Software Engineering

Lecture 6
GSL Peru 2014

Housekeeping

- Please turn in your High Level Product Specification
- No classes on holiday next Monday 28th and Tuesday 29th
- Friday’s are not optional
- Video Crews in this Thursday and Friday, 24th and 25th

Roadmap

Review

- Persona, Value Creation, Strategy
- Software Design

This Week

- Finish Super High Level Business Plan
- High Level Prototype Plan

Moving Forwards

- Executive Summary
- Prototype

Now that you have Use Cases...

What next?

Objective

- On Time
- High Quality
- Meets user/persona needs - product must allow user/persona to realize value

Software Project Management

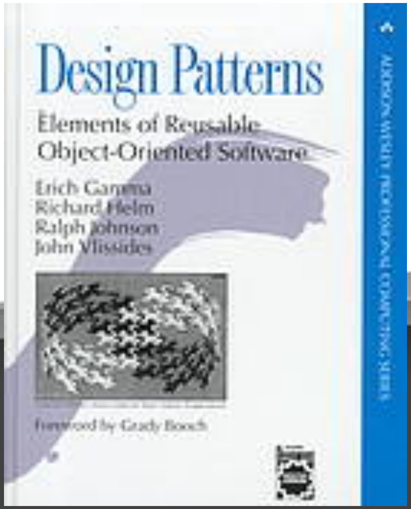
- Someone must manage the activities:
  - What needs to be done?
  - When? - Scheduling
  - Who?
    - Assignment
    - Resource management
    - Coordination - Team building
    - Morale management
    - Administrative

Simplified Project Management

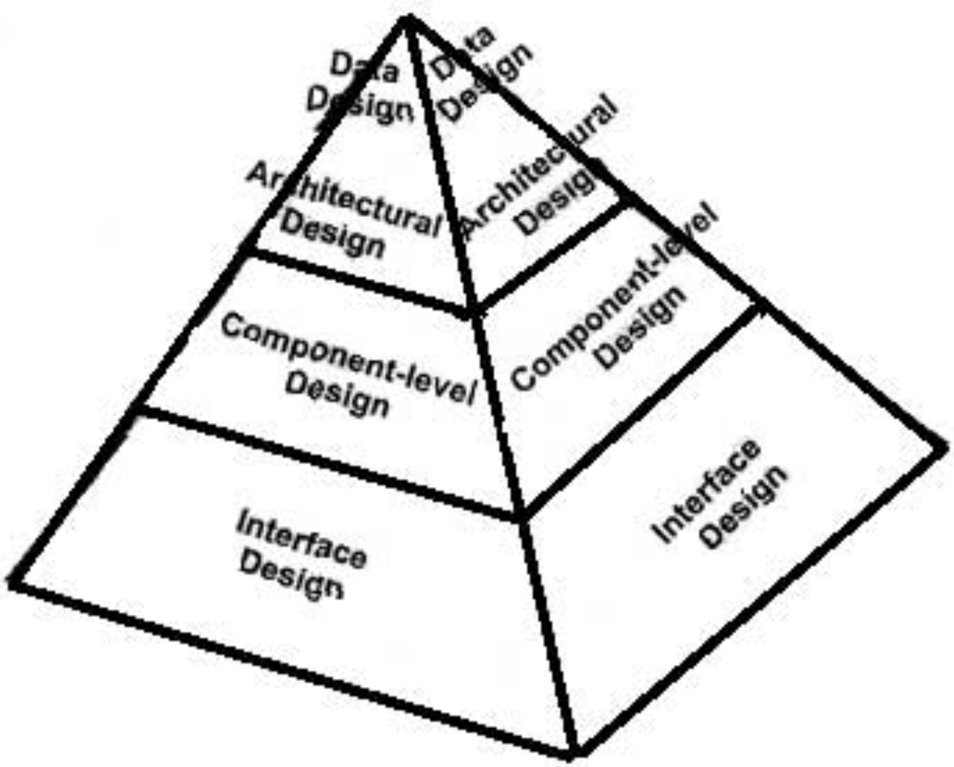
- Determine Tasks
- Order Tasks
- Estimate Tasks
- Estimate Productivity
- Calculate Time Required
- Estimate Available Time
- Create Schedule
- Track Progress

Software Design

- Coding != Software Design
- Need experience
  - Design Patterns: Elements of Reusable Object-Oriented Software
  - Use parts of another project as template
- Very difficult, even if you have experience


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Data Model



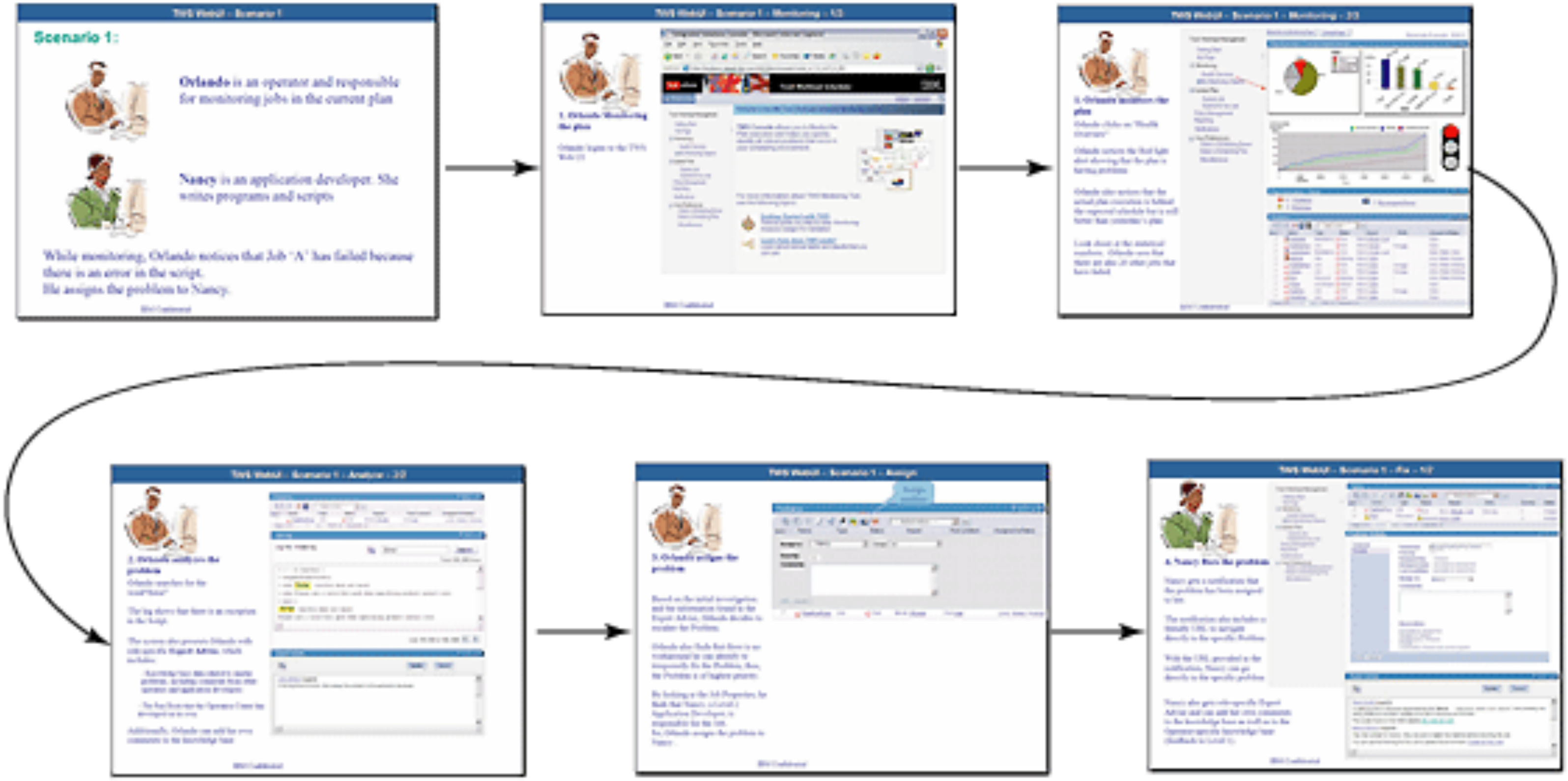
Design Model and its Elements

Where to start...

Analyze

- Understand how requirements translate to technology from the user/persona’s perspective.
- Do not rely just on your perspective.
- Break up components at the high level.
- Mock up UI.

UI Storyboard



TWS WebUI – Scenario 1

Scenario 1:

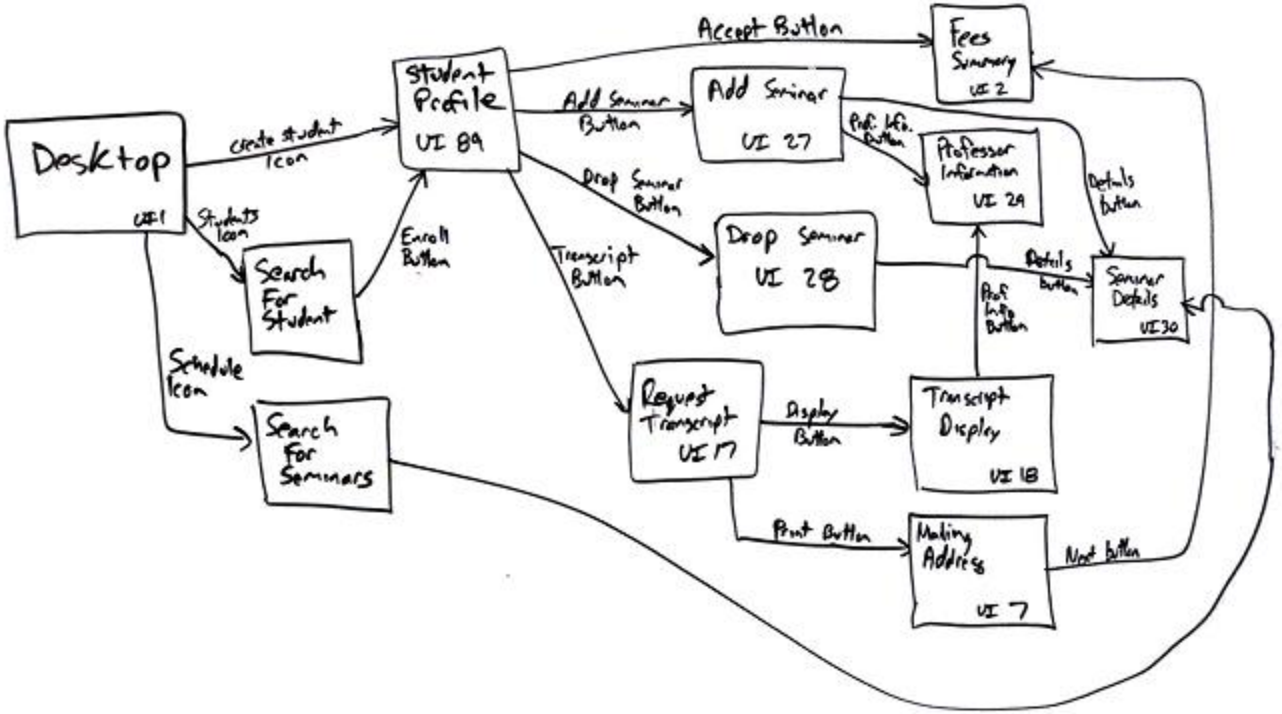
Orlando is an operator and responsible for monitoring jobs in the current plan

Nancy is an application developer. She writes programs and scripts

While monitoring, Orlando notices that Job "A" has failed because there is an error in the script.
He assigns the problem to Nancy.

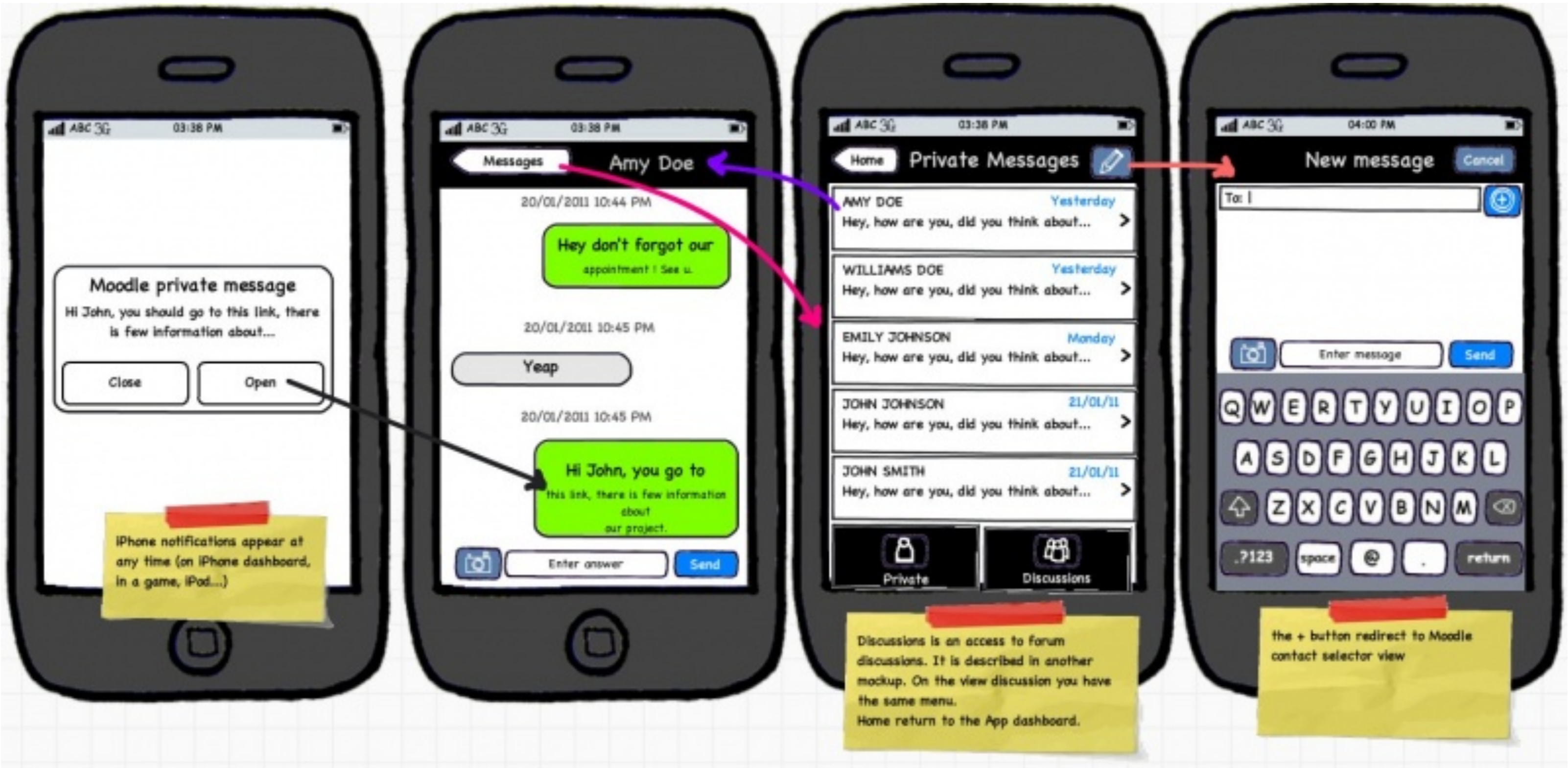
Source: http://www.ibm.com/developerworks/rational/library/06/0404_donatelli/

UI Flow Diagram (Storyboard)



Source: http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm

Create UI Mockups/Wireframes



Hi John, you should go to this link, there is few information about...

20/01/2011 10:44 PM

Hey don't forgot our appointment ! See u.

20/01/2011 10:45 PM

Yeap

20/01/2011 10:45 PM

Hi John, you go to this link, there is few information about our project.

AMY DOE	Hey, how are you, did you think about...	Yesterday
WILLIAMS DOE	Hey, how are you, did you think about...	Yesterday
EMILY JOHNSON	Hey, how are you, did you think about...	Monday
JOHN JOHNSON	Hey, how are you, did you think about...	21/01/11
JOHN SMITH	Hey, how are you, did you think about...	21/01/11

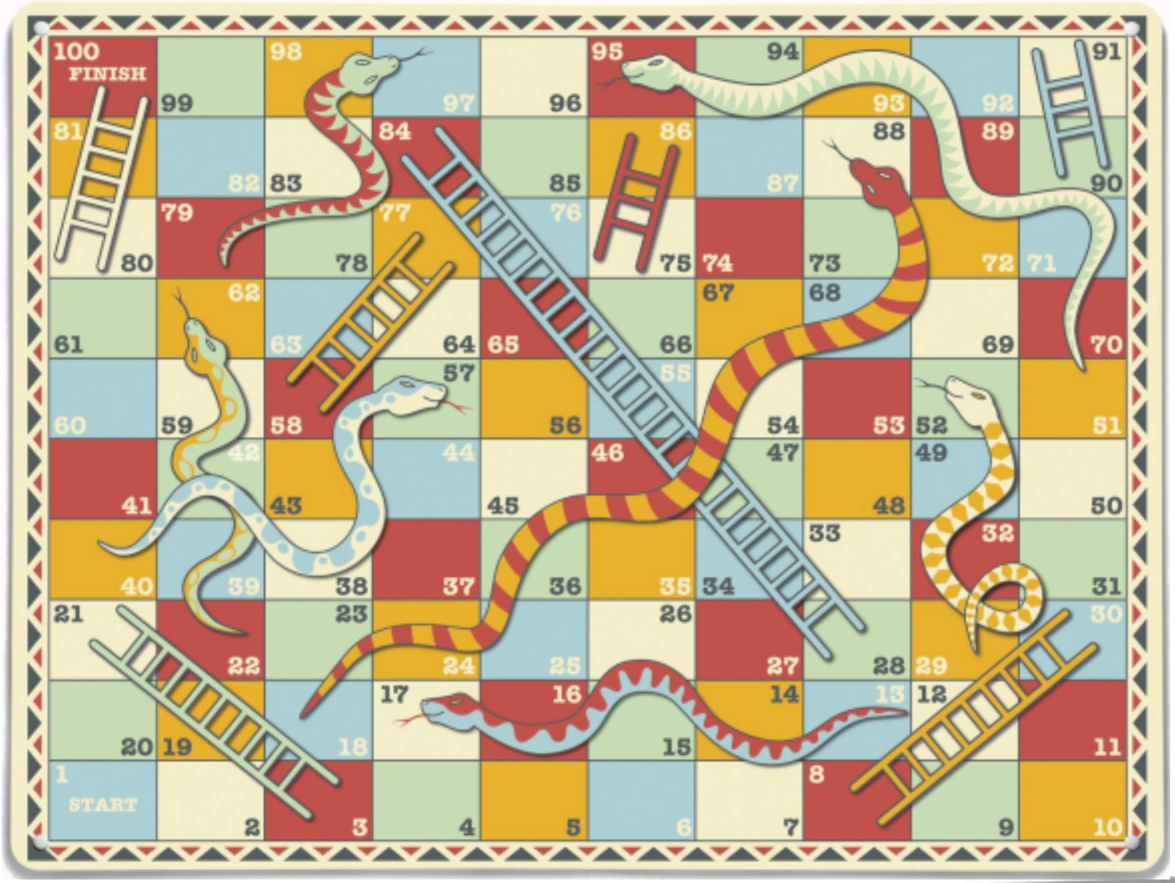
iPhone notifications appear at any time (on iPhone dashboard, in a game, iPod...)

Discussions is an access to forum discussions. It is described in another mockup. On the view discussion you have the same menu.
Home return to the App dashboard.

the + button redirect to Moodle contact selector view

Source: http://depann2000.com/gallery/temp/balsamiq-mockups-examples

Snakes and Ladders



Snake and Ladders Rules

- Players - 2-4 players move tokens around the board
- Moving - players must role a die, move specified number of spaces (1-6), and perform any actions
- Ladders - if a player lands on a ladder, they climb to the top of the ladder
- Snakes - if a player lands on a snake, they must slide down the snake to the bottom
- Winning - the player that lands on the last space by either landing on it or by using the ladder.

Determine Components

Data Modelling

- Identify Data Objects and Attributes needed to support Use Cases
- Examine them independent of processing
- Abstract objects at the level of users/personas

Snake and Ladders - Data Objects

- Game Board
- Squares
  - Start
  - Finish
- Ladders
- Snakes
- Players
- Dice

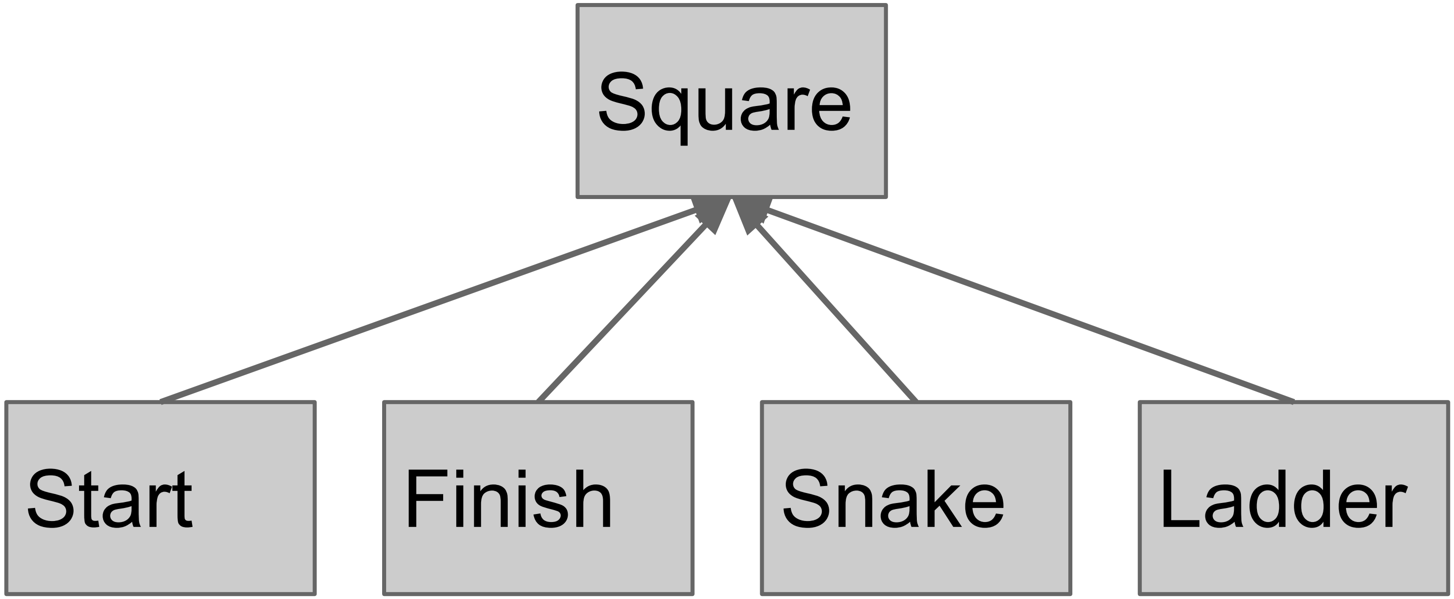
Determine Relationships and Interactions

Class Based Modelling

Expand Data Model to Class Model

- Objects
- Operations
- Relationships
- Collaboration

Snake and Ladders



Is Snake a reverse Ladder?

Responsibilities

- The responsibility should be generic as possible
- System logic should be distributed in a way to best solve the problem at hand
- Information and related behavior should reside in the same class
- Information regarding a specific item should only exist in a single class and spread across multiple classes.
- When applicable, responsibilities can be shared among related classes

Snake and Ladders - Responsibilities

- Game - keeps track of the state
- Square - keeps track of player on it
- Start - can hold multiple players
- Finish - knows its the winning square and game finish
- Snake - sends a player down
- Ladder - sends a player up
- Player - keeps track of the location, moves along the square
- Die - generates random number between 1-6

Collaboration

Classes can

- manipulate its own data
- collaborate with other classes

Collaboration identifies relationships

- is-part-of relationship
- has-knowledge-of relationship
- depends-upon relationship

Interfaces

- When defining relationships or collaboration, use interfaces
- Should be well defined
- Help insure modular design
- Use Abstract Base Class if interfaces are not available

Snake and Ladders

```
public class Game {
    private List<ISquare> _squares;
    private Queue<Player> _players;
    private Player _winner;
    ...
}
```

Snake and Ladders

```
public class Player {
    private String _name;
    private ISquare _square;
    ...
}
```

Snake and Ladders

```
public class Square implements ISquare {
    protected int _position;
    protected Game _game;
    private Player _player;
    ...
}
```

Snake and Ladders

```
public class Square implements ISquare {
    private Player _player;
    public boolean isOccupied() {
        return this._player != null;
    }
    public void enter(Player player) {
        this._player = player;
    }
    public void leave(Player player) {
        this._player = null;
    }
    ...
}
```

Snake and Ladders

```
public interface ISquare {
    public int position();
    public ISquare moveAndLand(int moves);
    public boolean isFirstSquare();
    public boolean isLastSquare();
    public void enter(Player player);
    public void leave(Player player);
    public boolean isOccupied();
    public ISquare landHereOrGoHome();
}
```

Snake and Ladders

```
public class Square implements ISquare {
    private Player _player;
    public void enter(Player player) {
        this._player = player;
    }
    ...
}
public class StartSquare extends Square {
    private List<Player> _players;
    public void enter(Player player) {
        this._players.add(player);
    }
    ...
}
```

Snake and Ladders

```
public class Player {
    public void moveForward(int moves) {
        _square.leave(this);
        _square = _square.moveAndLand(moves);
        _square.enter(this);
    }
    ...
}
```

```
public class Square implements ISquare {
    public ISquare moveAndLand(int moves) {
        return _game.findSquare(position, moves).landHereOrGoHom
    }
    ...
}
```

```
public class Game {
    public ISquare findSquare(...) {
        …
        return this.getSquare(target);
    }
    ...
}
```

Design Principles and Concepts

Design Principles

- The design process should not suffer from ‘tunnel vision.’
- The design should be traceable to the analysis model.
- The design should not reinvent the wheel.
- The design should “minimize the intellectual distance” between the software and the problem as it exists in the real world.
- The design should exhibit uniformity and integration.

Design Principles

- The design should be structured to accommodate change.
- The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.
- Design is not coding, coding is not design.
- The design should be assessed for quality as it is being created, not after the fact.
- The design should be reviewed to minimize conceptual (semantic) errors.

Fundamental Concepts

- Abstraction - data, procedure, control
- Informal Hiding - controlled interfaces
- Refinement - elaboration of abstraction
- Architecture - overall structure
- Modularity - division into modules
- Patterns - proven solutions
- Stepwise Refinement - sequence of decomposition
- Refactoring - process of improvement
- Structural Partitioning - vertical or horizontal
- Functional independence - single-minded function and low coupling

Prototype

- Build a prototype to verify technical assumptions
- Can be good way to iterate vague requirements
- Prototype != Beta or Production code
  - Misconception that the product is close to finished
  - Too many shortcuts
  - Rewrite for quality
- Not same as Agile Methodology

Circle Back

- Validate that the requirements are met.
  - Functional
  - Non-functional
  - Usability
- If users cannot extract value from your product, no sale.

Simplified Seven Design Principles

1. The Reason It All Exists - to provide value to its users
2. KISS (Keep It Simple, Stupid!)
3. Maintain the Vision - Clear Vision
4. What You Produce, Others Will Consume
5. Be Open to the Future
6. Plan Ahead for Reuse
7. Think

References

- Snake and Ladder example - Object-Oriented Design Principles - Oscar Nierstrasz (http://scg.unibe.ch/download/lectures/p2/P2-02-OODesign.pdf)