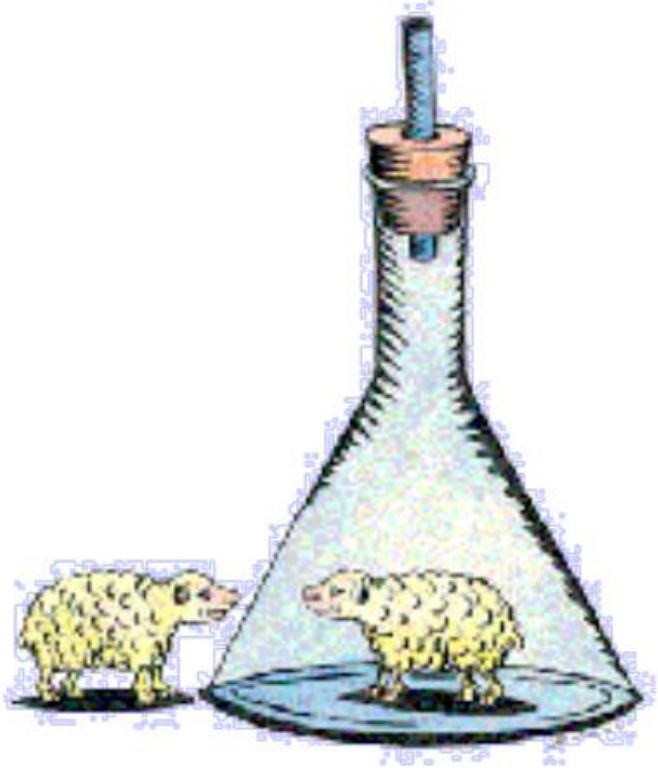
SOFTWARE RE-ENGINEERING

Unit 6

DESIGN PROBLEMS

Unclear & complicated

Duplicated (code clones)



CODE SMELLS

- A code smell is a hint that something has gone wrong somewhere in your code.
- If it stinks, change it
  - Duplicated Code
  - Long Method
  - Large Class
  - Long Parameter List
  - Divergent Change
  - Shotgun Surgery
  - Feature Envy

SMELL 1: LONG METHOD

- The longer a method is, the more difficult it is to understand it.
  - When is a method too long?
  - Heuristic: > 10 LOCs (?)

- How to detect?
  - Visualize LOC metric values of methods
  - “Method Length Distribution View”

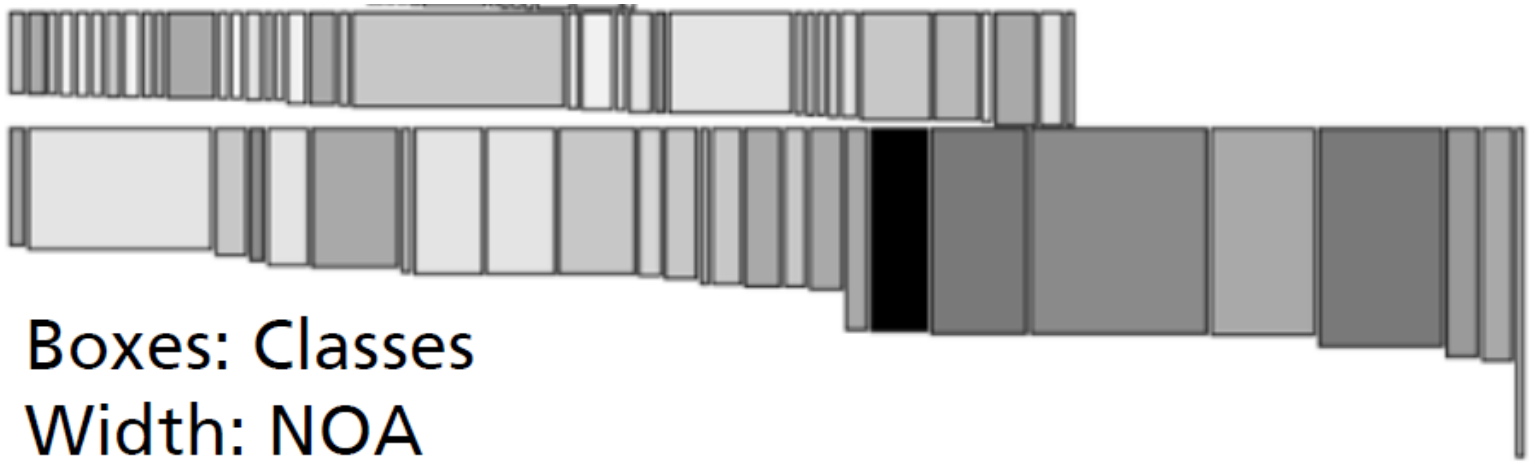
METHOD LENGTH DISTRIBUTION

SMELL 2: SWITCH STATEMENT

- Problem is similar to code duplication
  - Switch statement is scattered in different places
- How to detect?
- Visualize McCabe Cyclomatic Complexity metric to detect complex methods
  - “Method Complexity Distribution View”

SMELL 3: SYSTEM HOTSPOTS

- Classes that contain too much responsibilities
  - When is a class too large?
  - Heuristic: > 20 NOM
- How to detect?
  - Visualize number of methods (NOM) and sum of lines of code of methods (WLOC)
  - "System Hotspots View"

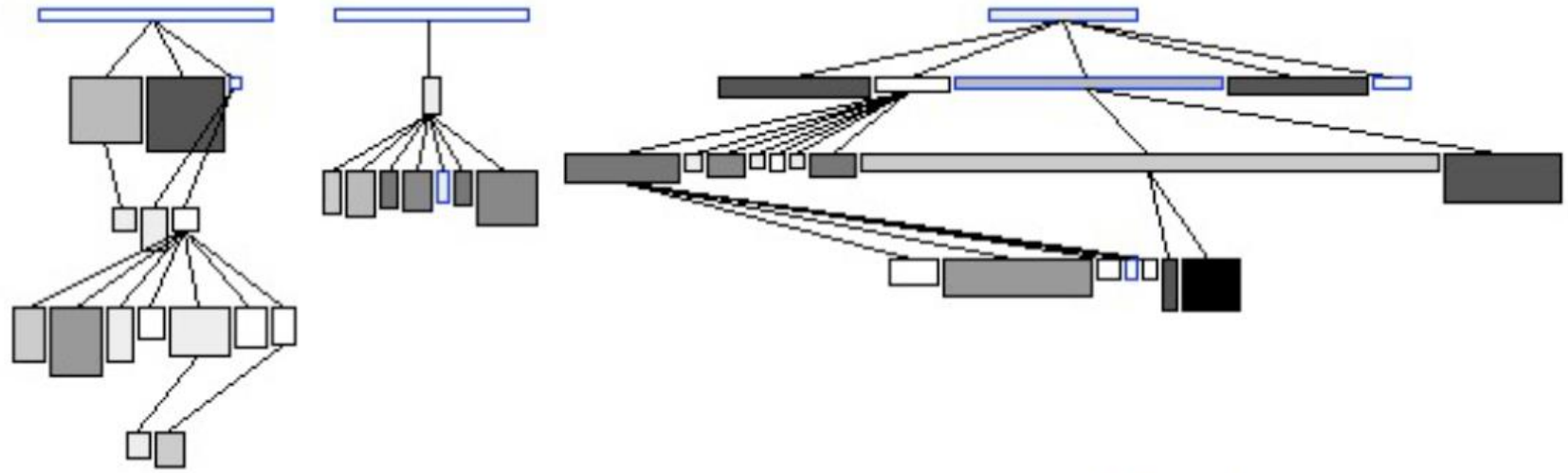


Boxes: Classes
Width: NOA
Height: NOM

SMELL 4: LAZY SUB-CLASS

- A class that is not doing enough to pay for itself should be eliminated
- How to detect?
  - Visualize inheritance structure with number of methods added (NMA),
  - overridden (NMO), and extended (NME)
  - “Inheritance Classification View”

INHERITANCE CLASSIFICATION



Metrics:
Boxes: Classes
Edges: Inheritance
Width: NMA
Height: NMO
Color: NME

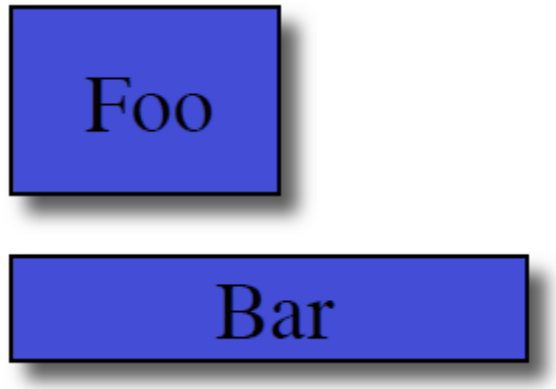
UNDERSTANDING EVOLUTION

- Changes can point to design problems
  - “Evolutionary Smells”
- But
  - Overwhelming complexity
  - How can we detect and understand changes?

- **Solutions**
  - The Evolution Matrix
  - The Kiviat Graphs

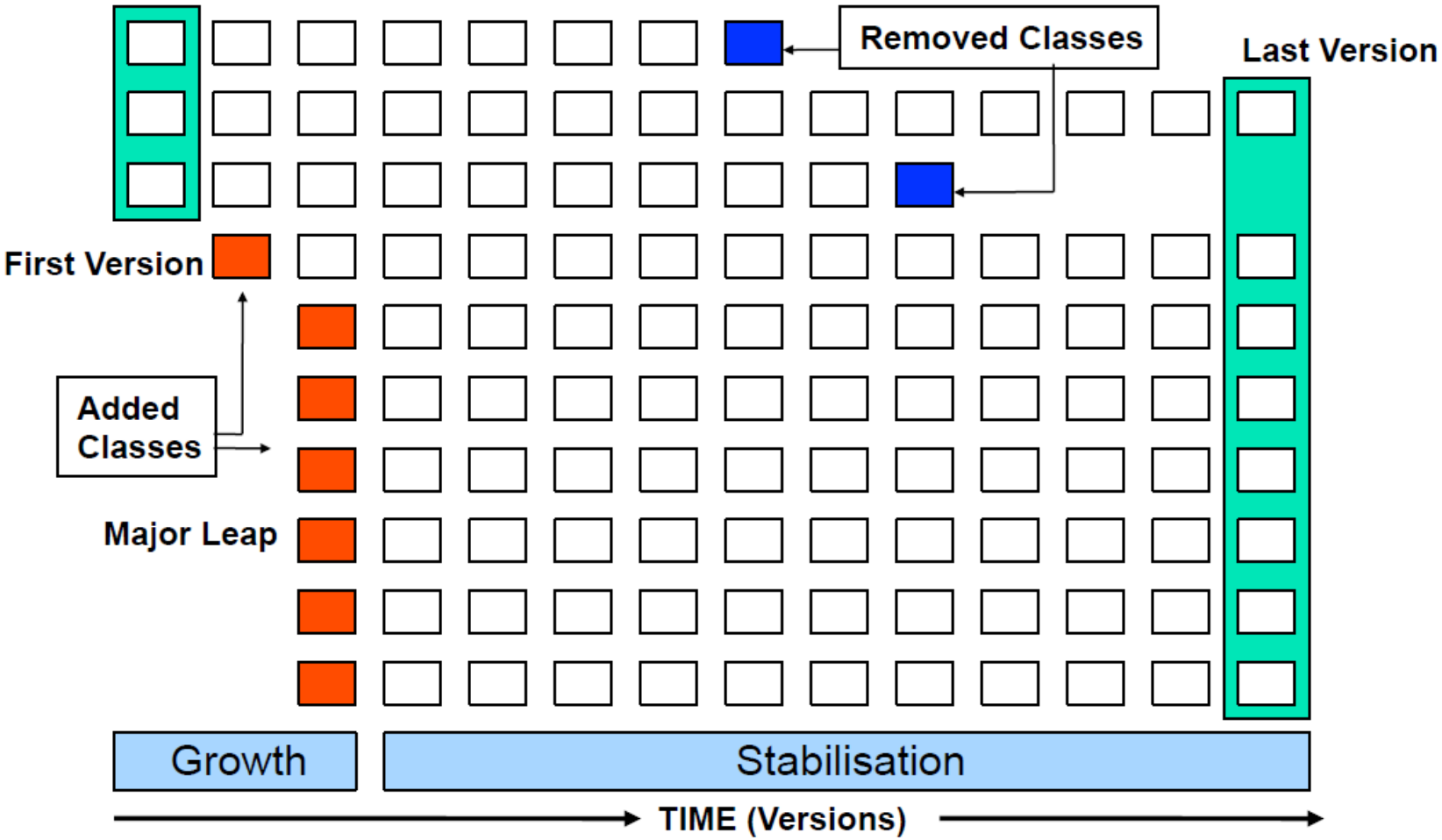
VISUALIZING CLASS EVOLUTION

- Visualize classes as rectangles using for
- width and height the following metrics:
  - NOM (number of methods)
  - NOA (number of attributes)



- The Classes can be categorized according to their “personal evolution” and to their “system evolution”
  - -> Evolution Patterns

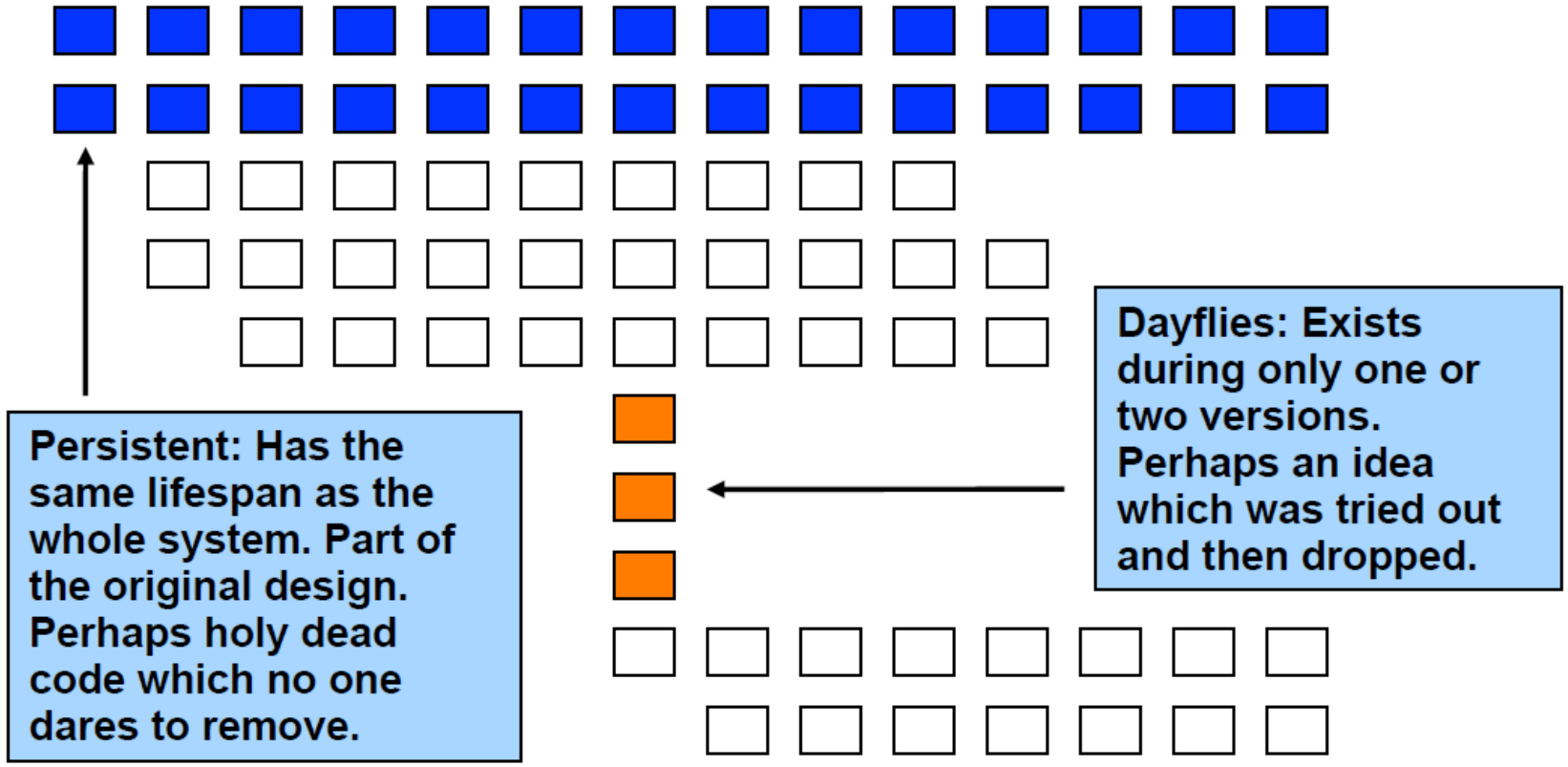
THE EVOLUTION MATRIX



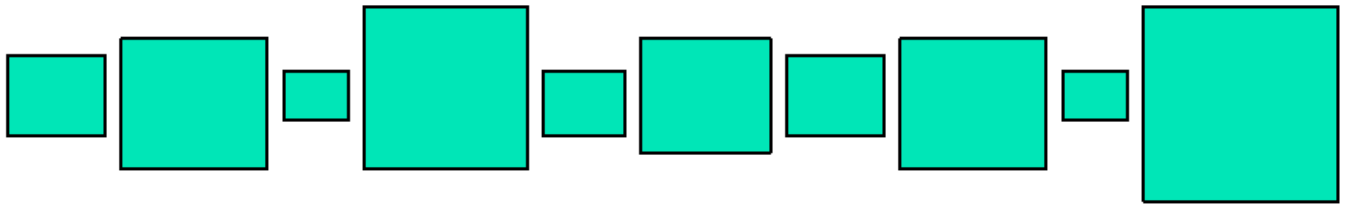
EVOLUTION PATTERNS & SMELLS

- Day-fly (Dead Code)
- Persistent
- Pulsar (Change Prone Entity)
- SupernovaWhite Dwarf (Dead Code)
- Red Giant (Large/God Class)
- Idle (Dead Code)

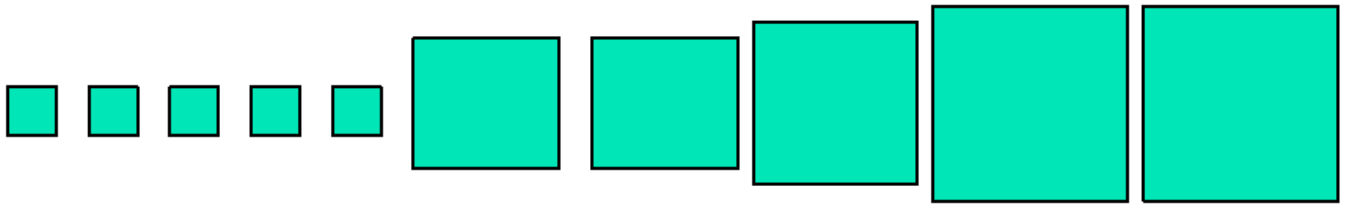
PERSISTENT / DAYFLY



PULSAR / SUPERNOVA

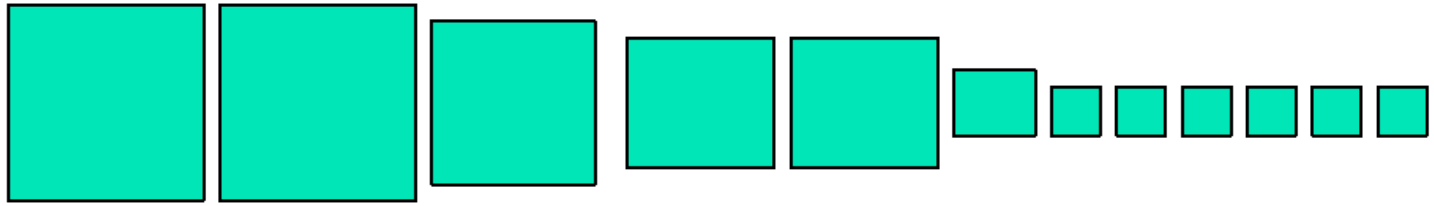


Pulsar: Repeated Modifications make it grow and shrink.
System Hotspot: Every System Version requires changes.

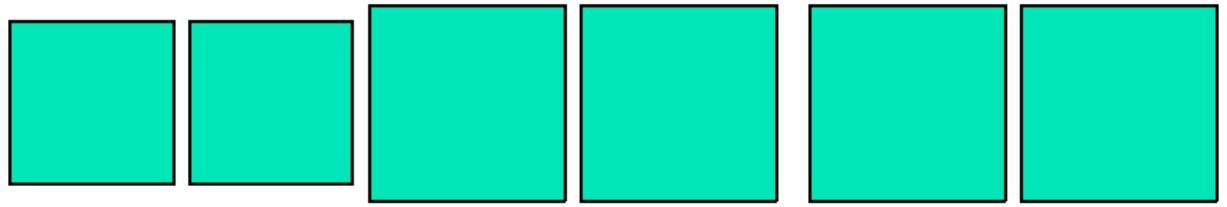


Supernova: Sudden increase in size. Possible Reasons:
- Massive shift of functionality towards a class.
- Data holder class for which it is easy to grow.
- *Sleeper*: Developers knew exactly what to fill in.

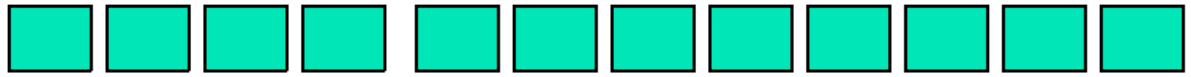
WHITE DWARF / RED GIANT / IDLE



White Dwarf: Lost the functionality it had and now trundles along without real meaning. Possibly dead code -> Lazy Class.



Red Giant: A permanent god (large) class which is always very large.



Idle: Keeps size over several versions. Possibly dead code, possibly good code.

EVALUATION: EVOLUTION MATRIX

- Pros
  - Understand the evolution of a system in terms of size and growth rate
  - Introduction of new classes
  - Remove of classes
  - Detection of Evolution Patterns & Smells
  - Dayflight, Persistent, White Dwarf, ...
- Cons
  - Scalability
  - Limited to 3 metric values per glyph
  - Fragile regarding the renaming of classes
    - What if the name of a class was changed?

EXTENDED POLYMETRIC VIEWS

- Goal:
  - Visualize n metric values of m releases
  - More semantic in graphs
  - More flexibility to combine metric values
- Solution:
  - Kiviat Diagrams (Radar Charts)
  - Each ray represents a metric
  - Encode releases with different colors