Reflection on Agile Development from a Non Software Developer

Dr. Lin Chiu
Franklin University
Columbus, Ohio

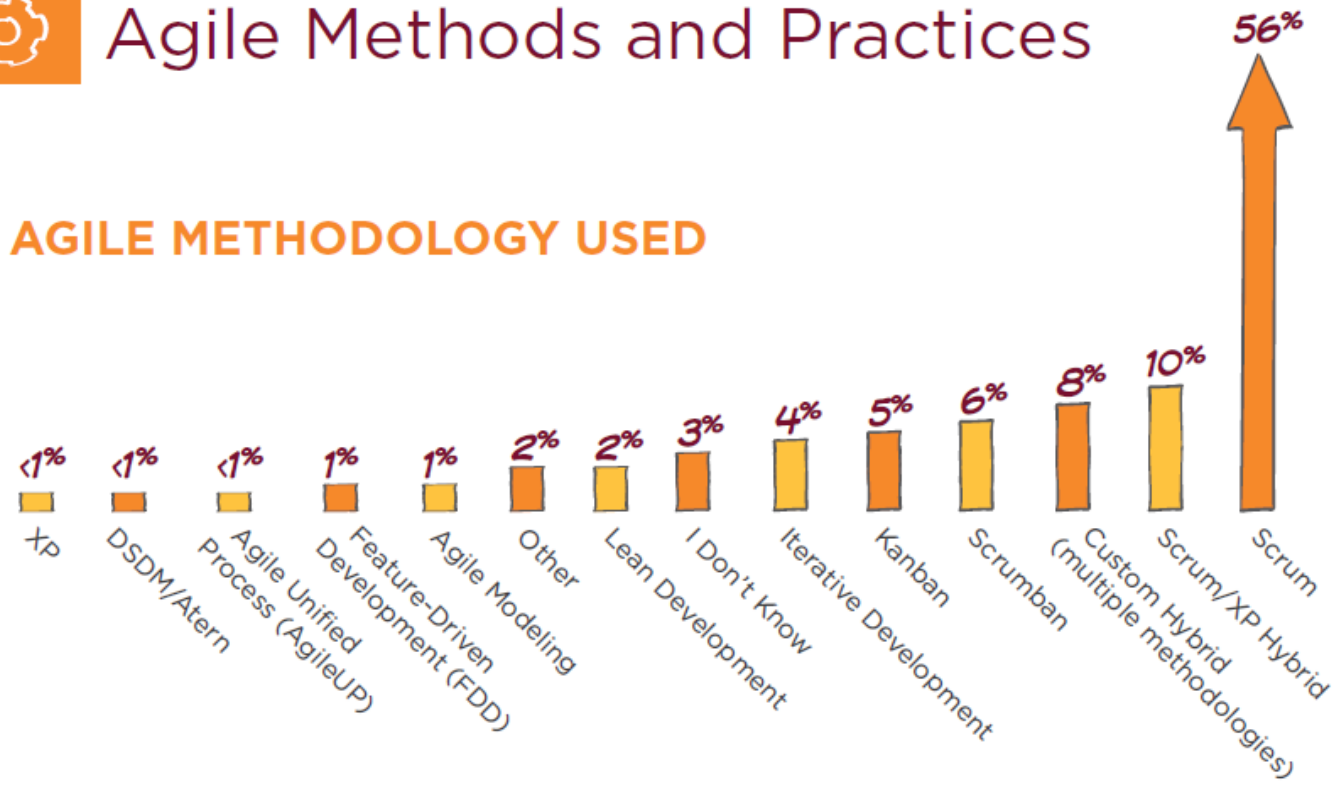
What is the fuss about Agile development?

STATE OF AGILE

Agile Methods and Practices

9th ANNUAL STATE OF AGILE™ SURVEY

AGILE METHODOLOGY USED



Source: 9th Annual State of Agile Survey by VersionOne

What is Agile Development?

AGILE VS WATERFALL

- Agile is iterative
- Agile is incremental
- Agile is adaptive
- Agile has limitations

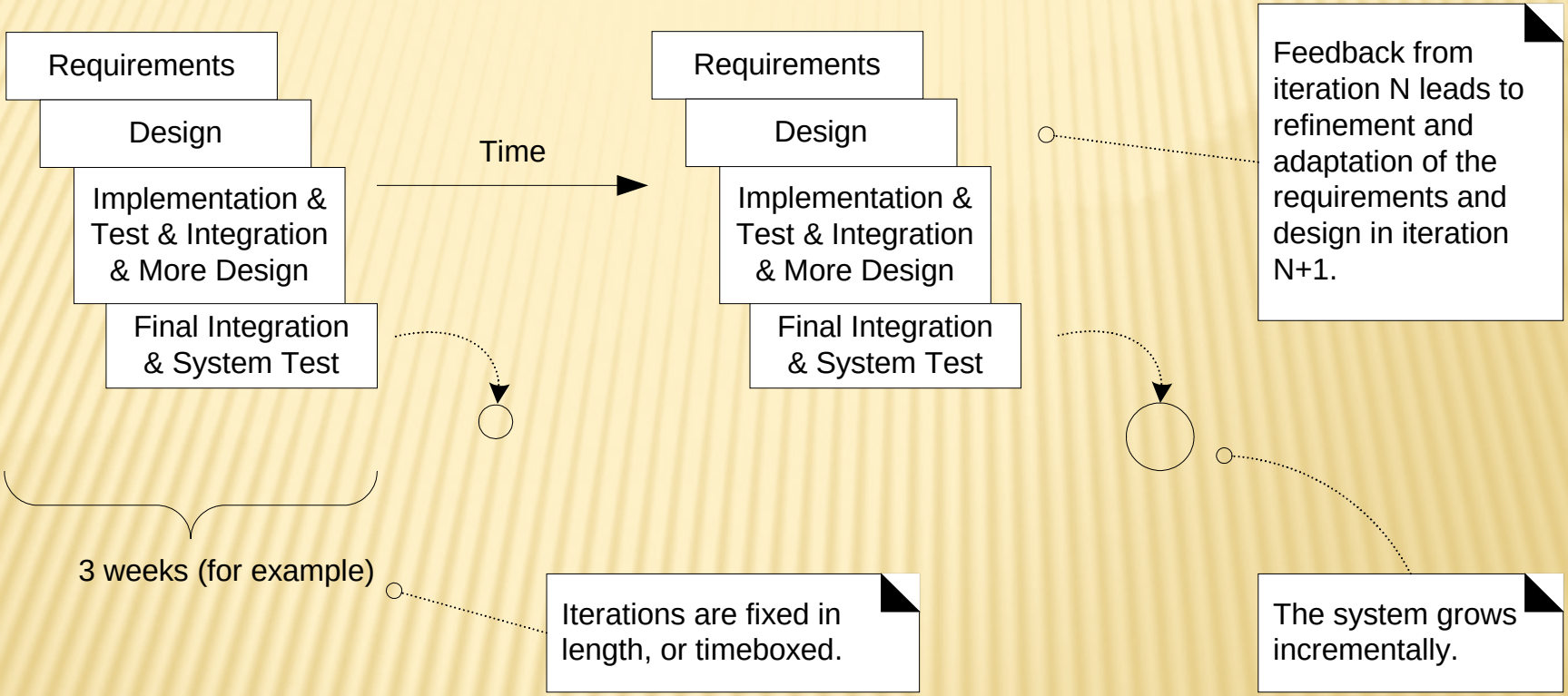
TOPICS COVERED IN THIS TALK

- Agile Unified Process(UP)
- Object Oriented Analysis (OOA)
- Object Oriented Design (OOD)
- Design Patterns
- Unified Modeling Language (UML)
- Test Driven Development (TDD)

(RATIONAL) UNIFIED PROCESS

- An iterative software development process
  - Flexible and open
  - Series of short cycles
  - Evolutionary
    - Lack of complete requirements
    - Learn and solve problem as you build
    - Feedback and adaptation evolve the specifications and design

Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development. (3rd).*



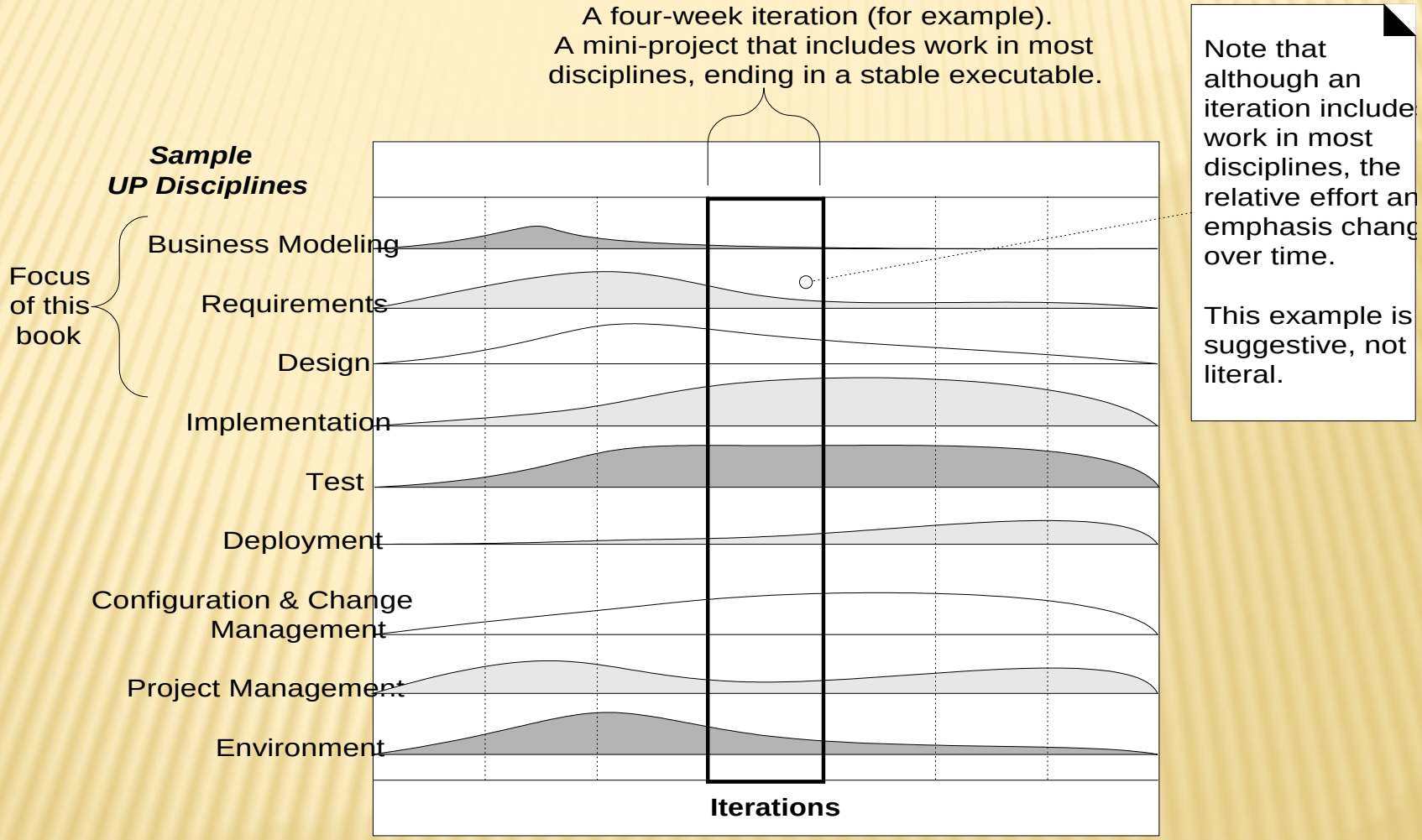
AGILE MODELING

- Build models:
  - Support understanding
  - Model critical parts
  - Use the simplest tool
  - Accept models will be incomplete and inaccurate
  - Tools for developers

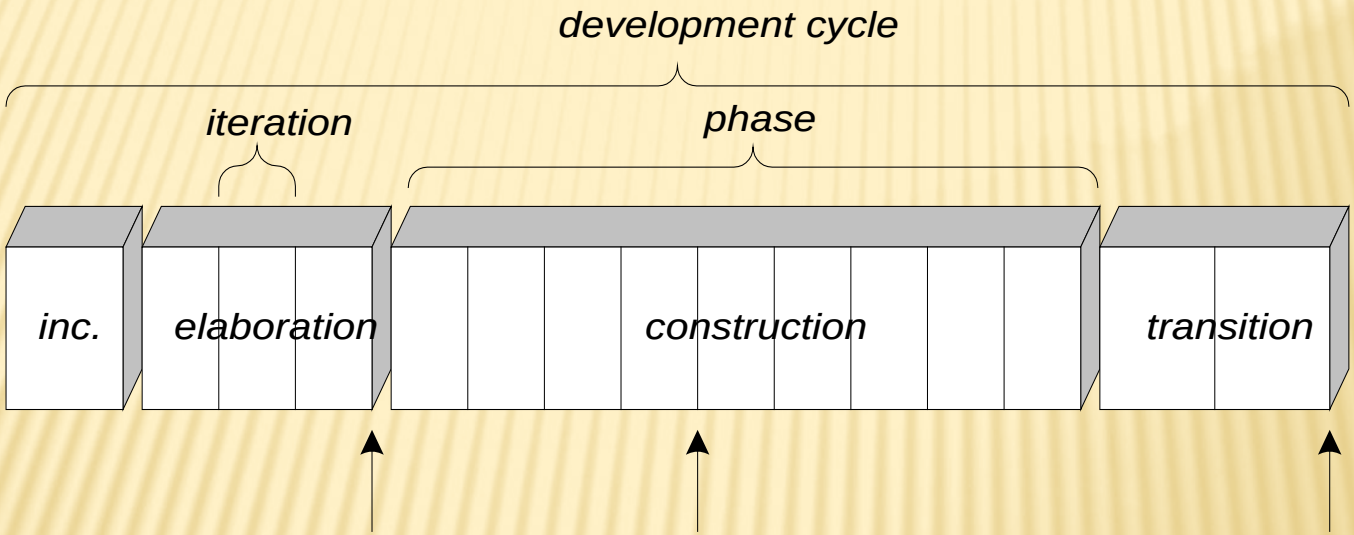
UP BLOCKING BLOCKS

- Disciplines
- Phases
  - Inception – vision, business model, scope, etc.
  - Elaboration – refined vision, iterative implementation of the core architecture, resolution of high risks, etc.
  - Construction – iterative implementation of the remaining low risk and easier elements and preparation for deployment
  - Transition – beta test, deployment

Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development. (3rd).*



Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development. (3rd).*



milestone

An iteration end-point when some significant decision or evaluation occurs

release

A stable executable subset of the final product. The end of each iteration is a minor release.

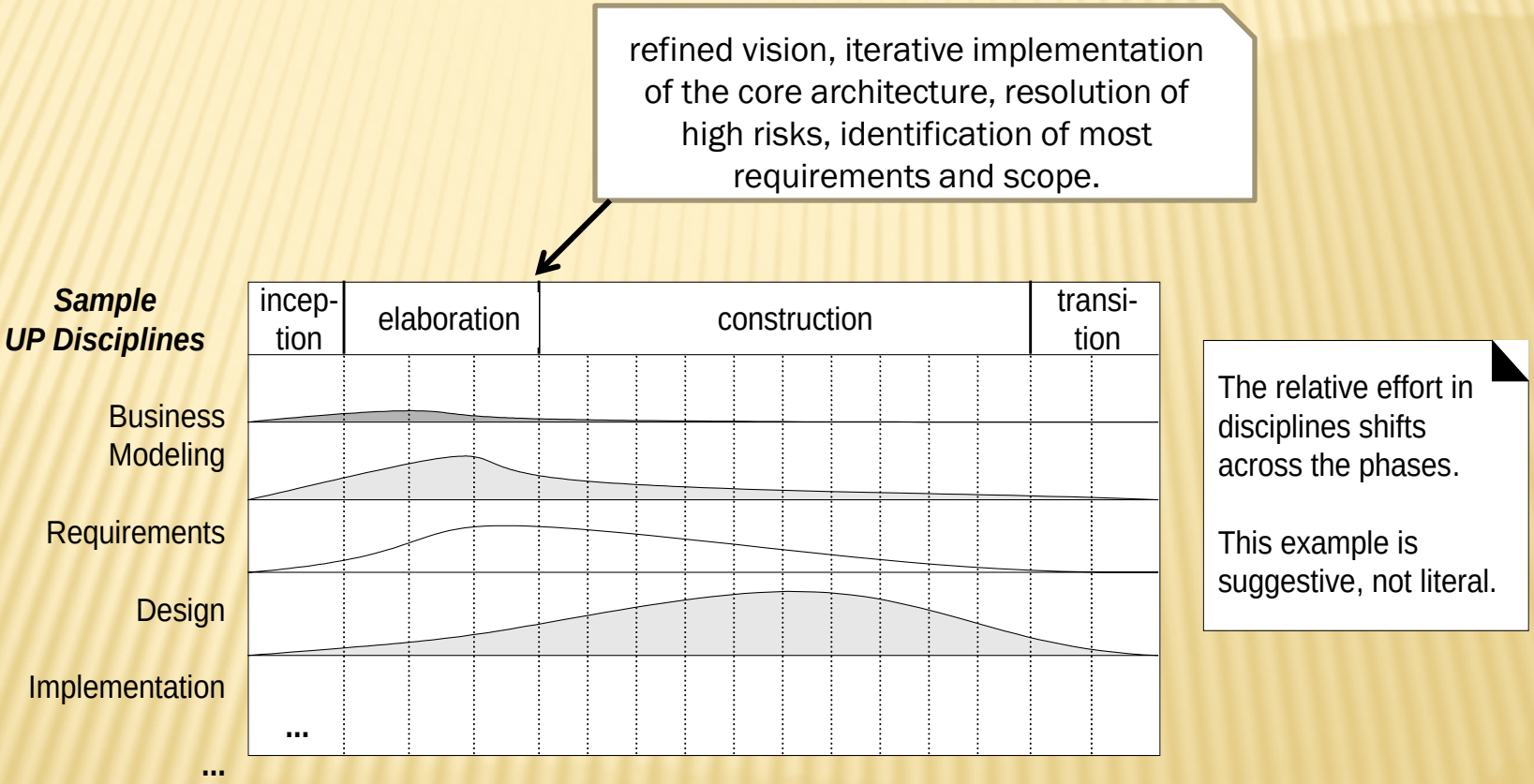
increment

The difference (delta) between the releases of 2 subsequent iterations.

final production release

At this point, the system is release for production use

Larman, C. (2004). *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development. (3rd).*



ANALYSIS AND DESIGN

Analysis emphasizes an investigation of the problem and requirements, rather than a solution.

Design emphasizes a conceptual solution that fulfills the requirements, rather than its implementation.

- Responsibilities
- Assigning responsibilities to objects!
- Patterns

OBJECT-ORIENTED ANALYSIS AND DESIGN

- OO Analysis – find and describe objects
  - Concepts from the domain
- OO Design
  - Define software objects (classes)
- Implementation
  - Make design concrete in a programming language

USE CASES

- Describe in text
- Interaction of a user and the system
- Be careful of implementation vs specification!
- The Dice Game Example

Play a Dice Game: Player requests to roll the dice. System presents results: If the dice face value totals seven, player wins; otherwise, player loses.

OOA/OOD WITH UML

- UML vs Thinking in Objects!
  - Visual modeling with diagramming notations
  - A tool of thought
  - A form of communication
- Learning UML
  - Look at the book examples
  - Alan Holub's UML reference card
  - Use library resources
  - *The Unified Modeling Language User Guide* by Booch, Rumbaugh and Jacobson

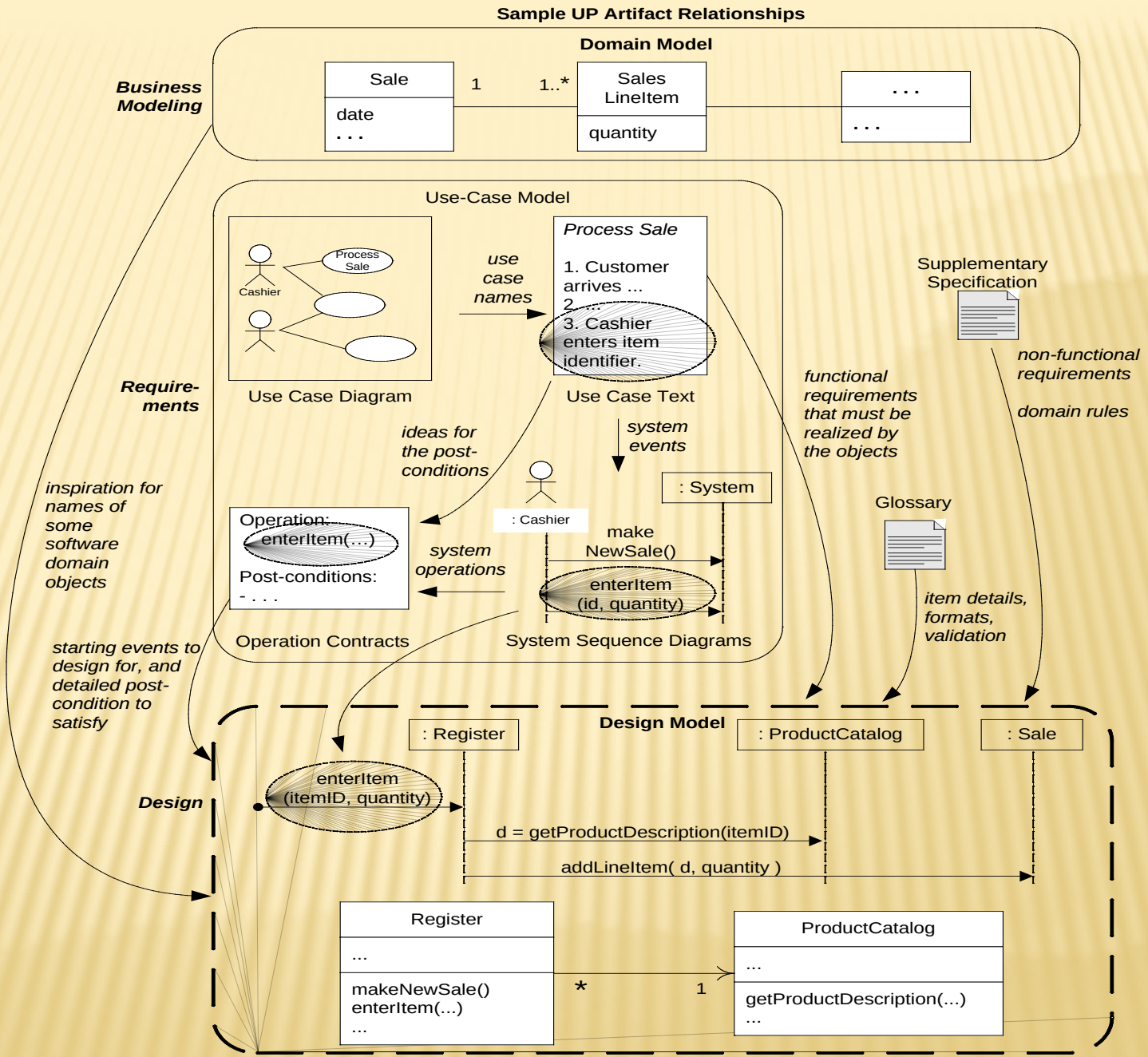
UML

UML is a visual language for specifying, constructing and documenting the artifacts of systems.

- UML as a sketch
- UML as a blueprint
- UML as a programming language

UML PERSPECTIVES

- Conceptual
  - Real world concepts
  - Domain model
- Specification
  - Software abstractions
  - No commitment to a particular implementation
- Implementation
  - Technology

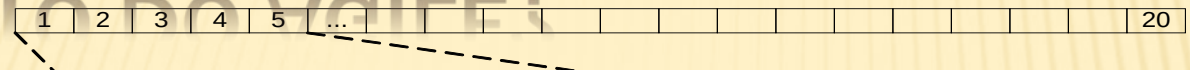

Sample UP Artifact Relationships
Domain Model
Business Modeling
Sale
date
. . .
1
1..*
Sales LineItem
quantity
. . .
. . .
Use-Case Model
Process Sale
Cashier
use case names
Process Sale
1. Customer arrives ...
2. ...
3. Cashier enters item identifier.
Use Case Diagram
Use Case Text
Supplementary Specification
non-functional requirements
domain rules
Require-ments
functional requirements that must be realized by the objects
ideas for the post-conditions
system events
: System
: Cashier
make NewSale()
enterItem (id, quantity)
Operation:
enterItem(...)
Post-conditions:
- . . .
system operations
Glossary
item details, formats, validation
inspiration for names of some software domain objects
starting events to design for, and detailed post-condition to satisfy
Operation Contracts
System Sequence Diagrams
Design Model
Design
: Register
: ProductCatalog
: Sale
enterItem (itemID, quantity)
d = getProductDescription(itemID)
addLineItem(d, quantity)
Register
...
makeNewSale()
enterItem(...)
...
*
1
ProductCatalog
...
getProductDescription(...)
...

HOW TO DO ITERATIVE AND EVOLUTIONARY ANALYSIS AND DESIGN

It is a misunderstanding to think there is no value in analysis and design before programming.

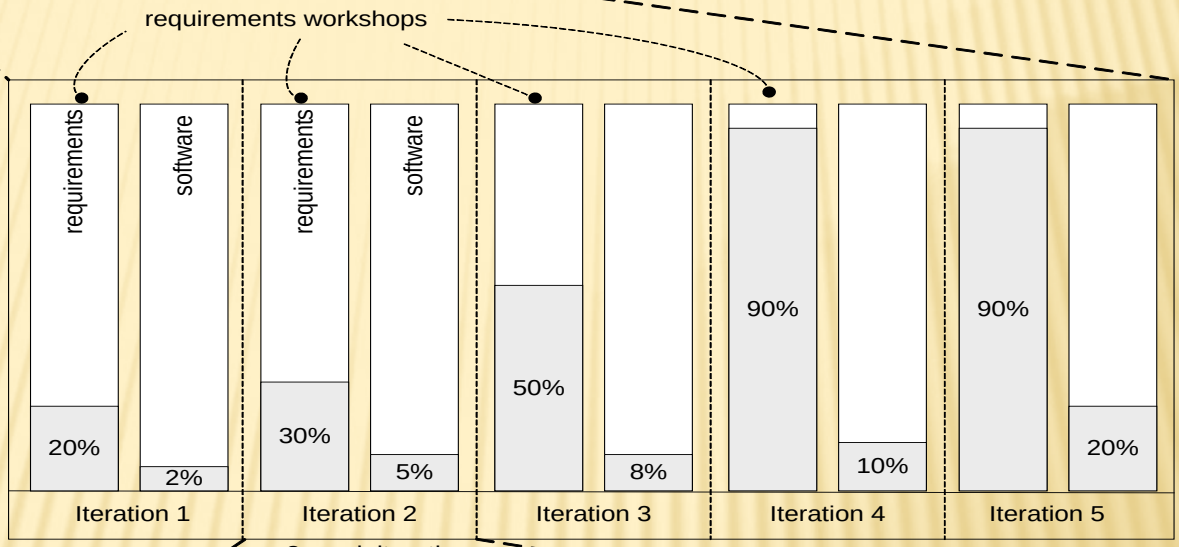
- Requirement workshop prior to iteration-1
- Risk-driven and client-driven planning
  - architecturally significant
  - high business value
  - high risk
- Iteration planning meeting prior to iteration-1

HOW TO DO AGILE?

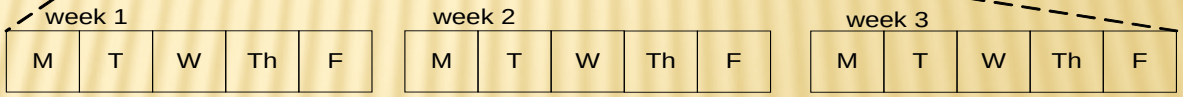


Imagine this will ultimately be a 20-iteration project.

In evolutionary iterative development, the requirements evolve over a set of the early iterations, through a series of requirements workshops (for example). Perhaps after four iterations and workshops, 90% of the requirements are defined and refined. Nevertheless, only 10% of the software is built.



a 3-week iteration



kickoff meeting clarifying iteration goals with the team. 1 hour

team agile modeling & design, UML whiteboard sketching. 5 hours

start coding & testing

Most OOA/D and applying UML during this period

de-scope iteration goals if too much work

final check-in and code-freeze for the iteration baseline

demo and 2-day requirements workshop

Use-case modeling during the workshop

next iteration planning meeting; 2 hours

BENEFITS

- Reduce failure
- Early attack on high risk issues
- Early visible progress
- Managed complexity
- Learning improves development process

GRASP

General Responsibility Assignment Software Patterns or Principles (GRASP)

Pattern – General principles and idiomatic solutions that can be applied

- A name
- Problem description
- Solution description

A learning aid for OOD with responsibilities.

- 9 patterns
  - Creator
  - Information expert
  - Low coupling
  - Controller
  - High cohesion
  - Polymorphism
  - Pure Fabrication
  - Indirection
  - Protected Variations

APPLYING GOF DESIGN PATTERNS

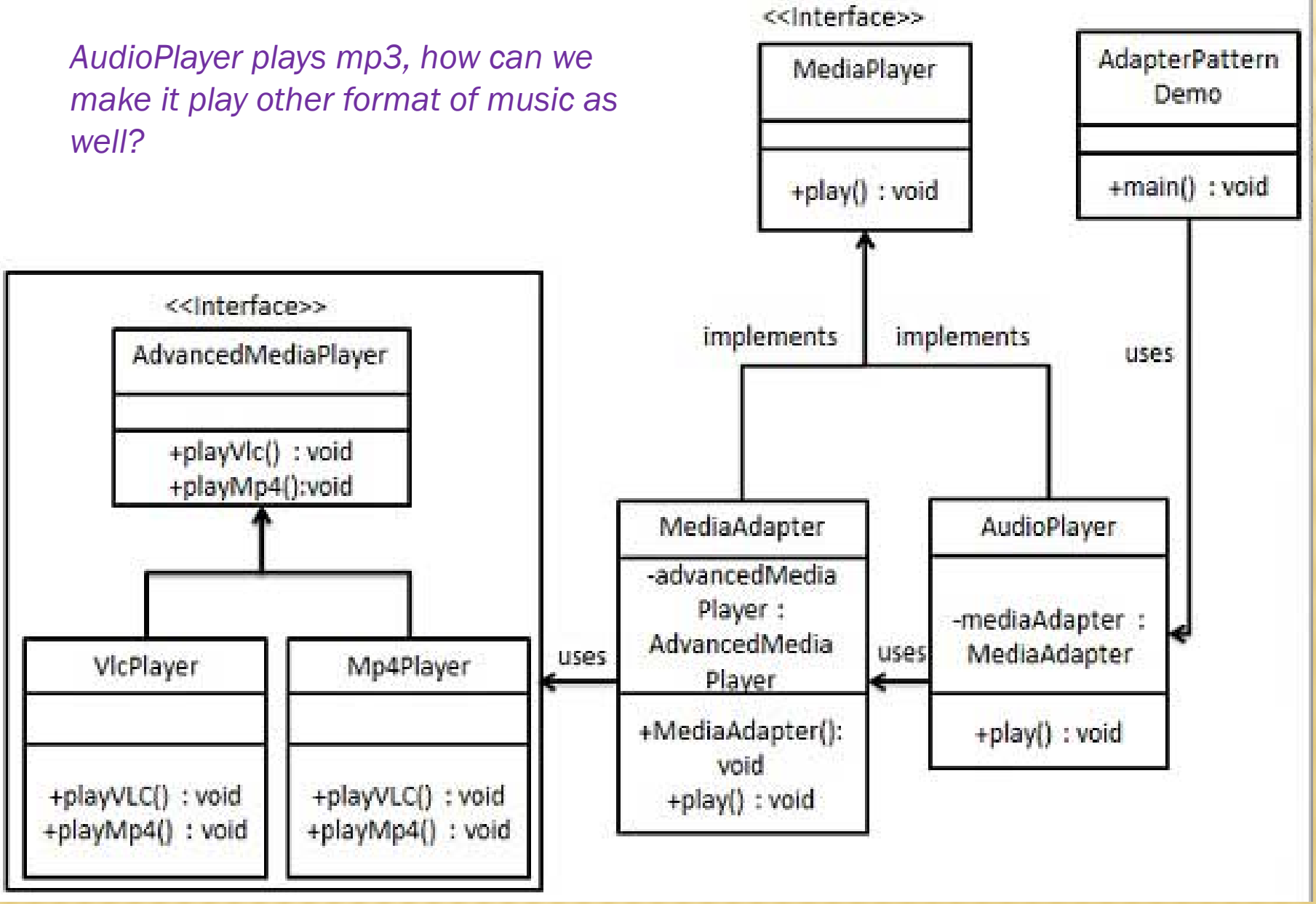
- Creational Patterns
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- Structural Patterns
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Facade
  - Flyweight
  - Proxy
- Behavioral Patterns
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

ADAPTOR

- Resolve incompatible interfaces
- Uses indirection
- Convert the original interface of a component into another interface, through an intermediate adapter object.

http://www.tutorialspoint.com/design_pattern/

AudioPlayer plays mp3, how can we make it play other format of music as well?



GRASP VS. GOF

- GRASP is at work in other patterns
  - *Adapter supports **Protected Variations** with respect to changing external interfaces or third-party packages through the use of an **Indirection** object that applies interfaces and **Polymorphism**.*
- Look for underlying principles
  - *Most design patterns can be seen as specializations of a few basic GRASP principles.*

TEST DRIVEN DEVELOPMENT

- Write the test first (also known as the test-first development)
- Show that test fails
- Write code to pass the test
- Run whole suite of tests!

WHY?

- Testing gets done!
- Programmer satisfaction
- Clarification of interface and behavior
- Provable repeatable verification
- Confidence to change things