Introduction to Structured Analysis and Design

Dave Levitt

CS 2000: Systems Analysis & Design

Agenda

- Questions?
- Vote on topics for next week’s class
- Intro. To Structured Analysis and Design
- Lab

Other Requirements Templates

- Besides use cases, use case models, and non-functional specifications, there are more traditional requirements templates available:
  - Industry standard, I.e. IEEE std. 830-1993 (proprietary)
  - Volere, (presented later)
  - Home grown
- Most of the better templates capture the same types of information (functional, non-functional, etc.).
- IEEE and Volere tend to be monolithic, which can lead to problems such as ????

Volere

- Developed by noted industry practitioners Suzanne and James Robertson.
- Available at www.**systemsguild**.com
- For more information, see “Mastering the Requirements Process”, Robertson & Robertson, 1999, Addison Wesley.

Volere (cont’d)

- Comparison to RUP:

– Project Drivers	→	Vision Document
– Project Constraints	→	Non-Functional Specification
– Functional Requirements	→	Use Case Model, Use Cases
– Non-Functional Reqmt’s	→	Non-Functional Specification
– Other	→	Actor Report, Data Definition, Domain Model, Project Plan

- Personal opinion: very comprehensive, but too monolithic.

History of Structured Methods

- Structured methods represent a collection of analysis, design, and programming techniques that were developed in response to the problems facing the software world, circa 1960’s to 1980’s. In this timeframe:
  - Most commercial programming was done in Cobol and Fortran, then C and BASIC.
  - There was little guidance on “good” design and programming techniques.
  - There were no standard techniques for documenting requirements and designs.
- Of course, while it was (and is still) possible to develop world-class software, it becomes harder and harder to do so as systems get larger and more complex.

History of Structured Methods (cont'd)

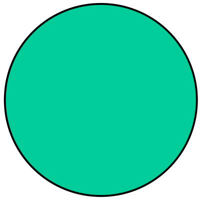
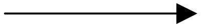
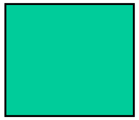
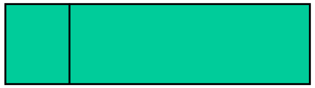
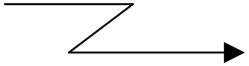
- Structured Methods emerged as a way to help manage large and complex software:
  - Structured Programming – circa 1967
    - Go To Statement Considered Harmful, Edgar Dykstra
  - Structured Design – circa 1975
    - Larry Constantine, Ed Yourdon
  - Structured Analysis – circa 1978
    - Tom DeMarco, Yourdon, Gane & Sarson, McMenamin & Palmer
  - Information Engineering – circa 1990 (James Martin)

Structured Analysis

- Primary artifacts are a data flow diagram (with data dictionary and mini-spec's), and entity relationship diagram

- A data flow diagram:
  - Shows processes and flow of data in and out of these processes.
  - Does not show control structures (loops, etc.)
  - Contains 5 graphic symbols (shown later)
  - Uses layers to decompose complex systems (show later)
  - Can be used to show logical and physical
  - Were a quantum leap forward to other techniques at the time, I.e. monolithic descriptions with globs of text!
  - Still used today to document business and/or other processes.

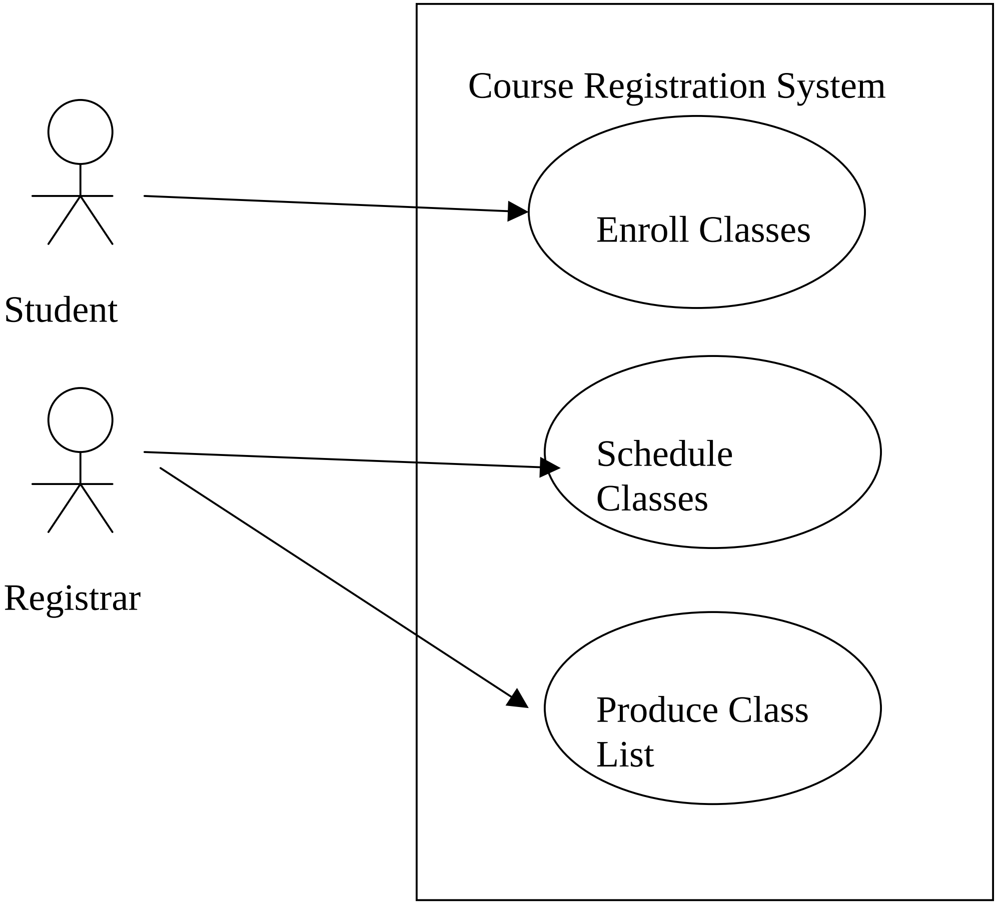
Structured Analysis (cont’d)

Symbol:	Meaning:	Description:
	Process	A series of one or more steps that converts inputs to outputs. Each process is followed by a mini-spec (shown later)
	Data Flow	Shows a data path (flow of data)
	External Agent	A source or sink of data. Lies outside the system
	Data Store	Data at rest, usually a file or database table
	Real-time link	A communication link. This symbol added later. When ???

Structured Analysis (con’t)

- To manage complexity, data flow diagrams are done in layers:
  - The uppermost layer is a context diagram.
    - Shows system boundary, I.e. the system, external agents, and data to/from the agents. Does this sound familiar?
  - The next layer is a level zero.
    - Shows primitive processes, data stores and data flows, and of course their relation to external agents,
  - The next layer level(s) is a level 1 through level ‘n’
    - Decomposes one of the processes from a level zero diagram.
    - If a level one diagram is overly complex (more than 7 +- 2 processes, it can be further be decomposed to a level 2-n, and so on.
  - Each lower layer “traces” to its higher layer (shown later).

Intermezzo #1



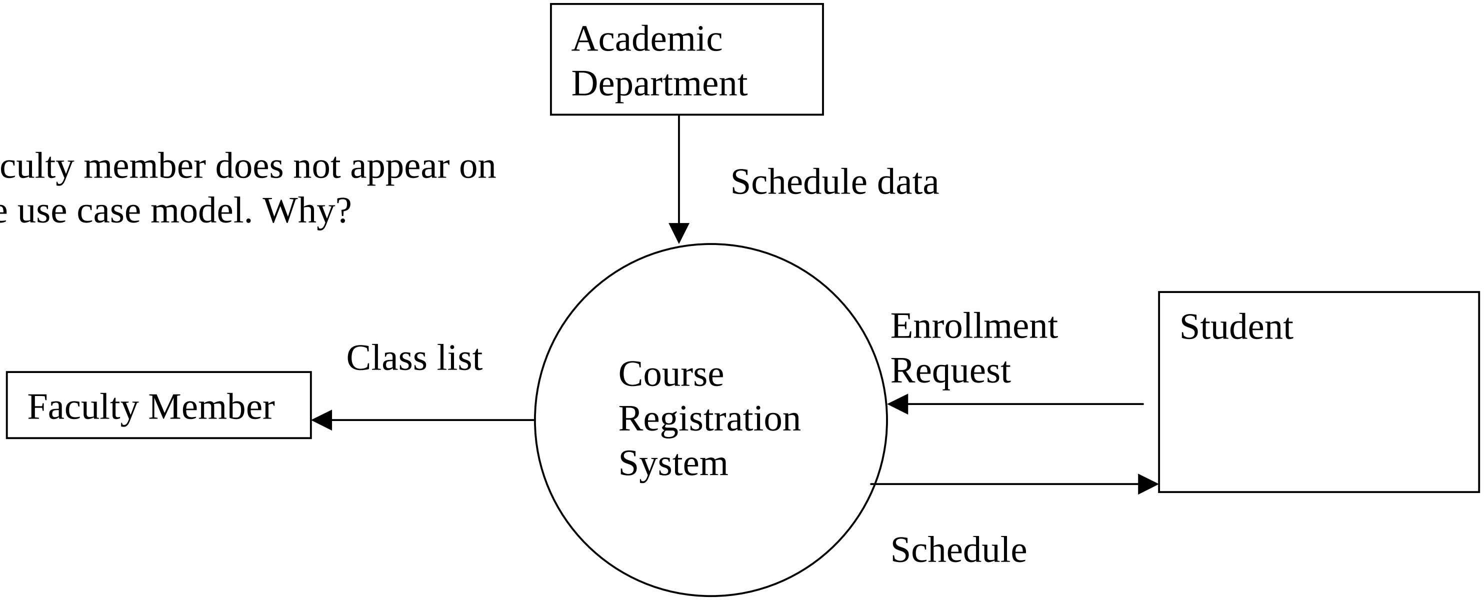
This use case model describes a simple course registration system.

We will soon see this same system represented by a set of data flow diagrams.

Structured Analysis (con't)

Context Diagram – Course Reservation System

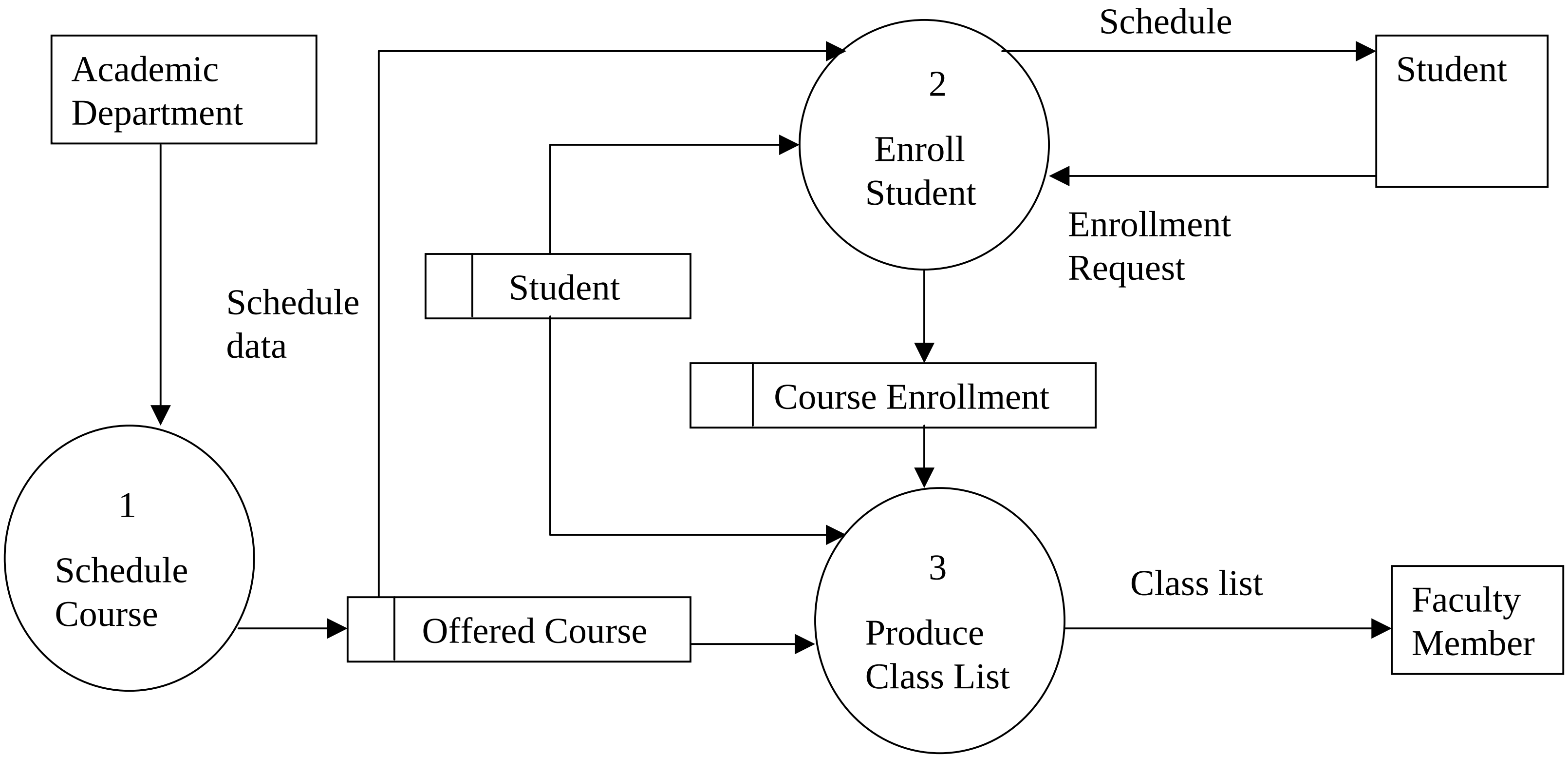
Faculty member does not appear on the use case model. Why?



Source: "Systems Analysis and Design in a Changing World", Satzinger, Course Technology, 2002

Structured Analysis (con’t)

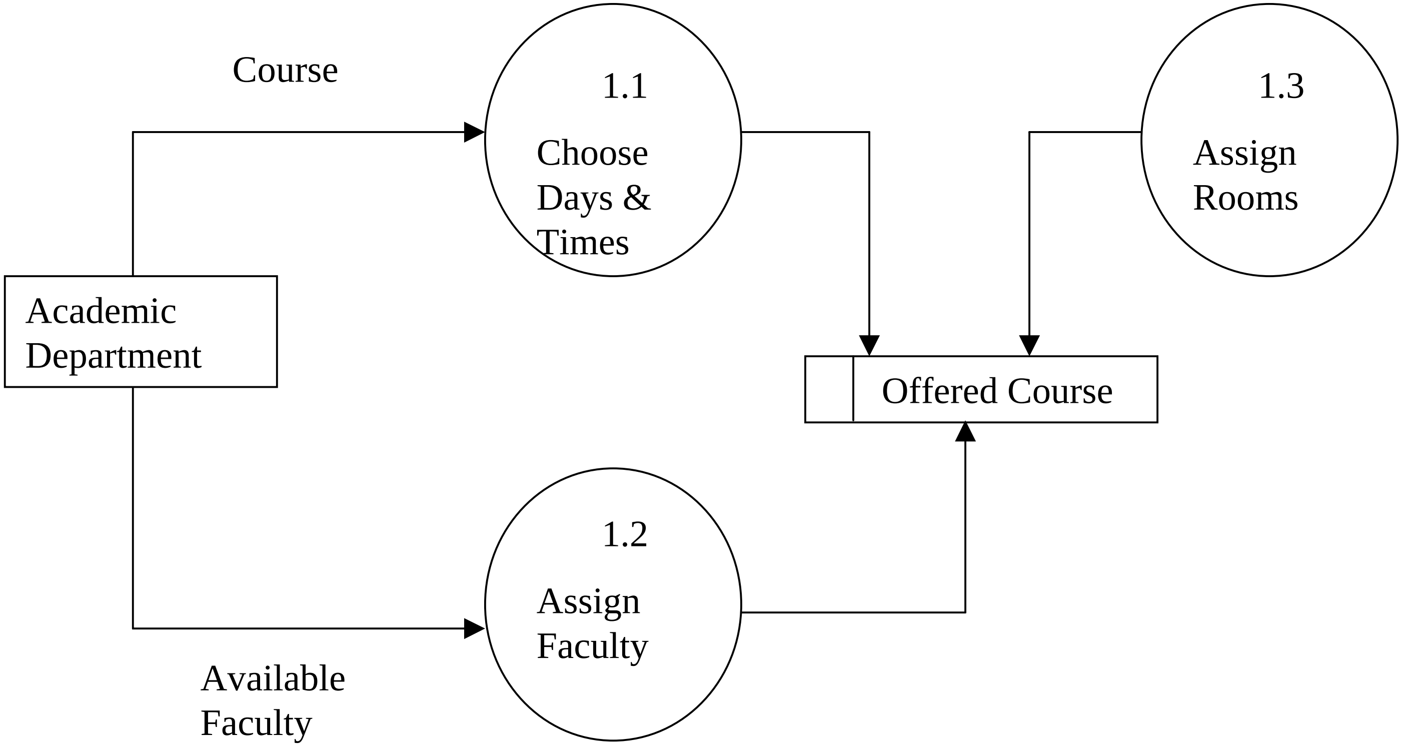
Level 0 Diagram – Course Reservation System



Source: “Systems Analysis and Design in a Changing World”, Satzinger, Course Technology, 2002

Structured Analysis (con’t)

Level 1 Diagram – Course Reservation System



Source: “Systems Analysis and Design in a Changing World”, Satzinger, Course Technology, 2002

Structured Analysis (cont’d)

- Observations about each diagram:
  - The **context diagram** partitions the entire system. It has only one process (the system), and from it, the data flows to/from the external agents.
  - The **level 0 diagram** decomposes the system into 3 processes: Schedule Courses, Enroll Student, and Product Class List. Notice that the 4 data flows represented in the context diagram are preserved in the level 1. This is required.
  - The **level 1 diagram** decomposes the Schedule Course process into 3 sub-process.
    - Note that the data flow Schedule Data from level 0 is broken into 2 sub data flows in the level 1: Course and Available Faculty. Also note that the Offered Course file is still preserved.

Structured Analysis (cont'd)

- Additional:
  - Create lower layer diagrams when the diagram is getting too complex. General rule is 7 +- 2 processes. Sound familiar
  - For each sub-process, a mini-spec will be written (shown later).
  - Note that a data flow diagram is concerned about data flow and functional decomposition. By contrast, a use case model (and use cases) are described in terms of actor's goals. More on this later.

Structured Analysis (cont'd)

- Rules for Data Flow Diagramming:
  - Process:
    - No process can have only outputs
    - No process can have only outputs.
    - A process has a verb-phrase label (sound familiar)
  - Data Store:
    - Data cannot move directly from one data store to another. It must flow through a process.
    - Data cannot be moved directly from an outside data source or sink to a data store. It must first go through a process.
    - A data store has a noun-phrase label. Hmmm, perhaps like a class name?
  - Source / Sink:
    - Data cannot move directly from a source to a sink. It must be moved by a process.
    - A source/sink has a noun-phrase label.

Adapted from Modern Systems Analysis and Design, 3rd edition, Hoffer, Prentice Hall, 2002.

Structured Analysis (cont'd)

- Rules for Data Flow Diagramming (cont'd):
  - Data Flow:
    - A data flow has only one direction of flows between symbols. This is called a net flow: Example: a read before an update will show one arrow for the update only.
    - A fork in a data flow (not shown here) means a copy of the data is going to more than one location.
    - A join in a data flow (not shown) means data is being received from more than one process, data store and/or data sink/source.
    - A data flow cannot loop back to itself. If it does need to loop back, it must flow through a process.
    - A data flow to a data store means an update (delete or change).
    - A data flow from a data store implies a read.
    - A data flow has a noun-phrase label.

Adapted from Modern Systems Analysis and Design, 3rd edition, Hoffer, Prentice Hall, 2002.

Structured Analysis (con't)

- Sample mini-spec for Choose Days and Times:
  - Begin:
    - Present a list of available days and times. Order the list in ascending order by day, then is ascending order by time.
    - Ask the user to select the desired day and time.
    - Update the offered course file.
  - End

Additional information:

Valid days are Monday through Saturday.

Valid times are 8:00 AM to 6 PM, in 3 hour increments. Example: 8:00, 11:00, etc.

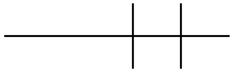
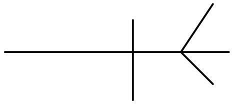
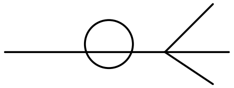
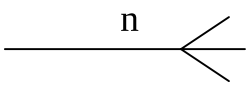
Intermezzo # 2

- Looking at diagram 1, there is an implication that these steps are done in this order. This is an example of functional decomposition.
  - Is this presumed order realistic from an end-users perspective? For example, would it be possible to assign a room, then go back and change a day and time? How would you handle this in in a data flow diagram? Perhaps another process is missing: Validate Course?
  - In use case driven model, which is goal oriented from the actor's perspective, this is not an issue. Why? Because there would be only one use case: Schedule Courses, and it would handle the validation, basic, and alternative flows in one neat package – the use case!

- If I were to write a use case for Enroll Student or Schedule Course, I might have a precondition like "Actor is authenticated". Whoops, is authentication missing from both models? Note how the concept of thinking about a precondition is a use case quickly exposes flaws in the model!
  - From an end-user perspective, which approach might you prefer: a use case driven approach or a structured analysis approach? How about from an analyst's perspective?

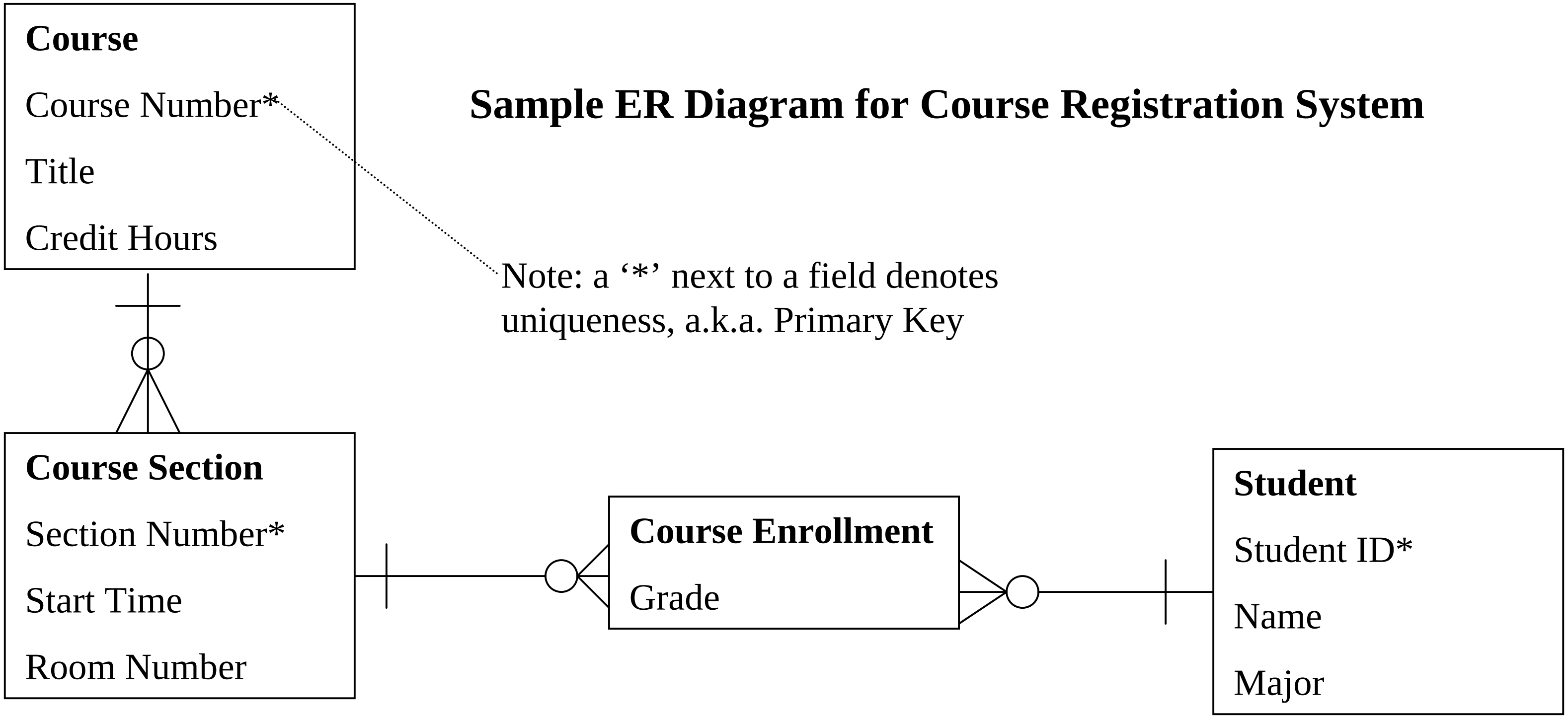
Structured Analysis (con’t)

- An entity-relationship (ER) diagram, at the analysis level is much like a domain model, except:
  - An ER diagram is on database entities, a Domain Model is based on abstractions (conceptual classes).
  - The notation is slightly different:

ER Symbol	UML Notation	Meaning
	1	1:1
	1..*	1 to many
	0..*	0 to many
	1..n	1 to some maximum, example: 1..40

Note: There is no counterpart to UML for ER diagrams, just accepted convention

Structured Analysis (con’t)



Sample ER Diagram for Course Registration System

Course

Course Number*

Title

Credit Hours

Note: a ‘*’ next to a field denotes uniqueness, a.k.a. Primary Key

Course Section

Section Number*

Start Time

Room Number

Course Enrollment

Grade

Student

Student ID*

Name

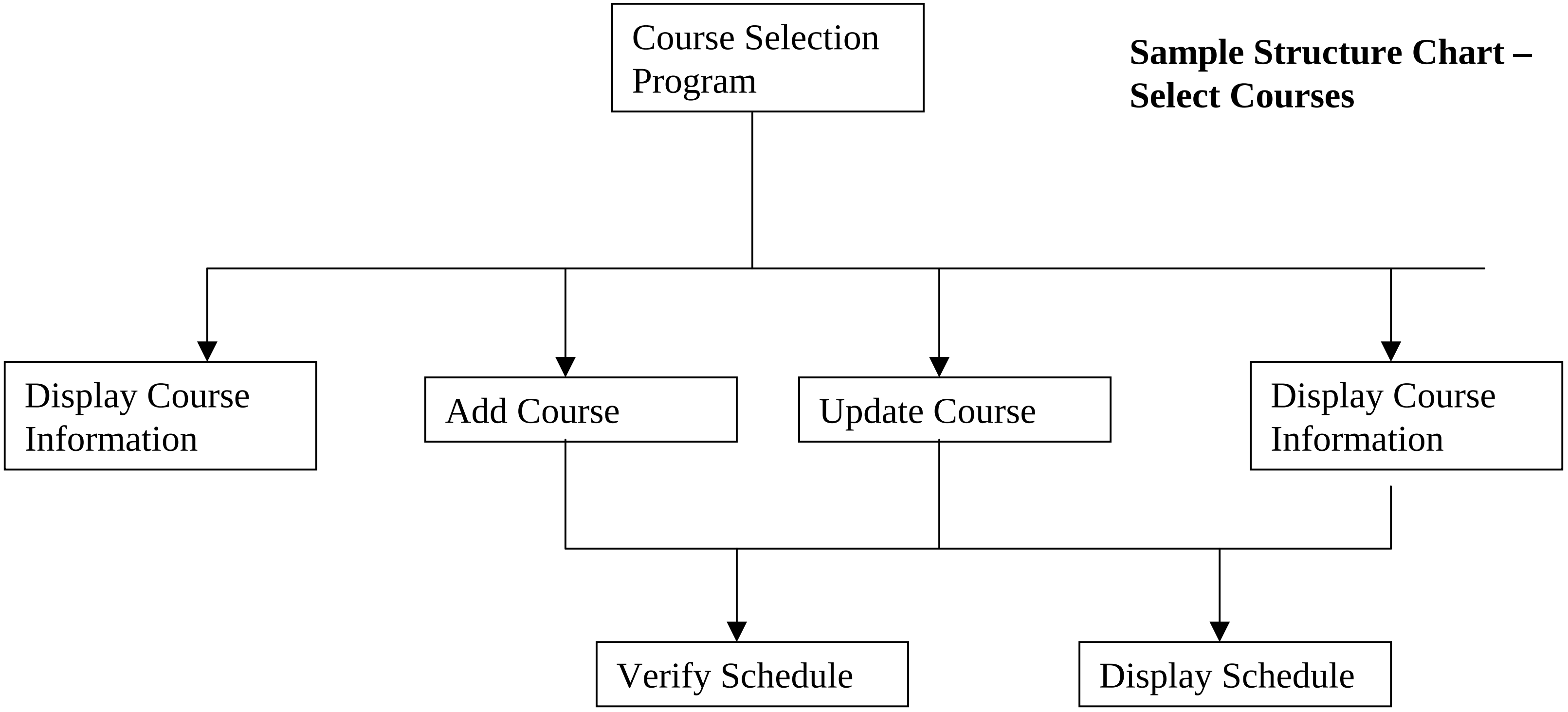
Major

Source: “Systems Analysis and Design in a Changing World”, Satzinger, Course Technology, 2002

Structured Design

- ***Fact: It is not possible to design a system without knowing something about how it will be implemented.*** Why? Because design is a blueprint for implementation.
  - Structured Design views the word as a collection of modules with functions, that share data with other (sub) modules. Example: structure chart (shown later)
  - OO Design views the world as a collection of cooperating objects sending messages to one another. Examples: class diagram, sequence diagram.
  - Structured Design, like OO Design is also based on design heuristics, such as coupling, cohesion, encapsulation, modularity, etc.,

Structured Design (con't)



Sample Structure Chart – Select Courses

Note: data and control flow not shown – for simplicity sake

Concluding Remarks
(a personal perspective)

- Use Cases and Non-Functional specifications are a preferred way to capture requirements over a monolithic requirements document.

- Use Case Modeling an entities and Domain Modeling are preferred to Structured Analysis because it focus on user’s goals and abstractions, not data entities and functional decomposition.

- OO Design makes more sense for OO languages.

- Systems development using a use case driven, architecture centric, and iterative development is (can be) more effective than waterfall methods based on structured techniques.

Concluding Remarks – (con't) (a personal perspective)

- Despite the advantages of OO and iterative development, they are not a panacea.
  - Remember: a fool with a tool is still a fool with a tool!
  - ***And of course, the three most important ingredients to a successful software project are ??????***

You've learned a lot this semester – Congratulations !!!!