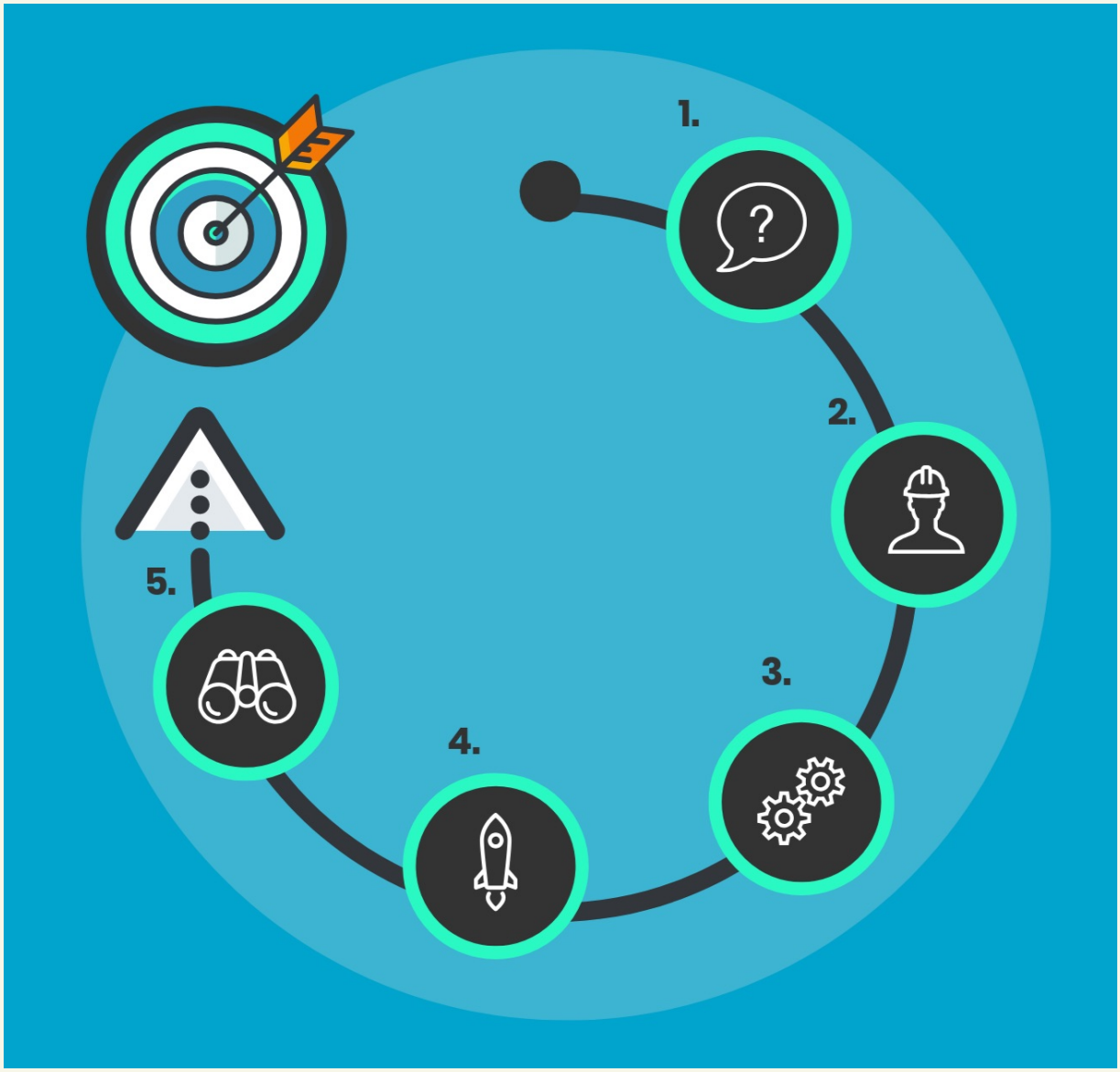
Software Engineering



Software Engineering

→ application of engineering to develop quality software in a cost-effective way to meet the target objective (software modeling, software development methodology, software security, software reuse, software testing, software economics → estimate resources & cost)

→ software → computer programs & associated documentation such as requirements, design models & user manuals (generic → sold to range of customer, customised)

→ software process : a) communication → pdj initialisation, requirements capture

b) planning → plan technical tasks, resources, work pdts, schedule & analyse risks

c) modeling → requirements analysis, design

d) construction → coding, testing

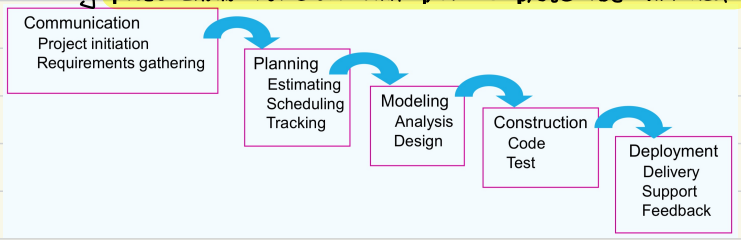
e) deployment → software delivery, evaluation & feedback

Prescriptive Process Models

1. Waterfall Model

→ five phases arranged in sequence, result of each phase is one/more documents that are approved

→ following phase should not start until previous phase has finished



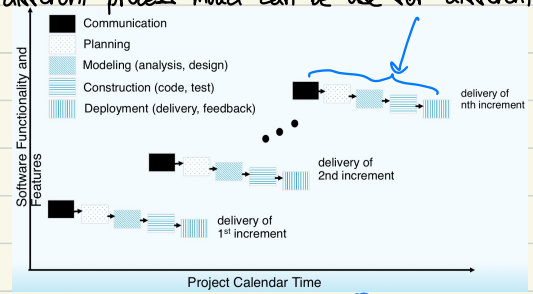
→ suitable situations: ① well defined requirements, ② adequate resources, pdj teams have ③ good expertise/experience on system, systems ④ not too large/complex, ⑤ no urgent need on any part of system

2. The Incremental Model

→ rather than deliver the system as single delivery → development & delivery broken down into increments

→ each increment delivering part of required functionality → first increment (core pdt address basic requirements), later increments (address modifications & additional functions & features)

→ different process model can be use for different increment



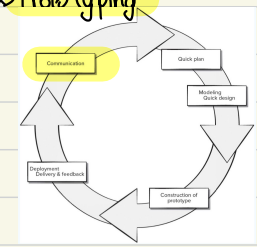
→ suitable situations : ① requirements well-defined, ② sufficient experience, ③ insufficient resources/manpower, ④ some critical requirements must be delivered asap

3. Evolutionary models (when user not sure about requirement)

→ they explicitly design to accomodate pdt that evolves over time → enable software engineer to develop increasingly more complete, accurate/better version of software

→ Two sub-models:

a) Prototyping



→ begins with communication, engineer & customer meet & define overall objectives for software

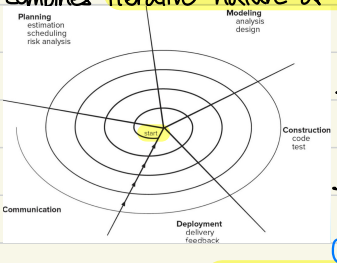
→ prototype deployed → evaluated by customer → feedback refine requirements for software

→ iteration occurs as prototype tuned to satisfy need of customer

↳ suitable situations: ① unclear requirements, intensive ② user-system interactions, ③ little experience, ④ small systems

b) The Spiral

↳ combines iterative nature of prototyping & controlled & systematic aspects of waterfall method



→ each loop in spiral represents an iteration in the process

→ an anchor point/milestone is defined at the end of each phase where a combi of work product & conditions are noted

→ risk are explicitly assessed & resolved throughout process

↳ suitable situations: ① large-scale systems, ② unclear requirements, ③ complex requirements, ④ high risk, ⑤ insufficient experience

Agile Software Development

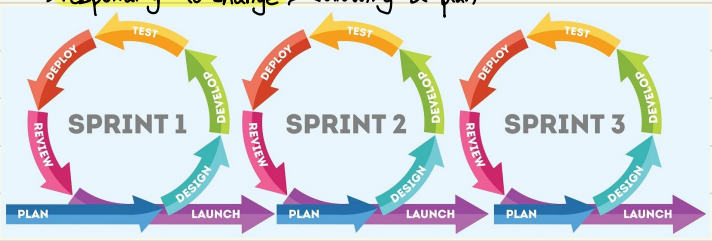
↳ a set of guidelines to address some issues in use of prescriptive software process models

a) individuals & interactions > processes & tools

b) working software > comprehensive documentation

c) customer collaboration > contract negotiation

d) responding to change > following a plan



- Abstraction → problem solving activity → focus on essential aspects & ignore others in problem
- Modeling → develop a machine independent solution by focusing on important aspect of problem, uses concepts & notations that have precise meaning (math can be used)
- Algorithm operating on its required data structure → forms logical model for computer program

Requirements for software

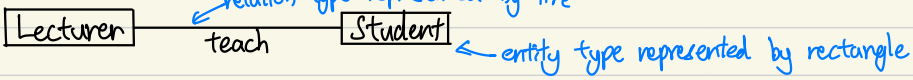
↳ classified into 2 aspects:

a) data → information to deal with (modeling of data/data modeling → technique: Entity-Relationship (ER) modeling

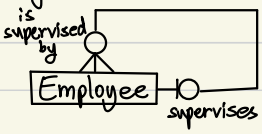
b) processes → tasks to perform (modeling of processes/process/behavioral modeling → technique: Data Flow Diagram (DFD) technique

Enity-Relationship (ER) Modeling

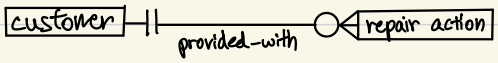
- describe the data in a system by entity types & relationship bw them by relationship types in ER diagram
- Entity type → specifies a type of things that have identical properties in target system
  - e.g. entity type (student), attribute of entity type (student address), instance of entity type (Michael Lee)
- Relationship type → specifies a type of interactions between things, these things are modeled by entity types
  - e.g. relationship type (lecturers teach students), instance of relationship type (Dr K.C. Lee teaches OO programming)



ER diagram

1. 

- Each (always each) employee supervises none or many employees
- Each employee is supervised by none or one employee / at least one employee

2. 

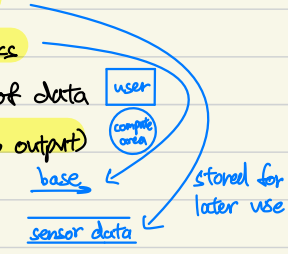
- each customer is provided with none or many repair actions
- each repair action is provided for one customer

pairs of cardinalities:
- || exactly one
- |< one or many
- o| zero or one
- o< zero or many

cardinalities:
- | one
- < many
- o zero

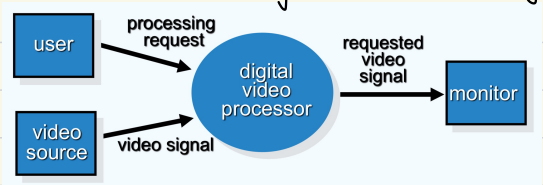
Data Flow Diagram (DFD) Technique (for requirement analysis only)

- model computer-based system from highest to lowest level using basic concept of information transformer, DFD describe processes & data they act on
- Data classified into two types: a) persistent data → stored in data repositories
  b) intermediate data → passed between sub-tasks
- DFD drawn using 4 basic concepts: a) External entity → producer/consumer of data
  b) Process → data transformer (Δ input to output)
  c) Data Flow → intermediate data involved
  d) Data Stores → a data repository kept

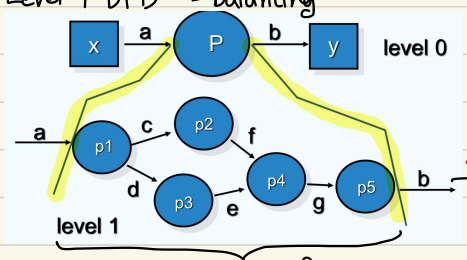


- always begin with a context level diagram (level 0)

E.g. Level 0 DFD → whole system shown as a single process



Level 1 DFD → balancing



→ in & out need to match level 0

zoom in on P

Data Dictionary

↳ Data specification → specification is included for each data flow & data store. To describe each data flow & store by combination of data using four basic structures:

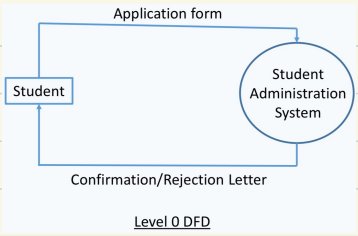
a) sequential combi of data items → Name = Surname + Other Name + (Middle Name) → notation → () → option combi of data item

b) combining multiple values of data item → Past Appointments = {Appointment}* → * → to indicate combi of multiple values of data item A tgt

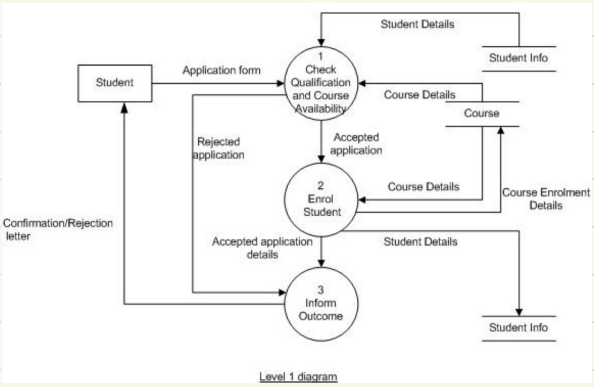
c) combining one from a number of data items → Person Identification = [Ic No | Fin No] → [..|..|..|..] → combi of one data item from data items

↳ process specification → specification is included for each lowest-level process (higher-level processes not specified). Can be shown using: narrative, pseudocode → program design language (PDL), equations, tables, diagrams etc.

E.g. Students sends in an application form containing their personal details & their desired course. The university checks that the course is available & that student has necessary academic qualifications. If course available & suitable, student enrolled in course & uni send confirmation letter. If course unavailable, student sent rejection letter. Draw a DFD diagram.



Level 0 DFD



Level 1 diagram

Structural Modeling (UML Modeling I)

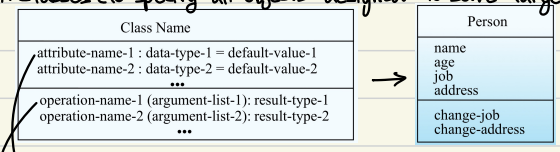
→ unified modeling language (graphs & models)

↳ modeling the structure of information & processes : basic UML modeling technique, class diagram

Class Diagram

↳ extension of ER modeling → class diagram describes both processes & data tgt instead of data only using object

1. Classes (to specify all objects designed to solve target problem)



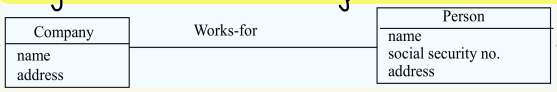
attribute describe its state (condition of an object at any moment, e.g. primary (for student) school/kindergarden)

operation describe its behavior (what object do)

2. Relationship types (to specify the types of interactions bw the above objects)

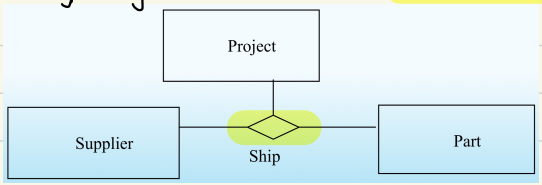
a) Association → represents interactions bw different objects (relationship)

→ binary association (line connecting two associated classes)



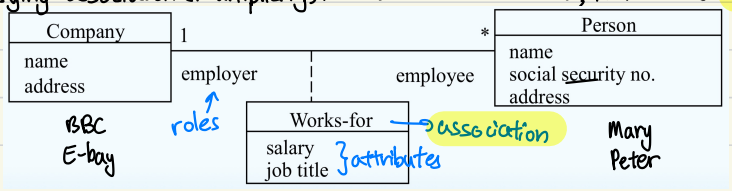
– person works for company

→ ternary & higher order association (diamond connected to all associated classes)



→ specifying association (multiplicity): 6..8 → from 6 to 8, 1..1 (simplify → 1), * → 0 to many, 0..1 → 0 to 1

E.g.



instance examples: (a group)

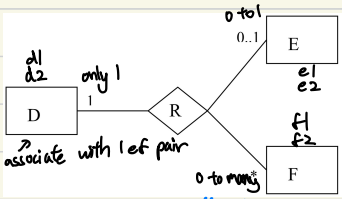
{(Mary, BBC), (Peter, BBC)} ✓

{(Mary, BBC), (Mary, E-bay)} ×

– Each person works for one company

– Zero to many people work for each company.

E.g.



→ write down possible pairs (e1f1, e1f2, e2f1, e2f2) for d

→ write down pairs for e (d1f1, d1f2, d2f1, d2f2) → e only can form max with one pair

→ write down pairs for f (d1e1, d1e2, d2e1, d2e2) → doesn't matter since is 0-many

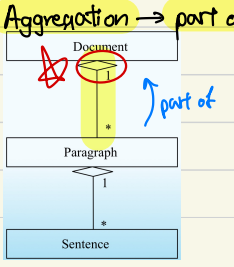
→ compare to given instance

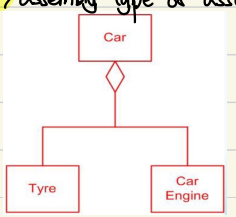
Is {(d2, e1, f1), (d2, e1, f2), (d2, e2, f1), (d1, e2, f2)} a possible set of R instances?

(different pair)

↳ e1 already associated with d2f1, e2 cannot

b) Aggregation → part of / assembly type of association



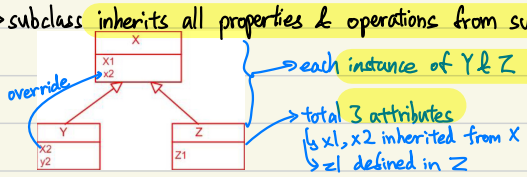


– A personal computer has a CPU

– Book titled "C programming for beginner" has 10 chapters

} instance of aggregation

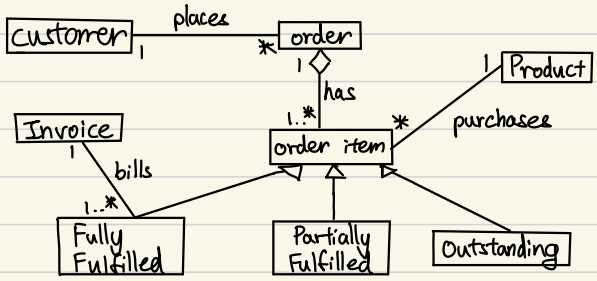
c) Generalisation → type & subtype kind of relationship
- → each object/instance of subclass is an object/instance of super class
- → subclass inherits all properties & operations from super-class & can override them by redefinition if needed



— Employees classified into daily-rated & monthly-rated

☆ — First, check if its generalisation, then aggregation, last association

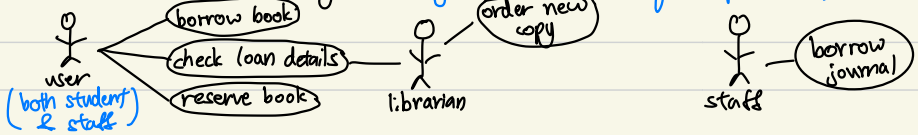
☆ E.g. [Customer place orders with company to purchase products that are sold by company.] A customer may have none/multiple orders placed. Each order is always placed by one customer. [It may have single/multiple order items.] Each item is to purchase one product. An order item could be fully fulfilled, partially fulfilled or outstanding (completely unfulfilled). [Periodically, an invoice is raised to bill customer for order items that have been fulfilled.]



Behavioral Modeling (UML Modeling II)
→ model occurrences & details of processes: use-case technique, statechart diagram & sequence diagram, activity diagram

☆ Use-Case Technique
- → basic method to specify requirements in Object-Oriented approaches
- → template: a) use-case: collection of user scenarios for usage of system (represented by ellipse)
  - b) primary actor: main user (represented by human symbol), other actor (represented by rectangle)
  - c) goal in context: objective
  - d) preconditions: property must be satisfied before use-case
  - e) trigger: action/event to start a use of use-case
  - f) scenario: main interactions bw system & actor in sequence (system-user interaction represented by line connecting a use-case & actor)
  - g) exceptions: errors & other behaviors

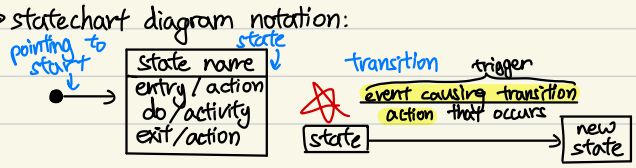
E.g. The library stores various items that can be borrowed, including book & journals. Books can be borrowed by both staff & students, but only staff members can borrow journals. When users borrow a book, their loan details are checked to ensure that they have no overdue books on loan & have not already borrow max permitted books. Draw use-case diagram. (Key criteria: consistency, completeness, correctness)

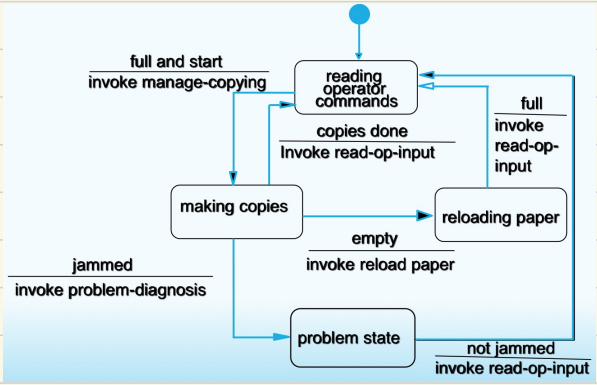


Statechart Diagram

→ used in UML to represent state machine models → model the behaviour of system in response to external & internal events

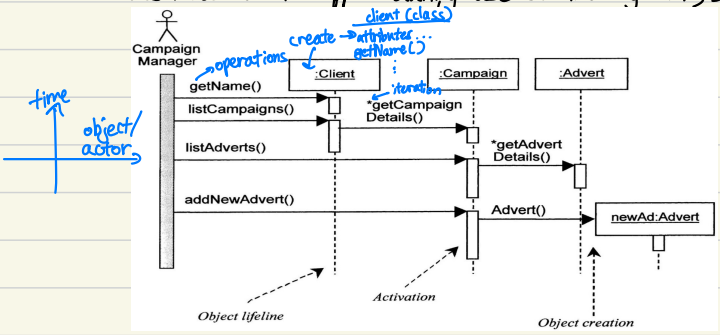
→ statechart diagram notation:





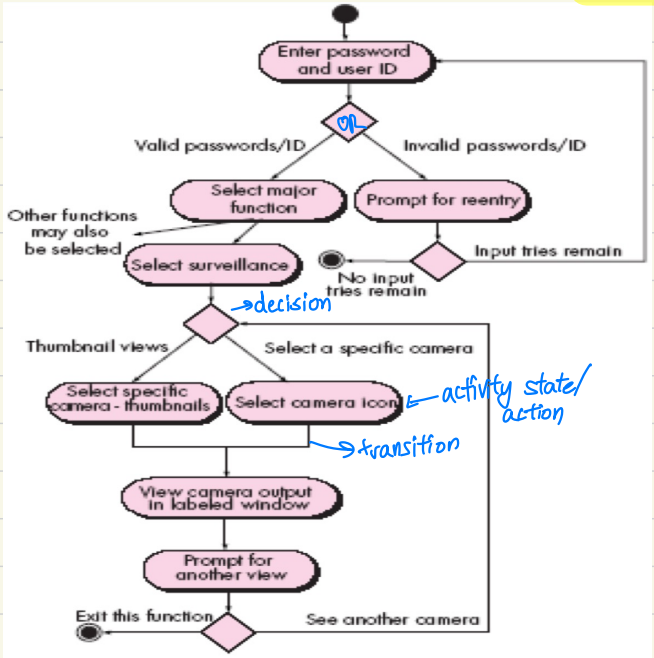
Sequence Diagram (used for behavioral modeling)

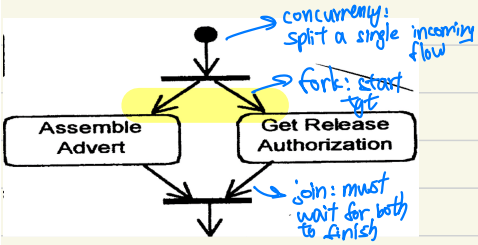
↳ describes what happen during use of the system, shows how use-case can be implemented



Activity Diagram (procedural flow)

↳ shows flow of actions (can be used to describe use-case, logic of operation, entire system)





Software Testing

- process of exercising a program with specific intent of finding errors prior to delivery to end user
- for accuracy of software → perform Verification (with given input, check if actual output is expected) & Validation (assessing the degree to which software system fulfill requirements) → V&V
- developer → understand system → test "gently", driven by delivery
- independent tester → learn about system → attempt to break it, driven by quality

Testing Techniques

1. Black-Box Testing (testing based solely on analysis of requirement, no internal testing)
   - Equivalence Class Testing (partition input space into small no. of equivalence classes)

     E.g. A program computes the monthly commissions for salesmen in a company according to following table based on his/her monthly sales total. Monthly sales total should never be in negative value

Monthly Sales Total (MST)	Monthly Commission (MC)
0 ≤ MST < 200,000	0% of Monthly Sales Total
200,000 ≤ MST < 1,000,000	1% of Monthly Sales Total
MST ≥ 1,000,000	2% of Monthly Sales Total

     Equivalence classes for MST: $(-\infty, 0)$; $[0, 200\,000)$; $[200\,000, 1000\,000)$; $[1\,000\,000, +\infty)$; non-digits

     Test Cases: ① Input: MST = -100; Output: Invalid Input
     ② Input: MST = 100 000; Output MC = 0
     ③ Input: MST = 300 000; Output: MC = 3 000
     ④ Input: MST = 2 000 000; Output: MC = 40 000
     ⑤ Input: MST = $7000; Output: Invalid Input

   - Boundary Value Testing (errors tend to occur near extreme values of input)

     E.g. Same example as above:

Monthly Sales Total (MST)	
0 ≤ MST < 200,000	→ 0, -100, 200 000
200,000 ≤ MST < 1,000,000	→ 200 000, 199 900, 1 000 000
MST ≥ 1,000,000	→ 1000 000, 999 900

     (① ② ③ ④ ⑤ ⑥ ⑦ ⑧) 200 000 - 100 → based on ①-⑧ → write out like this

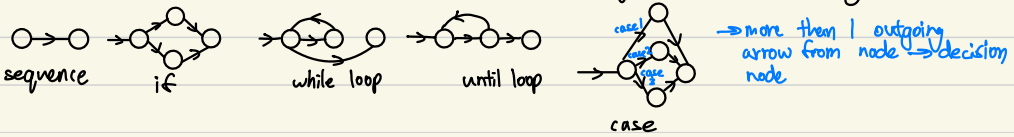
   - Decision Table-Based Testing (explore all feasible combi of input conditions)

     E.g. In the specification of a online stock trading program, buy order only placed if stock-name valid, quantity-to-buy must be in multiple of 1000 & available fund > sum of total cost & charges involved.

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Conditions								
Valid Stock?	No	No	No	Yes	No	Yes	Yes	Yes
Valid Quantity?	No	No	Yes	No	Yes	No	Yes	Yes
Sufficient Funds?	No	Yes	No	No	Yes	Yes	No	Yes
Actions								
Buy?	No	No	No	No	No	No	No	Yes

     → these are independent conditions, so have 2^3 rules.
     $x > 5$ & $x \leq 5$ not independent → just choose 1

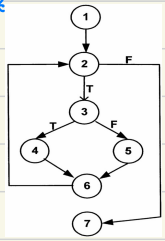
2. White-Box Testing / Structural testing (testing based on analysis of internal logic, e.g. code)
   - Control Flow Graph (CFG) → shows control flow b/w program statements during execution



E.g.

```
(1)  read (i,y)
     a:=1
     b:=1
(2)  while ( i < 100) do
(3)          c := i + 1
             z := f ( )
             if ( y < 0 )
(4)               a:=2
                  b:=2
             else
(5)               a:=3
                  b:=3
             endif
(6)          i:= i+1
             y:= y+c
     endwhile
(7)  print (a, b)
```

cut before & after jump to another location



→ Statement Coverage Testing (design test cases to execute each statement in program at least once)

E.g. Base on above → best test path (1,2,3,4,6,2,3,5,6,2,7) } minimal test suite

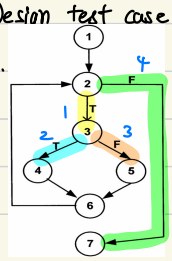
↳ Inputs: i=98, y=-1; Expected result: a=3, b=3

→ Branch Coverage Testing (same as Statement Coverage Testing)

→ Basis Path Testing (identify an essential set of paths through CFG)

1. Compute cyclomatic complexity → V(G) = E-N+2 (no. of linearly independent paths to be tested; E → total no. of edges; N → total no. of nodes)
2. Identify a basis set of V(G) linearly independent paths.
3. Design test case to force through each path identified in step 2.

E.g.



1. V(G) = 8-7+2 = 3
2. Paths: (1,2,7) ← least Δs
   ↳ (1,2,3,4,6,2,7)
   (1,2,3,5,6,2,7)
3. Input: i=101, y=1, expected results: a=1, b=1 → need check if it is feasible
   Input: i=99, y=-1, expected results: a=2, b=2
   Input i=99, y=1, expected results: a=3, b=3

- statement coverage → % of statement executed by test suite (else, endif, endwhile not statements, part of if & while) → $(\frac{12}{14})*100\% = 85.7\%$
- branch coverage → % of branches (outgoing edge at decision node) executed by test suite → 75% $(\frac{3}{4})$
- basis path coverage → % of linearly independent paths executed by test suite $(\frac{1}{3}) = 33.33\%$

Web Engineering

→ specific branch of software engineering that focuses on developing web-based applications

→ compared to software engineering:

a) web engineering must consider issues such as scalability, security & distributed systems

b) web engineering involves a broader range of tech (web development framework, web server & databases etc.)

c) web engineering require good understanding of web-specific protocols & standards (HTTP, HTML, CSS)

WebApp Attributes

1. Network intensiveness → webapp resides on network & serve needs of diverse community of clients
2. Concurrency → large no. of users may access web app at one time
3. Unpredictable load → no. of users may vary by orders of magnitude from day to day
4. Performance → if web app user need to wait long → decide to go elsewhere
5. Availability → users demand access 24/7
6. Data driven → use hypermedia to present text, graphics, audio, video to end-user
7. Content sensitive → quality & aesthetic nature of content determine quality of webapp.
8. Continuous evolution
9. Immediacy → time to market can be a few days/weeks
10. Security → protect sensitive content & provide secure modes of data transmission
11. Aesthetics

Web Engineering Process Model

→ characteristic of web app that must be taken into account:

1. incremental delivery → framework activities will occur repeatedly as each increment is engineered & delivered
2. frequently changes → Δs occur as result of evaluation of delivered increment / changing business conditions
3. short timelines

→ incremental process model used in nearly all situations:

a) faster time to market → release in small & manageable chunks

b) flexibility → adjust based on user feedback

c) risk reduction → mitigate risk of introducing major bugs/error

d) improved collaboration/alignment → b/w development team, pd's manager & stakeholder → improved pdt quality

e) cost-effectiveness → can avoid wasting resources on features that may not be impt to user

→ agile process model appropriate in many situations:

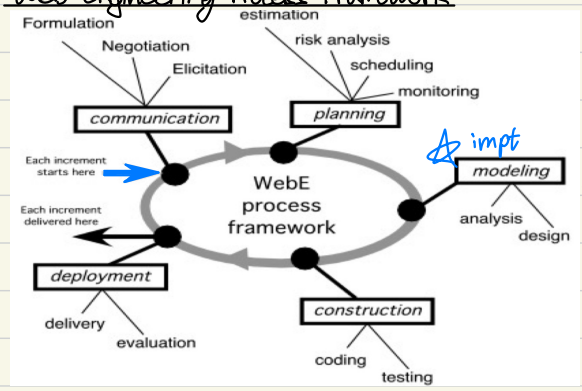
a) faster time to market → deliver software in small increments → respond more rapidly to changing market conditions, customer needs, emerging opportunities

b) increased collaboration & communication → agile emphasise impt of open communication channels → better alignment of goals & improved quality of output

c) improved adaptability & flexibility → can adjust approach in real time based on changing requirements, issue or new opportunities

→ webapp team roles: content manager, information architect (→ developer), web engineer (→ web publisher), business domain experts, support specialist, administrator (→ front/back)

Web Engineering Process Framework



→ Customer communication → describe problem that web app solve

→ Planning → task definition & timelire shedule for development of web app

→ Construction → tools & tech applied to construct & test web app that have been modeled

→ delivery & evaluation (deployment) → fire ture & deliver to end users, evaluation feedback presented to WebE team & increment modified as requirement

Analysis Modeling (communication)

1. Content Analysis / The Content Model (content provided by web app is identified) → data modelling used to identify & describe each data object
   - → information model → identify content objects web app needs to deal with, attributes of each object & relationship among those objects
   - → help to ensure content is organised in meaningful & efficient way → easier for users to find what they need, improve overall user experience
   - → E.g. ER diagram / Class Diagram in UML
     - Objects: book, author, customer ...
     - Attributes: book: title, authorName ...
     - Relationships: book vs author, book vs publisher

2. Interaction Analysis / The Interaction Model (manner user interacts with web app describe in detail)
   - → represented by:
     - a) functions of app → Use-cases
     - b) user actions with objects → Sequence diagrams
     - c) model behavior of webapp → State diagrams
     - d) how interface look like → user interface prototype → shows layout of user interface, content, interaction mechanism & overall aesthetic

3. Functional analysis / The Functional Model
   - → addresses 2 processing elements of web app:
     - a) user observable functionality that is delivered by web app to end-users (define other functions necessary to user)
     - b) operations contained within analysis class that implement behaviours associated with class (defines operation applied to web app)
   - → activity diagram used to represent processing flow

4. Configuration analysis / The Configuration Model (describe environment & infrastructure web app resides)
   - → Server-side: a) server hardware & OS environment specified
     - b) interoperability considerations on server-side must be considered
     - c) appropriate interfaces, communication protocols & other info satisfied
   - → Client-side: a) browser configuration issues identified
     - b) testing requirements should be defined

Design Modeling

→ 5 Major attributes to assess quality:

1. Usability → help features, interface & aesthetic features
2. Functionality → searching & retrieving capability, application domain-related features
3. Reliability/Security → error recovery
4. Efficiency/Performance → response time performance
5. Maintainability → Ease of correction/adaptability

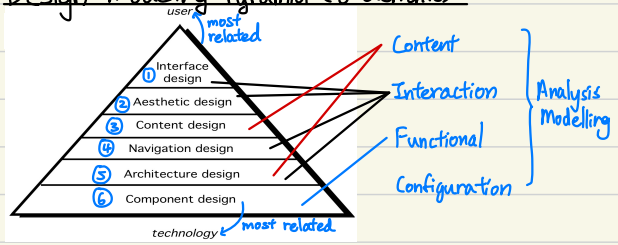
→ some non-functional requirements:

1. Security → ensure privacy of user/customers
2. Availability → measure percentage of time that a web app is available for use
3. Scalability → can web app handle significant variation in user/transaction volume

Robust

4. Time to Market
5. Consistency → architectural design → establish template that lead to a consistent hypermedia structure
   - → interface design → consistent modes of interaction, navigation & content display
   - → navigation mechanism used consistently
6. Identity → appropriate for business

Design Modeling Pyramid (6 Elements)



1. Interface design principles

a) anticipation → web app designed to anticipate user's next move
b) communication → communicate status of any activity initiated by user
c) consistency → navigation controls, menu, icons & aesthetics
d) fitt's law → time to acquire a target by user
e) latency reduction → use multi-tasking in a way that lets user proceed with work as if operation completed
f) learnability → minimise learning time & relearning time
g) readability → information readable by young & old
h) track state → state of user interaction tracked & stored so user can logoff & return later

2. Aesthetic Design

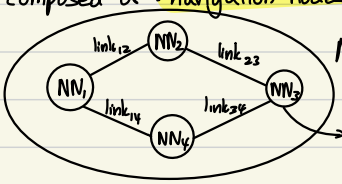
→ white space is ok, emphasize content
→ organise from top-left to bottom right & group thing tgt
→ no scroll bar & consider resolution & browser window size

3. Content Design

→ Design representation for content objects → has attributes (content-specific information (ID, Price) & implementation-specific attributes (font, size, color))
↳ mechanisms required to instantiate relationship to one another

4. Navigation Design (design & manage flow control)

- → When each user interacts web app, encounters a series of **navigation semantic units (NSUs)** → set of info & related navigation structure that fulfill subset of user requirements
- → **way of navigation (WoN)** → **best navigation way/path** to achieve their desired goal/sub-goal.
  - → composed of : **navigation nodes (NN)** connected by navigation links



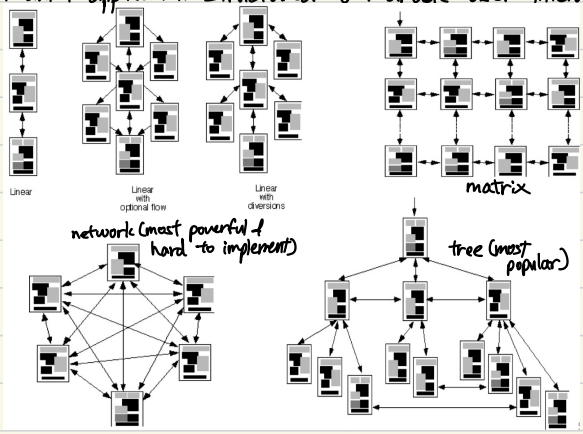
NSU → include links, buttons, menus, tabs etc.

→ webpage/object/functionality

- → **navigation syntax** (depicts technique, patterns) :
  - a) individual navigation link → text-based links, icons, buttons & switches
  - b) horizontal navigation bar → major content
  - c) vertical navigation column → list major content/objects
  - d) tabs & site maps

5. Architecture Design

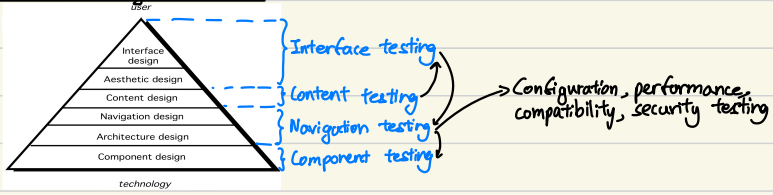
- → content architecture → manner in which **content objects are structured for navigation & presentation**
- → web app architecture → manner in which application structured to manage user interaction, handle internal processing task & present content



6. Component-Level Design

- → additional functions **for localised processing** → speed up transactions & minise network usage
- → provide **sophisticated database query & access**, establish data interfaces with external corporate systems

The Testing Process



1. Content Testing

- → to uncover **syntactic errors** (typo, grammer)
- → to uncover **semantic errors** (error in accuracy/completeness of info)
- → to find errors in **organisation/structure of content that is presented to user**

2. User Interface Testing
- interface tested to ensure visual content correct & is tested within the context of a use-case/NSU

3. Usability Test
- define set of usability testing categories & identify goals for each & design test for each goal → select participants who conduct test

4. Compatibility Testing
- define a set of "commonly encounted" client side computing configurations & their variants
- create a tree structure identifying each computing platform, OS, browser available, internet speed etc.

5. Component-Level Testing (set of test to uncover error in web app functions)
- use black-box (analyses functionality), white-box (correctness of internal code) test case
- database testing often an integral part of component-testing

6. Navigation Testing
- test Navigation links, redirects, bookmarks, frames & framesets (proper layout, download performance etc.), site maps (link takes user to proper content), internal search engines (accuracy & completeness of search)

7. Security Testing
- designed to probe vulnerabilities of client side environment (bugs in browser, email), communication between client & server, server-side environment (denial-of-service attacks)

8. Performance Testing
- Load Testing (determine how web app & server will respond to various loading conditions)
  throughput, $P = N \times T \times D$ (N: no. of concurrent users; T: no. of on-line transactions per unit time; D: data load processed by server per transaction)
- Stress Testing (exceed operational limits)
  - how system respond & notify users & other steps it take to mitigate problem, anything lost in process?

Software Project Management (SPM)

→ is a proper way of planning & leading software projects to ensure pdj are completed successfully on time, within budget to required quality standards

→ 4Ps in SPM:

1. **People** (most impt element)
   - → important to have skilled & motivated team that can work collaboratively to achieve goals
   - → to organise a team, need to consider difficulty & size of problem, time team will stay tgt & degree of sociability, required quality & reliability of system to be build, rigidity of delivery date.
   - → 4 different organisation paradigm:
     - a) closed paradigm (duplicate smt done before) → structure team along traditional hierarchy of authority
     - b) random paradigm (smt never done before) → structure team loosely & depend on individual initiative of team members

     (compromise)
     - c) open paradigm → some control like closed paradigm but also innovation using random paradigm
     - d) synchronous paradigm → organises team members to work on pieces of problem (→ natural compartmentalisation of problem) with little active communication among themselves

2. **Product Scope** (must be unambiguous & understandable at management & technical levels)

(product)
- → must more than source & object code. Includes document (→ include customer documents), test plans & results (e.g. installation guide)

(scope)
- → context → how does software build fit into larger system, what constraints?
- → information objectives → data objects required for input/output.
- → function & performance → type of function performed, any special characteristic to be addressed
- after scope defined → problem decomposition → into constituent functions (use-case), user-visible data objects (ER, class), into a set of problem classes (server/client)

3. **Process** (framework for carrying out activities in pdj in organised & disciplined manner)
   - → process framework established → consider pdj characteristics & degree of rigor required → define task set for each software engineering activity (e.g. task set = software engineering task, work pdt, quality assurance points, milestones)

4. **Project** (other work to make pdt a reality)
   - → approaches to take to have clear understanding of pdj scope, shedule & budget, risk & constraints that may impact the pdj

W⁵HH principle

- → Why is system being developed?
- → What will be done?
- → When will it be accomplised?
- → Who is responsible for each function?
- → Where are they organisationally located?
- → How will job be done technically & managerially?
- → How much of each resource is needed?

Critical Practices

1. Risk mangement
2. Cost & shedule estimation
3. Metric-based pdj management
4. Earned valve & defect tracking
5. People aware pdj management

Metrics to measure process & pdt

1. Process Metrics
   - → derive set of metrics based on outcomes that can be derived from process
   - → quality-related, productivity-related, code complexity etc.
2. Project Metrics
   - → use to assess pdt quality on ongoing basis & when necessary, modify technical approach to improve quality
   - → effort/time per task, error uncovered/hour or person-month, schedule vs actual milestone dates, distribution of effort on software engineering tasks

Size-Oriented Metrics (LOC, person-month, review hour)

E.g. A software system composed of 5 subsystem as below:
user interaction (LOC = 2400), sensor monitoring (LOC = 1100), message display (LOC = 850), system configuration (LOC = 1200), system control (LOC = 900)
Assuming your organisation produces 500 LOC/pm with burdened labor rate of $8000 per person-month (pm), estimate effort & cost required to build software using LOC-base estimation technique.

total LOC = 2400 + 1100 + 850 + 1200 + 900 = 6450 LOC

effort = $\frac{6450}{500} = 12.9 \approx 13$ pm → a month a person works

cost = 12.9 * 8000 = $103,200

Function Points (programming language independent compared to LOC) → lines of code

E.g. A software system has 10 external inputs, 20 external output fields, 25 different external queries, manages 4 internal logical files, & interfaces with 4 different legacy system (4 EIFs).
Assume all external input, output & user queries are of average complexity & all logical files & interfaces are of complex complexity, Sum(Fi) = 42. Compute FP for system. Weighting-factor given in table.

Parameter	Weighting factor		
	Simple	Average	complex
# of user inputs	3	**4**	6
# of user outputs	4	**5**	7
# of user inquiries	3	**4**	6
# of files	7	10	**15**
# of external interfaces	5	7	**10**

4 X 10 = 40
5 X 20 = 100
4 X 25 = 100
15 X 4 = 60
10 X 4 = 40
} add tgt
total weight → 340

important: FP = total weight × [0.65 + 0.01 × SUM(Fi)]

= 340 × [0.65 + 0.01 × 42] = 363.8

- where F_i (i = 1 to 14) are **complexity adjustment values** based on the ta
  1. Reliable backup/recovery needed?
  2. Any data communications needed?
  3. Any distributed processing functions?
  4. Performance critical?
  5. Will system run in an existing heavily utilized operational environment?
  6. Any on-line data entry?
  7. Does on-line data entry need multiple screens/operations?
  8. Are master files updated on-line?
  9. Are inputs, outputs, queries complex?
  10. Is internal processing complex?
  11. Must code be reusable?
  12. Are conversion and installation included in design?
  13. Is multiple installations in different organizations needed in design?
  14. Is the application to facilitate change and ease of use by user?
- Each F_i criteria are given a rating of 0 to 5 as:
  - No Influence = 0; - Incidental = 1;
  - Moderate = 2 - Average = 3;
  - Significant = 4 - Essential = 5

Story Points

→ unit commonly used in Agile software development for estimating relative size/effort of user stories/tasks, represent relative measure of effort/complexity

Measuring Quality

- → correctness → degree to which a program operates according to specification
- → maintainability → degree a program is amenable to change
- → integrity → degree a program is impervious to outside attack
- → usability → degree a program is easy to use

— Defect Removal Efficiency → DRE = $\frac{E}{(E+D)}$
E → no. of errors before delivery to end user
D → no. of defects found after delivery

* the higher the DRE, more defects were caught & fixed during testing

Estimation for Software Projects (1st step in planning)

→ pdj scope must be understood, elaboration is necessary, historical metrics helpful, at least 2 different techniques should be used, uncertainty is inherent in process

→ 4 estimation techniques:

1. Automated Tool-Based Estimation

→ for less experienced managers → can auto-calculate, can just feed in LOC/FP data

2. Empirical Model (COCOMO: constructive cost model)

→ general form: effort = tuning coefficient (constant no. derived from past pdj) × size (empirically derived exponent) → LOC/FP

3. Conventional Estimation Techniques (compute LOC/FP using estimates of information domain values) (or problem decomposition)

a) LOC/FP Approach → calculate FP first, productivity = 6.5 FP/pm

E.g. estimated LOC = 33 200, average productivity = 620 LOC/pm, burdened labor rate = $8000/month

cost of LOC = $\frac{8000}{620} = \$13$

total pdj cost = 13 × 33200 = $431 000

effort = $\frac{33200}{620} = 54$ pm

b) Process-Based Estimation

application function ↓	framework activities (tasks) →
→	effort required to accomplish each framework activity for each application function

Activity →	CC	Planning	Risk Analysis	Engineering		Construction Release		CE	Totals
Task →				analysis	design	code	test		
Function ↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DSM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

4. Past (similar) Pdj Experience

a) Estimation with use-case (historical-data per use case)

	use cases	scenarios	pages	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	12	5	560	3,366
Engineering subsystem group	10	20	8	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	10	6	1650	7,970
Total LOC estimate							42,568

Average productivity = 620 LOC/pm

Labor rate = $8000 per month

calculate same method as above

b) Analogous Estimation

→ identify completed pdj (source case) with similar characteristics to new pdj (target case) → identify difference bw target & source & adjust base estimate to produce an estimate for new pdj

→ angel software tool, distance = $\sqrt{(\text{targetP}_1 - \text{sourceP}_1)^2 + \ldots + (\text{targetP}_n - \text{sourceP}_n)^2}$

E.g. Case 1: Input = 5, Output = 3, time = 2 weeks. Case 2: Input = 7, Output = 4, time = 4 weeks. New case with inputs = 6 & output, we want to estimate time required to develop it, using analogous estimation

Case 1: Distance = $\sqrt{(6-5)^2 + (3-3)^2} = 1$ → smallest distance, use this ✓

Case 2: Distance = $\sqrt{(6-7)^2 + (3-4)^2} \approx 1.414$

∴ Estimate 2 weeks / 80% * 2 weeks

Estimation for 4th generation language: Application Points

→ application point (similar to FP) → compute no. of screens, no. of reports, no. of modules

→ Effort = $\frac{NAP(1-\%r)}{PROD}$ (NAP: total no. of application points in delivered system; %r: % of components reused; PROD: productivity (decided from table), unit = NAP/mth)

E.g. Estimate person-month needed if pdj has highly experienced team, reused code = 60%, no. of estimated application pt = 250.

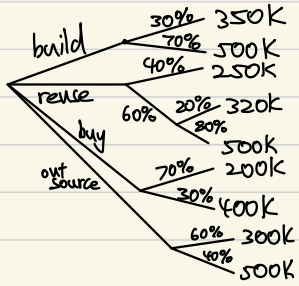
Developer Experience	VeryLow	Low	Nominal	High	VeryHigh
PROD (NAP/mth)	4	7	13	25	50

→ Effort = $\frac{250(1-\frac{60}{100})}{50} = 2$ pm

★Estimate best option

E.g. Software pdj manager faced with option to make/buy software. Draw decision tree to decision process & make recommendation base on 1) training team 2) cost. Refer to table for decision tree:

Option1: to build		Option2: Reuse			Option3: ToBuy		Option4: ToOutsource	
Simple	Difficult	MinorC	Major Changes		MinorC	MajorC	withoutC	withC
30%	70%	40%	60%		70%	30%	60%	40%
			Simple	Difficult				
$350K	$500K	$250K	20% $320K	80% $500K	$200K	$400K	$300K	$500K



Expected cost–build = 0.30 (350K) + 0.70 (500K) = $455 K

Expected cost – reuse = 0.40 (250K) + 0.60 (0.20)(320K) + 0.60 (0.80)(500K) = $378.4K

Expected cost – buy = 0.70(200K) + 0.30 (400K) = $260K

Expected cost – outsource = 0.60 (300K) + 0.40 (500K) = $380K

∴ training team criteria → build

∴ cost criteria → buy

Risk Management (2nd step in planning)

→ action that help software team understand & manage uncertainty & identify, address & eliminate sources of risk before they become threats to successful completion of pdj

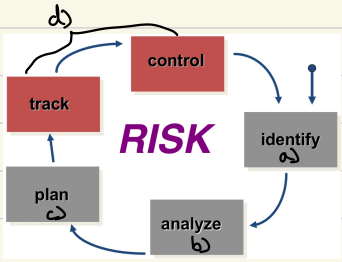
→ key elements of risk management:

a) identification → what are risks to pdj?

b) analysis & prioritisation → which ones really serious

c) planning → what shall we do?

d) monitoring → has planning worked?



1. Reactive Risk Management (pdj team react to risk when they occur)

→ mitigation → plan for additional resources in anticipation of fire fighting

→ fix on failure → resource found & applied when risk strikes

} if pdt don't respond well → pdj in jeopardy

2. Proactive Risk Management (formal risk analysis performed before they occur)

→ organisation corrects the root cause

→ examining risk sources that lie beyond the bounds of software & developing skills to manage change

Risk Components

→ performance risk → degree of uncertainty that pdt will meet its requirement → partition requirements into critical, desirable

→ cost risk → degree of uncertainty that pdj budget maintained → adopt design where system functionality adapted to a fixed cost

→ support risk → degree of uncertainty that software will be easy to correct, adapt & enhance

→ shedule risk → degree of uncertainty that pdj schedule will be maintained & pdt delivered on time

→ reuse software → ↓amount of estimation & overall cost

→ obtain a no. of independent estimate → generate more costing info, those hard to estimate → develop prototype to find problems that is likely to arise

Risk Table

→ high probability & high impact on top

Risk	Probability	Impact	RMMM
Identify all the risk	Estimate probability of occurrence	Estimate impact from (low impact on pdj success 1–5 high impact on pdj success)	Risk Mitigation Monitoring & Management

- Risk Exposure (Impact) → RE = P × C (C → cost to pdj should risk occur)

E.g. Only 70% out of 60 software components will be scheduled for reuse (→ others have risk). Risk probability is 80%. Average component is 100 LOC & each LOC is $14.00. What is the overall cost (impact) to develop components?

$60 \times (1 - \frac{70}{100}) = 18$ → need to develop from scratch

$cost = 18 \times 100 \times 14 = \$25\,200$

$RE = \frac{80}{100} \times 25\,200 = 20\,160$ → update table

Risk	Probability	Impact	RMMM
18	80%	~$20 200	RMMM

Risk Reduction Leverage

$RRL = (RE_{before} - RE_{after}) \div (\text{cost of risk reduction})$

- RE_{before} → risk exposure before risk reduction (e.g. 1% chance of fire causing $200K damage)
- RE_{after} → risk exposure after risk reduction (e.g. fire alarm costing $500 reduces probability of fire damage to 0.5%)

$RRL = ((\frac{1}{100} \times 200K) - (\frac{0.5}{100} \times 200K)) / 500 = 2$ → If RRL > 1.00 → worth doing

Project Scheduling (3rd step in planning)

↳ scheduling principles:

a) compartmentalisation → define distinct tasks

b) interdependency → indicate task interrelationship

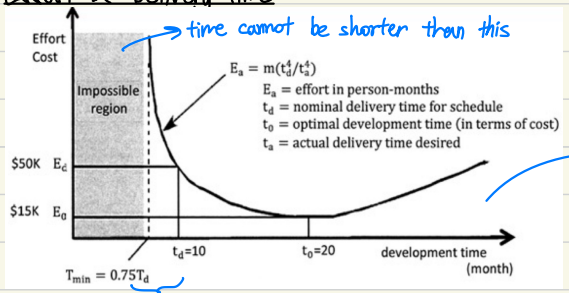
c) time allocation → assigned person days, start & ending time

d) effort validation → be sure resources are available

e) defined responsibilities (→ ppl assigned), outcomes (→ each task must have output), milestone → review for quality

f) contingency plan → risks

Effort & Delivery Time



→ time cannot be shorter than this

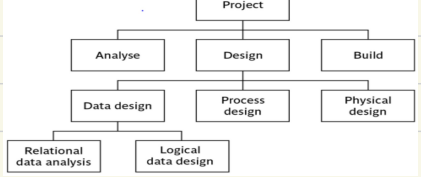
→ Is it possible to reduce delivery time to 8 month, 6 months?
↳ 8 possible, not 6 months → every pdj have overhead. Tmin = 0.75 × 10 = 7.5 mths. Development time < 7.5 mth impossible → high risk of failure

→ t > to → waste of time, cost ↑ as do unnecessary testing, documentation to fill up time, payment to members

→ cost ↑ non-linearly when try to accelerate pdj from td to Tmin
→ overtime → more errors, ppl fall sick → extra cost
→ replaced by premier staff → pay premier fees for those ppl

Work-based vs Product-based Decomposition

→ work-based structure → elementary block: activity components



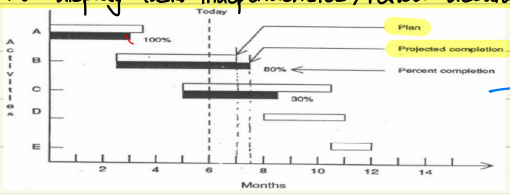
↳ product-based structure → elementary block: deliverable pdt

User-Stories-based breakdown (Agile uses this)
↳ follow a format: As a [type of user], I want [some goal] so that [some reason]

Project Scheduling Techniques

1. Gantt Chart

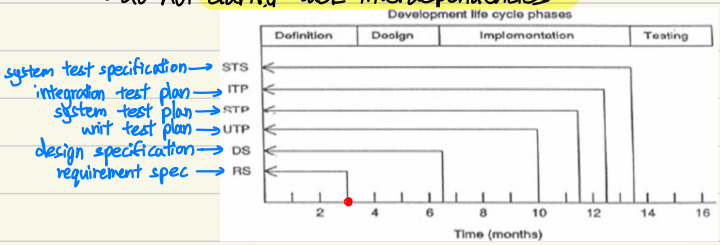
→ provide graphic representation of pdj plan, schedules & progress
→ does not display task independencies / reflect accurately slippage of completion dates



→ illustrate start & finish dates of terminal tasks & summary tasks of a project

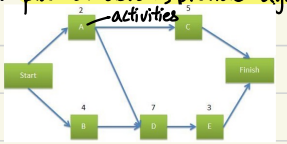
2. Milestones

→ represent key events on a bar chart
→ do not clarify task interdependencies



3. Network Diagrams

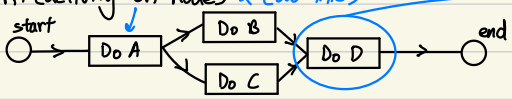
→ portray sequential relationships bw key events
→ show plan of action, provide dynamic process for pdj updates

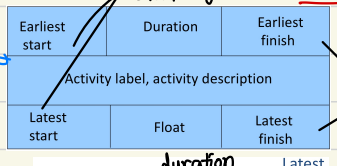


PAN (Precedence Activity Network)

→ help us to assess feasibility of planned pdj completion date (1), identify when resources will need to be deployed (2) to activities & calculated when cost will be incurred (3)

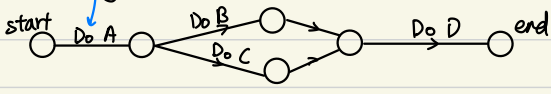
→ PERT (activity on node) ★ (do this)

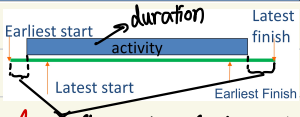




start Day 1 means start Day 2
finish Day 1 means END of Day 1

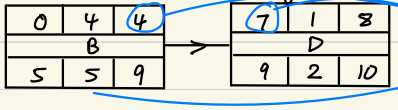
→ CPM (activity on arrow)





★ total float = last finish − earliest start − duration

→ critical path → all floats in path = 0, can have more than 1 / none critical path
→ sub-critical path → float close to 0 → depends on threshold (assume 1/2)
→ free float = ES (following) − EF (current), interfering float = float (current) − free float

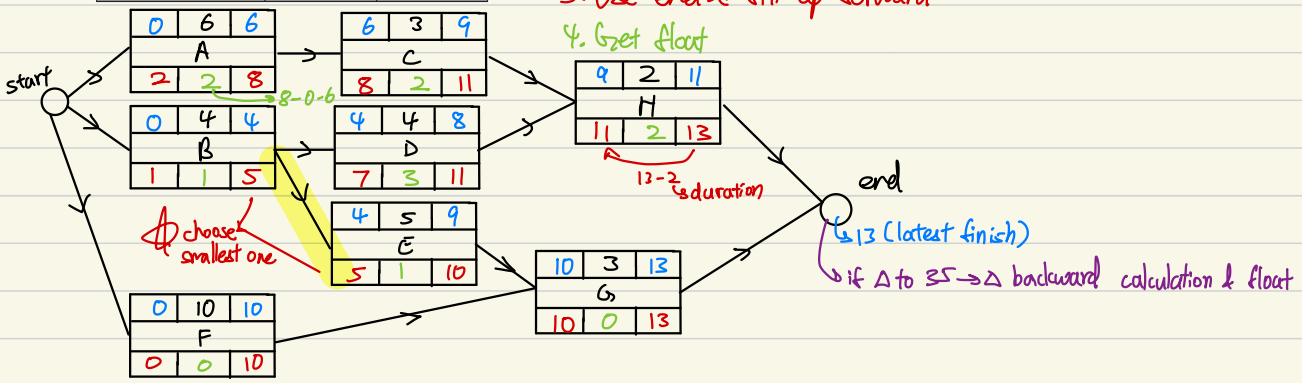


7 − 4 = 3 → B can be up to 3 days late & not affect other activities
5 − 3 = 2 → B can be further 2 days late, may affect D but not pdj end date

E.g. Create precedence activity network (PAN) in BS4335 using info given in table, update PAN if pdj given 35 days to complete.

Activity	Duration (days)	Precedent(s)
A Hardware Selection	6	
B System Configuration	4	
C Install Hardware	3	A
D Data Migration	4	B
E Draft Office Procedures	~~35~~	B
F Recruit Staff	10	
G User Training	3	E, F
H Install & Test System	2	C, D

1. Write down links & durations
2. Start from front (day 0) & fill in until end
3. Use end & fill up forward
4. Get float



∴ critical path F – G → float 0

↳ network diagram usage:

a) planning → list all activities & show sequence of activities to be performed
b) scheduling → time schedule, critical activities, update schedules easily
c) controlling → comparing actual activities against plan, update plan to accommodate revisions quickly
d) communicating → communicate plan to pdj members easily

Project Schedule Tracking (4th step in planning)

- conduct periodic pdj status meeting, evaluate result of reviews conducted through software engineering process
- determine if pdj's milestones have been accomplished by scheduled date, compare actual to planned start date
- meet informally with practitioners to obtain their subjective assessment of progress to date

Earned Value Management (EVM)

- pdj management technique used to track progress & current status of pdj in terms of scope, cost & schedule
- 3 basic elements:

1. Planned Value (PV) → budgeted cost of work scheduled (BCWS) → total cost of work scheduled
   - Budget at completion (BAC) → sum of BCWS values = $\Sigma_i(BCWS_k) = \Sigma(PV_k)$ for all task k
   - PV = (planned / completed %) * BAC
2. Earned Value (EV) → budgeted cost of work performed (BCWP) → total cost of work completed
   - EV = (actual / complete %) * BAC
3. Actual Cost (AC) → actual cost of work performed (ACWP) → amount of money spend so far (total cost taken to complete work)

→ Schedule performance index (SPI) = $\frac{EV}{PV}$ (>1 more work done than planned, =1 same, <1 less work done) — efficiency of pdj's utilising scheduled resources

→ Schedule variance (SV) = EV − PV { >0 good, <0 bad

→ Cost performance index (CPI) = $\frac{EV}{AC}$ (>1 earn more than spent, =1 same, <1 earn less than spent) — efficiency of cost utilised on pdj

→ Cost Variance (CV) = EV − AC { >0 good shape, <0 bad shape

→ Estimated cost at completion (EAC) → = $\frac{BAC}{CPI}$, if CPI fixed
→ = $\frac{BAC}{SPI}$, if SPI fixed

E.g. A $10 000 software pdj scheduled for 4 week. At end of third week, pdj 50% complete, actual cost to date $9000. What is PV, EV & AC? Is it healthy?

$PV = \frac{3}{4} \times 10K = 7.5K$

$EV = \frac{50}{100} \times 10K = 5K$

$AC = 9K$

$SV = EV - PV = 5K - 7.5K = -2.5K$

$SPI = \frac{EV}{PV} = \frac{5K}{7.5K} = 0.66$

$CV = EV - AC = 5K - 9K = -4K$

$CPI = \frac{EV}{AC} = \frac{5K}{9K} = 0.55$ → additional cost needed to complete pdj

$ECA = \frac{BAC}{CPI} = \frac{10k}{0.55} = 18\ 182$, if CPI stay the same

To-Complete Performance Index (TCPI)

- help pdj managers assess the required efficiency to achieve specific pdj objectives
- TCPI projection of required CPI needed to achieve pdj goals within budget

$$TCPI = \frac{(BAC - EV)}{(BAC - AV)}$$

- TCPI < current CPI → future work done at lower cost performance level
- TCPI > current CPI → pdj can afford to be more cost-efficient in remaining work

Quality Management

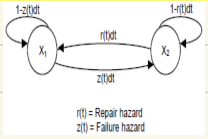
- to manage quality of software & its development process
- quality assurance = validation (are the requirements correct) + verification (are the requirements implemented correctly)
- cost of quality:
  1. prevention costs include: quality planning, formal technical reviews, test equipment, training
  2. internal failure cost include: rework, repair, failure mode
  3. external failure cost include: complaint resolution, pdt return & replacement, warranty work etc.

Software Reliability

- simple measure of reliability → mean-time-between-failure (MTBF) = MTTF + MTTR (MTTF = mean-time-to-failure, MTTR = mean-time-to-repair)
- software availability → probability program is operating according to requirements at given point

$$availability = \frac{MTTF}{MTTF + MTTR} \times 100\%$$

E.g. Figure shows state transition diagram of a dual identical-components system (x_1 & x_2). One runs front end & the other is redundant. If front end failure, redundant take control, max 5 second needed for switchover. Given mean-time-to-failure = 8816 hours. Compute availability of such a system.



$MTTR = 5_s = \frac{5}{(60 \times 60)} h$

$Availability = \frac{8816}{(8816 + \frac{5}{60 \times 60})} \times 100\% = 99.99998\%$

Assume redundant system always succeed in gaining control without losing any information/data

→ Software safety → software quality assurance activity that focuses on identification & assessment of potential hazards that may affect software negatively.