# Requirement Elicitation

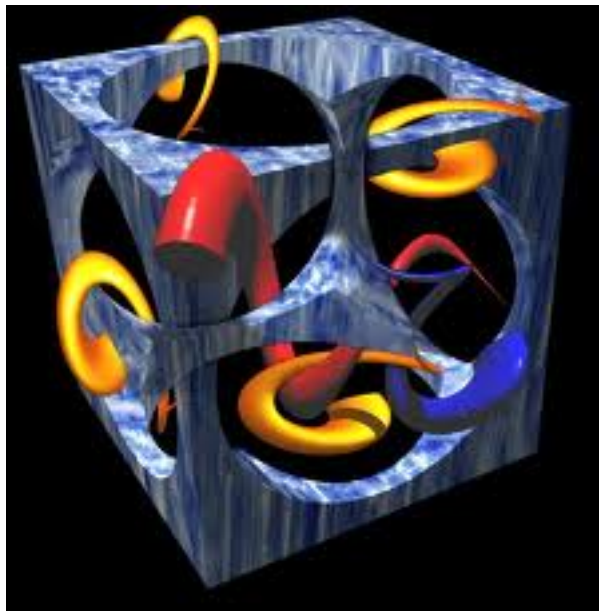
Phase 1

# Hmmm...


I'LL GO UP AND FIND OUT WHAT THEY NEED AND THE REST OF YOU START CODING!
SKELTON

# Problem Domain

- The **problem domain** is the area of expertise that needs to be examined to solve the problem
  - Often, as computer engineers, we don't have expertise in that area!!! How do you write HR system for hotels?
  - Must explore/learn/understand...



What's the inside of this?

# Requirement elicitation

- Requirement elicitation is the exploration/learning phase
- Requirement elicitation aims to get an understanding of the problem domain
- To get a **common vision** of what to build!

# Requirement Elicitation Topics

- Purpose
  - Why are we doing this? What are we trying to achieve? Who will use it?
- General characteristics of application
  - What kind of application is this? In what environment will the system be used?
- Scope of the system
  - What should be **in** and what should **not**
- Objectives and success criteria
  - When are we finished?
- What can we do with the system?
  - **Functional requirements:** Set of inputs, the behavior, and outputs
- Other criteria to judge the operation of a system
  - Non-functional requirements (GUI kind of?)

# The RAD

- The Requirements Elicitation and Analysis document (RAD) will be used to document the requirement elicitation (and later analysis)
  - Template on course page
- Document should to a large extent be understandable for non-computer professionals
  - Could form a basis for the contract
  - Can't be changed without negotiations with customer (in course: your supervisor)
    - Customer can (and often do) initiates changes...

NOTE: This is a programming course **DON'T** spend **too** much time on this (ask supervisor if problematic). Also see examples on course page

# Introducing: "The Monopoly Project"

*(thank's to group "Omen", Axel, Philip, Joanna and Kajsa)*

As a running "case" we'll implement a prototype of the board game Monopoly by Parker Bros (a few iterations)

- It's an application instance, there are other styles...
- Abbreviation: **MP** (on following slides)

# Problem Domain MP

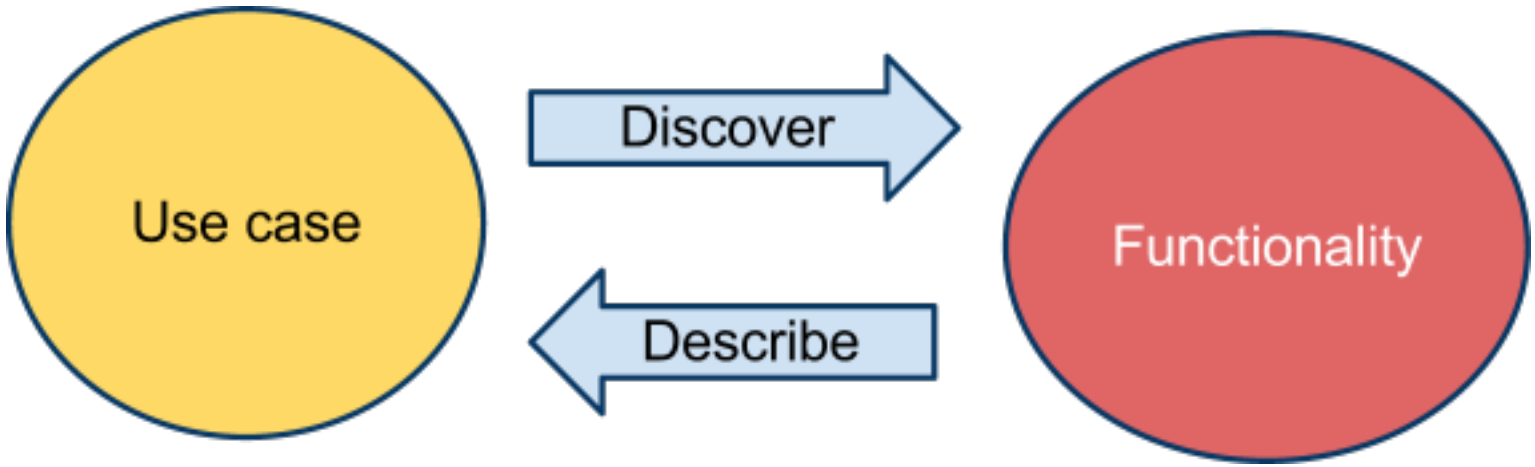
- Problem domain known (?) but note...
  - ...there are quite a few rules!
  - ...there are different sets of rules!
  - ...there are possible unspecified situations!
  - ...there are possible hidden rules (hard or impossible in physical word but possible with computers)?
- **Have to find them all....!!!**

# Requirements for MP

- We'll inspect beginning of the RAD for MP (on course page)
  - Purpose
  - General characteristics of application
  - Scope of the system
  - Objectives and success criteria

# Finding Functional Requirements

- To capture functionality we create **use cases**
- Known or apparent functionality described as **use cases**
- Start at either side



# Use Cases

- A **use case** describes a sequence of actions that provide a measurable value to an actor (user or possible another system)
- A use case is a **text document**
  - Often two columns, one for user, one for system
  - Template on course page
- Use case names begin with a strong verb
- Name use cases using domain terminology (DoMove...not Action ...)
- Use case text uses domain terminology (board, dice,...not array,  random ...)

# Finding Use Cases

- Techniques
  - Interviews
  - Questionnaires
  - User observation
  - Literature study
  - Workshops
  - Brain storming
  - Screen mock-ups

# UC Example for MP

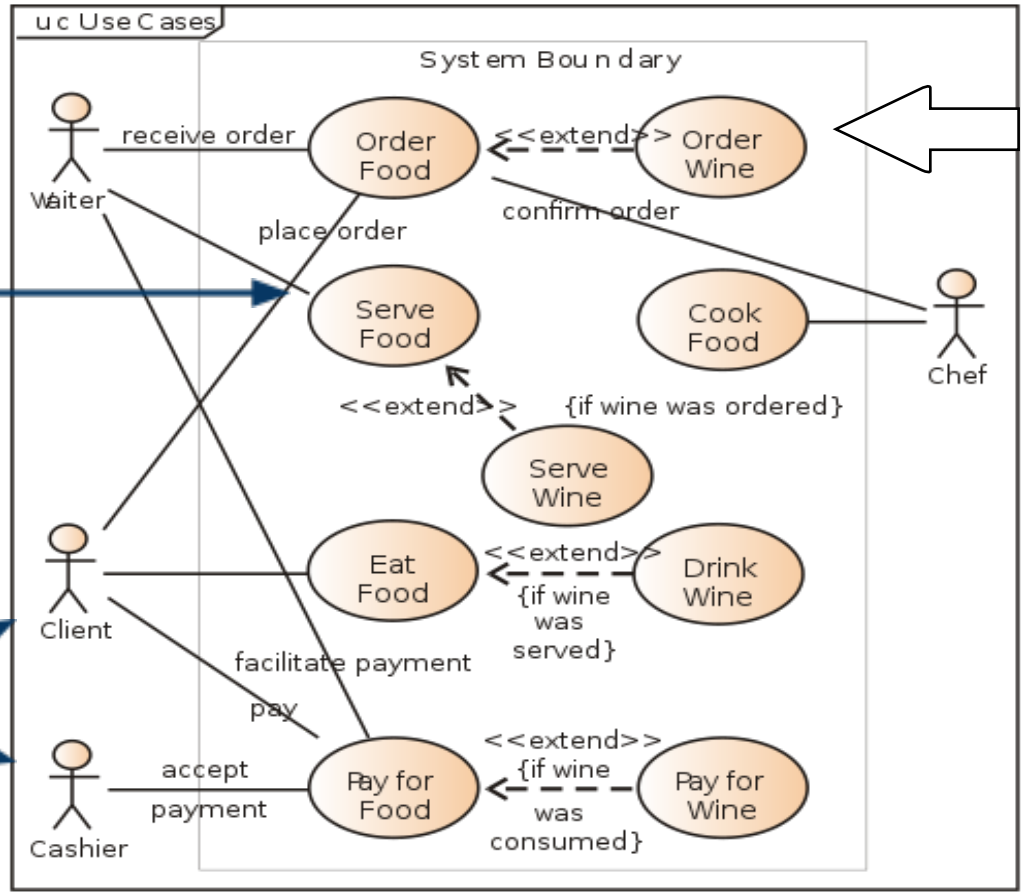
- The UC: Move (on course page)

# Use Cases Fine Prints

- Use case granularity
  - Too large, have to break down
  - Too small, trivial, possible part of other use case
- Use case "extends"
  - Inserting additional action sequences into the base use-case sequence
- Use case "includes"
  - An invocation of a use case by another one
- Use case refactoring
  - Must do! Else possibly end up with duplicate code

# UML for Use Cases

Use case diagram, not overly useful but can give an overview

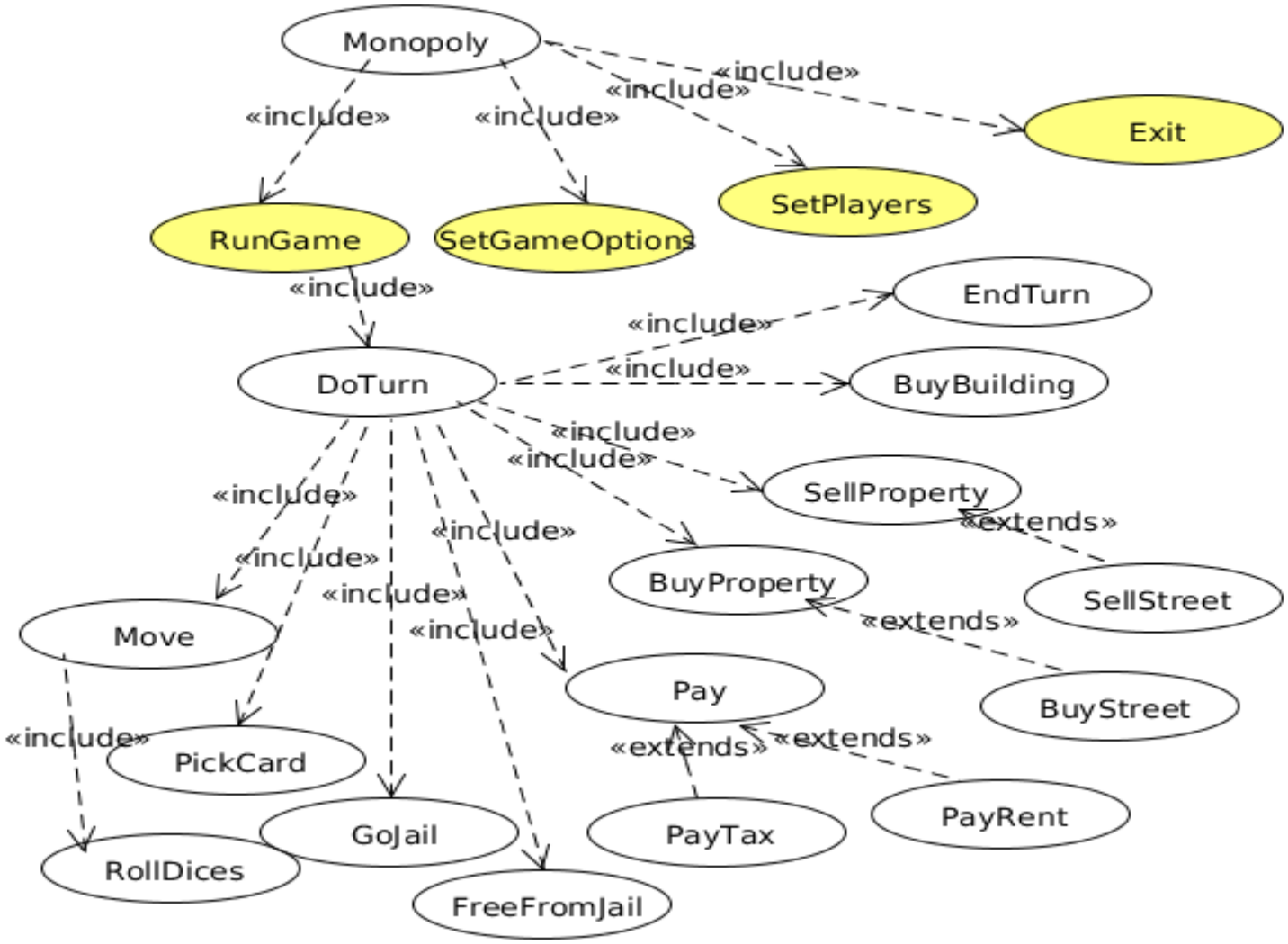



Use case

Actors

There will be a use case text named "OrderWine" to describe in detail

# Use Cases for MP



# Priority of Use Cases

- The use cases should be ordered by priority
  - High, implemented in first iteration
  - Mid, later iterations
  - Low, optional, possible never implemented
- High priority characteristics
  - Significant, central functionality
  - Substantial coverage of the solution, stress or illustrate a specific point of the solution (to be solved)

# UC Priority for MP

- What's your opinion??

# Use Cases Signals Exception Handling

- Flow and exceptions
  - Star with normal and possible alternative flow
  - Add exceptional cases
- Example: User enables auto reply on mail

**Normal flow:**

| User | System |
|---|---|
| 1. Selects auto reply | |
| | 2. System shows a ... |
| 3. User ... | |
| | 4. System... |
| | n. Incoming mail |
| | n+1. Auto replies to sender |

**Alternative (exception):** Can you think of a (funny) exception?

# Scenarios

- Finding the general interaction (use case) with the system is sometimes to complicated
- A scenario is special case of a use case (fixed data)
- Example: Schedule a meeting
  - **Use case:** What is a meeting in general? Location, remote participating, duration, who can participate individuals, groups, roles, rights...who can schedule a meeting .. etc. lot to analyze
  - **Scenario:** Pick an instance. Sven schedules a meeting with Lisa (remote) and all sales representatives at ... thu 10-11...

# Non-Functional requirements

- Usability
  - The ease of use and learnability of a human-made object (in our case GUI)
- Reliability
- Performance
- Supportability
  - Testability (yes, implicitly mandatory in course)
- Implementation
- Packaging and installation
- Legal

(See Wikipedia for long list, ...but don't use it...!)

# User Interface

- During requirement elicitation we also sketch a preliminary GUI
  - Initially envision the system.
  - Enables you to explore the problem space with your stakeholders
  - Enables you to explore the solution space of your system.
  - A vehicle to communicate the possible UI design(s) of your system
  - A potential foundation from which to continue developing the system

# Non-Functional for MP

- We'll inspect the RAD again

# GUI for MP

- Should look like a Monopoly game
  - Flat 2d look for now
- Possible to select different location (London,...Ullared)
  - Must be possible to change texts,
  - Internationalization,...must use internal representation (keys) for text.
- Possible small screen
  - Will use popup for details, dialogs for messages

# Hmmm...


Cause of Errors
Requirements
Code
Effort devoted to tasks
Code
Require-
ments

# Summary

- Requirement elicitation focus on
  - Understanding the **problem domain**
  - Finding functional and non-functional requirements
  - To create a shared vision of the project
- Documented in RAD

Next: From requirements to the domain model,
i.e. the analysis phase