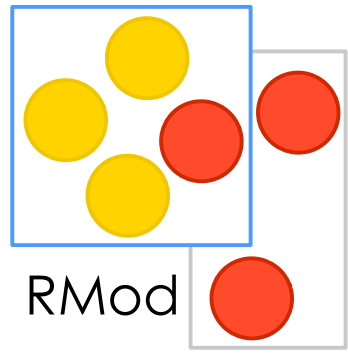
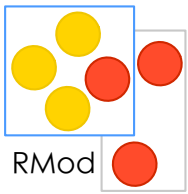


Elements of Design: Inheritance and Composition

Stéphane Ducasse
stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr/

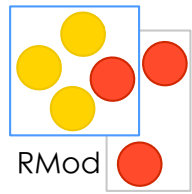
A Formatting Text Editor



With several possible algorithms

formatWithTex
formatFastColoring
formatSlowButPreciseColoring

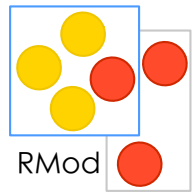
What's the best design?



How do we compare designs?

Identify your own criteria

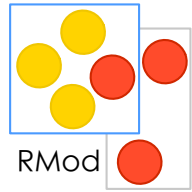
Code Smells



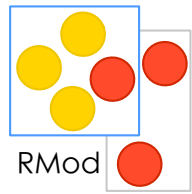

```
Composition » reformat
    formatting == #Simple
        ifTrue: [ self formatWithSimpleAlgo]
        ifFalse: [ formatting == #Tex
                    ifTrue: [self formatWithTex]

                ....]
```

Solutions



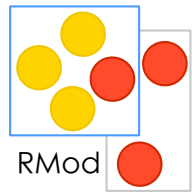
Inheritance?



Inheritance may not be the solution since:

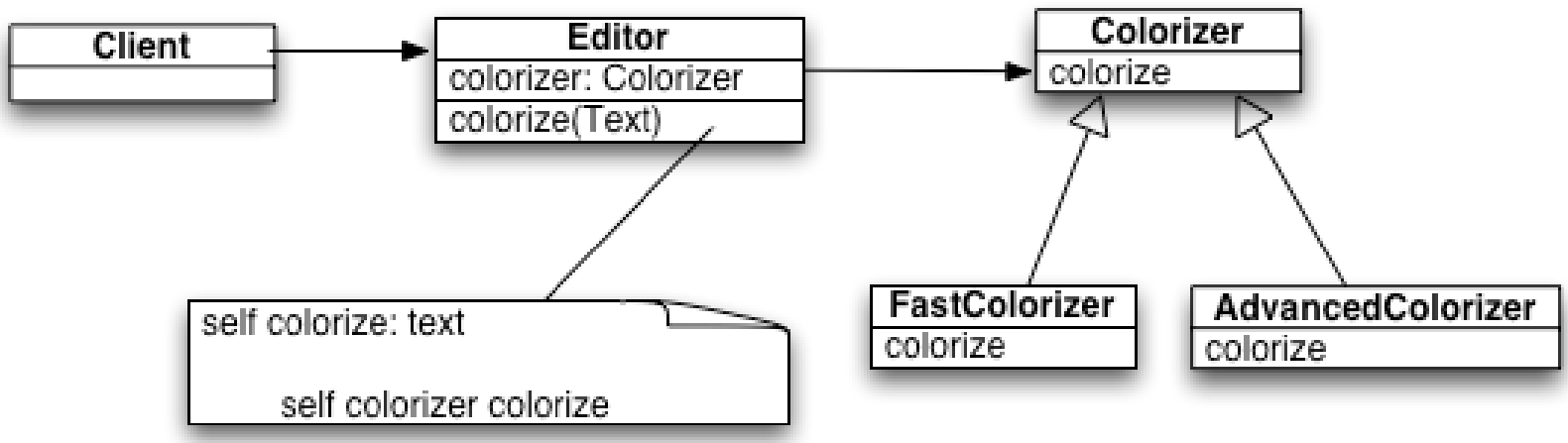
- you have to create objects of the right subclass
- it is difficult to change the policy at run-time
- you can get an explosion of subclasses, one implementing each combination of alternative functionalities
- no clear identification of responsibilities

Inheritance vs. Composition



- Inheritance is not a panacea
  - Require class definition
  - Require method definition
  - Extension should be prepared in advance
  - No run-time changes
    - Cannot load a new colorizer
  - Proliferation of subclasses
    - Example: editor with spell-checkerS, colorizerS, mail-readerS …

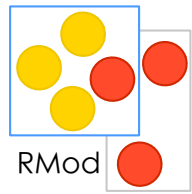
Solution: delegate to other objects



myEditor setColorizer: FastColorizer new.
myEditor setColorizer: AdvancedColorizer new.

STRATEGY design pattern

Composition Analysis



- Pros
  - Possible to change at run-time
  - Clear responsibility
  - Responsibilities broken into small chunks
  - Clear interaction protocol
- Cons
  - New class
  - Delegation must be explicit ⇒ more methods
    - in contrast, inheritance is implicit
  - Access to **self** is problematic in delegates
    - see Beck's SELF DELEGATION pattern