

# Object-Oriented Analysis/Design and Use Cases

## Object Oriented Analysis and Design

Aron Trauring

T++ Technical Skills Training Program

CUNY Institute for Software Design & Development (CISDD)

New York Software Industry Association (NYSIA)

December 6th, 2004

# What Is Analysis and Design?

- Analysis emphasizes an investigation of the problem and requirements, rather than a solution
- Usually qualified — requirements analysis, object analysis
- OOA emphasizes finding and describing the objects (concepts) in the problem domain
- Design emphasizes a conceptual solution that fulfills the requirements, rather than its implementation
- Usually qualified — database design, object design

- OOD emphasizes defining software objects and how they collaborate to fulfill the requirements
- OOP — implementing the design

## Why Analysis and Design?

- Goal: *Do the right thing (analysis), and do the thing right (design)*
- Success criteria: working code [*do the thing right*] which meets customer needs [*do the thing right*]
- Focus of OO Analysis and Design

# Object-Oriented Analysis and Design

- Anarchistic Model — OO is about “Those Who Know, Decide”
- “Desert Island” Skill — Ability to skillfully assign responsibilities to software components
- Doing it right determines flexibility, robustness, maintainability, and re-usability of software components

## How Much Analysis and Design?

- All methods and techniques apply to all processes — differ by how much not that there are stages
- Waterfall/SDLC — Finish analysis, then design and only then implement — predictive model
- Unified Process — Iterate, although spend some time doing the right thing — up front analysis — *before* doing the thing right
- Extreme Programming — Working code is the way to ensure you will do the right thing — minimal up-front A & D

## Three Steps of Every Iteration

1. Domain Model — the concepts, attributes, and associations that are considered noteworthy — static and behavioral
2. Domain Layer — software objects and their collaborations (protocols, interfaces) — Static and Behavioral
3. Software Implementation

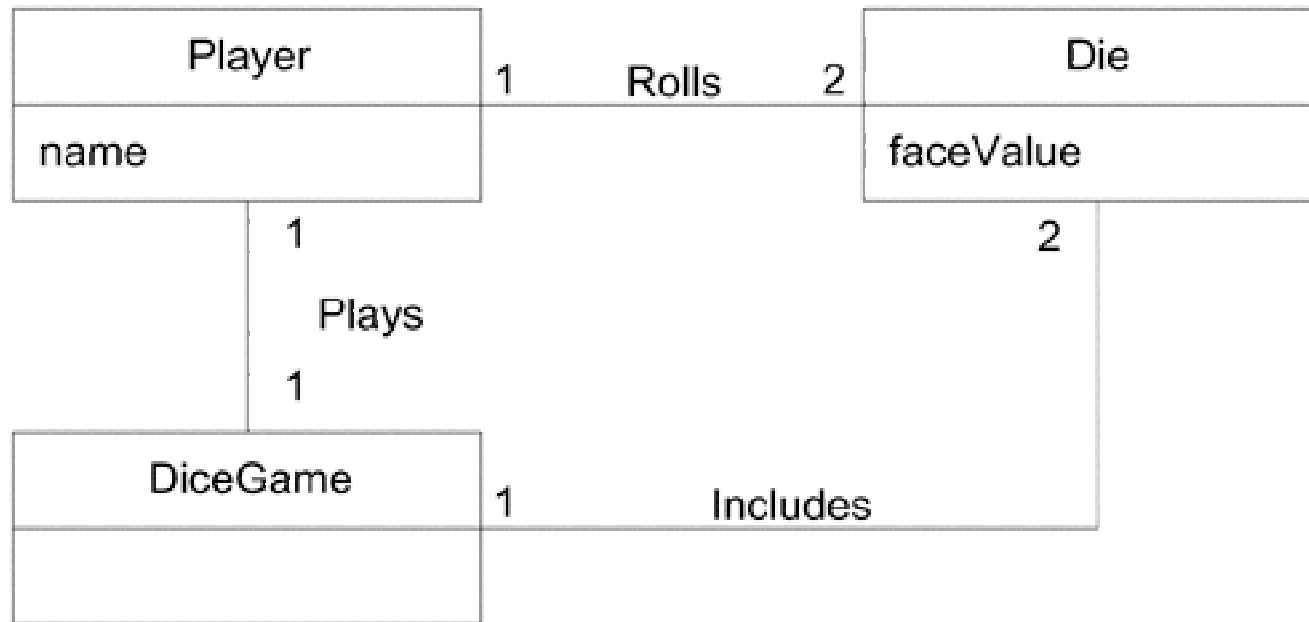
## Simple Example

- Dice Game: player rolls two die. If the total is seven, they win; otherwise, they lose
- Requirements — Use Case:

**Play a Dice Game:** A player picks up and rolls the dice. If the dice face value total seven, they win; otherwise, they lose

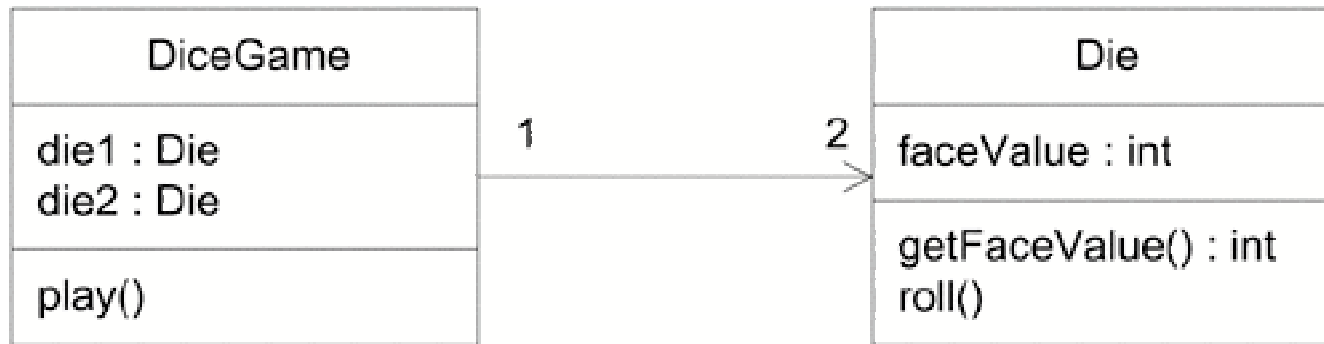
- Domain Model — visualization of concepts in the real-world domain through a set of diagrams

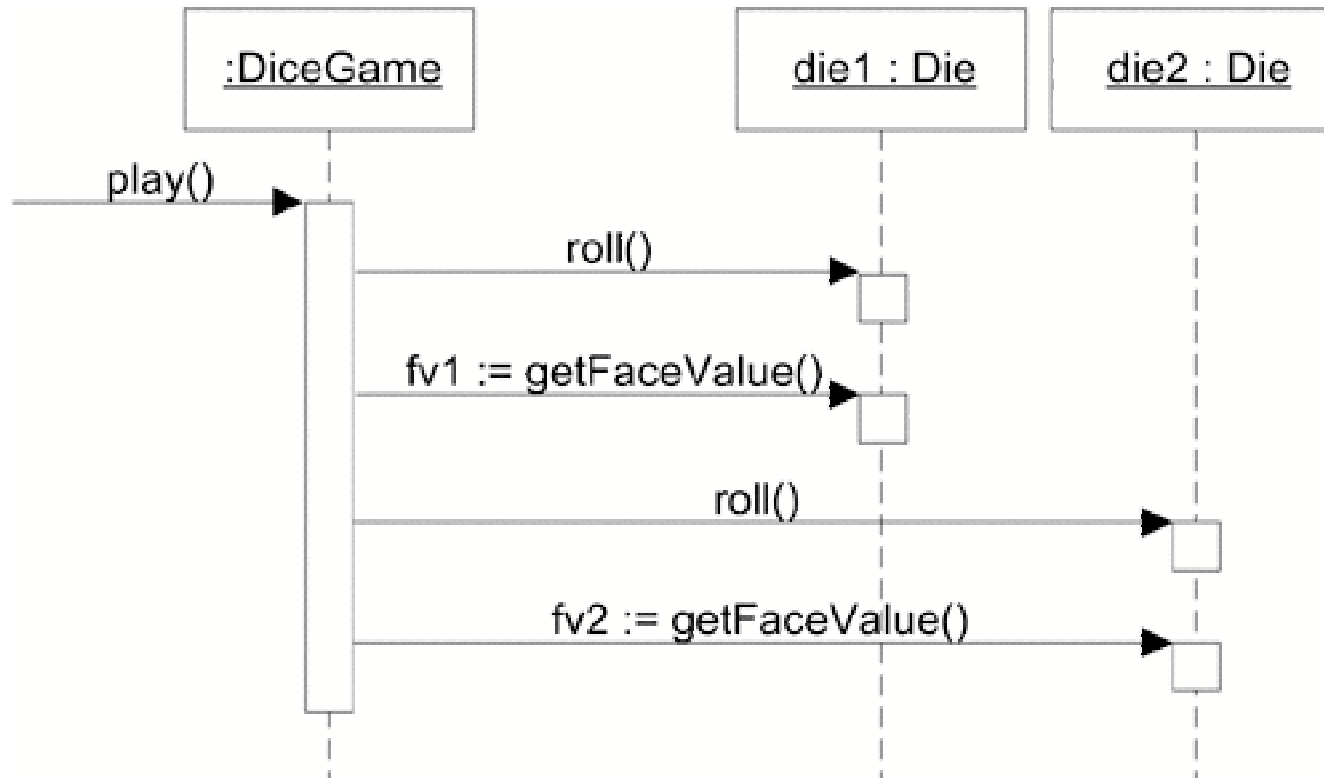
# Conceptual Class Diagram



- Domain Layer — “inspired by” the real world domain model

# Protocol Class Diagram and Interaction Diagram





- Software Implementation — "Inspired by" domain layer

T++ — CISDD — NYSIA

Use Cases - OOAD

- UML — probably not

## Class in the Three Perspectives

- Conceptual Die — a real-world die
- Protocol Die — software data type
- Implementation Die — Python (Java,C++,C#) Class
- Case Study 1 — Point of Sale System (POS)

## Information System Architectural Layers

- User Interface — thin layer, little responsibility
- Application Logic and Domain Objects — heart of the system
- Technical Services — usually application-independent and reusable across several systems

## What are Use Cases?

- Not specifically OO but first step in every analysis
- Use cases are little structured stories about the system
- In XP they are short stories called “User Stories”
- In SDLC they are Dickens novels
- Describe a set of scenarios, that is interactions, between the user and a system, related to a specific user goal

## Key Terms

- *Actor* — something with behavior,; a person (identified by role), computer system, or organization
- *Scenario* — specific sequence of actions and interactions between actors and the system under discussion (SUD)
- *Use Case* — collection of related success and failure scenarios that describe actors using a system to support a goal

## Informal Example

### Handle Returns

*Main Success Scenario:* A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item ...

*Alternate Scenarios:*

If the customer paid by credit, and the reimbursement transaction to their credit account is rejected, inform the customer and pay them with cash.

If the item identifier is not found in the system, notify the Cashier and suggest manual entry of the identifier code (perhaps it is corrupted).

If the system detects failure to communicate with the external accounting system, ...

# Why Use Cases

- Story telling is oldest tool of human communication
- Customer can write or be involved in writing use cases
- Forms the basis of discussion between customer and design teams
- Emphasis on customer goals and perspective — *do the right thing*

## How Much Detail?

- Brief — One paragraph. Early stages of analysis (all you do for XP)
- Casual — Multiple paragraphs
- Fully Dressed — architecturally significant and high-value use cases only (in UP)
- Only working code matters

## Guidelines

- *Root Goal* — focus on real goal not mechanism to achieve them — “log in” vs “identification and authentication”
- *Essential Style Writing* — “keep the UI out; focus on intent”

Focus on user *intentions* and system *responsibilities*

1. Administrator identifies self
2. System authenticates identity

***Not*** concrete actions

1. Administrator enters user id and password in dialog box(see Figure 2)

2. System authenticates Administrator
3. System display the “edit users” window (see Figure 3)

- *Write tersely*
- *Black-Box Use Cases* — focus on *responsibilities* not internal workings
- Take Actor and Actor-Goal perspective — *do the right thing*

## How to find Use Cases

1. Choose the SUD boundary
2. Identify the primary actors
3. Identify the goals for each primary actor
4. Define use cases that satisfy user goals

## Useful Use Cases

- Boss Test (User Goal) — “What have you been doing all day?”

“Logging In” – Bad

“Negotiating a Supplier Contract” — Good

- Size Test (Sub-function) — not too small

“Enter an Item ID” — Bad

“Authenticate User” — Good

## Use Case Format

Use Case Section	Comment
<b>Use Case Name</b>	Verb-Noun structure
<b>Scope</b>	System Under Design
<b>Level</b>	“User Goal” or “Sub-function”
<b>Primary Actor</b>	Calls on the SUD to deliver its services
<b>Stake-holders and Interests</b>	Who cares about this use case and what do they want?
<b>Preconditions</b>	What must be true on start worth telling readers?
<b>Success Guarantee (Postconditions)</b>	What must be true on successful completion worth telling readers?
<b>Main Success Scenario</b>	A typical, unconditional “happy path” scenario of success
<b>Extensions</b>	Alternative scenarios of success or failure
<b>Special Requirements</b>	Related non-functional requirements
<b>Technology &amp; Data Variations List</b>	Varying I/O methods and data formats
<b>Frequency of Occurrence</b>	Influences investigation, testing and timing of implementation
<b>Miscellaneous</b>	Such as Open Issues

## Scope

- User-Goal Level — to fulfill the goals of the primary Actor
- Sub-function Level — sub-steps required to support a User Goal (e.g. *Pay by Credit*)
- Sub-function may be shared by several regular use cases

## Stake-holders — What Should Be in Use Case?

- SUD is contract between stake-holders
- Use Case gives behavioral part of contract
- Captures *all and only* behaviors relating to stake-holder interests

# Preconditions

- Not tested in use case
- Assumed to be true
- Implies successful completion of a scenario from another use case

## Main Success Scenario

- Leave conditional and branching to extensions
- Typical success path that satisfies the interests of the stake-holders
- Capitalize Actors names
- Number each step
- Put "repeat" statement after steps to be repeated

## Three Kinds of Steps

1. An interaction between actors (may be SUD) — “System presents receipt to Customer”
2. A validation (usually by system) — “Cashier verifies...”
3. A state change by the system — “System records sale”

## Extensions

- All the rest of the success and failure scenarios
- Numbering keys back to Main branch

### Main:

...

3. Cashier enters item identifier

...

### Extensions:

...

3a. Invalid Identifier: **[Identify Condition]**

1. System signals error and rejects entry **[Response/Handling]**

...

- Write the condition as something that can be detected by system or actor
- Condition can arise in a range of steps
- At end of extension handling merge back with main scenario unless indicated otherwise
- Complex extensions may be included in separate use case (indicate branch by underlining name)
- Condition that can occur during any step is indicated by “\*a”

\*a. At any time, system fails:

## Special Requirements

- Non-functional requirements — “Language Internationalization”
- Quality attribute (performance, reliability, usability) — “System must respond within 30 seconds”
- Constraint — “Touch Screen on LCD: text must be visible from 1 meter”
- Related to specific use case
- May be gathered together later

## Technology and Data Variations List

- Technical variations on how things are done — bar code reader vs. keyboard entry
- Variations in data schemes — “UPC or ISBN data formats”
- Early constraints — try to avoid if possible