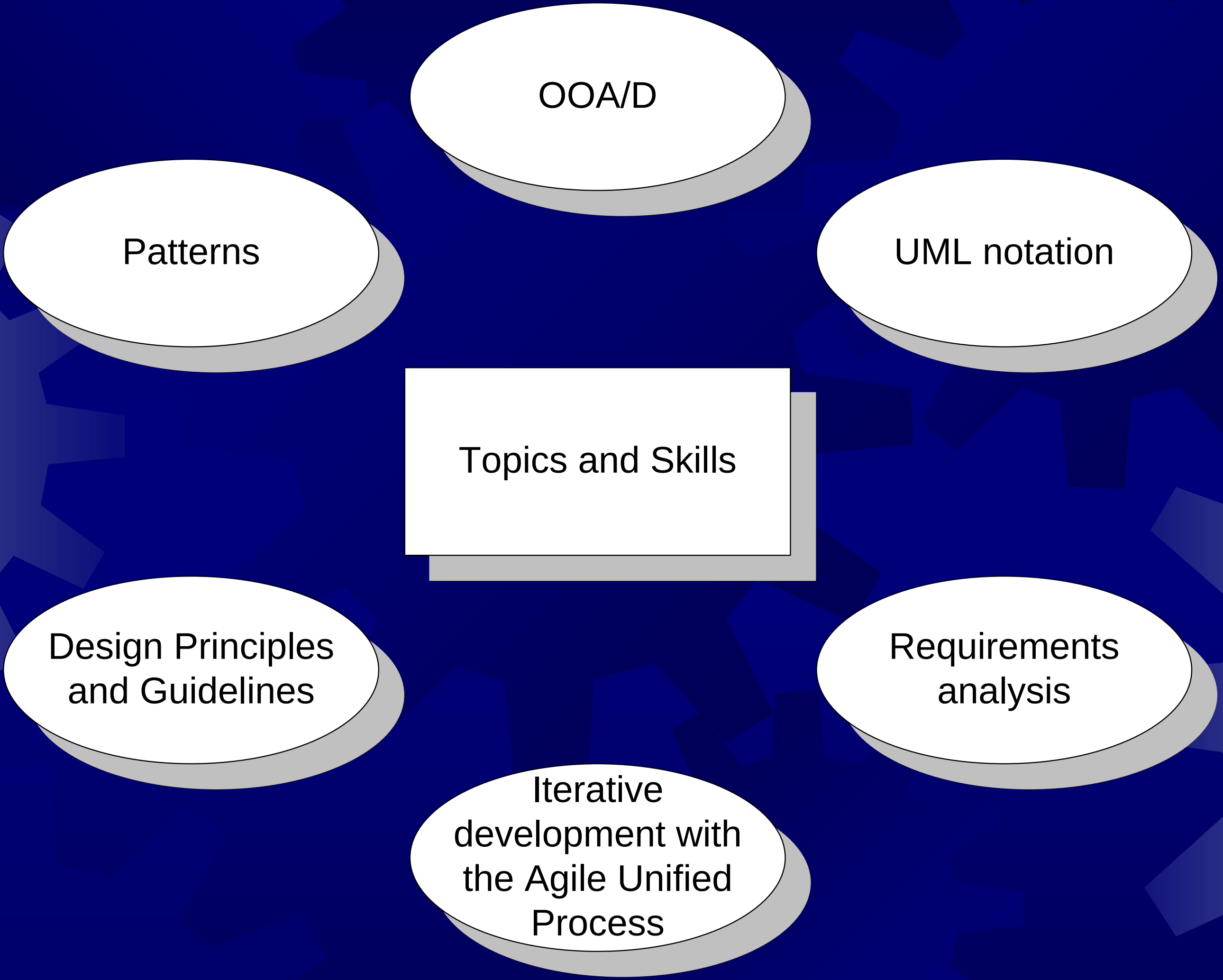
Object Oriented Analysis and Design Introduction

COMP 3831

Craig Larman: Chapters 1,2,3

Topics and Skills covered



Topics and Skills

OOA/D

UML notation

Requirements analysis

Iterative development with the Agile Unified Process

Design Principles and Guidelines

Patterns

What is the course about?

- Learn how to “**think in objects**”
- **Analyze the Requirements** of the problem domain
- **Design a solution**
  - Assign responsibilities to objects
  - Design patterns
- **Do Iterative Development** following the Agile Unified Process

AGILE = LIGHT + FLEXIBLE

- **Learn UML** 2 notation
- Apply best **Principles and Guidelines**
- Learn basic concepts of **Quality Assurance**
- Gain **Hands On** experience with the above

Unified Modeling Language (UML)

- Visual Language for
  - Specifying
  - Constructing
  - Documenting

  the artifacts of a system

- Notation used to record analysis and design

Agile Modeling

- Tools don’t compensate for bad design
- We need to gain good OO design and programming skills
- Over Analyzing and Designing - Death by UML Fever

Agile Modeling - uses UML as sketch

Agile Modeling - key to effective UML

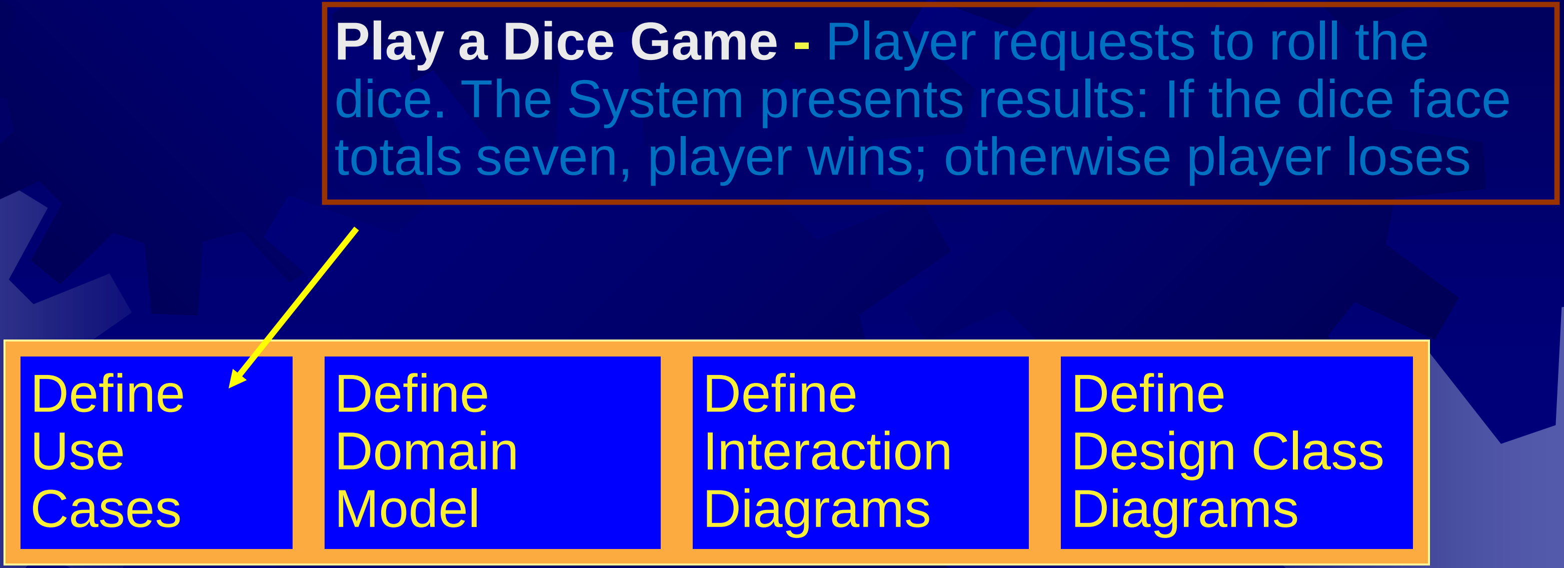
Patterns

- What are Design Patterns?
  - Certain tried-and-true solutions to design problems that can be expressed by both:

    - The Design problem
    - The Good design solution for the problem, in a given context

The Dice Game example

1. Define Use Cases



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

Define Use Cases

Define Domain Model

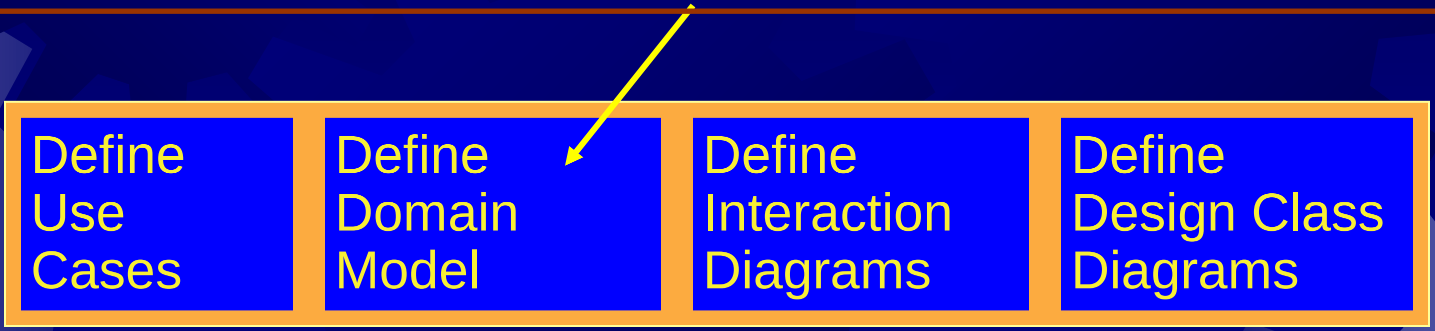
Define Interaction Diagrams

Define Design Class Diagrams

- Requirements Analysis identifies stories or scenarios of how people use the application
- Use Cases are stories written in a specific format
- Use Cases are not an OO artifact

2. Define a Domain Model

Object Oriented Analysis defines the domain model that shows all the noteworthy domain concepts: *Player*, *Die* and *DiceGame* with their associations and attributes.

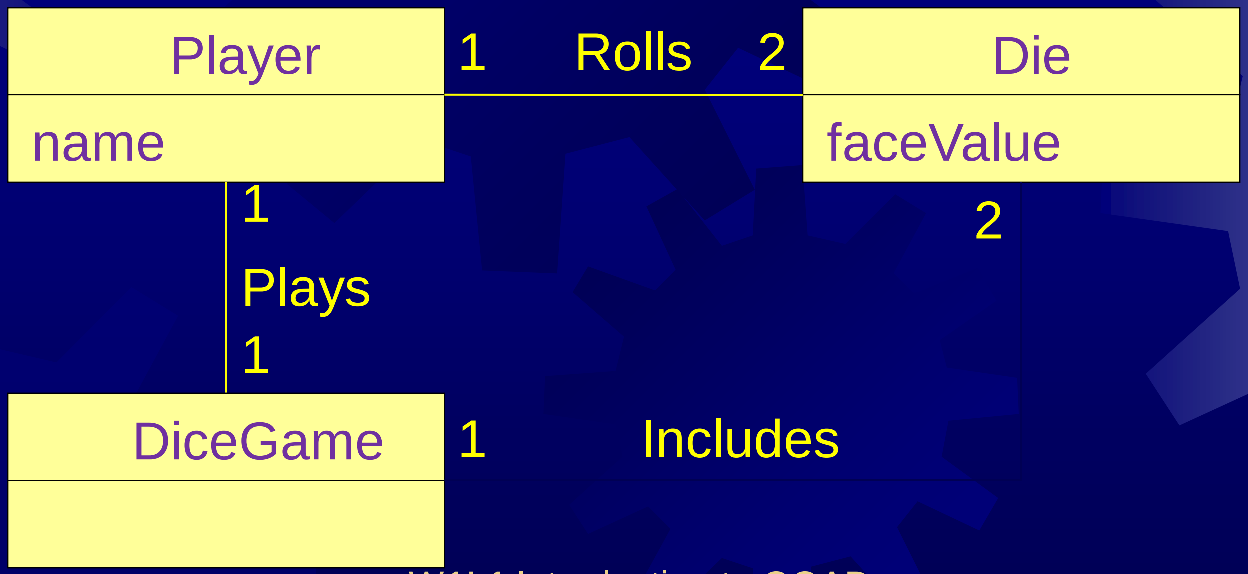


Define Use Cases

Define Domain Model

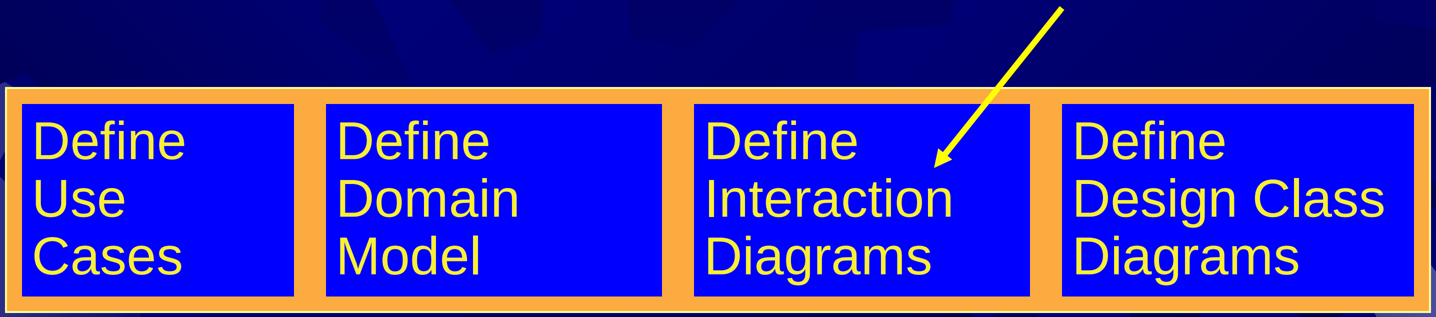
Define Interaction Diagrams

Define Design Class Diagrams



3. Assign Object Responsibilities and Draw Interaction Diagrams

Sequence Diagrams show messages between software objects

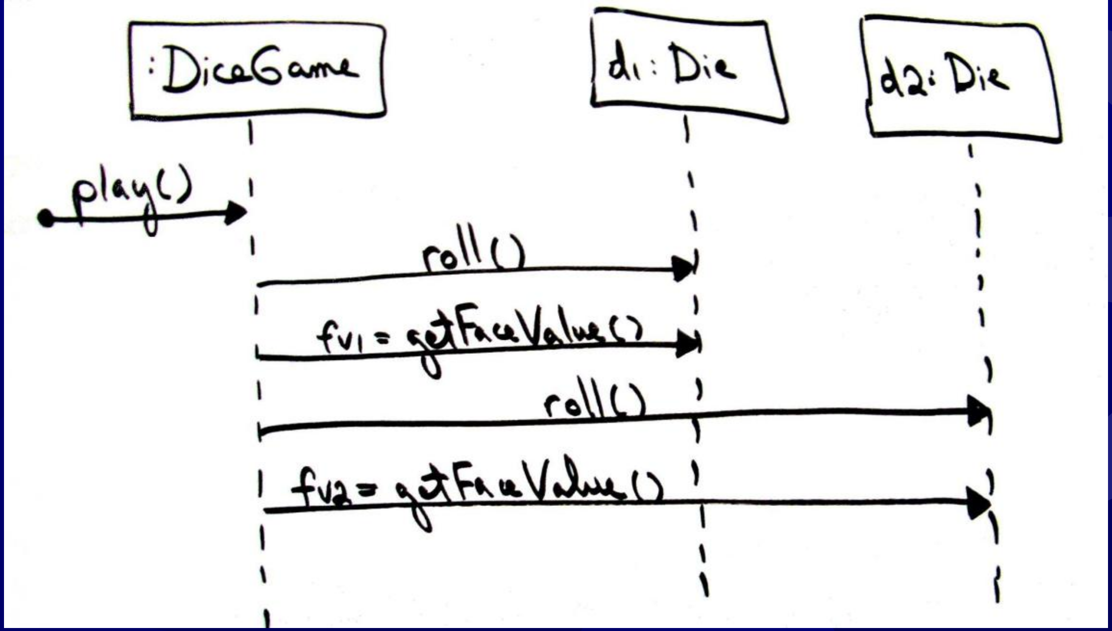


Define Use Cases

Define Domain Model

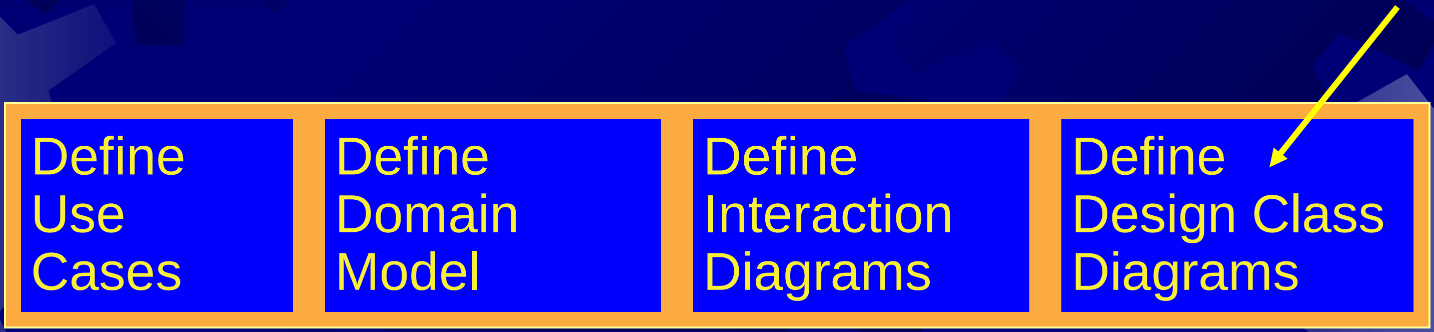
Define Interaction Diagrams

Define Design Class Diagrams



4. Define Design Class Diagrams

Class Diagrams show a static view of the class definitions with their relationships, attributes and methods.

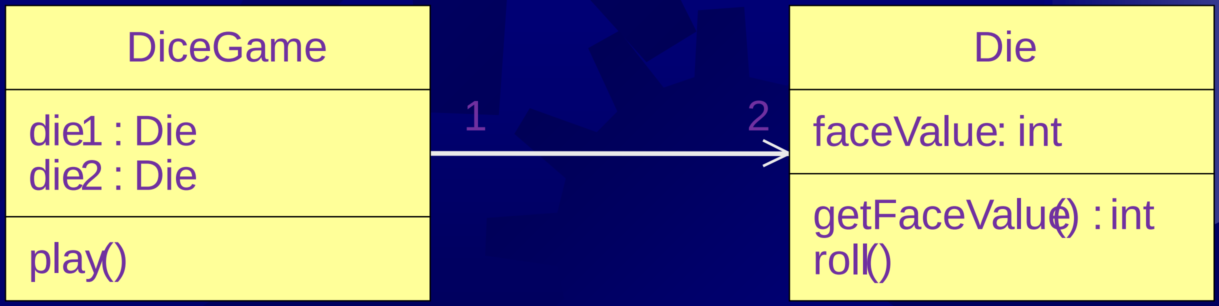


Define Use Cases

Define Domain Model

Define Interaction Diagrams

Define Design Class Diagrams



DiceGame

die1 : Die
die2 : Die

play()

1

2

Die

faceValue: int

getFaceValue() : int
roll()

Why OOA&D?

- Development time will be the same or even longer than using functional means
- Historically object oriented environments have been associated with slower execution

One Really Good Reason to Use OOA&D

- Allows the construction of a "clear" picture of a complex system.

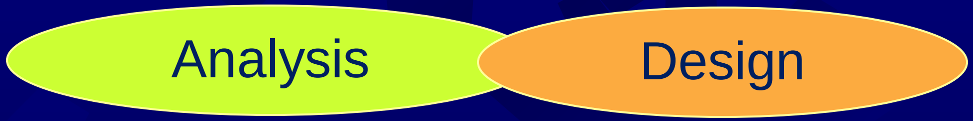
- Facilitates the construction of systems that are easier to:
  1. Understand
  2. Maintain
  3. Extend

Purposes of Analysis and Design

- To transform the requirements into a design of the system to-be.
- To evolve a robust architecture for the system.
- To adapt the design to match the implementation environment, designing it for performance.

Analysis (What?)

- Build the model of a real world situation
- Show important properties
- No implementation concepts or decisions
- No clear cut division between analysis and design



Analysis

Design

Design (How?)

- Design model based on the analysis model
- Implementation details added
- Data structures and algorithms
- Analysis classes gain implementation details

Analysis Versus Design

- Analysis
  - **Focus on understanding the problem**
  - Capture the Behavior
  - Understand the System structure, parts
  - Write the Functional Requirements
  - Create a small model

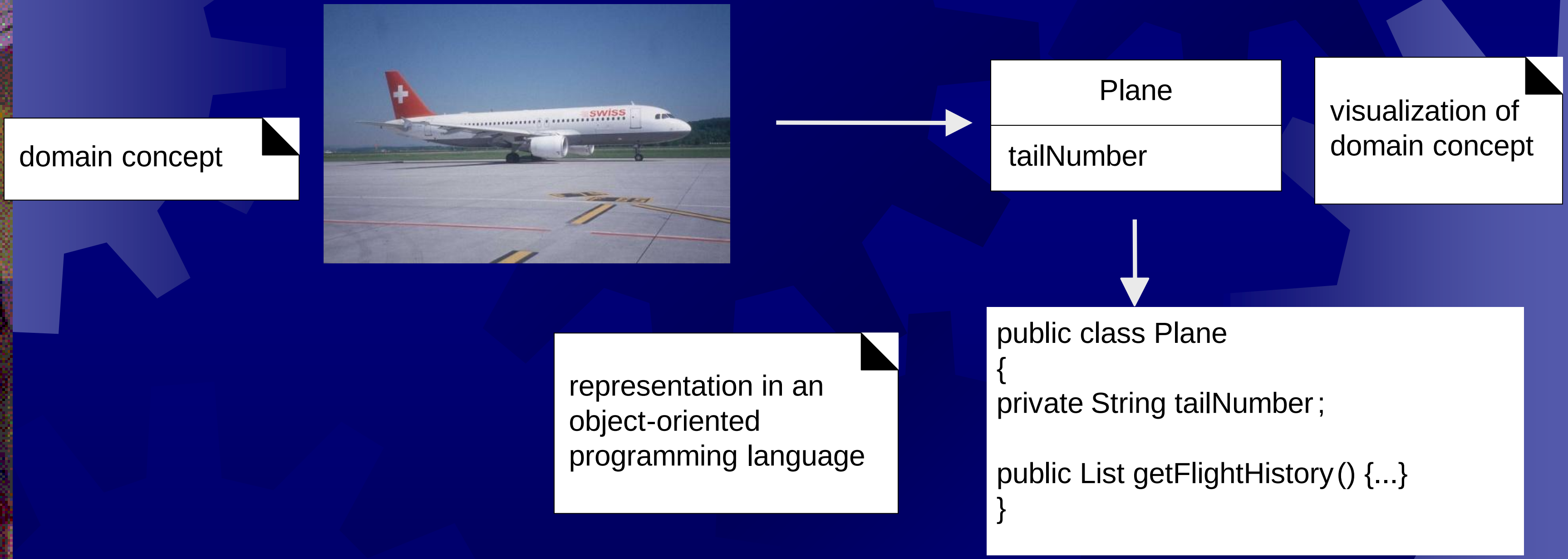
- Design
  - **Focus on understanding the solution**
  - Identify Operations and Attributes
  - Address Performance
  - Design is close to the real code
  - Include Non-functional requirements
  - Create the large model

Why Modeling?

1. Test before you build
2. Better Communication
3. Visualization
4. Break down complexity
5. Manage risks
   - All abstractions are incomplete and inaccurate
   - Different models can be made of the same system

Object-orientation - identify and represent the objects in the business domain

Example: A Flight Information System



domain concept

Plane

tailNumber

visualization of domain concept

representation in an object-oriented programming language

```
public class Plane
{
private String tailNumber;

public List getFlightHistory() {...}
}
```

Four Major Principles of OO

Abstraction	The essential characteristics of an entity that distinguish it from all other kind of entities and thus provide crisply-defined boundaries relative to the perspective of the viewer.
Encapsulation	The physical localization of features (e.g., properties, behaviors) into a single black box abstraction that hides their implementation (and associated design decisions) behind a public interface. Encapsulation is also referred to as information hiding
Polymorphism	Polymorphism is the ability to define a single interface with multiple implementations.
Inheritance	The mechanism that makes generalization possible; a mechanism for creating a new class using an existing classes as a foundation.

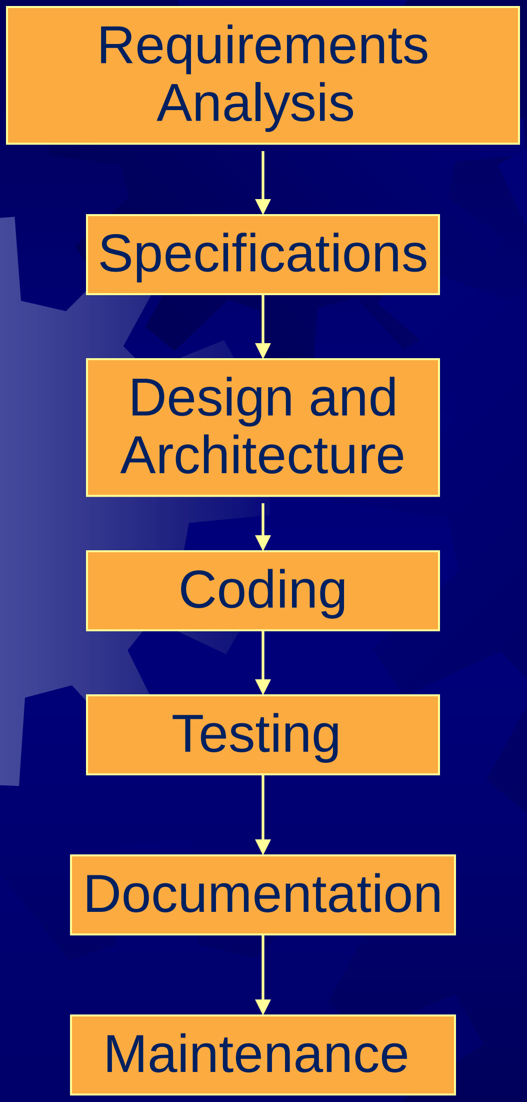
Questions

Iterative, Evolutionary, and Agile

Software Development Process

- Describes an approach to building, deploying, and maintaining software
- Software engineering is the profession that creates and maintains software applications by applying technologies and practices from computer science, project management, engineering, application domains, and other fields.
- Find repeatable, predictable processes or methodologies that improve productivity and quality.

Steps in Software Development Process



Requirements Analysis

Extracting the requirements of a desired software product may require skill and experience to recognize incomplete, ambiguous or contradictory requirements.

Specifications

Specifications is the task of precisely describing the software to be written. Specifications are most important for external interfaces, that must remain stable.

Design and Architecture

Design and architecture refer to determining how software is to function in a general way without being involved in details. Usually this phase is divided into two sub-phases.

Coding

Reducing a design to code may be the most obvious part of the software development job, but it is not necessarily the largest portion.

Testing

Testing of parts of software, especially where code by two different programmers must work together.

Documentation

An important (and often overlooked) task is documenting the internal design of software for the purpose of future maintenance and enhancement. Documentation is most important for external interfaces.

Maintenance

Maintaining and enhancing software to cope with newly discovered problems or new requirements. This can take far more time than the initial development of the software.

Types of Software Development Process

Waterfall Processes

The best-known and oldest process is the waterfall model, where developers follow the steps in order.

Iterative Processes

Development is organized into a series of short, fixed-length mini-projects called iterations. The outcome of each is a tested, integrated and executable *partial* system.

Agile Processes

Agile processes are built on the foundation of iterative development by adding a more people-centric viewpoint than traditional approaches. Agile processes use feedback, rather than planning, as their primary control mechanism.

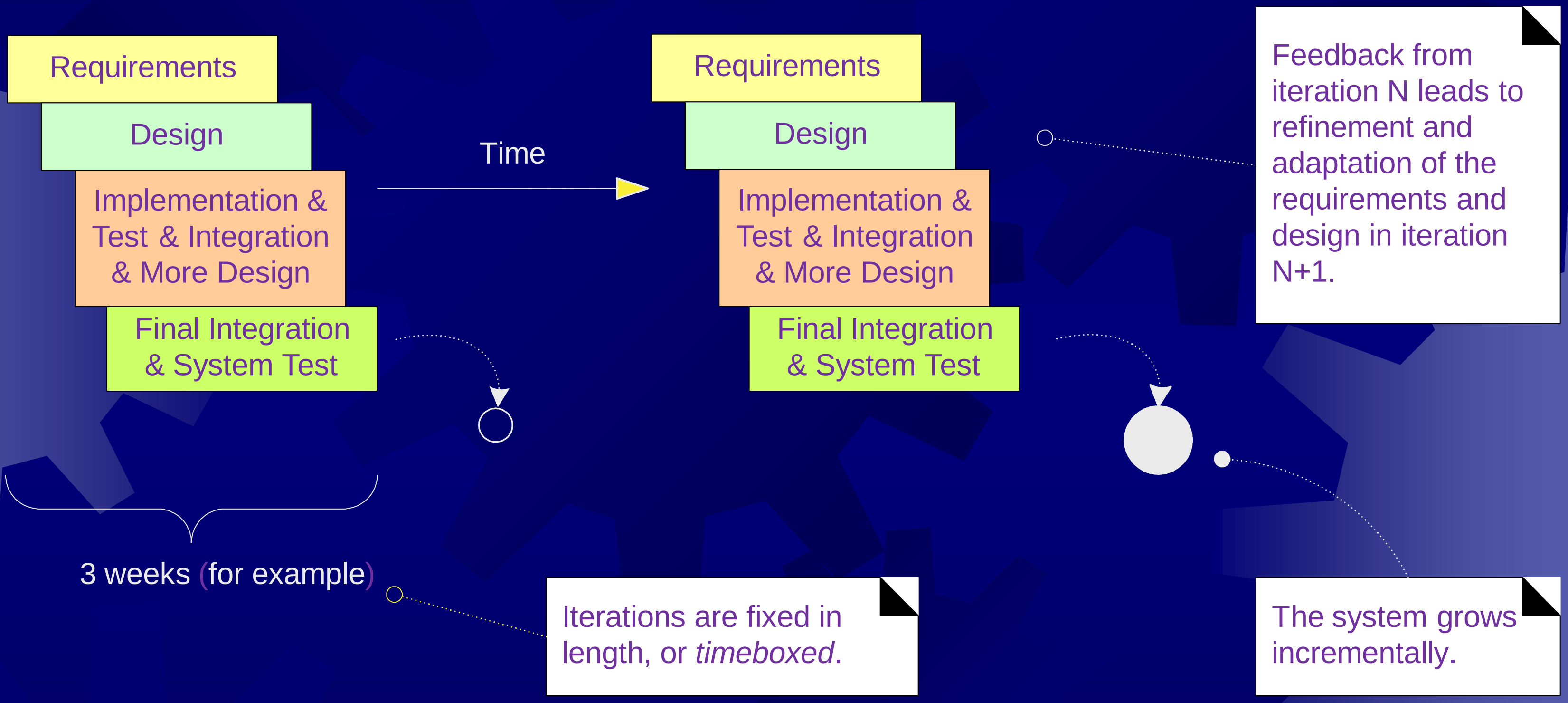
Iterative, Incremental and Evolutionary Development

Iterations = Short, fixed-length mini-projects

  - Repeating cycles
  - Early programming
  - Early testing

- System grows incrementally
- Feedback to evolve the specifications and design

Iterative Diagram

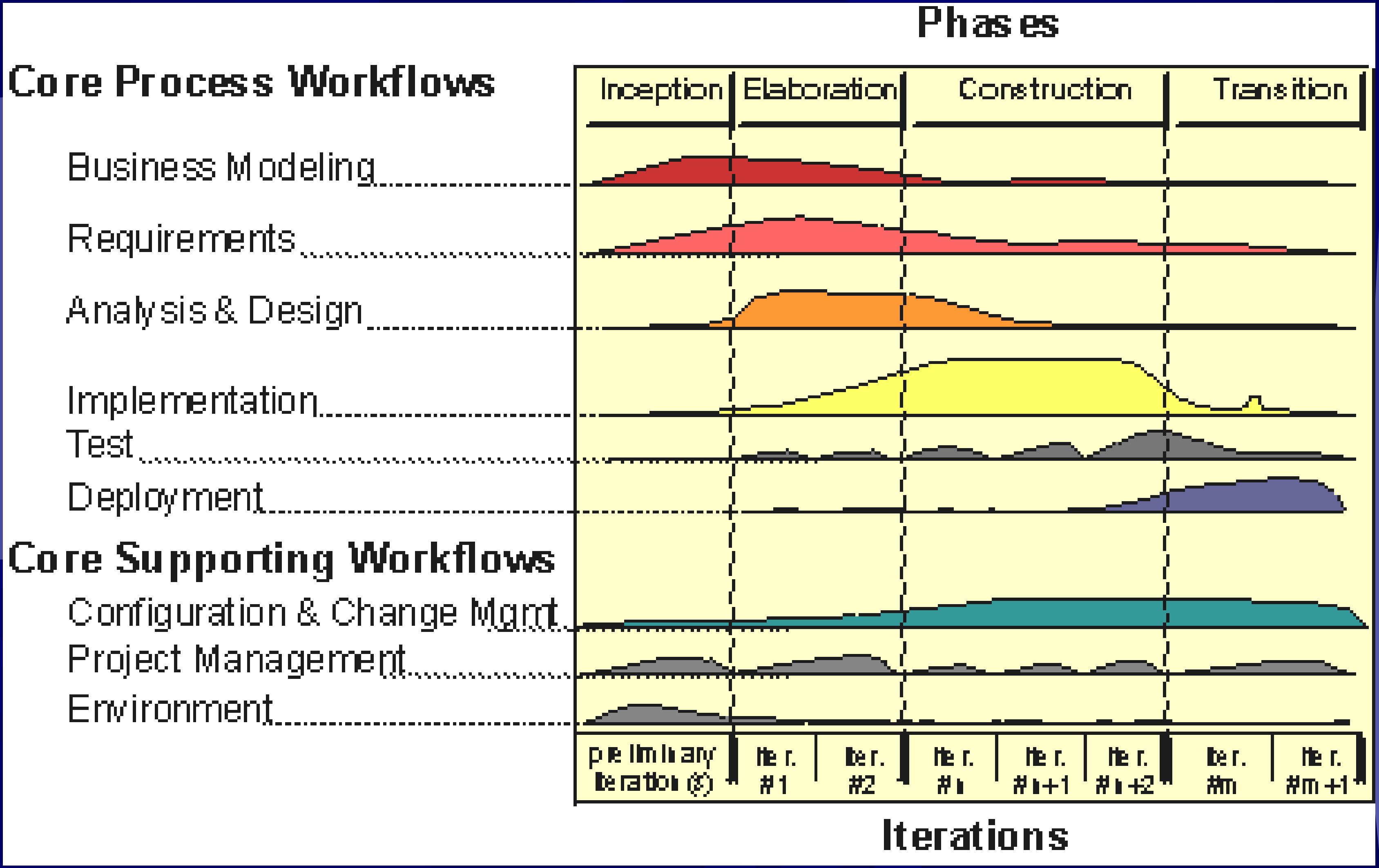

Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
Time
Requirements
Design
Implementation & Test & Integration & More Design
Final Integration & System Test
3 weeks (for example)

Feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1.

Iterations are fixed in length, or *timeboxed*.

The system grows incrementally.

The Unified Process (UP) – The Humpback Chart



UP Phases

- A RUP Lifecycle consists of 4 Phases
- Every Phase has one or more iterations
- An Iteration is like a mini waterfall complete development cycle

Inception	Elaboration	Construction	Transition
▪Identify the Objective ▪Create Business Case ▪Establish Project Management Infrastructure	▪Detail Design ▪Establish a project plan and a sound architecture	▪Write and grow the software	▪Deliver the System to the users

Disciplines and Artifacts

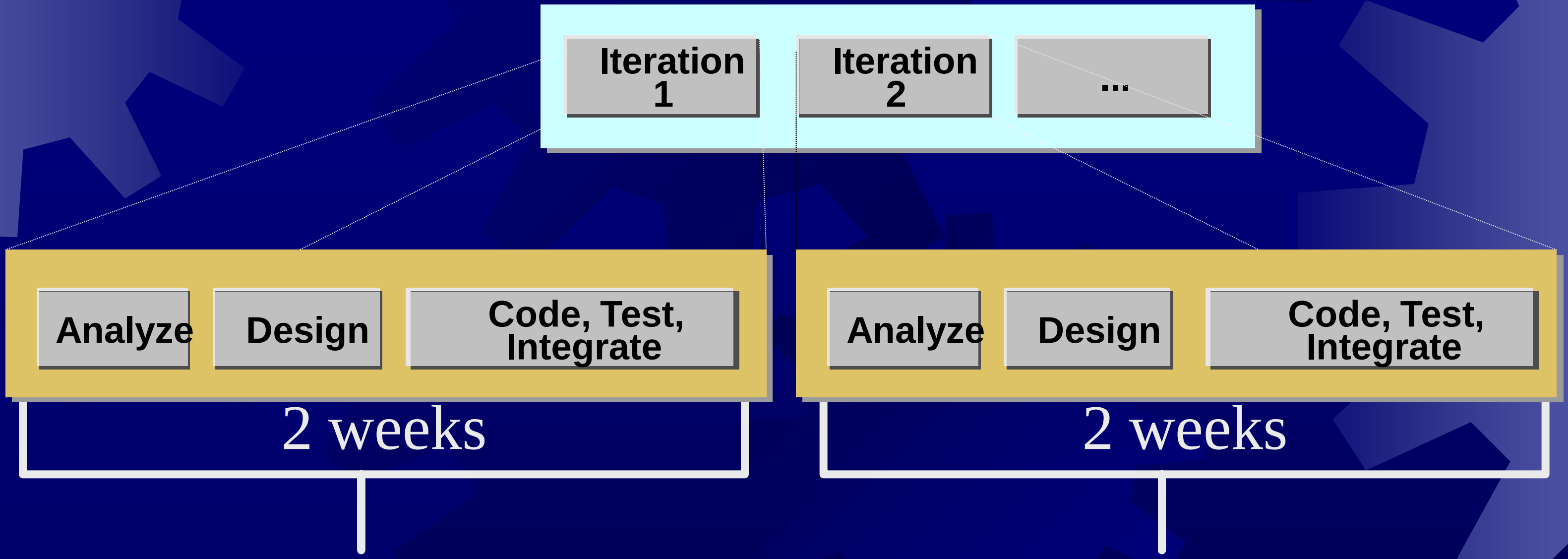
- **Discipline** = set of activities and the related artifacts in one subject area
- **Artifact** = any work product or deliverables: code, Web graphics, database schema, text documents, diagrams, models, etc.
- **Rational Unified Process** provides a complete set of Artifacts (template and guide) for each Discipline, within each Phase

Waterfall Errors and UP Benefits

- **Waterfall** is based on two very inaccurate assumptions:
  - We can freeze requirements
  - We can get the design right on paper before proceeding to the next step.
- **Unified Process** benefits
  - Risks are addressed early
  - Change is more manageable
  - Higher opportunity for reuse
  - Latitude for learning and improvement
  - Team focused on outcome

Iteration Time-boxing

- Iteration length between 2 to 6 weeks
- Small steps, rapid feedback, adaptation
- The result of each iteration is an executable system, may not be ready for production deployment



Risk-Driven, Client-Driven, Iterative Planning

Goals of early iterations

- High risks or unknowns
- High business value
- Core architecture

Paticipants in Requirements Workshops and Iteration Planning meetings

- Business People
- Development People
- Chief Architect

Agile Development = Rapid + Flexible

- Iterative, Evolutionary and Timeboxed Development
- Incremental delivery
- Adaptive planning

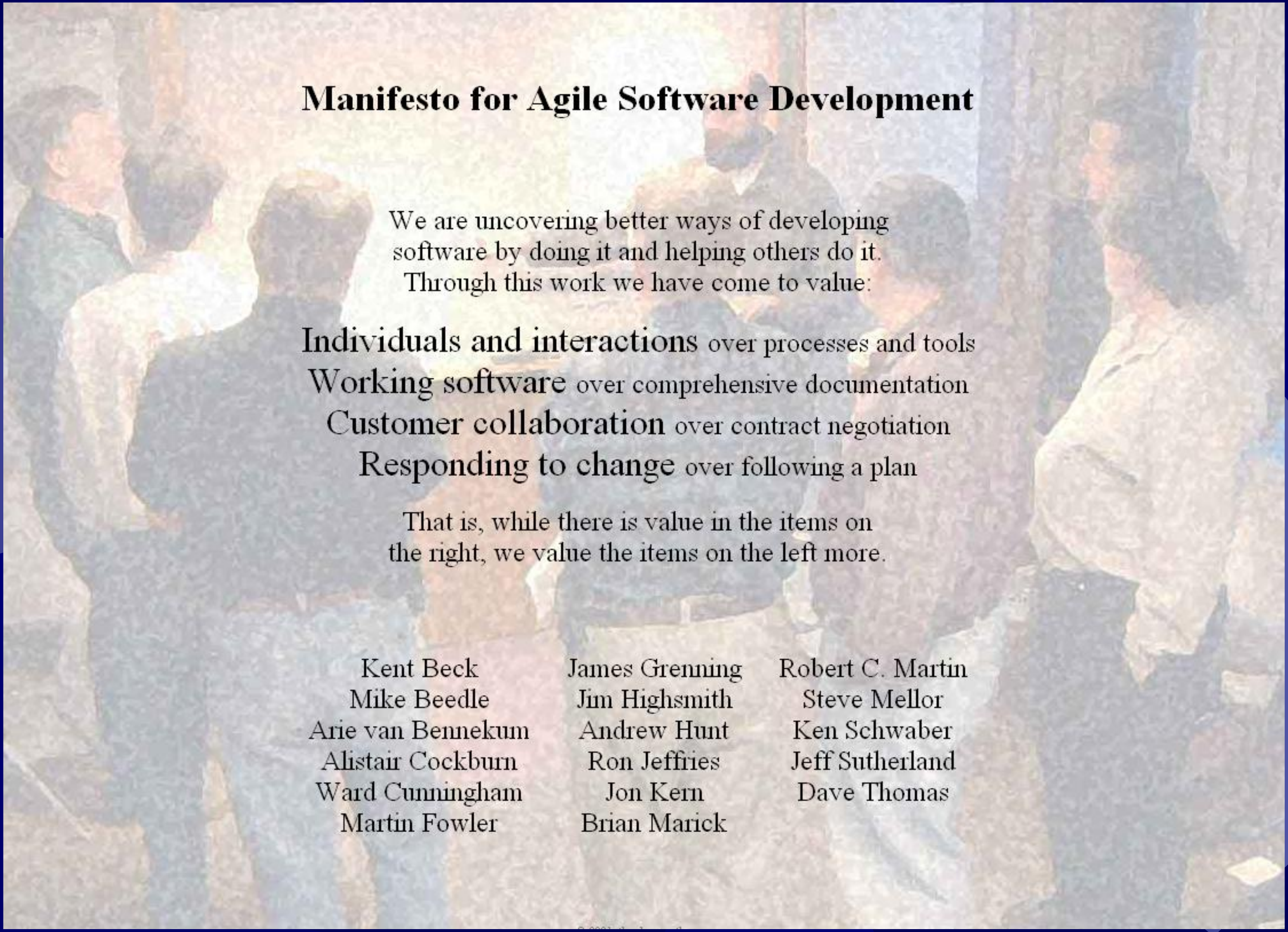
Work value

- Simplicity
- Lightness

Team value

- Communication
- Self-organizing teams

Manifesto for Agile Software Development



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Agile Development Principles (1)

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Agile Development Principles (2)

- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agile Development Principles (3)

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.

Agile Development Principles (4)

- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Questions and Conclusions