist in assigning ages to rock samples collected from the field

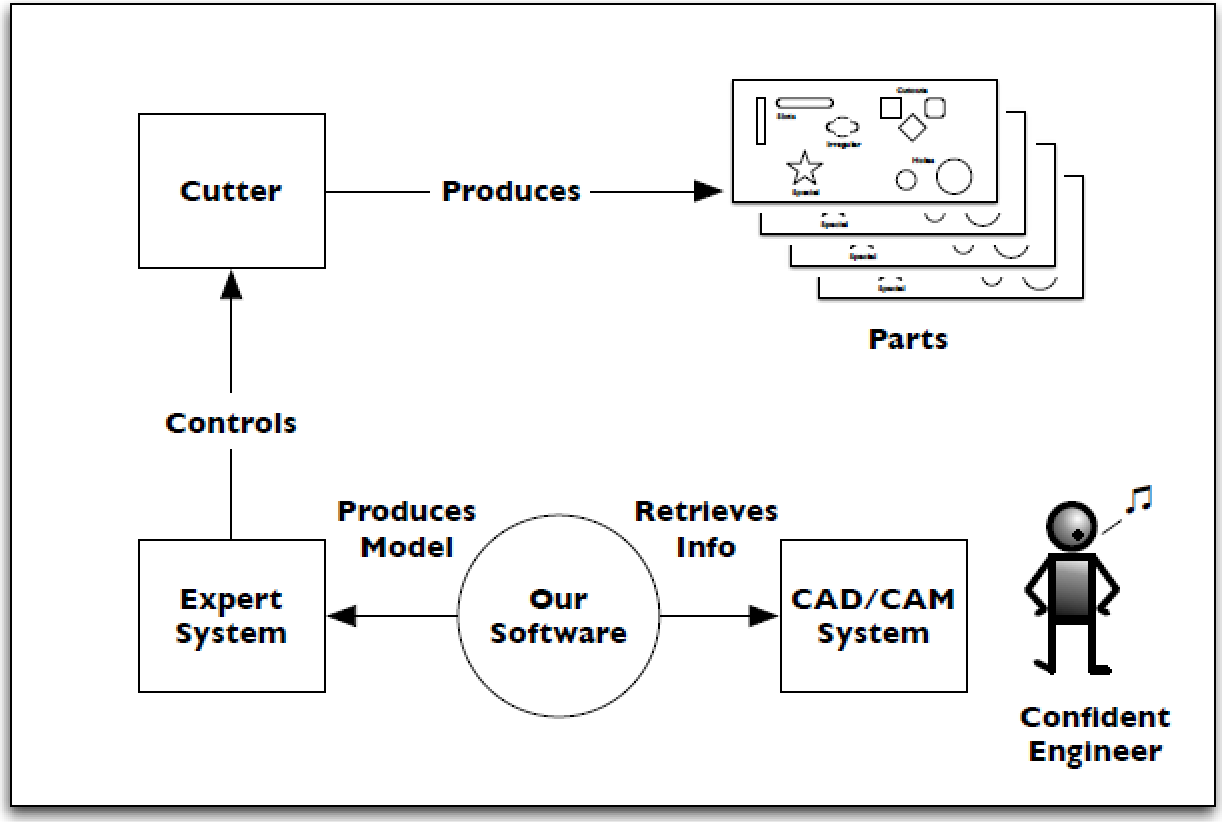
- Context Pattern: Desktop Application
  - Leads to: Model-View-Controller
    - Model Leads to: Database of Rock Samples
    - View Leads to: Collection Browser and Operations
    - Controller: Set of “glue” objects that invoke operations on selected samples, updates database, displays results

Review of CAD/CAM Problem



Design software that translates CAD designs that use the parts above into instructions for a machine that punches the actual part out of sheet metal

System Overview



Cutter

Produces

Parts

Controls

Expert System

Produces Model

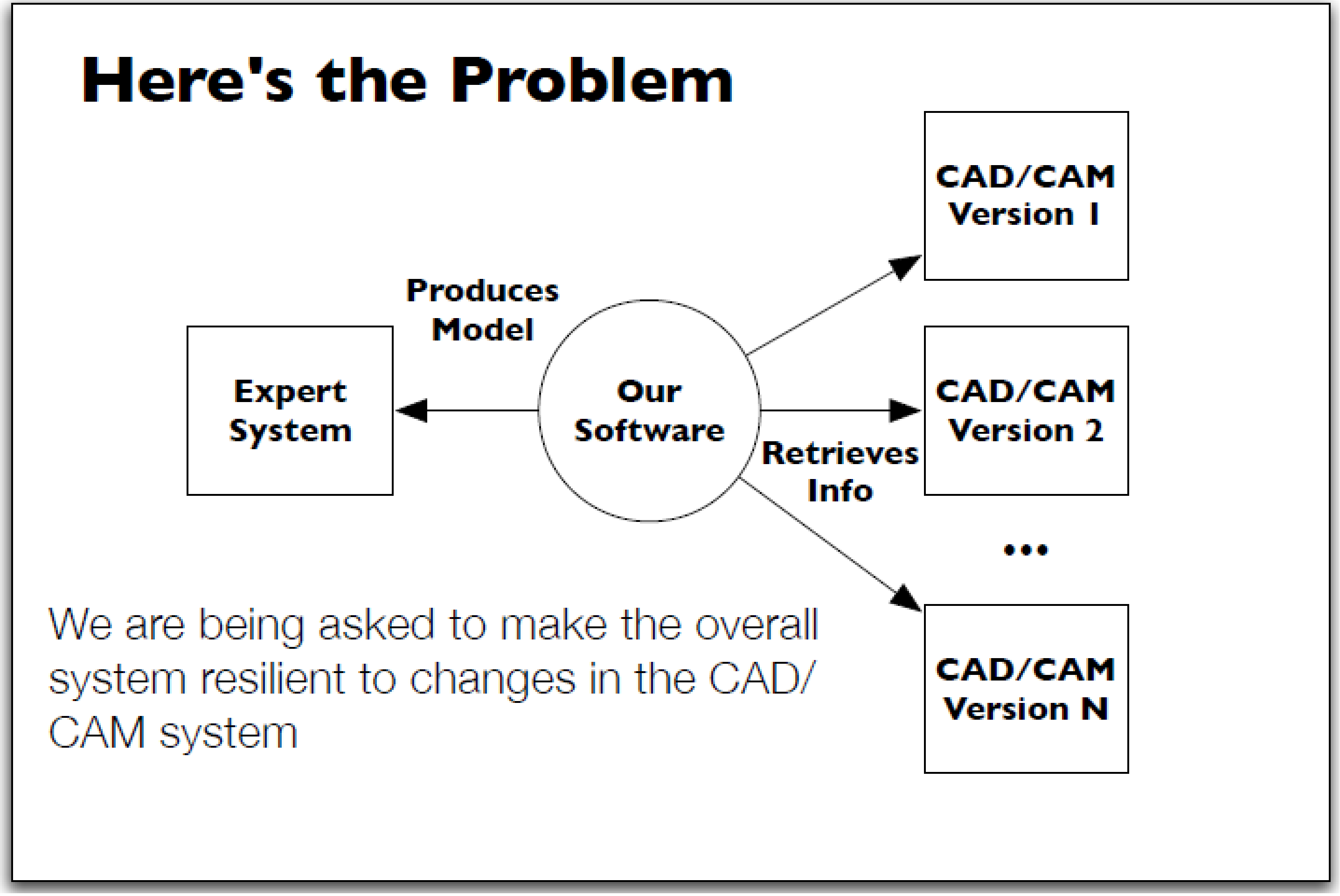
Our Software

Retrieves Info

CAD/CAM System

Confident Engineer

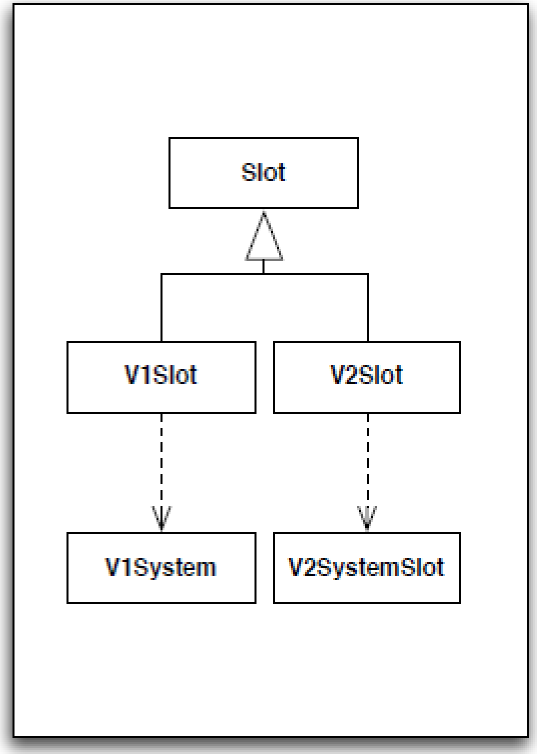
Example of encapsulation via software architecture…



Here's the Problem

We are being asked to make the overall system resilient to changes in the CAD/CAM system

The First Solution

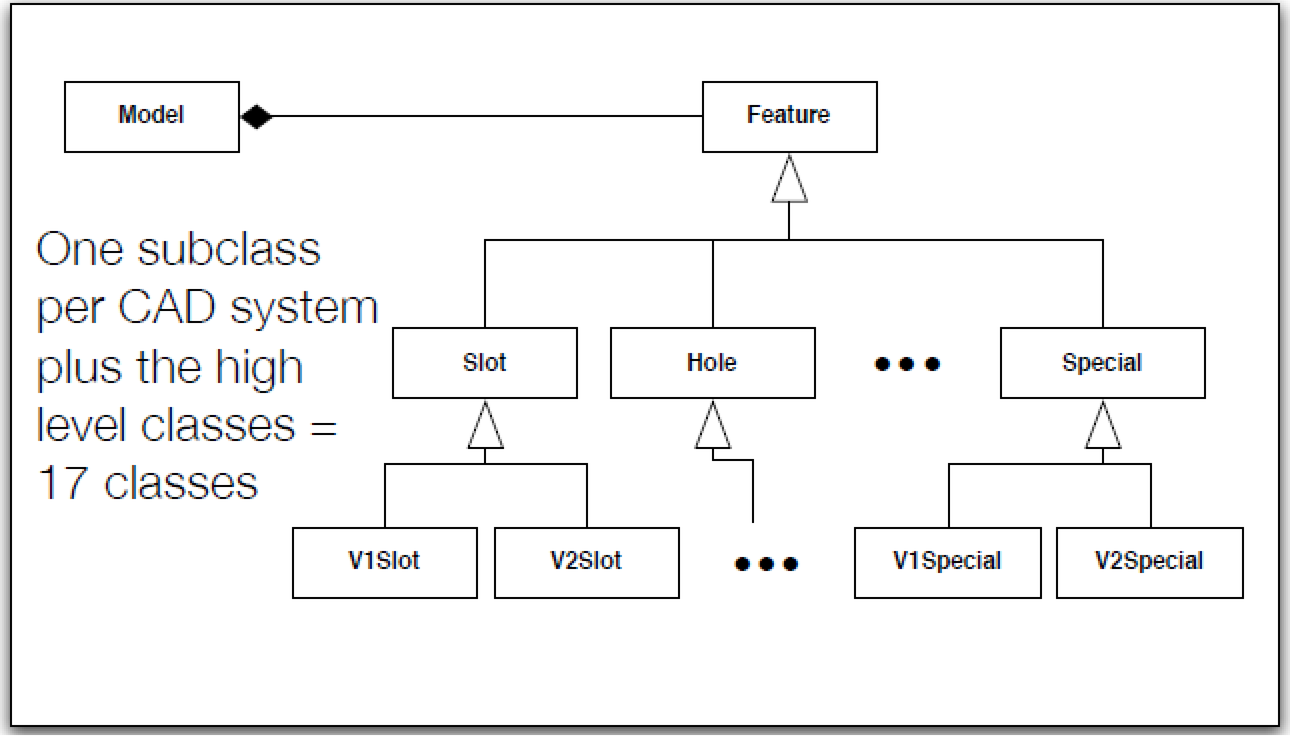


For each Feature class, the version 1 variation will have attributes that link to the version 1 model id and the feature id; it will then call the V1 library routines directly

The version 2 variation will simply wrap the Feature class that comes from the CAD system

The arrow with dashed line means "uses"

The First Solution



One subclass per CAD system plus the high level classes = 17 classes

Pattern Focus

To develop a better solution to this problem, let’s think in terms of patterns

Thinking in Patterns (big picture view)

Step 1: Identify the Patterns

Step 2: Analyze and apply the patterns

- 2a. Order the patterns by context creation
- 2b. Select pattern and expand design
- 2c. Identify additional patterns, add them to the set
- 2d. Repeat

Step 3: Add detail

Step 1: Identify the Patterns

For the CAD/CAM Domain, the possible patterns are

- **Abstract Factory:** Create parts for a particular CAD system
- **Adapter:** Adapt new CAD systems to the target interface
- **Bridge:** Implement the abstractions of the domain by "bridging" to a particular CAD system
- **Facade:** keep the complexities of the CAD system hidden from the expert system

Step 2a: Which pattern provides context for the others?

Look through all possible pairings of the identified patterns

- Does x provide context for y?
- Does abstract factory provide a context for bridge?
  - Look back at our Pizza shop example for inspiration
- To help with these decisions look at the patterns conceptually…

Step 2a: Which pattern provides context for the others?

- Abstract factory creates sets of related objects
- Adapter adapts existing class A to the interface needed by a client class B
- Bridge allows for different implementations to be used by a set of related client objects
- Facade simplifies an existing system A for a client class B

Step 2a: Which pattern provides context for the others?

- Abstract factory’s context is the structure of the objects its creating
  - Pizza is made of dough, sauce, toppings, etc.
- It does not provide context for other patterns
  - This is true of most “creational patterns”
  - So, scratch it off the list
- This leaves
  - Adapter <=> Bridge; Bridge <=> Facade; Facade <=> Adapter

Step 2a: Which pattern provides context for the others?

- Bridge <=> Adapter
  - Adapter will allow the expert system to access the OO interface of the new CAD system by making it conform to Feature and its subclasses
  - Bridge will ensure that Feature and its subclasses can access version 1 and 2 of the CAD system

Bridge provides context for Adapter

Step 2a: Which pattern provides context for the others?

- Bridge <=> Facade
  - Facade will simplify the complex interface of the first CAD system
  - Bridge will ensure that Feature and its subclasses can access version 1 and 2 of the CAD system
    - which means making use of the Facade

Bridge provides context for Facade

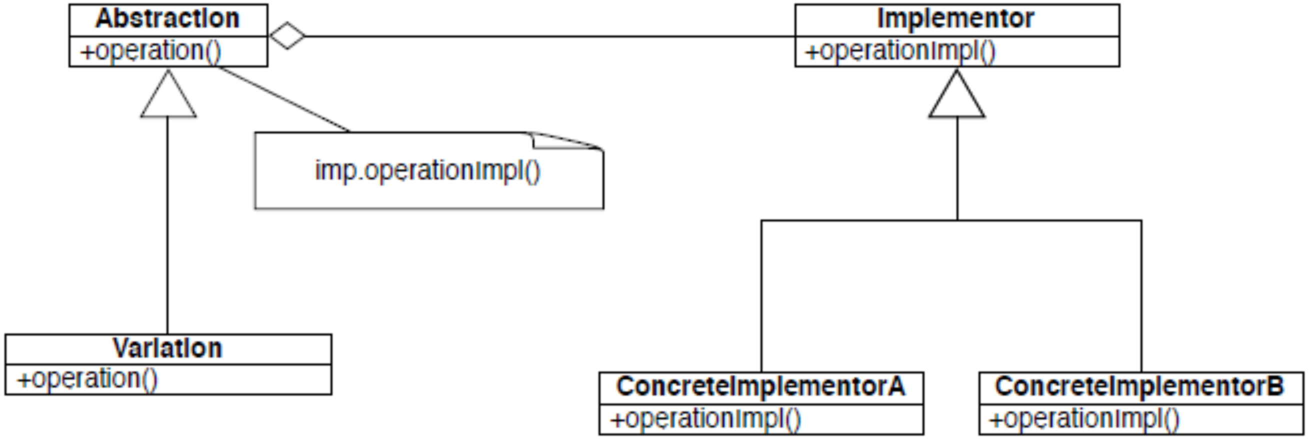
  - Since Bridge “wins” twice, its the outermost pattern

Step 2b: Select Pattern and Expand Design

How does Bridge fit into the conceptual whole of the design?

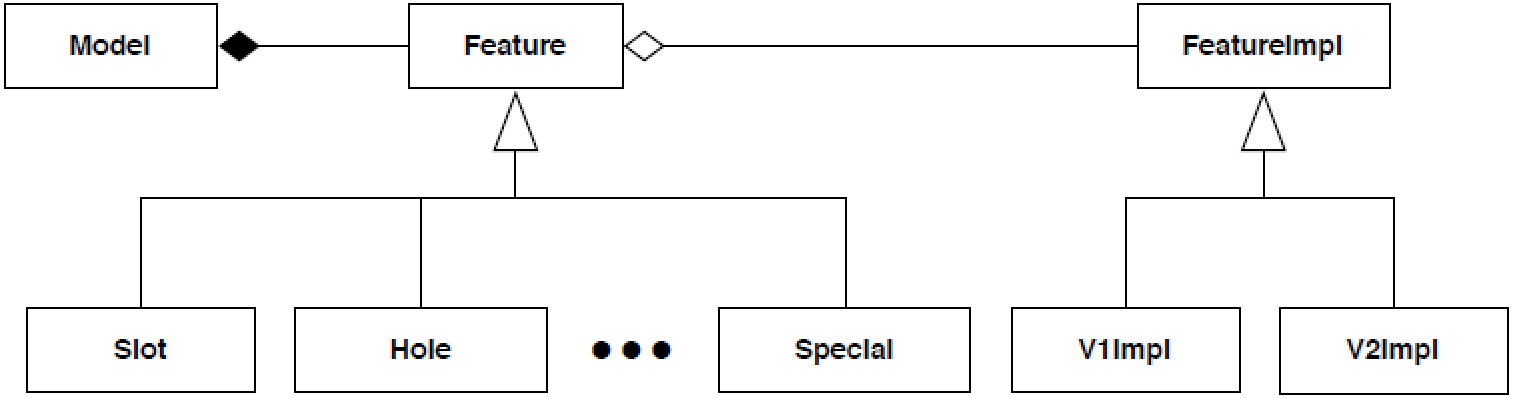
- What, exactly, provides a context for the Bridge pattern?
  - The elements of the problem domain!
    - Expert System uses Model
    - Model aggregates Features (abstractions)
    - Different CAD systems provide different types of features (implementations)
    - The Bridge pattern!!

Bridge Structure Diagram



Bridge in Context

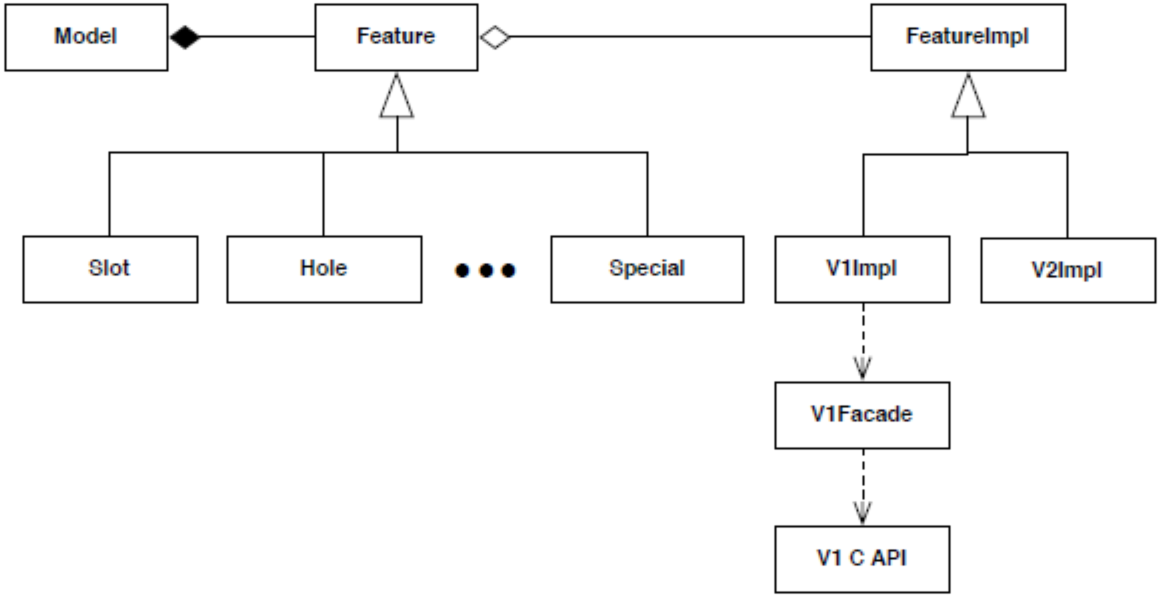
Assumes that Feature has a public interface that provides all of the information needed by the expert system



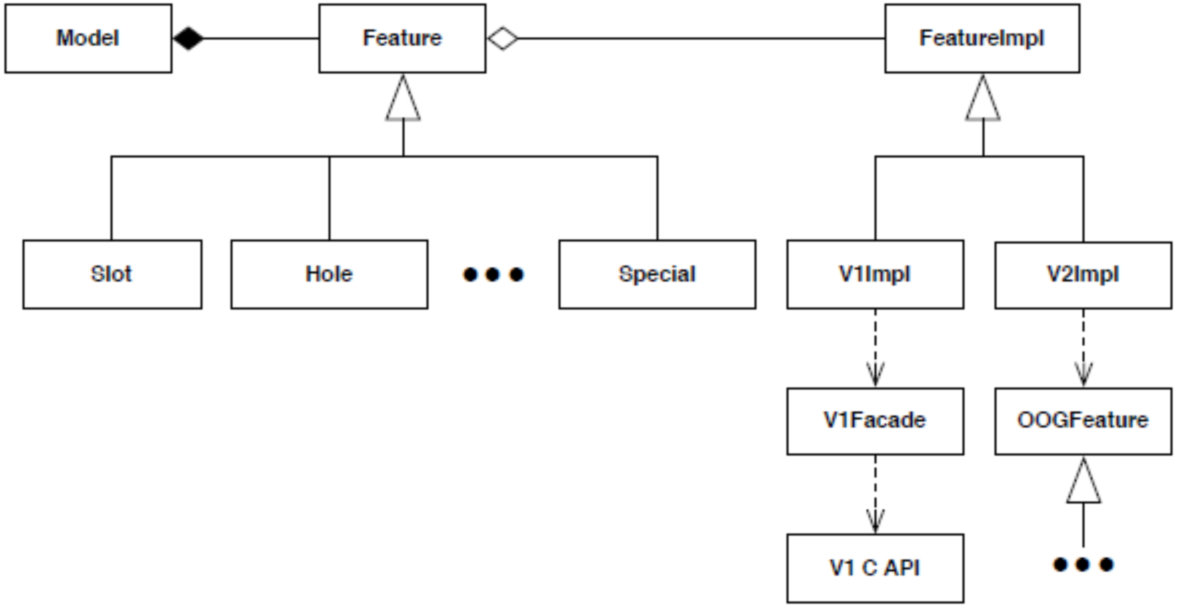
2c: Identify additional patterns

- All that is left in this particular system is to attach the V1 and V2 systems to the design
  - Adapter and Facade will do that for us, so no additional patterns are needed
- Looping back, we know that Adapter and Facade are independent of each other in this design
  - They can be applied in any order

Context for Facade



Context for Adapter (& Final Design)



Step 3: Add Detail

- At this point, we would start to add detail
- What exactly is the public interface of Feature and FeatureImpl
- How will each subclass of Feature implement that public interface by calling operations on FeatureImpl?

Is it better?

Is the new design better?

- The new design sounds simpler (especially because it can be explained using design patterns)

Now consider, what happens when V3 of the CAD system comes along…

- 6 new subclasses in 1st design; 2 new classes in the 2nd

Class Focus vs. Pattern Focus

In the first design, we got to a state that works but it wasn’t that maintainable

- it had a class-based focus that stuck parts together from the bottom up, creating a whole

In the second design, we started with the big picture, found the most suitable pattern and worked down, adding patterns that worked with the first one

- the patterns then deliver on good software qualities because that’s what they are all about!