Problem-Solution

CSCI 4448/5448: Object-Oriented Analysis & Design

Lecture 12a

Acknowledgement & Materials Copyright

- I'd like to start by acknowledging Dr. Ken Anderson
- Ken is a Professor and the Chair of the Department of Computer Science
- Ken taught OOAD on several occasions, and has graciously allowed me to use his copyrighted material for this instance of the class
- Although I will modify the materials to update and personalize this class, the original materials this class is based on are all copyrighted © Kenneth M. Anderson; the materials are used with his consent; and this use in no way challenges his copyright

Before we start: Project 2

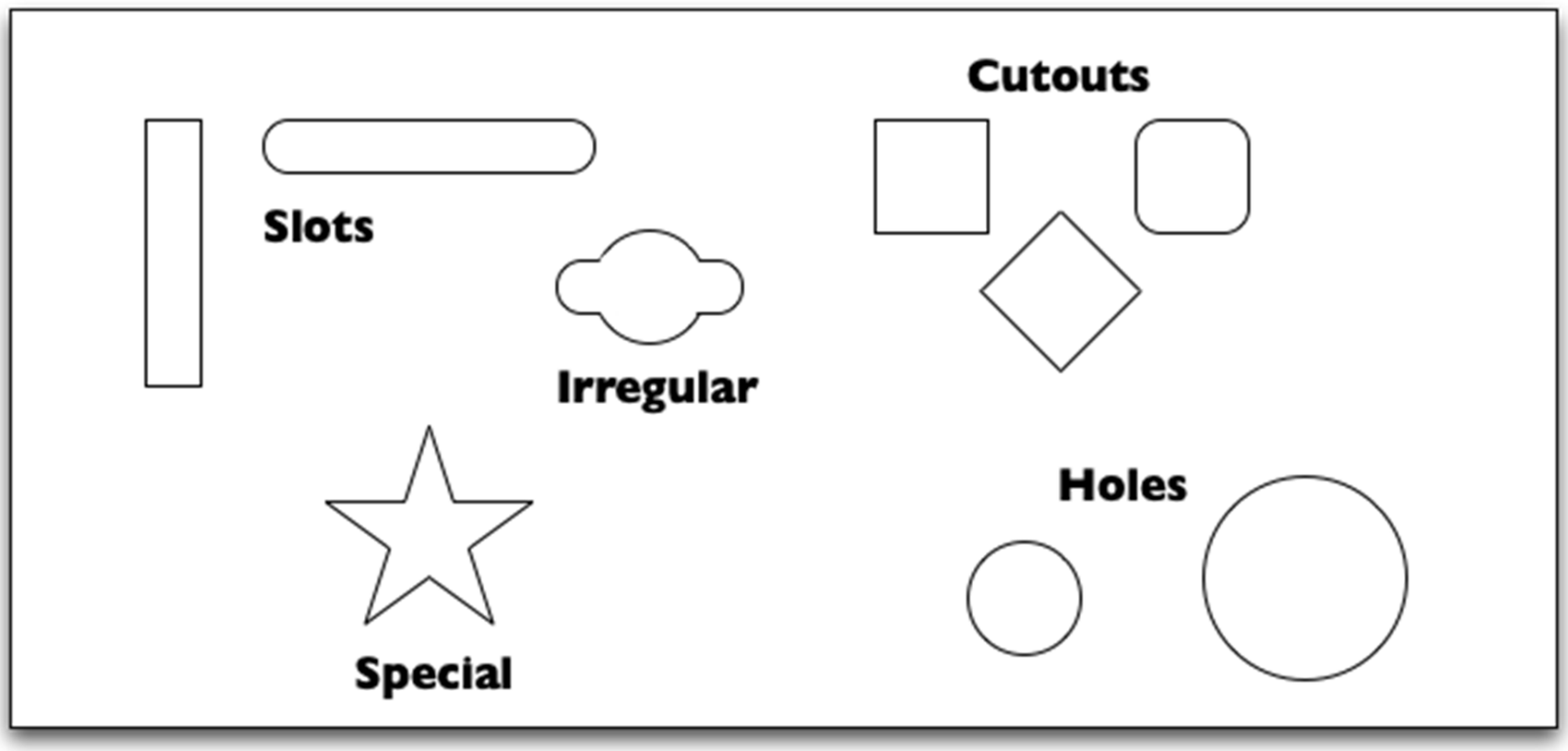
- The superclass/subclass Game inheritance structure isn't really needed for Project 2, but I asked for it in preparation for Projects 3 & 4, where the Game properties will start to become more unique
- Likewise, Employee/Cashier would make a good superclass/subclass inheritance structure, as we will soon see different types of employees in Projects 3 and 4
- If you didn't do these, you can always refactor in Project 3
- Things we'll be looking for in particular
  - No functional code in main() besides object instantiation
  - Comments in code that illustrate the six OO principles asked for
  - Clear 30-day output runs with the required summary information in Output.txt
  - An updated UML diagram that shows what changed in your code design since you turned in the first UML diagram, along with your comments

Goals of the Lecture

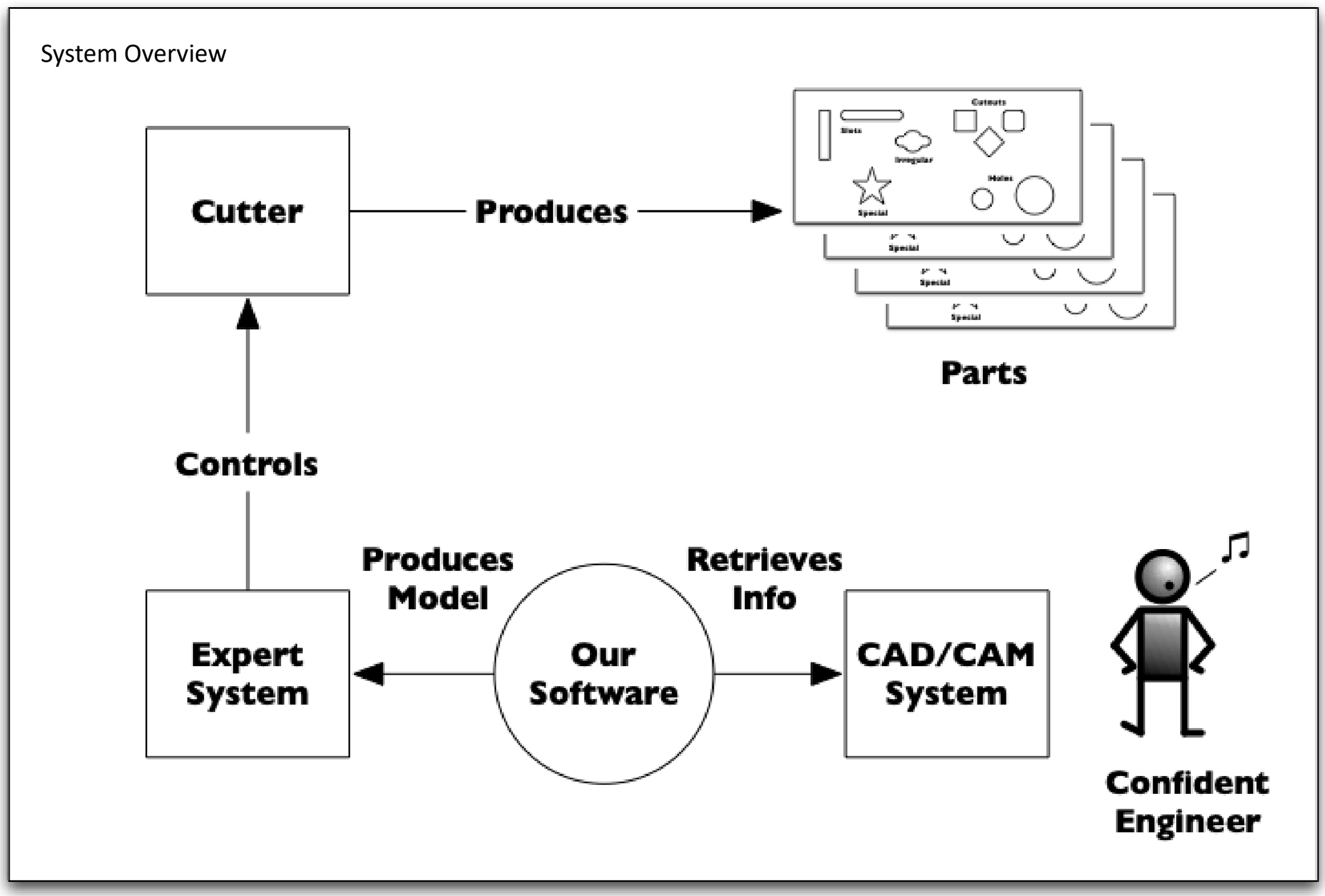
- Introduce and reflect on a design problem example
- Present an initial design to the problem domain
  - Highlight its strengths (if any) and weaknesses
- This example is from the Shalloway/Trott previous textbook…

The Problem Domain

- A company provides software that
  - allows engineers to create models for parts made out of sheet metal
  - generates the instructions needed by a computer-controlled cutting tool to actually make the part specified by the models
- Example part with five “feature” types



System Overview



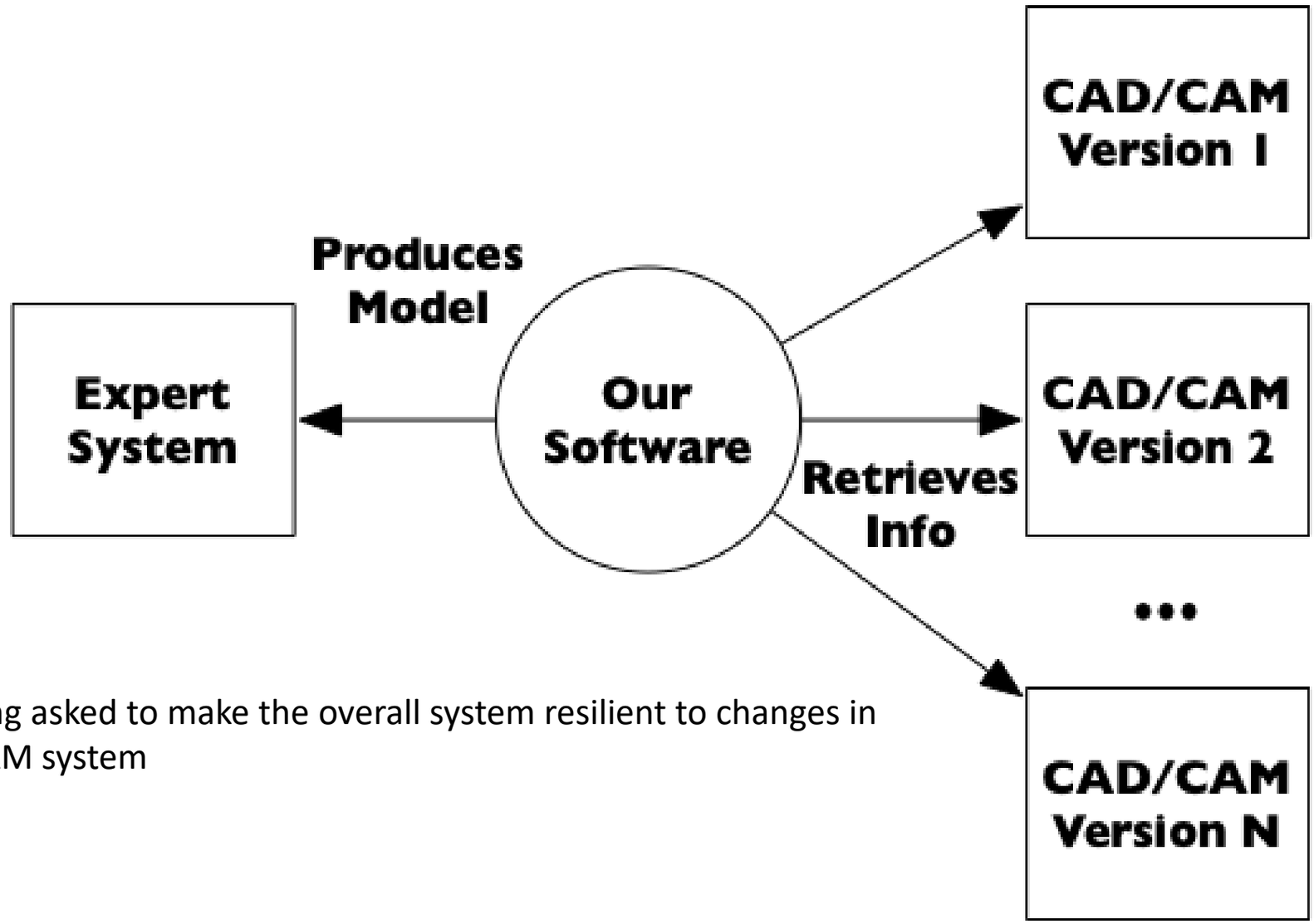
Nice system!

- The engineers get to use familiar tools when designing new parts
- The expert system encodes all the rules about how the cutter is used to create parts out of features
- Our software simply acts as the “glue” between these two major components
  - extracting information and converting it into a format that the expert system understands
- The use of existing CAD software was a good decision
  - Imagine if the original development team had been infected with **Not Invented Here** syndrome and had decided they needed to build a modeling tool
    - It would have increased expense and complexity
      - Plus their tool would likely have been non-standard
    - Sometimes, “buy” is the best option of a “buy vs. build” decision
      - be sure to leverage standards in your system designs

So, What's the Problem?

- So far, all I've presented is information about the application domain
  - What we are missing is details concerning what the problem might be
- **Don't confuse supplemental/domain information for a problem statement**
  - As designers, we need to know **what the problem is**

Here's the Problem



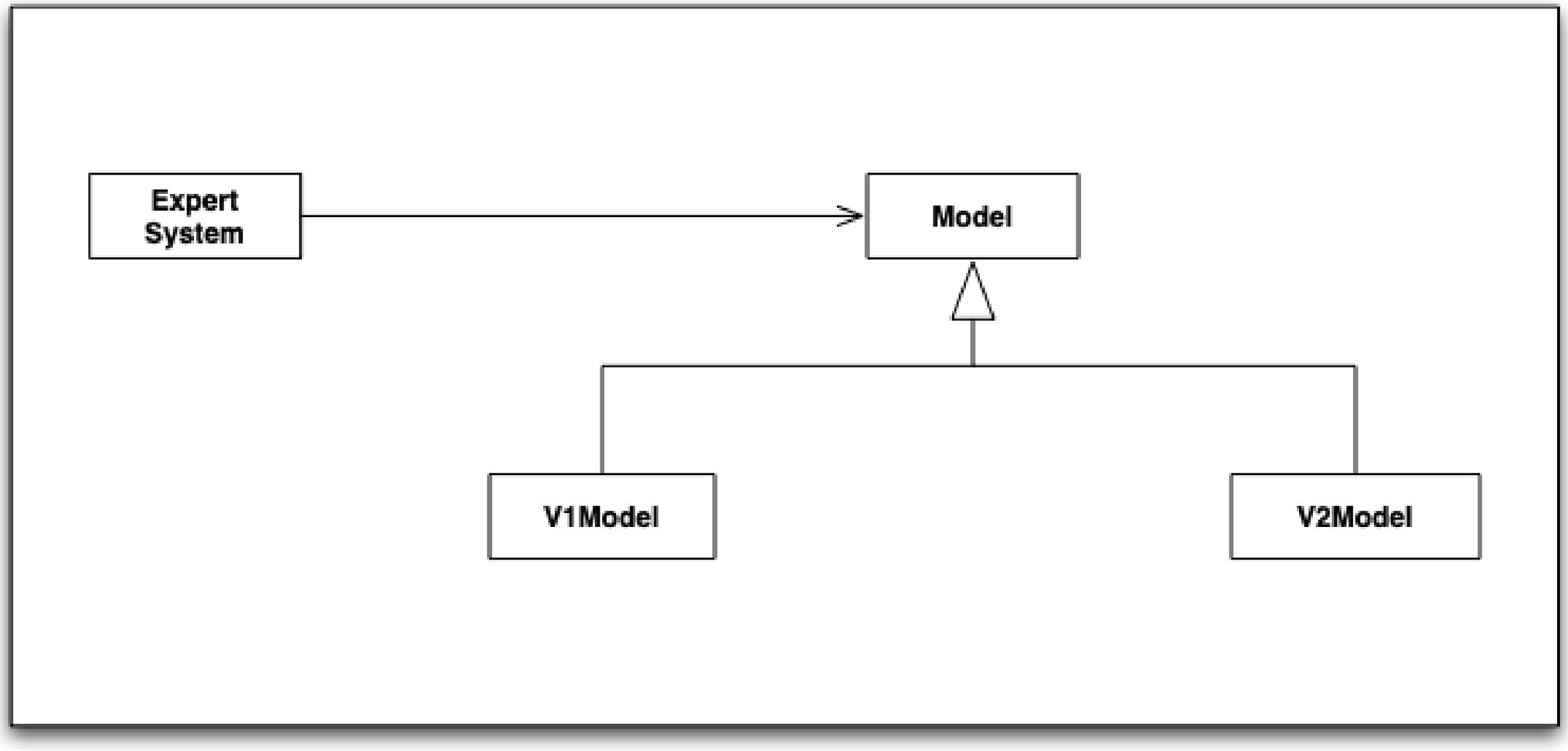
We are being asked to make the overall system resilient to changes in the CAD/CAM system

Example of encapsulation via software architecture…

Discussion

- Our problem is to allow the expert system to work with multiple CAD systems
  - currently different versions of the existing CAD system or (possibly) CAD systems from different vendors
- Why not replace the expert system?
  - It was an expensive piece of software to develop and embodies a significant amount of domain knowledge
    - Translating models into commands for the cutter is non trivial
    - Punching features in the wrong order produces defective parts
- This type of legacy system is common; you just have to incorporate it into your design

Our Approach



We want to provide the expert system with a single model that it understands; we will subclass this model to integrate the different versions of the CAD system

Understanding the Challenges

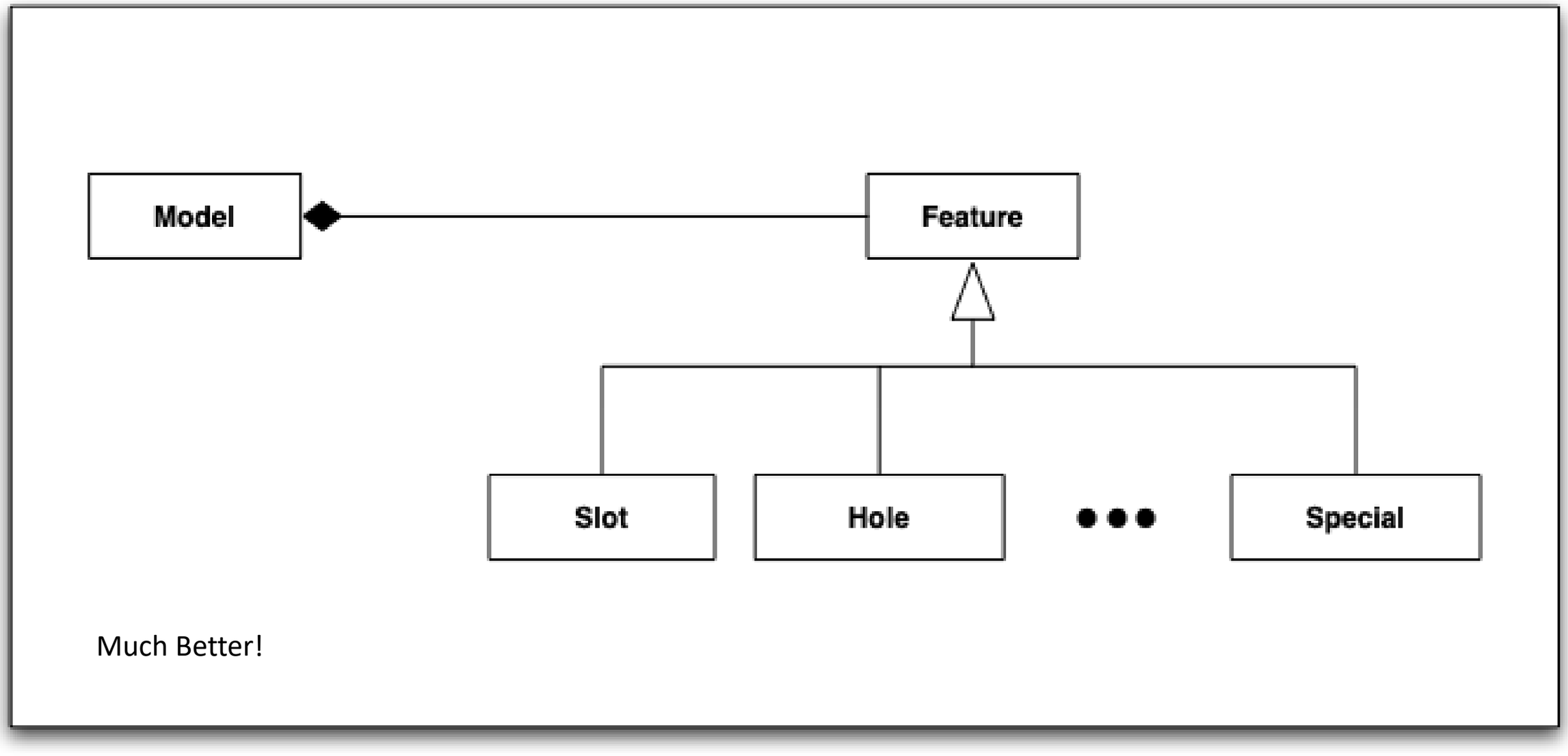
- The API of version 1 of the CAD system is NOT object-oriented
  - It is accessed via a set of library routines
    - (think C API)
- The API of version 2 of the CAD system is object-oriented
  - It provides an OO framework of classes to describe its models

Accessing the Version 1 API

- V1 API
  - model_t *get_model(char *name);
  - int number_of_features(model_t *model);
  - int get_id_of_ith_feature(model_t *model, int index);
  - feature_type get_feature_type(model_t *model, int id);
  - int get_x_coord_of_slot(model_t *model, int id);
- (Lovely C code, I almost know how it works.)
- To get the x coordinate of a feature, I need to do something like

```
model_t *model = get_model(“part XYZ”);
int num = number_of_features(model);
for (int i = 0; i < num; i++) {
        int id = get_id_of_ith_feature(model, i);
        switch (get_feature_type(model, id)) {
                case SLOT:
                        int x = get_x_coord_of_slot(model, id);
                        …
```

Version 2's API



Much Better!

Discussion: The Challenge is Clear

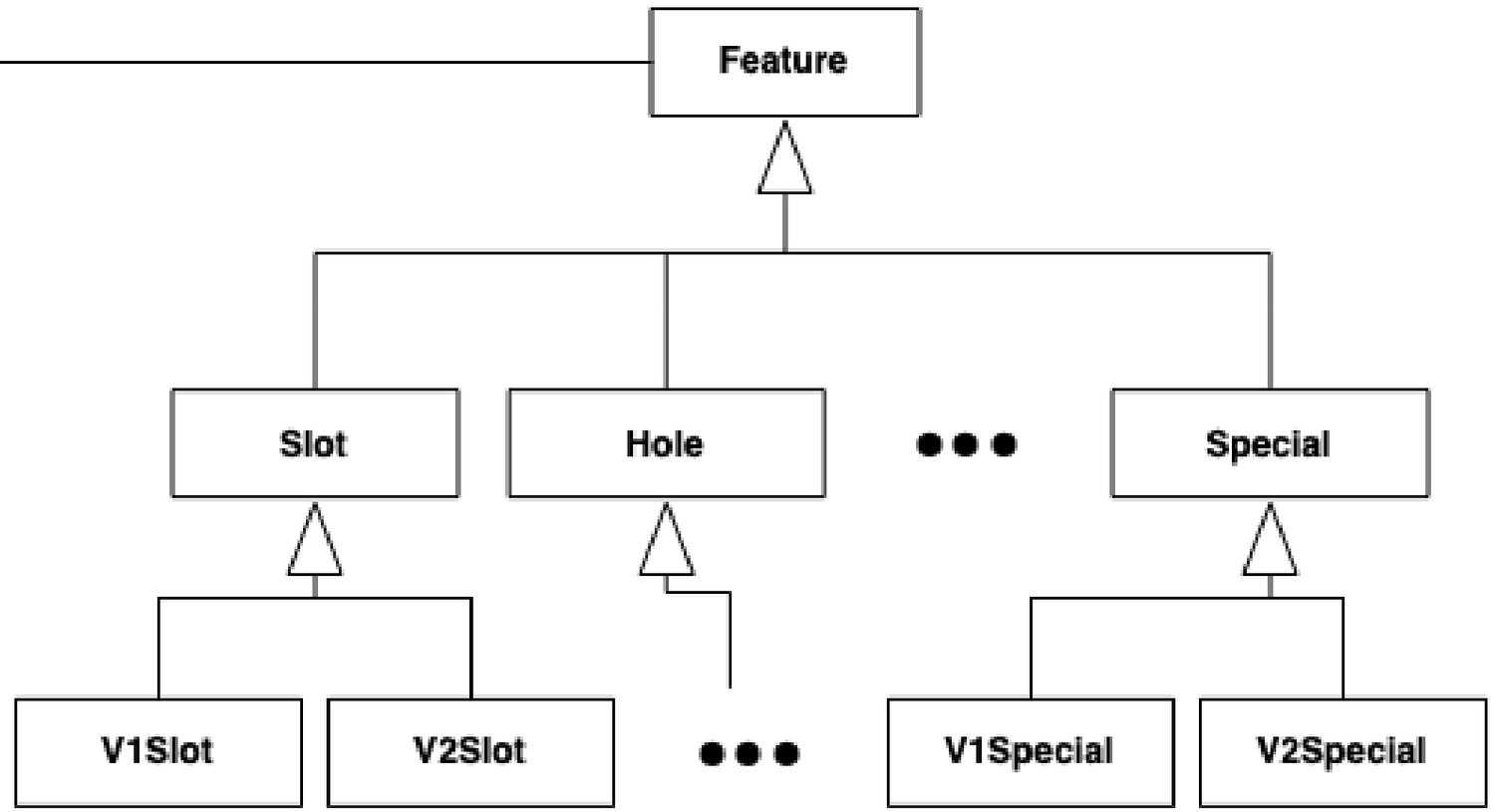
- We want to give the expert system an OO API
  - Version 2 provides us with a nice OO model, so our system will need to "wrap" those classes in some way
  - Version 1 provides only library routines, so our system will need to "hide" the non-OO API from the expert system
  - If we do this right, we will be able to write robust, polymorphic code for the expert system that doesn't change when support for a new CAD system is added to our system

First Attempt: Not so Great

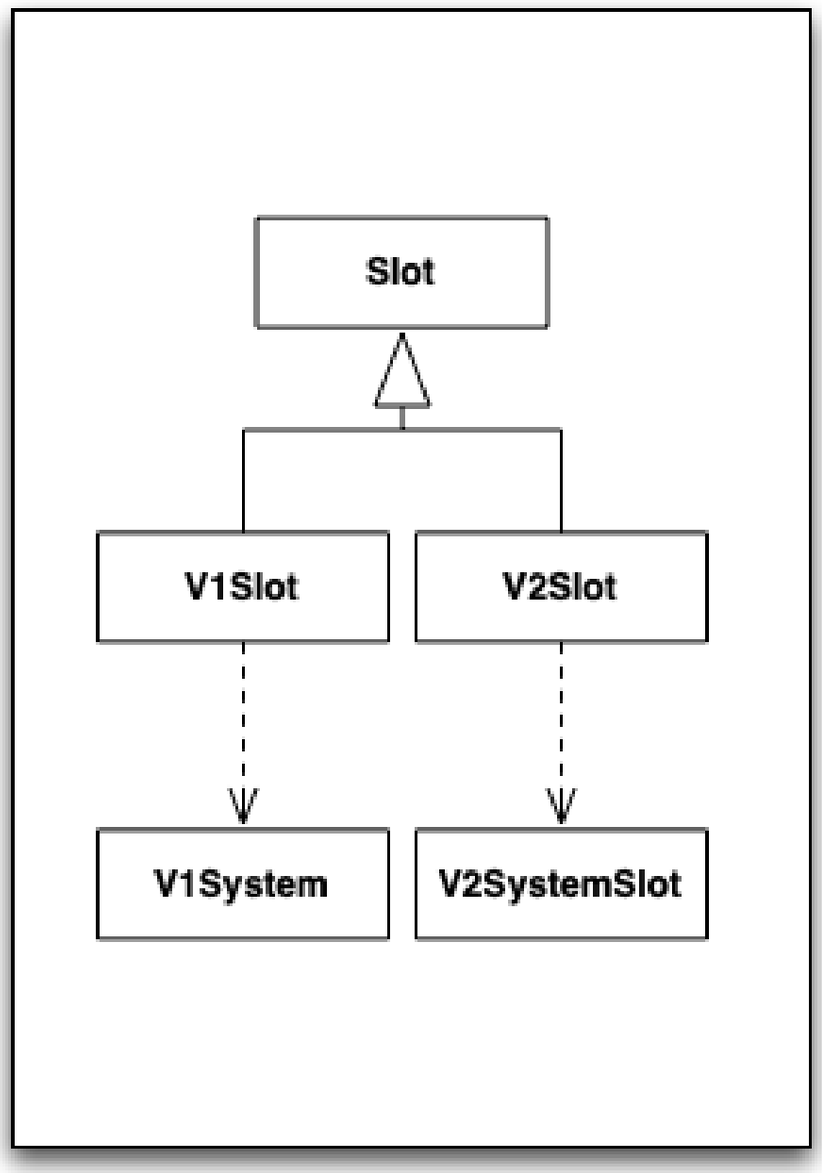
- This problem comes from the Shalloway/Trott book
- In the book, they make an initial attempt to solve the problem is…
  - “It is not a great solution, but it is a solution that would work.”
- The idea is to present an obvious elaboration of the approach outlined so far
  - and then highlight some obvious problems it has
  - these problems will be dealt with later as we learn about OO and patterns

The Basic Approach (I)

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



The Basic Approach (II)



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”

Note on Polymorphism

- The designer here says their goal is **not** to achieve polymorphism across Features
- In their design, they assign different sets of methods to different feature subclasses rather than trying to define all of the methods in the top level Feature class
  - The expert system needs to know the types of features it is dealing with
  - Abstracting those details away will prevent it from doing its job
- This situation is less than ideal (it would be nice to put the knowledge of what to do for a particular feature inside of it) but
  - Here we're in a situation where "figuring out what to do" cannot be isolated inside a single class; different combinations of features require different strategies

Note on Polymorphism

- Not achieving polymorphism means they are not striving to support client code like this
  - for (Feature f : features) {
    - f.doSomething();
  - }
- The expert system needs to differentiate among the various feature types; the design does achieve polymorphism across the V1* and V2* subclasses
  - Slot s = <retrieve a slot>; s.getLength(); // polymorphic across V1 and V2 subclasses

Problems with the Design

- The design has four problems that the authors highlight
  - 1. Redundancy among methods
    - Lots of duplicated code or highly similar code is likely across V1 subclasses
      - OO designers hate duplicated code!
  - 2. “Messy”, “Ill structured”, “Cumbersome”
    - something doesn’t feel quite right about the design
  - 3. Tight coupling
    - The design is tightly coupled to the different CAD systems; A lot of code will need to be changed or produced if a new CAD system is added or an existing one is changed
  - 4. Weak cohesion
    - core functionality is too widely dispersed across the various classes; Model is too simple a class

Potential for Class Growth

- The final problem is that the design does not scale nicely
  - (# of features * # of CAD systems) + 7 core classes
  - 5 features, 2 systems = 17 classes
  - 25 features, 10 systems = 257 classes (!!)
- Especially bad if something else about the system suddenly started to vary, even the “worst case” of “# of expert systems”

- Will the OO design patterns help us with these problems?…
- Are there design approaches that would help us work this problem?
- We’ll revisit this…