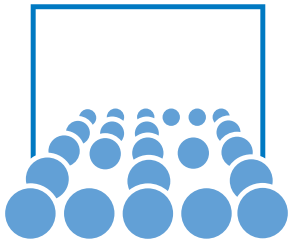
Software Design and Patterns

Advanced Programming Tutorials

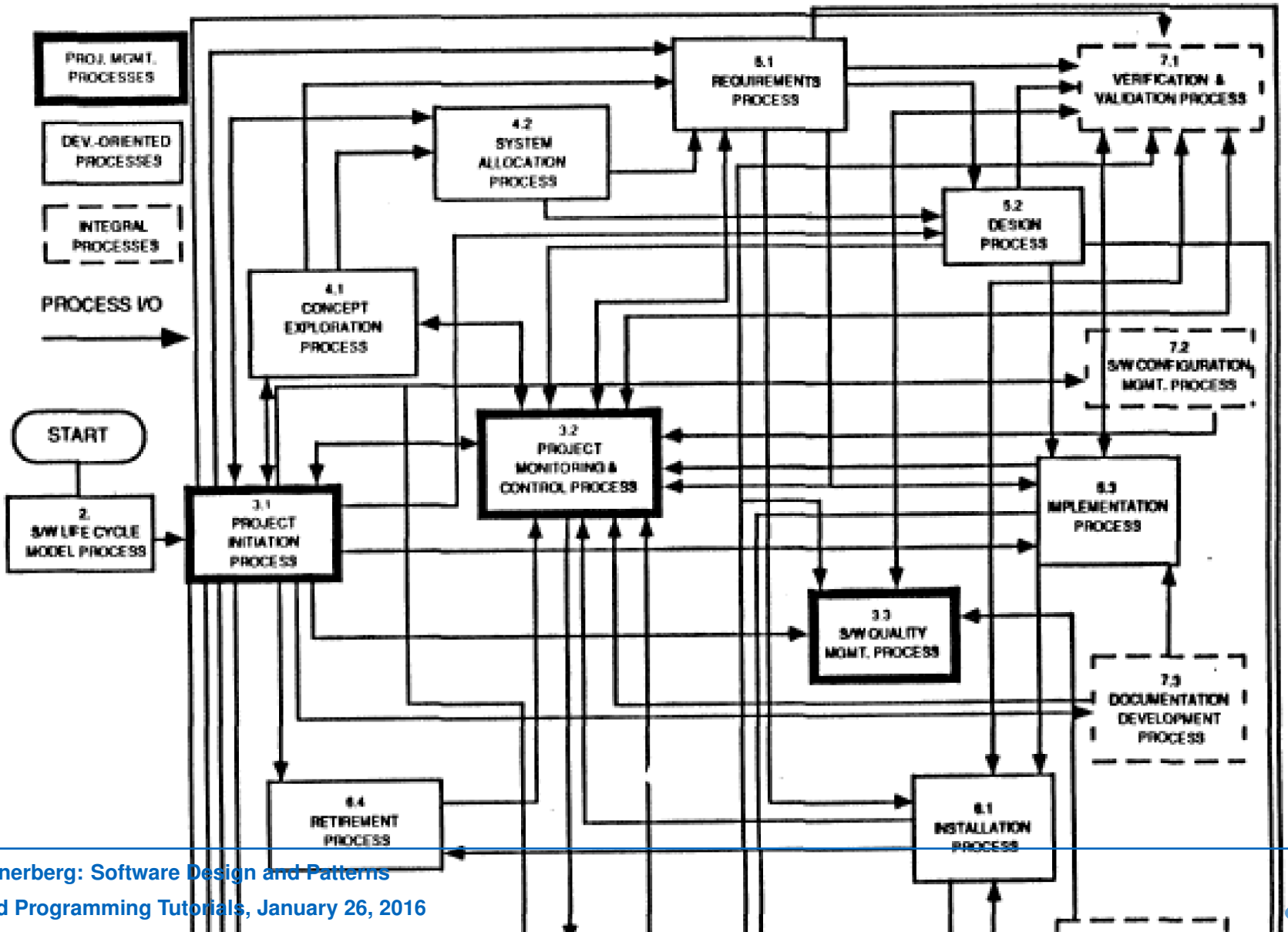
Erik Wannerberg
Supervisors: Roland Wittmann, Philipp Neumann

January 26, 2016



Content

What is this?!?


PROJ. MGMT. PROCESSES
DEV.-ORIENTED PROCESSES
INTEGRAL PROCESSES
PROCESS I/O
START
2. S/W LIFE CYCLE MODEL PROCESS
3.1 PROJECT INITIATION PROCESS
3.2 PROJECT MONITORING & CONTROL PROCESS
3.3 S/W QUALITY MGMT. PROCESS
4.1 CONCEPT EXPLORATION PROCESS
4.2 SYSTEM ALLOCATION PROCESS
5.1 REQUIREMENTS PROCESS
5.2 DESIGN PROCESS
5.3 IMPLEMENTATION PROCESS
6.1 INSTALLATION PROCESS
6.4 RETIREMENT PROCESS
7.1 VERIFICATION & VALIDATION PROCESS
7.2 S/W CONFIGURATION MGMT. PROCESS
7.3 DOCUMENTATION DEVELOPMENT PROCESS

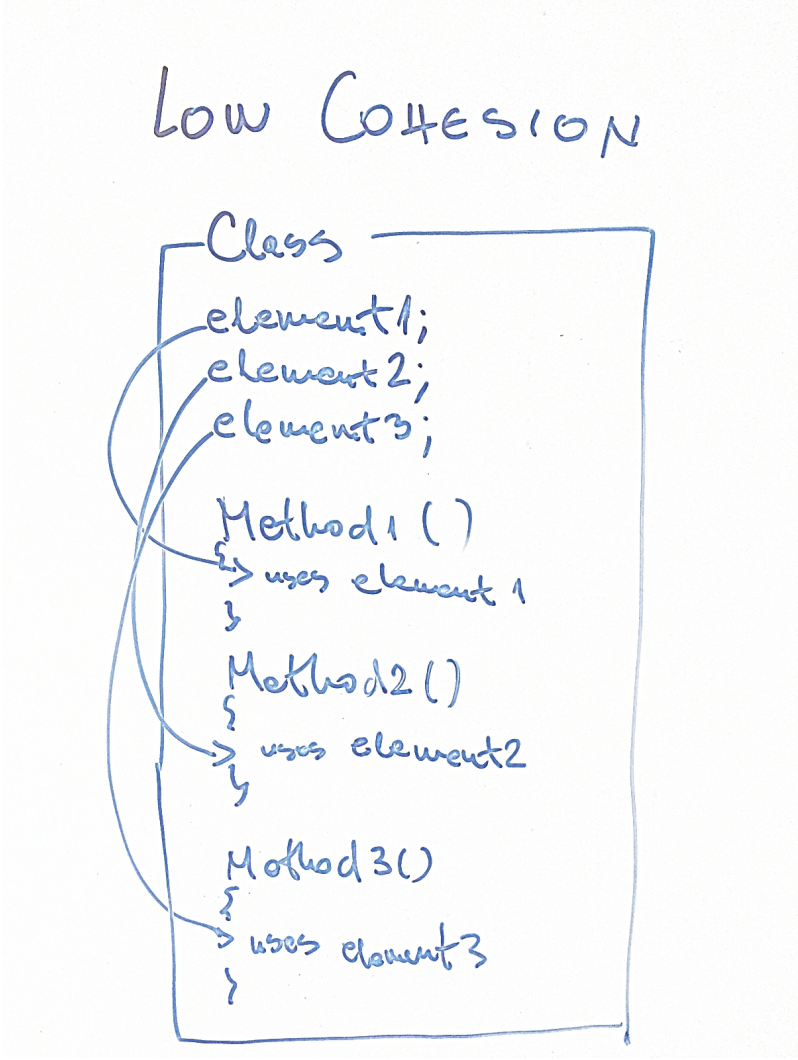
Software Engineering Principles

What do we want from our Software?

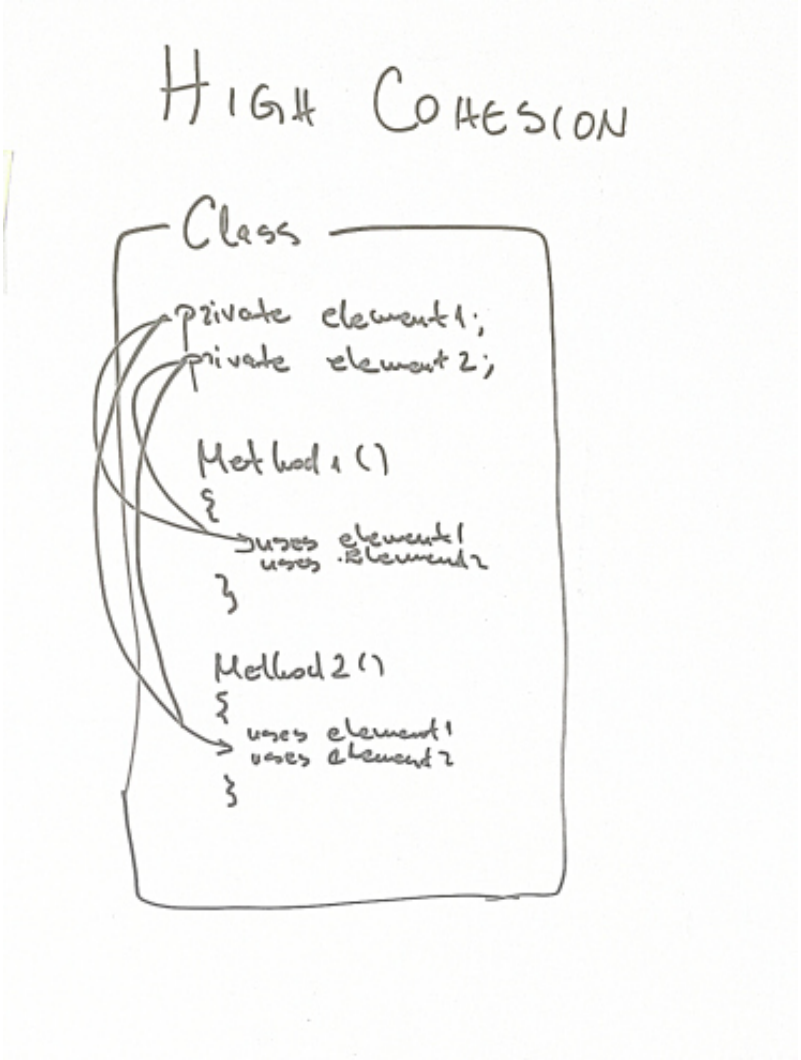
- Should be well-suited to the task (Performance etc.)
- Should be as easy to make as possible
- Should be as easy to modify as possible
- Should be as understandable as possible

Cohesion

Bad

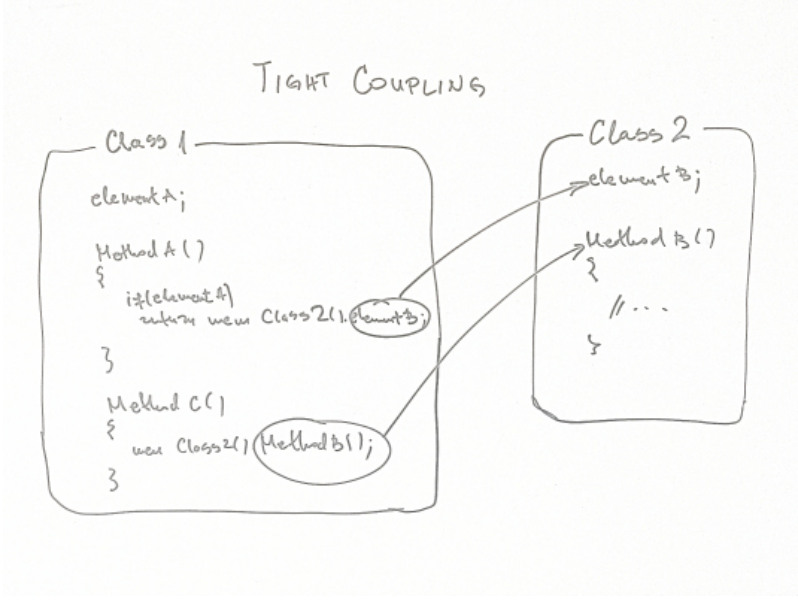


Good

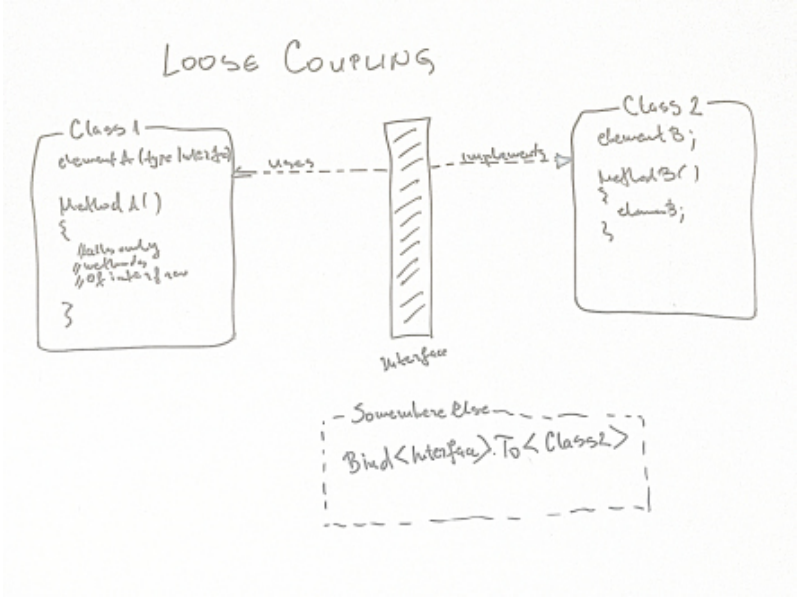


Coupling

Bad



Good



What are Design Patterns?

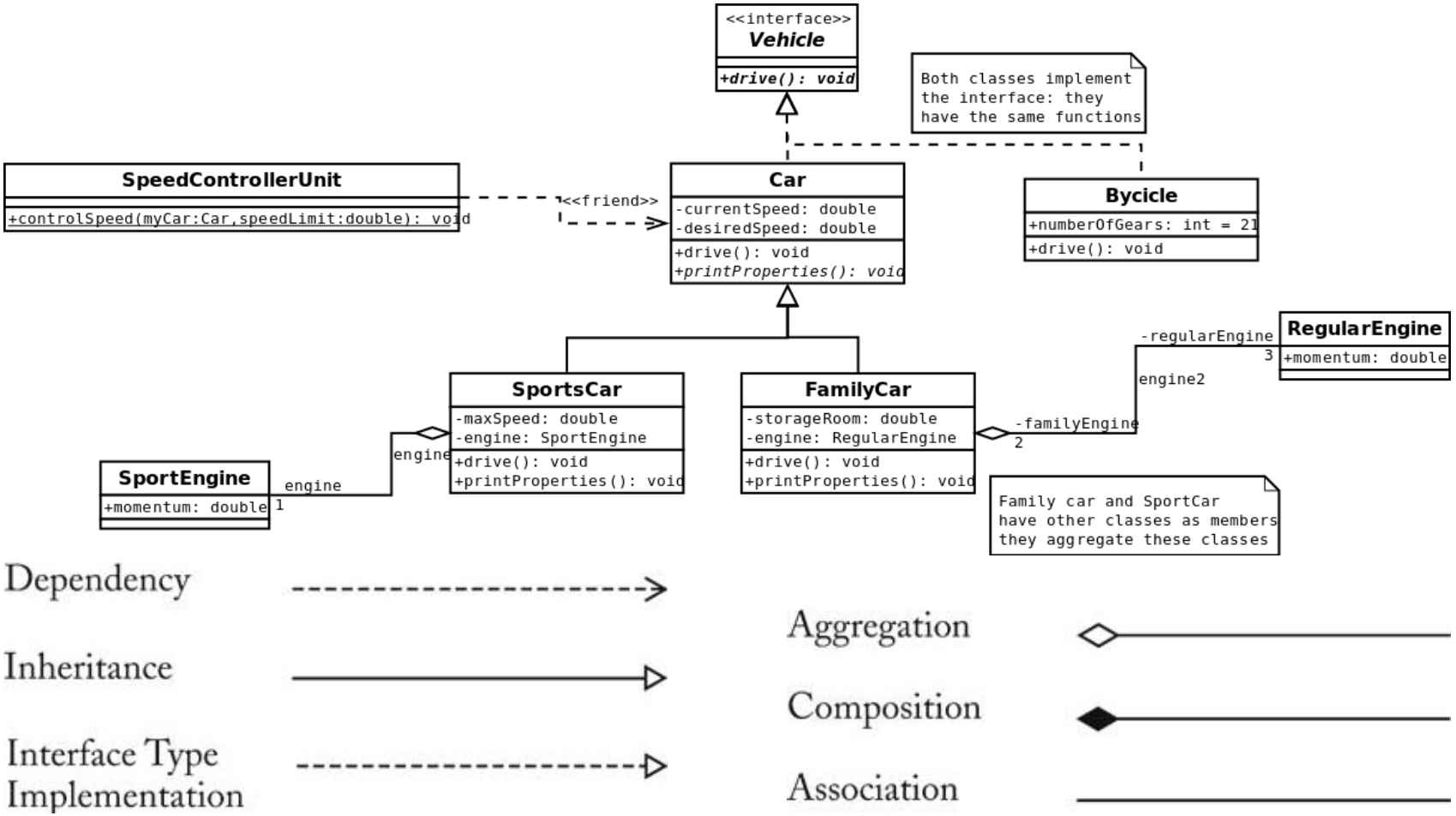
> *A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context.*

- from Design Patterns: Elements of Reusable Object-Oriented Software [1]

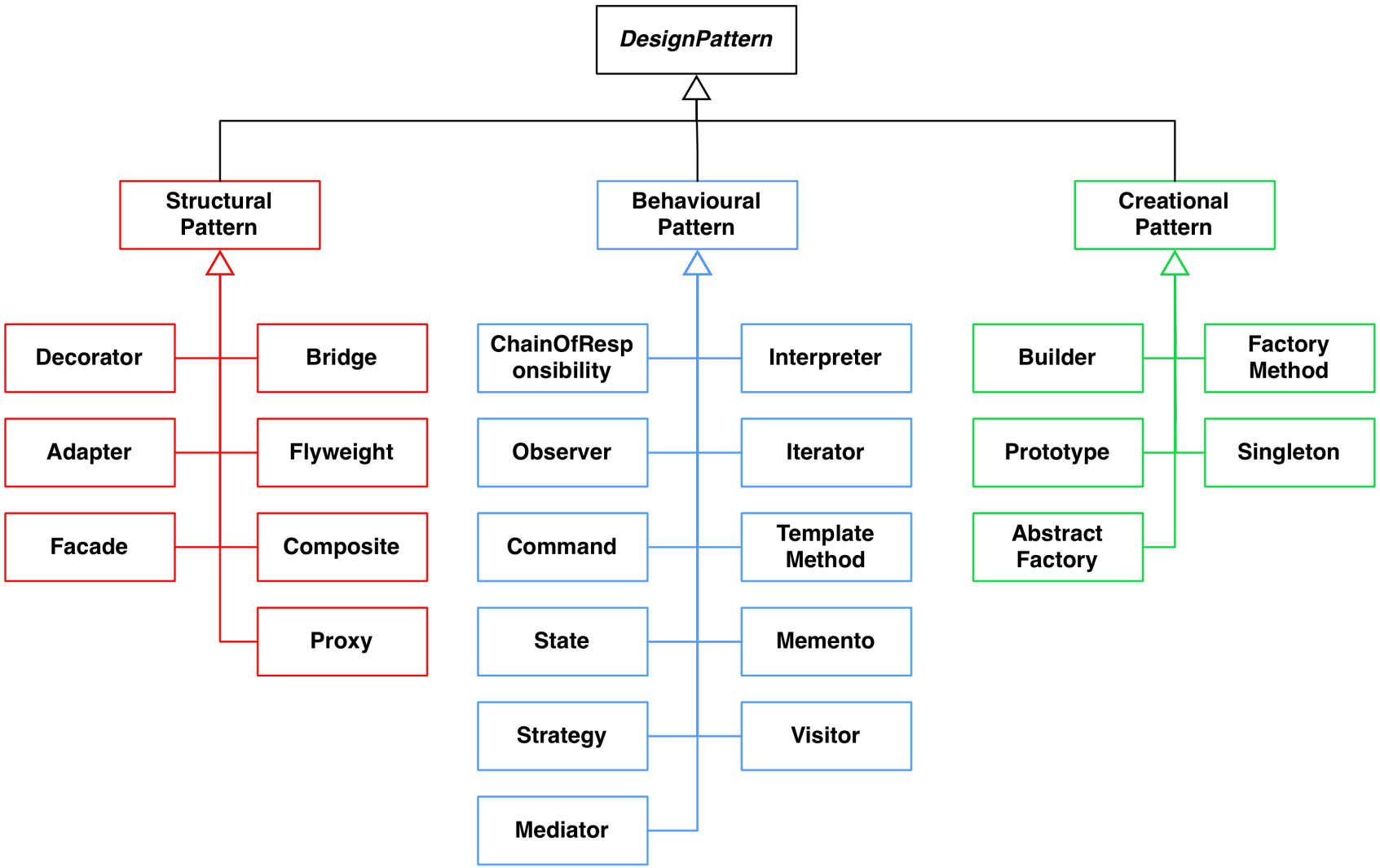
What makes Design Patterns Good?

- They are generalizations of detailed design knowledge from existing systems
- They provide a shared vocabulary to designers
- They provide examples of reusable designs
  - Polymorphism (Inheritance, sub-classing)
  - Delegation (or aggregation).

Short UML recap

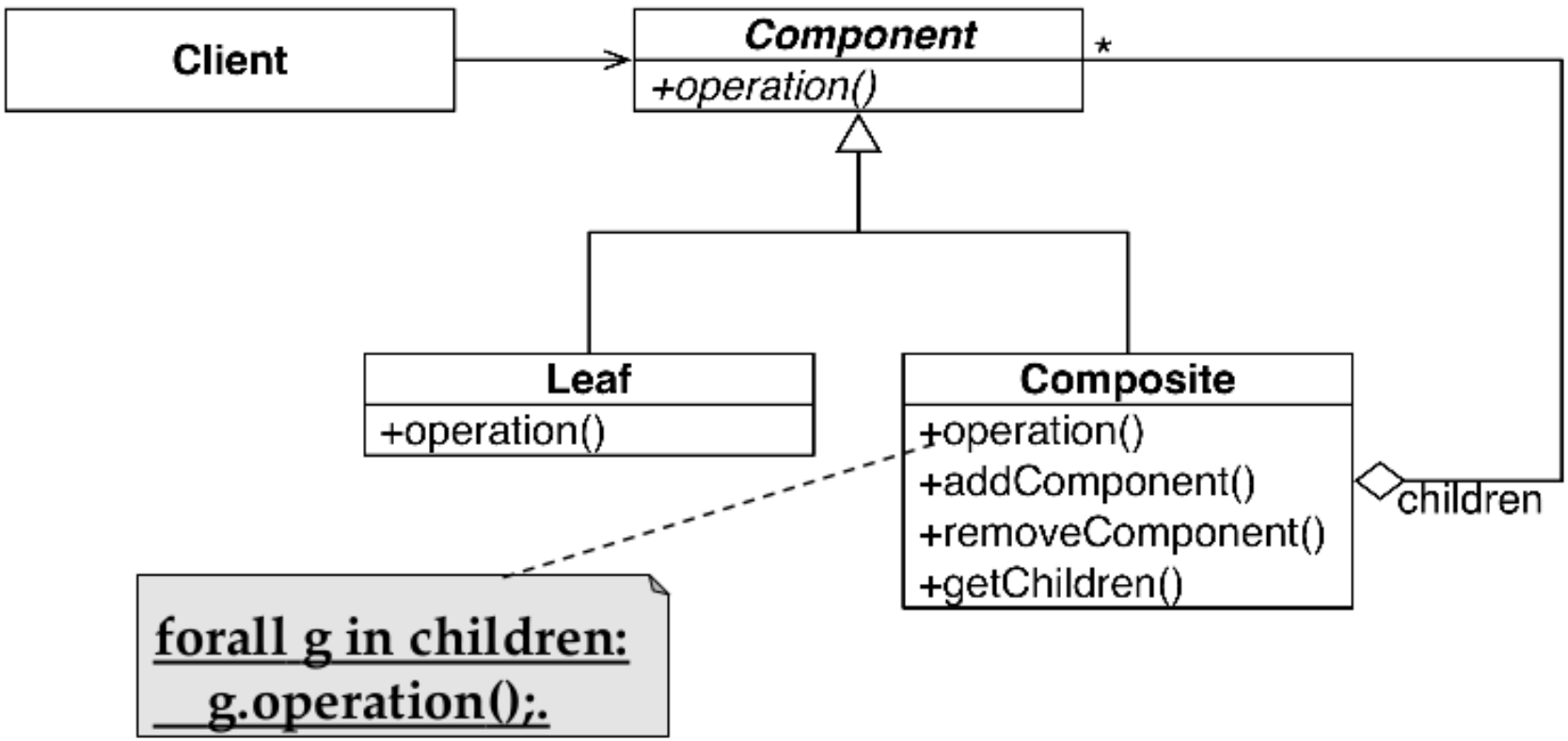


Some Design Patterns

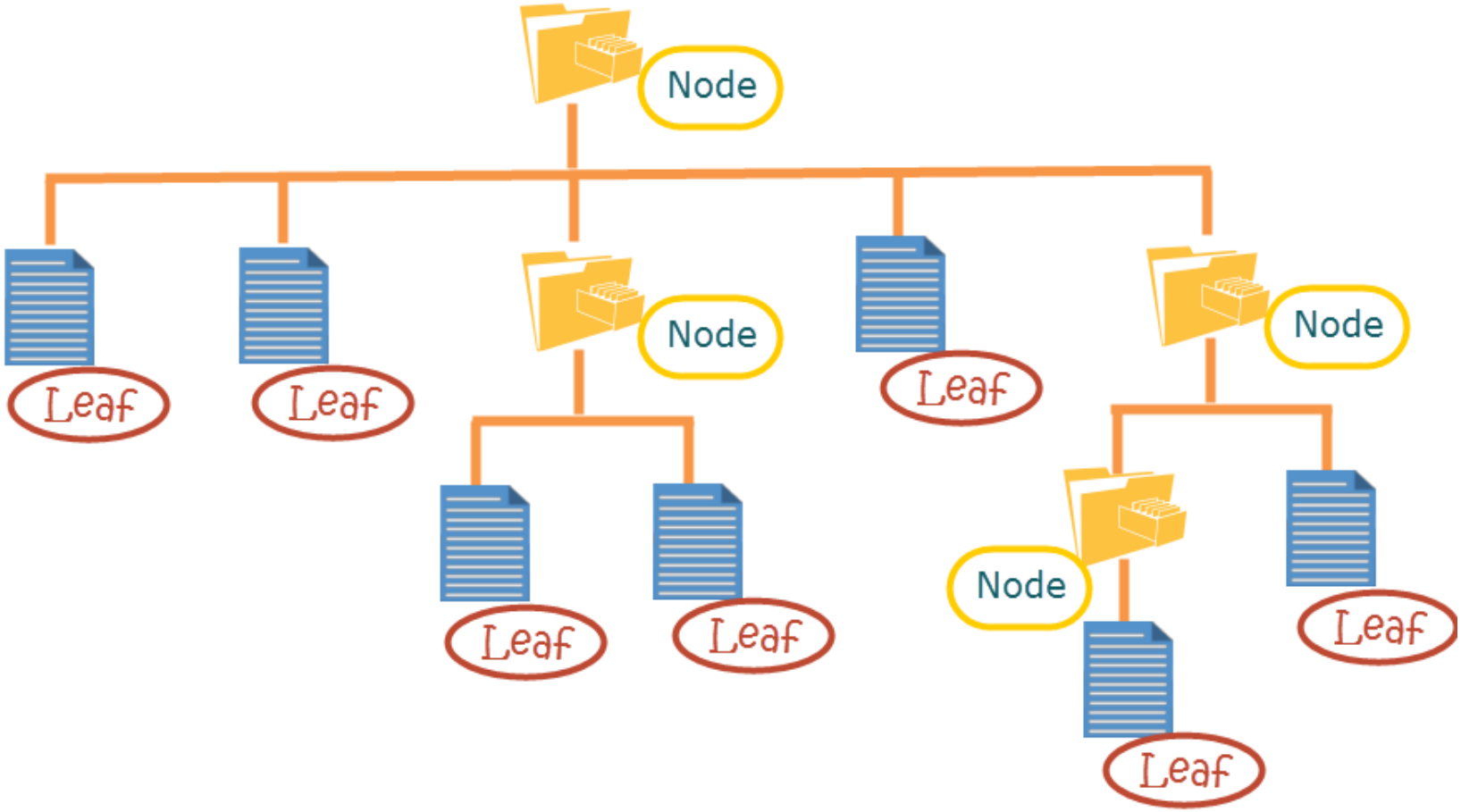


Structural pattern: Composite

...for building trees, lists, anything...



Structural pattern: Composite



Structural pattern: Composite

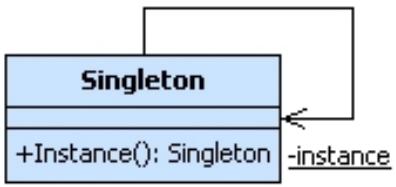
```cpp
class Component {
public:
    virtual void doStuff() = 0;
    virtual ~Component(){}
};
```

```cpp
class Leaf : public Component {
public:
    virtual void doStuff(){
        //do leaf stuff...
    }
};
```

```cpp
class Composite : public Component {
public:
    virtual void doStuff() {
        for(auto child : children){
            child.doStuff();
        }
    }
    //other methods for adding/removing/finding children etc.
private:
    std::list<Component> children;
};
```

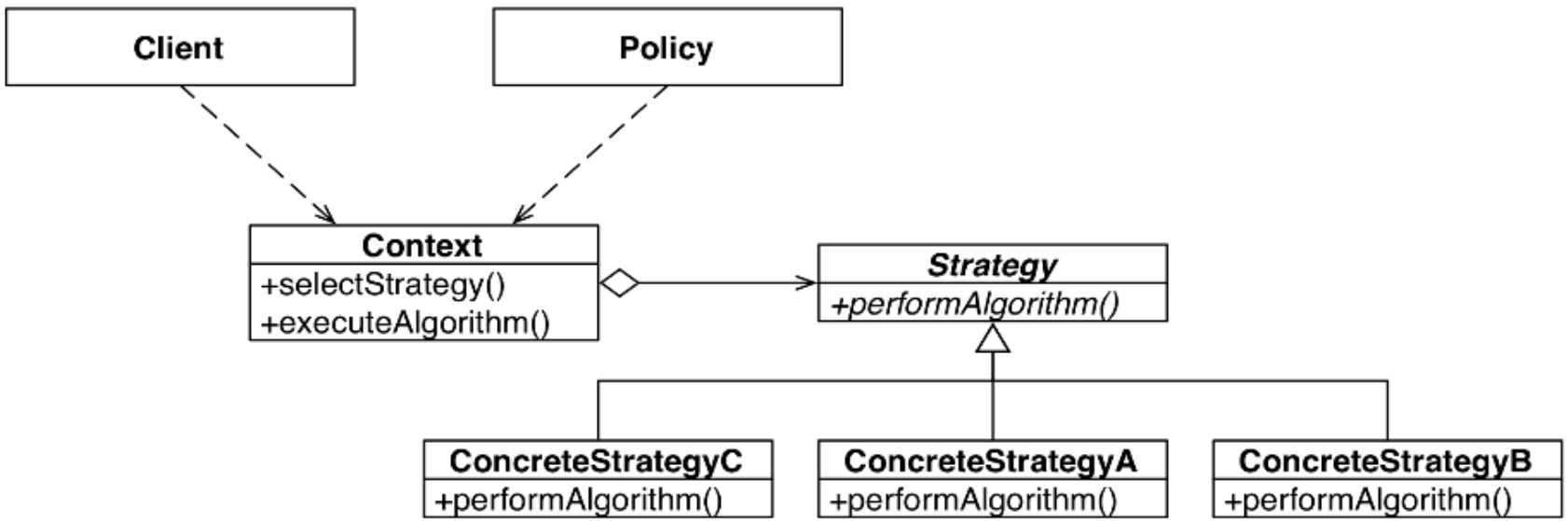
Creational pattern: Singleton

```
1  class Singleton {
2  public:
3      static Singleton& getInstance()
4      {
5          static Singleton uniqueInstance;
6          return uniqueInstance;
7      }
8
9      //... other methods with stuff you want to do...
10
11 private:
12     Singleton(){}
13     //private constructor =>
14     //can only call/create object from within class!
15
16     //(also have to do with copy assignment operator
17     //and copy constructor to make uncopyable)
18 };
```



Behavioural Pattern: Strategy

Dynamically change which strategy to use based on a Policy



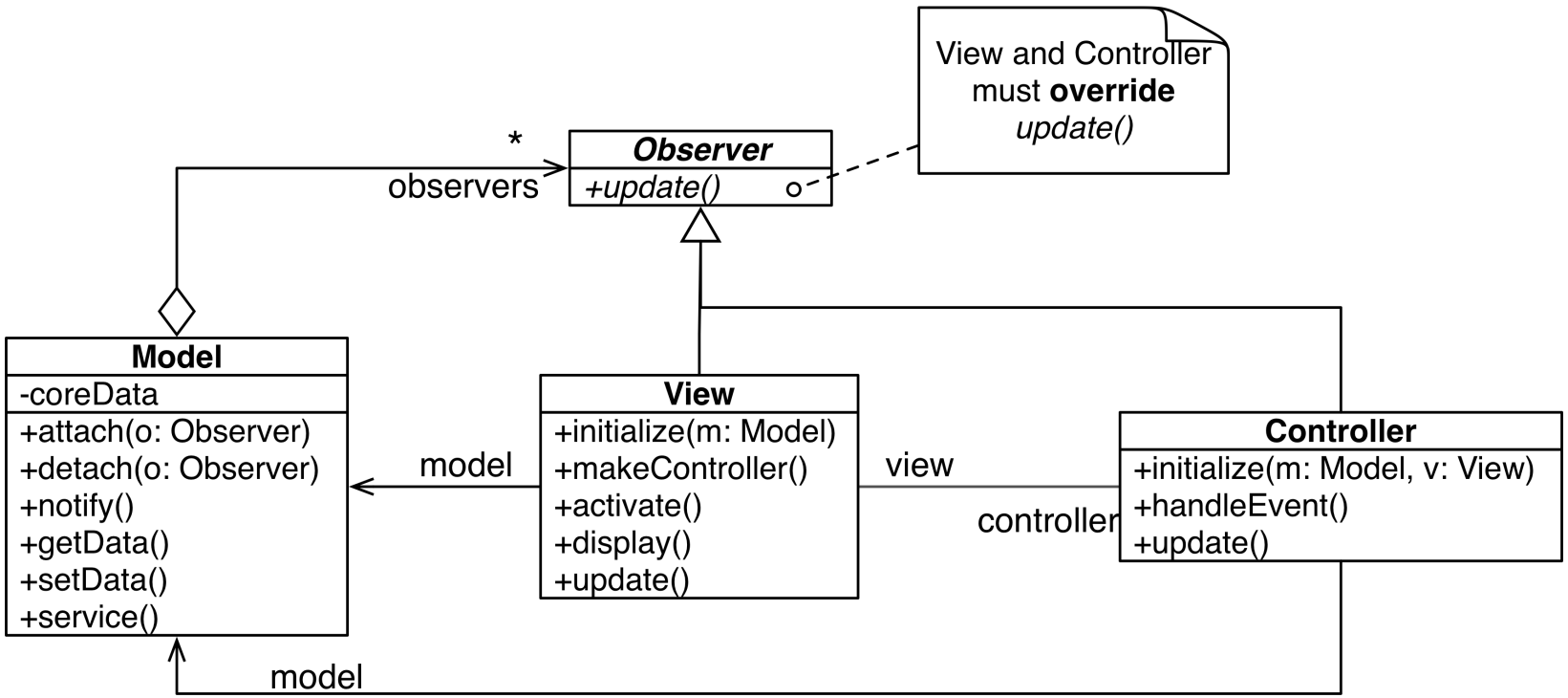
Behavioural Pattern: Strategy

```
12      //Client code
        ...
14      void runSimulationTimestep(){
            ...
16          context.executeAlgorithm()
            ...
18      }
        ...
```

```
        class PolicyChanger {
346         ...
            if(matricesWillSoonBeBig() == true)
348             context.selectStrategy(strategies.BIG_MATRIX_ALGORITHM);
            else
350             context.selectStrategy(strategies.SMALL_MATRIX_ALGORITHM);
        ...
352 };
```

Model – View – Controller

And it's getting complicated...



Potential pitfalls

- Just because it's a solution, it doesn't mean it's the solution (don't forget to think for yourself!)
- The complicated way vs. the simple way (first code - then see where you have a problem)
- What about performance (when you have hundreds of classes calling each other)?

Further reading

All patterns fetched from lecture *Patterns in Software Engineering*, running every Wintersemester (compulsory for CSE students, 3rd semester!)

More explanation on what Cohesion and Coupling is (source of images):
`http://thebojan.ninja/2015/04/08/high-cohesion-loose-coupling/`

Two quite complete books on design patterns, in `C++` and `java` respecitvely:

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
Design patterns: elements of reusable object-oriented software.
Pearson Education, 1994.
`http://www.uml.org.cn/c++/pdf/DesignPatterns.pdf`.

Eric Freeman, Elisabeth Robson, Bert Bates, and Kathy Sierra.
Head first design patterns.
" O'Reilly Media, Inc.", 2004.
`http://www.sws.bfh.ch/~amrhein/ADP/HeadFirstDesignPatterns.pdf`.

How and why you can**not** partially specialise function templates (for tutorial exercise):
`http://www.gotw.ca/publications/mill17.htm`