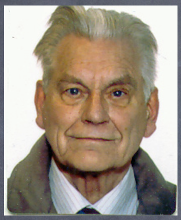
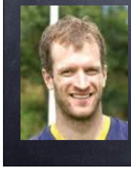
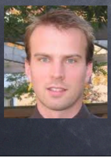
Heart of Technology: DCI

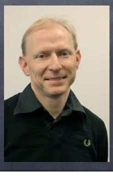
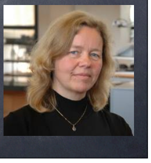
Jim Coplien
Gertrud & Cope
Scrum Training Institute
cope@gertrudandcope.com

Outline

- Credits
- Your grandfather's OO and the testing myth
- Implementing the Swarm vision
- It's about individuals and interactions
- Serious cross-cutting
- The form of function
- Details



The Original Object Vision

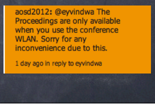
- As long as every object does its job well, the system will do its job well
- Shades of emergent system behavior

Classes?

- What is a large class?
- To *understand* or *test* a class, you must flatten the inheritance hierarchy.
- Most deep derived classes are therefore thousands of lines long
- Don't sweat class size: it's irrelevant

A very scary thought...

- "We could imagine taking the internet as a model for doing software modules. Why don't people do it?"

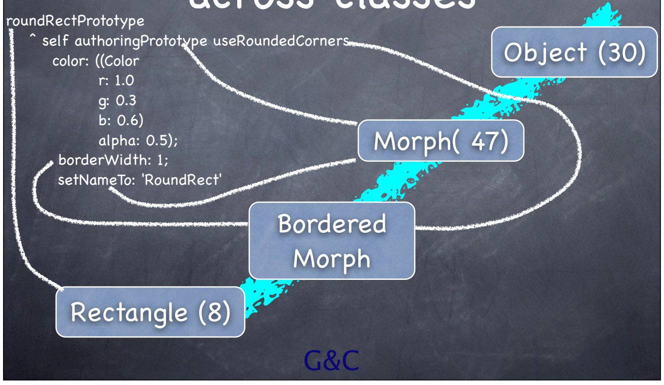

aosd2012: @eyvindwa The Proceedings are only available when you use the conference WLAN. Sorry for any inconvenience due to this.

1 day ago in reply to eyvindwa

How many classes?

```
roundRectPrototype
    ^ self authoringPrototype useRoundedCorners
        color: ((Color
                r: 1.0
                g: 0.3
                b: 0.6)
                alpha: 0.5);
        borderWidth: 1;
        setNameTo: 'RoundRect'
```

Object Behavior cuts across classes



```
roundRectPrototype
    ^ self authoringPrototype useRoundedCorners
        color: ((Color
                r: 1.0
                g: 0.3
                b: 0.6)
                alpha: 0.5);
        borderWidth: 1;
        setNameTo: 'RoundRect'
```

More on the Agile myths

- "[T]he results didn't support claims for lower coupling and increased cohesion with TDD"
  - Janzen & Saledian, "Does TDD Really Improve Software Design Quality," IEEE Software 25(2), 2008.
- "the effect of TDD on program design was studied... an unwanted side effect can be that some parts of the code may deteriorate."
  - Siniaalto and Abrahamsson, "Does TDD Improve the Program Code? Alarming Results from a Case Study." Cee-Set 2007.

We don't test models

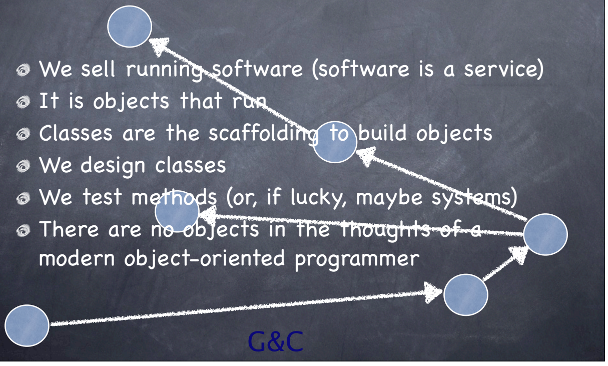
- If a Model test fails, what is the business consequence? Impossible to tell
- NO — we test use cases
- "It's easy to say what this class really does." Who cares?
- Software is never a product: it is a service

The Scatlology of Agile Architecture - Uncle Bob

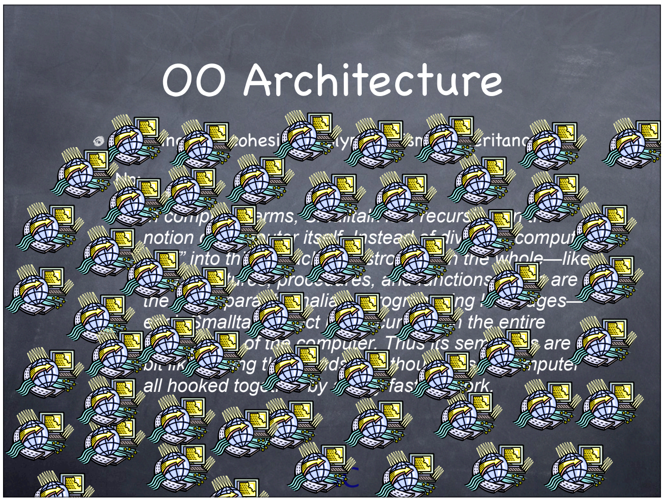
One of the more insidious and persistent myths of agile development is that up-front architecture and design are bad; that you should never spend time up front making architectural decisions. That instead you should evolve your architecture and design from nothing, one test-case at a time.

Pardon me, but that's Horse Shit.

OOP is about objects

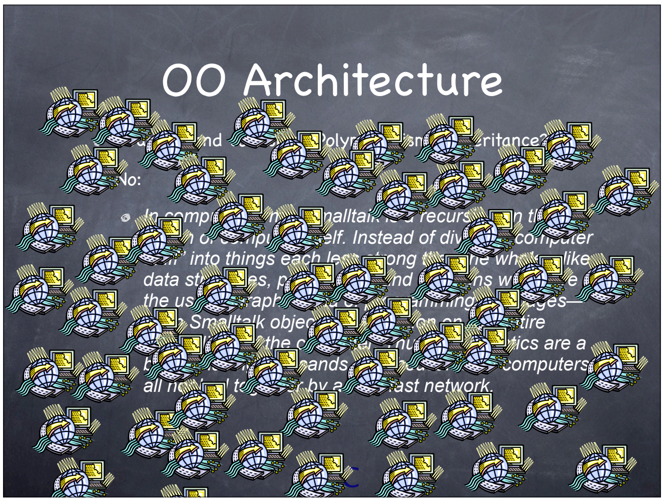


- We sell running software (software is a service)
- It is objects that run
- Classes are the scaffolding to build objects
- We design classes
- We test methods (or, if lucky, maybe systems)
- There are no objects in the thoughts of a modern object-oriented programmer



OO Architecture

- [illegible]ing and Cohesion? Polymorphism? Inheritance?
- No:
  - In computer terms, Smalltalk is a recursion on the notion of computer itself. Instead of dividing computer stuff” into things each less strong than the whole—like data structures, procedures, and functions which are the usual paraphernalia of programming languages—each Smalltalk object is a recursion on the entire possibilities of the computer. Thus its semantics are a bit like having thousands and thousands of computers all hooked together by a very fast network.



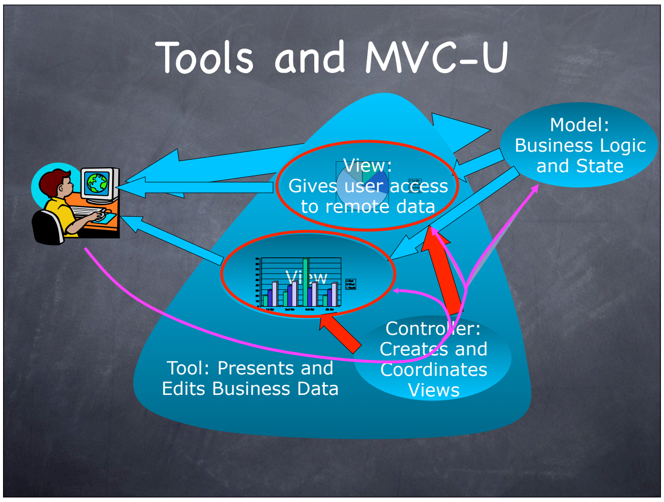
OO Architecture

- [illegible]ing and Cohesion? Polymorphism? Inheritance?
- No:
  - In computer terms, Smalltalk is a recursion on the notion of computer itself. Instead of dividing computer stuff” into things each less strong than the whole—like data structures, procedures, and functions which are the usual paraphernalia of programming languages—each Smalltalk object is a recursion on the entire possibilities of the computer. Thus its semantics are a bit like having thousands and thousands of computers all hooked together by a very fast network.

What is an object?

- State
- Identity
- Behavior
- Represents a stakeholder mental model
- Wrappers destroy this!

Tools and MVC-U

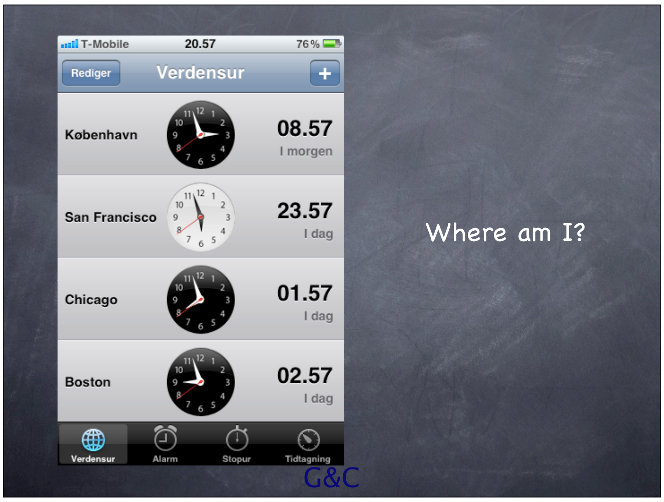


Listen to the tests?

There are two ways of constructing a software design. One is to make it so simple that there are obviously no deficiencies, and the other is to make it so complicated that there are no obvious deficiencies.

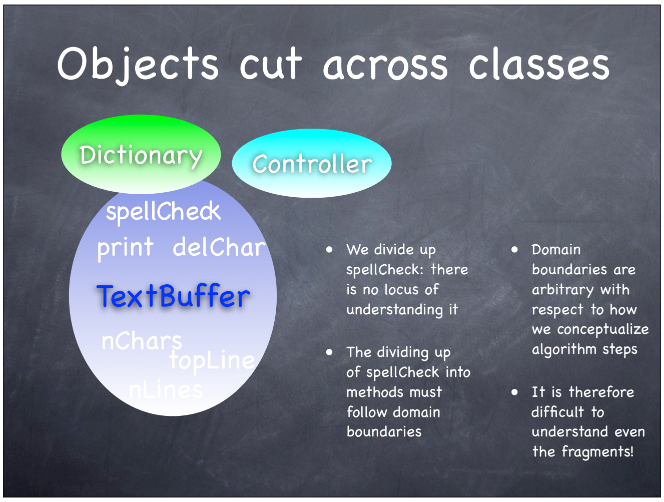
— Tony Hoare

(Feel the pain)



København	08.57	I morgen
San Francisco	23.57	I dag
Chicago	01.57	I dag
Boston	02.57	I dag

Where am I?



Objects cut across classes

Dictionary

Controller

spellCheck

print delChar

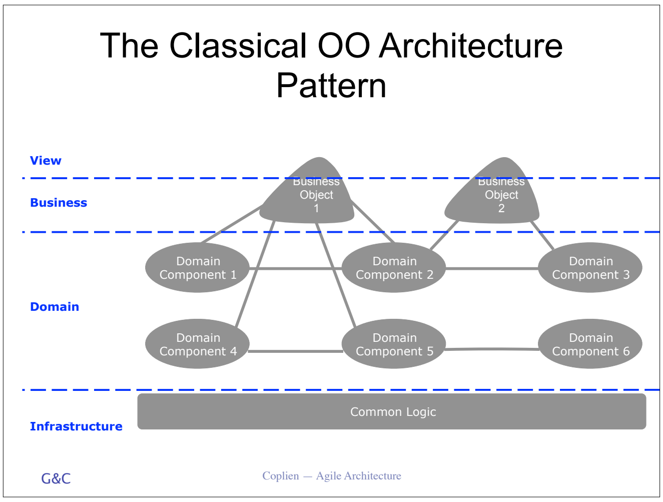
TextBuffer

nChars

topLine

nLines

- We divide up spellCheck: there is no locus of understanding it
- The dividing up of spellCheck into methods must follow domain boundaries
- Domain boundaries are arbitrary with respect to how we conceptualize algorithm steps
- It is therefore difficult to understand even the fragments!



The Classical OO Architecture Pattern

View

Business

Business Object 1

Business Object 2

Domain

Domain Component 1

Domain Component 2

Domain Component 3

Domain Component 4

Domain Component 5

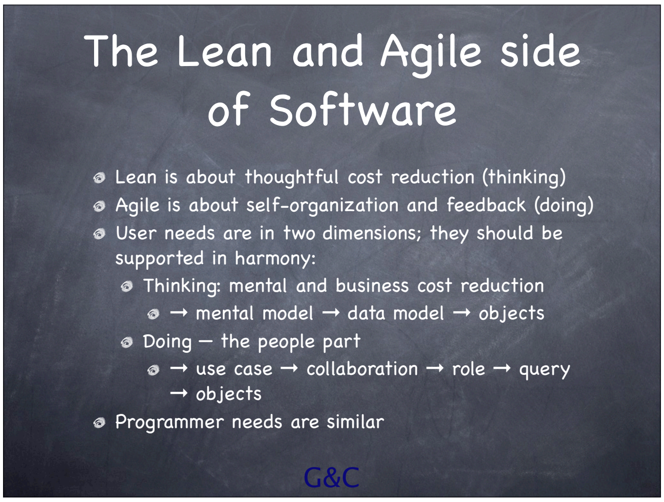
Domain Component 6

Infrastructure

Common Logic

G&C

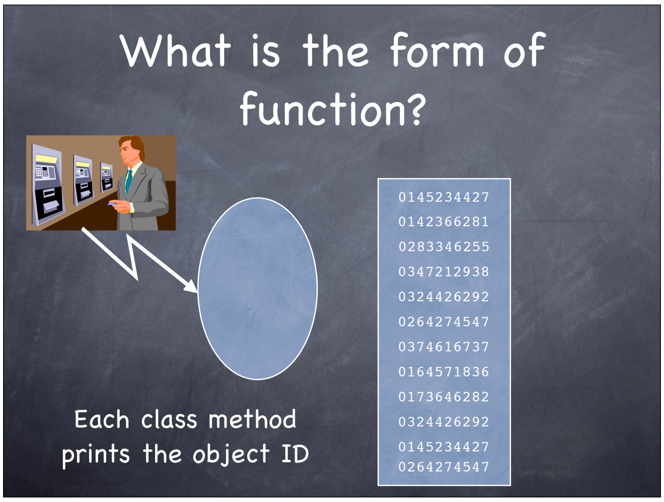
Coplien — Agile Architecture



The Lean and Agile side of Software

- Lean is about thoughtful cost reduction (thinking)
- Agile is about self-organization and feedback (doing)
- User needs are in two dimensions; they should be supported in harmony:
  - Thinking: mental and business cost reduction
    - → mental model → data model → objects
  - Doing — the people part
    - → use case → collaboration → role → query → objects
- Programmer needs are similar

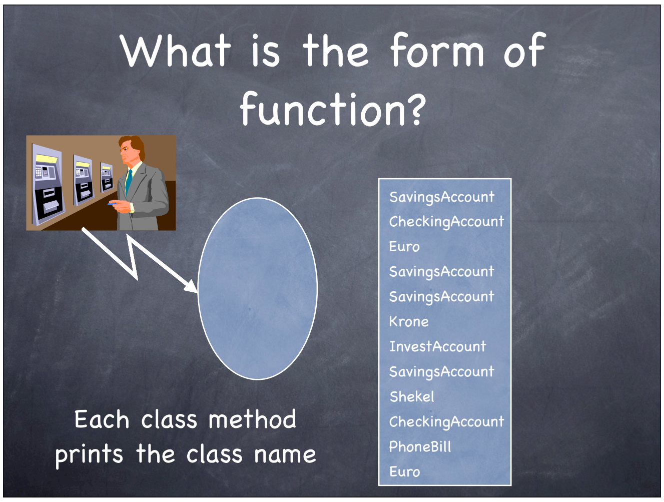
G&C



What is the form of function?

Each class method prints the object ID

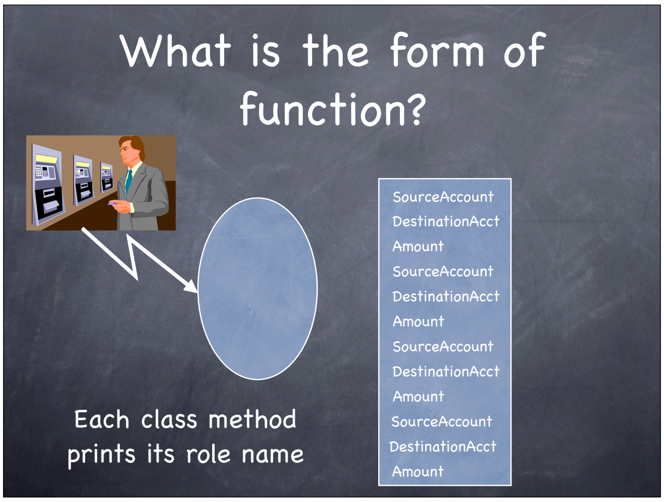
0145234427
0142366281
0283346255
0347212938
0324426292
0264274547
0374616737
0164571836
0173646282
0324426292
0145234427
0264274547



What is the form of function?

Each class method prints the class name

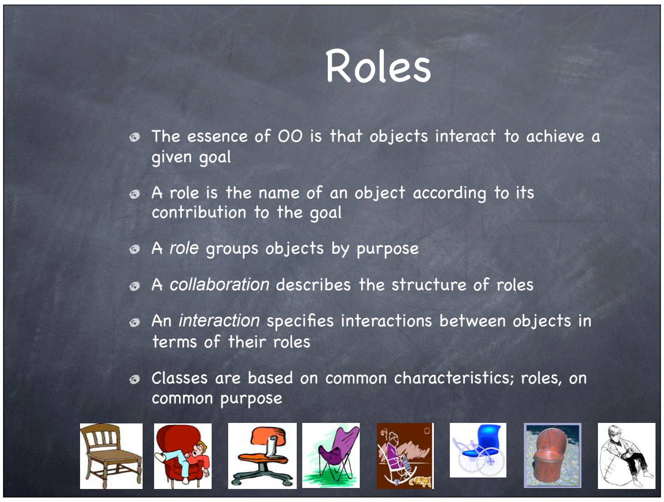
SavingsAccount
CheckingAccount
Euro
SavingsAccount
SavingsAccount
Krone
InvestAccount
SavingsAccount
Shekel
CheckingAccount
PhoneBill
Euro



What is the form of function?

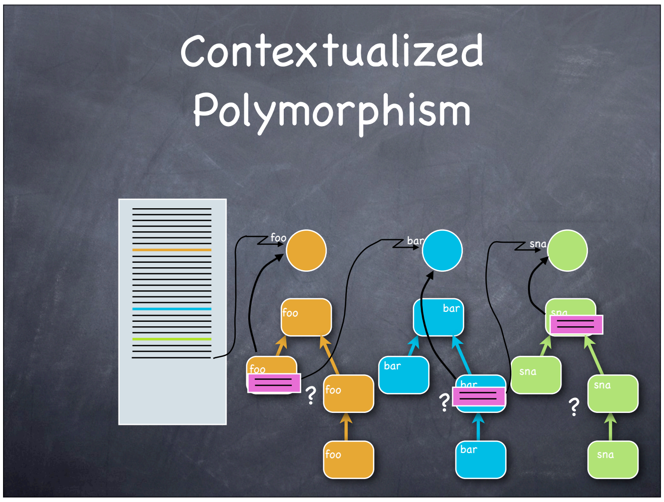
Each class method prints its role name

SourceAccount
DestinationAcct
Amount
SourceAccount
DestinationAcct
Amount
SourceAccount
DestinationAcct
Amount
SourceAccount
DestinationAcct
Amount

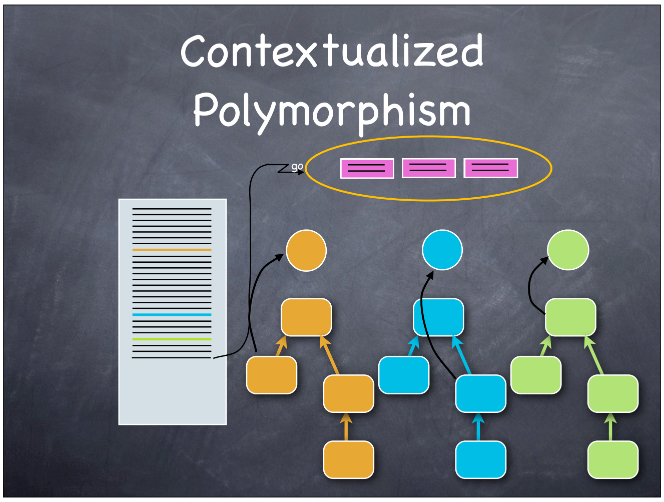


Roles

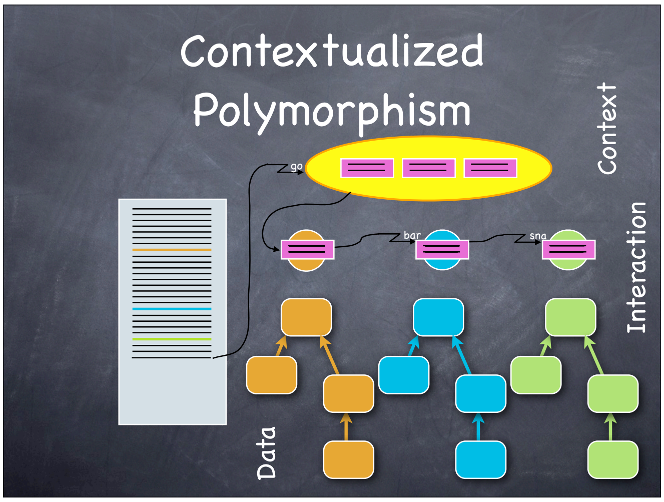
- The essence of OO is that objects interact to achieve a given goal
- A role is the name of an object according to its contribution to the goal
- A *role* groups objects by purpose
- A *collaboration* describes the structure of roles
- An *interaction* specifies interactions between objects in terms of their roles
- Classes are based on common characteristics; roles, on common purpose



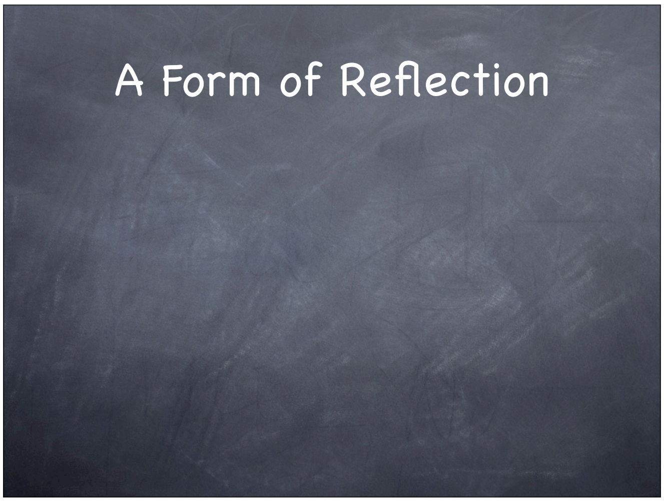
Contextualized Polymorphism



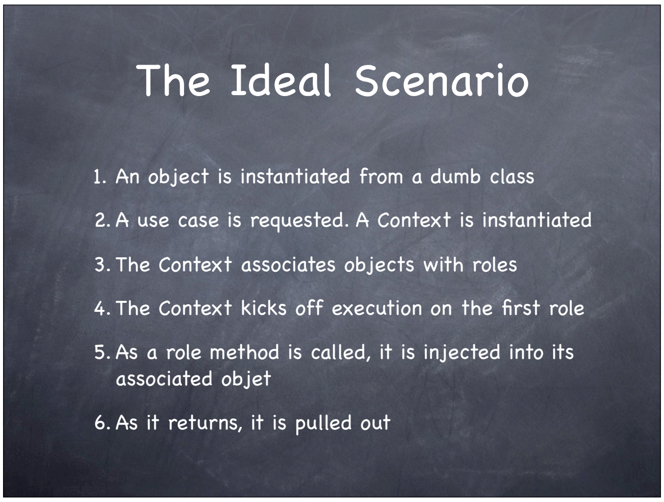
Contextualized Polymorphism



Contextualized Polymorphism



A Form of Reflection



The Ideal Scenario

1. An object is instantiated from a dumb class
2. A use case is requested. A Context is instantiated
3. The Context associates objects with roles
4. The Context kicks off execution on the first role
5. As a role method is called, it is injected into its associated objet
6. As it returns, it is pulled out

Variants

- Inject role methods into objects at Context instantiation and leave them there (Ruby)
- Pre-associate roles with the classes of the objects that will play those roles; fully type-safe (C++)
- Manually inject methods into objects at Context instantiation time; pull them out at Context destruction (Python)
- . . .

Start with good Domain Analysis

- Basic domain classes (yes, classes) are dumb
- Use subtyping if you like, but it has a cost in code comprehension

Introduce use cases piecemeal

- Program roles and their methods
- Contexts become the loci of understanding behavior

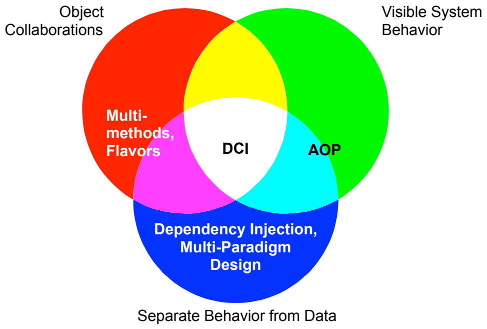
Map onto a programming language

- Most modern languages fake it well enough
- Can choose from across the spectrum of anarchical to tyrannical type systems
- Natural expression in Scala, very good in Ruby, possible in C++
- Marvin language allows native DCI programming

The usual retorts...

- Use multiple dispatch
  - That's chooses one algorithm based on multiple types, rather than multiple algorithms based on multiple types
- Use aspects
  - The cutpoints are still dictated by the class structure
- Use mix-ins
  - Close, but how do they talk to each other?
- Just use objects (e.g., self)
  - A good start, but not enough
- Use multi-paradigm design
  - Undesired decoupling and lack of cohesion

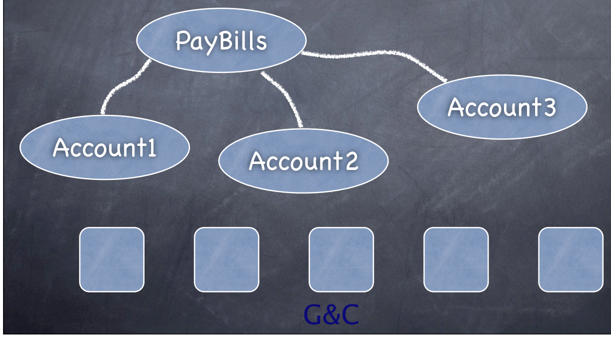
DCI and the Six Wise Men and the Elephant



A Generalization: Cascaded Contexts

- A Bank Account is not an object that encapsulates the state of the balance
- It is, instead, a collection of activities that compute the balance
- It is a collection of potential scenarios between the end user and the system...
- That is a use case: a Context

Generalizing Contexts



Some Metaphors

- AOP is syntax of local expression; DCI is system-level semantics
- AOP tunes an existing behavioural base; DCI provides such a base
- AOP gives syntactic help to one small aspect of cross-cutting; DCI undoes the cross-cutting

Realisation of These Values in the DCI Architecture

- Software has a Lean part and an Agile part
- The Agile part is the rapidly changing revenue generator and should be nourished
- DCI is about object thinking
- Object thinking supports human mental models, and DCI supports the behavioral mental models that complement MVC mental models of form

Conclusion

- Lean architecture reduces rework and cost
- Agile software production meets end user expectations
- MVC brings the human side of Agile beyond the team to the code
- DCI separates the shear layers to ease maintenance
- New processes and organizations amplify the benefits