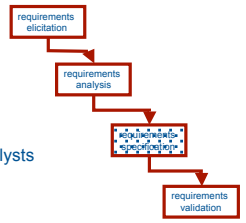
07 Requirements

- Readings
  - [cK99] Cris Kobryn, Co-Chair, "Introduction to UML: Structural and Use Case Modeling," UML Revision Task Force Object Modeling with OMG UML Tutorial Series © 1999-2001 OMG and Contributors: Crossmeta, EDS, IBM, Enea Data, Hewlett-Packard, IntelliCorp, Kabira Technologies, Klasse Objecten, Rational Software, Telelogic, Unisys http://www.omg.org/technology/uml/uml_tutorial.htm
  - [OSBB99] Gunnar Övergaard, Bran Selic, Conrad Bock and Morgan Björkande, "Behavioral Modeling," UML Revision Task Force, Object Modeling with OMG UML Tutorial Series © 1999-2001 OMG and Contributors: Crossmeta, EDS, IBM, Enea Data, Hewlett-Packard, IntelliCorp, Kabira Technologies, Klasse Objecten, Rational Software, Telelogic, Unisys http://www.omg.org/technology/uml/uml_tutorial.htm
  - [laM01] Maciaszek, L.A. (2001): Requirements Analysis and System Design. Developing Information Systems with UML, Addison Wesley Copyright © 2000 by Addison Wesley
  - [cB04] Bock, Conrad, Advanced Analysis and Design with UML http://www.kabira.com/bock/
  - [rM02] Miller, Randy, "Practical UML: A hands-on introduction for developers," Copyright © 2002 TogetherSoft, Inc. [now at Borland site] http://bdn.borland.com/article/0,1410,31863,00.html

SW Requirements Specification

- How do we communicate the Requirements to others?
  - It is common practice to capture them in an SRS
    - But an SRS doesn't need to be a single paper document
  - Purpose
  - Contractual
  - Baseline
    - for evaluating subsequent products
    - for change control
  - Audience
    - Users, Purchasers
    - Systems Analysts, Requirements Analysts
    - Developers, Programmers
    - Testers
    - Project Managers



What about RFP/RFB/RFIs?

- RFP = 'SRS' written by the procurer (or by an independent RE contractor)
  - Selected developer may create a more detailed 'SRS'
    - developer's understanding of the customers needs
    - basis for evaluation of contractual performance
    - Not typically response to RFP
  - IEEE Standard recommends SRS jointly developed by procurer and developer
- When to issue RFP/RFB?
  - Early (conceptual stage)
  - Late (detailed specification stage)

Appropriate Specification

- Consider two different projects:
  - Tiny project, 1 programmer, 2 months work
    - Programmer talks to customer, then writes up a 5-page memo
  - Large project, 50 programmers, 2 years work
    - Team of analysts model the requirements, then document them in a 500-page SRS

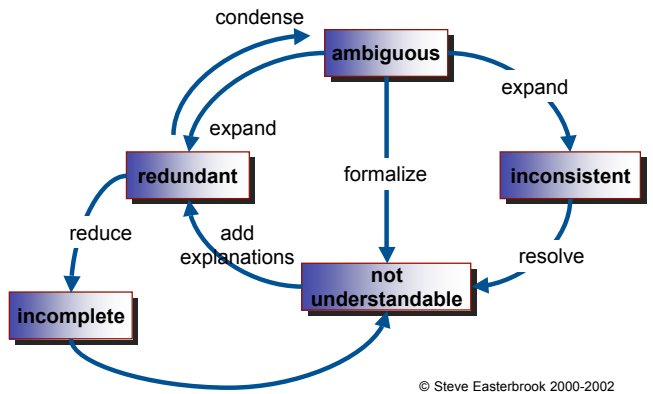
	Project A	Project B
Purpose of spec?	Crystalizes programmer's understanding; feedback to customer	Build-to document; must contain enough detail for all the programmers
Management view?	Spec is irrelevant; have already allocated resources	Will use the spec to estimate resource needs and plan the development
Readers?	Primary: Spec author; Secondary: Customer	Primary: programmers, testers, managers; Secondary: customers

Hints & Guidelines

- Validity (or "correctness")
  - expresses only the real needs of the stakeholders (customers, users,…)
- Completeness
  - specifies all the things the system must do and all the things it must not do!
  - Conceptual Completeness
    - e.g. responses to all classes of input
  - Structural Completeness
    - e.g. no "to be determined" functions
- Consistency
  - not self-contradictory
  - satisfiable
  - all terms used consistently
  - inconsistency can be hard to detect especially in timing aspects and business logic
- Necessary
  - doesn't contain anything that isn't "required"
- Ambiquity
  - every statement can be read in exactly one way
  - defines confusing terms
    - e.g. in a glossary
- Verifiablility
  - includes a process exists to test satisfaction of each requirement
  - every requirement is specified behaviorally
- Understandablility (Clarity)
  - e.g. by non-computer specialists
  - technical notations should only be used as backup (e.g. in an appendix)
- Modifiability
  - easy to change and modify
  - good structure and cross-referencing
  - must be kept up to date!

see *IEEE-STD-830-1993*

There is no Perfect SRS



Typical mistakes

- Noise
  - text that carries no relevant information to any feature of the problem.
- Silence
  - a feature that is not covered by any text.
- Over-specification
  - text that describes a feature of the solution, rather than the problem.
- Contradiction
  - text that defines a single feature in a number of incompatible ways.
- Ambiguity
  - text that can be interpreted in at least two different ways.
- Forward reference
  - text that refers to a terms or features yet to be defined.
- Wishful thinking
  - text that defines a feature that cannot possibly be validated.
- Jigsaw puzzles
  - distributing key information across a document and then cross-referencing
- Duckspeak requirements
  - requirements that are only there to conform to standards
- Unnecessary invention of terminology
  - e.g. 'user input presentation function'
  - e.g. 'airplane reservation data validation function'
- Inconsistent terminology
  - inventing and then changing terminology
- Putting the onus on the development staff
  - i.e. making the reader work hard to decipher the intent
- Writing for the hostile reader
  - fewer of these than friendly readers

from Steve Easterbrook © 2000-2002; he *adapted from Kovitz, 1999*

IEEE std offers different templates

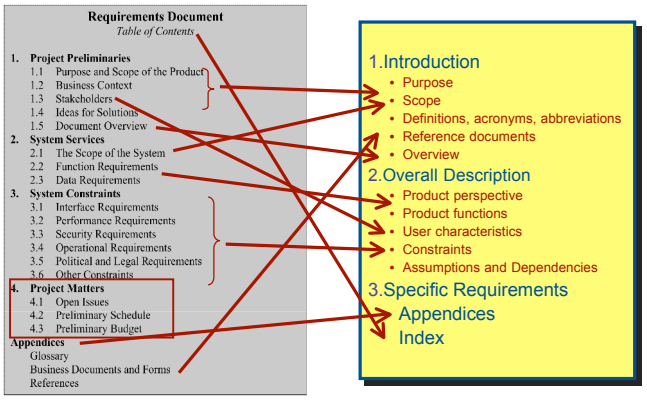
1. Introduction
   - Purpose
   - Scope
   - Definitions, acronyms, abbreviations
   - Reference documents
   - Overview
2. Overall Description
   - Product perspective
   - Product functions
   - User characteristics
   - Constraints
   - Assumptions and Dependencies
3. Specific Requirements

   Appendices

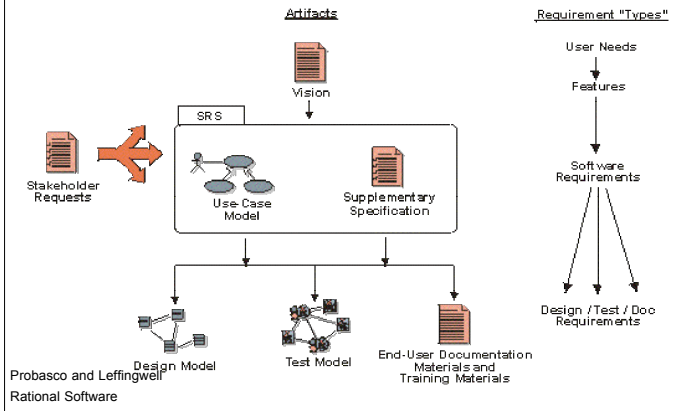
   Index

- External stimulus or external situation
  - e.g., for an aircraft landing system, each different type of landing situation: wind gusts, no fuel, short runway, etc
- System feature
  - e.g., for a telephone system: call forwarding, call blocking, conference call, etc
- System response
  - e.g., for a payroll system: generate pay-cheques, report costs, print tax info;
- External object
  - e.g. for a library information system, organize by book type
- User type
  - e.g. for a project support system: manager, technical staff, administrator, etc.
- Mode
  - e.g. for word processor: page layout mode, outline mode, text editing mode, etc
- Object
  - e.g., in a patient monitorng system, objects include patients, sensors, nurses, rooms, physicians, medicines, etc.

Maciaszek vs. IEEE



SRS using a UML "package" construct



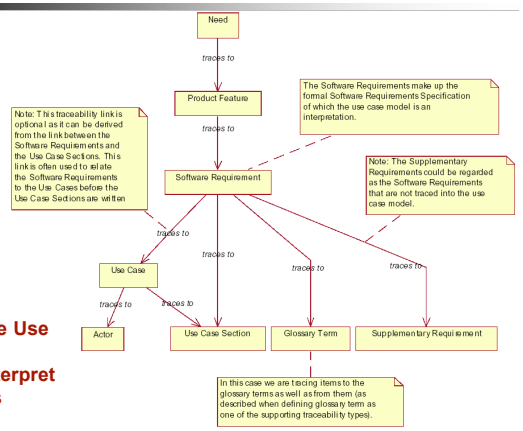
SRS using a UML "package" construct

- SRS
  - may include a single document, multiple documents, use case specifications and even the graphical use case model which describes relationships amongst the use cases.
- Stakeholders
  - system analyst
  - use-case specifier
  - designers
  - implementers
  - project manager
  - testers

"Combining Software Requirements Specifications with Use-Case Modeling," Leslee Probasco and Dean Leffingwell

www.spc.ca/downloads/srs_usecase.doc

UML & SRS



- **Features Drive Use Case Models**
- **Use Cases Interpret Requirements**

Example

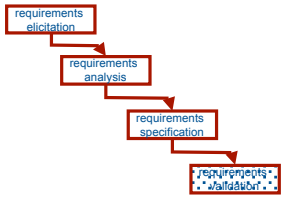
- openCMSstruts
  - http://opencmsstruts.sourceforge.net/vision.html
- We'll come back to UML & RUP

SRS format and style

- Modifiability
  - well-structured, indexed, cross-referenced, etc.
  - redundancy should be avoided or must be clearly marked as such
  - An SRS is not modifiable if it is not traceable...
- Traceability
  - Need a way of referring to each requirement
  - Backwards - the specification must be "traced"
    - each requirement traces back to a source or authority
    - e.g. a requirement in the system spec; a stakeholder; etc
  - Forward - the specification must be "traceable"
    - each requirement will eventually trace forwards to parts of the design that satisfy it

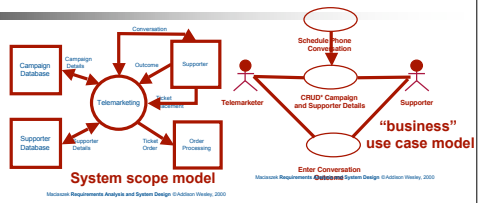
Requirements Negotiation & Validation

- Negotiation
  - Based on draft of document
- Validation
  - Based on (almost) complete document
- Issues
  - Scope
  - Dependencies
  - Risks
  - Priorities



Issues

- Scope
- Dependencies
- Risks
  - Technical
  - Performance
  - Database integrity
  - Development process
  - Political
  - Legal
  - Volatility



System scope model

Requirement	R1	R2	R3	R4
R1	X	X	X	X
R2	Conflict	X	X	X
R3			X	X
R4		Overlap	Overlap	X

Requirements dependency matrix

COMPUTER SCIENCE

Seque to a quick UML overview

COMPUTER SCIENCE

But first, let's discuss Project 1

- Goal - Develop a Software Requirements Specification for a Course Management System "tool" to be designed, developed and incorporated in the Sakai framework using the RUP template or the IEEE Std. or the OpenCms-Struts "template"
  - Vision Document
  - SRS
    - Introduction Section 1: Purpose, Scope, Definitions, Acronyms and Abbreviations, and provide References
    - Overall Description Section 2: a list of names and brief descriptions of all use cases and actors, along with applicable diagrams and relationships
    - In Section 3, for each use case diagram in Section 2 define a use-case report, making sure that each feature or requirement is clearly labeled and traceable to the Vision document.
    - Appendices, including: a) Table of contents, b) Index, and c) use-case storyboards or user-interface prototypes, if needed.
- Process
  - Identify Stakeholders -- October 7
  - Develop questionnaires -- October 7
  - Plan and carry out interviews -- by October 28
  - Define Vision Document
  - Define Use-Cases
  - Complete the SRS

  } -- November 4

COMPUTER SCIENCE

How shall we do this?

- Pick the CMS "tool"
  - Common or different for each group?
  - Project 1 prelim: each group email me the selected tool **and get approval**
- Develop questionnaires
  - Stakeholders to be interviewed
    - Department staff & associate chair
    - OIT
    - Registrar
    - CCBIT
    - Faculty
    - Yourselves
  - Sample "portal" questions/interview outline
- Plan and carry out interviews
  - 30 minutes/stakeholder-group
  - In class or another scheduled time -- last weeks of October?
  - **Videotape!**
- For selected "tool," define Vision Document
  - Assign one member of the group for this?
- For selected "tool," define Use-Cases
  - Assign rest of the group for this?
- Complete the SRS

COMPUTER SCIENCE

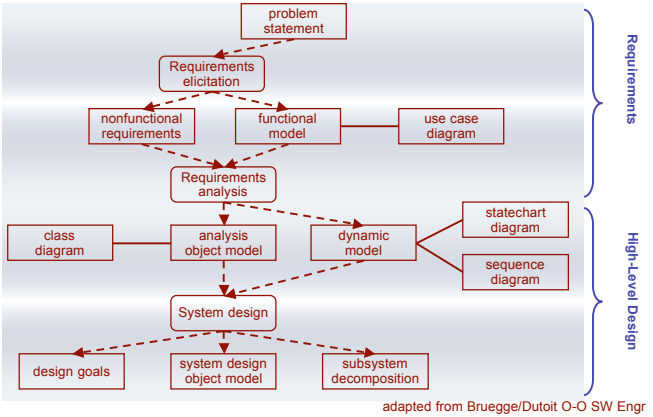
Note - UML overheads are adapted from

- "Introduction to UML: Structural and Use Case Modeling," Cris Kobryn, Co-Chair UML Revision Task Force Object Modeling with OMG UML Tutorial Series © 1999-2001 OMG and Contributors: Crossmeta, EDS, IBM, Enea Data, Hewlett-Packard, IntelliCorp, Kabira Technologies, Klasse Objecten, Rational Software, Telelogic, Unisys
- "Behavioral Modeling," Gunnar Övergaard, Bran Selic, Conrad Bock and Morgan Björkande, UML Revision Task Force, Object Modeling with OMG UML Tutorial Series © 1999-2001 OMG and Contributors: Crossmeta, EDS, IBM, Enea Data, Hewlett-Packard, IntelliCorp, Kabira Technologies, Klasse Objecten, Rational Software, Telelogic, Unisys
- MACIASZEK, L.A. (2001): Requirements Analysis and System Design. Developing Information Systems with UML, Addison Wesley Copyright © 2000 by Addison Wesley
- "Analysis and Design with UML," Rational Copyright © 1997 by Rational Software Corporation
- "Practical UML: A hands-on introduction for developers," Copyright © 2002 TogetherSoft, Inc.

O-O problem solving

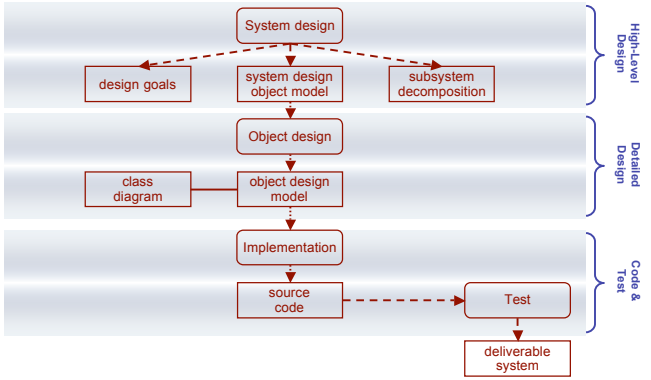
- underlying tenet begins with the construction of a model
  - a **model** is an abstraction of the underlying problem
  - the **domain** is the **actual world** from which the problem comes
- **models** consist of **objects** that interact by sending each other **messages**
  - objects have things they know (**attributes**) and things they can do (**behaviors** or **operations**)
  - values of an object's attributes determine its **state**
- **classes** are the "blueprints" for objects
  - a class wraps attributes (data) and behaviors (methods or functions) into a single distinct entity
- objects are **instances** of classes.

O-O System Development



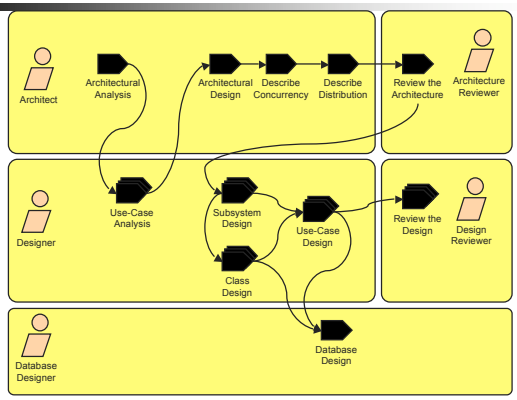
adapted from Bruegge/Dutoit O-O SW Engr

O-O System Development

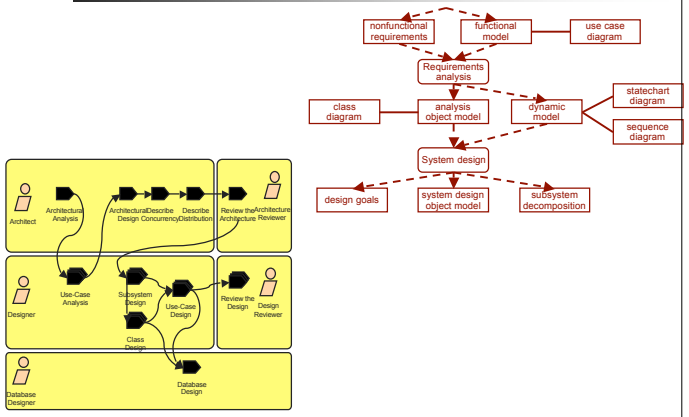


adapted from Bruegge/Dutoit O-O SW Engr

Rational Unified Process



Rational Unified Process



UML Overview

The UML is a graphical language for { specifying, visualizing, constructing, documenting } the artifacts of software systems

UML Goals

- an easy-to-learn but semantically rich visual modeling language
- unify the Booch, OMT, and Objectory modeling languages
- ideas from other modeling languages + industry best practices
- enable model interchange and define repository interfaces
- provide a common vocabulary to talk about object-oriented software design.

Unifying Concepts in UML

- classifier-instance dichotomy
  - e.g., an object is an instance of a class OR a class is the classifier of an object
- specification-realization dichotomy
  - e.g., an interface is a specification of a class OR a class is a realization of an interface
- building blocks:
  - model elements (classes, interfaces, components, use cases, etc.)
  - relationships (associations, generalization, dependencies, etc.)
  - diagrams (class diagrams, use case diagrams, interaction diagrams, etc.)

Well-formedness rules

- Well-formed: indicates that a model or model fragment adheres to all semantic and syntactic rules that apply to it.
- UML specifies rules for: naming, scoping, visibility, integrity & execution (limited)
- However, during iterative, incremental development it is expected that models will be incomplete and inconsistent.

What is use case modeling?

- use case model
  - a view of a system that emphasizes the behavior as it appears to outside users. A use case model partitions system functionality into transactions ('use cases') that are meaningful to users ('actors')
- example: Make Appointment
  - use case for the medical clinic
    - actor is a Patient
    - connection between actor and use case is a communication association (or communication for short)

Use-Case diagrams

- emphasis is on *what* a system does rather than *how*
- Use case diagrams are closely connected to scenarios
  - a **scenario** is an example of what happens when someone interacts with the system, e.g.,
    "A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot."
  - a **use case** is a summary of scenarios for a single task or goal
  - an **actor** is who or what initiates the events involved in that task

Use Case Modeling: Core Elements

Construct	Description	Syntax
use case	A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.	
actor	A coherent set of roles that users of use cases play when interacting with these use cases.	
system boundary	Represents the boundary between the physical system and the actors who interact with the physical system.	

Use Case Modeling: Core Relationships

Construct	Description	Syntax
association	The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.	
generalization	A taxonomic relationship between a more general use case and a more specific use case.	
extend	A relationship from an *extension* use case to a *base* use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.	

The ESU University

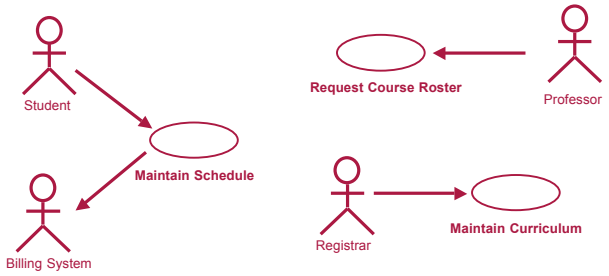
- wants to computerize their registration system
- Registrar sets up the curriculum for a semester
  - One course may have multiple course offerings
- students select 4 primary courses and 2 alternate courses
- once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- students may use the system to add/drop courses for a period of time after registration
- professors use the system to receive their course offering rosters
- users of the registration system are assigned passwords which are used at logon validation

Use Cases

- A use case is a pattern of behavior the system exhibits
  - Each use case is a sequence of related transactions performed by an actor and the system in a dialogue
- Actors are examined to determine their needs
  - Registrar
  - Professor
  - Student
  - Billing System

Use Case Diagram

- Use case diagrams are created to visualize the relationships between actors and use cases



Documenting Use Cases

- A flow of events document is created for each use cases
  - Written from an actor point of view
- Details what the system must provide to the actor when the use cases is executed
- Typical contents
  - How the use case starts and ends
  - Normal flow of events
  - Alternate flow of events
  - Exceptional flow of events

Maintain Curriculum Flow of Events

- This use case begins when the Registrar logs onto the Registration System and enters his/her password. The system verifies that the password is valid (E-1) and prompts the Registrar to select the current semester or a future semester (E-2). The Registrar enters the desired semester. The system prompts the Registrar to select the desired activity: ADD, DELETE, REVIEW, or QUIT.
  - If the activity selected is ADD, the S-1: Add a Course subflow is performed.
  - If the activity selected is DELETE, the S-2: Delete a Course subflow is performed.
  - If the activity selected is REVIEW, the S-3: Review Curriculum subflow is performed.
  - If the activity selected is QUIT, the use case ends.
- ...

Documenting use cases

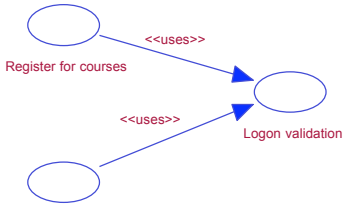
- **Brief Description**
- **Actors** involved
- **Preconditions** necessary for the use case to start
- **Detailed Description** of flow of events that includes:
  - **Main Flow** of events, that can be broken down to show:
    - **Subflows** of events (subflows can be further divided into smaller subflows to improve document readability)
  - **Alternative Flows** to define exceptional situations
- **Postconditions** that define the state of the system after the use case ends

Narrative use case specification

Use Case	Add a course to the curriculum
Brief Description	This use case allows a Registrar to enter a new course. …
Actors	`Registrar`
Preconditions	Registrar has a valid password (E-1), has selected a semester default or E-2), and has selected the `Add` (S-1) function at the system prompt
Main Flow	The system enters the `Add a Course` subflow
Alternative Flows	The `Registrar` activates the `Delete, Review,` or `Quit` functions
Postconditions	If the use case was successful, the Registrar has accessed the `Add a Course` function

Uses and Extends Relationships

- As the use cases are documented, other use case relationships may be discovered
  - A **uses** relationship shows behavior that is common to one or more use cases
  - An **extends** relationship shows optional behavior



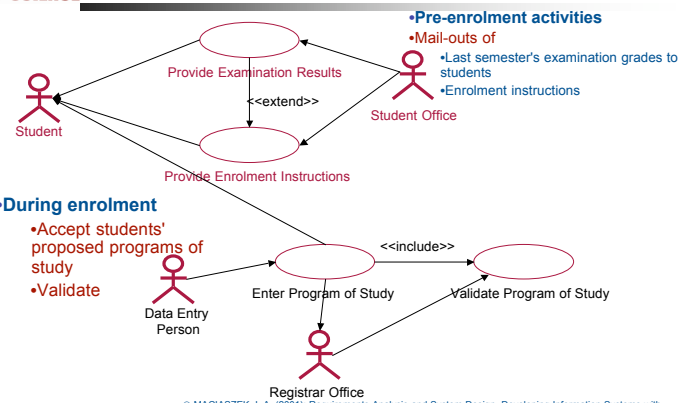
University Enrollment

- The **university** offers
  - Undergraduate and postgraduate degrees
  - To full-time and part-time students
- The **university structure**
  - Divisions containing departments
  - Single division administers each degree
  - Degree may include courses from other divisions
- **University enrolment** system
  - Individually tailored programs of study
  - Prerequisite courses
  - Compulsory courses
  - Restrictions
    - Timetable clashes
    - Maximum class sizes, etc.

University Enrolment (cont)

- The system is required to
  - Assist in pre-enrolment activities
  - Handle the enrolment procedures
- **Pre-enrolment activities**
  - Mail-outs of
    - Last semester's examination grades to students
    - Enrolment instructions
- **During enrolment**
  - Accept students' proposed programs of study
  - Validate for prerequisites, timetable clashes, class sizes, special approvals, etc.
- Resolutions to some of the problems may require consultation with academic advisers or academics in charge of course offerings

Example 4.12



When to model use cases

- Model user requirements with use cases.
- Model test scenarios with use cases.
- If you are using a use-case driven method
  - start with use cases and derive your structural and behavioral models from it.
- If you are not using a use-case driven method
  - make sure that your use cases are consistent with your structural and behavioral models.

Use Case Modeling Tips

- Make sure that each use case describes a significant chunk of system usage that is understandable by both domain experts and programmers
- When defining use cases in text, use nouns and verbs accurately and consistently to help derive objects and messages for interaction diagrams (see Lecture 2)
- Factor out common usages that are required by multiple use cases
  - If the usage is required use <<include>>
  - If the base use case is complete and the usage may be optional, consider use <<extend>>
- A use case diagram should
  - contain only use cases at the same level of abstraction
  - include only actors who are required
- Large numbers of use cases should be organized into packages

Use Case Realizations

- The use case diagram presents an outside view of the system
- Interaction diagrams describe how use cases are realized as interactions among societies of objects
- Two types of interaction diagrams
  - Sequence diagrams
  - Collaboration diagrams

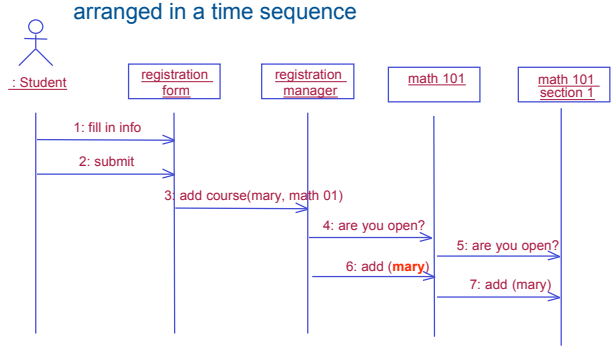
sequence diagram

- an interaction diagram that details how operations are carried out
  - what messages are sent and when
  - are organized according to time
  - time progresses as you go down the page
  - objects involved in the operation are listed from left to right according to when they take part in the message sequence.

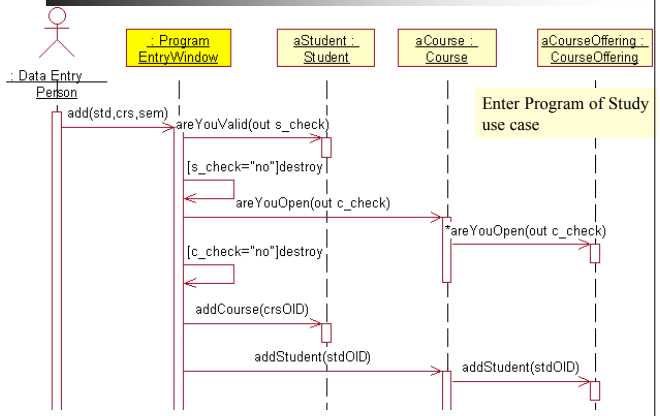
Symbol	Meaning
→	simple message which may be synchronous or asynchronous
- - →	simple message return (optional)
▶	a synchronous message
↗ ↘	an asynchronous message

Sequence Diagram

- A sequence diagram displays object interactions arranged in a time sequence



Example 4.17 – Maciaszek



Collaboration Diagrams

- also interaction diagrams
- focus on object roles instead of the times that messages are sent