17-Design

- Readings
  - OOAD Using the UML

    - will post …

Jackson System Development (JSD)

- Phases
  - the modeling phase
    - Entity/action step
    - Entity structure step
    - Model process step
  - the network phase
    - connect model processes and functions in a single system specification diagram (SSD)
  - implementation phase
    - examine the timing constraints of the system
    - consider possible hardware and software for implementing our system
    - design a system implementation diagram (SID)

Student Loan Example

- Functional requirements:
  - before getting a loan, there is an evaluation process after which agreement is always reached
    - a **TE transaction** records each step of the evaluation process
    - a **TA transaction** records the overall loan agreement
  - a student can take any number of loans, but only one can be active at any time
    - each loan is initiated by a **TI transaction**
  - the student repays the loan with a series of repayment
    - each repayment transaction is recorded by a **TR transaction**
  - a loan is terminated by a **TT transaction**
  - two output functions are desired:
    - an inquiry function that prints out the loan balance for any student,
    - a repayment acknowledgment sent to each student after payment is received by the university
- Non Functional requirements
  - to be implemented on a single processor
  - inquiries should be processed as soon as they are received
  - repayment acknowledgments need only be processed at the end of each day.
  - Note: generates a stream of data over a long-period of time

Step 1: Entity/action step

- Actions have the following characteristics:
  - an action takes place at a point in time
  - an action must take place in the real world outside of the system.
  - an action is atomic, cannot be divided into subactions.
- Entities have the following characteristics:
  - an entity performs or suffers actions in time.
  - an entity must exist in the real world, and not be a construct of a system that models the real world
  - an entity must be capable of being regarded as an individual; and, if there are many entities of the same type, of being uniquely named.

Candidates

- Entities/Description:
  - student
  - system
  - university
  - loan
  - student-loan

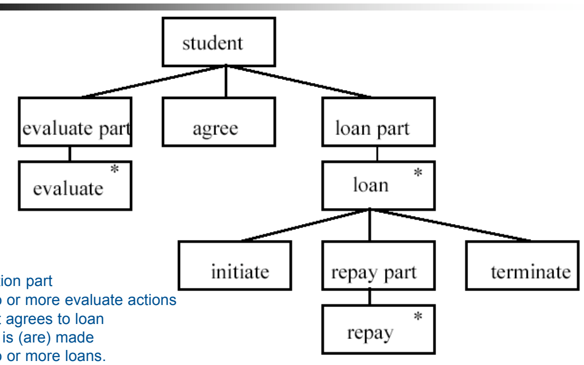
Actions/Attributes:

- **evaluate** -action of university? (university performs the evaluation); action of student? (student is evaluated)
  - attributes: student-id, loan-no, date of evaluation, remarks
- **agree** - action of university? (university agrees to loan); action of student ? (agrees to loan)
  - attributes: student-id, loan-no, date of agreement, amount of loan, interest rate, repayment period)
- **make loan** - action of university
  - attributes: student-id, loan-no, date of loan, loan amount, interest rate, repayment period
- **initiate** - action of university? (university initiates loan); action of student? (student initiates loan); action of loan? (is initiated)
  - attributes: student-id, date initiated
- **repay** - action of loan? (loan is repaid); action of student? (student repays the loan);
  - attributes: student-id, date of repayment, amount of repayment
- **terminate** - action of loan (loan is terminated);
  - attributes: student-id, date of termination, remarks

Focus on:

- Entities/Description:
  - student
- Actions/Attributes:
  - evaluate -action of student; student? (student suffers the action, is evaluated);
    - attributes: student-id, loan-no, date of evaluation, remarks
  - agree - action of student
    - attributes: student-id, loan-no, date of agreement, amount of loan, interest rate, repayment period)
  - initiate - action of student
    - attributes: student-id, date initiated
  - repay - action of student
    - attributes: student-id, date of repayment, amount of repayment
  - terminate - action of student
    - attributes: student-id, date of termination, remarks

Step 2: Entity structure step



(1) evaluation part
- zero or more evaluate actions

(2) student agrees to loan

(3) loan(s) is (are) made
- zero or more loans.
- loan is a sequence of initiate action, iteration of repay actions, a terminate action

Model Process

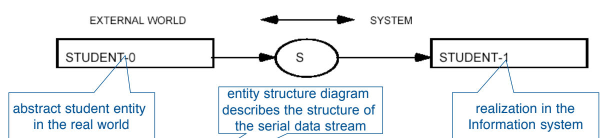
- Primary building block of a JSD design
  - contains all actions characterizing a key real-world process
- Actions are structured into a tree
  - only the leaf nodes of the tree are real-world actions
  - interior nodes are conceptual
  - interior nodes can be annotated to show choice or iteration
  - traversals of this tree constitute the only "legal" sequences of actions for this process
- Model process tree defines a regular expression
  - set of traversals is a regular set

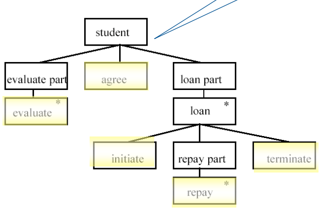
Model Processes

- A model process is a particular view of the system
  - various model processes provide different views
  - model process is multiply instantiated for different instances
  - model processes are often annotated with informal specifications and notations
  - same action may appear as part of more than one process
- Model Processes and Data
  - actions on data hang off of model process leaf nodes
  - global data is necessary too
    - for functions that must combine data from >1 model process
    - to assure consistency between model processes
    - to coordinate between different instances of the same model process
    - to coordinate between different models of the same entity

UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2003

Step 3: Model process step






```
STUDENT-1 seq
   read S;
   EVAL   iter (while TE)
      process TE; read S
   EVAL  end
   AGREE  seq
      process TA; read S
   AGREE  end
   LOAN-PART iter (forever)
      INIT   seq
        process TI; read S
      INIT
      REPAY  iter (while TR)
         process TR; read S
      REPAY  end
      TERM   seq
        process TT; read S
      TERM  end
   LOAN-PART end
STUDENT-1 end
```

Use JSP to create a program for the process

- a **TE transaction** records each step of the evaluation process
- a **TA transaction** records the overall loan agreement
- each loan is initiated by a **TI transaction**
- each repayment transaction is recorded by a **TR transaction**
- a loan is terminated by a **TT transaction**

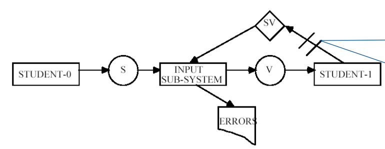
UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2003

Error handling

- a real-time system (but slow-running) system
- information is collected as it arrives from the real-world
- entity model process is synchronized with the actions of the real world entity
- the state vector of a model process's "program" has a "counter" … and if it "points" to repay component of a student's process, then an 'E' (evaluate), 'A' (agree) or 'I' (initiate) transaction **must be recognized as an error**

```
STUDENT-1 seq
   read S;
   EVAL   iter (while TE)
      process TE; read S
   EVAL  end
   AGREE  seq
      process TA; read S
   AGREE  end
   LOAN-PART iter (forever)
      INIT   seq
        process TI; read S
      INIT
      REPAY  iter (while TR)
         process TR; read S
STU
```

counter



state vector (SV) connection
-- one process can examine the SV of a 2nd process
-- the double lines indicate that an inquiry process, over its life, will examine many student processes

UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2003

Total System Model

- At the Network Phase, weave Model Processes together incrementally to form the total system specification
  - also add new processes during this phase: e.g., input, output, user interface, data collection
- Goal is to indicate how model processes communicate with each other, use each other, are connected to user and outside world
- Linkage through two types of communication:
  - Message passing
  - State vector inspection
- Indicates which data moves between which processes
  - and more about synchronization

Model Process Communication

- Fundamental notion is Data Streams
  - can have multiple data streams arriving at an action in a process
  - can model multiple instances entering a data stream or departing from one
- Two types of data stream communication:
  - asynchronous message passing
  - State vector inspection
- These communication mechanisms used to model how data is passed between processes

University of Massachusetts Amherst · Department of Computer Science · CMPSCI 520/620 Fall 2004

Message Passing

- Data stream carries a message from one process activity to an activity in another process
  - must correlate with output leaf of sending model process
  - must correlate with input leaf of receiving model process
- Data transfer assumed to be asynchronous
  - less restrictive assumption
  - no timing constraints are assumed
  - messages are queued in infinitely long queues
  - messages interleaved non-deterministically when multiple streams arrive at same activity

University of Massachusetts Amherst · Department of Computer Science · CMPSCI 520/620 Fall 2004

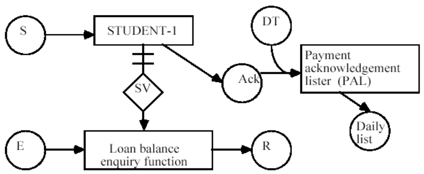
State Vector Inspection

- Modeling mechanism used when one process needs considerable information about another
- State vector includes
  - values of all internal variables
  - execution text pointer
- Process often needs to control when its state vector can be viewed
  - process may need exclusive access to its vector
- Could be modeled as message passing, but important to underscore characteristic differences

University of Massachusetts Amherst · Department of Computer Science · CMPSCI 520/620 Fall 2004

Network Phase -- the SSD

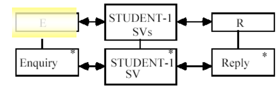
- loan balance inquiry function (LBE) is connected to the Student-1 process by state vector (SV) connection
- The function to produce the student acknowledgments data stream (ACK) is embedded in the student-1 process in the repays component



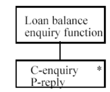
- DT is an input signal at the end of the day--a daily time marker--that tells the payment acknowledgment lister (PAL) function to begin
- The ACK and DT data streams are rough-merged, that is, we don't know precisely whether a repayment acknowledgment will appear on today's or tomorrow's daily list.

Designing the LBE function w/ JSP

(i) input and output data structures:

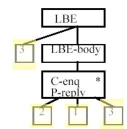


(ii) basic program structure



(iii) list of operations:
1 - write 'loan balance for', student-id, 'is', balance
2 - get STUDENT SV (student-id)
3 - read E

(iv) elaborated program structureand text:

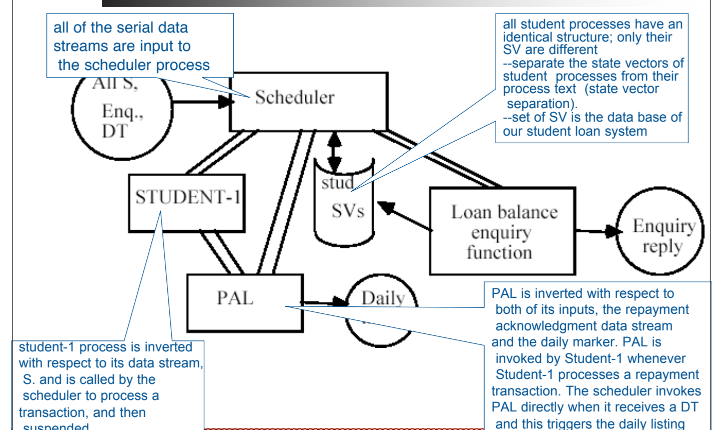



```
LBE   seq
  read E;
  LBE-BODY itr     (forever)
    get STUDENTSV(student-id);
    write 'loan balance for ', student-id,
      ' is ', balance'
    read E;
  LBE-BODY   end
LBE  end
```

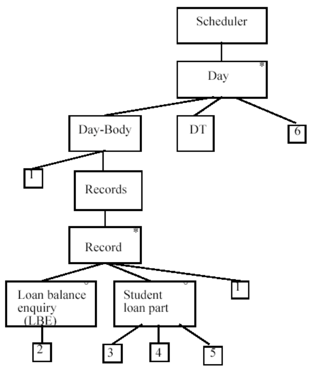

Implementation Phase

- Use of inferences encouraged by understandings gleaned from the network phase
- Network Phase suggests ideal traversal paths through model processes and their local data
  - suggests internal implementation of model processes
  - studying use of model processes suggests internal structure of their data
- Communication by data streams and state vector inspection often suggest real implementations
  - But often not

The SID



Design of the scheduler in JSP



- records from the serial data stream (loan balance inquiries and student loan transactions) are read and processed in real-time
- at the end of the day, a daily time marker--perhaps a signal to the system from the operator--is input

List of operations:

1-read input
2-call LBE(inrec)
3-get SSV(student-id)
4-call student-1(srec, ssv)
5-put SSV(student-id)
6-call PAL(DT)

- PAL is invoked & processes payment acknowledgments that have been previously generated in real-time whenever a student repayment is made and stored in a buffer

JSD and JSP

- In JSD, the principles of JSP are extended into the areas of systems analysis, specification, design and implementation
- In JSP, a simple program describes a sequential process that communicates by means of sequential data streams; its structure is determined by the structure of its input and output data stream
- In JSD, the real world is modeled as a set of sequential model processes that communicate with the real world and with each other by sequential data streams (as well as by a second read-only communication called state vector connection). The structure of a model process is determined by the structure of its inputs and outputs.
- The JSD implementation step embodies the JSP implementation technique, program inversion, in which a program is transformed into a procedure
- Other JSP techniques, such as the single read-ahead rule and backtracking, and principles, such as implementation through transformation, are used in JSD

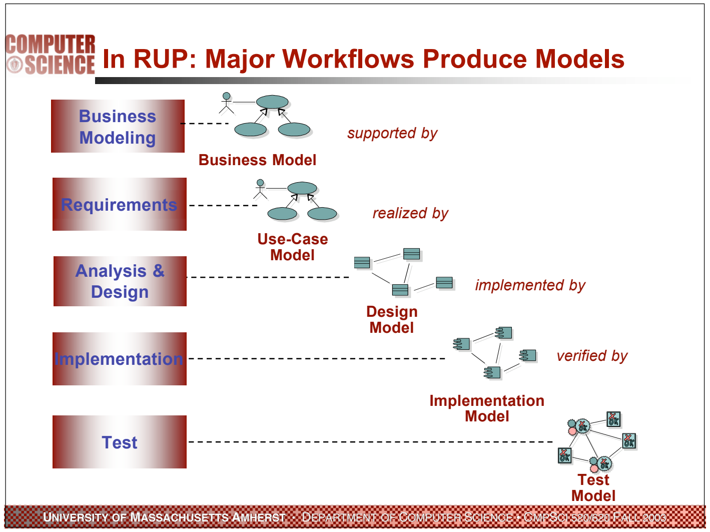
Comments/Evaluation

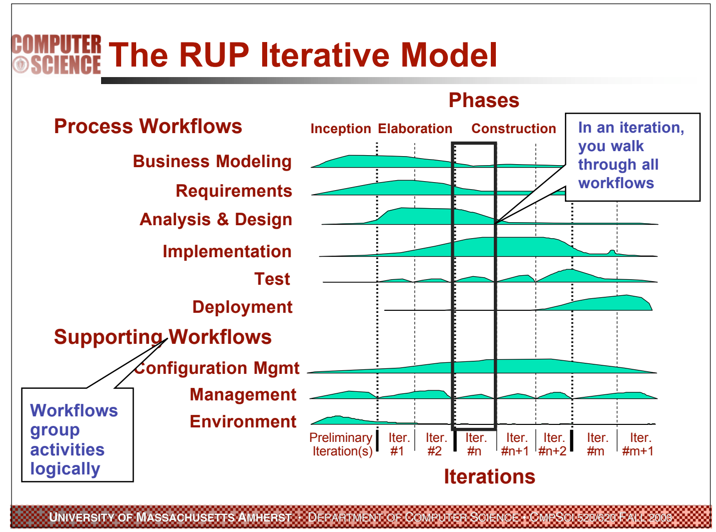
- Focus on conceptual design
  - But difficult to build a system this way
- Based upon model of real world
- Careful (and experienced) analysis of the model generally points suggested implementation tactics, though
  - Parnas notions of module not perceptible here
  - Not an iterative refinement approach either
- Treatment of data is very much subordinated/secondary
- Does a good job of suggesting possible parallelism
- Contrasts strongly with Objected Oriented notions (eg. Booch, UML)

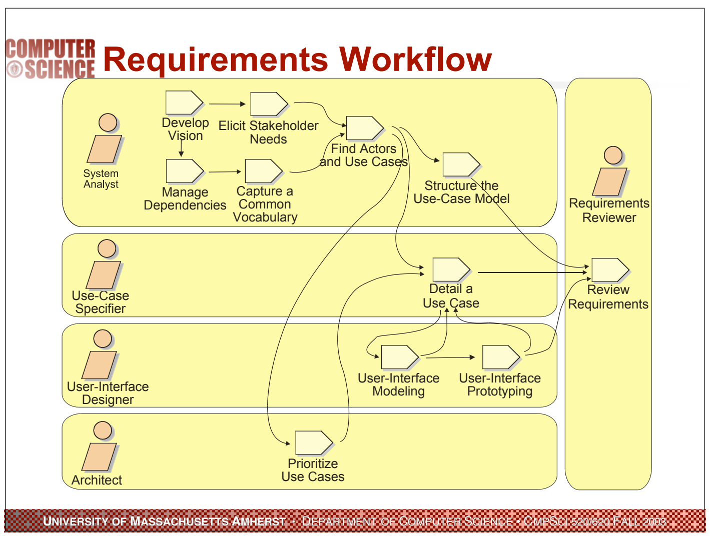
Rational Unified Process

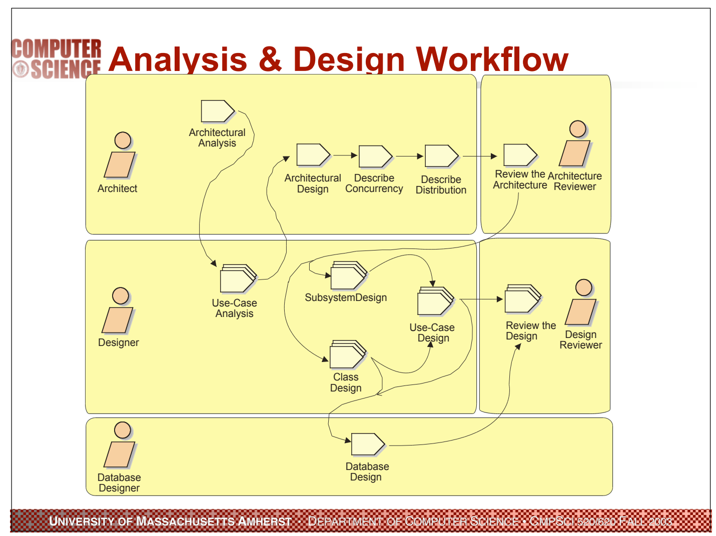
- The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of a software-intensive system
- A software development process defines Who is doing What, When and How in building a software product
- The Rational Unified Process has four phases: Inception, Elaboration, Construction and Transition
- Each phase ends at a major milestone and contains one or more iterations
- An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release

The RUP presentation is adapted from **OOAD Using the UML**

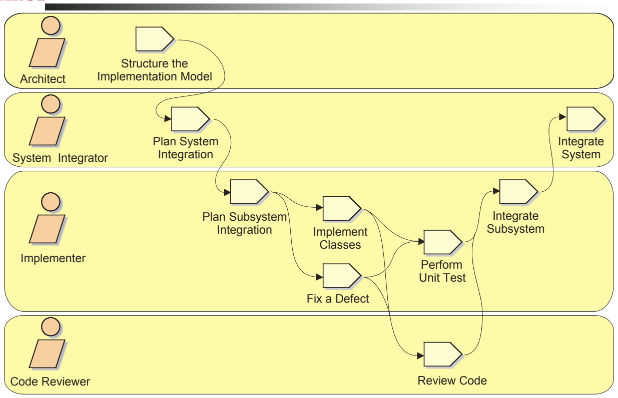

COMPUTER SCIENCE
In RUP: Major Workflows Produce Models
Business Modeling
Business Model
supported by
Requirements
Use-Case Model
realized by
Analysis & Design
Design Model
implemented by
Implementation
Implementation Model
verified by
Test
Test Model
UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004


COMPUTER SCIENCE
The RUP Iterative Model
Phases
Process Workflows
Inception
Elaboration
Construction
Transition
In an iteration, you walk through all workflows
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Supporting Workflows
Configuration Mgmt
Management
Environment
Workflows group activities logically
Preliminary Iteration(s)
Iter. #1
Iter. #2
Iter. #n
Iter. #n+1
Iter. #n+2
Iter. #m
Iter. #m+1
Iterations
UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004

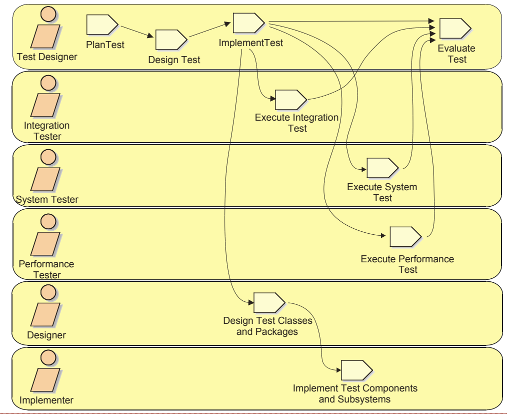

COMPUTER SCIENCE
Requirements Workflow
System Analyst
Develop Vision
Elicit Stakeholder Needs
Find Actors and Use Cases
Structure the Use-Case Model
Manage Dependencies
Capture a Common Vocabulary
Requirements Reviewer
Use-Case Specifier
Detail a Use Case
Review Requirements
User-Interface Designer
User-Interface Modeling
User-Interface Prototyping
Architect
Prioritize Use Cases
UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004


COMPUTER SCIENCE
Analysis & Design Workflow
Architect
Architectural Analysis
Architectural Design
Describe Concurrency
Describe Distribution
Review the Architecture
Architecture Reviewer
Designer
Use-Case Analysis
SubsystemDesign
Use-Case Design
Class Design
Review the Design
Design Reviewer
Database Designer
Database Design
UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004

Implementation Workflow

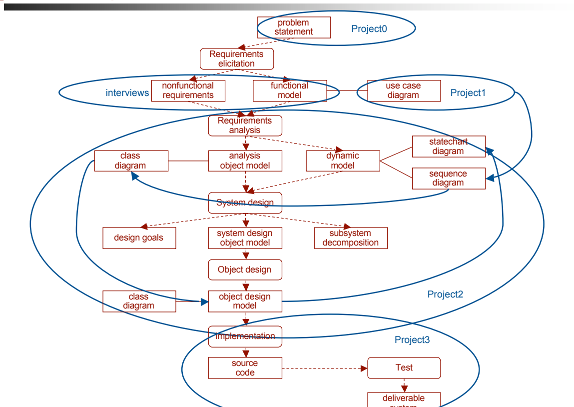


Test Workflow



UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004

O-O System Development



adapted from Bruegge/Dutoit O-O SW Engr

UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004

A Minimal Iterative Process

Getting Started: (do this once)

1. Capture the major functional and non-functional requirements for the system.
   - Express the functional requirements as use cases, scenarios, or stories.
   - Capture non-functional requirements in a standard paragraph-style document.
2. Identify the classes which are part of the domain being modeled.
3. Define the responsibilities and relationships for each class in the domain.
4. Construct the domain class diagram.
   - This diagram and the responsibility definitions lay a foundation for a common vocabulary in the project.
5. Capture use case and class definitions in an OO CASE tool (e.g., Rose) only when they have stablilized.

UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004

A Minimal Iterative Process

Getting Started: (do this once)

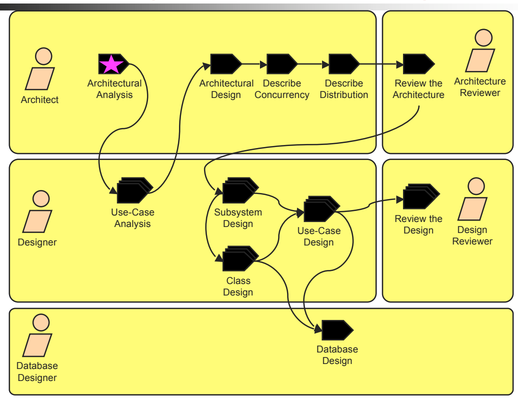
6. Identify the major risk factors and prioritize the most architecturally significant use cases and scenarios.
   - It is absolutely imperative that the highest risk items and the most architecturally significant functionality be addressed in the early iterations. You must not pick the "low hanging fruit" and leave the risks for later.
7. Partition the use cases/scenarios across the planned iterations.
8. Develop an Iteration plan describing each "mini-project" to be completed in each iteration.
   - Describe the goals of each iteration, plus the staffing, the schedule, the risks, inputs and deliverables.
   - Keep the iterations focused and limited (2-3 weeks per iteration). In each iteration, conduct all of the software activities in the process: requirements, analysis, design, implementation and test.

A Minimal Iterative Process

For each iteration: (repeat until done)

1. Merge the functional flow in the use cases/scenarios with the classes in the domain class diagram
   - Produce sequence (and collaboration) diagrams at the analysis level.
2. Test and challenge the sequence diagrams on paper, or whiteboard
   - Discover additional operations and data to be assigned to classes
   - Validate the business process captured in the flow of the sequence diagram
3. Develop statechart diagrams for classes with "significant" state
   - Statechart events, actions, and most activities will become operations on the corresponding class
4. Enhance sequence diagrams and statechart diagrams with design level content
   - Identify and add to the class diagram and sequence diagrams any required support or design classes (e.g. collection classes, GUI and other technology classes, etc.)
5. Challenge the sequence diagrams on paper/whiteboard, discovering additional operations and data assigned to classes.

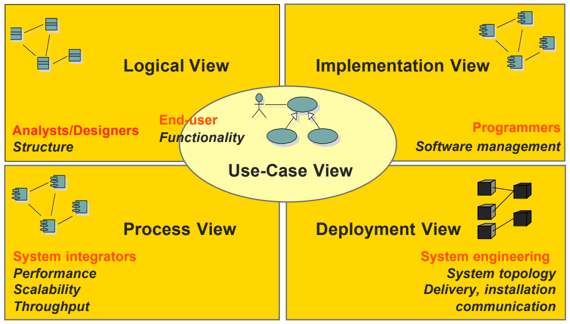
RUP Design - Where to Begin?



Architectural Analysis Topics

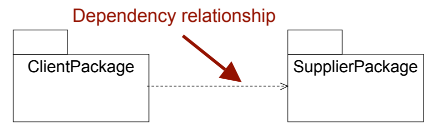
- Key Architectural Analysis Concepts
- Modeling Conventions
- Analysis Mechanisms
- Key System Concepts
- Initial Architectural Layers
- Architectural Analysis Checkpoints

“4+1 View” Model



Packages

- a general purpose mechanism for organizing elements into groups
- a model element which can contain other model elements
- uses
  - organize the model under development
  - configuration management
- can related to one another using a dependency relationship



- implications
  - changes to the Supplier package may affect the Client package
  - Client package cannot be reused independently because it depends on the Supplier package

UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004

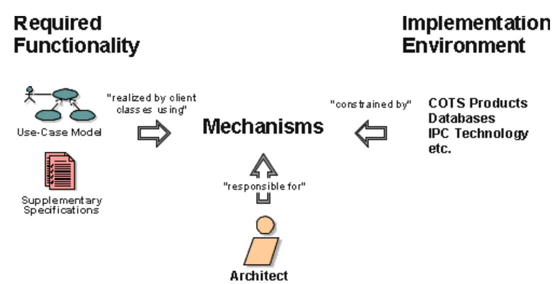
Example: Modeling Conventions

- Use Case View
  - Use Cases will be named with short active phrases such as “Submit Grades”
- Logical View
  - A Use Case Realization package will be created that includes:
    - At least one realization per use case traced to the use case
    - A “View Of Participating Classes” diagram that shows the participants in the realization and their relevant relationships
  - Classes will be named with noun names matching the problem domain as much as possible.

UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004

Architectural Mechanisms

- Analysis Mechanisms (conceptual)
- Design Mechanisms (concrete)
- Implementation Mechanisms (actual)



UNIVERSITY OF MASSACHUSETTS AMHERST • DEPARTMENT OF COMPUTER SCIENCE • CMPSCI 520/620 FALL 2004

Sample Analysis Mechanisms

- Persistency
- Communication (IPC and RPC)
- Message routing
- Distribution
- Transaction management
- Process control and synchronization (resource contention)
- Information exchange, format conversion
- Security
- Error detection / handling / reporting
- Redundancy
- Legacy Interface

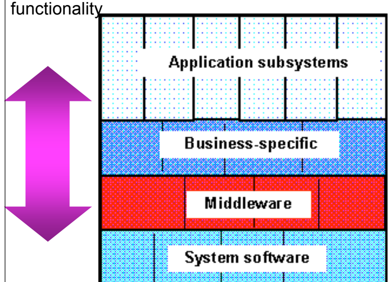
Identify Key Concepts

- Define preliminary entity analysis classes
  - Domain knowledge
  - Requirements
  - Glossary
  - Domain Model, or the Business Model (if exists)
- Define analysis class relationships
- Model analysis classes and relationships on Class Diagrams
  - Include brief description of analysis class
- Map analysis classes to necessary analysis mechanisms

Analysis classes will evolve

Typical Layering Approach

Specific functionality



General functionality

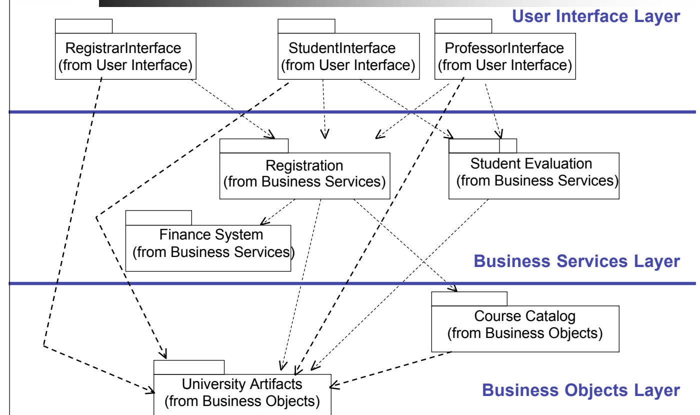
Distinct application subsystem that make up an application - contains the value adding software developed by the organization.

Business specific - contains a number of reusable subsystems specific to the type of business.

Middleware - offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.

System software - contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers and so on.

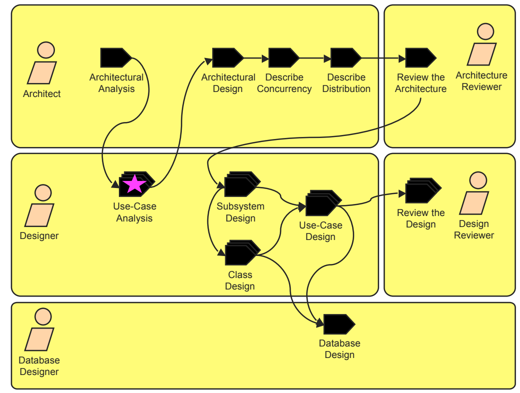
High-Level Organization of the Model



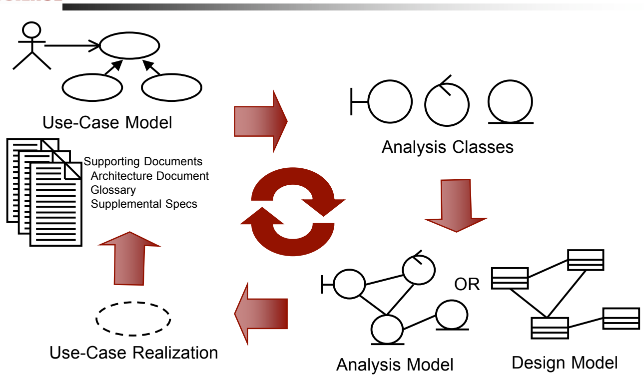
Architectural Analysis

- General
  - Is the package partitioning and layering done in a logically consistent way?
  - Have the necessary analysis mechanisms been identified?
- Packages
  - Have we provided a comprehensive picture of the services of the packages in upper-level layers?
- Classes
  - Have the key entity classes and their relationships been identified and accurately modeled?
  - Does the name of each class clearly reflect the role it plays?
  - Have the entity classes been mapped to the necessary analysis mechanisms?

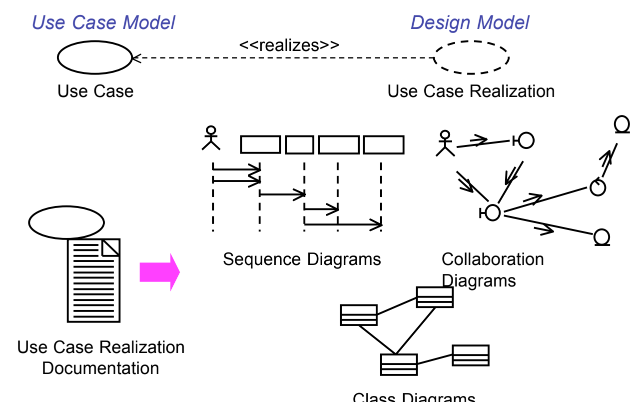
So whqat do we do next?



Use Case Analysis Overview



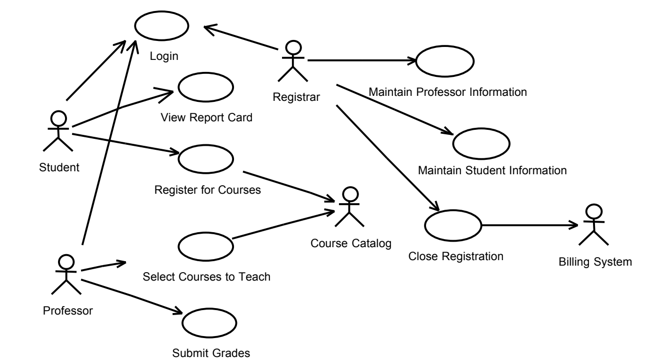
What is a Use-Case Realization?



Use Case Analysis Steps

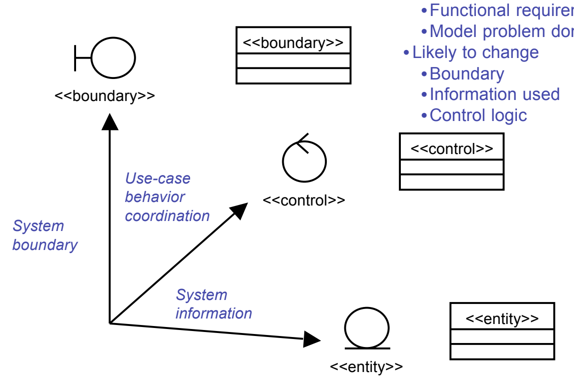
- Supplement the Descriptions of the Use Case
- For each use case realization
  - Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes

University Course Registration System



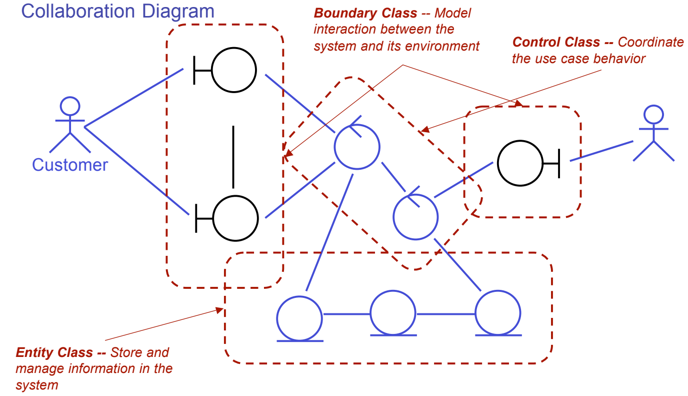
What is an Analysis Class?

- Early conceptual model
  - Functional requirements
  - Model problem domain
- Likely to change
  - Boundary
  - Information used
  - Control logic

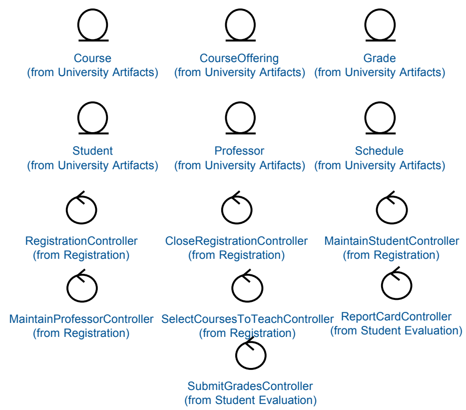


The Roles

Collaboration Diagram



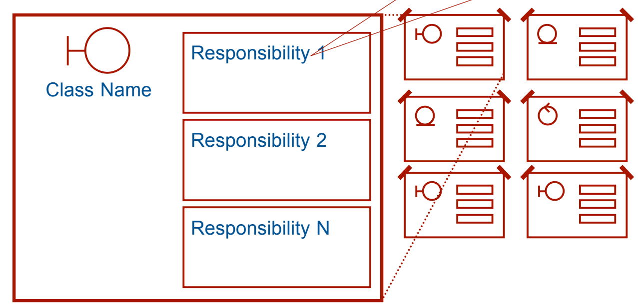
Example: Entity & Control Classes



Describe Responsibilities

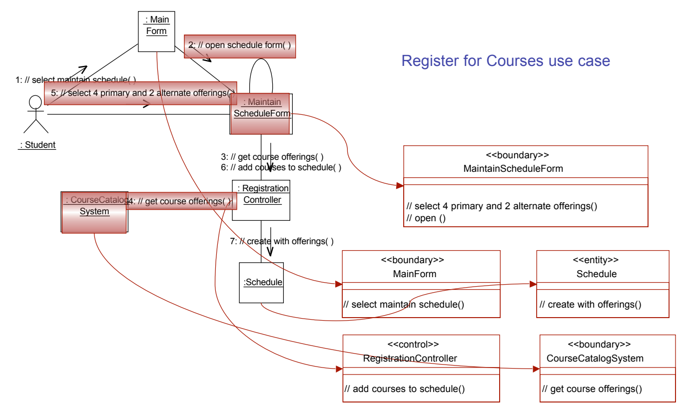
- What are responsibilities?
- How do we find them?

•First cut at class operations
  •Actions that object can perform
  •Knowledge object maintains
  •Non-functional requirements
•Class should have **multiple** responsibilities



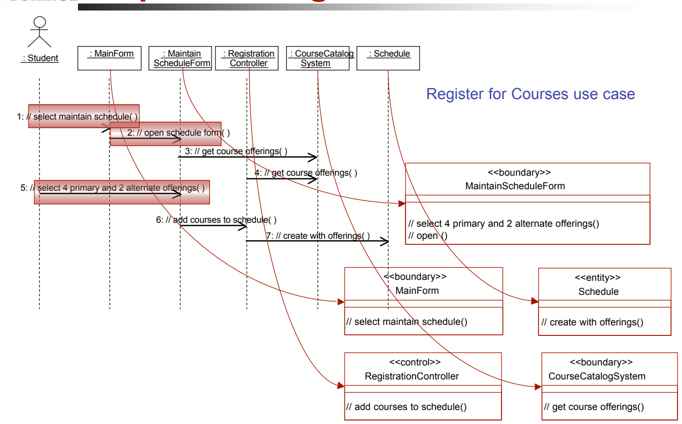
University of Massachusetts Amherst · Department of Computer Science · CMPSci 520/620 Fall 2004

Class Responsibilities from a Collaboration Diagram



University of Massachusetts Amherst · Department of Computer Science · CMPSci 520/620 Fall 2004

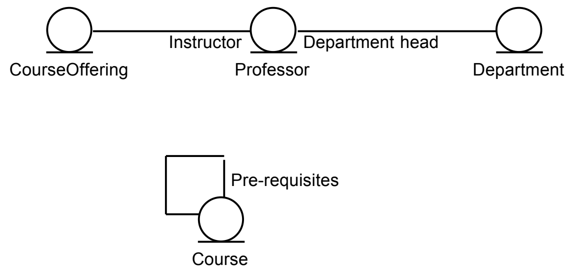
Class Responsibilities from a Sequence Diagram



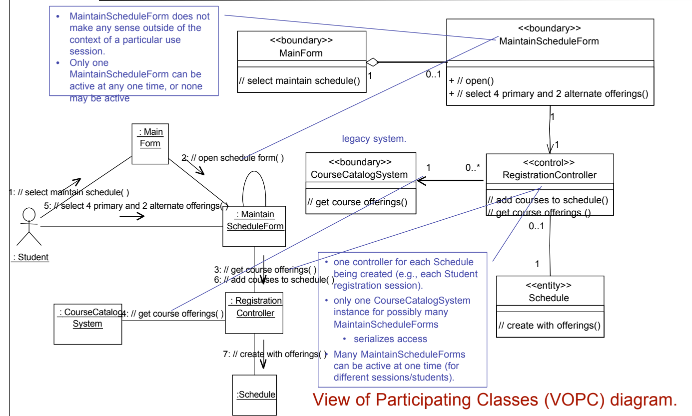
University of Massachusetts Amherst · Department of Computer Science · CMPSci 520/620 Fall 2004

What are Roles?

- The “face” that a class plays in the association



Example: Finding Relationships



View of Participating Classes (VOPC) diagram.