LOYOLA UNIVERSITY CHICAGO

CENTER FOR TEXTUAL STUDIES AND DIGITAL HUMANITIES

DIGH 401 - Introduction to Computing

Fall Semester 2014

Week 6

Today's Class

- Weekly Exercise 1 & 2
- XML (Part 2)
- Software Engineering

XML Exercise 1

Choose 2 of the following items:

- a car

- a sports team

- 1 Musical CD including each song per album per CD (each album has 5 songs)

Encode these items in XML and consider how you might structure your XML to ensure future extensible design and usage.

DIGH 401 - Introduction to Computing

Weekly Exercise 1 - Example Solution

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<vehicle>
<car id=”1” type=”SUV” manufacturer=”Ford”>
<model>Focus</model>
<variant>Hatchback</variant>
<doors>5</doors>
<trim>Sport</trim>
<colour>Black</colour>
<year>2014</year>
<fuel>Unleaded</fuel>
<transmission>Manual</transmission>
…
</car>
<boat id=”1” type=”motor” manufacturer=”Princess”>
...
</boat>
</vehicle>
```

To the Lighthouse

(Here Mr. Carmichael, who was reading Virgil, blew out his candle.)

3

But what after all is one night? A short space, especially when the darkness dims so soon, and so soon a bird sings, a cock crows loudly, or a faint green quickens, like a turning leaf, in the hollow of the wave.

It seemed now as if, touched by human penitence and all its tool, divine goodness had parted the curtain and displayed behind it, single, distinct, **the hare erect**; the wave falling; the boat rocking, which, did we deserve them, should be

DIGH 401 - Introduction to Computing

Weekly Exercise 2 - Example Solution

```
<?xml version="1.0" encoding="UTF-8"?>
<novel>
<page no=”198”>
<line no=”1”>
<title align=”centre” style=”underline” colour=”black”>To the Lighthouse</title>
</line>
<line no=”2” type=”empty”></line>
<para no=”1”>
<line no=”3”>(Here Mr. Carmichael, who was reading</line>
…
</para>
<line no=”6”>
<section align=”centre” style=”heading” colour=”blue”>3</section>
</line>
...
<line no=”11”><rend style=”highlight” colour=”green”>green</rend>...</line>
...
</page>
</novel>
```

XML Validation

SCHEMA

defines:

- elements and attributes allowed in a document
- child elements
- the order of child elements
- the number of child elements
- whether an element is empty or can include text
- data types for elements and attributes
- default and fixed values for elements and attributes

XML Validation

SCHEMA

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<calendar type="personal" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
<date>2011-09-15</date>
<title>DIGH 401</title>
<location>Room 201 Cudahy Library</location>
</calendar>
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="calendar">
  <xs:complexType>
    <xs:sequence>
<xs:element name="date" type="xs:date"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="location" type="xs:string"/>
</xs:sequence>
    <xs:attribute name="type" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Software Engineering Methods

Introduction

- the more lines of code, the harder it becomes to manage, debug, and maintain
- nearly impossible to write, test, and maintain a program with millions of lines of code by yourself
- software engineering helps solve this issue
- collaborative, coordinated efforts
- create guidelines to help consistency and deliverables

Software Engineering Methods

Introduction

- moves beyond single, inspired genius to a group of competent programmers
- a variety of methods have been developed by CS
- each method has advantages and disadvantages
- same basic goals of various methods
    - make it easy to write large computer programs within a reasonable time period
    - create software that works reliably

Software Engineering Methods

Waterfall Model

- divides a large project into distinct parts
- each part fully completed before the other part can even begin
- model divides a software project into 4 distinct, mutually exclusive steps
    - analysis
    - design
    - implementation
    - testing

Software Engineering Methods

Waterfall Model - Analysis

- firstly analyse what the program is supposed to do
- requirements are initially specified by the client
- questions will be asked by the programmers until clarification of requirements
- analysis will then be complete and no further changes may be requested
- requirements often formalised as 'program specifications' or 'specifications'
- analysis step freezes specifications within this model

Software Engineering Methods

Waterfall Model - Design

- programmers develop and write a plan to meet requirements
- deciding how to divide the program into smaller parts
- need to select programming language to use
- choose specific compiler and other tools
- many times the design phase does not reflect the client's requirements

Software Engineering Methods

Waterfall Model - Implementation

- writing code for each assigned portion of the program may begin
- monitor progress during this phase and see what needs to be done next
- design misunderstandings become more apparent during this phase
- design flaws can now make the program harder to create
- coding stops and parts are now collected and assembled

Software Engineering Methods

Waterfall Model - Testing

- ensures that the entire program basically works as intentioned
- bugs are fixed and program further tested to ensure no knock-on effects
- testing phase completed, and program is considered ready for delivery
- client may end up with a program that almost meets their needs but not quite
- changes or omissions require a new analysis and restarting of process

Software Engineering Methods

Waterfall Model - Conclusions

- this model assumes a number of things
    - progression through a series of steps without deviation
    - time per step can be predicted
    - each step accurately translates the client's requirements
- this method has not consistently produced large, reliable programs on schedule
- stalled phase can seriously damage a project
- errors can be easily compounded from one phase to the next
- biggest flaw is perceived to be the method's rigidity

Software Engineering Methods

Extreme programming (XP)

- opposite direction to over-structured waterfall method
- four different and mutually overlapping phases
    - coding
    - testing
    - listening
    - designing
- advocates of XP argue that the only true important product is code
- coding is not simply typing out programming commands
    - involves assessing alternatives
    - communicating effectively with client, other programmers...

Software Engineering Methods

Extreme programming (XP) - sequence

XP follows this general sequence

- client defines program requirements
- a simple program, prototype, is developed and acts as a model for the client
- programmers implement actual program with client providing feedback
- with a working model that the client likes, additional features, requirements can be added as required
- program can evolve slowly over time with feedback and requirements from the client

Software Engineering Methods

Extreme programming (XP) - perceived issues

- time scheduling is nearly impossible with XP
- relationship between client and programmers may be crucial to success
- programmer departures can cause delays between new programmer and client
- constant communication between all parties is crucial to the success of an XP project

Software Engineering Methods

Computer Aided Software Engineering (CASE)

- generally accepted practices for writing reliable software on time
- simple example might be avoiding spaghetti programming
- CASE tools are meant to simplify the practices of software engineering
- some common CASE tools include
    - project modellers
    - code generators
    - source code formatters
    - revision control
    - project management

Software Engineering Methods

Modelling a large project

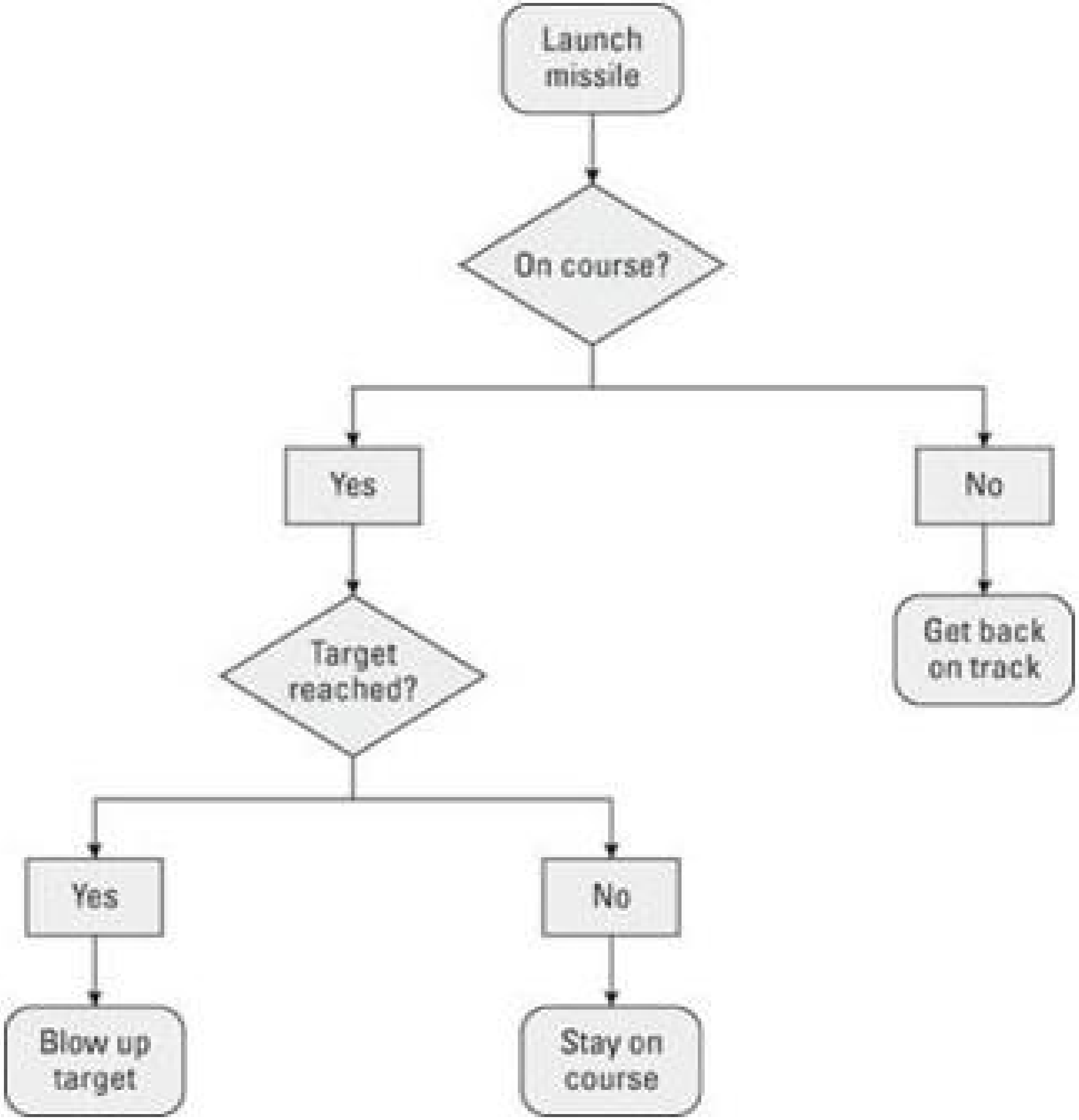
- design before coding
- everyone needs to understand the design before programming
- consistent model helps understanding
- everyone needs to use the same design model
- a number of models exist such as
    - flowcharts
    - unified modelling language (UML)

Software Engineering Methods

Modelling a large project - flowcharts

- one of the earliest modelling methods
- flowcharts must specify every action and decision that the program goes through
- describes how a program works at the conceptual level
- work for smaller programs but have a couple of problems with larger programs
    - easy to become too large and confusing to complete & take too much time
    - can become so complicated that no one uses them

DIGH 401 - Introduction to Computing



Software Engineering Methods

Modelling a large project - Unified Modelling Language (UML)

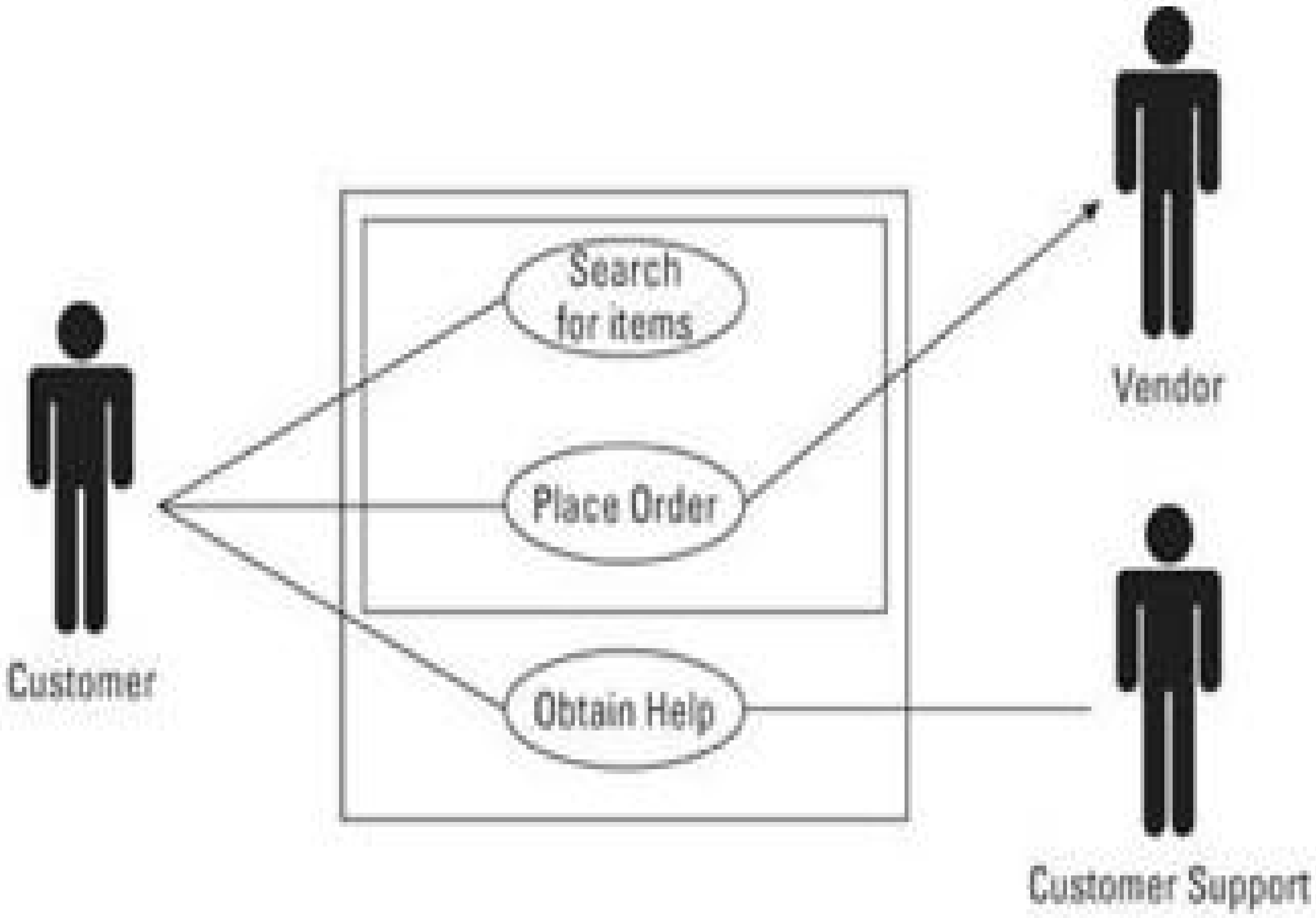
- provides a way to visually design a program
- universally understood symbols and diagrams
- UML offers different types of models for defining a program
    - functional models
    - object models
    - dynamic models
- pick and choose the type of diagram required to define your program

Software Engineering Methods

Modelling a large project - Unified Modelling Language (UML)

Functional Model (Use Case Diagram)

- describes how a user interacts with the program
- displaying users as stick figures and program functions as ovals removes a lot of the confusion


Search for items
Place Order
Obtain Help
Customer
Vendor
Customer Support

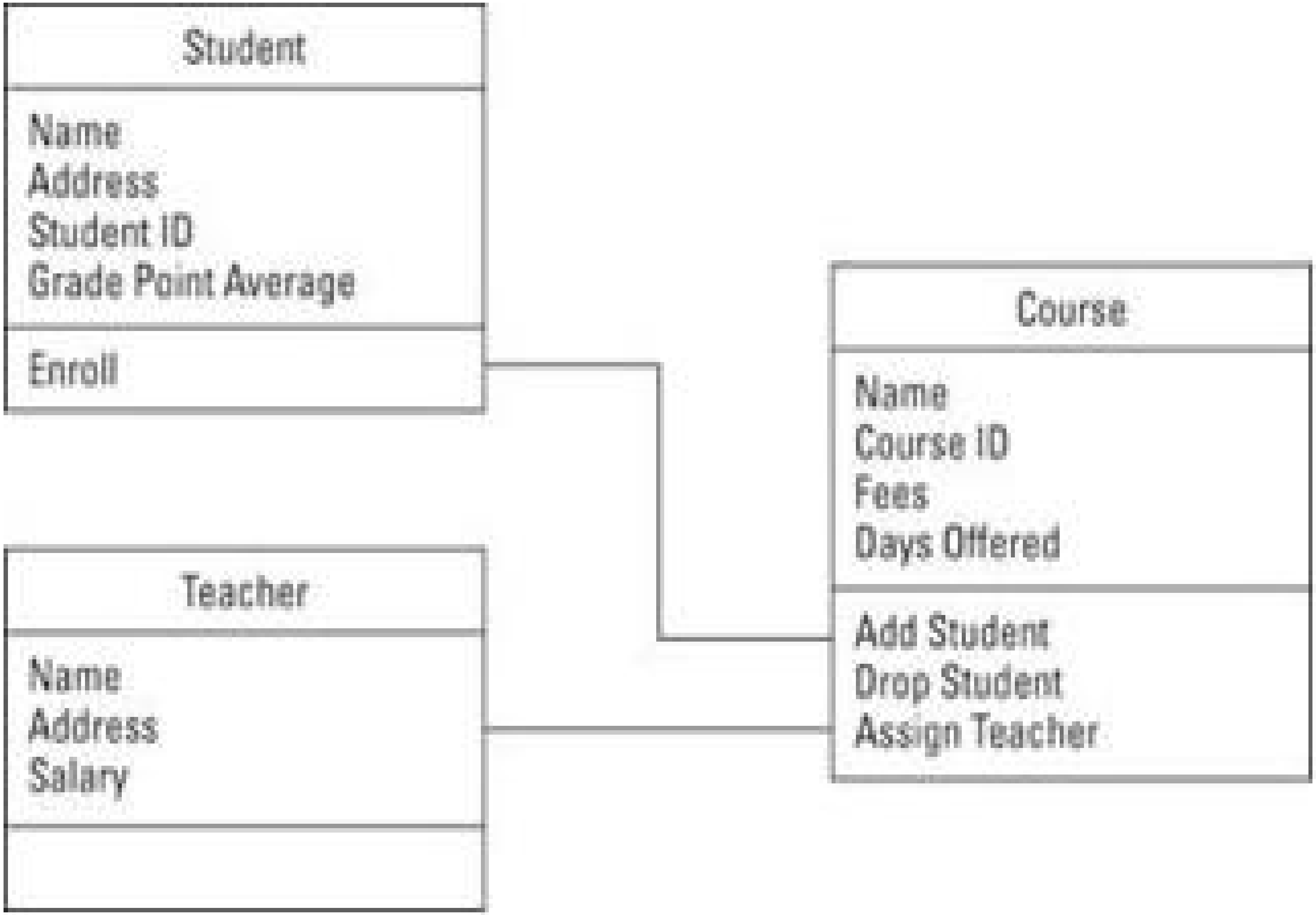
Software Engineering Methods

Modelling a large project - Unified Modelling Language (UML)

Object Model (class diagram)

- defines how to divide a program into parts or objects
- each class diagram defines an object's name, data (properties), and operations (methods)
- object's properties define the type of data it can hold
- object's methods define what type of actions the object can perform

DIGH 401 - Introduction to Computing



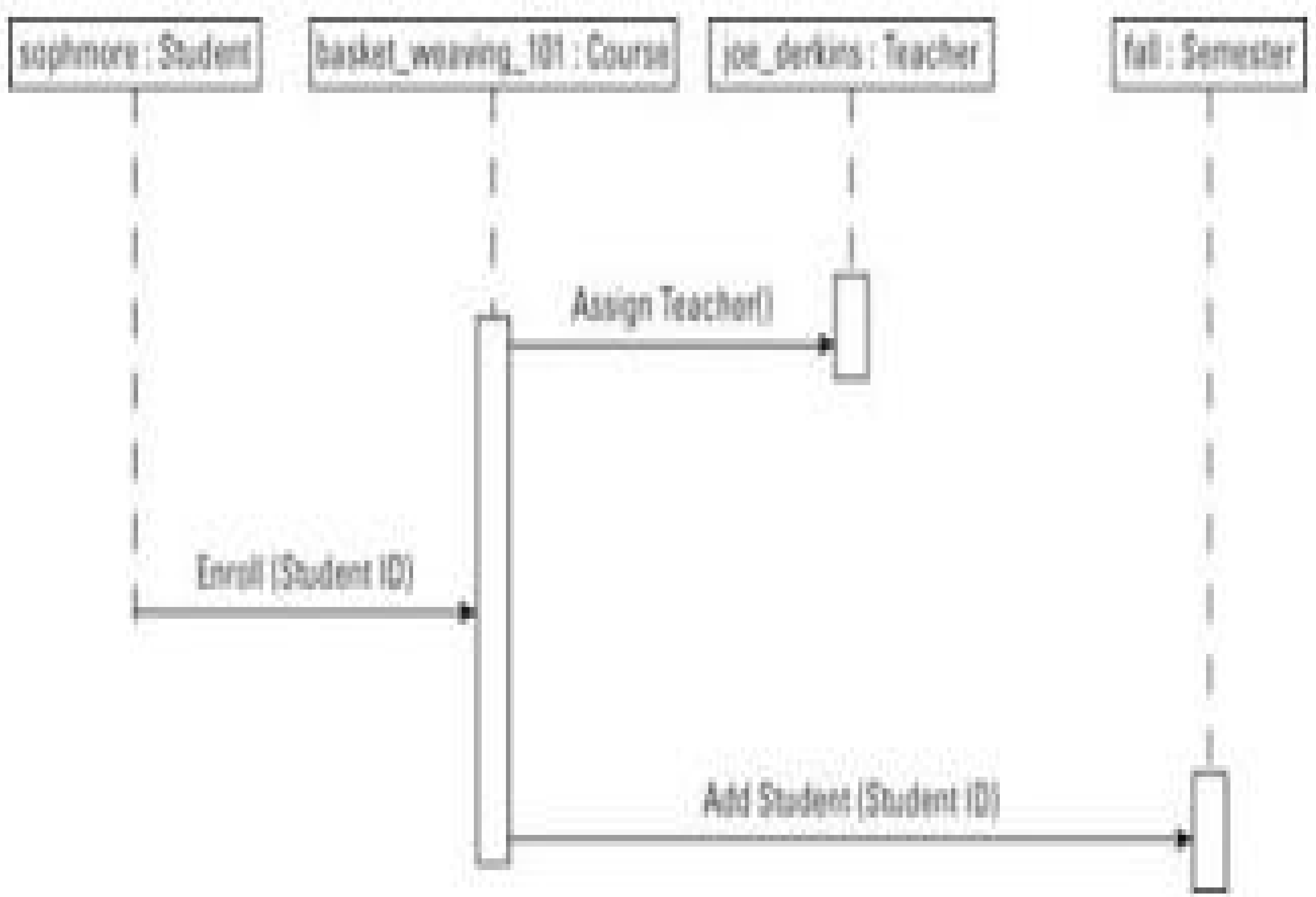
Software Engineering Methods

Modelling a large project - Unified Modelling Language (UML)

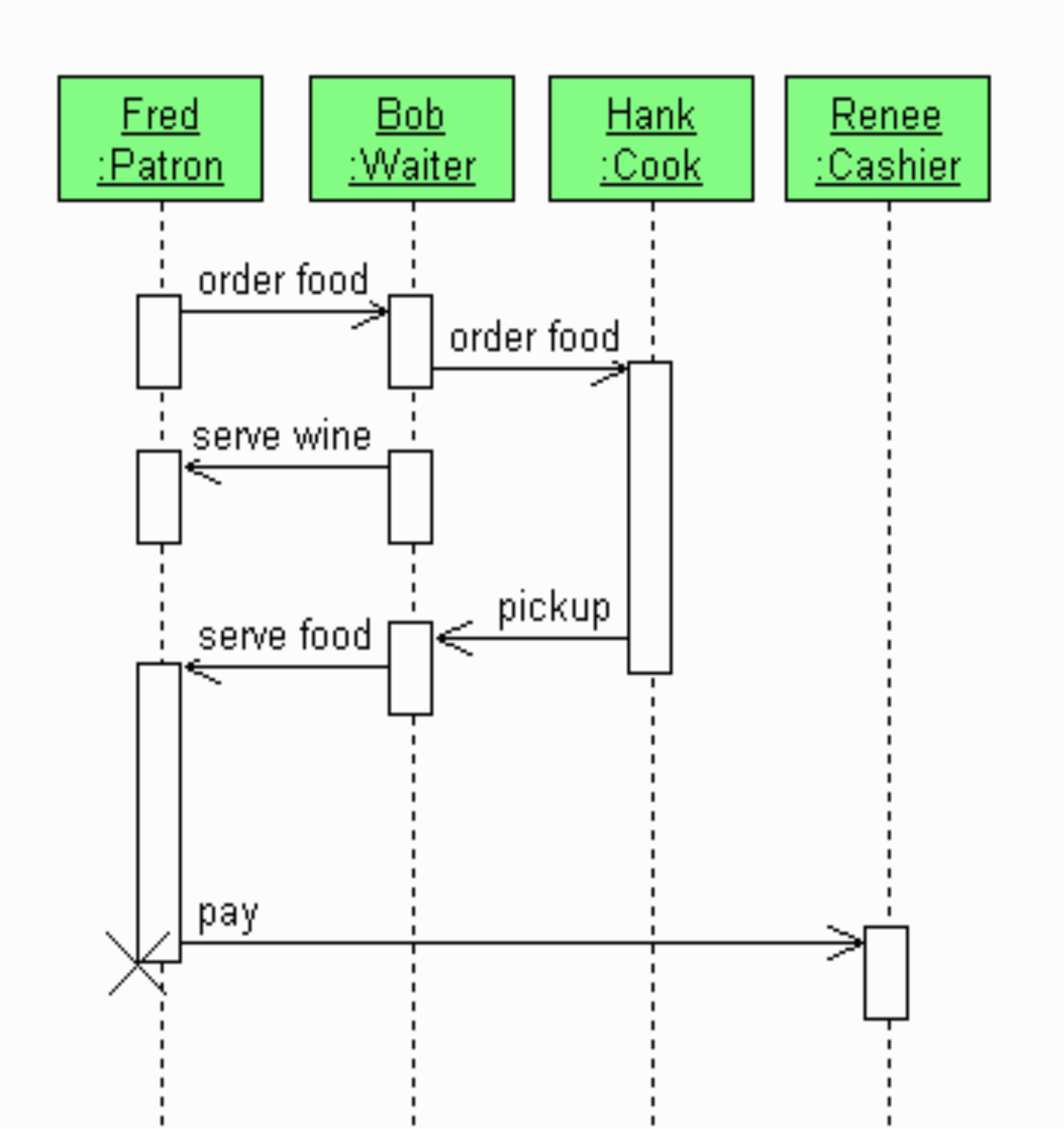
Dynamic Model (sequence diagram)

- describes the sequence a program follows
- similar to flowcharts because they describe what a program does at a given time
- they do not necessarily describe how a program works
- show you the order of occurring events
- show you messages passed from one object to another
- show you the actual names of all interacting objects

DIGH 401 - Introduction to Computing



DIGH 401 - Introduction to Computing


Fred
:Patron
Bob
:Waiter
Hank
:Cook
Renee
:Cashier
order food
order food
serve wine
pickup
serve food
pay