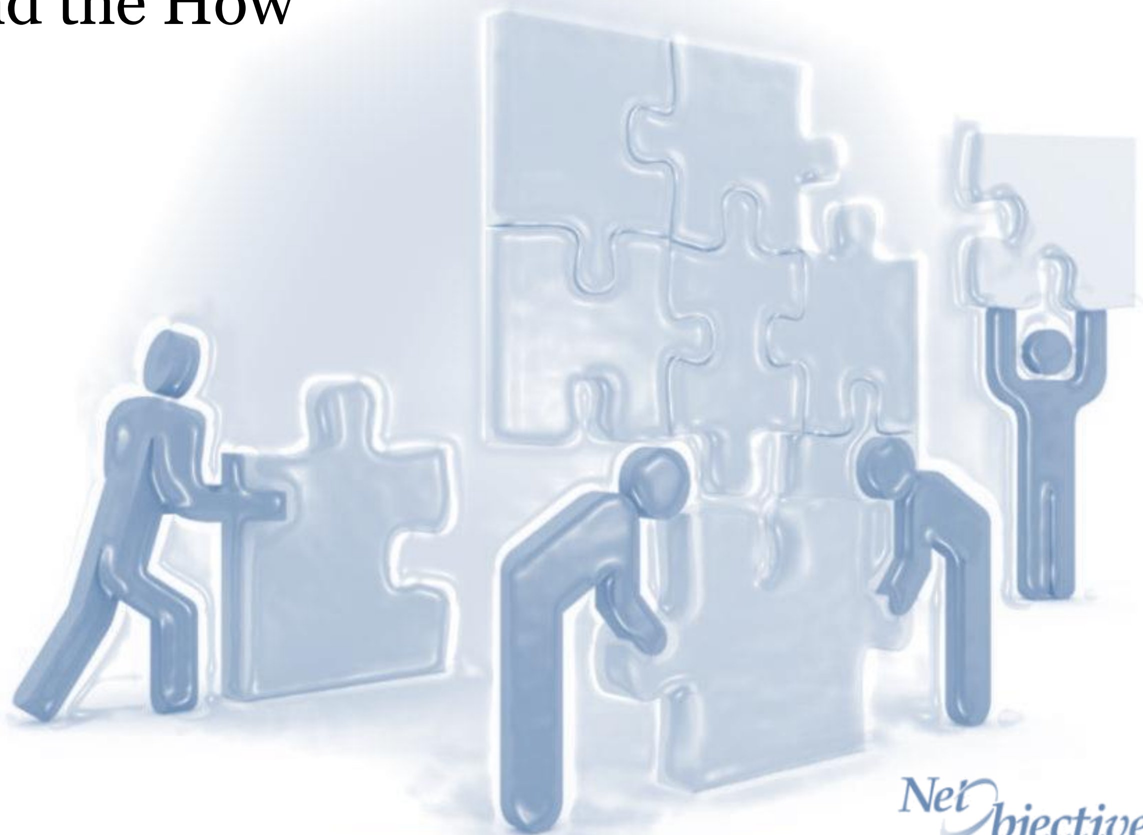
Agile Architecture

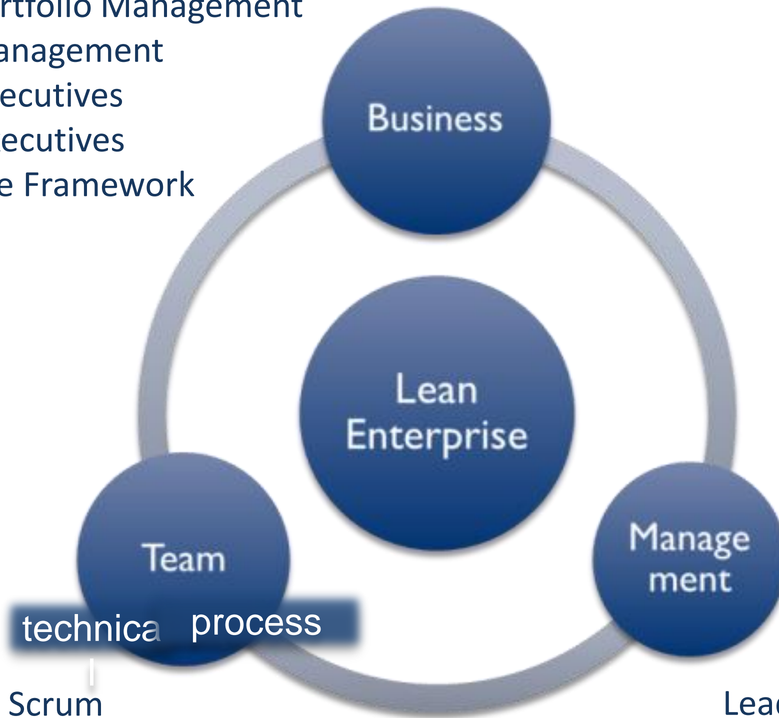
The Why, the What and the How



BECOMING
LEAN-agile

Net Objectives

Product Portfolio Management
Product Management
Lean for Executives
SAFe for Executives
Scaled Agile Framework



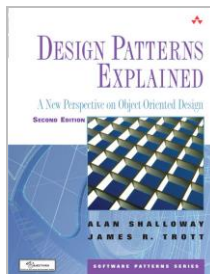
Net Objectives

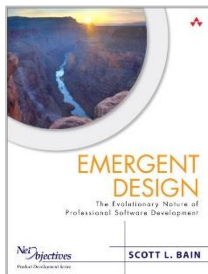
ASSESSMENTS
CONSULTING
TRAINING
COACHING

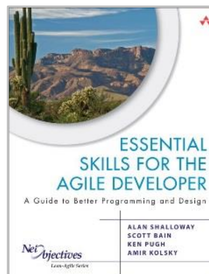
Kanban / Scrum
ATDD / TDD / Design Patterns
SAFe Scrum/XP

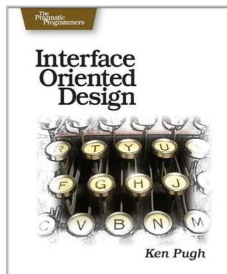
Leading SAFe
SAFe Program Consultant
Lean Management
Project Management

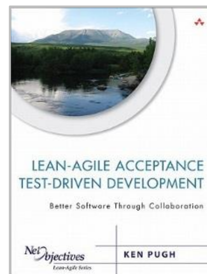
podcasts

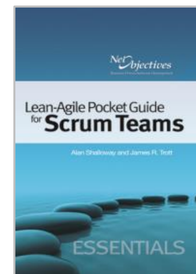


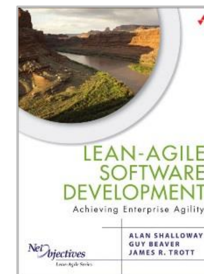


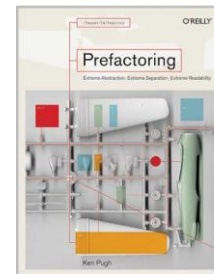










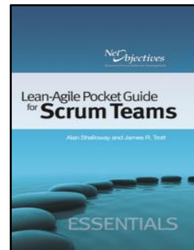


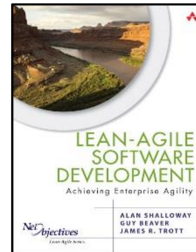
© Copyright Net Objectives, Inc. All Rights Reserved

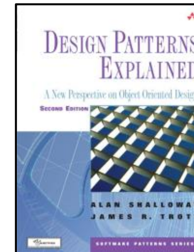
Al Shalloway

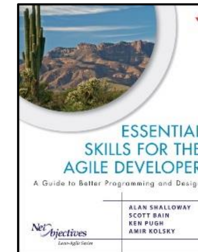
CEO, Founder

alshall@NetObjectives.com









Co-founder of Lean-Systems Society
Co-founder Lean-Kanban University

© Copyright Net Objectives, Inc. All Rights Reserved

What are objects?

How much time does typing take?

If you have a choice between A & B and one is easy to undo & the other isn't, which do you chose?

© Copyright Net Objectives, Inc. All Rights Reserved

when adding functionality,
where's the problem?

is it in...

writing the new code?

integrating it into the new system?

and which is likely to be the source of difficulties?

Net Objectives

© Copyright Net Objectives, Inc. All Rights Reserved

Units: Agile Architecture

1. **Purpose of Architecture**
2. Perspective Models
3. Useful Practices

© Copyright Net Objectives, Inc. All Rights Reserved

TABLE WORK

Purpose

Why have an **architecture**?

and, by the way, what is an architecture?

Net Objectives

© Copyright Net Objectives, Inc. All Rights Reserved

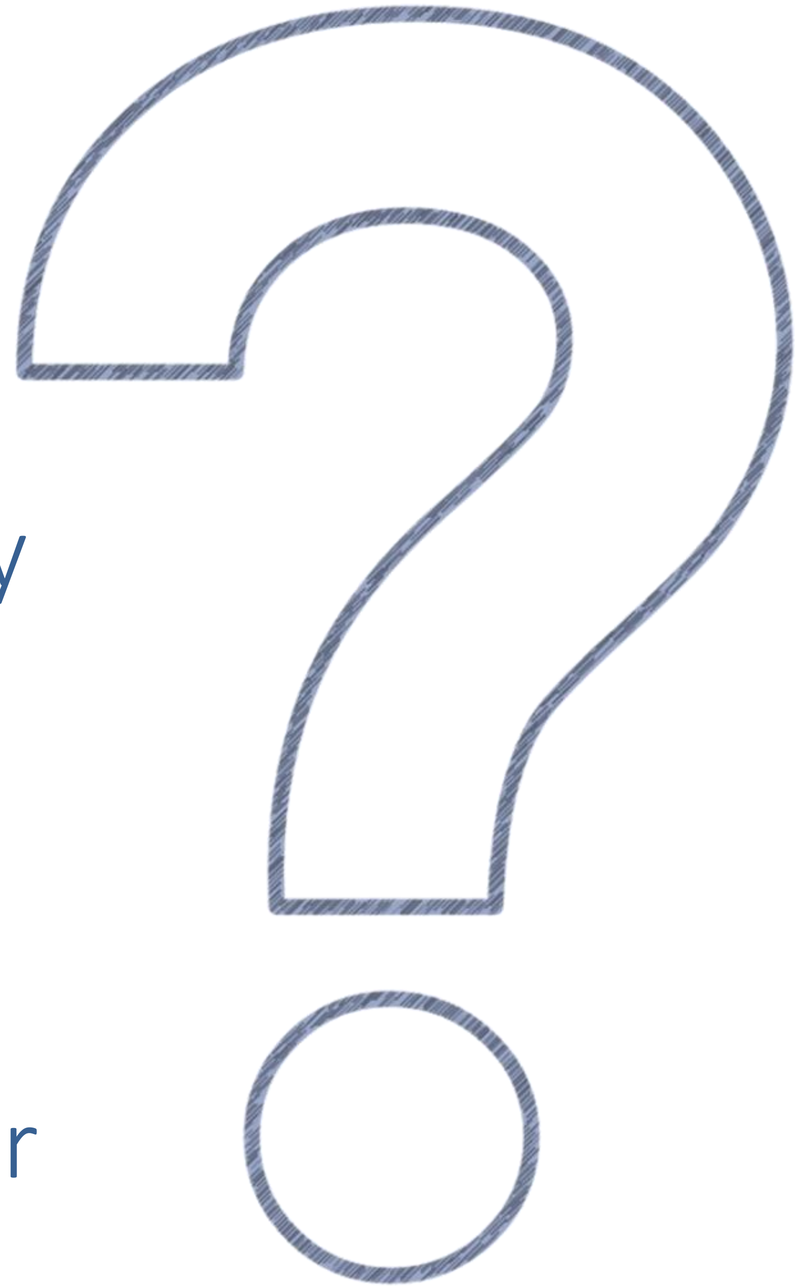
architecture

what is architecture?

- Underlying technology
- Structure
- Layers

What can't be changed?

What provides context for change?



© Copyright Net Objectives, Inc. All Rights Reserved

architecture

when you want to extend an architecture for performance, scalability or security reasons, which causes problems:

architecture or **code quality**

© Copyright Net Objectives, Inc. All Rights Reserved

Desired Outcomes

A **Vision** – context for local decisions

Enable **new implementations** in the system

Enable **extensions** to the system

- Functional
- Performance / scalable
- Security

Foster **testability**

De-couple systems from the application

Facilitate reuse of common functions across teams

Provide an **example of standards**

all with minimal risk

© Copyright Net Objectives, Inc. All Rights Reserved

architecture

what if…

- integrating functionality wasn't hard?
- we weren't tied to our system's architecture?
- we were truly object-oriented?
- we could be prepared for the unknown?

© Copyright Net Objectives, Inc. All Rights Reserved

Not trying to anticipate change

Cannot prevent change

Designing to accommodate* change requires

- Layered architectures
- Code clarity and maintainability

*Change is not the problem, it's the damage that it causes. What if it didn't cause damage.

© Copyright Net Objectives, Inc. All Rights Reserved

Classic Architecture	Agile Architecture
Create a structure that contains everything	Create a structure that can change
Anticipate everything	Prepare for anything
Understand the big picture so have a framework to hold things	Understand the big picture so can evolve to hold things
Learn what you need to up front	Set things up to use new ideas as they become apparent

© Copyright Net Objectives, Inc. All Rights Reserved

architecture

the real question

How do we allow for evolutionary architecture?

there is a parallel between agile architecture and agile discovery

Net Objectives
© Copyright Net Objectives, Inc. All Rights Reserved

how?

Emergent design

Testing at **Behavior** and **Functional** levels

Behavioral testing (ATDD, BDD)

- Test perspective is from customer

Functional test

- Test perspective is from coder

Why test-first in both cases?

© Copyright Net Objectives, Inc. All Rights Reserved

Types of Refactoring

Refactoring Bad Code
Code "smells"
Improve Design without changing Function.
Refactor to improve code quality
A way to clean up code without fear of breaking the system

© Copyright Net Objectives, Inc. All Rights Reserved

Types of Refactoring

Refactoring Bad Code	Refactoring Good Code
Code "smells"	Code is "tight"
Improve Design without changing Function.	A new Requirement means code needs to be changed
Refactor to improve code quality	Design needs to change to accommodate this.
A way to clean up code without fear of breaking the system	A way to make this change without fear of breaking the system

© Copyright Net Objectives, Inc. All Rights Reserved

Testability

Code that is difficult to unit test is often:

1. **Tightly Coupled**: "I cannot test this without instantiating half the system“
2. **Weakly Cohesive**: "This class does so much, the test will be enormous and complex!“
3. **Redundant**: "I'll have to test this in multiple places to ensure it works everywhere"

© Copyright Net Objectives, Inc. All Rights Reserved

Testability and Design

Considering how to test your objects before designing them is, in fact, a kind of design

It forces you to look at:

- the public method definitions
- what the responsibilities of the object are

Easy testability is tightly correlated to loose coupling and strong cohesion

© Copyright Net Objectives, Inc. All Rights Reserved

Units: Agile Architecture

1. Purpose of Architecture
2. **Perspective Models**
3. Useful Practices

© Copyright Net Objectives, Inc. All Rights Reserved

Perspective Models

1. **Abstraction**
2. Creation
3. Structure

- Conceptual
- Specification
- Implementation

thinking points

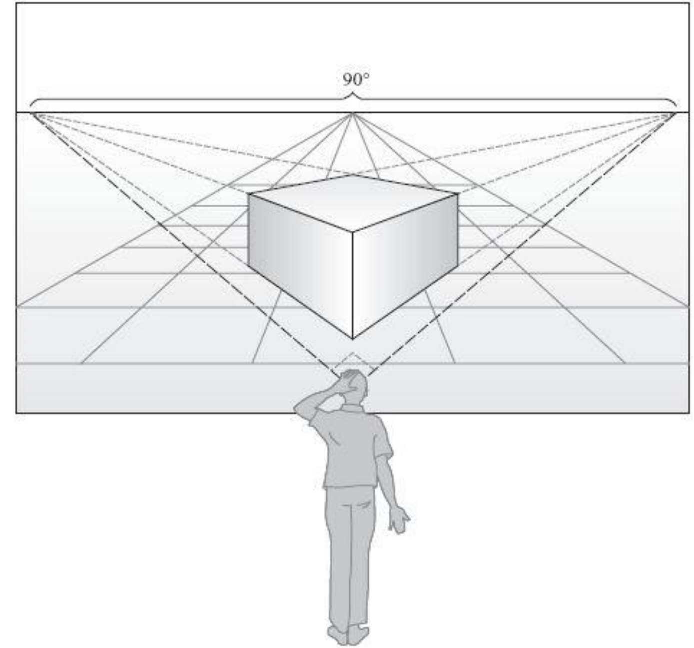
© Copyright Net Objectives, Inc. All Rights Reserved

Three Perspectives of Abstraction

Conceptual
What you want

Specification
How you use it

Implementation
How it actually works



UML Distilled: A Brief Guide to the Standard Object Modeling Language, Second Edition, by Martin Fowler, Addison-Wesley Pub Co, ISBN: 0321193687

© Copyright Net Objectives, Inc. All Rights Reserved

Use of Perspectives

Any entity (class, method, delegate, etc…) in a system should operate at one of these perspectives:

- Conceptual
- Specification
- Implementation

UML Distilled: A Brief Guide to the Standard Object Modeling Language, Second Edition, by Martin Fowler, Addison-Wesley Pub Co, ISBN: 0321193687

© Copyright Net Objectives, Inc. All Rights Reserved

Perspective Models

1. Abstraction
2. **Creation**
3. Structure

- Create objects
- Use objects

thinking points

© Copyright Net Objectives, Inc. All Rights Reserved

Principle: Separate Use From Construction

The relationship between any entity A and any other entity B in a system should be limited such that

A **makes** B or

A **uses** B,

and **never both**.

© Copyright Net Objectives, Inc. All Rights Reserved

Normal Constructor

C++

```
class ByteFilter {
  public:


    ByteFilter();
    virtual ~ByteFilter();
}
ByteFilter::ByteFilter () {
  // do any constructor behavior here
}






  // the rest of the class follows


void main {
  ByteFilter *myByteFilter;
    // . . .
    myByteFilter = new ByteFilter();
    // . . .
}
```

Net Objectives
© Copyright Net Objectives, Inc. All Rights Reserved

Encapsulating the Constructor

C++

```
class ByteFilter {
  public:
    static ByteFilter* getInstance();
  protected:
    ByteFilter();
    virtual ~ByteFilter();
}
ByteFilter::ByteFilter () {
  // do any constructor behavior here
}
ByteFilter* ByteFilter::getInstance() {
     return new ByteFilter();
  }
Void ByteFilter::returnInstance( ByteFilter *bf2Delete) {
    delete bf2Delete;
}
  // the rest of the class follows


void main {
  ByteFilter *myByteFilter;
    // . . .
    myByteFilter = ByteFilter::getInstance();
    // . . .
}
```


© Copyright Net Objectives, Inc. All Rights Reserved

Accommodating Change: Complexity

C++

```
class ByteFilter { // this is an abstract class.
  public:
    static ByteFilter* getInstance();
  protected:
    ByteFilter();
};


ByteFilter* ByteFilter::getInstance() {
  if (someDecisionLogic()) {
    return new HiPassFilter();
  } else {
    return new LoPassFilter();
} }
//Note: both HiPassFilter and LoPassFilter derive from ByteFilter
//      but implement the filtering differently


void main () {
  ByteFilter *myByteFilter;
    // . . .
    myByteFilter = ByteFilter::getInstance();
    // . . .
}
```

No Change!

© Copyright Net Objectives, Inc. All Rights Reserved

In C++, Objects on the Stack

C++

Instead of this…

```
ByteFilter myByteFilter;
//          .
//          . Object gets used
//          .
// Object falls out of scope, memory is cleaned up
```

…we prefer this – “Encapsulating Destruction”

```
ByteFilter* myByteFilter = ByteFilter::getInstance();
//          .
//          . Object gets used
//          .
ByteFilter::returnInstance(myByteFilter);
```


© Copyright Net Objectives, Inc. All Rights Reserved

Using Smart Pointers

C++

```
auto_ptr<ByteFilter> myByteFilter = ByteFilter::getInstance();
//          .
//          . Object gets used, exception-safety ensured
//          .
```

Net Objectives

© Copyright Net Objectives, Inc. All Rights Reserved

Perspective Models

1. Abstraction
2. Creation
3. **Structure**

- Architecture
- Application

thinking points

© Copyright Net Objectives, Inc. All Rights Reserved

architecture is layered

One layer should not know about another

© Copyright Net Objectives, Inc. All Rights Reserved

Units: Agile Architecture

1. Purpose of Architecture
2. Perspective Models
3. **Useful Practices**

© Copyright Net Objectives, Inc. All Rights Reserved

Useful Practices

1. **Understand objects**
2. Simplify adding new functionality
3. Simplify Interfaces
4. Design for the Unknown
5. Avoid redundancy

- Objects are not data with methods
- Objects are manifestations of concepts or behavior

thinking points

© Copyright Net Objectives, Inc. All Rights Reserved

Useful Practices

1. Understand objects
2. **Simplify adding new functionality**
3. Simplify Interfaces
4. Design for the Unknown
5. Avoid redundancy

- Use dependency injection
- Separate **use** from **construction**

thinking points

© Copyright Net Objectives, Inc. All Rights Reserved

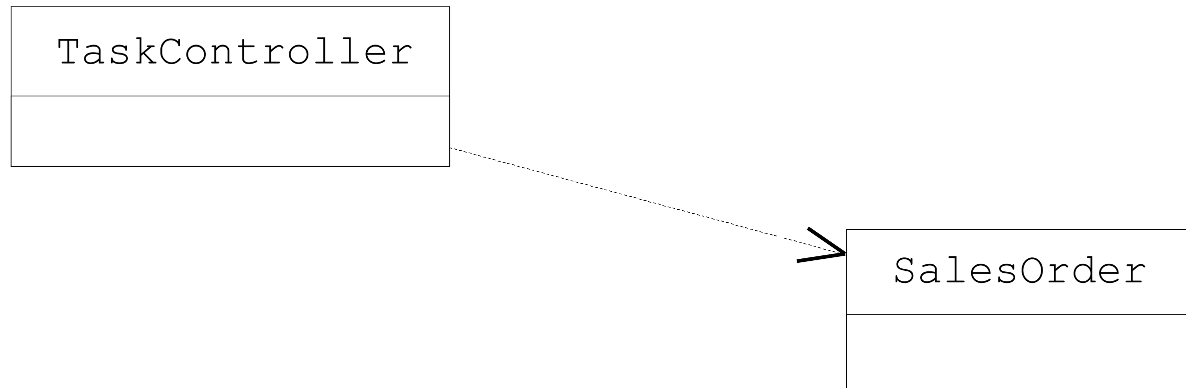
The Problem

We are writing a web-enabled sales order system

A customer signs in and fills out the order

© Copyright Net Objectives, Inc. All Rights Reserved

Initial Solution



Have `TaskController` instantiate `SalesOrder` that handles filling out the sales order, etc.

© Copyright Net Objectives, Inc. All Rights Reserved

The Challenge

As soon as we get new variations in tax we have to modify the `SalesOrder` object

Where should we add the new responsibility if taxation begins to vary?

© Copyright Net Objectives, Inc. All Rights Reserved

New Requirement

Multiple Tax Domains

Eventually need to handle different tax rules

- US Tax
- Canadian Tax

One Solution

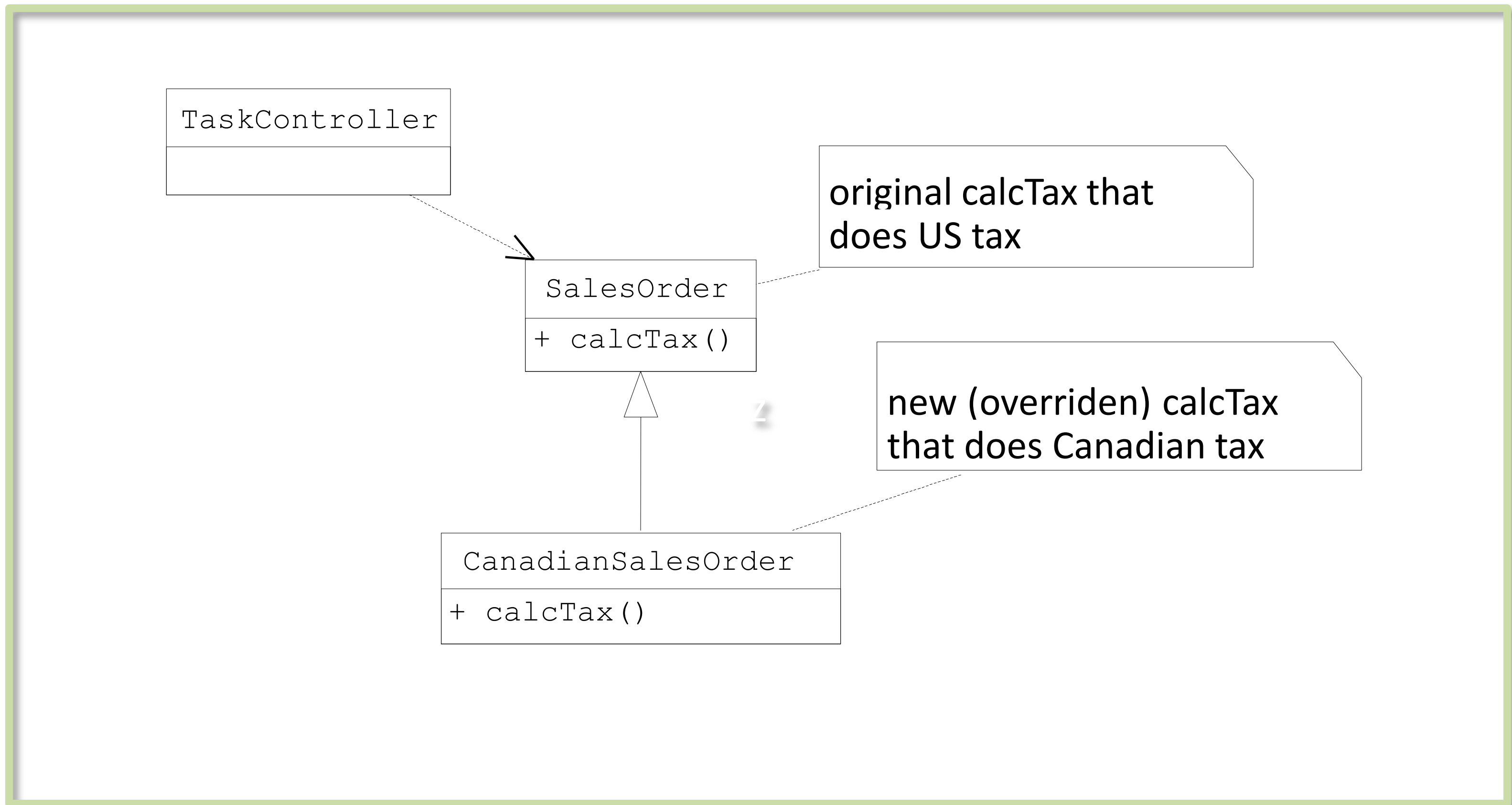
```
method calcTax

  //    use switch on type of tax rule to be used
  //       TYPE US:
  //         calc tax based on US rules
  //         break
  //       TYPE CANADIAN:
  //         calc tax based on Canadian rules
  //         break
```


© Copyright Net Objectives, Inc. All Rights Reserved

Direct Inheritance for Specialization

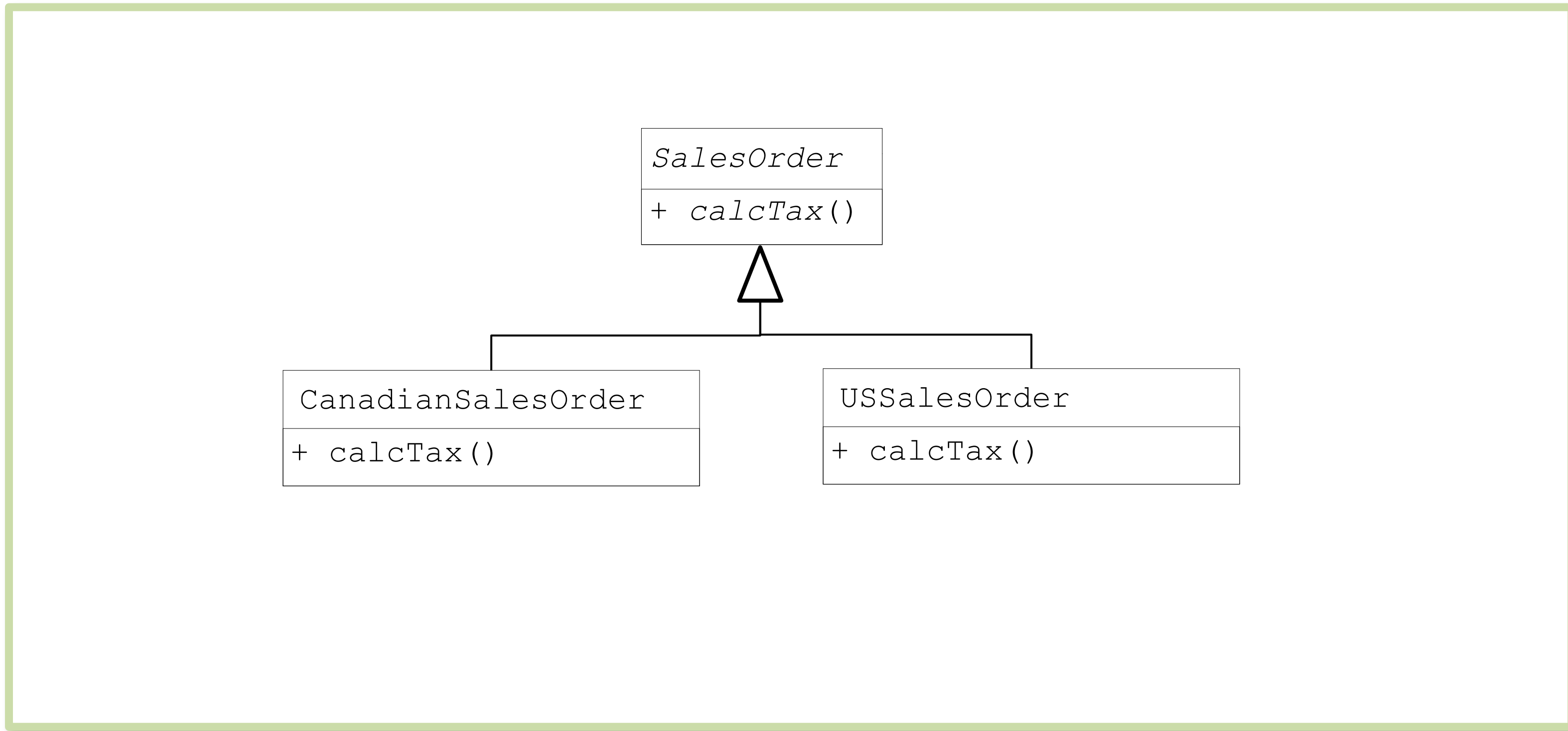
We can solve this problem by specializing our first `SalesOrder` object to handle the new tax rules



© Copyright Net Objectives, Inc. All Rights Reserved

Abstract Inheritance for Specialization

This is a bit better…



Net Objectives

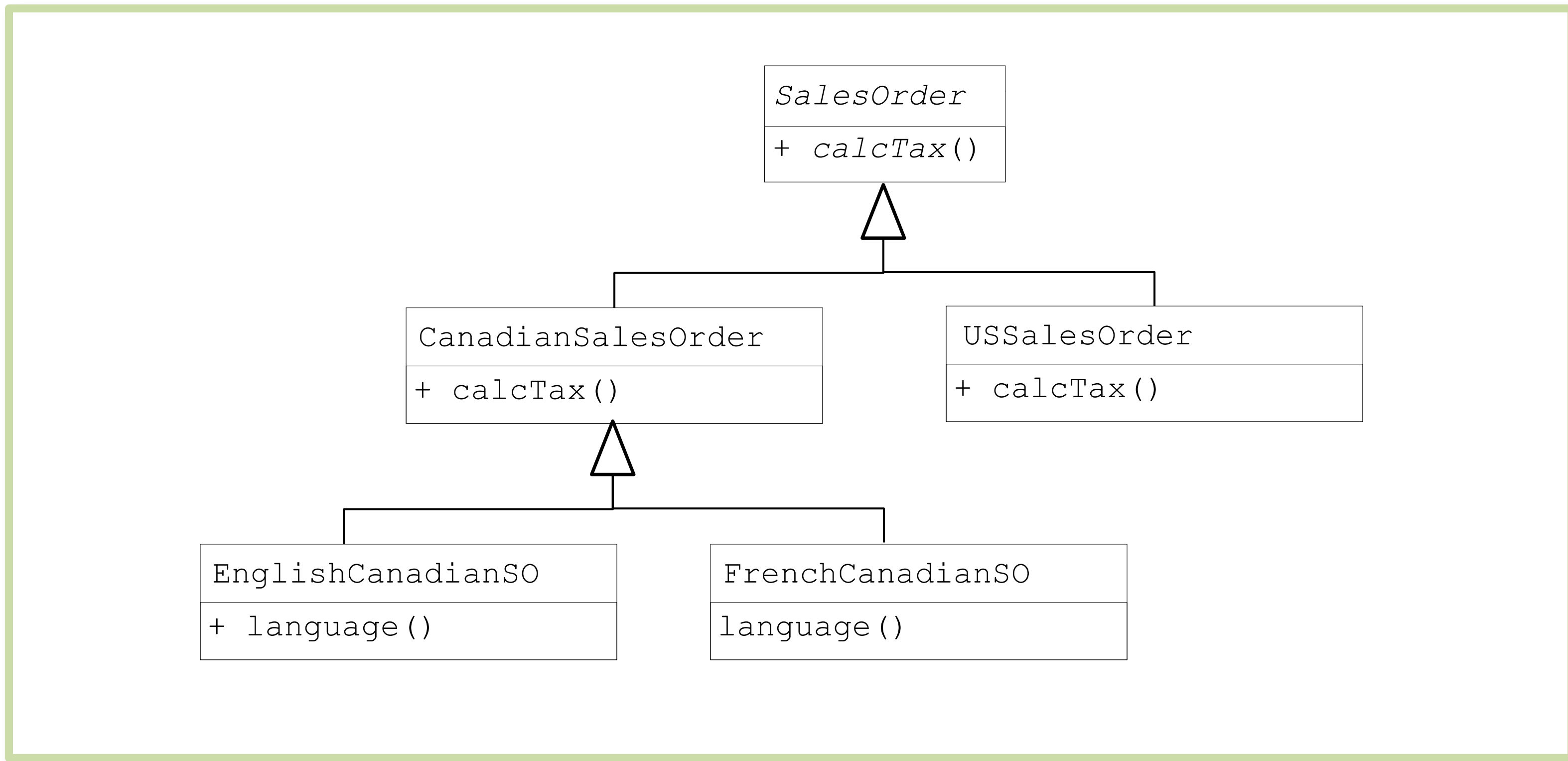
© Copyright Net Objectives, Inc. All Rights Reserved

May Lead to Maintenance Problems

If we get new variations, where do we put them?

We either start using lots of switches (which makes the code difficult to understand), or we start over-specializing (which still makes things difficult)

What if we need to switch tax algorithms at runtime?

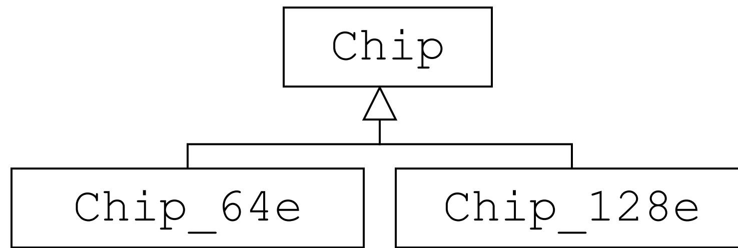


© Copyright Net Objectives, Inc. All Rights Reserved

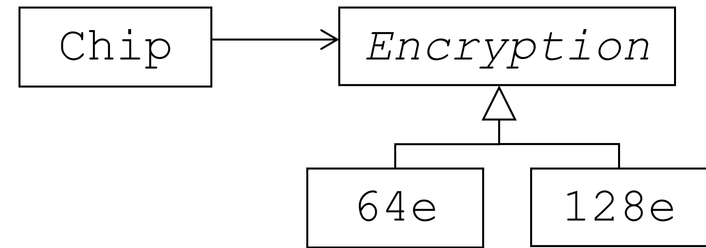
Proper Use of Inheritance

Define a class that encapsulates variation,

contain (via delegation) an instance of a concrete class derived from the abstract class defined earlier



Class Inheritance to Specialize



Class Inheritance to Categorize

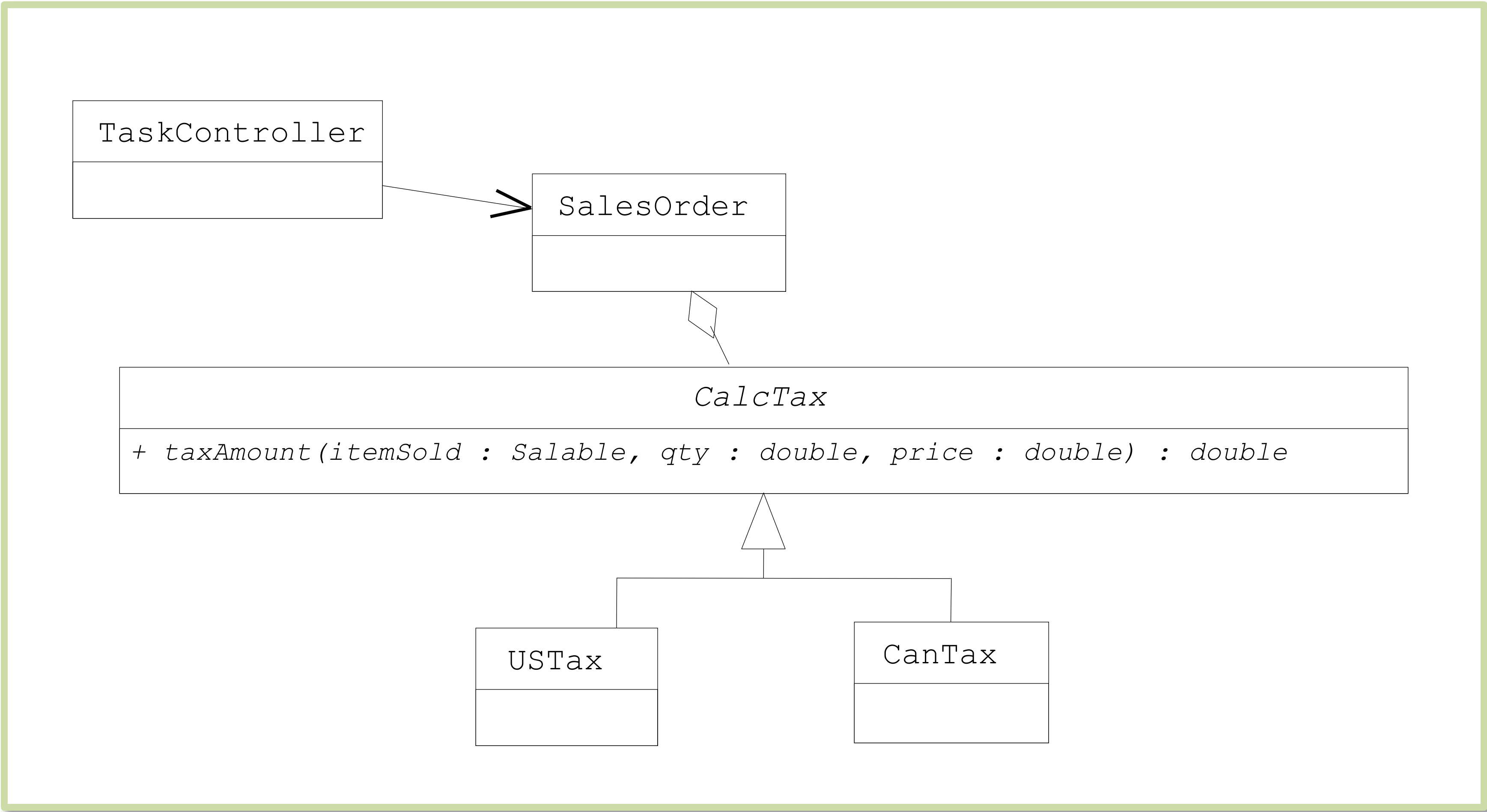
Who is the variation being hidden from?

1. Decoupling of concepts
2. Deferring decisions until runtime
3. Small performance hit

Net Objectives

© Copyright Net Objectives, Inc. All Rights Reserved

Dependency Injection



Strategy Pattern

© Copyright Net Objectives, Inc. All Rights Reserved

Useful Practices

1. Understand objects
2. Simplify adding new functionality
3. **Simplify Interfaces**
4. Design for the Unknown
5. Avoid redundancy

- Use Dependency Inversion Principle

thinking points

© Copyright Net Objectives, Inc. All Rights Reserved

Dependency Inversion Principle

High level modules should not depend on low-level modules. Both should depend on abstractions.

Abstractions should not depend on details. Details should depend on abstractions.

> *"The modules that contain the high-level business rules should take precedence over, and be independent of, the modules that contain the implementation details."*

Agile Software Development: Principles, Practices and Patterns Bob Martin, pg 127.

Net Objectives
© Copyright Net Objectives, Inc. All Rights Reserved

Useful Practices

1. Understand objects
2. Simplify adding new functionality
3. Simplify Interfaces
4. **Design for the Unknown**
5. Avoid redundancy

- Scott's Magic Card
- Encapsulate that!

thinking points

© Copyright Net Objectives, Inc. All Rights Reserved

The Situation

Multiple processors in a real-time environment

There will be a performance problem… just don’t know where / how

What can we do?

© Copyright Net Objectives, Inc. All Rights Reserved

The Options

Figure out where / how the performance problem will show up

Just use what appears best at first and fix it later if it becomes clear later that we need something else

Use Scott Bain's magic consultant card

© Copyright Net Objectives, Inc. All Rights Reserved

How would I design
if I knew that
no matter what I did
it would be wrong?

© Copyright Net Objectives, Inc. All Rights Reserved

The Magic Consultant Card

Scott Bain, a Net Objectives' Design Patterns Instructor, has a magic card.

It tells you how to solve design problems.

However, it requires that you explicitly and concisely state what your problem is.

Fortunately, we know what our problem is.

But we do not know the nature of the collection we need.

© Copyright Net Objectives, Inc. All Rights Reserved

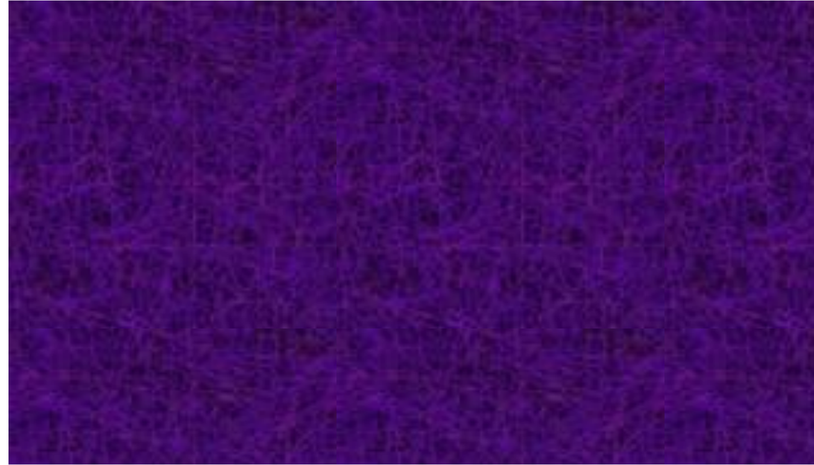
The Magic Consultant Card

Instructions

1. Hold magic card face down in front of you
2. State aloud three times what your problem is
3. Flip the card over
4. Read the card

© Copyright Net Objectives, Inc. All Rights Reserved

The Magic Consultant Card



© Copyright Net Objectives, Inc. All Rights Reserved

The Magic Consultant Card

The Good News

The magic card is always right!

The Bad News

It doesn't tell you how to do what it tells you to do!

This is where design patterns come in. In many cases a pattern exists to help you see how to solve your problem.

© Copyright Net Objectives, Inc. All Rights Reserved

The Net Objectives Patterns Repository

Behavior	Strategy	Bridge	Template Method	Null Object	
Sequence	Decorator	Chain of Responsibility	Template Method		
Workflow	Template Method	Visitor	Bridge	Null Object	
Cardinality	Decorator	Chain of Responsibility	Proxy	Observer	Composite
Construction	Singleton	Abstract Factory	Builder	Factory Method	Prototype
Selection	Chain of Responsibility				
Structure	Composite	Template Method			
Entity	Facade	Adapter	Proxy		
Relationships	Observer	Command	Mediator	Visitor	
Dependencies	Mock Object				

http://www.netobjectivestest.com/PatternRepository/index.php?title=Main_Page

Useful Practices

1. Understand objects
2. Simplify adding new functionality
3. Simplify Interfaces
4. Design for the Unknown
5. **Avoid redundancy**

- Avoiding redundancy is impossible
- We can manage it
- Shalloway's Principle

thinking points

© Copyright Net Objectives, Inc. All Rights Reserved

Shalloway's Law

If 'N' things need to change and 'N>1', Shalloway will find at most 'N-1' of these things

Net Objectives

© Copyright Net Objectives, Inc. All Rights Reserved

Shalloway’s Principle

Avoid situations where Shalloway’s Law Applies

© Copyright Net Objectives, Inc. All Rights Reserved

Summary

review

Design for change

Use models of perspective

- Abstraction
- Creation
- Structure

Use good practices

- Dependency injection
- Model the unknown
- Shalloway's Law

You've still gotta think!

© Copyright Net Objectives, Inc. All Rights Reserved

Questions

email: **alshall@NetObjectives.com**

twitter: **@alshalloway**

Webinars

May 28 **The Challenges of Team-Based or Evolutionary Based Methods In Achieving Agile at Scale**

© Copyright Net Objectives, Inc. All Rights Reserved