Reminder: quizzes are 15% of final grade

Why is software engineering necessary for IT? What is the motivation for software engineering?

Software engineering solved the software crisis in the 1960s/70s

* First computer showed up in 1940s
* Computers were originally used for military
* US Department of Defense (DoD) project used computers
* **Software crisis** in 1960s/70s
  + However, projects were **always late** and took years to complete
  + **Unreliable, very buggy**: succeeds one day and fails another
  + **Expensive**: $1-10 million to develop, maintain, and upgrade
* **Software development** (before software engineering)
  + Try-and-fix ← the way they used to develop a system
  + Let it run until a bug and then fix it
* **Solution to software crisis**: use software engineering
  + Software engineer is a copycat of conventional engineering
  + Example: how to build a bridge?
    - Wrong way: try-and-fix
      * Build a bridge
      * Let people walk on it
      * Sacrifice people and let them die
      * Do it again
    - Right way: using engineering
      * **Investigation for feasibility by doing field work**: survey river/bank and decide whether or not it is feasible
      * **Design the project**: draw blueprint
      * **Implementation**: construct the bridge
      * **Test**: not open to public for months
      * **Deliver**
      * **Maintain**
      * **Dead/retire**

What is the major difference between software engineer and conventional engineering?

(build bridge vs. build a rocket control system)

1. Software vs. physical things
2. Lack of visualization: people cannot see your effort when building software systems
3. Moving target: customers feel more free to change specifications for software projects

QUIZ QUESTION

**Reminder: software engineering is a copycat of conventional engineering**

Key differences between software and conventional engineering are:

* the lack of **visualization** in software engineering
* **moving targets**: clients won’t be afraid of **asking for changes in specs** for software engineering because it is not physical
* The communication between the developer and client is difficult for software engineering

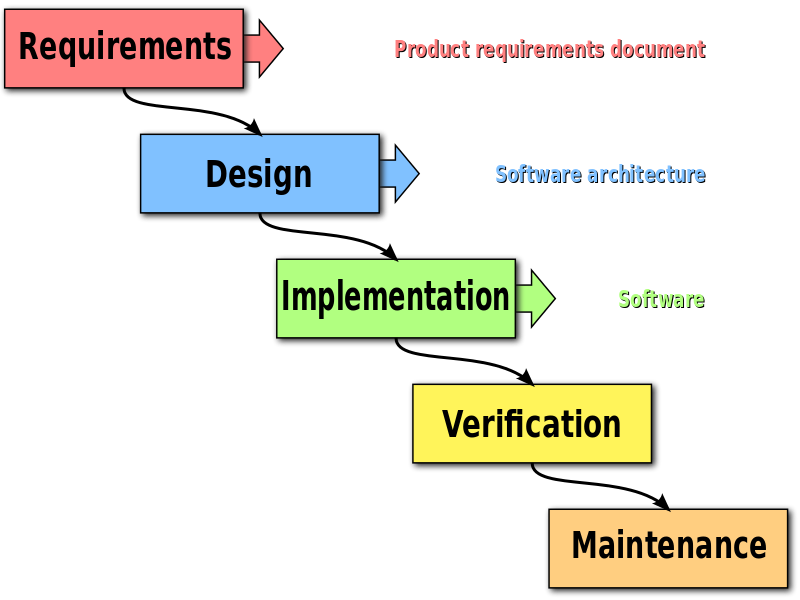
What is computer science?

* Problem solving using computers effectively
* Main difference between this field and others is that we use computers

What is data science?

* Computer science + statistics + math
* Machine learning vs. data mining
  + Machine learning
    - Small size data
    - Can afford to use complex algorithms, lots of statistics
  + Data mining
    - Hard-core database
    - Involves big data: trade time/space for efficiency, hashing, indexing
    - Try to extract knowledge from raw data
    - Knowledge discovery
* Deep learning
  + Huge data + deep algorithms

What is the waterfall model for software engineer?

**One life-cycle model, waterfall model for software engineering**: progress flows from the top to the bottom like a cascading waterfall

1. Investigation: specification
2. Planning: people/money/s/h
3. System design
4. Implementation
5. Integration
6. Testing
7. Delivery
8. Maintenance
   1. Legacy language: COBOL (common business-oriented language), pays $65k/year
9. Retire

**Computable vs. Infeasible**

* Several aspects of complexity
  + **Time**: O(n^2), O(log n)
  + **Space**: how much space your algorithm will use
* **Efficiency**: time/space complexity together
  + If we are efficient, we are using little time and space
* Generally speaking, **there is a give-and-take between time and space**
  + If we want it to take little time, it may take more space
  + If we want to use little space, it may take more time
  + There is usually a choice between a smart algorithm or a smart data structure to solve a problem
* Niklaus Wirth
  + Turing Award winner
  + Wrote a book called Algorithms + Data Structures = Programs

**Big data era**: O(1) is a fixed number of auxiliary variables

* Variable O(n) is a no-no
  + If an algorithm for big data takes O(n), it takes a long time, so you'd rather aim for O(1)
* Even before the big data era, O(n2) is the boundary
* O(n3) is not acceptable
  + Algorithms with this time complexity should be rewritten
  + Even if it gives optimal results, it takes too long
  + Should settle for sub-optimal results instead if it means it’ll take less time

In-place = 0 auxiliary space

**Criteria between science/engineering vs non-science/engineering (like art)**

* Engineering: application of science to real world problems
* Science is more objective
  + **Testing**: reject/accept/falsify based on evidence
  + **Repeatability**: provide source code
* Art is more subjective

We should have a measurement of goodness of a system

Example: Apple vs. Windows machine (Dell, Thinkpad)

* Apple? → expensive, looks prettier, has better battery
* What is the most important feature of Apple? What makes it successful?

Using what we know about Apple, what do we have to consider to make our programs successful?

What is the objective criteria we should use to evaluate things in science and engineering?

* **User-friendliness** ← most important for customers
  + Led to commercial success
* **Robust**: no crash for user errors
  + Include tests/assert to ensure programs behave the way we want them to
  + **Never assume user is intelligent**
    - Example: input your age: someone can type 0
      * Have sanity checks to make sure people don’t do stupid shit
    - Be defensive! Don’t allow people to delete important information.
* **Understandability**: easier to your peers/self
  + Clean logic
  + Save effort and time
* **Reusable**: make a library

**Generations of Programming Languages**

* 1st gen: 1950s, binary machine code
  + Example: 00010110
  + Good for machines, CPU
  + Not good for humans
* 2nd gen: Assembly
  + Example: add ax, 01: 010101: 01: add, 01: ax, 01: #
  + add ax, 01: English-like language/sentence
  + Much easier to communicate
  + There is a 1-1 correspondence between assembly and machine language
  + For some, this was a huge jump from machine language
    - Understability significantly improved
    - Revolutionary
* 3rd gen
  + Big 3 statements in any programming language
    - Loops
      * Loop invariant / recursion
      * Should be ~10 lines or < 20 lines
    - Conditions
    - Sequences;\n (von Neumann)
  + Examples of 3rd generation programming languages
    - Pascal language
      * First programming language in the whole world for many years
      * Pure 3rd-gen language
      * Created by Niklaus Wirth who won a Turing award for it
      * Useless by industry
        + nobody used it because it was too pure
    - C: 2.5-gen, assembly+pascal.
      * Very flexible
      * C++ vs. Java
        + Java is more pure in terms of object-orientedness
  + **Turing Complete**
    - A programming language is said to be Turing complete if it can be used to simulate any Turing machine
* 4th gen
  + We are currently at this point
  + 4.0: Object Oriented (OO) languages
    - Examples
      * Java
      * C#
      * C++ ← not a pure OO language, but is partially OO
      * Smalltalk
      * Ada
    - **Inheritance**: most important feature: of OO
      * Better reuse
      * Modularity
        + Better than copy and paste

Example: if we have a student, then a grad\_std (graduate student) would just be a student + a new feature

Copy and paste is bad because there is **no correct software system**

“my system has no bug!” = lies

There are always errors and bugs

We should try to localize features in order to localize errors

* + - * + Instead of using copy and paste, you can use inheritance because all of the code is in one place
        + If there are bugs in the code, it is easier to fix that way
  + 4.1 database SQL
    - Example: create dean’s list of boys
      * + SELECT name
        + FROM std\_DB
        + WHERE gender=’M and gpa>=3.5;
    - Optimal evaluation plan
      * More selective: smaller number of tuples will pass
      * What if more females have higher GPA?
      * GPA is selective in public colleges, eg. CCNY
        + In some private college like Harvard, most GPAs are >3.6
        + In that case, using >=3.5 GPA is not a good filter
    - Database: a huge list of small tables
      * Don’t use one big table! This is bad practice
      * Instead, use multiple smaller tables
      * std\_DB: class def:
        + decide attributes of interest: ssn, name, address, department, ~~dept\_chair,~~ college, ~~college\_president,~~ country~~, country\_president~~

any problems with these attributes?

Irrelevant features

There is no logical relation between the country’s president and a student’s college

Space redundancy: waste space by having dumb attributes

Update redundancy: integrity

* + - * dept\_chair\_DB: can also be called “dictionary”
        + Dept, chair
        + CS, Akira Kawaguchi
        + Math, <insert math department chair>
      * college\_p\_DB: USA dt
      * Keep many macro information to speed up queries
        + gpa, hotness (popularity) table
        + Example: if many people in NY look up “Obama”, Google will put the search results in servers in NY so that the search is faster
* 5th gen
  + Created in 1990, Japan

Three programming paradigms

* Imperative
  + Uses statements that change a program’s state
  + Consists of commands for the computer to perform
  + Focuses on describing how a program operates
  + Python: functional programming style but not functional programming
    - OO
    - Recursive program is very slow
* Functional
  + Treats computation as the evaluation of mathematical functions
  + Avoids changing-state and mutable data
  + A declarative programming paradigm
    - Programming is done with expressions or declarations instead of statements
  + LISP: 1950s, <10 phd students in MIT.
    - Functional programming: google/fb
    - Influenced Scala and Haskell
    - No assignments and no loops, only recursion
* Logic
  + Largely based on formal logic
  + Any program written in a logic programming language is essentially a set of sentences in logical form
    - Expresses facts and rules about some problem domain
  + Prolog: based on predicates
    - Tiger(X) // X is a tiger or not?
    - x=”tiger woods” => false
    - Tiger(X)
      * animal(X)
      * body\_stripes(X)
      * eat\_meat(X)
      * not\_climb\_tree(X)

Quiz: 1 question on algorithm design

* If he’s asking us to make an **efficient** algorithm, try to come up with O(1) or O(log n)

Generally speaking

* O(n^3) is terrible. Should not be used.
* O(n^2) is still pretty bad
* In big data, aim for O(1) or O(log n)
  + Hashing using dictionaries (like in Python) is very fast
    - Time complexity: O(1)
  + Binary search/dictionary search
    - Time complexity: O(log n)
    - #1, most important algorithm

Time complexity

* Big-O: worst case, upperbound; <= time complexity
  + Example: O(n) means the time complexity is <= n
* Small-o

QUIZ REVIEW

Waterfall Model (quiz)

* Waterfall model: don’t confuse with life cycle model it specific
  + don’t do this on the quiz!!!!))!)))! he asking for definition of waterfall model and life cycle model
    - life cycle model - steps taken to develop the system
* Popular life cycle:
* Note: build and fix is NOT a life cycle model
* Before (?) Software engineering
  + Pros: strict principle, complete doc
  + Cons: time consuming, lack of visualization, moving target, **lack of communication** between clients and developers
* System quality assurance (SQA)
  + Used by an American engineer who moved to Japan
  + For every phase, there should be an independent check out for the system to satisfy the specification
  + We should introduce an objective party
    - The objective party is independent of the developers, who are likely to be more subjective
    - They are very objective and can give critical feedback
  + ra(?) → planning team → design team
  + Specification is a legal document
    - It is very important between the client and developer
    - Specification is the only documentation between the two parties
* US Department of Defense (DoD): Ada, major funding for software engineering
  + Their data is very important. Very concerned about safety.
* Waterfall model is used because there is low risk
  + The first life cycle model
  + Not really different from actual engineering
    - Except the visual of work
  + cons: lack of communication, lack of visualization, moving target
* scale: toy system do not work with a lot people? wt

Machine learning vs data mining/science

* statistics: small data
  + NIH: 20 patients
* How to infer information from small data?
  + What is small data? ~1000 data points

How to measure the software quality? How good or bad is the software system?

* Most important software quality
  + 1) satisfy specification (to get paid $$$)
  + 2) user-friendliness (reason why Apple is so rich)
  + 3) understandability by other coders
  + **reliability vs robustness** (they are different):
    - Robust against errors: if the user gives a silly input, the system won’t crash; defensive
      * Should at least respond decently if the input was silly
      * logically robust: random anchor in quicksort
      * many servers everywhere
    - Example of how to make system robust
      * division(a, b)
      * User inputs 2 numbers: 2, 0 → crash
        + 1. be user friendly: “input 2 numbers 2nd cannot be 0!” ← be clear to the user
        + if the user still enters 0, call them stupid and tell them 2nd cannot be zero!!!!! (again)
        + or use assert(num2 != 0) to be defensive
        + at this point it’s the user’s fault, not the programmer’s.
    - Reliability:
      * program performs as intended

Efficiency (both time and space)

* O(log n), 2 log n, 10 log n are all good
* efficiency: <O(n) sublinear
* (Note: linear alg is considered efficient)
* Hashing (O(1)) and sorting (O(log n))
  + Since sorting is shit compared to hashing, why is sorting not dead if hashing is constant?
    - While O(1) is average case, collisions can occur in hashing and can make it worse.
    - Knuth theorem: regardless of hashing algorithms, you will always have collisions given large enough input (?)
* vivid:
  + what’s the optimal/best hashing function?
    - Uniform probability - the more random, the better
      * the best hashing function: ask a monkey to throw a ball
        + eg: you have a lot of bins, and give the monkey a ball.

You want probability of ball going in any bin to be equal

* + - * + don’t give a ball to a human, they might throw it into the same bin every time.
        + n bins, n balls. What is the probability for bin # k is empty?

P(not in k) =( (n-1)/(n) )^n = (1-1/n)^n = 1/e

37% rule → (1/2.7)

https://math.stackexchange.com/questions/420018/proving-lim-limits-n-to-infty-left1-fracxn-rightn-textex

does this equals e???LOL what does the limit have to be m → infinity… yes <https://www.wolframalpha.com/input/?i=limit+as+n+approaches+-infinity+of+>(1%2B(1%2F(n)))%5En

* + Other possible function: H(n) → 2 \* n (probably not a good hashing function lol)
  + n values to hash, should have 2n bins
    - increase the space to facilitate good hashing to avoid collision
  + 37% rule:
    - Company wants to hire a new employee and there are 100 possible candidates
      * in one day, they can only interview one candidate.
      * what is the optimal strategy to proceed?
        + Greedy (?) hire then fire when you get someone better
        + machine learning!: heartthrob to make a wise decision.

have to learn how good the candidate is. but how do you learn? we use some candidates as learning targets.

* + - * + “cold math (heartless method)”:

just interview first k candidates, remember their performance and never hire them.

from k+1, find something better than the best of top of k, hire him/her

eg. first don’t commit to friendship, learn how people are, and then you decide after you know how people are.

* + - * + Tindr method: swipe right and take what you can get :) LOL
        + what if you have 100 candidates? what is a good K?

1 is too rushed

99 you will have regrets

follow 37% rule!

So first 37 doesnt mean shit and is the learning process

* + To be nice to a potential husband, you should spend some time with them before you marry them
    - Gonna make this highest level, important lessons of software engineering
* Usually used with waterfall to assist
* reflect main function of system in order to communicate with clients/peers
* embedded in waterfall model
* pros: better communication between client and developers, better testing?
* OS: you want to build a programming language/basic. however, for the system to communicate with hardware, you should have a simulation/prototype of CPU/hardware.
  + top 2 coders: one guy takes care of the programming language. the other makes the prototype.
    - Program language guy: Bill Gates
    - Rapid prototyping guy: Allen (now the owner of seattle seahawks)
  + Eventually, when they make sure both are right, they just use the prototype ?? prototype will be thrown out later.??? ??? idk either
  + ROM-Basic
* rapid prototyping assists waterfall model;
  + They are usually used together
  + Ensures that the communication is better
  + python/matlab: started as prototyping languages. but they were powerful enough to be made their own language

Rapid Prototyping

* Used 90% of the time (waterfall + rapid prototyping)
* Pros
  + Better and more communication between the clients and developers
    - Also better communication among the developers
  + Reduce the overall risk
* Cons
  + moving target because of more communication (perhaps too much communication…)
    - the client has more say about what they want
    - They are more likely to change what they want as the final product
    - Example: smartphones
      * Pro: Provide a lot of communication
      * Con: But too much distractions

Third life cycle model: **Evolution**

* start from something simple → complex
  + like a monkey → human // LOL
* Four views
  + We partition the system into a number of builds
  + Then use waterfall and rapid prototyping to construct each build
  + Let the client decide if he likes it
    - If he does, then work on the first partition
    - Continue if the client and the developer like each other
  + Continue building one partition at a time until you are finished
* Pros:
  + reduce overall risk on both sides
    - For the developer: developers build a part of the system and then get feedback instead of building the entire system at once
    - For client: if the developer is bad, client can just stop working with developer in the future
  + reduces number of errors
  + improves communication
* Cons:
  + not that popular method. Why?
  + More time consuming? Partially true, but not totally true
  + Fatal problem:
    - Hard to integrate
    - If you have a problem you need to solve in the later partitions, the first partitions that were built already may be useless
    - the build partition needs to be wisely done!!! or else you suffer.

Last life cycle method: **spiral** - niche life cycle model

* waterfall + rapid prototype + risk analysis for each phase
* Rigorous risk analysis
  + Requires math and statistics to do
  + How to identify all of the risks?
  + How to reduce the risks to a target threshold?
  + The error rate(?) for the risks should be < 10%
  + Example risk analysis for a thief: we have an expert thief who can steal without being detected. 99% success rate, 1% rate to be caught by police
    - What are the chances that he will get caught within 1 year
    - 0.99^365 = chance that he is successful every day
    - So 1 - 0.99^365 = chance that he does get caught within a year
      * x < 1, lim n → infinity, x^n = 0, where n is the number of days the thief continues to steal for
      * eventually he will get caught tho cuz x^n will become increasingly smaller
  + How to estimate the rate of success and risk?
    - This is very time consuming(?)
    - We need mathematicians and statisticians on the side to calculate the risks
    - This is why this method is usually only used for military systems, which have a lot of money and time
    - Military and systems requiring national security use this life cycle model
* pros:
  + reduce overall risk using statistics (not using communication, unlike previous life cycle models)
* cons:
  + Only good for military system where money and time is not a big deal
  + Utility is limited to military or national security

QUIZ: HE MIGHT ASK US TO DRAW USE-CASE DIAGRAM

Use-case diagram

* First UML
* Semi-formal
* Purpose is to identify
  + 1) all the users and their nature
  + 2) all possible services the system should provide
* Symbols
  + Box for system
  + stick figures for users (outside box)
  + ellipses for use cases (inside box)

Example: vending machine (PAST MIDTERM QUESTION)

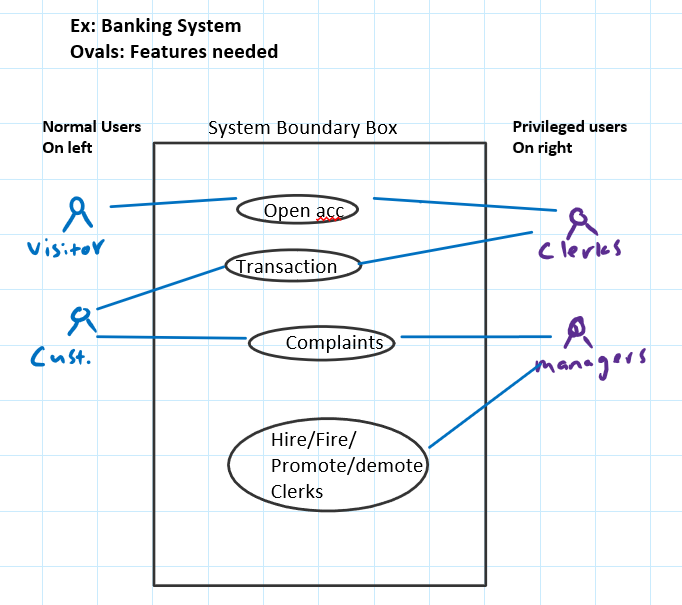
* Customers have to put in 30 cents to buy a drink
* If you use exactly 30 cents, you get a drink
* Number of tokens in this state diagram: how many possible coins do we have?
  + Types of coins: ~~Pennies,~~ nickels, dimes, quarters → 3 tokens
    - pennies make the state machine too complicated >:O
  + For each coin, draw a rectangle coming out of it
  + If there is another step, draw more rectangles coming out of the lowest level of rectangles
  + Continue until you have coins that sum up to 30 cents
    - if they ask for something that looks like an FSM but is UML, draw this weird state diagram

**10/08/2020** – **Lecture wk 6.0 - Techniques to describe systems in SE**

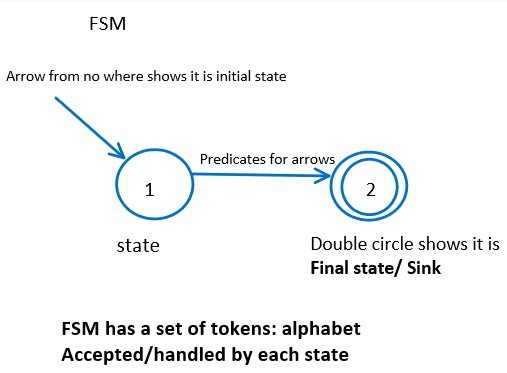
* **Possibly on the midterm:   
  3 different techniques to describe systems in SE: (System Description techniques)**
  1. **Informal:** Natural Language
     + **Ex:** English
     + **Pros:** Easy to create and understand
     + **Cons:** prone to ambiguity and mutual misunderstanding
  2. **Formal:** Math, formal language, codes, logic, finite state machine
     + **Ex:** Math
     + **Pros:** no ambiguity,
     + **Cons:** hard to create and understand
  3. **Semi-formal:** combination of informal and formal, diagrams, pseudocode
     + **Ex:** UML diagrams
     + **Pros:** Not as ambiguous as **NL**. Easier to create and understand than **Formal**.
     + **Cons:** room for ambiguity (math),

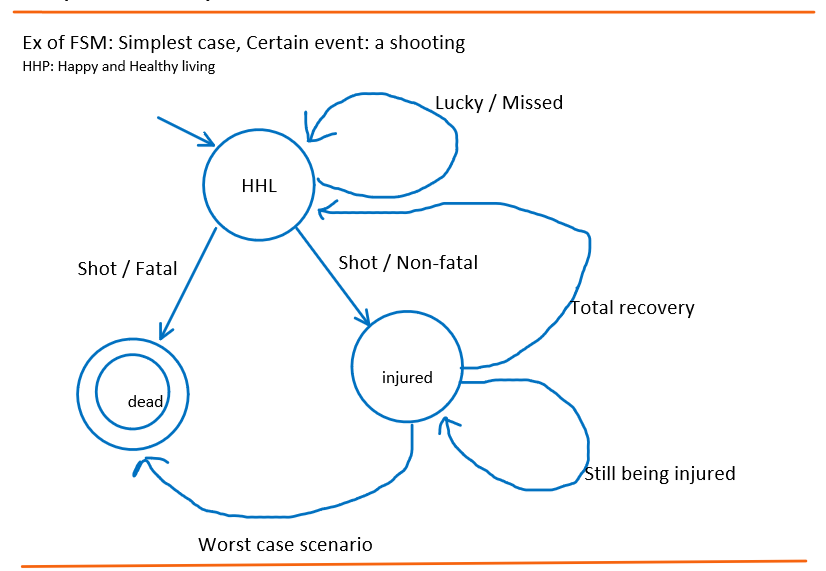
Unified Modeling Language (UML): is a general purpose developmental, modeling language in the field of SE that is intended to provide a standard way to visualize the design of a system.

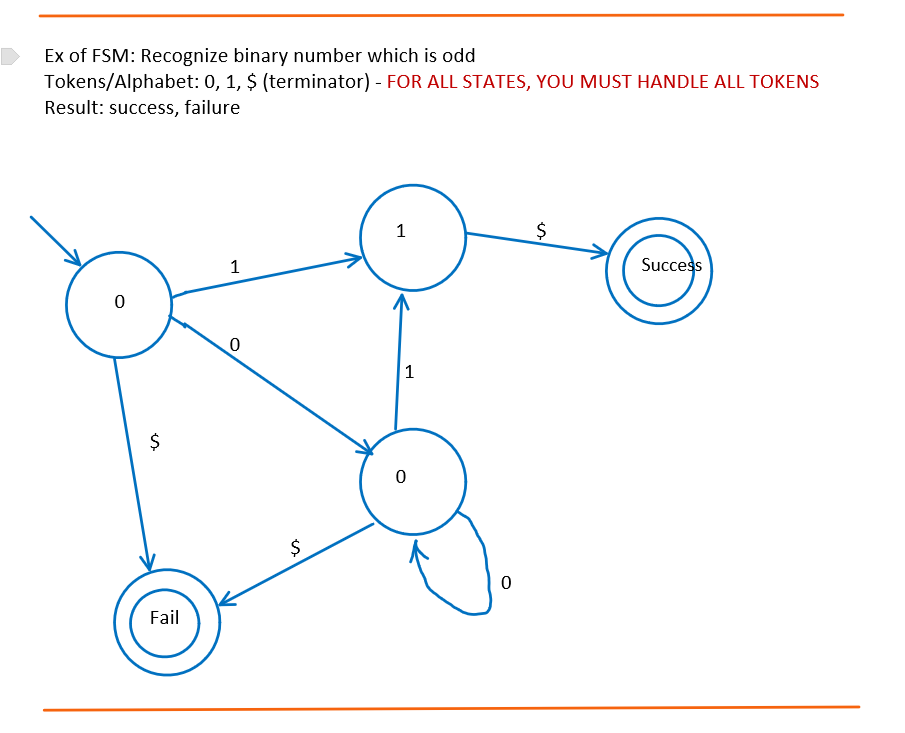
* **1st UML - Use Case Diagram:**
* **Use case diagram is a UML tool to reflect perspective based analysis**
* **Relation of system/subsystem with all different users**

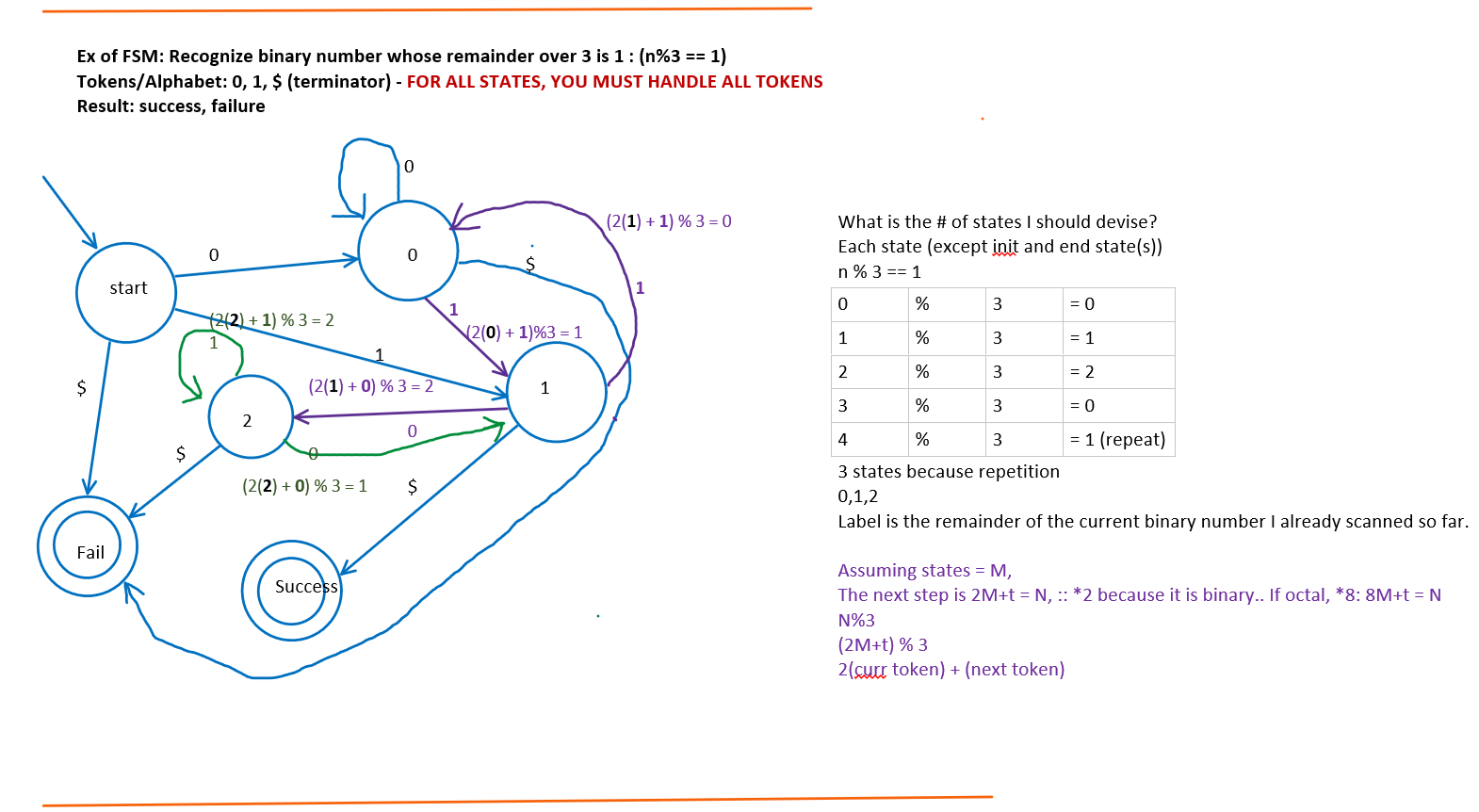


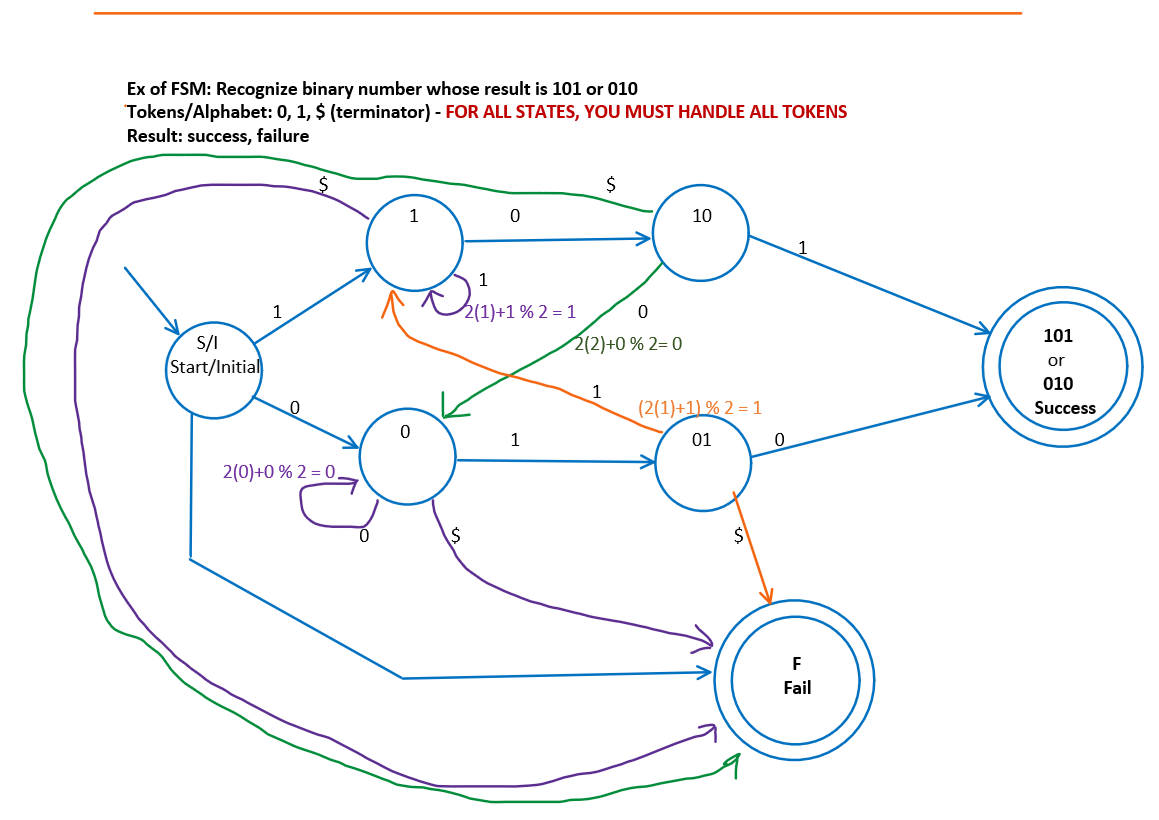
Finite State Machine (FSM) is a small part of turring machine, Generally used as a pattern recognizer





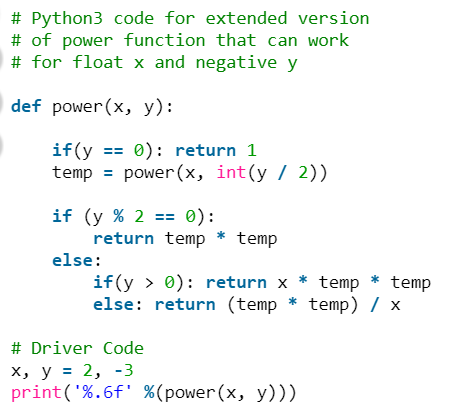






**10/13/2020** – **Lecture wk 7.0 – Quiz 1 review + UML cont.**

**Design an algorithm with sublinear time complexity. O(log n)**



pow(2,5) = 32

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Call | X | Y | Tmp | return |
| pow(2,5) | 2 | 5 | =  Tmp=power(2, 4) |  |
| pow(2,6) | 2 | 6 | = 8 | temp\*temp -> |
|  |  |  |  |  |

**Find adjacent paid A[i] != A[i+1] where A[0] != A[n-1].**

Solved via **BINARY SEARCH METHOD**

The loop invariant states that the first index and last index is not equal, thus, if you reduce the search from init 0, to mid, we can also see that the A[0] != A[mid]… then repeat til there are 2 elements left. This will result in the last 2 elements being pairs.

--------- end of review from quiz 1 ---------------

State chart is a UML and a close variant of FSM.   
State chart is about logic/algo

State Chart

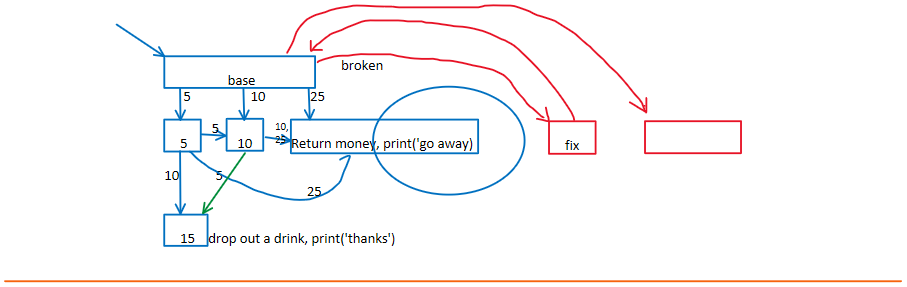


2) All are rectangles

3) Can go back to base state without any predicate (loop)

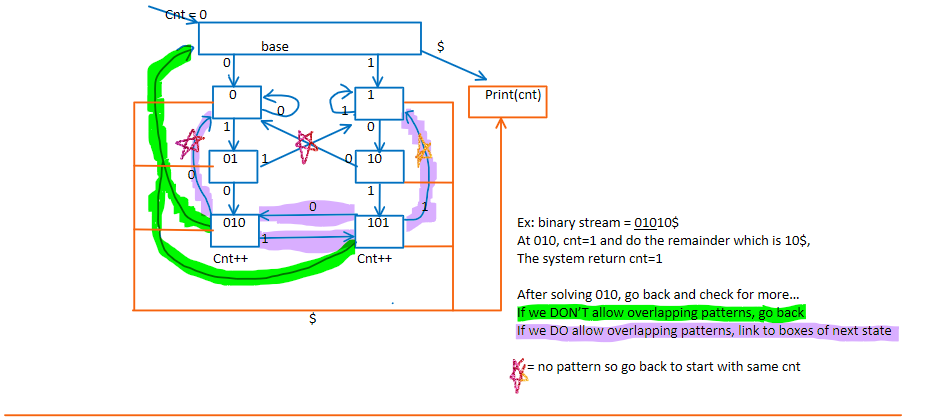


|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Ex: model a vending machine    15 cents for a drink, otherwise return money.  Only accept 5 cents, 10 cents, and 25 cents. |  |  |
|  |  | Fire, stolen, damaged |
|  |  |  |





Ex: count # of patterns 010, 101 in a binary stream, ended by $

-------------------------------end of state chart------------------------------

For data in UML: Entity/Relation (E/R) diagram (aka class diagram)

**EX:** Suppose we have class/DB def/schema: of students:

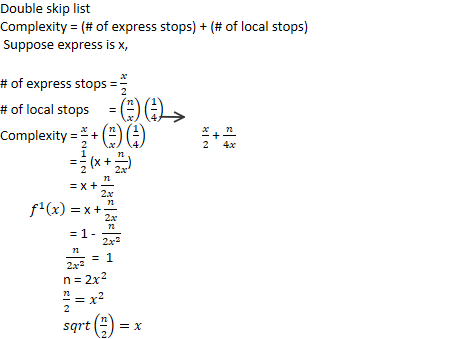
* Information entailed: ssn, name, dept, addr, gender, deptChair, college, coll\_pres

**Q:** Is the above correct? why/where? **WRONG due to…**

* Data duplication/redundancy:
  1. deptChair because deptChair depends on dept and each student with same dept will have same deptChair so its redundant.. –we should have a different data table for dept since redundant..   
     otherwise if deptChair changes, then we have to modify EVERY student’s information
  2. coll\_pres for same reasons

**10/15/2020** – **Lecture wk 7.1 – Quiz 1 review + UML cont.**

**Modeling** Instead of single link list, if a double link list is used, what is the optimal number of skip nodes (express stations) for N original nodes (local stations)? Justify your conclusion with mathematical reasoning details.



------ end of quiz 1 review -----------------------------------------------------

**Third Normal Form (3NF): normal form: industry standard for any data struct/ data base**

* **(3NF):** all non-key are directly determined by key/index
* <https://www.geeksforgeeks.org/third-normal-form-3nf/>

**Functional dependency**: suppose we have A,B,C,D. --- A depends on B.

* It is a constraint between two sets of attributes in a relation from a database. In other words, a functional dependency is a constraint between two keys
  1. **EX:** Student identification number (SIN) determines Name, Address, and birthdate.   
      **SIN, Course 🡪 date completed**: SIN and Course determine the date completed
  2. **EX:** **dept 🡪 deptChair**: Dept represents deptChair… deptChair does not represent dept

**Key or index:** for each **tuple/data-item**: they must be uniquely identified by **key/index**. Can be multiple attributes

* **Ex: key/index** can be… for example: ssn, name, addr, age, dept, college
  1. Which element would be key for this case?
     + SSN can determine name, addr, age, dept, college
* **Ex: suppose we have…a student roster**: ssn, courseID, grade

|  |  |  |
| --- | --- | --- |
| Ssn | courseID | grade |
| 1. | 322 | A+ |
| 2. | 322 | C- |
| 3. | 322 | B |
| 2. | 322 | B |

* Each row is called a **TUPLE**

Which attribute can be the key? (remember the same student may take the course again)  
- SSN + courseID   
In order to prevent design flaw, add 2 more attributes. Year and semester.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ssn | courseID | grade | year | semester |
| 1. | 322 | A+ | 2020 | Sp |
| 2. | 322 | C- | 2020 | Fa |
| 3. | 322 | B | 2020 | Sp |
| 2. | 322 | B | 2020 | Sp |

Which attribute can be the key? (remember the same student may take the course again)  
- SSN + courseID + year + semester   
- those keys are also the **3NF**

**10/20/2020** – **Lecture wk 8.0 – UML cont. .. E/R diagram 🡪 Entity/Relation diagram**

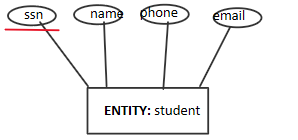
**E/R diagram** shows dependencies.

* **SEMI-FORMAL method** because there is natural language involved with diagram

The reason why E/R diagram is so important is because later on..

* each entity will be a table and database
* each relation will be a table and database

E/R diagram



**The underline shows that it is the key/index**

**Small ovals are attributes**

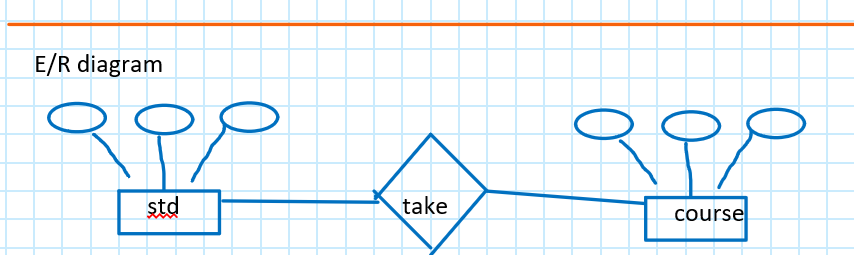
**Entity tells us the stable data store**

**Relation is the diamond which tells us the relation between entities**

**Links: lines - many to many relations**

**angular arrow -> many to 0 or 1**

**rounded arrow -) many to exactly 1**

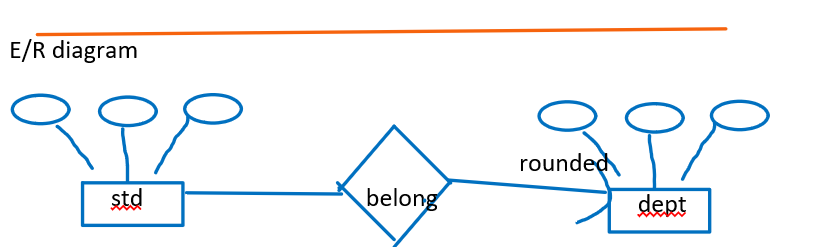


Many and each student may take many courses

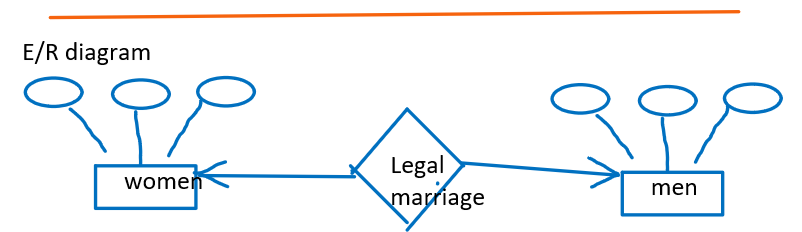
Entity: plural noun

Relation: verb

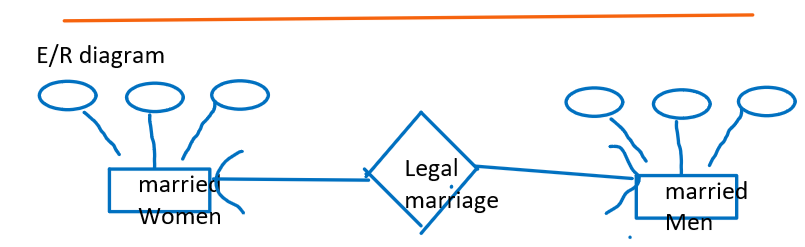
1. Check if have relation or not: name?



Many student and each may belong to a single dept

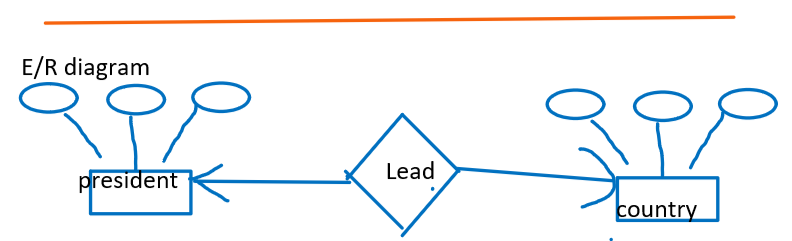


Angular arrows because just because women doesn’t get married doesn’t mean they are not women



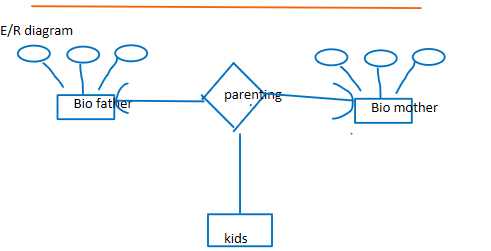
Rounded because 1 to 1,

Because when not married, they are not "married"



President can only be president of 1 country

But country doesn’t need president to be a country



3-way relation

(this table is incorrect)

(will see why in later example)



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | E/R diagram |  |  |
|  |  |  |
|  |  |  |

unary relation

Many to many

Many students may befriend many students



|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | E/R diagram |  |  |
|  |  |  |
|  |  |  |

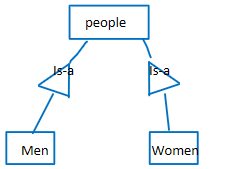
unary relation

Many to 0 or 1

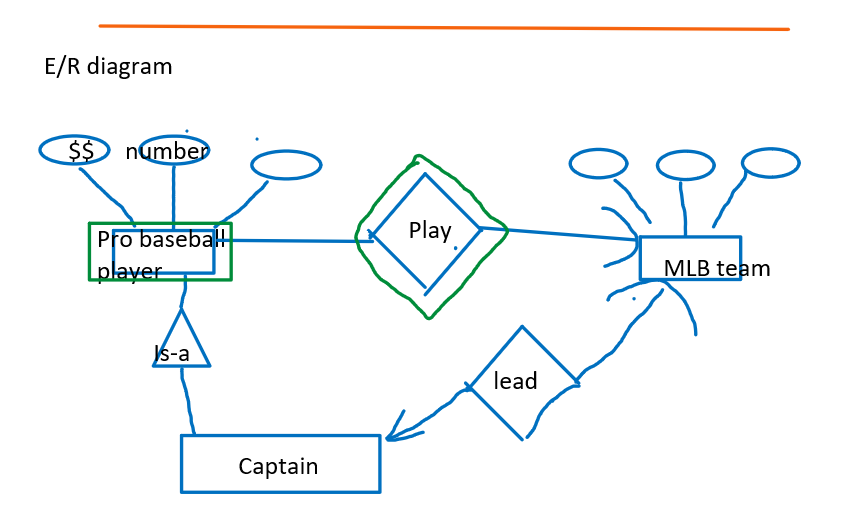
Many students may have 0 or 1 favorite friend



E/R diagram



Is-a is like inheritance



Pro player can only be pro player if in a team but can only be in a single team

But team doesn’t need pro player to be a team

A team is still a team without captain.

-------

How to index player?

Cannot find index/key from local attributes of player

So we call the player entity a "weak entity" and borrow from a different entity. Double edge the weak entity and the relation

Ex: Yankee number 10, or Giants number 10.

But there is ambiguity when mentioning Giants since there are multiple State giants.

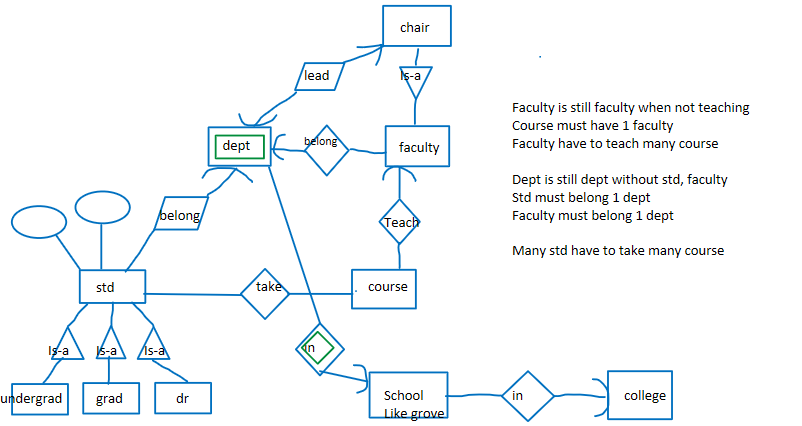
Ex: NY giants, NJ giants.

So, we may need to create another relation and entity such as "sport"

**10/22/2020** – **Lecture wk 8.1 – UML cont.**

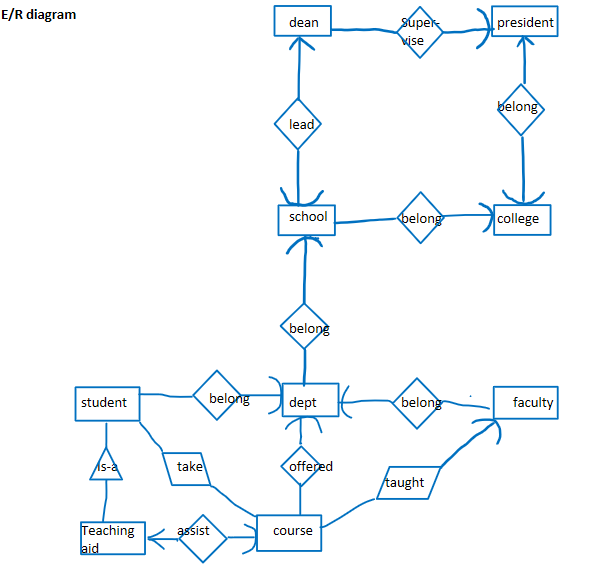
**E/R diagram**

* **Ex: ER diagram for college**



**Draw the ER diagram for college: (POSSIBLE ON MIDTERM)**

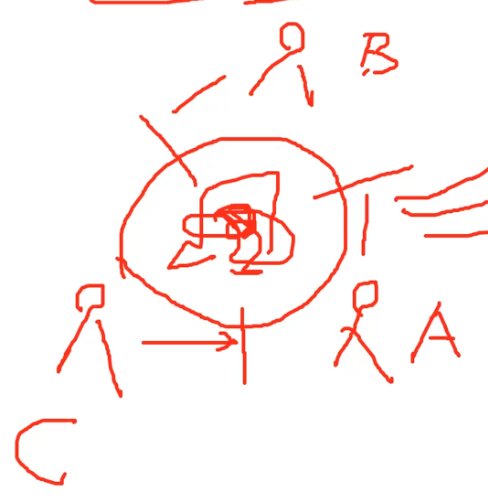
* **All students and instructors belong to one and only one department**
* **All courses are offered by one department**
* **A student can take several courses and one course and have many students**
* **Each course is taught by one instructor and assisted by one student as a teaching aid**
* **A department belongs to one school and a school belongs to one college**
* **Each school and college have one and only one dean and president respectively**
* **Deans are supervised by the president**

****

****

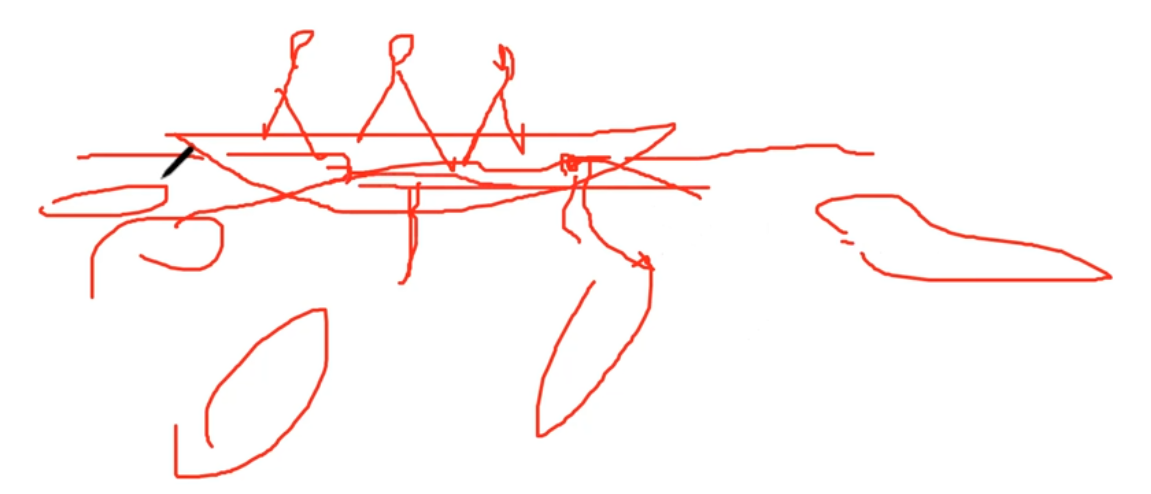
**3-banker paradox:**

* Suppose we have 3 bankers eating at the same table but each only have 1 chop stick each.
* No one will be able to eat with only 1 stick each. Thus, there is a **deadlock**



* How to better organize so someone can eat?
  1. Better organize: civil policy to avoid **deadlock**
  2. **policy:** formulate a policy so first come first serve as well as whoever has priority
  3. **IN CS:** Priority Queue: maybe weighted based on age, health, weight,
  4. **BUT** what if.. first person eats all the food or good parts?
  5. **Solution:** round robin / time sharing operating system
     + each person takes set amount of bites and changes turn
  6. [**https://en.wikipedia.org/wiki/Dining\_philosophers\_problem**](https://en.wikipedia.org/wiki/Dining_philosophers_problem)
* what if we have 4 chopsticks in total? Is it still a deadlock?
  1. No deadlock so we don’t need the policy.
  2. No because at least 1 person will have 2 chopsticks..

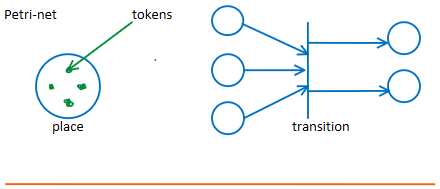
**Another ex:** Suppose there is a boat in the middle of ocean. There are 3 people on boat and there is leak in boat that can be resolved with 1 less person.



Who should leave the boat? (must be legal and civil)

* Just because age, health, weight, no one should be forced to jump.
* **Solution:** randomization (rock-paper-scissors) based on luck.

**Petri-net**: is a dynamic concept. Logic based. NOT UML. **Formal** method



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | After firing transition | |  |  |
|  |  |  |  |  |  |  |
|  |  |  | |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |  |

Firing:

In order for this transition to be fired,

* # of tokens >= # of links
* NEVER LINK 2 places or transitions directly

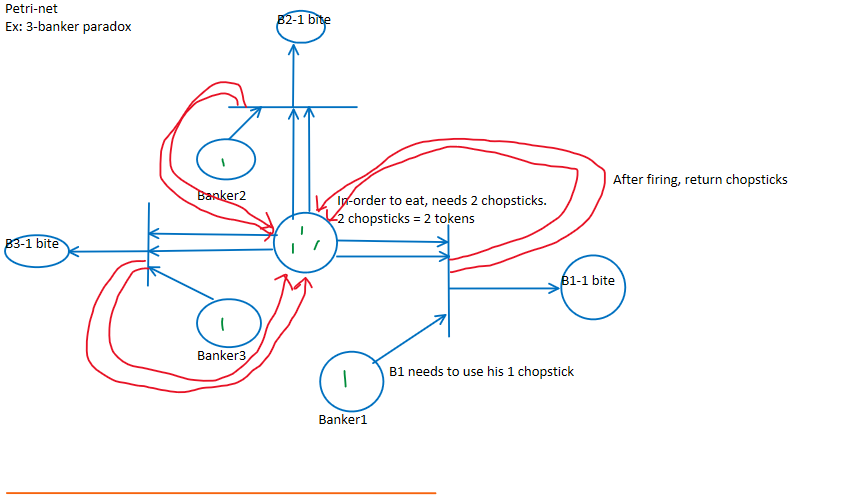
Recall: token = dots, links = arrows

If condition not met for firing: transition blocked

After firing,

* Remove # of tokens per # of links from input places, and add # of tokens per output links to output places





**10/27/2020** – **Lecture wk 9.0 – Petri-net**

**Petri-net is a formal method**

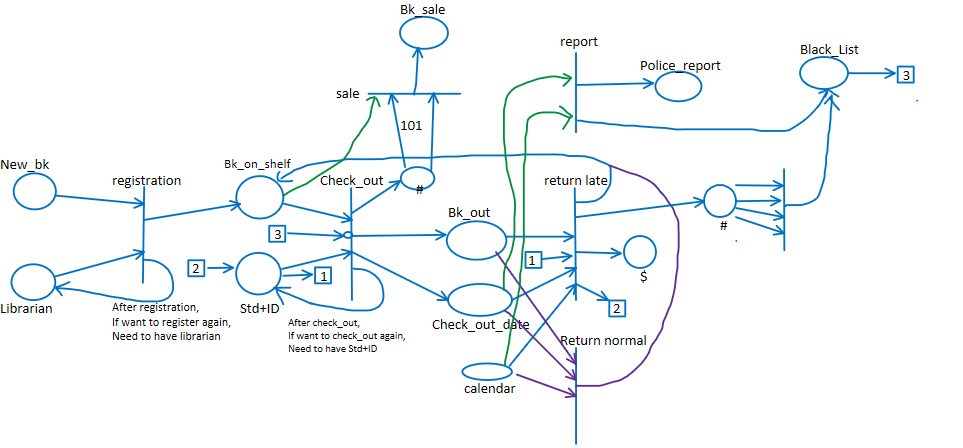
Petri-net

Ex: Library System (MOST LIKELY ON MIDTERM/FINAL)

1. New books must be registered then put on shelf
2. To checkout book, student must have id
3. Book returned later than deadline will be fined
4. Book returned later > 1yr will be reported to NYPD
5. Book checked out > 100 times will be put on sale
6. Students who returned later > 3 times OR reported will be put on black list

Places: nouns, transition: verbs/actions

Transitions are AND gates. All input places should have tokens # >= links

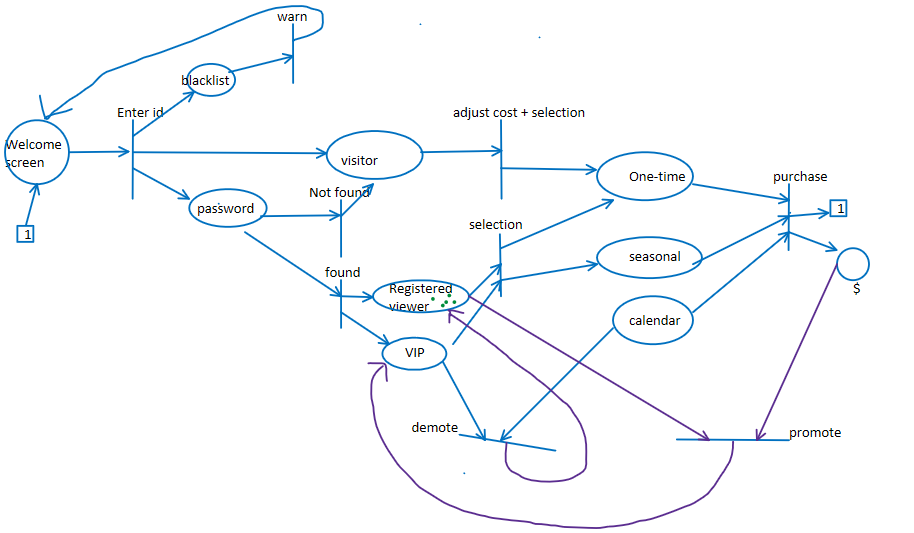




Petri-net

We are about to develop a **ticketing system for sports teams**.

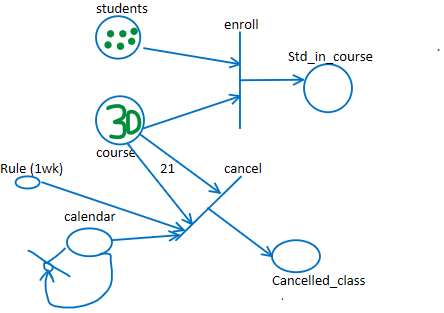
* There are three types of customers: visitors, registered viewers and VIPs, with different prices.
* To buy a ticket, a customer first types in his/her id from the welcome screen.
* If the id cannot be found in DB, the customer is assumed to be a visitor and go directly to the ticket order screen (visitor must pay more).
* If the id is on the blacklist, the system displays a warning message then returns to the welcome screen.
* If password does not match the one in the system, the customer is transferred to the ticket order sub system as a visitor.
* If the password matches, s/he enters the ticket order screen.
* In the ticket order screen, the customer is asked what type of ticket to be purchased.
* There are two types of tickets, one-time and seasonal.
* A visitor can only purchase the former, the other two can buy both with different prices.
* The customer may either choose one (if s/he is allowed) and pay for it as many times as s/he wants to or quit from the system.
* The system then returns to the welcome screen.
* A registered viewer who purchased more than $500 worth cumulatively will be promoted to a VIP.
* A VIP who hasn’t made any purchase in the last year is demoted to a registered viewer.



Petri-net

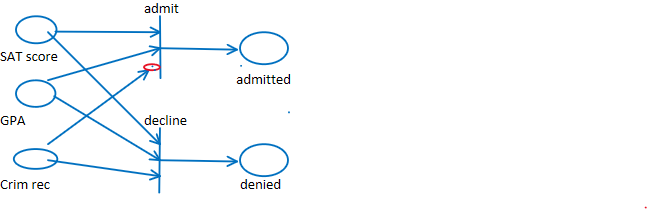
Ex: course system

1. A student can take at most 6 courses
2. A course can have at most 30 students
3. Any course (in the 1st week) with < 10 students enrolled will be cancelled



Petri-net

Ex: student trying to get into UNI, simple example

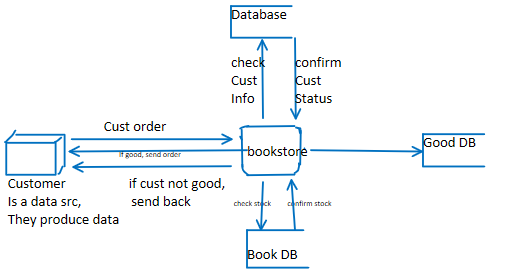




Data Flow Diagram ( ) (optional):

* Rounded rectangles are actions/procedures
* Open rectangle for database
* 3D rectangle for data src/destination
* Arrow for data

Ex: AMAZON company

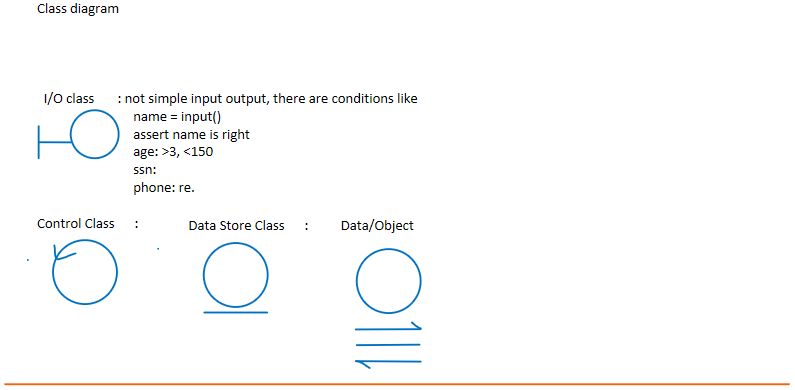


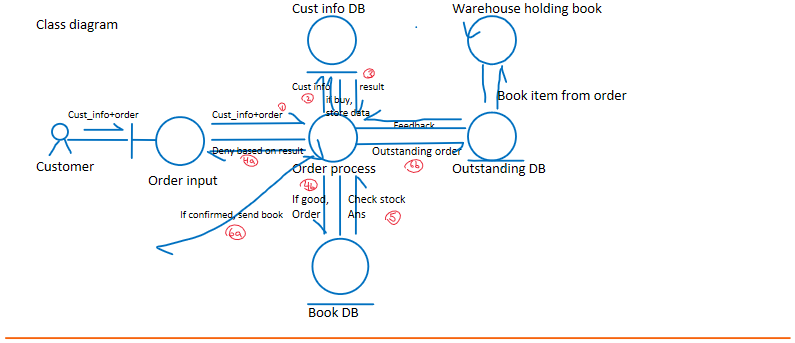


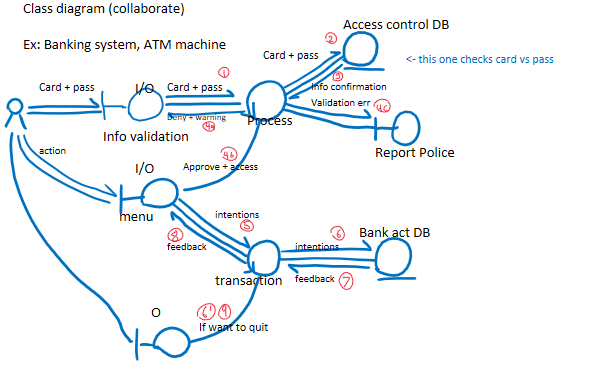
**10/29/2020** – **Lecture wk 9.1 – Data Flow Diagram (DFD) (AKA bubble graph) + class diagram (collaboration)**

**Class diagram:** based on classes. Object Oriented (OO). **Semi-Formal Method**

* **I/O class:** Circle with handle
* **Control Class:** circle with arrow, (the function/ the glue)
* **Data store class:** circle with underline
* **Data/Object:** circle with arrows and line underneath







Output: good bye screen

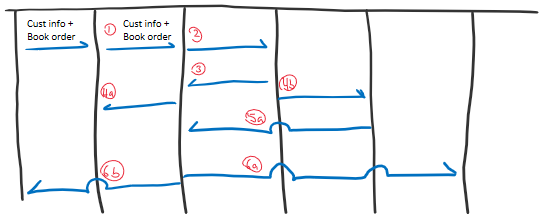
This is good example of collaboration class diagram because there are many classes collaborating together



Sequence class diagram: friendly to computer since it is in tabular form, each column can be represented with 0s and 1s

Sequence class diagram mostly equivalent as collaboration class diagram





**11/03/2020** – **Lecture wk 10.0 – semi review**

How to deal with large/big project?

* Software engineering

How to measure software quality?

* Fulfilling the specifications (MOST IMPORTANT)
* Understandability (for programmers)
* User-friendliness (for users)
* DRY (don’t repeat yourself)
* Robustness (resilient to errors / not easily crashing)
  1. Remind users to input with constraints like integer values
* Reliability (does what it needs to do)
* Efficiency (time + space + k-complexity)
* Security (hashing, data/system hacking prevention)

In System Software, **how to measure productivity?** : Planning, $$$$$, costs **(POSSIBLE CONCEPT QUESTION)**

1. **Measured by # of human hours (VERY POPULARLY USED)**
   * Pros: Easy to measure
   * Cons: easy to cheat, rewards bad and punishes good
2. **Measured by fixed $$ amount (overall price tag)**
   * Pros: Solved the cheating problem, easy
   * Cons: who measures $$ amount may not be reasonable at all. Almost random prices?
3. **Measured by passing through expert committee (can be done via machine learning: ensemble learning)**
   * Pros: ask a large group of experts, close to right price
   * Cons: extra costs, dependent on honesty/competency of experts
4. **Measured by # of lines [method to count lines]-> (wc – L) \* $10 (VERY POPULARLY USED)**
   * Pros: easy, no lawyers/arguments
   * Cons: easy to cheat (intentionally boost # of lines), encourages bad programming
5. **Measured by bidding (fair) – 2nd lowest price method (possibility of lower quality +** 
   * Pros: fairness, open, no lawsuits
   * Cons: risk to client due to low cost, hard to find bidders
6. **Measured by weighted sum (similar to method 4) (ex: for->$20, assignment->$5, rec->$30, System->$30) (Slightly better than method 4)**
   * Pros: easy, slightly better than method 4, reflected the effort more closely
   * Cons: easy to cheat

**3 aspects of complexity in software engineering.**

* **Time Complexity:** complexity which describes the amount of time it takes to run an algorithm. Can generally be found based on the amount of steps the program will take to finish. In software development, time complexity is highly sought after to be at its lowest which entails its efficiency.
* **Space Complexity:** complexity which describes the amount of memory space required to solve an instance of a program. It is the memory required to execute a program. In software development, space complexity is typically sought after to be at its lowest but sacrificed due to time complexity. (No Free Lunch)
* **Kolmogorov Complexity:** complexity which formalizes the concept of simplicity and understandability. In software development, k-complexity is expected as a normalcy from programmers for “simple and clean” methodology/philosophy when programming.

**Unified modeling Language (UML):**

* Is a modeling toolkit that guides the creation and notation of many types of diagrams.

**Sequence class diagram:** (should be semi-formal)

* Is a type of interaction diagram because it describes HOW and in what ORDER/flow a group of objects works together.
* A tabular visualization
* friendly to computer since it is in tabular form, columns can be represented in binary

**Data Flow Diagram:** (semi-formal)

* is concerned with the flow of information/data and processes in the system working with the information/data.
* Is about how data flow and how it will be used in system.
* Shows what kind of information will be input and/or output to/from the system and user.
* a diagram visualization

**3 different techniques to describe systems in SE: (System Description techniques)**

* 1. **Informal:** Natural Language
     + **Ex:** English
     + **Pros:** Easy to create and understand
     + **Cons:** prone to ambiguity and mutual misunderstanding
  2. **Formal:** Math, formal language, codes, logic, finite state machine
     + **Ex:** Math
     + **Pros:** no ambiguity,
     + **Cons:** hard to create and understand
  3. **Semi-formal:** combination of informal and formal, diagrams, pseudocode
     + **Ex:** UML diagrams
     + **Pros:** Not as ambiguous as **NL**. Easier to create and understand than **Formal**.
     + **Cons:** room for ambiguity (math),

**11/10/2020** – **Lecture wk 11.0 – After test review**

**11/12/2020** – **Lecture wk 11.1**

**OOA, OOD, OOP**

**Object Oriented Analysis:** analyze system using object as the center (More conceptual)

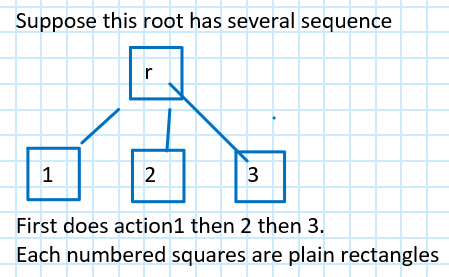
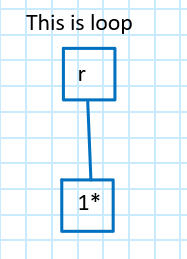
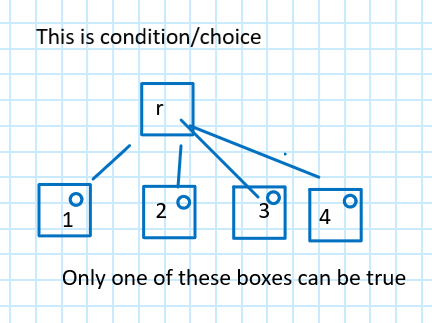
**Object Oriented Design:** Design the system around the object( uses logic/ algorithm and data.)

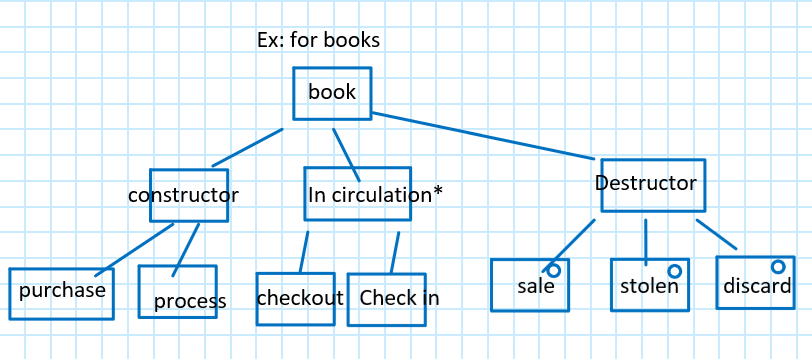
**Object Oriented Programming:** Easy to reuse/inheritance

**OOA + OOD = Jackson System Development (JSD)**

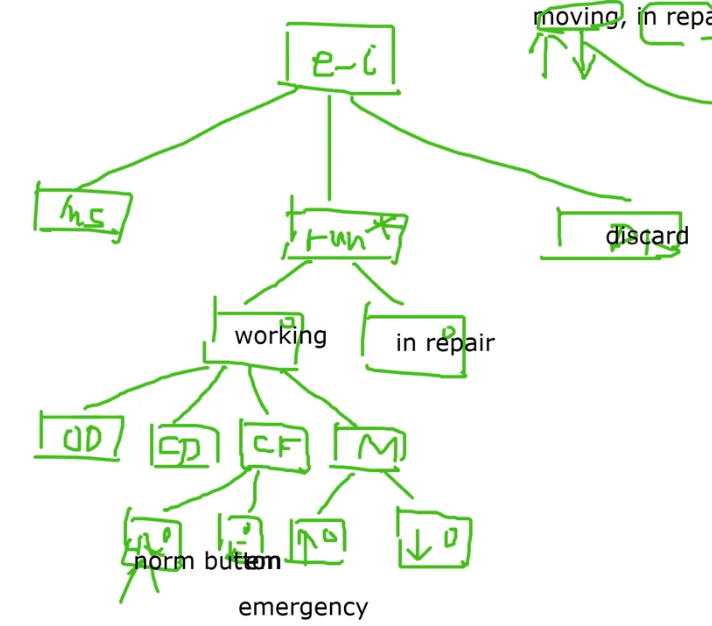
**Jackson System Development (JSD):**

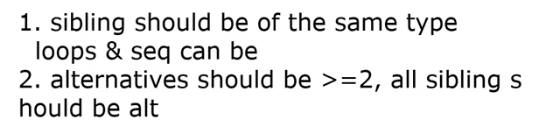
* Based around objects/entities
* Steps:
  1. Step1: identify the objects/entities and their services
     + Ex: lets say we want to analyze ccny library… we don’t want to analyze based on the system or its functions, but we want to identify the object (most important)
     + Whats the most important object in the library? : books, customers   
       (WE DON’T CARE FOR LIBRARIANS)
     + Books: what are the services/actions?:
       - Checkout, checkin, purchase, initial process books in(constructor), wait list, (destructor): on sale, stolen, discard.
     + Customer:
       - New std/card (constructor), (normal) checkout, checkin, pay fine, steal, renew, (destructor): blacklisted, ejected, transfer, graduate, disappear, cancel
  2. Step2: build a Jackson tree to organize the **services**
     + 3 types of actions for any programming language, conditional, loops, sequence



**Another ex of jackson tree for elevator control system: (combination of OOA and OOD)**

* **Elevator Car:**
  1. **Constructor:** install,
  2. **Destructor:** discard,
  3. **Normal Service:** Open door, Close door, moving, in-repair, stop, choose floor,
* 

 **… basically keep actions on same level.**

**Design phase:** high level design, low level design

* **High level design:**
  1. architecture design: # of modules/classes and their communications
     + Ex: what is the criteria to partition system into diff modules/classes
       - Just like project, who does which part
     + Ex: CPU design:
       - AND, OR, NOT gates
       - 1 not good idea is to partition by each gate.
       - Good idea is to divide by.. .alu team, criteria team…
     + Ex: school:
       - 1 not good idea is to partition by gender
       - Better idea is to partition by major department
  2. Divide and Conquer Philosophy:
     + Easy to conquer if split by some features which share similar features
  3. **Loose coupling and close cohesion** shows good system design
* **Low Level design:**
  1. Detailed design: logic/alg/data struct

**Another Software Quality measure:**

* **Coupling:** 
  1. How close/loose between 2 modules/sub-systems.
  2. A good system has loose couple because they have less similarities so its easier to integrate and separate.
* **Cohesion:**
  1. How close/loose those internal features are
  2. A good module has close cohesion. If its loose, why put into same module?

**11/17/2020** – **Lecture wk 12.0**

**System Theory:**

* Whats the relationship between **large(total) vs sum of parts**
  1. People working on the whole system vs separate people working on different partitions.
  2. Large is harder than sum of parts.
  3. A scenario at which sum of parts is better is a 4x100meter race at which they pass the batons in race

**Team Format:** democratic (all equal), too many channels/comms, no authority, leader(dictator): decide everything, total authority: fire/hire/pro/demote, ds, alg, nice leader/kind

**Group Cohesiveness:** Types of people in a team (The ideal team has a mix of all 3)

* **People person:** loves meetings and presentations and parties
* **Get the job done**
* **Perfectionist**

**Cohesiveness:** (Types of groups)

* **1. Very close:** members like family/siblings
  1. **Pros:** happy, ego-less programming, help each other
  2. **Cons:** Cannot be critical so will lead to bad system, quality suffer, dangerous to higher-ups
* **2.** **Very cold:** enemies to each other
  1. **Pros:** randomize anchor so have to work hard not to get attacked
  2. **Cons:** feel bad, waste lots of efforts to guard against each other.
* So whats the correct cohesive group?:
  1. Not too close/loose
  2. Be professional… after work, don’t grab beer

**11/19/2020** – **Lecture wk 12.1**

**More into High level design…**

**Cohesion:** Relation inside a module (ideally as close as possible)

* **7 different levels:**
  1. **Coinsidental** (trash bin)
     + Module: no chance to reuse at all.
       - Ex: some module which…: register a student, speeding, arrest a thief
       - It is COINSIDENTAL to put these functions within a module as the chance of using all functions is too rare.
  2. **Logical** (share some physical features) (classified trash bin)
     + - Ex: take 322, eng 101, drop ps202,
       - The classes are irrelevant but they share a common ground of them
       - Not reusable (bad, not acceptable)
       - (logical bin)
  3. **Temporal**:
     + Actions happen by time sequence but not required
       - Ex: go into subway, buy ticket, slide the ticket, go in platform.
  4. **Procedural**:
     + Early actions are required for later actions.
       - Ex: you have to finish all credits to graduate. You cannot graduate without finishing all credits.
     + **Procedural** is like cause & effect. Beyond reasonable doubt.
  5. **Communication**: many actions over the same common data
     + - Ex: login cunyfirst DB, change my address in DB, change pswd in DB, out of the DB.
  6. **6a)** **Functional**: one action (FP)   
     **6b)** **informational**: (one well designed class), 3NF, (OO)

**Ex of cohesion level:**

* Cheating in the final exam, get F as the course grade.
  + - This is **TEMPORAL** because the chance of this happening is quite high but not overwhelming
* Take 221, take 322, take 479 (IF 479 has pre-req of 322)
  + - This is **PROCEDURAL** because you cannot take 479 without take 322
* Take 221, take 322, take 479 (IF 479 DOES NOT have pre-req)
  + - This is **TEMPORTAL**
* Someone with high IQ, has lots of money in life.
  + - This is **TEMPORAL** because the chance of this happening is quite high but not overwhelming
* Type in the resume using word, print out a picture, scan in a receipt
  + - This is **Logical** because they are office related tasks.

**What is the gold standard to tell what is cause and effect?? And what is correlation?**

* Double blind testing.

**Testing**:

* 1. Active testing(db, randomization)(cause and effect): setup, participation design, experimentation.
  2. Passive testing(correlation): by observation… , Passive testing is generally performed if active testing is not available.
     + Ex: married people live longer by X years
  3. A/B testing:
     + Testing on certain percentage user unknowingly to see how they like it.
     + Generally only performed by large corporations

**11/19/2020** – **Lecture wk 13.0**

**Coupling: (between 2 modules)**

* **Different levels from worst to best**
  1. **Content:** two modules have access to private info
     + **Ex:** 2 classes: cl1 can access/revise private member variable in cl2
     + **Ex:** in C++/C#, there is a method, “friend” which allows this.
     + This is very bad practice. No privacy, lost control of your own module
  2. **Common:** 2 modules share the same global variable
     + **Ex:**1 module changes time and 1 module reacts to time
  3. **Control Coupling:** module 1 calls module 2 and uses the return data from module 1
     + only need to know input and output
  4. **Stamp:** two modules have communication but not all of them are useful
     + **Ex:** module1 sends select data over to module2 and module2 evaluates with the data
     + **Ex:** from numpy import random, array, empty
  5. **Data:** all information is useful
     + **Ex:** import numpy
  6. **No Direct relation:** 2 independent modules uses the same separate module 3
     + **Ex:** module1 and module2 don’t use each other but they both use module 3

**When asked question about finding all possible coupling relationships, draw a matrix/table**

**Ex:** M1: has some code… and has L1   
 M2: has some code… and has “goto L1”

**This is Content because M2 has access to M1**

**Low Level Design:**

* Data structure, alg, complexity
* By default: as efficient as possible.
* Understandability

**12/1/2020** – **Lecture wk 14.0**

**Bottle question: Log(n) - (radix sort)**

* 1000 bottles, 1 poison, how to find how many tests needed? Log(1000) = 10. Need 10

**How to find angle between hour and minute hands on a clock question:**

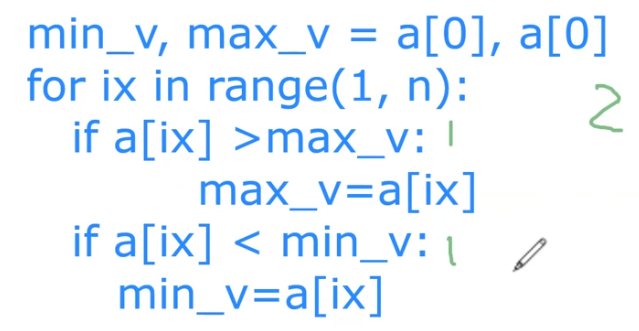
* Divide and conquer:
  + - Calculate the angle from 0/12(top of clock) to the minute and hour hand and subtract each other
    - Angle between minute hand and top of clock = 360/60 = 6 M
    - Angle between hour hand and top of clock = (360/12)\*(h+m/60)

**Algo/math question:**

* **Eval the max and min val for any array.**
  + - **Can this be done in sublinear time?**
    - **Yes, by contradiction….** 
      * **Euclid proof of the number of prime numbers is infinite**
      * Given in lecture 14.0 - 25minutes in

**Time complexity problem:**

**Given…**



**What is the time complexity??**

**Currently we have… every element, 2 comparisons = O(2n)**

**Can we make this code more efficient?? Yes..**

* **Instead of comparing min and max with every element,** 
  1. **compare A[2i] with A[2i+1]… so we are traversing by 2 increments, A[1] and A[2] then A[3] and A[4]**
  2. **compare smaller of these 2 with min**
  3. **compare larger with max**
* **This causes every iteration having 3 comparison but n/2 iterations… Thus, 3\*n/2 = O(1.5n)**

**Data Compression:** The reduction of size of data

* Which image format is the most default used?
  1. JPEG: Joint pic expert group (industry standard)
     + The power house behind JPEG is the Fourier transform.
* Different types of compression:
  1. Lossless compression:
  2. Lossy compression:
* Which video format is the most default used?
  1. Mpeg4 / MP4
* Music: 1D vector
* Image: 2D matrix
* Video: 3D medscan; 3D tensor;
* 4D: (3D + 1D tme): 4D tensor
* Music/sound compression: disc, mpeg3/mp3 (1D)
* Torch: tensor facebook
* Tensorflow: tensor
* **The fundamental reason why we can compress data is due to stat redundancy**

**12/3/2020** – **Lecture wk 14.1**

**Legal and Security responsibility with increasing and more power given to programmers**

**Pan-computing, universal digitizations**

**Military:**

* **ITAR-control:** must be a US citizen or green card holder for entry

**Testing:**

* There are no correct system:
  + - Incomplete induction
* Math/complete induction
  + - Enumerate lots of cases
      * 1+2+…+n=n(n+1)/2
* Always keep in mind and write down:
  + - “for this system, so far we didn’t find any problem; if you do, let us know, we’ll fix it at our best.”

**Testing vs debugging:**

* **Testing is by others that don’t know the testee**
* **Debugging is done by programmer (part of the coding phase)**

**Black box testing vs White-box testing:**

* 1. **Black box testing:**
     + We treat the system as a black box, we cannot see.
     + We **only** know the input and output
     + Check if input and output correspond with each other.
  2. **White-box testing: (glass-box testing) (structural testing)**
     + We can see inside the system as well as the input and output
* Suppose we have a function “int func1(int val, bool G):”
  + - For black-box testing, we can only see what we input and output.
    - For white-box testing, we see everything
      * For white-box, we have to ensure exhaustively all cases/branches are covered
* The white-box testing is generally done by the programmer while they are programming
* The black-box testing is generally used by testers (because no one likes reading others’ codes)

**Observation vs Experimentation testing:**

* 1. **Passive Observation**
     + Clean hands
  2. **Active Experimentation (necessary)**
     + Dirty hands

**Static vs Dynamic testing:**

* **Static is more reading**
* **Dynamic is more running. (collects stats)**

**Top-down vs Bottom-up testing:**

* **Assume a tree.** 
  1. **Top-down:**
     + **Tests level by level starting from root going downwards.**
     + **To test, write simulation stubs**
  2. **Bottom-up:**
     + **Tests level by level starting from**
     + **To test, write simulation driver**
* **Top-down testing is more generally used to ensure the foundation is steady. (the overall feature is most important)**