Analysis

Slide Series #3

Analysis

During analysis we try to create a model of the problem domain (i.e. not the software) as a collection of interacting objects

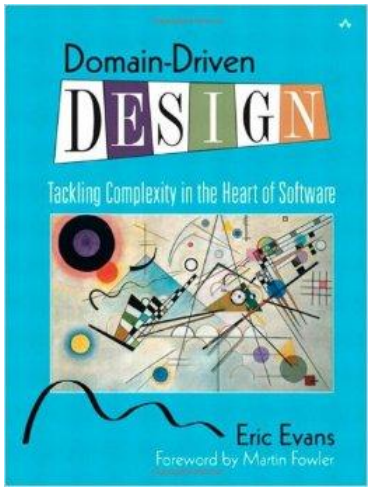
- I.e. an object oriented model
- This is the **domain model**
  - This is the core of our application
- The model is an abstraction of some problem
  - What do we mean to abstract? Are you good at abstracting (this is optional)?

Have to find...

- Objects and how they are related (associations)
- Classes for the objects
- To a lesser degree; attributes, behavior (methods)

Picture: Not much to say but it's a bit fun.

Domain Driven Design

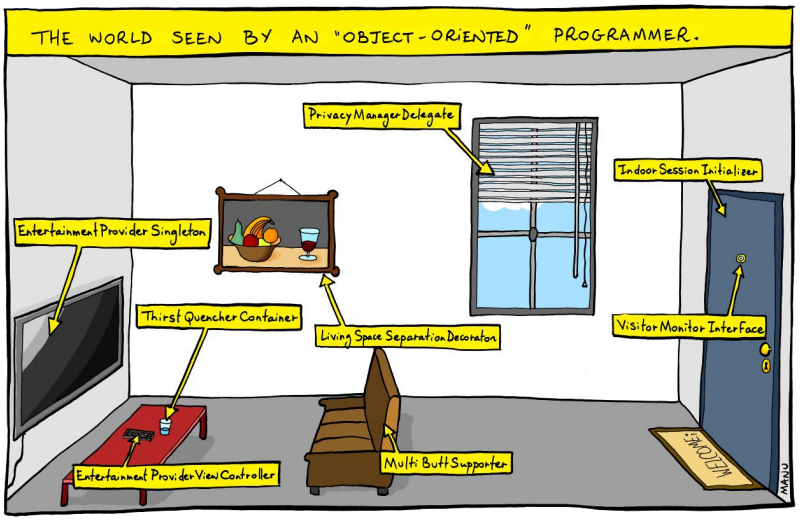


During this phase we adhere to domain driven design

- Using the language of the domain (problem area)
- Placing primary focus on the core domain and domain logic,
  - Objects is not just data, they have behaviour!
  - We use a fat model (vs anemic = just data)
- Solutions to the problem is in a domain model
- Design application on model of the domain.

For short: "Focus on the model" (more later)

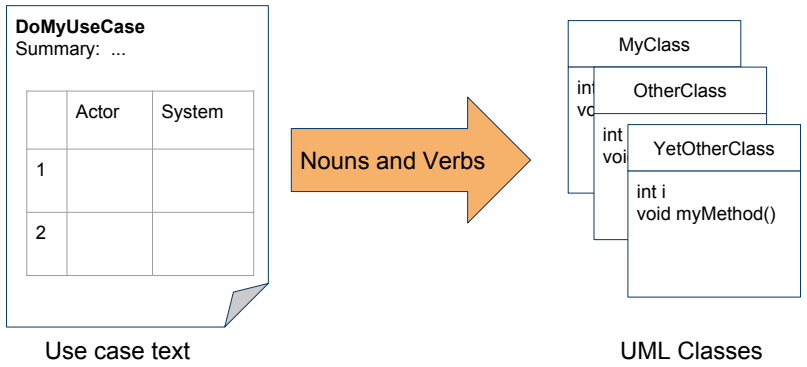
Non Domain Driven Language



This is a baaad

- Should use the language of the problem domain
- … no technobabble!!!

Extracting Classes



Have the use cases from RAD, simple method:

- Underline nouns in use cases, will become classes
- Underline verbs in use cases will become methods
  - Sometimes hard to know which method belongs to which class …
  - … for now put them in any that seems sensible, more later …
  - … or leave out for now, will show up later!
- Include as much as possible.
  - Easy to skip later, …

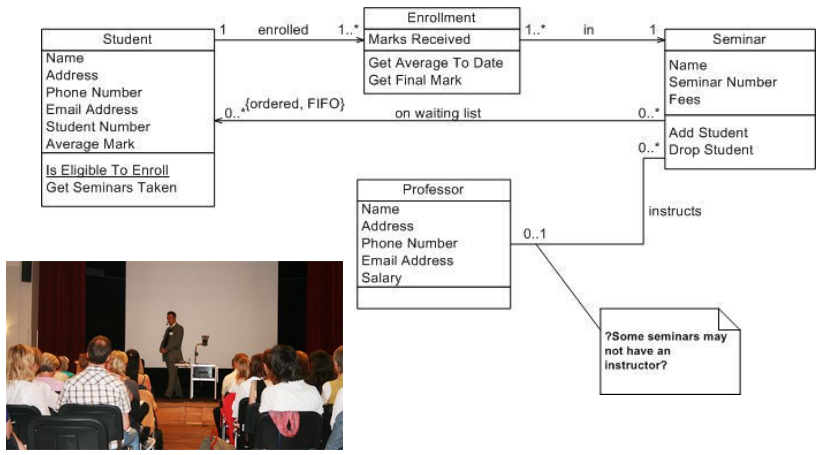
MP : Extracting Classes

- Monopoly
- Dice
- Piece
- Board
- Space (Street, Electricity, etc. …)
- Jail
- Card
- Rent
- Player
- Balance
- Building
- Bank
- Deed (Lagfart)

Not all of this will end up as classes

- Some will be (atomic) values

UML Class Diagram



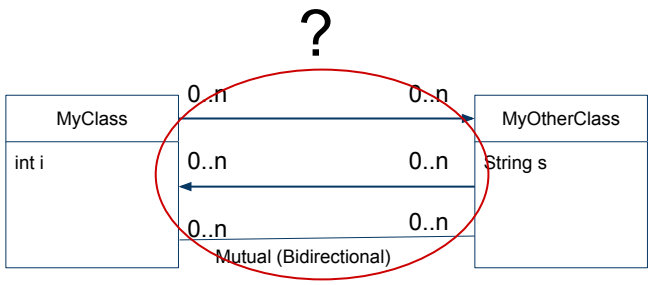
Model represented as an UML class diagram

- Possibly have to break down
  - Also see Package diagrams later
- A static view
- NOTE: Associations and multiplicity is between objects

The diagram has a meaning.

- Symbols, notations etc should end up as runnable code!
- Exercise: Transform diagram to Java!

Associations



Have found many classes from nouns. Possibly some methods from verbs.

- Must find associations to create a class diagram
- We don't distinguish between association, composition and aggregation.

How to find the associations?

- Examine use cases: has, owns, knows, is a, has a, ...
- Visualize the real (physical) situation
- Possibly: Skip directions for now, just note the association

Mutual (bidirectional) associations are bad (or at least we avoid)

- Must keep two object in synch (reference each other)
- Domino effects (change one, affect other)
- Classes not understood in separation

MP : Associations

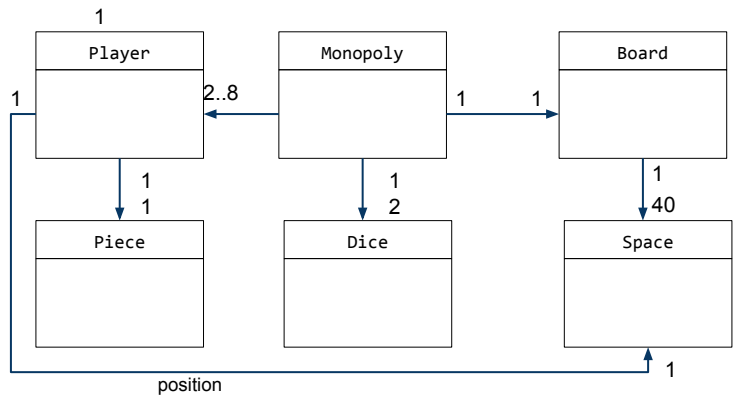
Some considerations

- Player has (shared) dices or …
- … is it Monopoly that has dices?
- Board has spaces or does a space reference the board?
- Player has position (a Space) or a space has visitors (Players)?
- Player owns spaces or space has an owner?

Which directions seems most natural/useful?

- What questions do we need to answer?

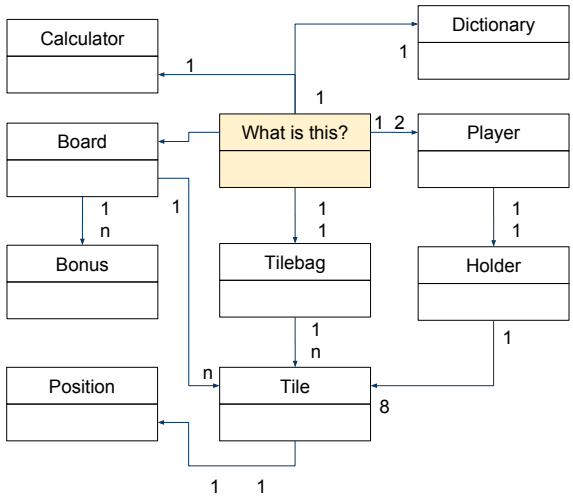
MP : Domain Model



The first domain model (iteration 1)

- Targeting the highest priority use cases: Move and End Turn
- I.e. here are the (minimal number of) classes we need to be able to run the use cases (hopefully?)
- Remainder: This is a model of the domain NOT the software

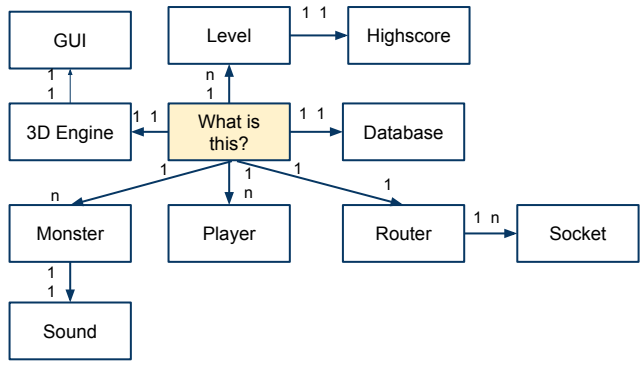
Other Domain Model



What is this?

- The model should be able to communicate something!

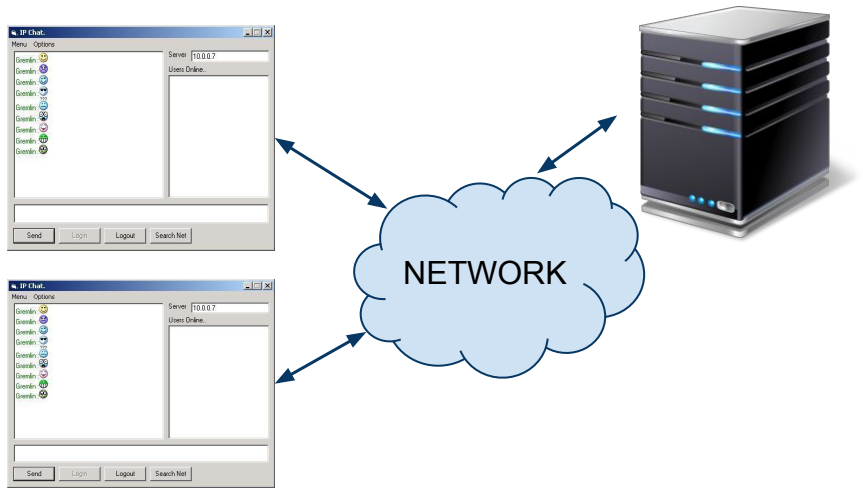
Yet Another Domain Model



What is this?

- This is NOT a model of some problem domain …
- … it's a mess of technical details and domain concepts

Chat Application Model



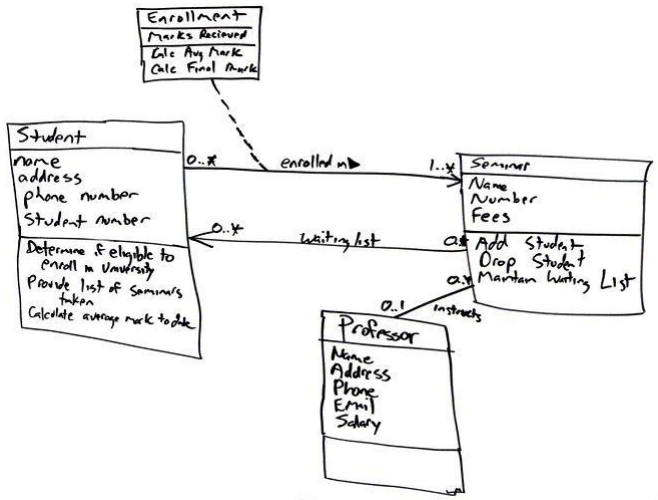
Create domain model for this!

- Have no use case texts, “try on the fly”...

Don’t include technical solution in domain model.

- No network in model
- If doing a client/server application, create one domain model (later to be put on client or server side).

Efficient modelling



Optimal is to first draw on whiteboard!

- Very fast drawing
- Very fast communication, everyone can participate
- Use phone/camera to document

Later, Tools to draw UML

- When model getting more stable
- UMLet plugin to Eclipse, fastest possible
- Linux : Dia
- Mac/Win? ...

Important Object Characteristics

- Unique identity?
- Equality?
- Persistence?
  - Will any objects survive the execution of the program?
- Lifecycle.
  - When is object created?
  - How long does it exist?
  - When destroyed?
- ... other ...

Some valuable characteristics to note for the model objects (if applicable)

- Use stereotypes to annotate (example <<Persistent>> for surviving objects)

MP Characteristics

- Player names must be unique!
- Space names must be unique!
- No objects will survive ... (for now)
- All objects (model) created at start of game, exists until end.
- More .. ?!?

Finishing RAD

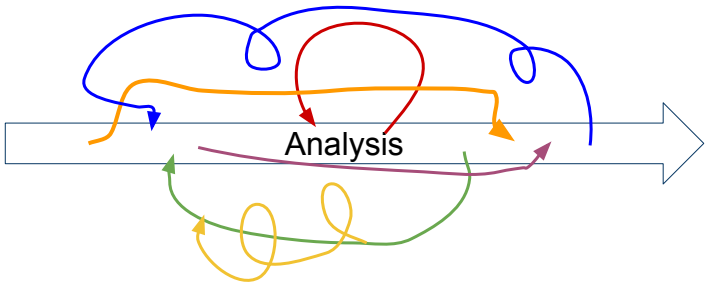
2.3.3 Domain Model ✓

APPENDIX
Domain model UML diagrams ✓

From previous phase we have recorded requirement elicitation in the RAD

- Analysis (i.e domain model) is also documented in RAD!

Analysis: Real world version



Same as for RE!

- It's not linear!

Summary

Analysis focus on building a domain model

- We used the requirements from RAD (use cases) to extract the domain model
- We expressed the model as an UML-class diagram
- We documented model in RAD

Next: From domain model to first implementation