


# Abstraction Strata in Software Design

Amnon H. Eden  
Department of Computer Science, University of Essex  
Center for Enquiry, Amherst, New York

SEUJP-FC, May 21, 2004




## Contents

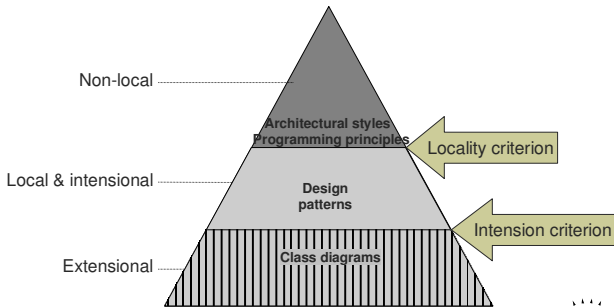
- Abstractions
  - Programming principles
  - Architectural styles
  - Design patterns
- The Local/Intensional hierarchy
  - The Locality criterion
  - The Intension criterion
- Conclusions

2

Abstraction strata in software design




## The Intension/Locality hierarchy



3

Abstraction strata in software design




## Programming principles

- Rules that govern all programs in a given “paradigm”
- Example: *Information Hiding*

$\forall c, m, x \bullet$   
 $Member(m, c) \wedge Access(x, m) \Rightarrow$   
 $Public(m) \vee Member(x, m) \vee Friend(x, m)$

4

Abstraction strata in software design



### [ Information Hiding (Cont.) ]


- In C++, *information hiding* is enforced by the compiler

```

template <class T> class Stack {
public:
    void push(T);
    // ...
private:
    T * theStack;
    int size;
};
Stack<complex<float> > si;

int foo() {
    return si.size;
}
    
```

← Compilation error

5  Abstraction strata in software design

### [ Example: Universal base class ]


- All classes must inherit (possibly indirectly) from class `Object`

$\forall c \bullet$   
 $Class(c) \Rightarrow Inherit^*(c, Object)$

- For example: Smalltalk, Eiffel, Java


```

graph TD
    Object --> Collection
    Object --> Magnitude
    Object --> Undefined
    
```

6  Abstraction strata in software design

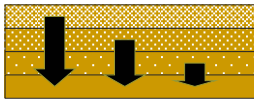
### [ Architectural styles ]


- Styles:** A catalogue of configurations [Perry & Wolf 92, Garlan & Shaw 96]
- Each style** describe –
  - Types of components
  - Types of connectors
  - Topology (constraints on possible compositions)
- Non-locality:** “*Architectural designs are typically concerned with the entire system*” [Monroe et. al 97]

7  Abstraction strata in software design

### [ Example: Layered Architecture ]

- AKA: **Abstract Machine Model**
- Principles:
  - Each “entity” belongs to a *layer*  $\forall e \exists !k \in \{N\} \bullet Layer(e) = k$
  - Every entity may only depend on modules in same or lower layers  $\forall x, y \bullet Depends(x, y) \Rightarrow Layer(x) \geq Layer(y)$



8  Abstraction strata in software design

## [ Layered Architecture II ]

Examples:

ISO Communication Protocols

UNIX Operating System

9 Abstraction strata in software design

## [ Example: Pipes and Filters ]

- “Each component has a set of inputs and a set of outputs.
- “A component reads streams of data on its inputs and produces streams of data on its outputs.” [Garlan & Shaw 93]

[Dean & Cordy 95]

10 Abstraction strata in software design

## [ Other architectural styles ]

Broker architecture

Broadcast-control

11 Abstraction strata in software design

## [ Design patterns ]

- Motivation:
  - A vocabulary of design decisions
  - Dissemination of “good design” practices
- Definition:
  - “Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution.” [Jim Coplien]

12 Abstraction strata in software design

### [ Example: *Recursive Composite* design pattern ]

- **Intent:** “Compose objects into tree structures to represent part-whole hierarchies.
  - “treat individual objects and compositions of objects uniformly.”
- **Example:** Files and directories

13 Abstraction strata in software design

### [ *Recursive Composite II* ]

The file system:

Collaboration diagram:

14 Abstraction strata in software design

### [ *Recursive Composite III* ]

- Structure:

- Participants:
  - Component
  - Leaf
  - Composite
  - Client
- Collaborations:
  - ...

15 Abstraction strata in software design

### [ *Recursive Composite IV* ]

- Solution formulated in Z:
  - [CLASS]
  - [SIG]
  - [METHOD]

Recursive Composite

*Composite, Component* : CLASS

*Leaves* :  $\mathbb{P}$  CLASS

*Operation* : SIG

$(Composite, Component) \in Inherit$

$\forall x \in Leaves \bullet (x, Component) \in Inherit$

$(Operation \otimes Composite, Operation \otimes Component) \in Invoke$

*ANY* ::= CLASS  $\cup$  METHOD

*RefToMany, Inherit* : CLASS  $\leftrightarrow$  CLASS

*Member* : ANY  $\leftrightarrow$  CLASS

*Invoke* : METHOD  $\leftrightarrow$  METHOD

*Signature* : METHOD  $\rightarrow$  SIG

$\otimes$  : SIG x CLASS  $\rightarrow$  METHOD

16 Abstraction strata in software design

## [ Example: *Strategy pattern* ]

- Intent
  - Let each member of a family of algorithms vary independently from clients that use it
- **Constraints:**
  - input depends on the algorithm
  - calculations are based on the client's abstraction (not using client's implementation or global data)

17

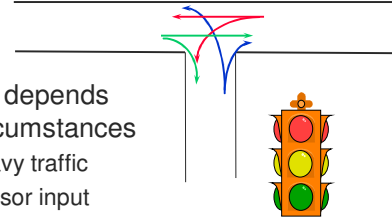
Abstraction strata in software design



## [ *Strategy II* ]

- Motivating example: Traffic lights controller

- Policy depends on circumstances
  - Heavy traffic
  - Sensor input
  - ...



18

Abstraction strata in software design



## [ *Strategy III* ]

- Controller's policies include:
  - "Dumb" policy: change the green route every 5 seconds
  - Midnight policy: change to green whenever a "sensor" detects a vehicle
  - Rush hour policy: double the "green time" in the busy route
  - ...

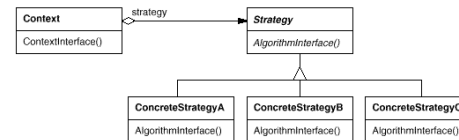
19

Abstraction strata in software design



## [ *Strategy IV* ]

- Solution:
  - Encapsulate each strategy in a separate class
  - Context's interface provides all necessary inputs



- Consequences:
  - Easy to add new strategy (or remove an existing one, etc.)
  - Easy to factor out similar strategies (using inheritance)

20

Abstraction strata in software design




## [ Strategy V ]

- Solution formulated in Z:

Strategy
$Context, Strategy: CLASS$
$ConcreteStrategies: \mathbb{P} CLASS$
$Algorithm: SIG$
$ContextInterface: \mathbb{P} SIG$
$(Context, Strategy) \in Member$
$\forall c \in ConcreteStrategies \bullet$ $(c, Strategy) \in Inherit$
$\forall c \in ConcreteStrategies$ $\exists s \in ContextInterface \bullet$ $(Algorithm \otimes c, s \otimes Context) \in Invoke$


21

Abstraction strata in software design 

## [ Questions ]

- “Design” Vs. “Architecture”
  - Is the difference just a matter of scale?
  - Is the difference qualitative or merely quantitative?
- Language: How can we specify architectural styles? Design patterns?
  - Can we use Class Diagrams?
  - Can we use other UML language?


22

Abstraction strata in software design 

## [ The Locality criterion ]

- Statement  $\varphi$  is *local* iff it is preserved under any expansion.
- Non-local statements:
  - *Information hiding*
  - *Universal base class*
  - *Layered architecture*
  - *Pipes and Filters*


23

Abstraction strata in software design 

## [ The Locality criterion (Cont.) ]

- Local statements:
  - *Recursive composite* design pattern
  - *Strategy* design pattern
  - Class diagrams

24

Abstraction strata in software design 

## The Intension criterion

- Statement  $\varphi$  is *extensional* iff it is preserved both under any expansion and under any reduction that preserves its signature.
  - (*Signature*: Constant and relation symbols)

25

Abstraction strata in software design



## Conclusions

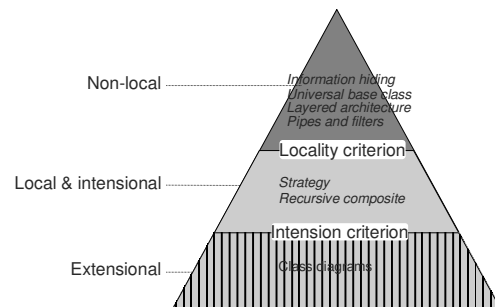
- Intensional statements:
  - Any non-local statement (e.g., programming principles, architectural styles)
  - *Recursive composite* design pattern
  - *Strategy* design pattern
- Extensional statements
  - Class diagrams
  - Any program (depending on the abstraction used)

26

Abstraction strata in software design



## The Intension/Locality hierarchy revisited



27

Abstraction strata in software design



## Conclusions

- Architectural styles, programming (or design) principles share the same abstraction level
- UML class diagrams are too impoverished to express patterns and styles

28

Abstraction strata in software design



## Open questions

- Architectural mismatch
- Extend *locality* to behavioural models

29

Abstraction strata in software design



## Bibliography

- A. H. Eden, R. Kazman. "**Architecture, Design, Implementation**". *Proceedings of the 25th International Conference on Software Engineering (ICSE' 25)*, May 2003, Portland, OR.
- E. Gamma, R. Helm, R. Johnson, J. Vlissides (1994). **Design Patterns: Elements of Reusable Object Oriented Software**. Addison-Wesley.
- D. Garlan, M. Shaw. "**An Introduction to Software Architecture**." In V. Ambriola, G. Tortora, eds., *Advances in Software Engineering and Knowledge Engineering 1993*, Vol. 2, pp. 1—39. New Jersey: World Scientific Publishing Company.
- D. E. Perry, A. L. Wolf. "**Foundation for the Study of Software Architecture**." *ACM SIGSOFT Software Engineering Notes* 1992, 17 (4), pp. 40—52.

30

Abstraction strata in software design



## References

- T. R. Dean, J. R. Cordy. "**A Syntactic Theory of Software Architecture**." *IEEE Trans. on Software Engineering* 21 (4), Apr. 1995, pp. 302—313.
- R. T. Monroe, A. Kompanek, R. Melton, D. Garlan. "**Architectural Styles, Design Patterns, and Objects**." *IEEE Software* 14 (1), Jan. 1997, pp. 43—52.

31

Abstraction strata in software design

