Bournemouth University

software systems research centre

Software Systems Modelling

Dr Keith Phalp (and others...)

Outline for today

- Overview of the unit and how the unit will be delivered.
- Describe core content and scheme of work.
- What we are looking for at level H.
- Gentle intro, to give some context and generate discussion.
    - For Software models (and UML), how did these evolve, what were the motivations, what where the alternatives, how do we know what is best and so on?
- Find out your own backgrounds and skills.
- Recap some basic understanding of OO and UML.
- Outline exercises.

Background to this part of the course

- Software Systems Modelling means many things to many people.
- In essence, we are looking at a variety of ways in which modelling is used to help us to build software systems.
- To some extent this has been a very long term goal of software engineering.
- However, some other (perhaps lower level) generic modelling techniques, e.g., patterns, may also prove useful.
- Will try to consider both approach 'method' and modelling notations (e.g., UML diagram).
- Will try to learn from your experiences, and bring in expertise for specific topics.

My approach to the unit and level H

- Final year is often a big change in culture and approach.
- You have experience and expertise (you can do the work), but this is about building your critical powers and reflection.
- You will still have to demonstrate competency in software modelling, but greater emphasis on critique.
- This is assessed by coursework (30%) and examination (70%).
- For this year many of the topics will be presented (often by guest speakers) as lectures often with suggested reading.
- There should be no major surprises.
- My aim is to get coursework out (and back in) early and to try to get feedback to you before Christmas.
- Revision lectures before Christmas will review material covered and highlight topics.

Doing versus reflecting

At level H, you need to not only ‘do things’ but also reflect upon them,

and the processes and concepts.

E.g., Expected that you ought to be able to ‘do’:

Class diagrams, sequence diagrams, refactor, etc..

However, its the reflection that makes it H-level.

We will try to take this approach throughout.

Though you often illustrate an arguments with examples in different notations.

My background

- Wide ranging interests with a focus on Software Engineering. Of particular relevance to this unit:
  - Taught Systems design (various methods) since 1992, and have used Systems Design for project builds.
  - Originally SASD (Yourdon), then OO from mid-90s; OMT (Southampton), Coad and Yourdon (Bournemouth) and then UML (BU)
  - Have written about aspects of UML Design (e.g., OOPSLA 99) and specification (extensively throughout last decade or so). .
  - Also taught PDOA here at BU, though this tends to focus on requirements and analysis rather than the design phase.
  - Hence, not tied to one approach, rather it is about choosing, adopting or even evolving approaches for different problems.
  - Principal Investigator on EC funded project (total 2.3M Euros) which investigated and developed a model driven toolset, working with leading industrial and academic partners (2005-2009), and which was rated highly by European Commission.
  - Much experience of Empirical Software Engineering (which is, in essence, trying to investigate whether methods, tools, notations etc., used in SE really help).

Unit Spec and ILOs

1) Demonstrate expert knowledge of the changing nature of software-intensive systems.
2) Select appropriate techniques systematically from the range of methods and tools available to develop such systems.
3) Demonstrate expertise of object oriented modeling, and apply the Unified Modelling Language (UML) to produce appropriate software design models for software-intensive systems.
4) Apply Model Driven Architecture (MDA) and patterns in order to create designs for business applications effectively.
5) Understand the professional issues, implications and impact of the production of software systems models.

The weighting of coursework to examination is 30:70.

Indicative Assessment Information

Learning outcomes 2, 3 and 4 will be assessed by coursework, equivalent to 2000 words. A typical assignment will be to produce a series of software design models appropriate to a given scenario or scenarios.

Typically, learning outcomes 1, 3 and 5 will be assessed by end of year examination.

Core content

- Core Topics (Skills)
  - For some given scenario be able to produce appropriate design models (***UML***)
  - To be able to ***select*** (or at least argue the merits) of different techniques
    - Clearly this suggests exposure to alternatives *(Question to you from your experience: what could these alternatives include?)*
  - To be able to show how you would use ***MDA***
  - To be able to show how you would use ***patterns***.
- Discursive Topics:
  - Demonstrate expert knowledge of the changing nature of software-intensive systems.
  - Some context or perspective on methods.
  - For example, where did these modelling methods all come from, how did they evolve, how different are they really, and how does this help us choose the best approach for a given problem?
  - Impact of what you are doing, professional issues.

Draft Teaching Scheme

Unit week	w/c	Topics for Lecture / Presentation	Supporting Seminar Session	Presenter / Lecturer
1	03/10/11	Overview of the Unit. Explanation of the approach for the unit this year. Finding out your Software Modelling Background and Experience. What models and method your or your placement companies used. Evolution of Software Systems Models. May (if time allows) recap class diagram approach and evolution of UML. What advantages of OO over previous approaches (simple example to contrast different perspectives).	OO Modelling examples from Small Scenarios. E.g., tea, Hovel, Car Park	Keith Phalp
2	10/10/11	Recap of OO Modelling (the mainstream usage) and contrast with / discuss other appraoches (e.g. SASD). More parts of the UML, e.g., Sequence diagrams Statecharts. Understanding what design means for large scale Software Systems, Architectural Issues. Assessing and Measuring Designs.	Modelling exercises	Keith Phalp
3	17/10/11	Design Patterns 1: Pt 1, Introduction, What is a pattern? History of patterns in SE. Why are patterns useful? Gang of Four Design Patterns (Gamma et al). Categorising GoF - creational, structural, behavioural, Creational patterns, Structural patterns, Behavioural patterns, Larman's GRASPs (General Responsibility Assignment Software Patterns)	Patterns exercise	Richard Gunstone
4	24/10/11	Model Driven Development (experiences from development of MDA tools). Presentations of existing tools and what they bring for Software Systems Modelling.	Literature based exercise	Keith Phalp
5	31/10/11	The Darker Parts of UML: A tour of the darker corners of UML as used by a practising business analyst. These lectures and activities will present the practical value of some of the elements, definitely including Statecharts, Activity Diagrams and even OCL. We will also briefly review the value of the whole UML project.	UML Exercises	Steve Webster
6	07/11/11	The Darker Parts of UML 2	UML Exercises	Steve Webster
7	14/11/11	Design Patterns 2: Pt 1: Broader initiative relating to patterns, IBM Patterns for eBusiness, Applications of patterns. Have patterns lived up to their promise? Commonly used UML: Case study, Examples of Patterns being used, Frameworks and Relationship to Patterns: Recommended reading.	Patterns exercise	Richard Gunstone
8	21/11/11	Support for Assignment Hand in. The Hand in date is Monday 28th. Therefore support this week and lecture next on additional topic.	Support for Assignment	Keith Phalp
9	28/11/11	Domain Specific Languages & Converge	No seminar	Laurie Tratt
10	05/12/11	Revision Session (Lecture Only)	Revision	Keith Phalp
11	12/12/11	Revision surgery Sessions. Also aim to give assignment feedback this week (may need specific session)	Feedback Session	Keith Phalp

What have you used?

- So far we have gathered an impressive list of experience.
- *Most have used class diagrams*
- *Fair few used class diagrams in placement*
- *Smattering of use case experience (again placement)*
- *Some with significant industrial experience, e.g., from British Aerospace, Class, Sequence and state-charts.*
- *Some experience with structured.*
- *Those OO all used UML*
- *Range of languages, mostly various incarnations of C, C++, Java, C#.*
- What about methods (often, wrongly, called methodologies).
- What about frameworks (e.g., Eclipse), model driven, patterns, any other pertinent SSM techniques?

Where did Methods come from?

Software “crisis”

more like “malaise”

Software “Engineering” 1968 (nearly as old as me!!)

Methods for programming, then design, then analysis…(first structured then …)

The need to understand.

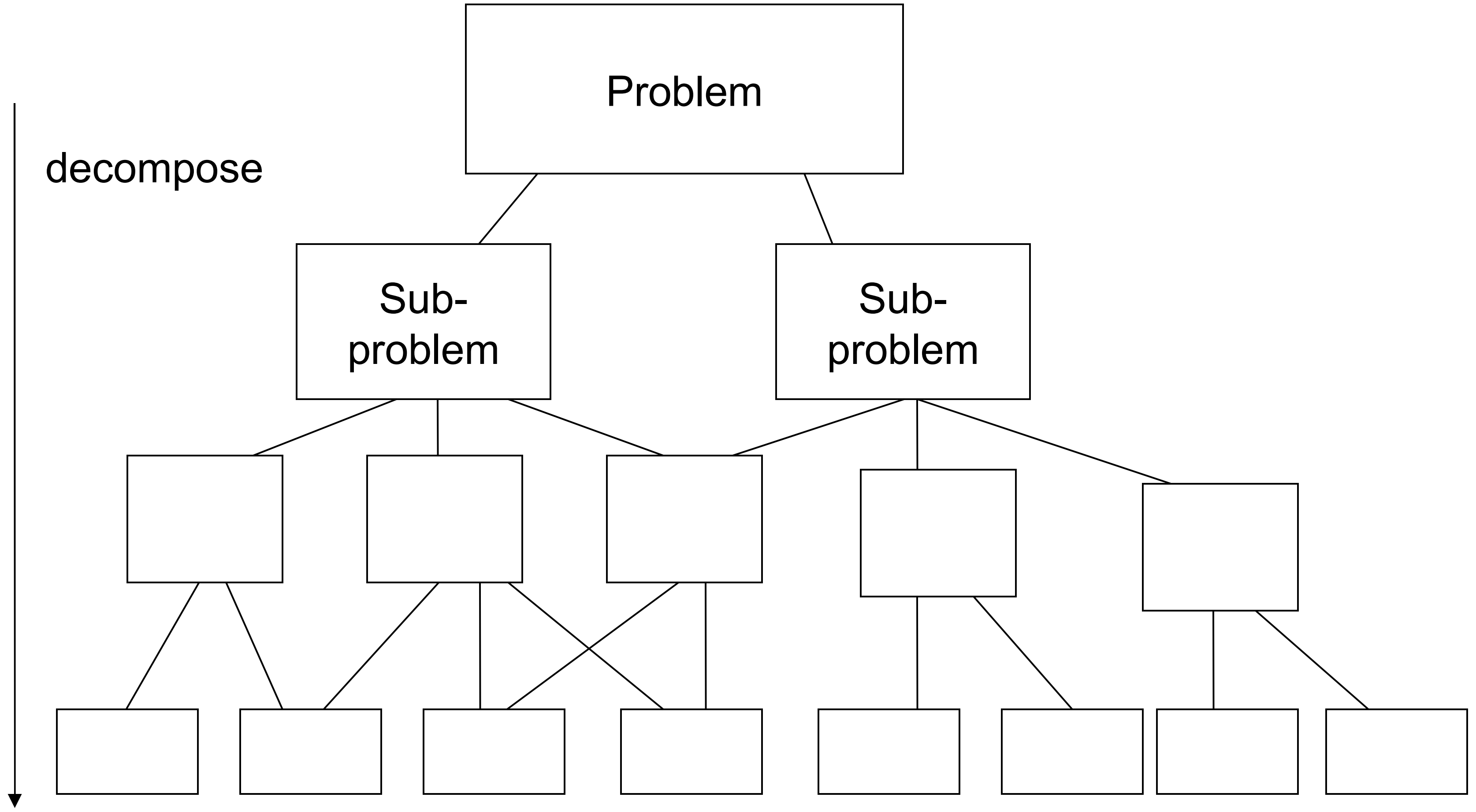
The need to conceptualise.

The enforcement of documentation.

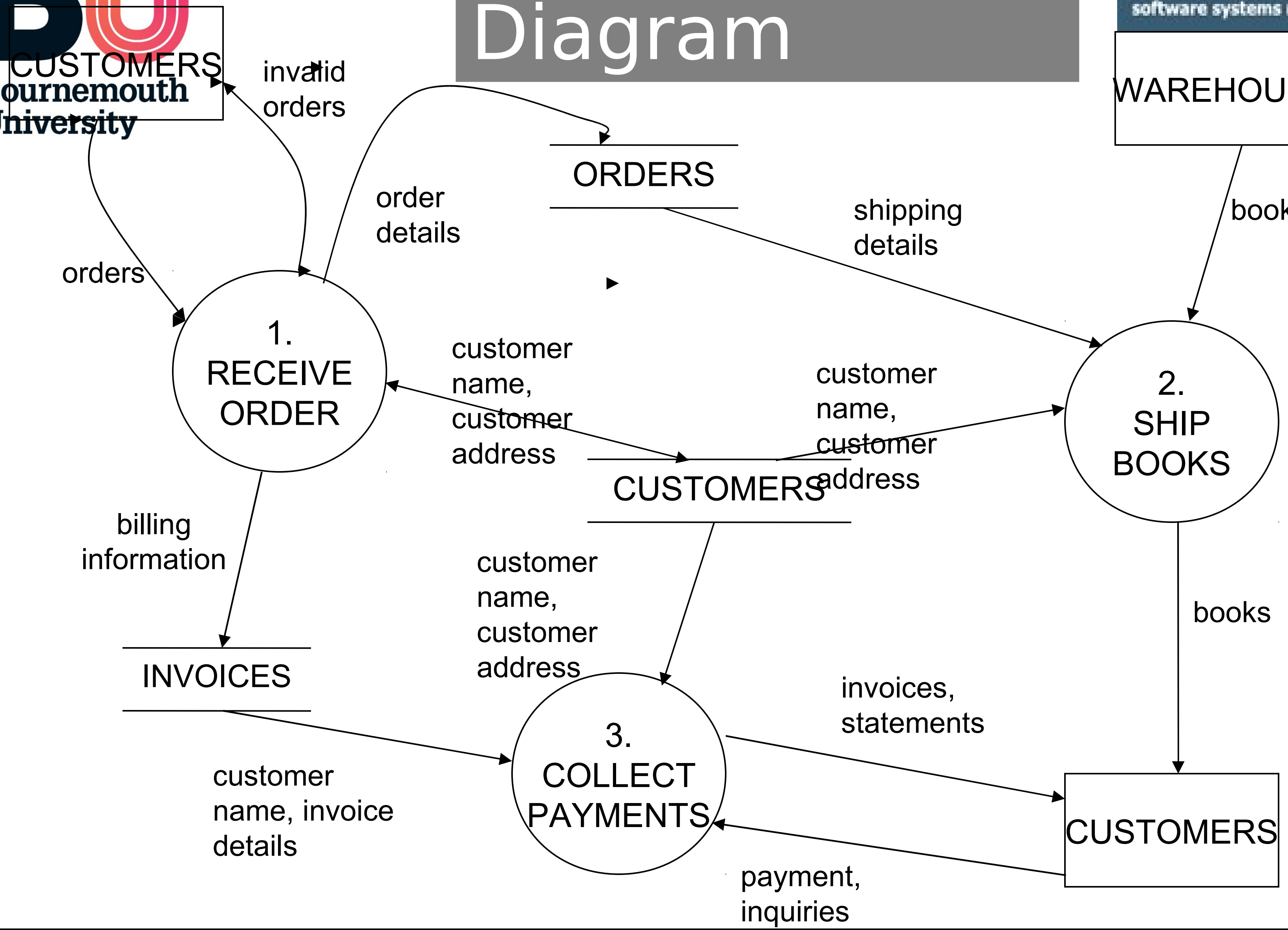
Other reasons? ***(Suggestions…)***

What was there before our current crop?

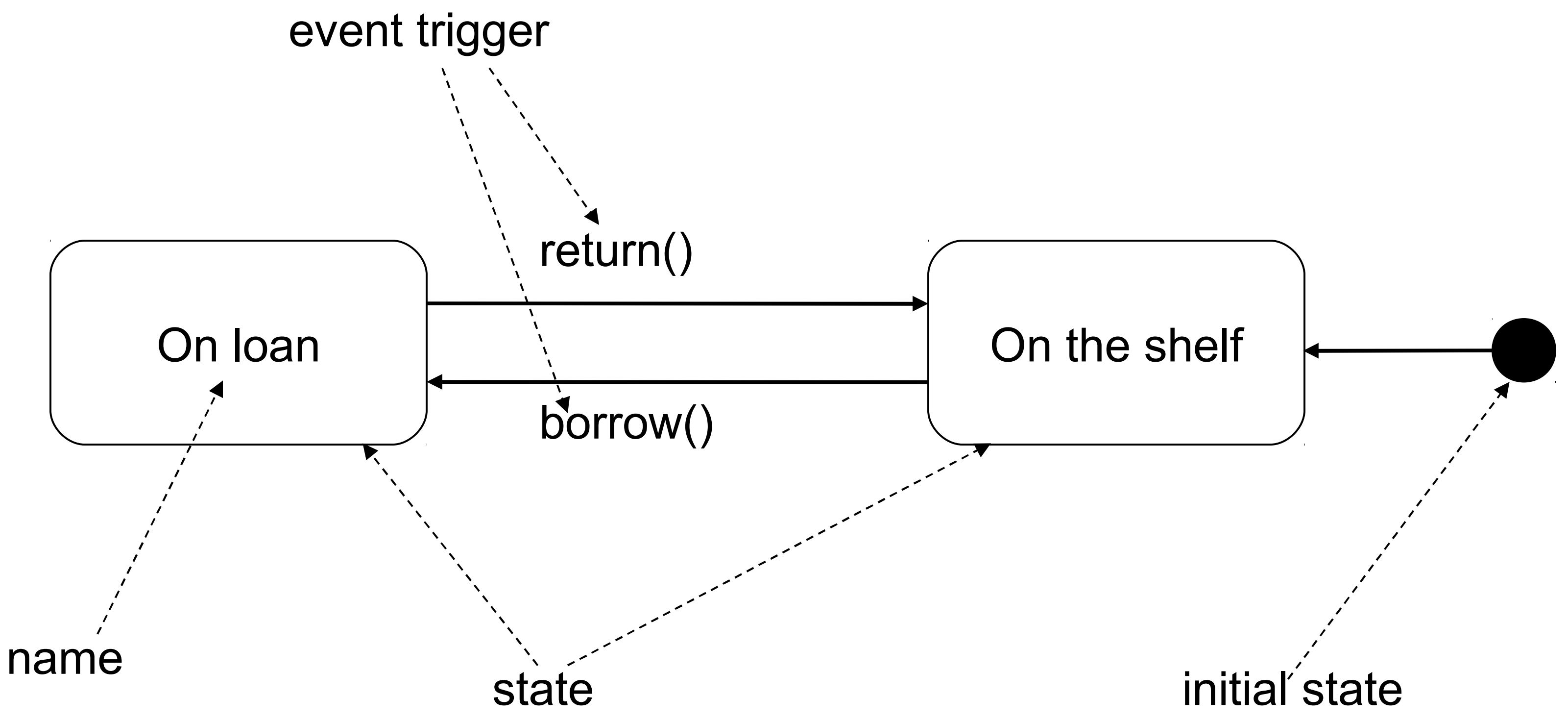
Top-down decomposition



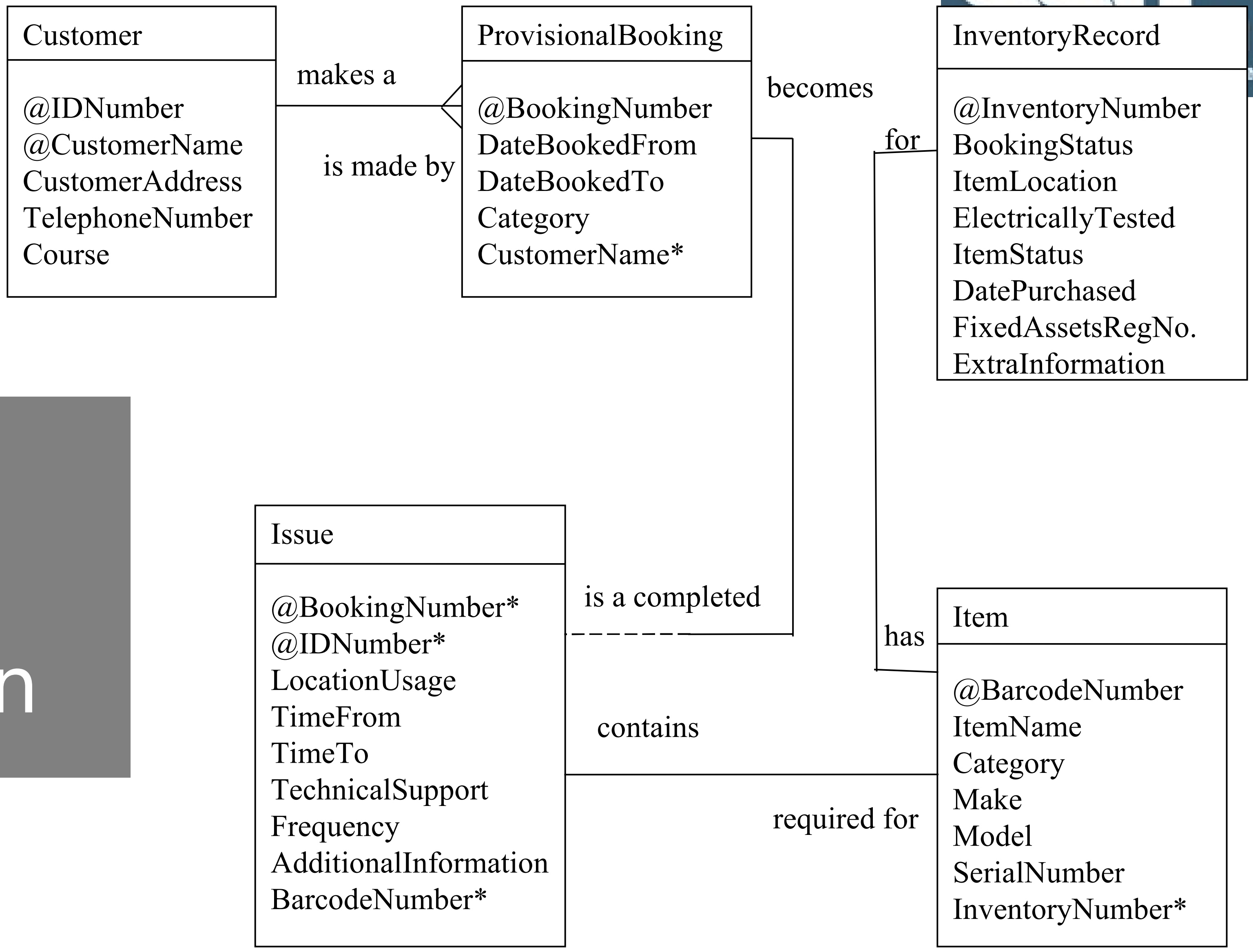
Data Flow Diagram



Finite State Machine



ERD Basic Version



OO (and other 'Modern' ideas)

Object-orientation (OO). A case of reverse engineering. OO a programming approach (Simula67, Algol 68, later: Smalltalk, C++, Ada (sort of), Java), then led to OO Design (very succesfull in adoption and many incarnations) and then to OO Analysis (which has never really lived up to its promise).

Patterns (Gamma, onwards...),

SOA, and variants..

refactoring,

OMT, C&Y OO, Booch, Schlaer & Mellor, OOSE, UML

Model driven (MDD and MDA),

Domain Specific Languages (DSLs).

Frameworks for programming (e.g., Eclipse)

What about functional languages, declarative approaches, executable specification?

Motivations for OO

Failures of existing approaches. Software crisis (malaise).

Increasing problem domain complexity.

Increasing size of software.

Inter / intra project communication problems *(e.g., different analysis teams, and design teams using different models).*

Maintenance ***(How much of development is maintenance – your estimates)***.

Legacy Systems (t***ales from the Process Project and SEBPC***).

The Hope for Reuse ***(tales from Bosch) – and product lines.***

Notational Arguments (Coad's Canyons and the 'too many views' argument), plus issues with 'existing' approaches.

Motivations for OO

Failures of existing approaches.

Increasing problem domain complexity.

Increasing size of software.

Inter / intra project communication problems *(e.g., different analysis teams, and design teams using different models).*

Maintenance ***(How much of development is maintenance – your estimates)***.

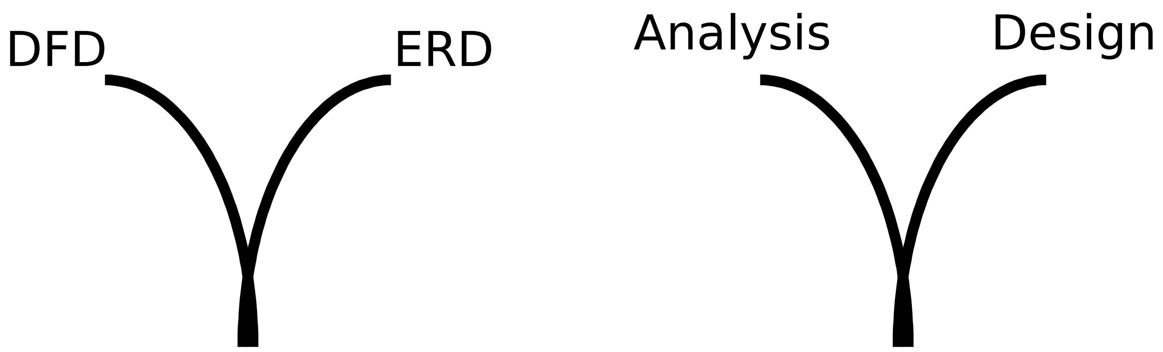
Legacy Systems (t***ales from the Process Project and SEBPC***).

The Hope for Reuse ***(tales from Bosch) – and product lines.***

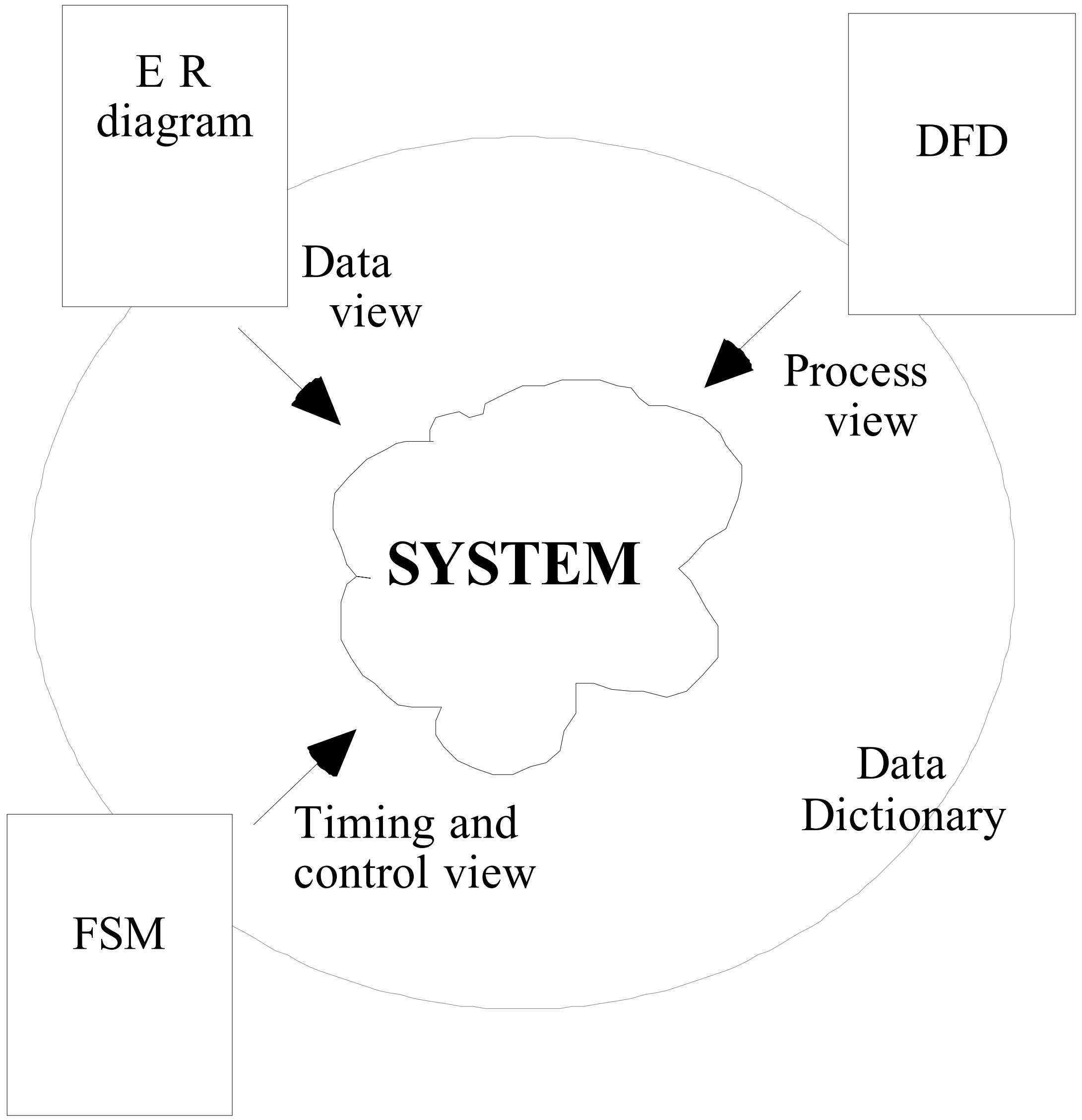
Notational Arguments (Coad's Canyons and the 'too many views' argument), plus issues with 'existing' approaches.

A Continuum of Representation

Peter Coad suggested two 'Grand Canyons'



These canyons bridged by a common notation


E R
diagram
DFD
Data
view
Process
view
SYSTEM
Timing and
control view
Data
Dictionary
FSM

Limitations of DFDs

Data flow approach focuses on system processing issues. Processing requirements are quick to change.

Stored information spread all over the DFD hierarchy.

Conversion of DFD to code (or even to structure chart) is very arbitrary.

At last: UML

- Unified Modeling Language (UML) is one of a number of approaches to providing a notation for OO.
- Builds upon other approaches, e..g., Booch, OMT, OOSE.
- Hugely popular & widely used.

OR

- A general purpose visual modelling language that is used to specify, visualize, construct and document artefacts of a system.
- UML provides notation to describe OO designs; geared for Object Oriented systems

- Parts of UML could be applicable to other programming paradigms, indeed, arguably are not really OO (e.g., use cases (OOSE) procedural in nature, statecharts – were part of STATEMATE).

Evolution of UML

- In essence this is nothing more than a collection of different modelling techniques: Booch, Jacobson, Rumbaugh.
- By early 1990's there was disparity among different camps, and a standard was needed (or at least this was argued).
- Huge impact of three major camps (the three amigos) combining forces within UML.
- Taken on by Object Management Group (OMG), and major success with version 1.1, 1997. Changes, at 2.0, often use 2.1, and now 2.3.
- Original set of diagrams was: Class, Object, Use Case, Sequence, Collaboration, Statechart, Activity diagram, Component diagram, Deployment diagram.

Further Exercises

A recap of finding objects (can also use to contrast OOA with OOD. (First Tea & then Pizza)

Extras: Have you ever tried CRC cards? (Worth trying). Introduction to the approach also given

BONUS MATERIAL: A quick whizz through (or a recap) of classes in OO (UML), and techniques for finding classes.

Two versions available: (the more abstract argument based, or the more code-based).