W4156

OOAD III: OO Analysis

Recap

- OOAD paradigm
    - The problem is managing ***complexity***
    - OO attempts to control complexity through ***abstraction*** and ***encapsulation***
    - Within the paradigm
        - ***Everything*** is an ***object***
        - We ***model*** our *problem domain* in the paradigm (everything is Object!)
        - ***Language features*** support the paradigm

- We applied to 1-2 examples and saw potential power/usefulness

- Don’t yet have the tools to translate problems into OO code ……

Agenda

- ❑ Defining the analysis phase
- ❑ Process
- ❑ Worked Example

OOAD?

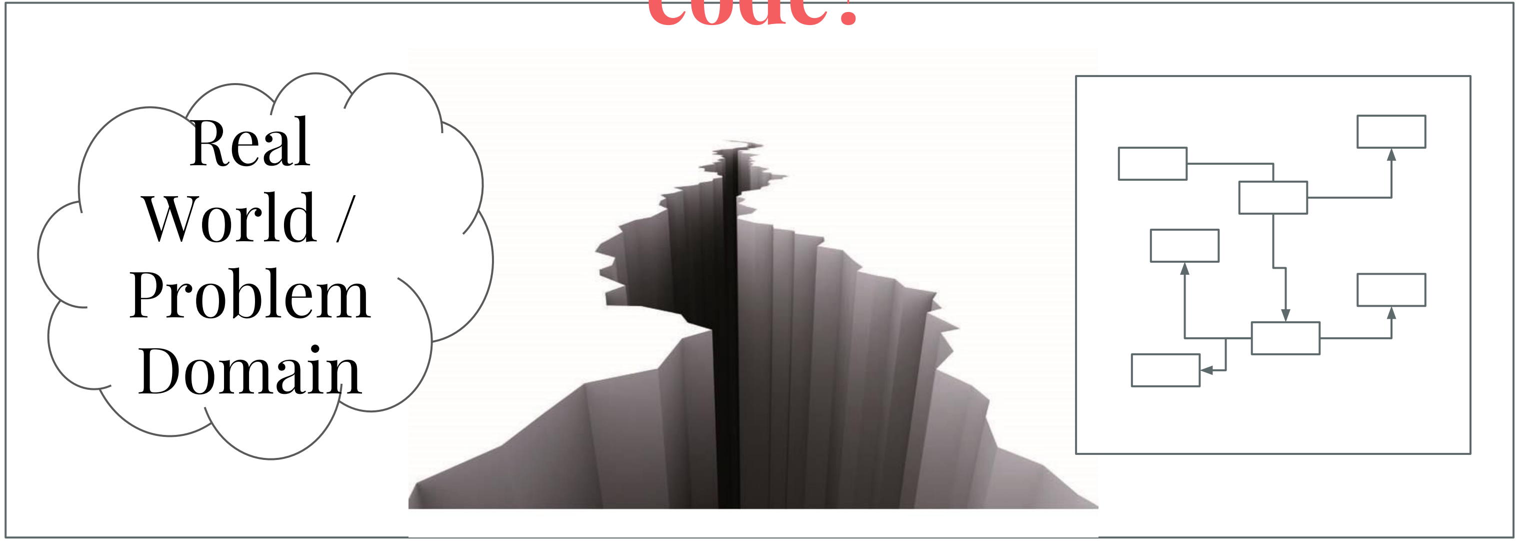
OOAD

"Object-oriented analysis and design is the method that leads us to an object-oriented decomposition"

- Booch

Note: There are many many different OOAD methods (all variations on the same goal). We will teach a simplified one (which should yield 80% of the value)

How do I ‘process’ my problem into OO code?



Ideally I would like techniques / series of steps
Apply to problem, turn handle and generate code[1] ……

[1] as you know by now it is never that easy / mechanical but we can get close

Simplified OOAD Process

Analysis Process

1. ~~Requirements~~
2. Define functionality of system
3. Develop conceptual model
    a. Identify conceptual classes
    b. Deduplicate
    c. Responsibilities and Key Elements
    d. Identify Relationships

Note: This does not yield complete code!!!

In simpler terms …

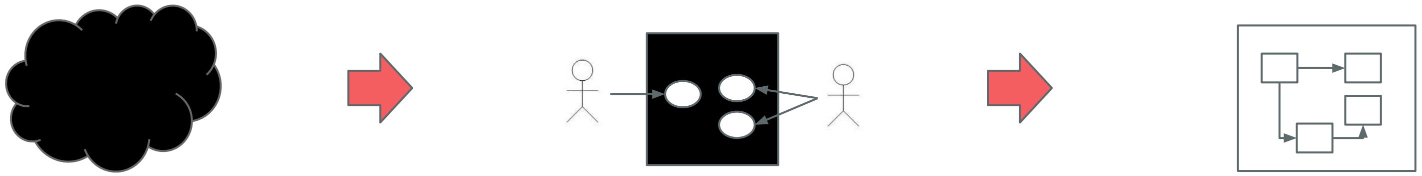
1) I find it helpful to think about the system we are trying to define a “fuzzy black box”

2) Use case analysis is ‘prodding’ at the box to define scope and function

(the notation is system boundary, actor and use-case)

3) Conceptual class identification starts to build internal structure

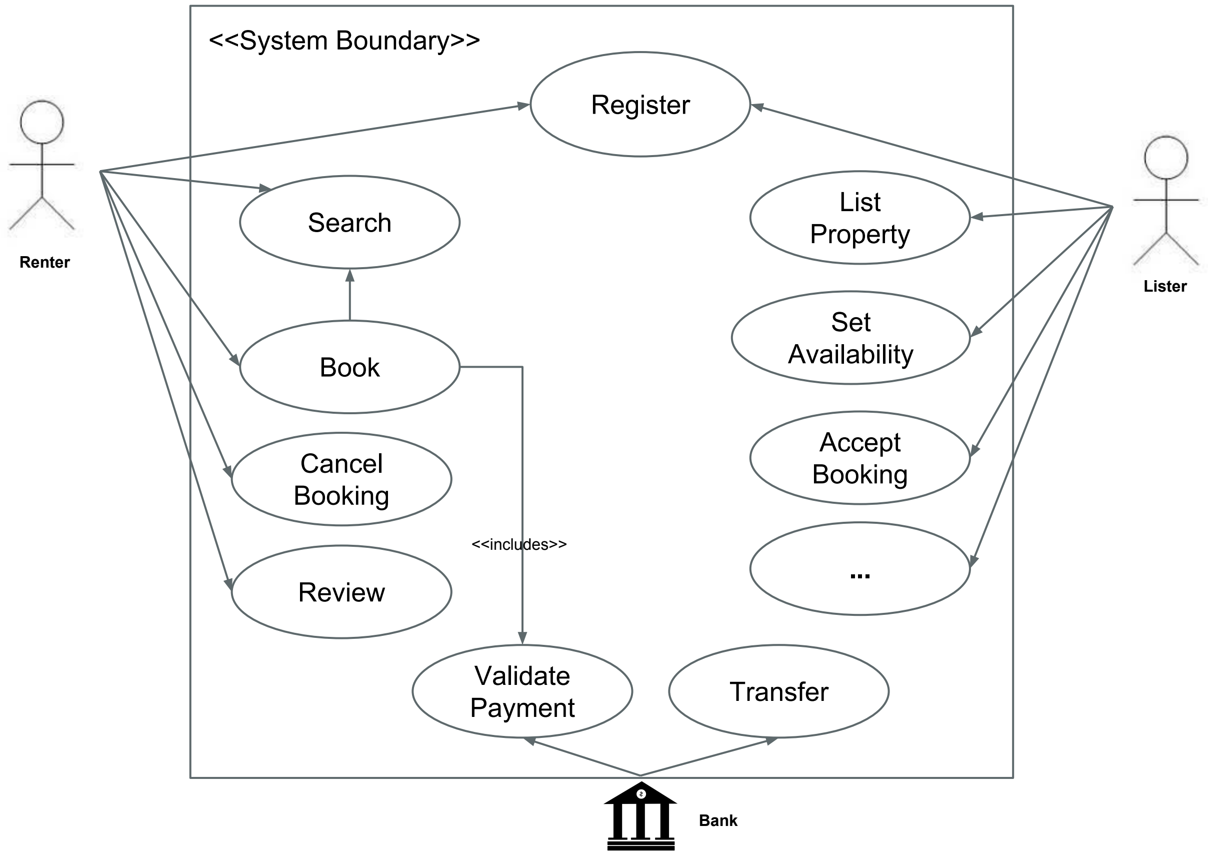
(design often ‘descends’: conceptual»logical»physical)



~~1. Gather Requirements~~

- Book assumes that a functional specification has been produced
- We will replace this with our PRD which serves some of the same purpose

2. Define Functionality



Note: You may see use-cases referred to as the primary requirements technique - https://en.wikipedia.org/wiki/Use_case#Advantages, https://en.wikipedia.org/wiki/Use_case#Misconceptions

Title	List Property
Actor	Lister
Precondition	Lister is registered and in good standing
Basic Flow	1. Select list property 2. Enter property details 3. Select house rules and cancellation policy 4. List Property
Alternate	4b. If property is duplicate, can not be registered or address cannot be validated the listing will be rejected

Title	Search
Actor	Renter
Precondition	Renter is registered (?and in good standing?)
Basic Flow	1. Renter selects search 2. Renter enters search parameters 3. Renter is presented a set of search result 4. Renter can examine details of individual listings
Alternate	-

Title	Cancel
Actor	Renter
Precondition	● Valid booking for this renter ● Rental has not yet begun (**how to cancel a live rental?)
Basic Flow	1. Renter selects booking 2. Renter selects cancel** 3. Booking is cancelled subject to cancellation policy 4. If a refund is due to the renter it is issued[A] 5. Payment to the lister is issued [A](may be zero, partial or complete)
Alternate	1b. Lister is shown ‘not available’ if dates not available or booked
Rules	- Cancellation Policy is set per booking not per property (special events/holidays). - Policy can be changed but will not retroactively impact live bookings

… and many many more

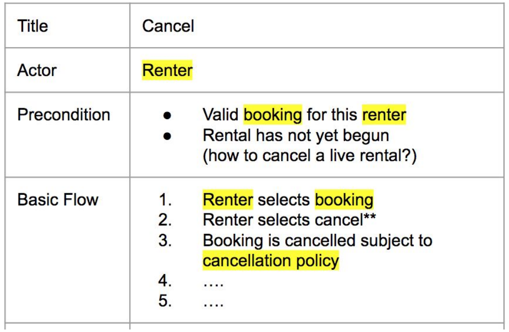
The act of writing use-cases is very likely to expose issues/options

3a. Identify Conceptual Classes

We now need to identify the ***conceptual*** classes of our system

1. Many techniques of which the most common is identifying ***nouns***

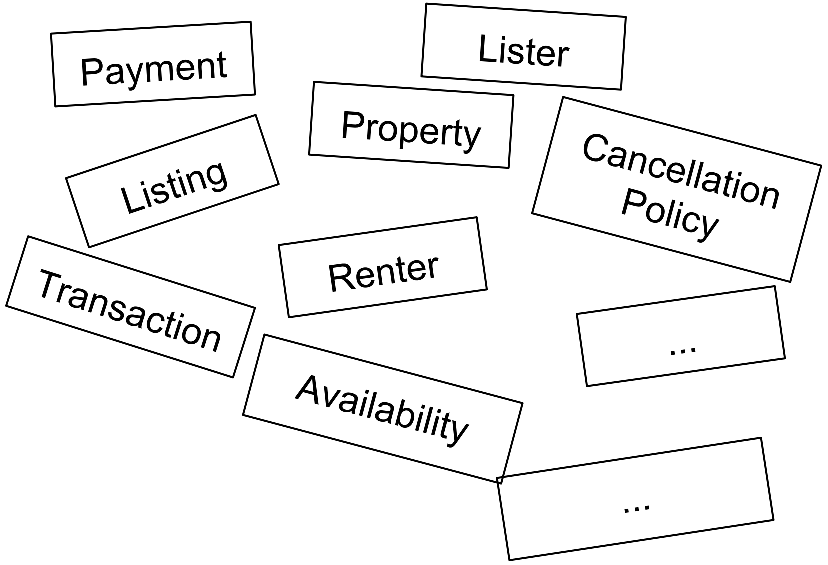
2. Rip through the use-cases highlighting nouns



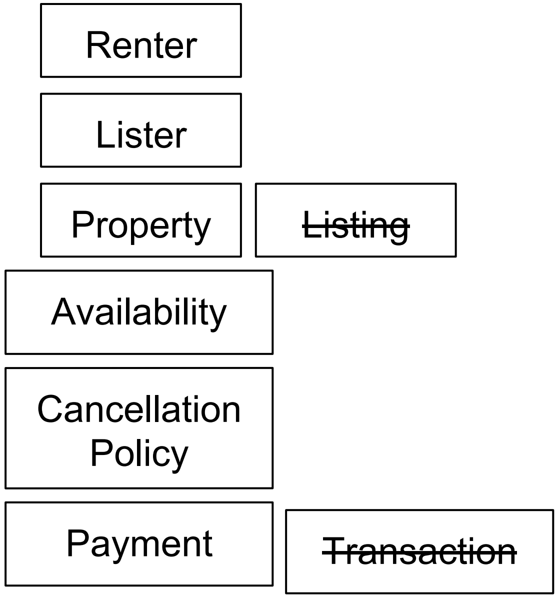
Title	Cancel
Actor	Renter
Precondition	● Valid booking for this renter ● Rental has not yet begun (how to cancel a live rental?)
Basic Flow	1. Renter selects booking 2. Renter selects cancel** 3. Booking is cancelled subject to cancellation policy 4. …. 5. ….

3a. Conceptual Classes

We can rip through requirements (user stories or use-cases) to extract nouns



3a. Distill/De-dupe Conceptual Model

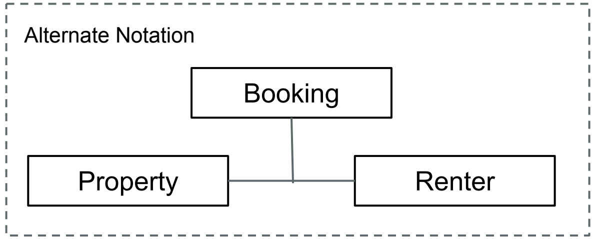


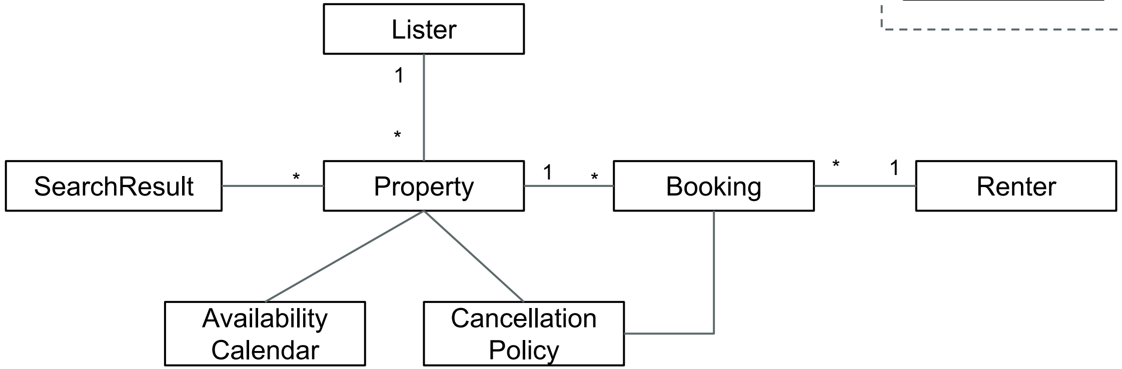
We can then ‘thin’ out duplicates.

We will likely ask ‘is X the same as Y?’ or ‘what do you mean by Z?’

This is fabulous as we are developing common domain language.

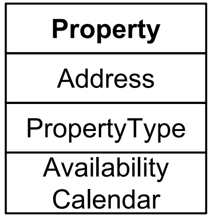
3a. Identify Relationships





Ignoring code for a second. Have we uncovered more of the business rules and have a better understanding of the problem?

3a. Add Key Data Elements



Other Examples

1. An Inventory system for a small shop

2. Software to manage the timing chip for races

Are we done?

Conceptual Model: Closer but no Cigar

“Hey Ewan:
- This is a fine and dandy ‘conceptual model’ of my domain.
- We uncovered significant understanding of our problem domain and system
- But this is not ‘working code’ / does not deal with many concerns”
    - Persistence
    - Data Types
    - Algorithms
    - …

“Yes”
- Output of analysis (conceptual model) is **interim step**
- Need to translate this conceptual model into implementation

Pop Quiz

Question	Answer
OO analysis and design turns __________ into __________	
Analysis yields a _________ <conceptual model / complete code>	
What is the relationship between OOAD and Agile?	
What is the relationship between requirements and OOAD?	

What the books don't say / Industry Perspective

In my industry experience (within my field)

- In the majority of cases I have ***not*** produced ***complete*** class and sequence diagrams *before* coding
- I have ***not*** followed a OOAD process as ***formally*** as we will lay out

How it generally works:

- Teams will perform analysis/design *within* each iteration
  - Whiteboard *a lot* (using semi-formal notations).
    - This builds team consensus of the problem (conceptual level)
  - Discuss *use-cases an awful lot*
- When it comes to development / writing code
  - Low-Moderate complexity: engineers can do OOAD 'on the fly'/in heads/notepads as they write code
- If problem is very hard or significant
  - We will be more formal / go into more detail
  - *Having the formalisms has been a gift in the scenarios where I have been asked to help resolve complex issues*
  - *In these scenarios I will define requirements, enumerate use-cases, show how a design can accommodate all use-cases*
  - Where people were taking at crossed purposes I will be a bit more formal

From a teaching perspective I want you have the tool so you can be flexible:

- Have an excellent command of the theory and process
- You can follow the process as you learn / to learn
- As you master you will be able to handle problems of greater and greater complexity 'on the fly'
- When things get to the complexity you *can not* handle on the fly the process/techniques are there to help

Warning 1

- The Springer OOAD book is well written w.r.t. OOAD concepts.
- ***However***, it sometimes implies *waterfall/predictive* methodology (chapter 6)
  - ***Sequential Plan***[1]: “requirements ↠ analysis ↠ design ↠ construction”
  - ***Document Heavy***: “SRS”
  - ***Silo’ed Roles***: “Analysts ↠ System Analysts ↠ Developers”

- Our response:
  1. Take OOAD concepts from the book
     a. Understand there is no ‘single’ method. Everything is ‘mix and match’
     b. Ignore the predictive/waterfall discussion of process
  2. Incorporate into an agile methodology within the project
     a. OO model will *evolve* over iterations of growth and refactor

[1] Given we have covered methodologies we should be able to identify and link these concepts!

Warning 2

- The Springer OOAD book also implies a classic 2-tier architecture

We will cover architectural styles later:

- We will assume a single process ‘monolith’
- This OOAD works within a small-medium size problem domain. We will discuss other architectural styles in later lectures

Reading

Reading	Optionality
Springer OOAD (chapter 6)	Required