

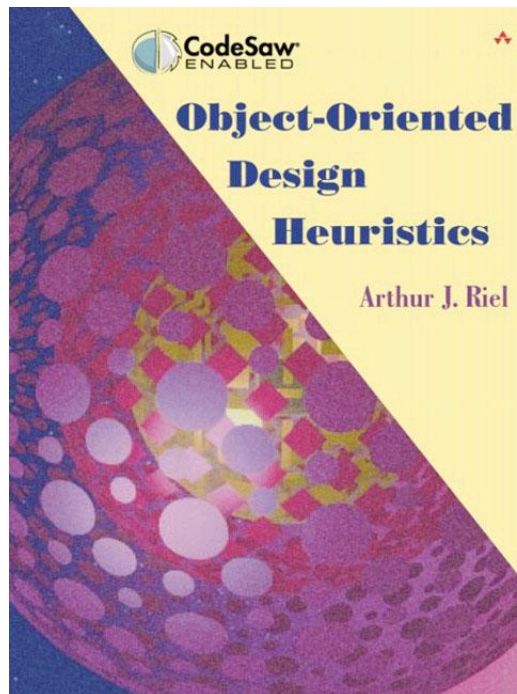
Object-Oriented Design Heuristics

Harald Gall, Prof. Dr.

<http://seal.ifi.unizh.ch>

Design Heuristics

- Object-Oriented Design Heuristics by Arthur Riel, Addison-Wesley, 1996.



Goal

- Insights into oo design improvement.
- More than **sixty guidelines** are language-independent and allow one to rate the integrity of a software design.
- The **heuristics** are not written as hard and fast rules; they are meant to serve **as warning mechanisms** which allow the flexibility of ignoring the heuristic as necessary.

Classes and Objects

The Building Blocks of the Object-Oriented Paradigm

Hidding Data

- Heuristic #2.1
All data should be hidden within its class

No Dependence on Clients

- Heuristic #2.2

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

Support Class = 1 Clear Responsibility

- Heuristic #2.3
Minimize the number of messages in the protocol of a class

Supporting Polymorphism and Communication

- Heuristic #2.4
Implement a minimal public interface which all classes understand (e.g. operations such as copy (deep versus shallow), equality testing, pretty printing, parsing from a ASCII description, etc.).
- To send the same message to different objects
- To be able to substitute them

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

Clear Public Interface

- Heuristic #2.5
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

- Heuristic #2.7

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

- Heuristic #2.8
A class should capture one and only one key abstraction

Strengthen Encapsulation

- Heuristic #2.9
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

- Heuristic #2.11
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: Depends on the domain
- Do they have different behavior? So they are more distinct classes

Topologies of Action-Oriented vs. Object-Oriented Applications

Support 1 Class = 1 Responsibility

- Heuristic #3.1

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

Support 1 Class = 1 Responsibility

- Heuristic #3.2

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

Model and Interfaces

- Heuristic #3.5
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

- **Heuristic #3.3**
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
- **Heuristic #3.4**
Beware of classes having methods that only operate on a proper subset of the instance variables.

One Class: One Responsibility

- One responsibility: 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 developer 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

The Relationships Between Classes and Objects

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 pattern
- 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

- Heuristic #4.5
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.

Coherence in Classes

- Heuristic #4.6
Most of the **methods** defined on a class should be using most of the data members most of the time.
- Heuristic #4.7
Classes should **not contain more objects** than a developer can fit in his or her short term memory. A favorite value for this number is six.
- Heuristic #4.8
Distribute system intelligence vertically down narrow and deep containment hierarchies.

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

- Heuristic #4.9

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

- Heuristic #4.11
The **semantic information** on which a constraint is based is best placed in a **central third-party object** when that information is **volatile**.
- Heuristic #4.12
The **semantic information** on which a constraint is based is best **decentralized** among the classes involved in the constraint when that information is **stable**.

The Inheritance Relationship

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

Inheritancs for Specialization

- Heuristic #5.1
Inheritance should only be used to model a specialization hierarchy.
- Vererbung ist ein White-box-Entwurf
- Aggregation oder Komposition definieren einen Black-box-Entwurf
- Richtiger Einsatz von Vererbung erleichtert das Programmverständnis enorm!

Base Class Knowledge

- Heuristic #5.2

Derived classes must have **knowledge of their base class** by definition, but base classes should **not** know anything **about their derived classes**.

Base Class Data is Private

- Heuristic #5.3
- All data in a base class should be private, i.e. do not use protected data.

Inheritance Depth

- Heuristic #5.4

Theoretically, inheritance hierarchies should be deep, i.e. the deeper the better.

- Heuristic #5.5

Pragmatically, inheritance hierarchies should be no deeper than an average person can keep in their short term memory. A popular value for this depth is six.

Controversial

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

Interfaces

- Heuristic #5.8
Factor the **commonality of data**, behavior, and/or interface as high as possible in the inheritance hierarchy.

- Heuristic #5.10
If two or more classes have common data and behavior (i.e. methods) then those classes should each inherit from a **common base class** which captures those data and methods.

Inheritance (2)

- Heuristic #5.9
If two or more classes **only share common data (no common behavior)** then that common data should be placed in a class which will be contained by each sharing class.
- Heuristic #5.11
If two or more classes **only share common interface** (i.e. messages, not methods) then they should inherit from a common base class only if they will be used polymorphically.

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

Dynamic semantics

- Heuristic #5.14
Do **not** model the **dynamic semantics** of a class through the use of the **inheritance** relationship. An attempt to model dynamic semantics with a static semantic relationship will lead to a toggling of types at runtime.
- Heuristic #5.15
Do **not** turn **objects of a class into derived classes** of the class. Be very suspicious of any derived class for which there is only one instance.

Multiple Inheritance

Prove Multiple Inheritance

- Heuristic #6.1

If you have an example of multiple inheritance in your design, assume you have made a mistake and **prove** otherwise.

Question it

- Heuristic #6.2

Whenever there is inheritance in an object-oriented design ask yourself two questions:

- 1) Am I a special type of the thing I'm inheriting from?
- 2) Is the thing I'm inheriting from part of me?

- Heuristic #6.3

Whenever you have found a multiple inheritance relationship in a object-oriented design be sure that no base class is actually a derived class of another base class, i.e. **accidental multiple inheritance**.

The Association Relationship

Containment

- Heuristic #7.1

When given a choice in an object-oriented design **between a containment relationship and an association relationship**, choose the **containment relationship**.

Class Specific Data and Behavior

Use of Class Variables & Methods

- Heuristic #8.1

Do not use global data or functions to perform **bookkeeping** information on the objects of a class, class variables or methods should be used instead.

Summary

- Use the guidelines for
 - insightful analysis
 - critical reviews
 - as guide for better oo design
 - to build reusable components and frameworks