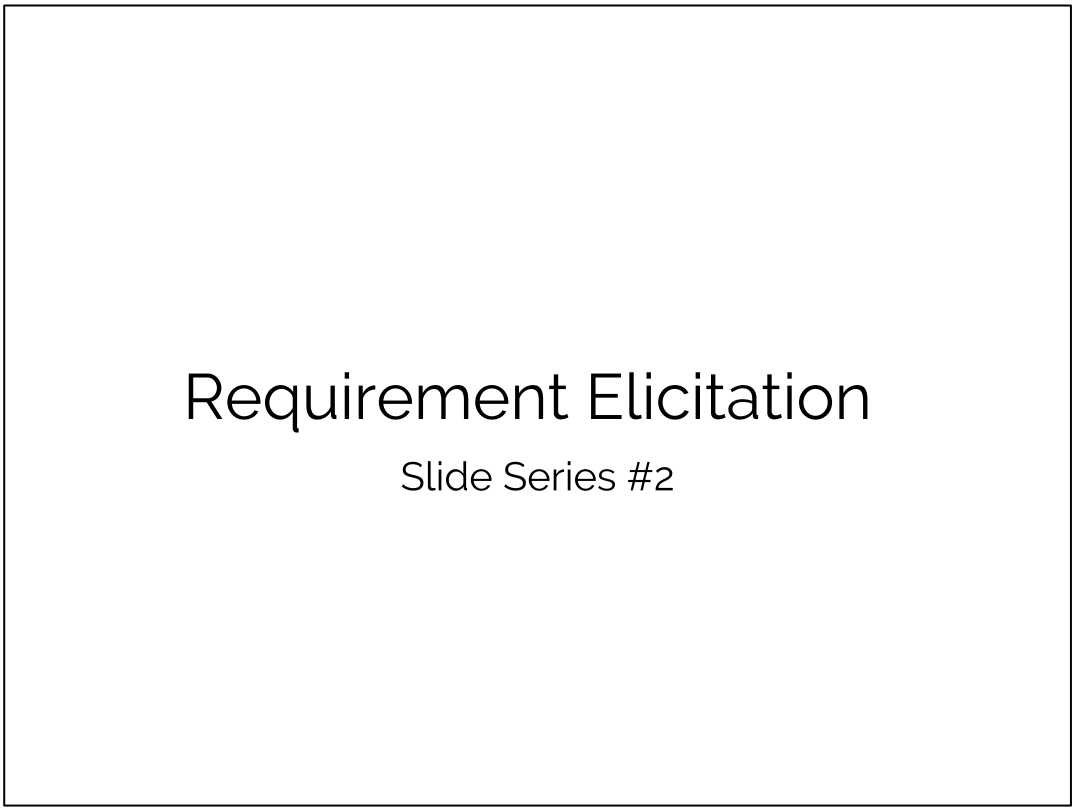
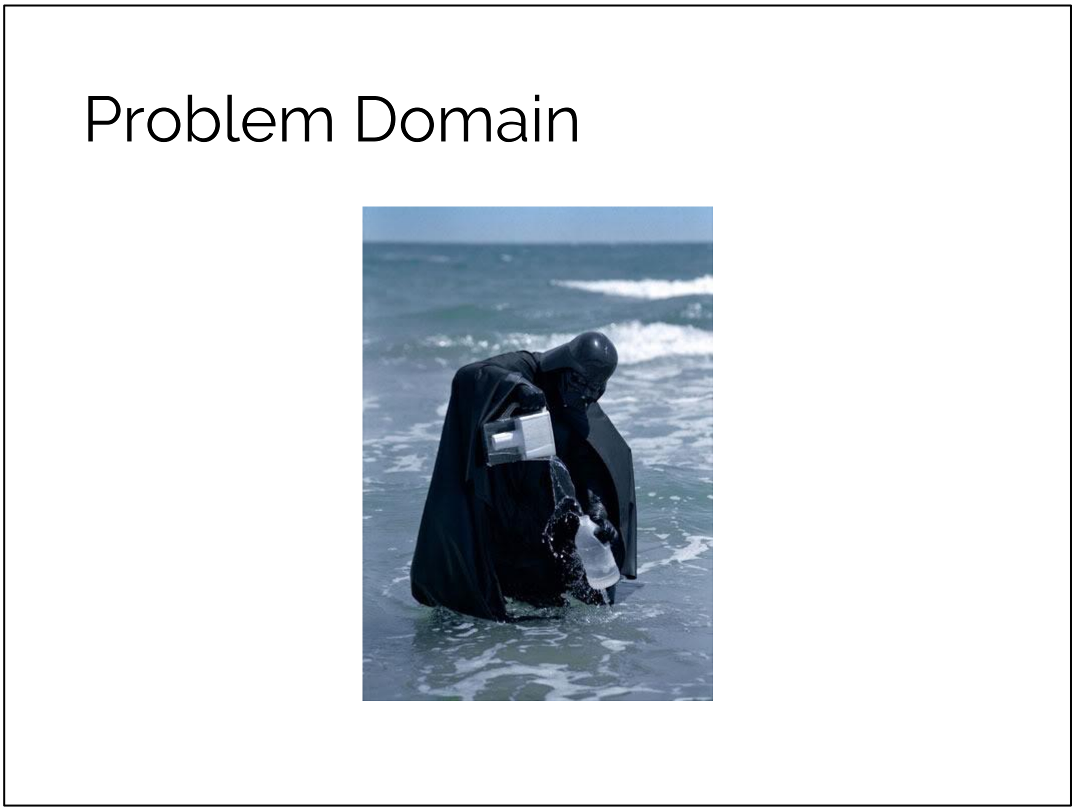


Requirement Elicitation

Slide Series #2

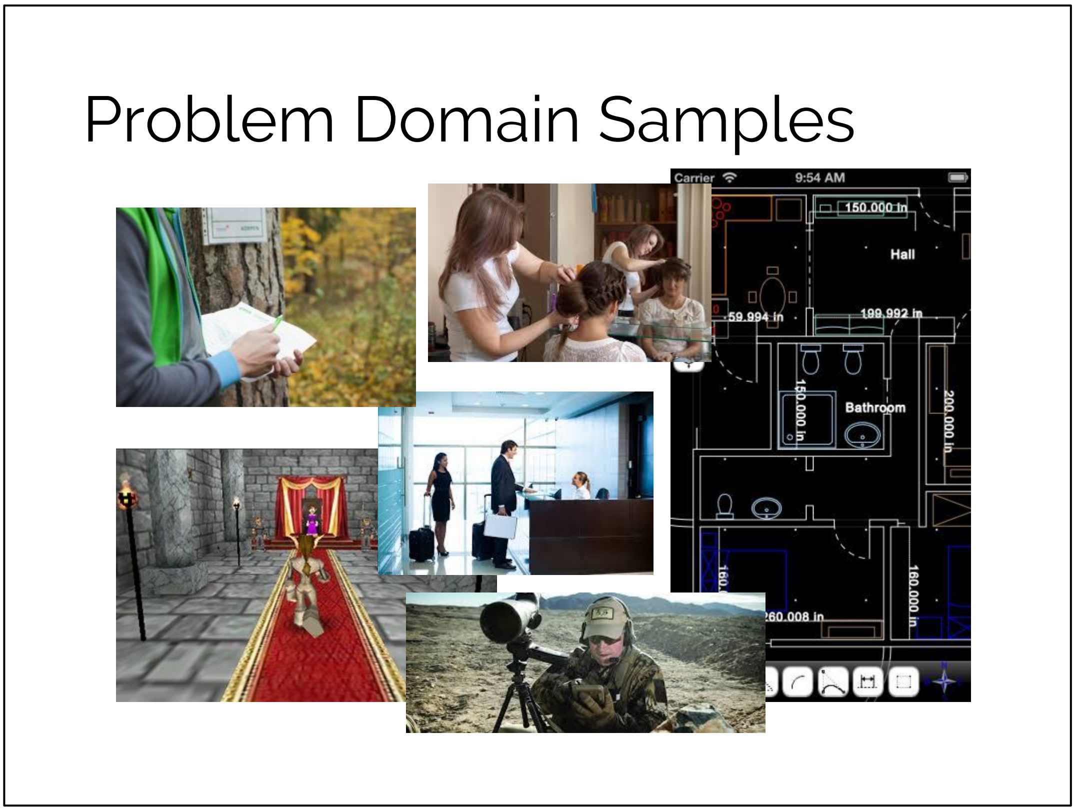


Problem Domain

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.

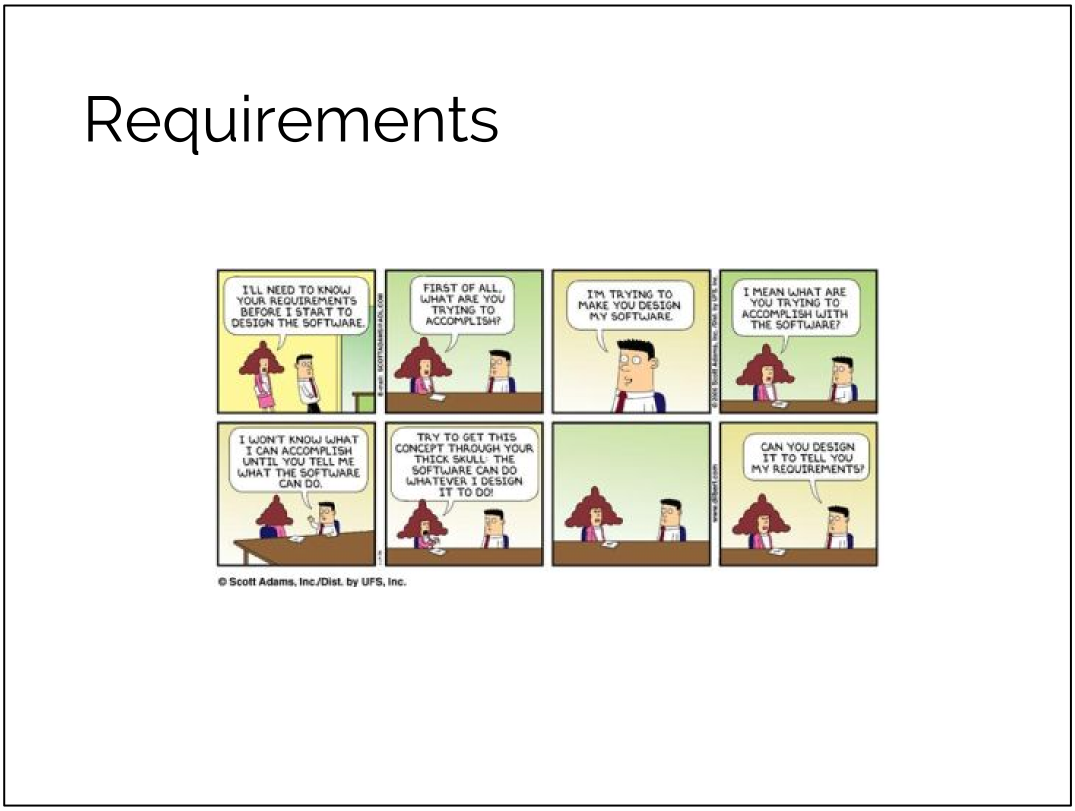
Picture: Sometimes the domain is very hard to understand!



Problem Domain Samples

Which are the domains?

- What do we know about the domains?
- Some typical concepts for each domain?



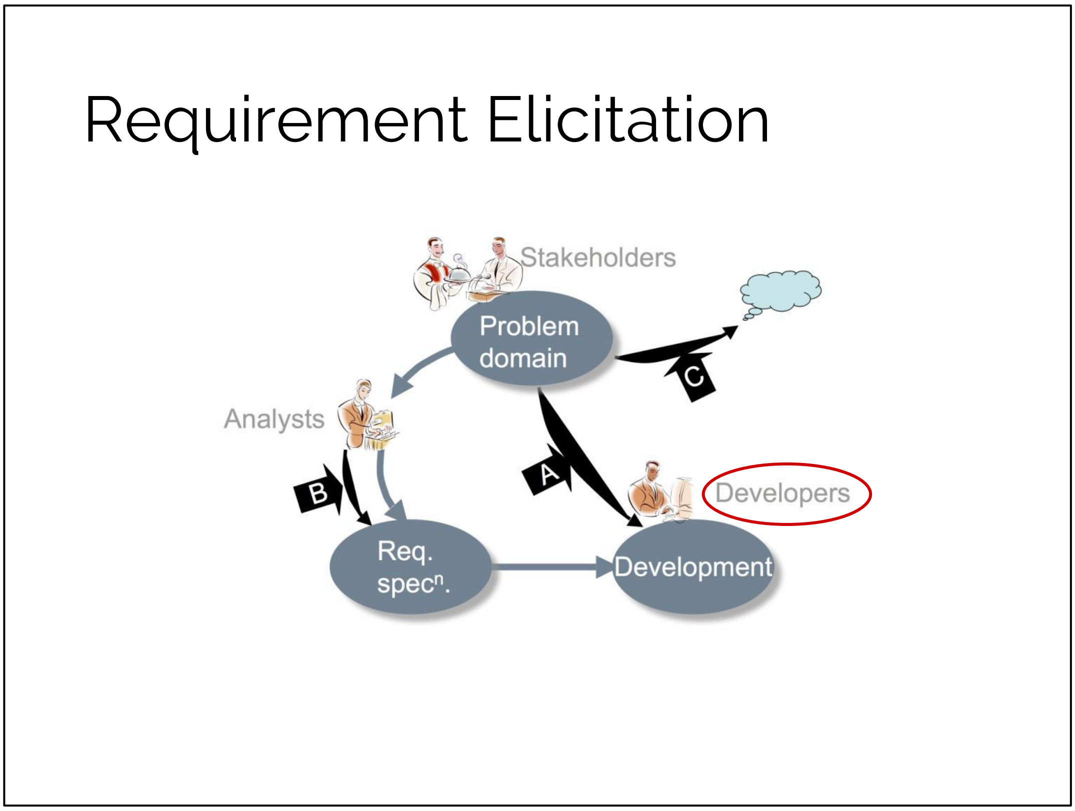
Requirements

I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?

© Scott Adams, Inc./Dist. by UFS, Inc.

Software Requirements is a field within software engineering that deals with establishing the needs of stakeholders that are to be solved by software …

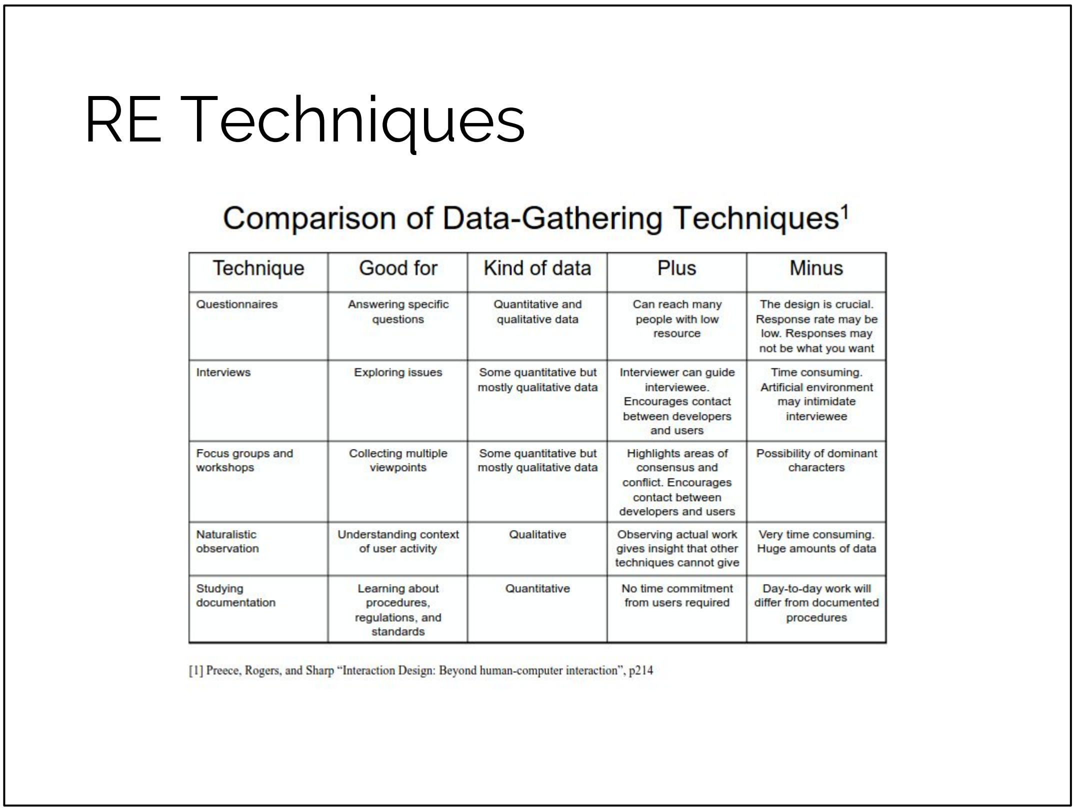
- … in some problem domain
- Very hard area …



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 <u>common vision</u> 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

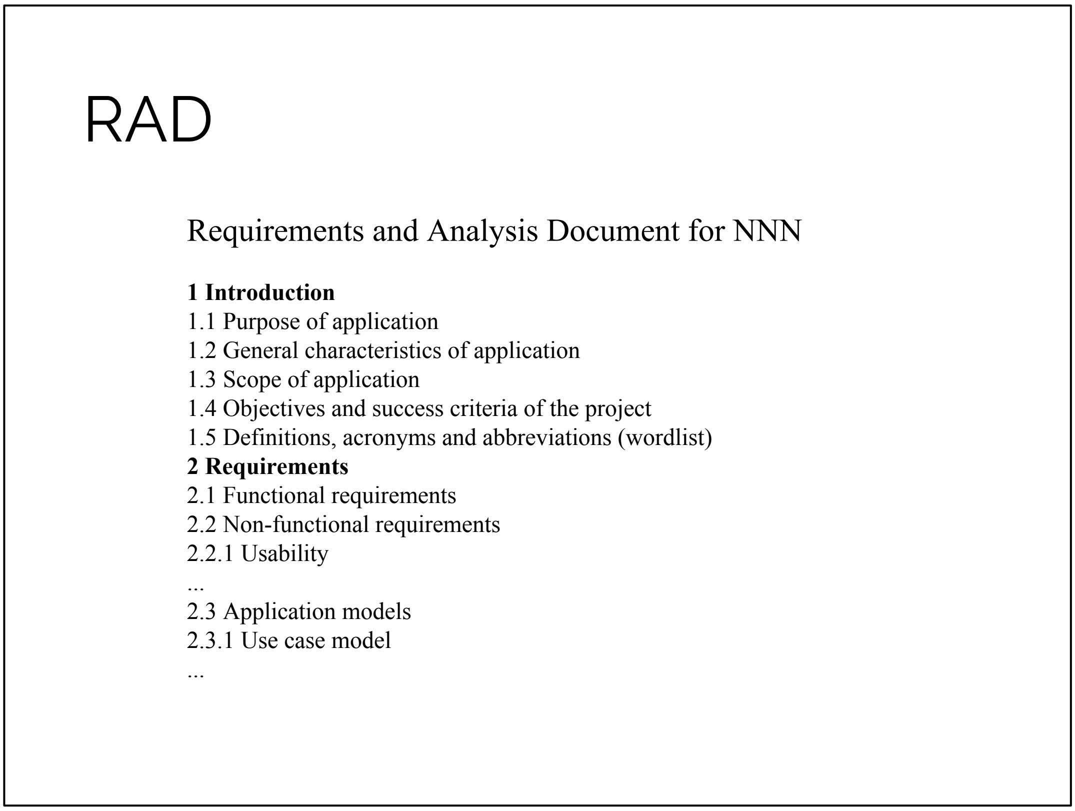
Comparison of Data-Gathering Techniques[1]

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 NNN

1 Introduction
1.1 Purpose of application
1.2 General characteristics of application
1.3 Scope of application
1.4 Objectives and success criteria of the project
1.5 Definitions, acronyms and abbreviations (wordlist)
2 Requirements
2.1 Functional requirements
2.2 Non-functional requirements
2.2.1 Usability
...
2.3 Application models
2.3.1 Use case model
...

Outcome of RE documented in first section of the Requirements Analysis Document (RAD)

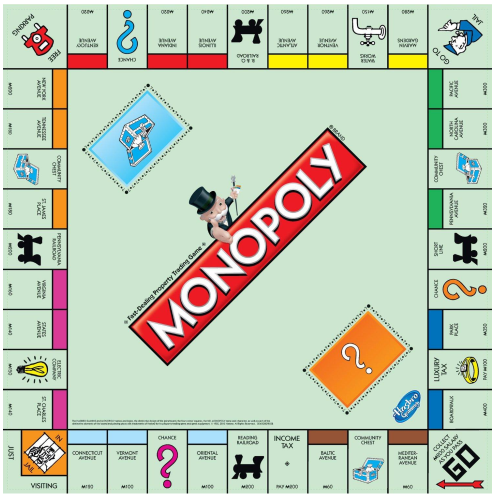
- This documents targets the customer (i.e. non technical/non programmer)
- This is about communication, no absolute rules how to write
- Section one gives a broad definition of application
    - This RAD is course specific … reality is much more complex (a simplified document)

For sections 1.1 and 1.2 see Application Software

Purpose RAD

- Describes the system in terms of models, 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.

- It's an application instance, there are other kind of applications/styles/ways...
    - More to come
- Abbreviation: MP (on slides)

MP: Problem Domain (MP)

The Game of
MONOPOLY

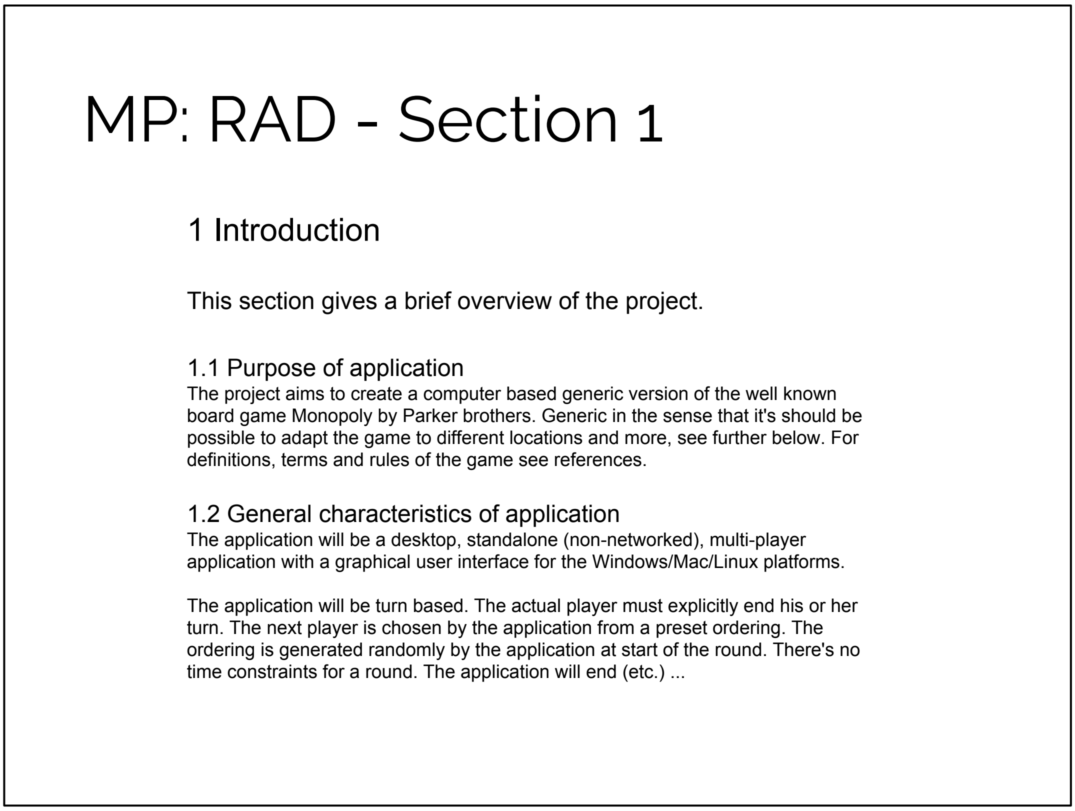
BRIEF IDEA OF THE GAME

RULES

For from Three to Seven Players

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 rules
    - Hard or impossible in physical world but very possible with computers?



MP: RAD - Section 1

1 Introduction

This section gives a brief overview of the project.

1.1 Purpose of application

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, see further below. For definitions, terms and rules of the game see references.

1.2 General characteristics of application

The application will be a desktop, standalone (non-networked), multi-player application with a graphical user interface for the Windows/Mac/Linux platforms.

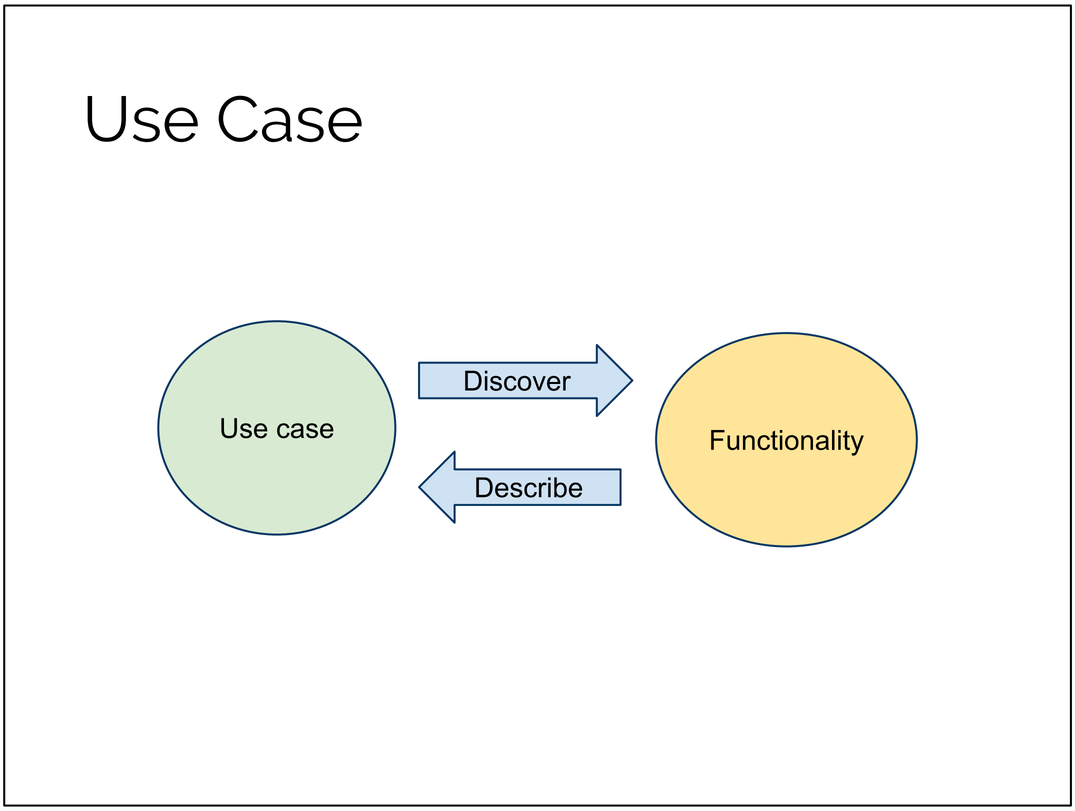
The application will be turn based. The actual player must explicitly end his or her turn. The next player is chosen by the application from a preset ordering. The ordering is generated randomly by the application at start of the round. There's no time constraints for a round. The application will end (etc.) ...

Monopoly RAD see course page!

Functionality

The purpose that something is designed or expected to fulfil

- The range of operations that can be run on a computer or other system



Use Case

Use case

Discover

Describe

Functionality

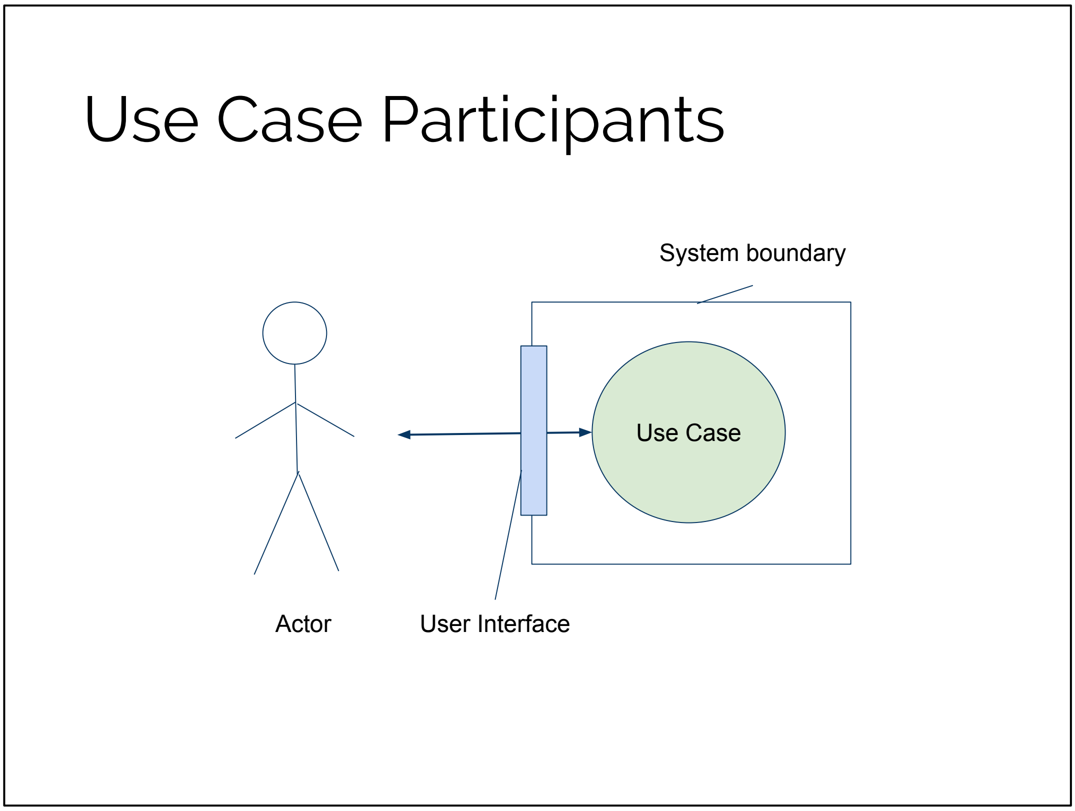
To find and/or describe functionality we create use cases …

- 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: The use case does not describe the inner working of the system, it's from outside (the actors view)

Known or apparent functionality we describe as use cases

- Start at either side

Who invented use cases?

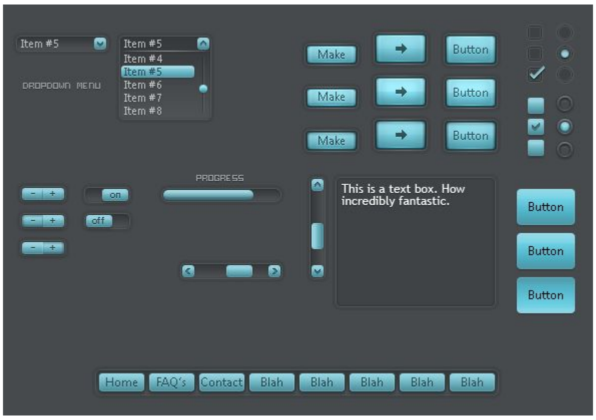


Use Case Participants

Participants

- **Actors**: 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 boundary**: Defines the scope of the system, use cases inside should be implemented by application.

User Interface



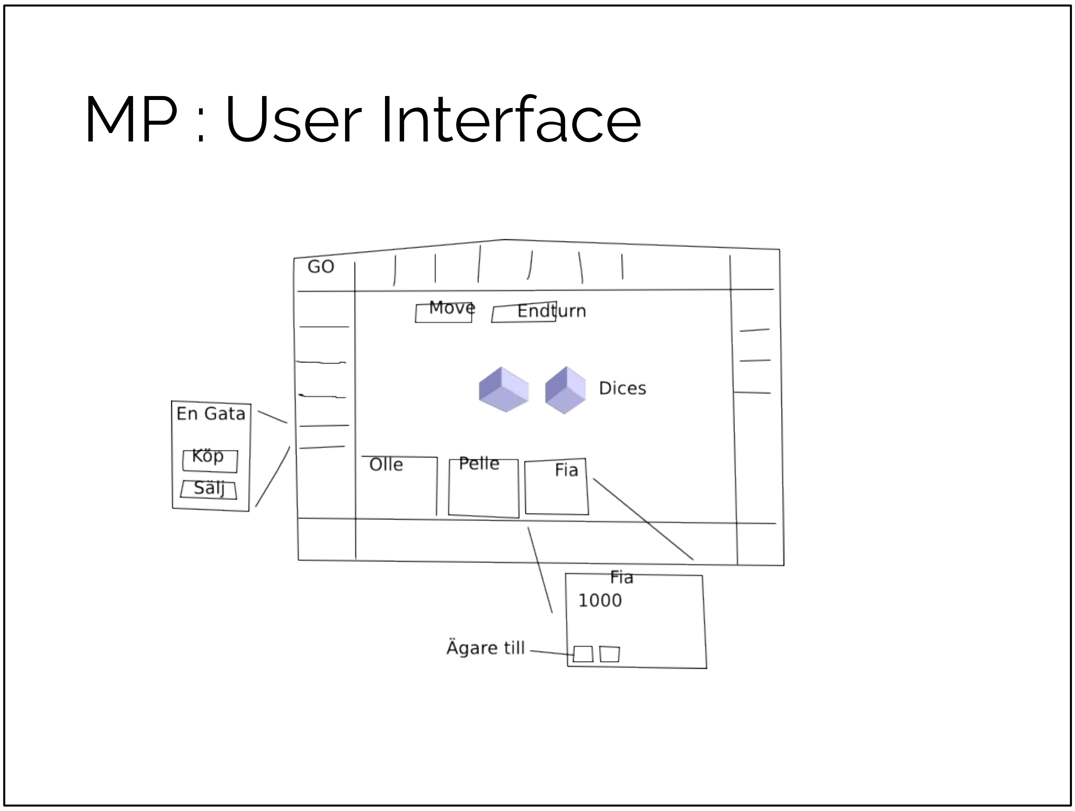
This is a text box. How incredibly fantastic.

To create the use cases we need a preliminary user interface

- GUI will participate in use case like: Customer clicks button, system show dialog ...
- We'll sketch a simple initial GUI

Also GUI sketch let you

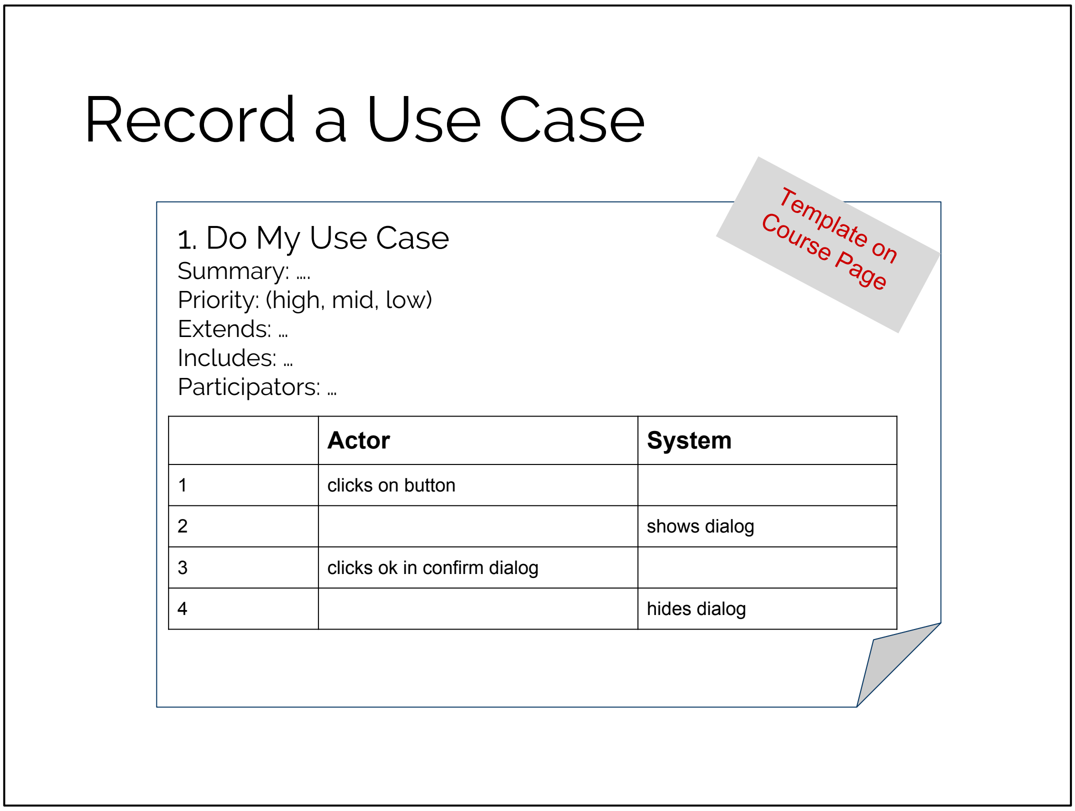
- Initially envision the system (important for customers).
- Enables you to explore the problem space with your stakeholders
- Enables you to explore the solution space of your system.
- A vehicle to communicate the possible UI design(s) of your system
- A potential foundation from which to continue developing the system (finding use cases)



MP : User Interface

Some considerations

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



Record a Use Case

Template on Course Page

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

A use case is recorded as a text document
- A use case has a name (and better an unique id)
  - Name use cases using domain terminology
    - Correct for MP: player, board, dice, ...
    - Wrong for MP: array, randomGenerator, subclass ... (technical details)
  - Use case names begin with a strong verb
- 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 precise (missing facts may affect later stages)
- Corner cases!
- 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?

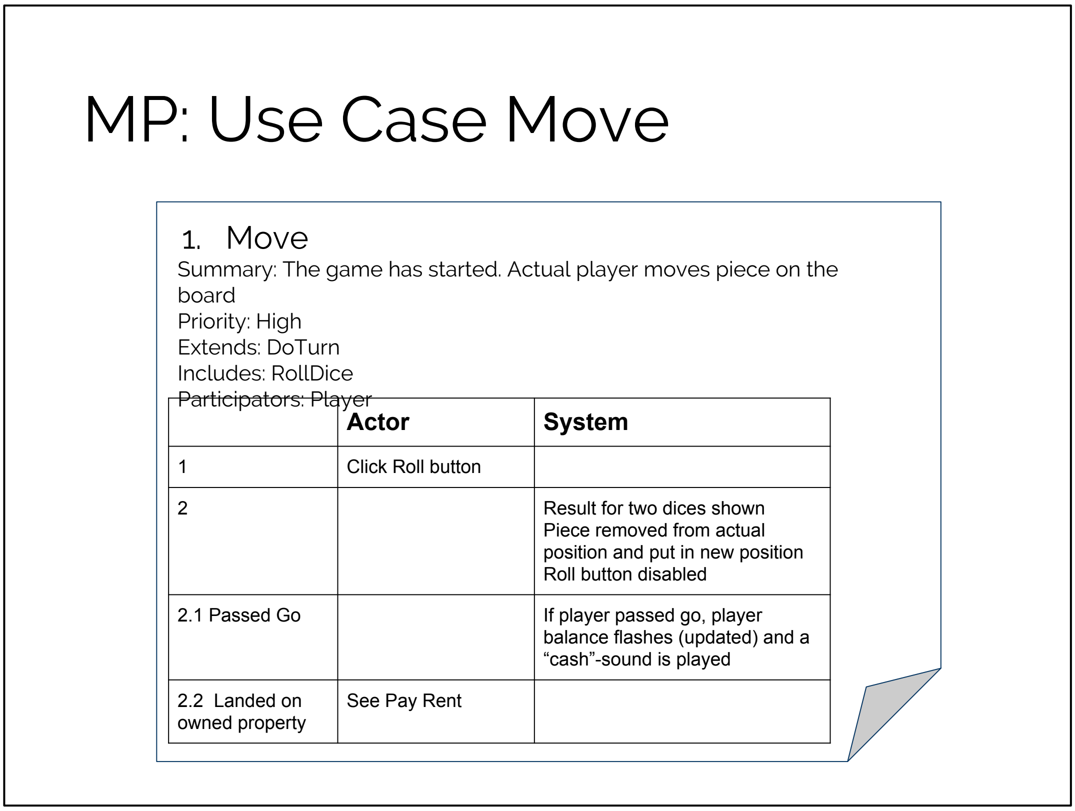
The use cases should be ordered by priority

- 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)

Writing effective use cases



MP: Use Case Move

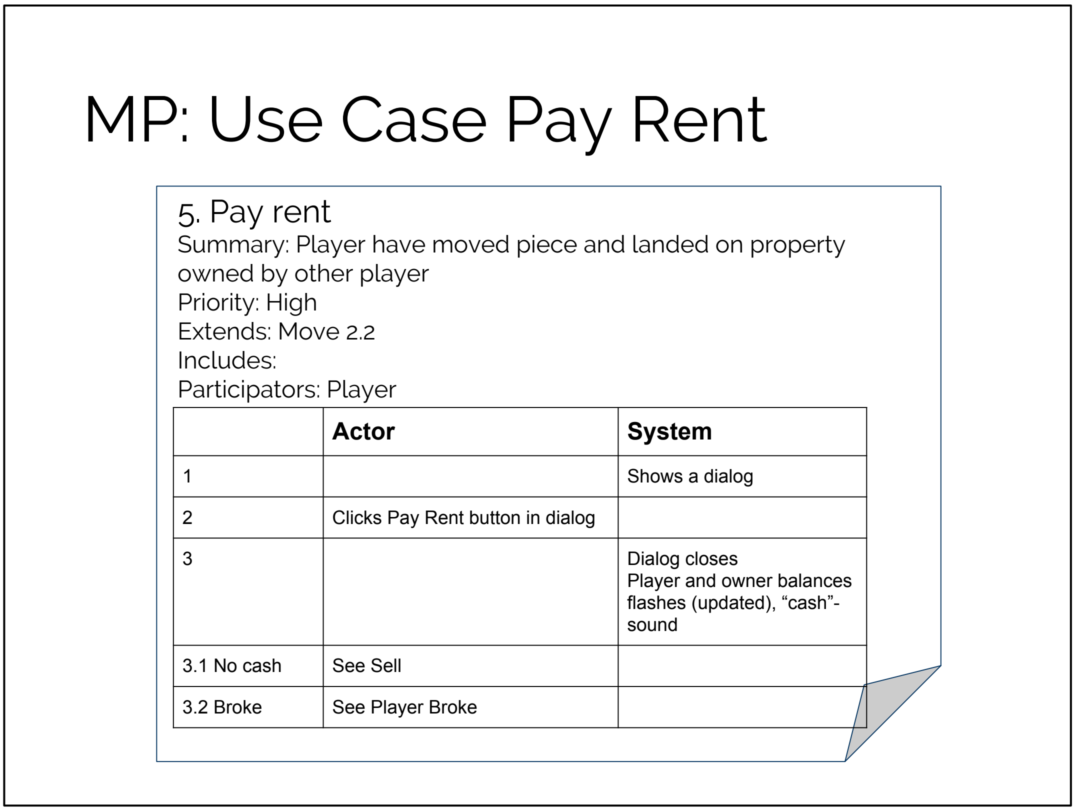
1. Move

Summary: The game has started. Actual player moves piece on the board
Priority: High
Extends: DoTurn
Includes: RollDice
Participators: 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 slide
    - Same on both dices
    - Lands on “Go to Jail”
    - ….



MP: Use Case Pay Rent

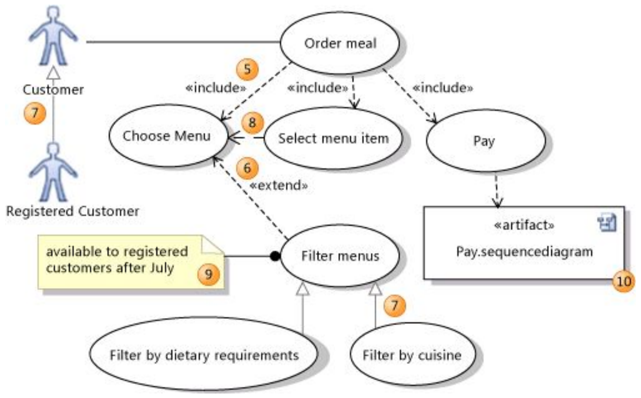
5. Pay rent
Summary: Player have moved piece and landed on property owned by other player
Priority: High
Extends: Move 2.2
Includes:
Participators: Player

	Actor	**System**
1		Shows a dialog
2	Clicks Pay Rent button in dialog	
3		Dialog closes Player and owner balances flashes (updated), “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

Include and Extend Use Cases

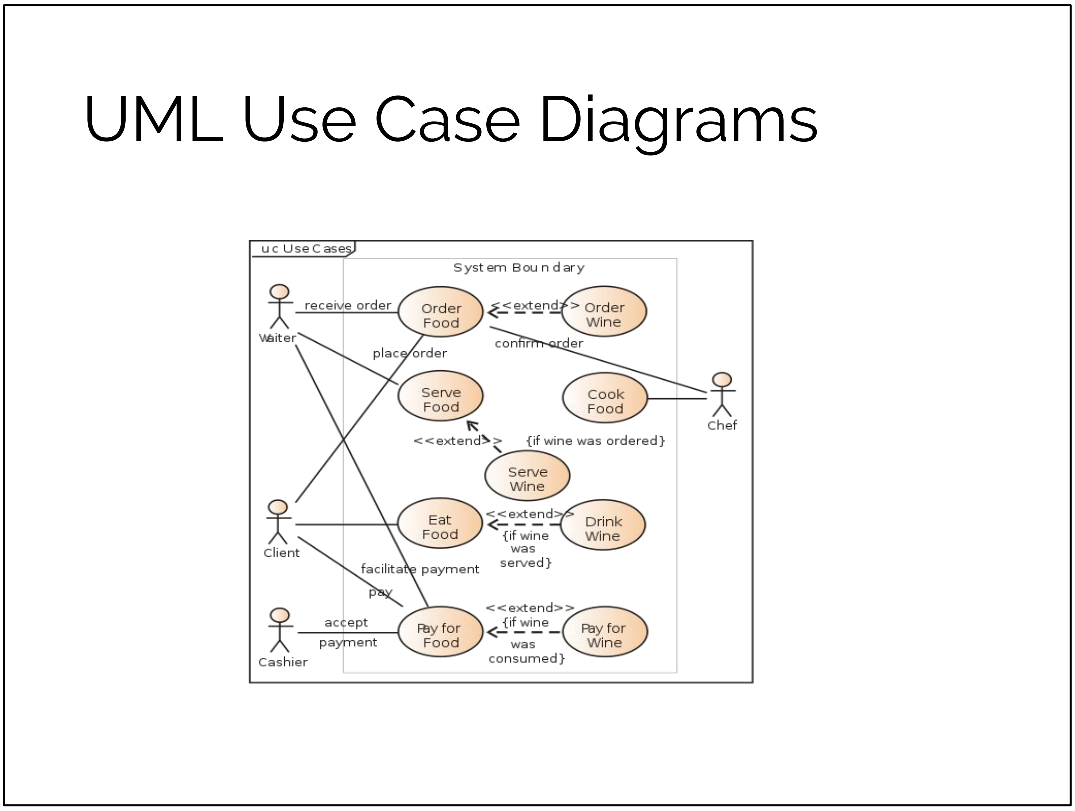


Use case granularity
- Too large, have to break down
- Too small, trivial, possibly part of another use case

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]()

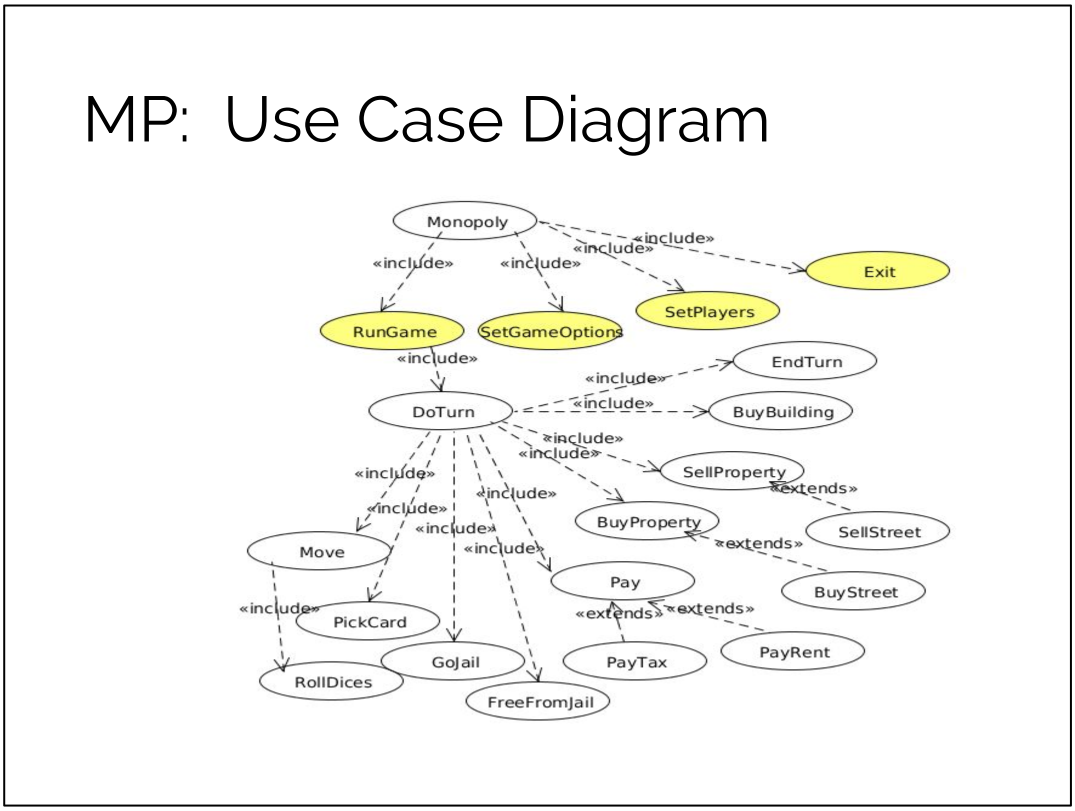


UML Use Case Diagrams

UML use case diagrams not overly useful, but …

- Gives an overview

Document in RAD section 2.3.1 (pictures in Appendix)

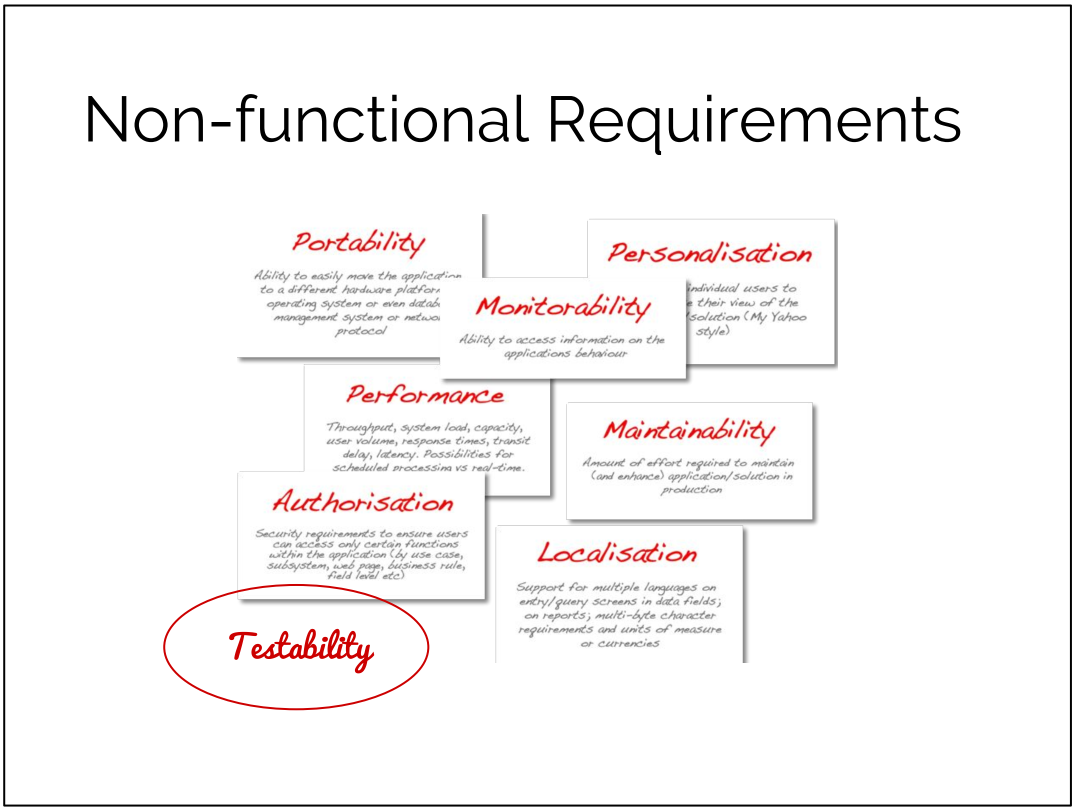


MP: Use Case Diagram

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

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 datab management system or netwo protocol

Personalisation

individual users to their view of the solution (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

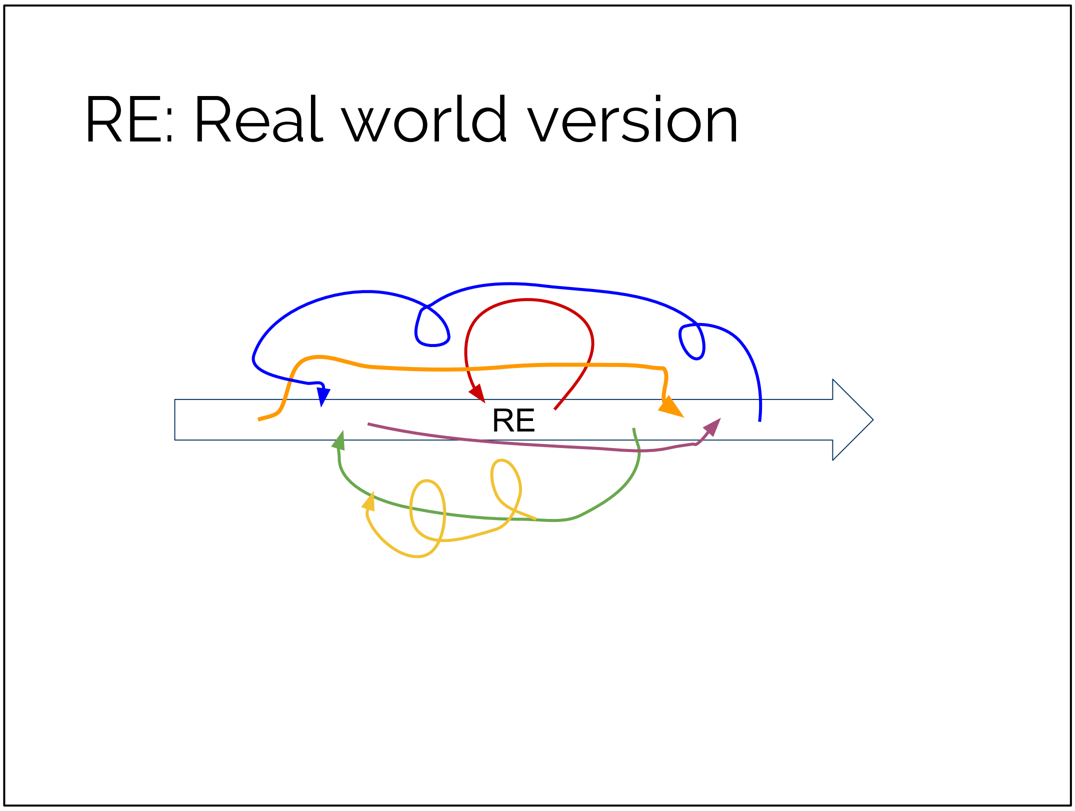
1. Introduction ✓
2.1 Functional Requirements ✓
2.2 Non-functional Requirements ✓
2.3.1 Use Case Model ✓
2.3.2 Use case priority ✓
2.3.3 Domain model ✗
2.3.4 User interface ✓

APPENDIX
Use case model (UML) and texts ✓
GUI sketch ✓

Sections ticked off in slide should have a first preliminary version.

- 2.3.3. Domain model will be documented during next phase
- As noted, pictures in APPENDIX

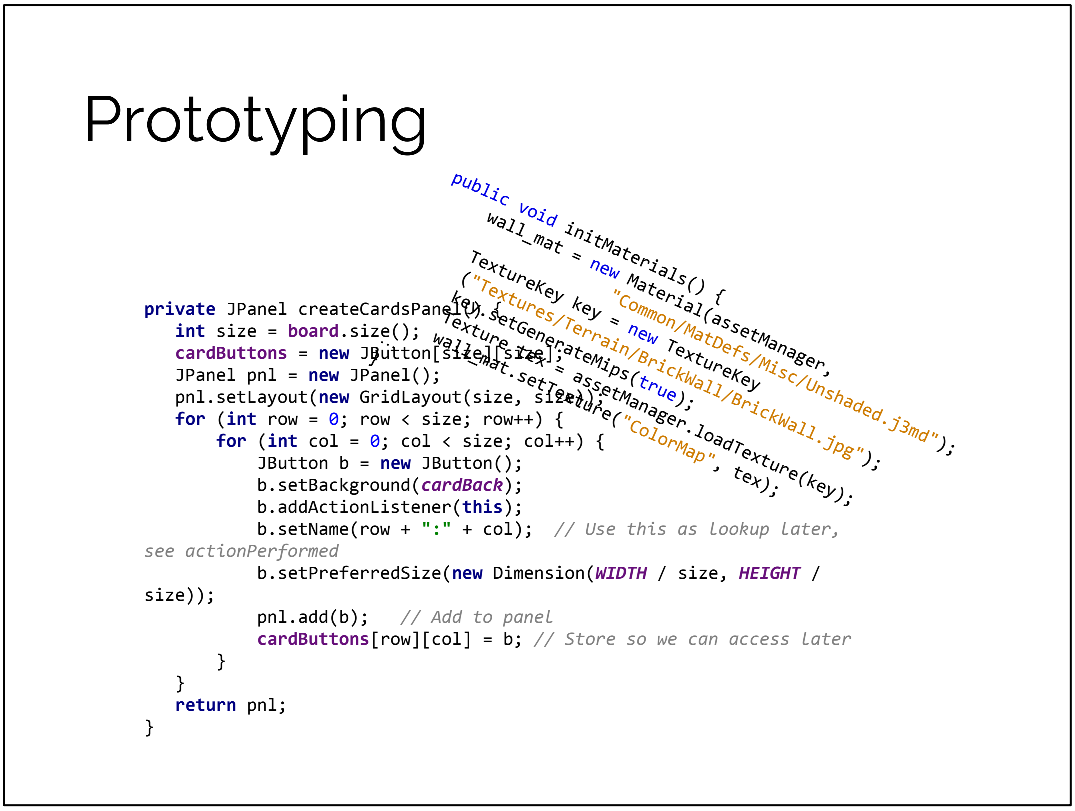
This is of course for the first iteration, more to come …



RE: Real world version

Have done RE in a linear fashion

- In reality RE is more of a parallel iterative process
- Also: Later stages may affect previous



Prototyping

```
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);
}
```

```
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;
}
```

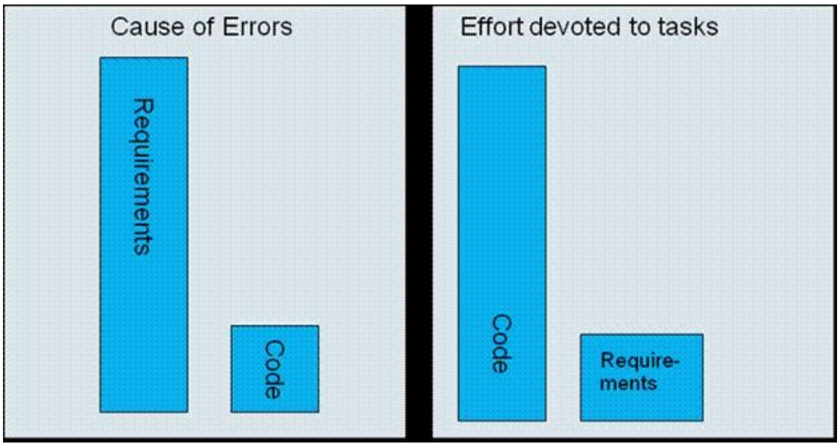
During this phase you should start out technical prototyping

- Technical prototyping for now
    - GUI
    - Services (file handling, sound, graphics, Android, etc….)
    - - Hard code, mock anything you need.

NOTE: There should always be a clean interface to any service, … find it!

- More to come …

Impact of RE



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

Summary RE

Requirement elicitation focus on

- Understanding the problem domain
- To create a shared vision of the project
- Finding functional and nonfunctional requirements
- A preliminary GUI
- RE documented in RAD
- RE is an non-linear iterative process

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