Arthur Riel Design Heuristics

- from Object-Oriented Design Heuristics by Arthur Riel
- Read the Heuristics…
  - Find reasons why
  - We will discuss them during the lectures

Different Relationship

- Uses
- Contains
- Specializes

Hidding Data

- All data should be hidden within its class

No Dependence on Clients

- Users of a class must be dependent on its public interface, but a class should not be dependent on its users

Support Class = one Clear Responsibility

- Minimize the number of messages in the protocol of a class

Supporting Polymorphism and Communication

- Implement a minimal interface that all classes understand
- To send the same message to different objects
- To be able to substitute them

Remember:

```
grObjects do: [:each | each translate: 10@10]
```

- Example: Object>>printString, Object>>copy...

Clear Public Interface

- Do not put implementations details such as common-code private functions into the public interface of a class

- Example:
  - Private/protected in C++
  - Private method categories in Smalltalk

- Do not clutter the public interface of a class with items that clients are not able to use or are not interested in using

Minimize Classes Interdependencies

- A class should only use operations in the public interface of another class or have nothing to do with that class

Support a Class = one Responsibility

- A class should capture one and only one key abstraction

Strengthen Encapsulation

- Keep related data and behavior in one place
- Spin off non related information into another class

- -> Move Data Close to Behavior

Object: a Cohesive Entity

- Most of the methods defined on a class should be using most of the instance variables most of the time

Roles vs. Classes

- Be sure the abstractions you model are classes and not the roles objects play

- Are mother and father classes or role of Person?
- No magic answer: Depend on the domain
- Do they have different behavior? So they are more distinct classes

Support one Class = one Responsibility

- Distribute system intelligence horizontally as uniformly as possible, i.e., the top-level classes in a design should share the work

Support one Class = one Responsibility

- Do not create god classes/objects (classes that control all other classes). Be very suspicious of class whose names contains Driver, Manager, System, SubSystem

Model and Interfaces

- Model should never be dependent on the interface that represents it. The interface should be dependent on the model

- What is happening if you want two different Uis for the same model?

Basic Checks for God Class Detection

- Beware of classes that have many accessor methods defined in their public interface. May imply that data and behavior is not being kept at the same place

- Beware of classes having methods that only operate on a proper subset of the instance variables.

One Class: One Responsibility

- One responbility: coordinating and using other objects
  - OrderedCollection maintains a list of objects sorted by arrival order: two indexes and a list

- Class should not contain more objects than a developper can fit in his short-term memory. (6 or 7 is the average value)

Classes Evaluation

- Model the real world whenever possible
- Eliminate irrelevant classes
- Eliminate classes that are outside of the system
- A method is not a class. Be suspicious of any class whose name is a verb or derived from a verb, especially those that only one piece of meaningful behavior

Minimizing Coupling between Classes

- Minimize the number of classes with which another class collaborates
- Minimize the number of messages sent between a class and its collaborators
  - Counter example: Visitor patterns
- Minimize the number of different messages sent between a class and its collaborators

About the Use Relationship

- When an object use another one it should get a reference on it to interact with it

- Ways to get references
  - (containment) instance variables of the class
  - Passed has argument
  - Ask to a third party object (mapping…)
  - Create the object and interact with it (coded in class: kind of DNA)

Containment and Uses

- If a class contains object of another class, then the containing class should be sending messages to the contained objects (the containment relationship should always imply a uses relationships)

- A object may know what it contains but it should not know who contains it.

Representing Semantics Constraints

- How do we represent possibilities or constraints between classes?
  - Appetizer, entrée, main dish…
  - No peas and corn together…
- It is best to implement them in terms of class definition but this may lead to class proliferation
- => implemented in the creation method

Objects define their logic

- When implementing semantic constraints in the constructor of a class, place the constraint definition as far down a containment hierarchy as the domain allows

=> Objects should contain the semantic constraints about themselves

Third party constraint holder

- If the logic of a constraint is volatile, then it is better to

Classes - Subclasses

- Superclass should not know its subclasses
- Subclasses should not use directly data of superclasses
- If two or more classes have common data and behavior, they should inherit from a common class that captures those data and behavior

Controversial

- All abstract classes must be base classes

- All base classes should be abstract classes
  -> Not true they can have default value method

Fundamental: Avoid Type Checks

- Explicit case analysis on the type of an objects is usually an error.
- An object is responsible of deciding how to answer to a message
- A client should send message and not discriminate messages sent based on receiver type