Not your Grandfather’s Architecture
Taking Architecture into the Agile World

James O. Coplien
Gertrud&Cope, Mørdup, Denmark
What is architecture?

- Architecture is the essence of structure: *form*
  - Structure obfuscates form!
- Lean architecture: just-in-time delivery of functionality, just-in-time pouring material into the forms
- Agile architecture: one that supports change, end-user interaction, discovery, and ease of comprehension (of *functionality*)
What is its value?

- Architecture supports “what happens there”
- Habitable code — by the people who develop it and the people who use it
- Architecture is what makes code feel familiar
- A good architecture reduces waste and inconsistency — muda and mura
  - Less rework
  - System consistency
Architecture and OO

• OO is a paradigm — a way of talking about form
• OO’s foundations: to capture the end user’s mental models in the code
• OO captures
  – The entities (objects) that users know about
  – The classes that serve as sets of such objects
MVC: The Embodiment of the OO Vision

- User model -> into the code -> presented back to the user
- The goal of views is direct manipulation

The goal of the controller is to coordinate multiple views
Architecture is more than that

• The form of the business domain
  – What the system *is*
  – Domain *model* (as in MVC)
  – What the programmer cares about
  – Deliver as abstract base classes
  – Eric Evans’ *Domain-Driven Design*, *Multi-Paradigm Design for C++*

• The form of the system interactions
  – What the system *does*
  – *Role* models: OORAM
  – What the end user cares about
  – Has long eluded the OO crowd
Back to OO: Other forms in the end-user’s head

- Users think more about the roles played by the objects than the objects
  - What-the-system-does again!
  - Money transfer from a bank account: the roles are Source Account and Destination Account
  - Savings, checking, investment account objects can all take on these roles — so can your phone bill

- The association from roles to objects, for a given use case, is also part of the end user model
Yet a few more forms!

- How about the algorithm?
  - The algorithm also has form in the user’s head
    - Start transaction
    - Debit Source Account
    - Credit Destination Account
    - End transaction
  - In FORTRAN I could argue the correctness of program functionality; I can’t do that in Java
  - Object orientation has served the programmers (the discovery process, architecture) but not the end users and customers — and not quality (Hoare)
These forms beg a new architecture
Tricks with Traits

• Need to compose the generic algorithms of method-ful roles with the classes whose objects play those roles
• This is a simple class composition
• Can use Traits (à la Schärli) to glue classes together
  – Extra “hidden” field in Smalltalk classes
  – Current Squeak implementation maps the method name into every class using it
  – Trivial application of templates in C++
template <class ConcreteDerived>
class TransferMoneySink: public MoneySink
{
public:
    void transferFrom(double amount, MoneySource *src) {
        deposit(amount);
        updateLog("Transfer in", DateTime(), amount);
    }
};

template <class ConcreteDerived>
class TransferMoneySource: public MoneySource
{
public:
    // Parameters
typedef double Currency;
    virtual Currency availableBalance(void) = 0;
    virtual void withdraw(Currency) = 0;
    virtual void updateLog(string, DateTime, Currency) = 0;

    // Role behaviors
    void transferTo(Currency amount, MoneySink *recipient) {
        // This code is reviewable and testable!
        beginTransaction();
        if (availableBalance() < amount) {
            endTransaction();
            throw InsufficientFunds();
        } else {
            withdraw(amount);
            recipient->deposit(amount);
            updateLog("Transfer Out", DateTime(), amount);
            recipient->updateLog("Transfer In", DateTime(), amount);
            gui->displayScreen(SUCCESS_DEPOSIT_SCREEN);
            endTransaction();
        }
    }
};
Injecting the roles into classes

class SavingsAccount:
    public Account,
    public TransferMoneySink<SavingsAccount> {
        public:
            typedef double Currency;
            Currency availableBalance(void);
            void withdraw(Currency);
            void deposit(Currency);
            void updateLog(string, DateTime, Currency);
            Currency interest(void) const;
    };

class InvestmentAccount:
    public Account,
    public TransferMoneySource<InvestmentAccount> {
        public:
            typedef double Currency;
            Currency availableBalance(void);
            void withdraw(Currency);
            void deposit(Currency);
            void updateLog(string, DateTime, Currency);
            Currency dividend(void) const;
    };

(dumb)
What do I get?

- Polymorphism is gone
- All objects that play the same role process the same messages with the same methods
- Algorithms read like algorithms rather than fragments
- Rapidly evolving functionality is separated from stable domain logic
- Can reason about system state and behavior, not just object state and behavior
Or, from an Agile perspective:

- Allows me to connect with the user mental model
  - Users & interactions instead of processes and tools
- Can employ shared customer vocabulary
  - Customer collaboration, not contracts
- Can reason about form of task sequencing
  - More likely to deliver working software
- Exposes the changing part for ready update
  - Embracing change
Learn more at:

- Baby IDE:  
  - [http://heim.ifi.uio.no/~trygver/themes/babyide/babyide-index.html](http://heim.ifi.uio.no/~trygver/themes/babyide/babyide-index.html)

- Agile Architecture, the book: manuscript:  
  - [http://sites.google.com/a/gertrudandcope](http://sites.google.com/a/gertrudandcope)

- Two Grumpy Old Men:  
  - ROOTS 2008
• cope@gertrudandcope.com