SFWR ENG 3A04 Summary

Author: Kemal Ahmed
Instructor: Dr. Ridha Khedri
Date: Fall 2014

Math objects made using [MathType](); graphs made using [Winplot]().

Please join GitHub and contribute to this document. There is a guide on how to do this on my GitHub.

Table of Contents

Lecture 2

Hierarchy of Requirement Specifications

Pre Requirements:
- Requirements:
  - Requirements Document
    - System Specifications
    - Other Documents
      - Legal

- Security
- Privacy
  - Architectural Design
    - Types:
      - Dynamic
      - Stable
      - Determined by:
        - Elements
        - Connectors
    - Detailed Design

Traceability Matrix

Traceability Matrix: a method of showing how each of the elements satisfies a requirement. You can use this to determine if a feature is necessary or if you are missing a feature.

Elements (E_i) \ Requirements (R_i)	R_1	R_2	R_n
E_1		P	P
E_2	T		
E_n			

Early Assignment Details

- The assignment can be submitted to a contest
- 2014-15 connect
- dx.org/connect
- Deadline: April 1st, 2015
- Prize: $2000

Requirements Cont.

Business Event (BE): the first, initiating input to a system that, but worded in the form of an event

Note: time can be an event, e.g. time to update your clocks

Environment / system interactions:
- *I/O between system and user*
- look at the system as a black box
- the last output occurs when the "business has been carried out"

Viewpoints (VP):
- *A target set of requirements*
- Think of it as different perspectives of how someone would want the system to be designed

- Includes things like who is using your product, but also who will be affected, such as economic perspective, i.e. cost

The more <u>viewpoints</u> you have, the better the representation of the system because you get a better overall perspective.

e.g. 1)
For a BE_1, you have a list of VPs from VP_1 to VP_n, and for BE_2 you have a list of VPs from VP_1 to VP_m.

If you have 2 viewpoints that have little relevance, you don't get rid of it. Instead, you mark them as void. This is because you may need it for the next BE(s)

Functional Requirements: fundamental reason for the system to exist
Non-functional Requirements: properties the system must <u>have</u>, e.g. precision, availability, security, usability, look, etc.; it is based on the environment of the system; more qualitative
Constraint: global issue that <u>shapes</u> the requirements; quantitative limits

Determine functional, *then* non-functional requirements.

Scenario: interactions between the system and the user / environment (could be time)

Mode: what you think it means, but formally, a non-empty set of equivalent states
- reflexive
- transitive
- symmetric
- $x'Ry$ and $y'Rx$

Complete graph with n nodes is K_n.

Design Space
- Hardware-hiding modules:
  - Language to communicate with the hard drive
  - Virtual Machine hiding module
- Behaviour hiding modules:
  - Controller classes: sequence of events
  - Change due to requirements
- Software decision-hiding modules:
  - Algorithms
  - Physics constants
  - Theorems (i.e. math)
  - Data types
    - n-Tuple; a record
      - n gets

- - *n* sets
  - ▪ Set
    - • IsMember
    - • IsEmpty
    - • Insert
    - • Remove
  - ▪ List
    - • IsEmpty
    - • GetHead
    - • GetNext (last element)

Asynchronous operation: process operates independently of other processes
Synchronous operation: other processes finish before some other process has finished
Blocking: process causes other processes to stop
Non-blocking: process runs without stopping other processes
[More](#)

Semaphore:

Protocol: a method of communication

MVC: the way every software program is analyzed
Model: (a.k.a. Data level) constants and stored data the system interacts with
View: (a.k.a. Interface) what the users see and how they interact with the system
Controller: (a.k.a. Business Logic) what processes the data from the model

**Connector**: an indicator of interaction among components
Signature-based connector: single operation; works as long as you communicate using the correct inputs (like Radio)
**Protocol-based connector**: multiple operations; when communicating, both communicate with each other and confirm a connection (like WiFi)

Formal model: a representation of what you are going to build, based on math
Informal model: not formal

Configuration topology: different shapes of networks, including bus ignore arrows

ring

star

mesh

hierarchy

and extended star
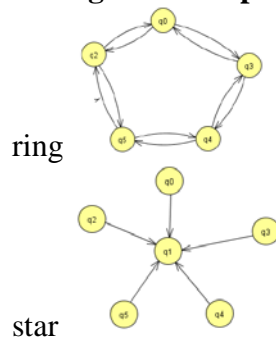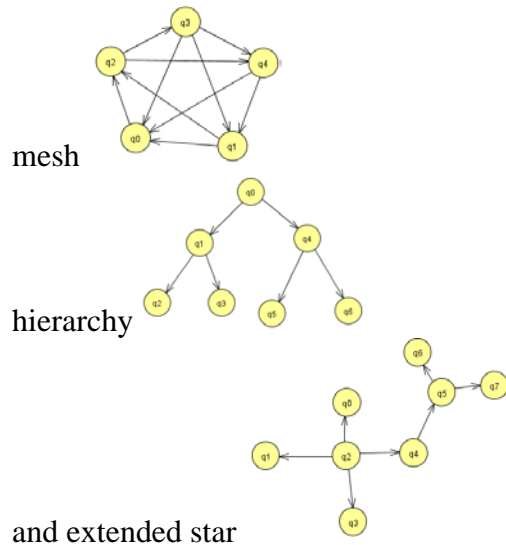
**Unified Modelling Language (UML)**:

Class Name
Attributes: name: String address: String
Operations / Functions

It is usually organized in **structural diagrams**, which show relationships between classes through connectors.

Diagram Types

Architecture Description Language (ADL): languages describing the software and hardware architecture of a system

- usually graphical syntax
- Supports the tasks of architecture creation, refinement and validation
- Provide a basis for further implementation
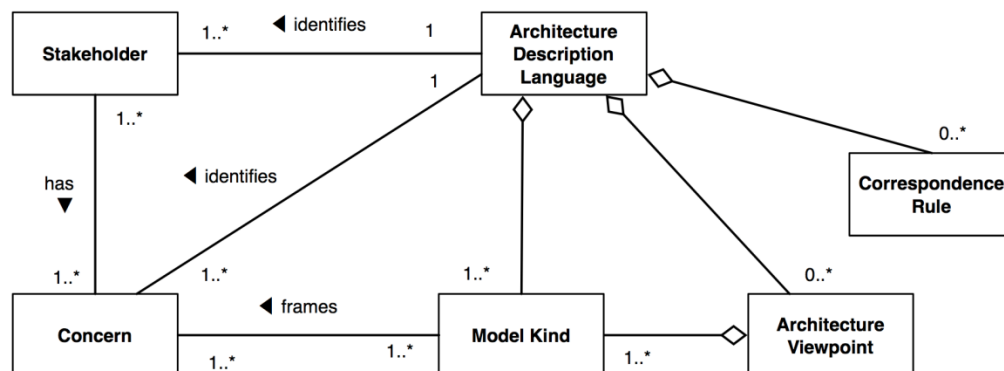
Inheritance: [*identified by arrows*] the child gets some of its data / functions from the parent objects, although local functions have higher precedence; the child class can access the parent's classes <u>publicly</u>

<u>Example</u>:
```
class Dog
        Eat;
        Walk;
        Bark;
        Play;
end;
class Cat extends Dog    //is
        Purr;
        Bark = null;
end;
```

Aggregation: [*hinge identified by hollow diamonds*] something is made of independent parts that can exist without the parent object (think: is it useful on its own?); the child can access the parent's classes <u>privately</u>

```
class Cat includes Dog; //has
        Eat = Dog.Eat;
        Walk = Dog.Walk;
        Play = Dog.Play;
        Purr;
end;
```

Composition: [*hinge identified by black diamonds*] parts are dependent on the parent object to exist

```
class Pet
        Eat;
        Walk;
        Play;
end;
class Dog extends Pet    //is
        Bark;
end;

class Cat extends Pet    //is
        Purr;
end;
```
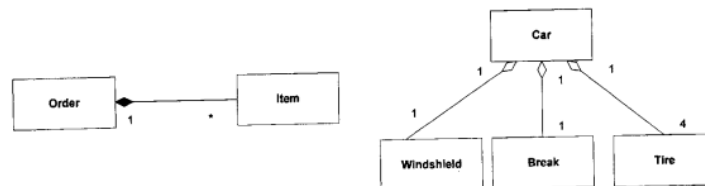

Figure: Composition (left) Aggregation (right)
The item doesn't exist without the order; the windshield is useful without the car existing.

Note: "include" and "extend" mean different things here than in use case diagrams

It's especially important to have low coupling when you can't change the higher level object

Dependencies: [*identified by dashed arrows*] if if a class, X, depends on another class, Y, then changes to the elements Y will lead to the changes of X
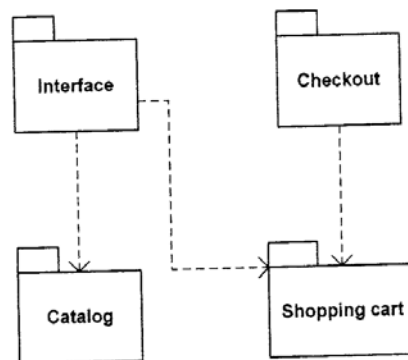
Structural
Composite Structure Diagram
- Rectangle: structural classes
- Ellipse: abstract construct of relationship between classes

Component Diagram
- Balls: class that outputs
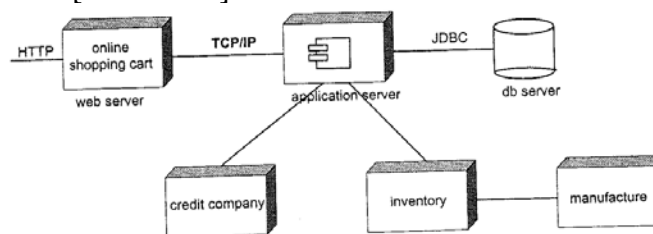- Sockets: class that takes input from balls

Package Diagram: package structure
- Folders: packages

Deployment Diagram: physical hardware, software, network connections
- Cubes: computing resources
- Cylinders: database [sometimes]


Behavioural
Use Case: how system reacts to BEs
- Communication between actors
- **Actors**: [*represented by a stick figure*] does not have to be a human
  - provide BEs
- *Include*: mandatory behaviour; the child needs the parent to exist
- *Extend*: optional behaviour; the child can exist without the parent

- Uses:
- "Use Case" ⇔ "Scenario"
- Each ellipse is a use case
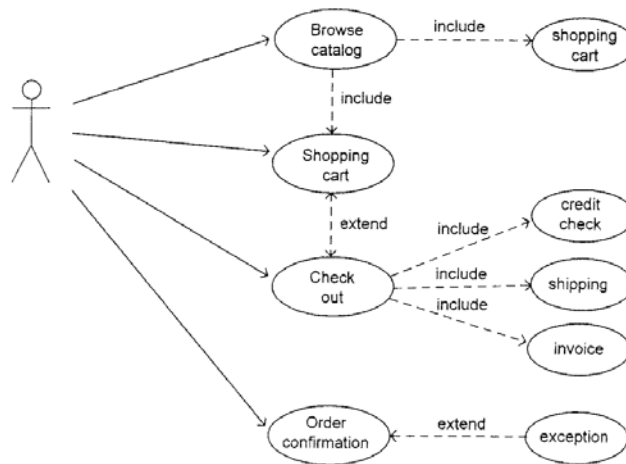
Activity Diagram: data and control flow of system
- Rounded rectangles: actions in system
- Solid hub: fork and joint points
- Surrounded disk: terminate
- Diamond: decision
- Disk: start point
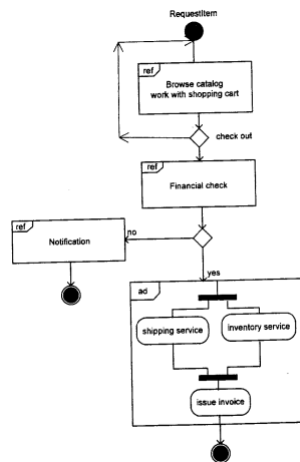
Sequence Diagram: how flow thru classes to fulfill requirements
- Rectangles on top identify classes
- Arrows show flow of data and how they fulfill requirements
- Smaller boxes inside the bigger boxes are other implementations of the same object
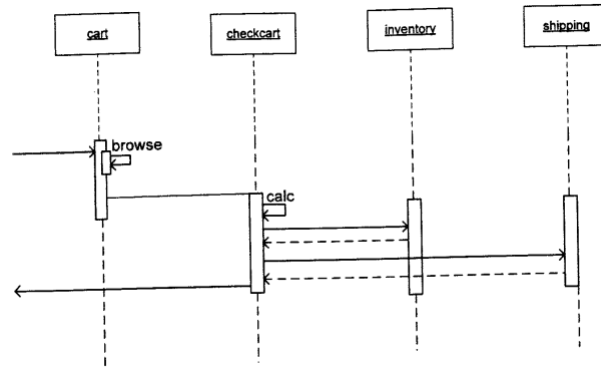
4+1 Model:
- Scenario: overall encompasses other views
- Logical View:
- Physical View: how software interfaces with equipment, hardware, etc.
- Development View: how classes and directories are organized
- Process View: communication between classes
- User Interface View: look & feel of product
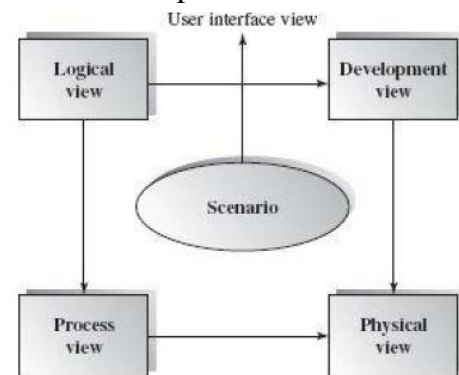
Abstract Data Types

ADTs: the study of structures

Types of ADTs

For a given Set, what are the Functions of the $_{Set}$? (S, F_S)

$(\mathbb{N}, F_\mathbb{N}),$
$(\mathbb{Z}, F_\mathbb{Z})$ $\subseteq (\mathbb{R}, F_\mathbb{R})$

Algebra: $(\mathbb{C}, \{+, \cdot, \ldots\})$

Signature defines how number types change after an operation
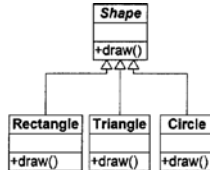$+: \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

Numbers in an ADT must be:
- Finite

- Discrete
- Countable: there is only one number for each number
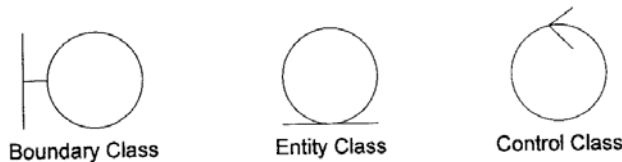 - $f : \mathbb{N} \rightarrow S$

Object Oriented Analysis & Design

Generalization / pattern: [*denoted by hollow triangle arrow*] inheritance relationship

Order Processing System (OPS):

Secrets:
- Boundary classes:
  - Hardware-hiding
  - Virtual Machine
  - Interface
- Entity classes: data structure
- Controller Classes: algorithm

Polymorphism: being able to access different functions with the same function name
- Horizontal overloading: having multiple functions within the same class, usually for different input types
- Vertical overloading: having functions from a parent and child class
  - Take a Lion, Tiger, Bear, and Fish. They are all Animal objects. Say the animal object has a function, hasClaws=true. The Fish object also has a function hasClaws, except its value is hasClaws=false

Types of Architectures
- block flow architecture (62)
- batch sequential architecture (63-64)
- pipe and filter architecture (64-68)
- Process control architecture (68)
- Repository architecture (71-73)
- **Blackboard architecture (75-78)**
- Hierarchical architecture (81)
- Main-Subroutines Architecture (82)
- Master-slave architecture (83)

- Layered Architecture (84-86)
- **model-view-controller (MVC) architecture**

State Diagrams
input/output