

# Analysis – then and now

- Recap of terms
- What analysis should be?
- *Traditional and ‘modern’ approaches*
- *Problems with analysis*
- Other recent approaches
- *Problem frames*
- *Process oriented approaches*
- Recap and Implications

# Recap of terms

- The **problem domain** (application domain):
  - that **part of the world** within which the **problem exists** (and within which the solution system will operate)
- **Requirements:**
  - the **effects** that the solution system is **required to produce** in the **problem domain** (PD)
- The **solution system** (SS) (aka. the machine):
  - the **system** (to be produced) that will **bring about** the **required effects** in the **problem domain**
- The **current system** (CS)
  - a *pre-existing* (solution) system that serves a purpose similar to that of the new solution system

# Where analysis fits (recap)

**Tasks :-**

**analysis**

**specification**

**design**

**Systems :-**

**problem  
domain**

**interface**

**solution  
system**

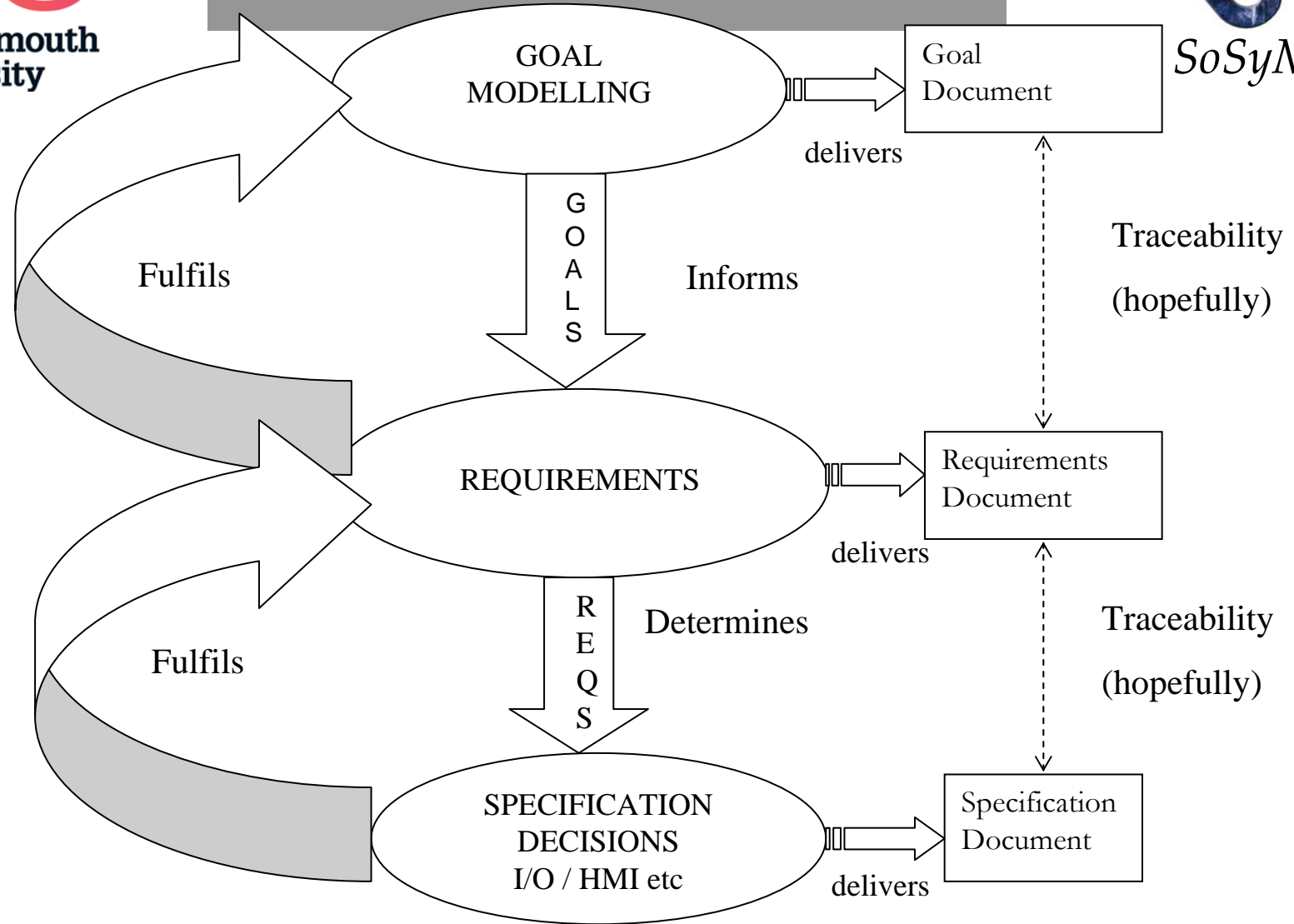
**Outputs :-**

**analysis/  
requirements  
document**

**specification  
document**

**design  
document**

# Requirements in place



## So analysis should be:

- *Describing / considering the goals*
- Gaining understanding of the problem:
  - Modelling the domain / context
  - Modelling the processes within the domain
  - *(Examining any pre-existing solution system)*
- Describing the Problem
- Describing the processes
- Discovering the requirements (problems)
- Documenting the requirements (problems)

*So is it?*

# Definitions of Analysis

- **Inconsistency.** Bray gives the following examples:
  - *“the analysis of requirements*
  - *analysis provides a description of what a system will do*
  - *the process of modelling a system in its environment*
  - *most attention in the analysis phase is given to elaborating the functional requirements*
  - *analysis is the identification, analysis and specification of requirements for a specific application*
  - *study of a specific domain of interacting objects for the purpose of understanding and documenting their essential characteristics*
  - **designing** a solution to a problem
  - *analysts, designers and implementers should all possess the same systems model*
  - *the thought processes flow so naturally from analysis to design that it may be difficult to tell where analysis ends and design begins*
  - *the models that are produced during design show how the various parts of the system will work together; the models produced during analysis show what is in the system and how those parts are related to one another”*

# Historically

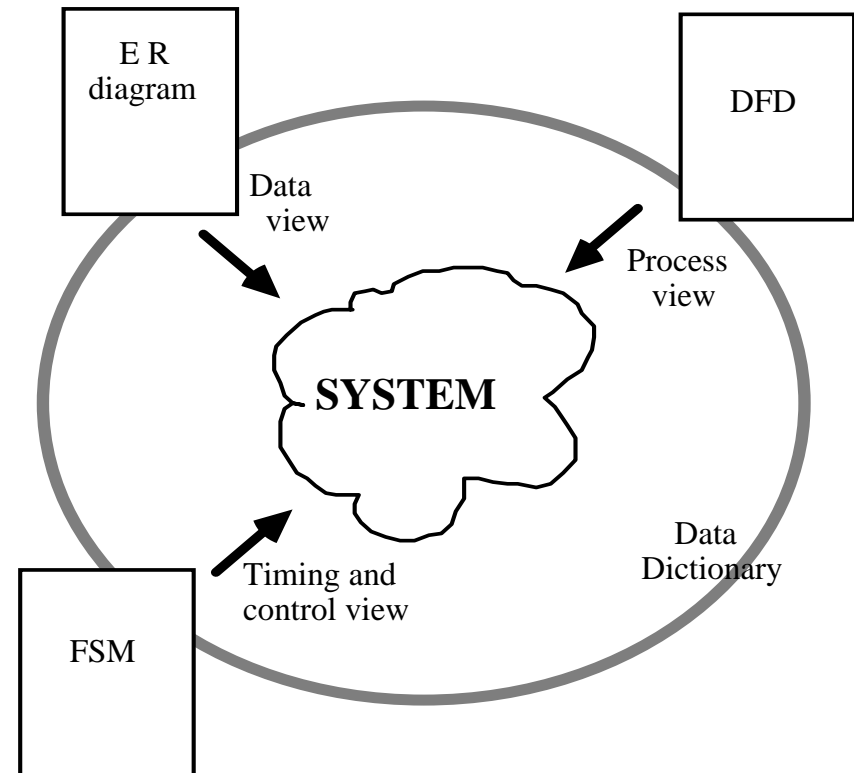
- “Traditional” analysis (1950’s)
- Structured Analysis (late 1960’s)
- “Modern” Structured Analysis (late 1980s)
- Object Oriented Analysis (1990s)
- Problem Domain Oriented Analysis (2000)
- Process Oriented Requirements Engineering
  - from mid to late 1990s, but latterly issues resurfaced within:
    - Requirements Engineering for Business Needs and IT Alignment (2005 onwards).
    - Visual Development Environments (VIDE) – 2006 on

# Analysis (SA & OOA)

- *“There’s a big temptation to believe that you can describe the application domain and the machine all together, in one combined description. . . . .*
- *But if you only make one description, you’ll surely be tempted to put things into it that describe only the machine, and to leave out things that describe only the application domain. After all, you have to describe the machine sooner or later, don’t you ?*
- *You can see the results clearly in many object-oriented modelling descriptions. Often they are accompanied by fine words about modelling the real world. But when you look closely you see that they are really descriptions of programming objects, pure and simple. **Any similarity to real-world objects, living or dead, is purely coincidental.**”*

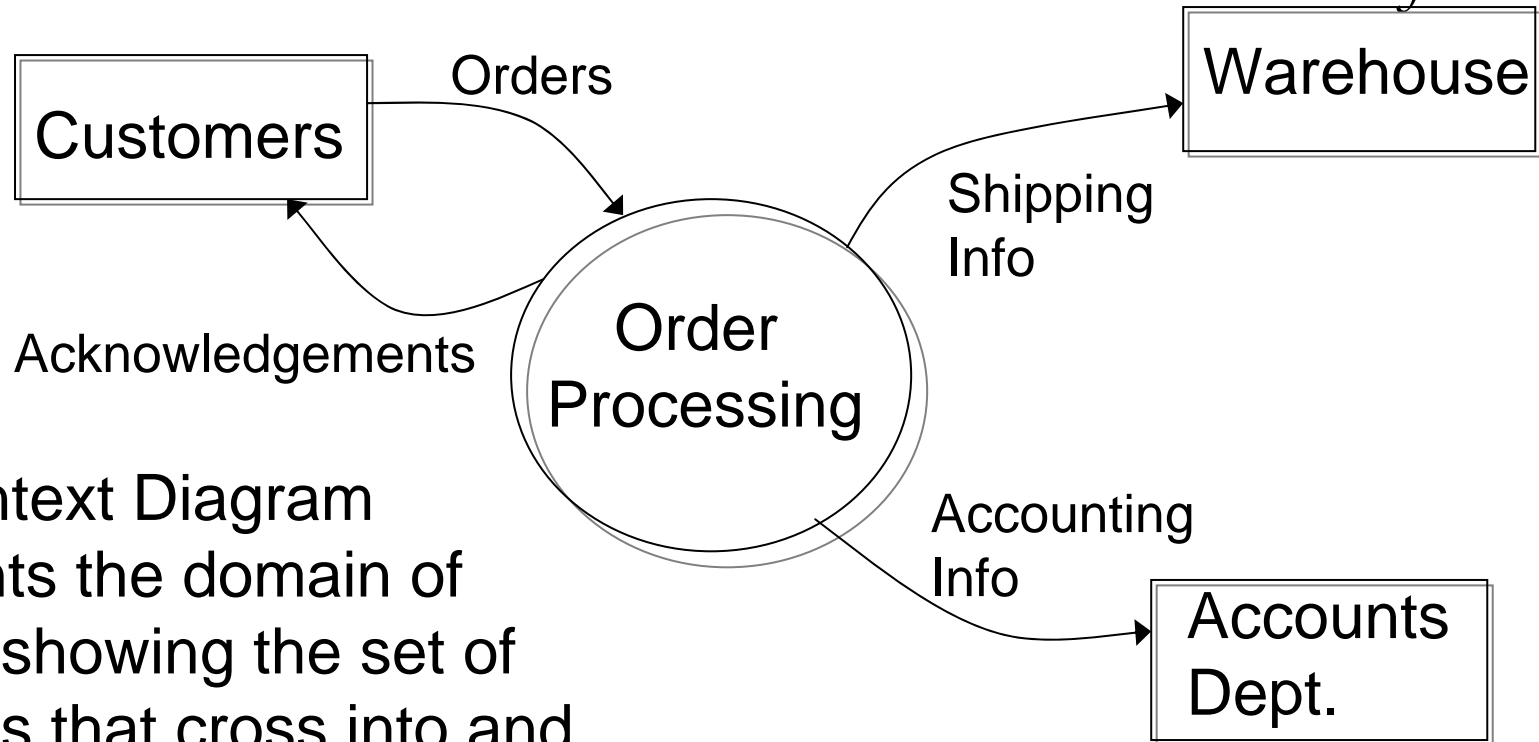
Michael Jackson [JACK95]

- Familiar with SA/SD?
- Move to graphical modelling notations; partitioned, levelled and minimally redundant.
- Structured Systems Analysis (Gane & Sarson in 1977).
- Structured Design (Yourdon and Constantine 1979).
- Tom deMarco's: Structured Analysis and System Specification from Yourdon Press in 1979.
- Many years taught Yourdon's "Modern Structured Analysis", 1989.



- Essential model ‘the essence of the system’.
  - This contains an Environmental Model (describing the environment) and a Behavioural model.
  - Environmental model consists of:
    - **Context Diagram**, Statement of Purpose & Event List.
- BUT “analysis paralysis” (bigger problems, ever “enhanced” method), dubious value added.
- Heavy bias towards DFDs: often amounts to procedural specification. (*Sometimes want that*)
- Problems where no pre-existing system.

# Context Diagrams



“The Context Diagram documents the domain of study by showing the set of data flows that cross into and out of the domain.”

Tom DeMarco, *Structured Analysis and System Specification*, 1978.

# Problem and System

- DeMarco's 'domain of study' is the system.
  - “Boxes (representing sources and sinks) are used rather sparingly in Data Flow Diagrams, and usually not in a very rigorous fashion. *Since they represent something outside the area of our major concern, they exist only to provide commentary about the system's connection to the outside world.*” - DeMarco.
- In Structured Analysis, the Context Diagram shows the context of the *system*, not the context of the *problem*.
  - Contrast with Problem Frames (later), or process models.
- *NB: Still very useful to delineate OUR area of interest or system scope.*

- Seamless Development
  - Object-oriented programming promised the idea of encapsulation and this presented a way to represent things in the real world.
  - A software object could be represented as a design object which was a representation of an analysis object.
  - The assumption was that to represent the real world as an object was simple.

# Objects in Analysis?

- Often objects in the ‘problem domain’ are endowed with properties which real world objects would never exhibit.
  - When did you last send a message to a paycheck?
  - What reply would you get back if you sent a message to an aeroplane?
  - What methods does a tax return perform in response to messages it receives?
  - When the sun rises, does it send a message to each bird to tell it to start singing?

# OO Again (or is it UML?)

- Bray notes:
  - The lift controller. Analysis has a ‘floor object’, which has methods within it.
  - YRRS: Has methods within people in the problem.
- In addition OOA (and the UML in general) does not allow understanding of behaviour, process, or dependencies among events, (*use cases*), all of which are important in understanding user processes.

# Objects are fixed

- Each object belongs to a fixed class, determined when the object is created.
- But the world is not like this:
  - pupils become teachers
  - Students become graduates: well some of them 😊
  - Bills become laws
  - Partnerships become corporations
  - Doctors become lawyers
  - Cotton mills become offices or hotels
  - Caterpillars become butterflies

# Inheritance

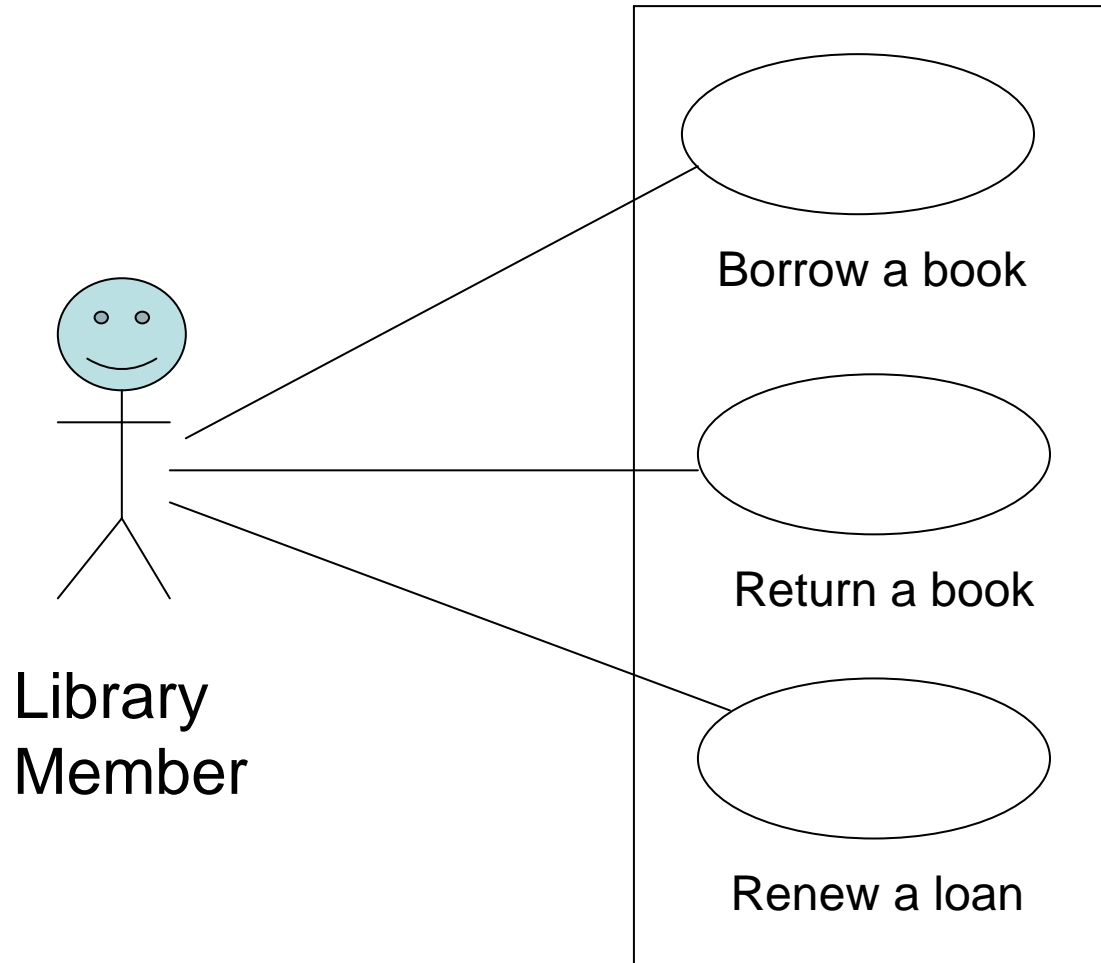
- Each object inherits properties and behaviour from just one class at the next level up in the tree. That's single inheritance. (Unless we have C++).
- But the world is not like this:
  - The logistics manager wants to classify the company equipment as production plant, office equipment and distribution vehicles.
  - The finance director classified it as owned, rented and leased.
  - The two classifications can't coexist in the same single-inheritance hierarchy.

# Active and Passive

- Objects are reactive rather than active. If you don't send a message to an object, it won't do anything.
- But the world is full of individuals, like
  - People (who may initiate interactions)
  - Vessels in chemical plants
  - Government departments
  - who all do things spontaneously.
- All these restrictions have programming solutions - but they are not so good at representing the richness of the world.

- UML tells us that the way to do requirements engineering is to capture the functions that users want when they use the computer.
- It's the first question we ask our customers, "What functionality do you want?"
  - This way madness lies.

# A Use Case Example

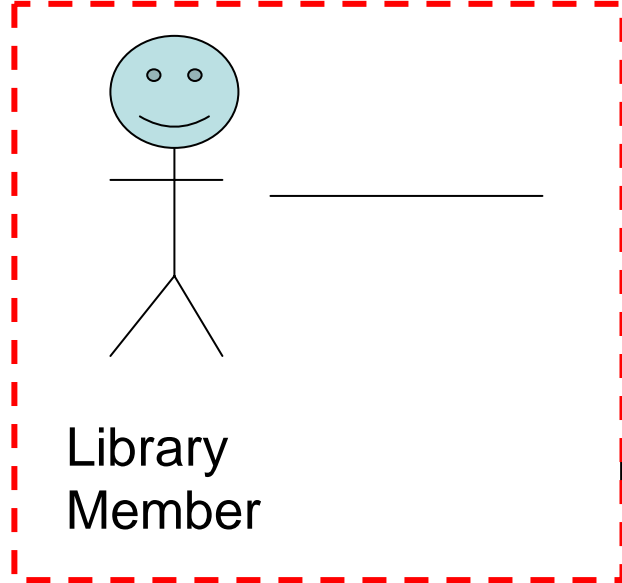


# The UC Problem



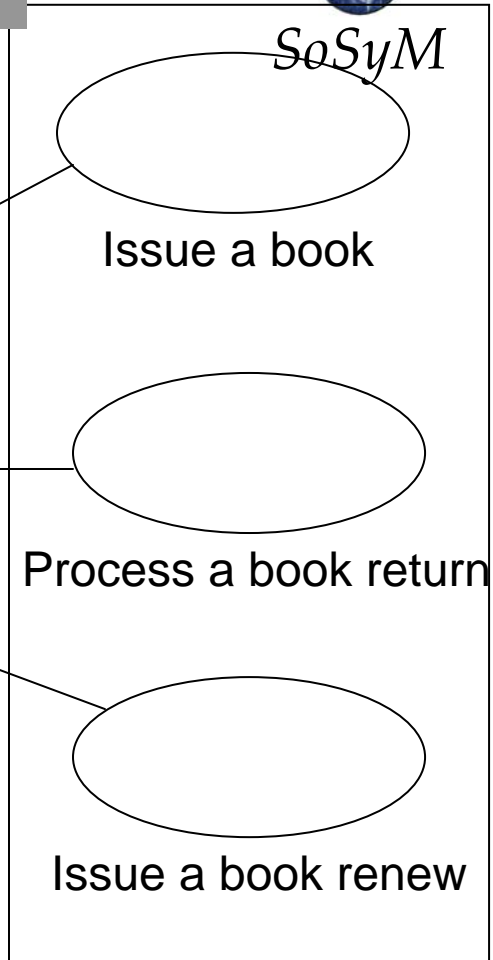
SoSyM

The Library Member's requirements: borrow a book, return a book, renew a loan



Library Member

Librarian



Issue a book

Process a book return

Issue a book renew

The Librarian has to: issue a book, process a book return, and issue a book renewal. The Library Member's requirements aren't quite the same as the Librarian's tasks.

# Use Case problems

- So what happens when a librarian's job is to loan books out and collect returns?
- What happens to the Library Member when we cannot show actors one step removed from the machine?
  - Do we just forget about them or fudge them?
  - But isn't a library's purpose to provide books and services to its library members?
  - And if we ignore this fact, what are we essentially doing?
  - We're ignoring the problem context, and we're ignoring the requirement: **and that's bad**
- *Dependencies in Descriptions (big problem to come)*
- ***Note. We will (of course) adopt use cases anyway!***

## Story So Far

- SA/SD – concentration on system context.
- Tendency to model existing system (rather than the ‘essence’ or ‘business need’, or even the process.
- Still provides useful notational tool-kit
- OOA. Real world objects? Not really.
- Use cases: Oh dear, oh dear oh dear...
- So what else is there?

# Homework / Exercise

- EX 1 Find an example of analysis (structured or OO, or even use cases) that you don't like and:
  - Say what's wrong with it. Remember:
    - Does it really model the problem or domain?
    - Are the objects real?
    - Does it reflect the problem?
  - IFF you can't find one, try and invent one.
  - Send to me (as 1 or 2 Slides: e.g., picture and critique).
- EX 2: Types of Requirements (courtesy if Ian Bray).
  - Identify type from given list.