Requirement Elicitation

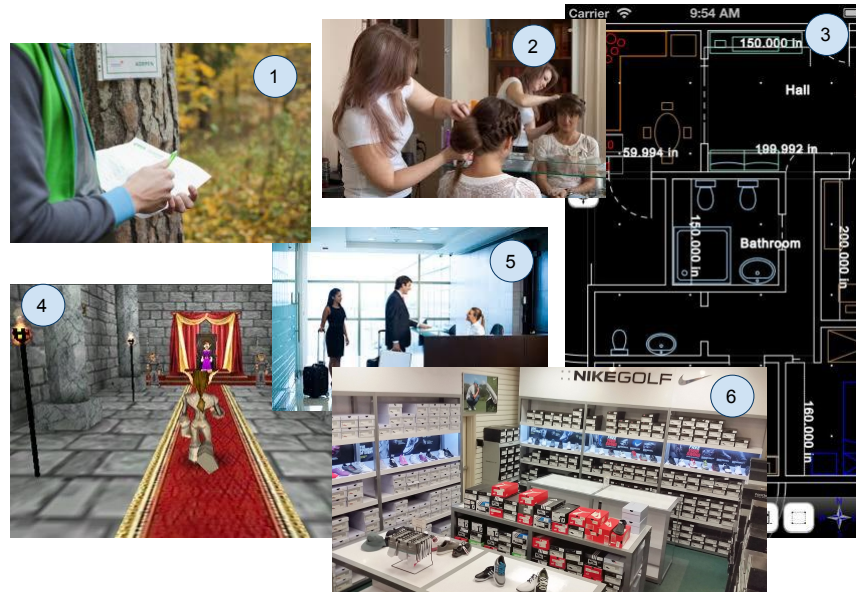Slide Series #2

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!
-   Must explore/learn/understand … (and consult domain experts, … but (probably) not in this course …)
    -   Of course many levels of understanding …
    -   … we need to abstract out the aspects that are relevant for the problem the application is aimed to solve.

Picture: Sometimes the domain is very hard to understand?!?!!

Domain Language

What is "the language" of the domain!
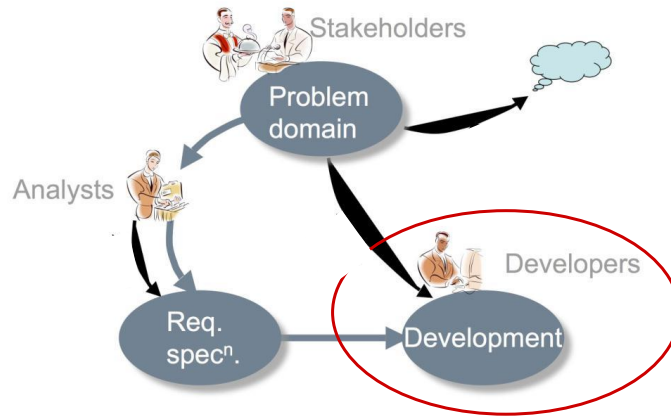- Notes, positions, timeslots, distance, player, guest, …?

Requirements

Software Requirements is a field within software engineering that deals with establishing the needs of stakeholders that are to be solved by software …
- … in some problem domain
- Very hard area …

Requirement Elicitation

[Requirement elicitation](#) (RE) is the process to gather the requirements
- During RE we aim to get an understanding of the problem domain ...
- ...and to get a common vision of what to build!
- If not
    - ... you'll end up building multiple (different) applications inside one application
    - ... or just the wrong application
    - ... or just fail

RE Techniques

Comparison of Data-Gathering Techniques[1]

Technique	Good for	Kind of data	Plus	Minus
Questionnaires	Answering specific questions	Quantitative and qualitative data	Can reach many people with low resource	The design is crucial. Response rate may be low. Responses may not be what you want
Interviews	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee. Encourages contact between developers and users	Time consuming. Artificial environment may intimidate interviewee
Focus groups and workshops	Collecting multiple viewpoints	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters
Naturalistic observation	Understanding context of user activity	Qualitative	Observing actual work gives insight that other techniques cannot give	Very time consuming. Huge amounts of data
Studying documentation	Learning about procedures, regulations, and standards	Quantitative	No time commitment from users required	Day-to-day work will differ from documented procedures

[1] Preece, Rogers, and Sharp "Interaction Design: Beyond human-computer interaction", p214

This is a course, we have to emulate …
- The group must act as stakeholders
- For now, probably mostly informal methods to gather knowledge.

RAD

Requirements and Analysis Document for ...

1. Introduction
1.2 Definitions, acronyms and abbreviations

2. Requirements
2.1 User interface
2.2 Functional requirements
2.3 Non-functional requirements

3. Use cases
3.1 Use case listing

4. Domain model
5. References

Outcome of RE documented in sections 1-3 of the Requirements Analysis Document (RAD)
- This documents targets the customer (i.e. non technical/non programmer)
- This is about communication, no absolute rules how to write

Purpose RAD
- Describes the system in terms of models (GUI, Class diagram), functional and nonfunctional requirements and serves as a contractual basis between the customer and the developer. The RAD must be written in the language of the customer's domain of business/expertise. Under no circumstances should any "computerese" terminology creep into this document.
- Audience: The customer, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design).
- **It's ok to update the RAD during the process (you should), don't try to write a final version for iteration 1**

Monopoly Case (MP)

As a running "case" we'll implement a prototype of the board game Monopoly by Parker Bros (with some twist).
- It's an application <u>instance</u>, there are other kind of applications/styles/ways...
    - More to come
- Abbreviation: MP (on slides)

MP: Problem Domain (MP)

Problem domain known (?) but note...
- There are quite a few rules!
- There are different sets of rules!
- There are possibly unspecified situations!
- There are possibly contradicting rules!
- There are possibly hidden/inconsistent/undefined rules
    - Hard or impossible in physical world but very possible with computers?

MP: Iteration One

Start!

Here we start iteration one for monopoly case!
-    Iteration and explanations a bit entangled

MP: Defining

The project aims to create a computer based generic version of the well known board game Monopoly by Parker brothers. Generic in the sense that it's should be possible to adapt the game to different locations and more, *From RAD* er below.
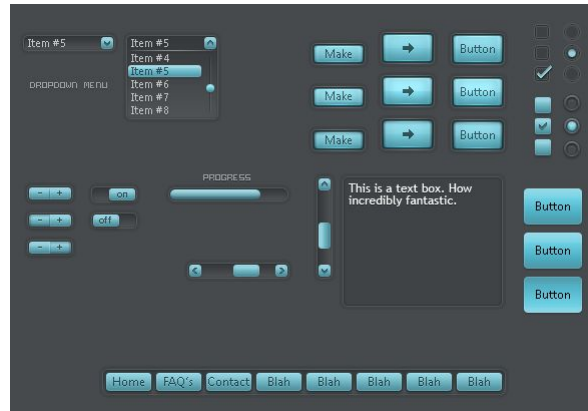
...

Some general characteristics:

- The application will be turn based. The actual player must explicitly end his or her turn. The next player is chosen by the application from a preset ordering. The ordering is generated randomly by the application at start of the round.
- There's no time constraints for a round.
- The application will end according to the rules or possible be canceled.
- If the game is canceled the player with most resources will be the winner.
- The application will handle all of the bank's responsibilities.
- The application will use a GUI very similar to the original game.
- The application does not include a computer-player. It's impossible to play the game alone (a person can of course choose to play against herself).
- The application does not save interrupted games or collect any statistics (high score or other).
- ...

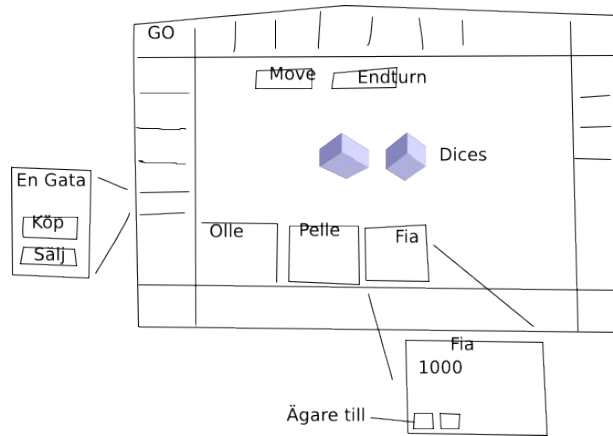Here we try to define the application.

User Interface

To create the use cases we need a preliminary graphical user interface
- GUI will participate in use case like: Customer <u>clicks button</u>, system <u>show dialog</u> ...
- We'll sketch a simple initial GUI

Also GUI sketch let you
- Initially envision the system (important for customers).
- 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 (finding use cases, upcoming ...)

MP : User Interface

Some considerations
- Should look like a traditional Monopoly game
- Flat 2d look for now
- Popups? Switching views?
- Animations later?
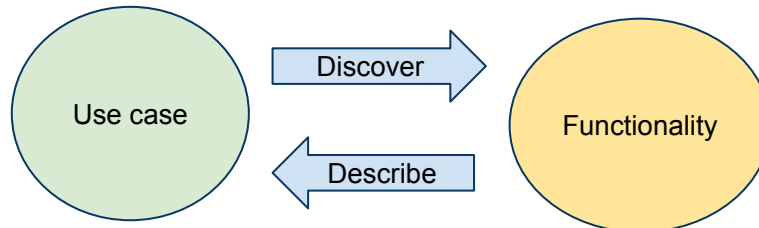- Any twist …?

Functional requirements

The purpose that something is designed or expected to fulfil
  - The range of operations that can be run on a computer or other system

Functional requirements = functionality

Use Case

To find and/or describe functionality we create <u>use cases</u> …
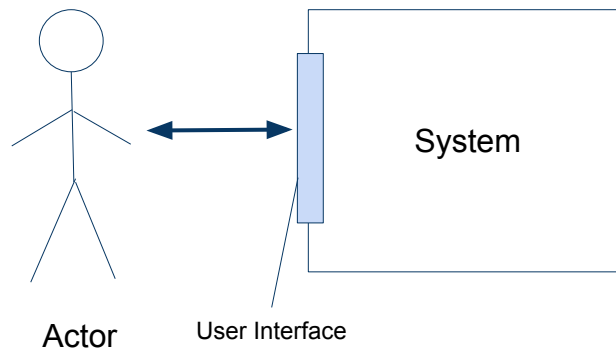- A use cases is a short story telling the interaction between a user and the application/system
    - We use application and system informally and interchangeable
- A use case describes a sequence of actions that provide a measurable value to an actor (user or possibly another system)
- NOTE: <u>The use case does not describe the inner working of the system</u>, it's from outside (the actor's view)
- Use GUI and domain language to describe the flow.

Known or apparent functionality we describe as use cases
- Start at either side

<u>Who</u> "invented" use cases?

Use Case Participants

Participants
- Actor: An actor is a person, organization, or external system that plays a role in one or more interactions with your system.
- The user interface, is the space where interactions between humans and machine occur.
    - We will use a graphical user interface (GUI)
- System: The application.

Record a Use Case

1. Do My Use Case
Summary: ….
Priority: (high, mid, low)
Extends: …
Includes: …
Participators: …

	Actor	System
1	clicks on button	
2		shows dialog
3	clicks ok in confirm dialog	
4		hides dialog

Template on Course Page

A use case is recorded as a <u>text document</u>
- A use case has a name (and better an unique id)
    - Name use cases <u>using domain terminology</u>
        - Correct for MP: player, board, dice, …
        - Wrong for MP: array, randomGenerator, subclass … (technical details)
    - Use case names begin with a <u>strong verb</u>
- Normally two columns, one for user, one for system (incl. GUI)
- Numbered steps for the flow of actions/event (no commonly accepted numbering standard)

The quality of the UC's will have impact later
- Let UC text be as focused (short) as possible but try to be <u>precise </u>(missing facts may affect later stages)
- <u>Corner cases</u>!
- Make it a short play (one person emulating the system, playing really dumb), does it work?

We always start out with **normal flow** (interaction works as simple and normal as possible)
- Then we add **alternate flows**
    - Often have alternative paths in the sequence of actions, alternate flows
        - Depending on outcome of response, or other…
- Then we add **exceptional flow**
    - How will the sequence of actions behave if we get an exception?

The use cases should be <u>ordered by priority</u>
- High, implemented in first iteration
- Mid, later iterations
- Low, optional, possibly 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)

Writing [effective use cases](#)

MP: Use Case Move

1. Move

Summary: The game has started. Actual player is in turn
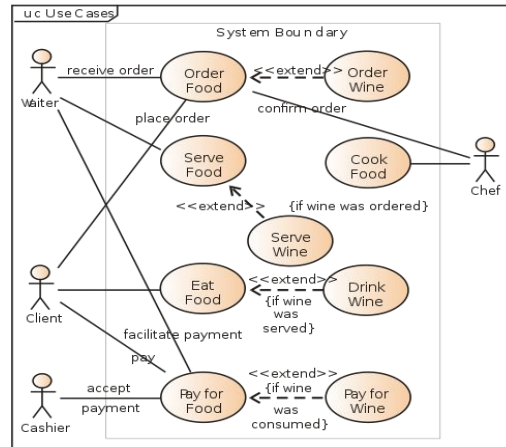Priority: High
Extends: DoTurn
Includes: Roll Dices
Participators: Actual player

	Actor	System
1	Click Roll button	
2		Result for two dices shown Piece removed from actual position and put in new position Roll button disabled
2.1 Passed Go		If player passed go, player balance flashes (updated) and a "cash"-sound is played
2.2 Landed on owned property	See Pay Rent	

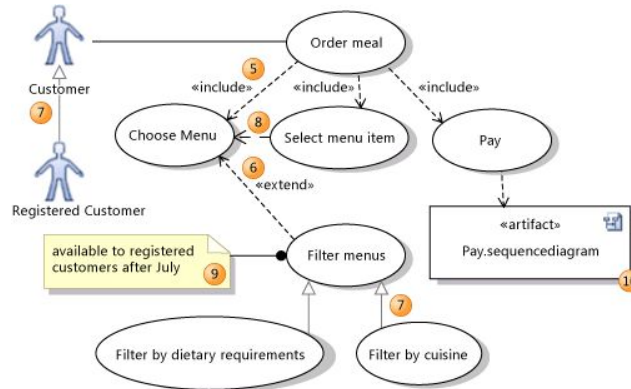MP: Starting out with use case "Move" …
- Focus on normal flow
- Alternate or exceptional flow as multilevel numbering starting from normal flow step number
  - If alternate flow "small" place here, else refer to other UC
  - NOTE Alternate flow 2.1 doesn't need any action from Actor!
- Many more alternate flows missing in slides
  - Same result for both dices?
  - Lands on "Go to Jail"
  - etc….

UML Use Case Diagrams

UML use case diagrams not overly useful, but gives an overview

Include and Extend Use Cases

Use case granularity
- Too large, have to break down
- Too small, trivial, possibly part of another use case

Use case **extends**
- Inserting additional action sequences into the base use-case sequence
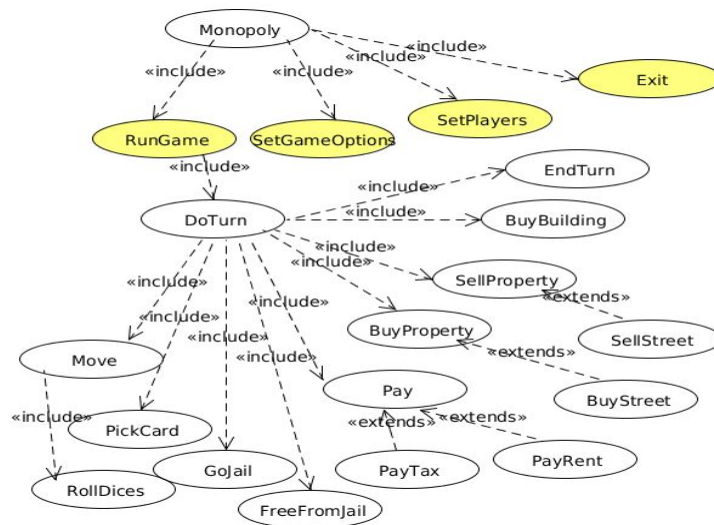
Use case **includes**
- An invocation of a use case by another one

Use case refactoring
- Must do! Else possibly end up with duplicate code

MP: Use Case Diagram

Probably not all found or all needed.
- Anyway gives an overview

MP: Use Case Pay Rent

5. Pay rent
Summary: Player have moved piece and landed on property owned by other player
Priority: High
Extends: Move 2.2
Includes:
Participators: Player

	Actor	System
1		Shows a dialog
2	Clicks Pay Rent button in dialog	
3		Dialog closes Player and owner balances updated (flashes), "cash"-sound
3.1 No cash	See Sell	
3.2 Broke	See Player Broke	

Yet an use case text from Monopoly
- This use case needs interaction from actor
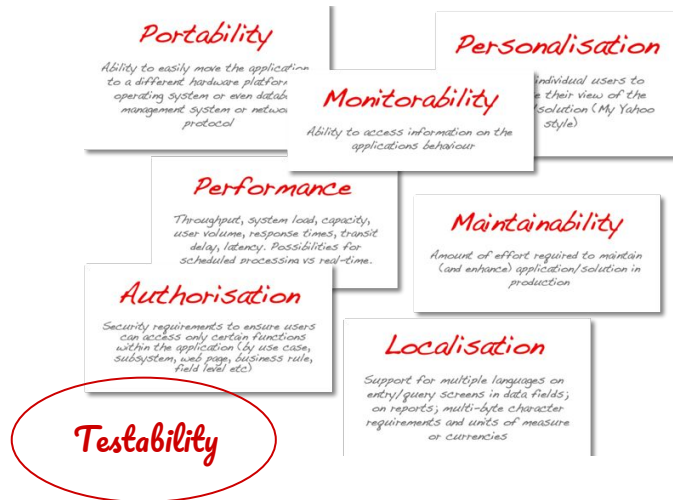- NOTE: There should be a GUI sketch of dialog

MP: High Priority Use Cases?

Which UCs seems very central?

Non-functional Requirements

Portability
Ability to easily move the application to a different hardware platform, operating system or even database management system or network protocol

Personalisation
...Individual users to ...e their view of the ...olution (My Yahoo style)

Monitorability
Ability to access information on the applications behaviour

Performance
Throughput, system load, capacity, user volume, response times, transit delay, latency. Possibilities for scheduled processing vs real-time.

Maintainability
Amount of effort required to maintain (and enhance) application/solution in production

Authorisation
Security requirements to ensure users can access only certain functions within the application (by use case, subsystem, web page, business rule, field level etc)

Localisation
Support for multiple languages on entry/query screens in data fields; on reports; multi-byte character requirements and units of measure or currencies

Testability

Non-functional requirements
- Usability, the ease of use and learnability of a human-made object
- Reliability, probably not applicable (NA) to us
- Performance, probably NA
- Supportability
- Testability (yes, implicitly mandatory in course more to come…)
    - This means: The code we write should be possible to test!
- Implementation (any restrictions? Yes, Java in this course)
- Packaging and installation
- Legal

Some non-functional examples from MP:
- Possible to select different location (Alingsås, Warszawa, Ouagadougou,…)
    - Must be possible to change texts,
    - Internationalization,…must use internal representation (keys) for all text
- Possibly small screen
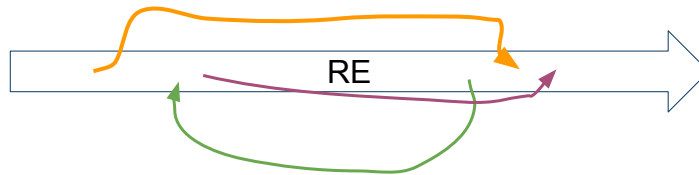   - Will use popup for details, dialogs for messages
- And of course testability …

RAD so far

1. Introduction ✓

2. Requirements
   2.1 GUI ✓
   2.2 Functional Requirements ✓
   2.2 Non-functional Requirements ✓

3. Use Cases ✓
   3.1. Use case listing ✓

   4. Domain model ✗

Sections ticked off in slide should have a first preliminary version.
   - This is of course for the first iteration, more to come …

RE: Real world version

RE

Have done RE in a linear fashion
- In reality RE is more of a parallel iterative process
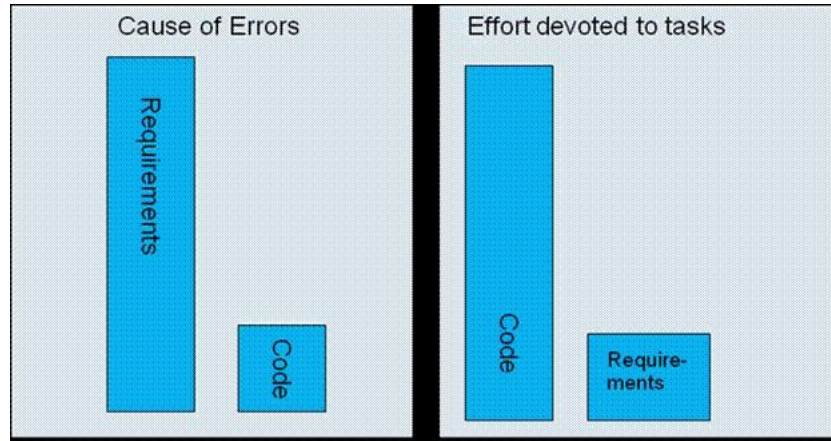- Also: Later stages may affect previous

Prototyping

```java
private JPanel createCardsPanel() {
    int size = board.size();
    cardButtons = new JButton[size][size];
    JPanel pnl = new JPanel();
    pnl.setLayout(new GridLayout(size, size));
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            JButton b = new JButton();
            b.setBackground(cardBack);
            b.addActionListener(this);
            b.setName(row + ":" + col);  // Use this as lookup later,
see actionPerformed
            b.setPreferredSize(new Dimension(WIDTH / size, HEIGHT /
size));
            pnl.add(b);  // Add to panel
            cardButtons[row][col] = b; // Store so we can access later
        }
    }
    return pnl;
}
```

```java
public void initMaterials() {
    wall_mat = new Material(assetManager,
        "Common/MatDefs/Misc/Unshaded.j3md");
    TextureKey key = new
        TextureKey("Textures/Terrain/BrickWall/BrickWall.jpg");
    key.setGenerateMips(true);
    Texture tex = assetManager.loadTexture(key);
    wall_mat.setTexture("ColorMap", tex);
```

During this phase you should start out technical prototyping
- Technical prototyping for now
    - GUI
    - Services (file handling, sound, graphics, Android, etc.…)
    - Hard code, mock anything you need.

Impact of RE

Summary RE

Requirement elicitation focus on
- Understanding the problem domain
- To create a shared vision of the project
- Finding a preliminary GUI
- Finding functional and nonfunctional requirements

RE documented in RAD

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