if d[v] > w[u,v]
    d[v] = w[u,v]
    N[v] = u

Software Eng CS-208

Repeatable → quality of product should be consistent

Drawback of Waterfall Method → No feedback, recurrsive

Novel method → maybe third party involved
maybe don't give everything to customer

Agile Dev

Agile variants

1) Extreme Programming
- Kent Beck   Chryslier Comprehensive Compensation payroll project
- Rapid feedback → ppl hurt for waiting months for their product
- Assume Simplicity

Rapid feedback
- Time b/w action & feedback is critical
- Make a something from what info we have & get feedback.

Assume Simplicity
- Whatever is to be done, it is very easy.
- Think of it to be easy & just start working. This will give initial momentum. 98% of anything will be easy.
- If we think of it as hard, we will delay everything

Incremental Change
- If something is to be done, change something bit by bit. Don't do everything all at once. But do it quickly.
- Design changes little at a time
- do not ~~embrace~~ work against change but embrace them.
- Plan changes little at a time.
- Telling changes at a stage where dev is still going on is better than making the whole product & then apply changes.

- Real time systems is where this method cannot be done. like Space Programs, Safety measure programs. Also, large projects would not be recommended to follow this.
- Companies like delloite do just the requirements, elicitation phase ie consultancy & planning.
- But alot of disconnects b/w product & customer as time progresses.

2) Scrum

- H Takeuchi & I Nonaka 1986
- Do alot of meetings & be quickly done.
- Don't be overwhelmed by the quickness tho, do it in a clean way too
- Sprint & do it quickly.

24 hr meet with team
30 days with client

Product Backlog ⇒ Sprint Backlog $\xrightarrow[\text{sprint}]{}$ Working S/W ↓ increment.
↳ prioritize it
↳ a part of product backlog is decided to be done
↳ this part is deployed.

Once a sprint starts, do not go back to the client. Although do meet with the team weekly.

Sprint → plan, build, test, review.
↳ but don't change the requirements.
The product backlog can only be changed after sprint.

Scrum Team Compo
- Scrum Master (team leader)
  - ↳ Process coach & team faciliator
  - ↳ Remover of roadblocks.
- Scrum Team
  - ↳ individuals which work on 'sprints'
  - ↳ comprises of diversely skilled individuals.
- Product Owner
  - ↳ gives feedback.

Sprint
- time boxed, usually 2 weeks → 1month.
- define workload
  - ↳ workload does not change during sprints.
  - ↳ if workload changes, but re-start the sprint (but avoid it)

Project Backlog
- What is done in one sprint.
- It is high priority features.
- Product backlog ~~changed~~ can be changed.

Sprint Backlog
- features & func^n for a single sprint.
- broken down into user stories (bunch of points, said in very informal way, but are required. tech points may not be included
- Tech reqs are discussed in this.

Scrum Meetings.
- Product owner & team.
- Review of product backlog
- Negotiation on the features to include in sprint.

Daily Scrum meeting.
- 15 min STAND-UP meeting.
- What have you done } Answers 3 questions.

- Internal, product owner maynot be involved.
- for the benefit of the team, it is not report to master.

Sprint Review.
- Product owner review it. May accept/reject.

Sprint Retrospective.
↳ What could ~~to~~ have been better.

Benefits of Scrum.
- Doesn't overwhelm with feedback. It is restricted.
- Toyota Effect → 4x industry avg productivity
  (achieves it) 12x better quality
  become market leader no matter what
- No/Less management pressure on teams.
- High buisness value.

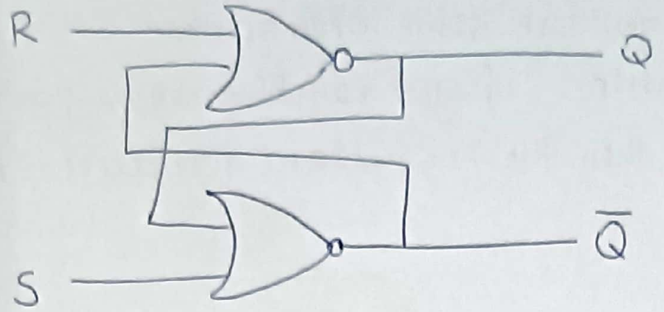
CS-204 DAA

Analysis of Prim's Algo

function MSTPrim

```
for i = 1 to n
    visited[i] = 0
    d[i] = ∞                                  ] O(N)
    N[i] = -1
TE = {1}                                      ] O(1)     initiallisation
visited[1] = 1
for all (1, u) ∈ E
    d[u] = w[1,u]
    N[u] = 1
for i = 2 to n
    Get u ∈ V st d[u] is min and visited[u] = 0   ] O(nlogn)
    TE = TE U {(u, N(u))}
    visited[u] = 1
    for all (u,v) ∈ E such that visited[v] = 0
                                  avoiding cycle
        if d[v] > w[u,v]
            d[v] = w[u,v]
            N[v] = u
```

Type 3 NOR based SR latch



Truth table

S	R	Q	$\bar{Q}$
0	0	Memory state	
0	1	0	1
1	0	1	0
1	1	Prohibited State	

memory state / hold state => won't change untill a new trigger is given

CS-208 Comp Eng.

Open Source Software Development

Like Android

Source code is available to the general public for use & and/or modification from its org. design usually free of cost.

Its redistributional. And also modifiedable & creation of derivatives is permitted.

The license must not discriminate against any user.

Imp → The license must apply to all parties whom soft is distributed

What are halloween documents? → By Microsoft

Software ~~tu~~ Licensing Taxonomy?

use-restricted => made available to some community like students

Share-ware => Certain features free, certain purchasable

free-ware => all free

Royalty-free binaries => would be free, ~~ta~~ but would only come along a certain purchasable soft

Royalty-free libraries => source code available but not modifiable

What is usenet? → first social network

↳ code sharing was accelarated with the diffusion of usenet.

AT & T → American Company

↳ Starting enforcing IP rights related to unix. Usually every thing ppl developed was freely available. But this started a new era

Second Era

General Public License → if you use some open source S/W, then you have to make it free available. If you combine some part of your product with open source then the resultant product should also be freely available.

Third Era

Debian Software Guidelines → allowing comercialising soft which used open source.

Since General public software was hurting the open source moment as no one would use it comercially, they released debian.

GNU full form?

Benefits of open source:

Quality → since alot of ppl involved

Customizability → since source code available

Freedom →

Support Options →

Cost

Try before you buy

Disadvantages

Ownership → No one is owner so creates problems.

Most adv of Open Source:

Less dependence on vendours, lower cost, easier to customise, better security.

Open Source Philosophy:

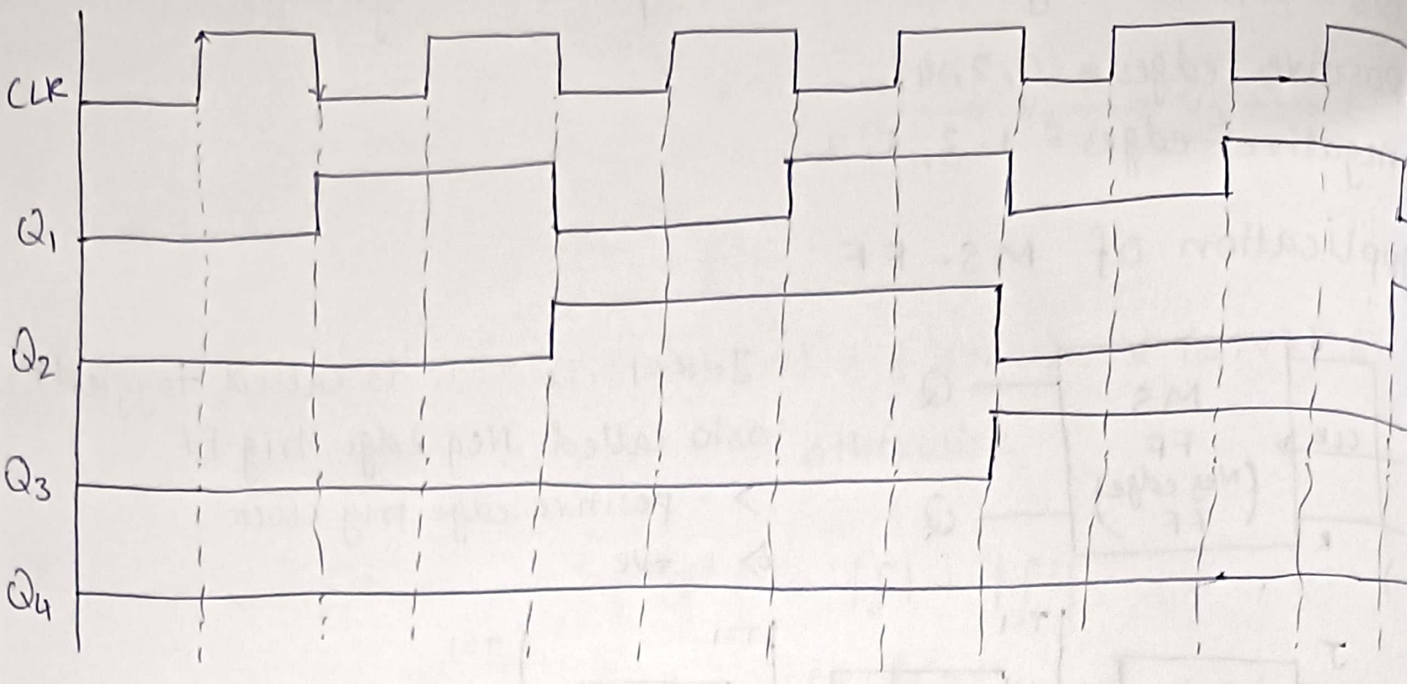
Users should be treated as co-developer.

Early releases, frequent integrating, several versions, high modularisation, Dynamic decision making structure

Kind of like Agile → users should also have a stake in it.

more like choreography where users come & make it work Orchestry.

(freq divider)



Applications: processor
(something to do with dividing frequency)

Q4	Q3	Q2	Q1
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
⋮			
1	1	1	1

CS-208 Soft Eng

Open Source

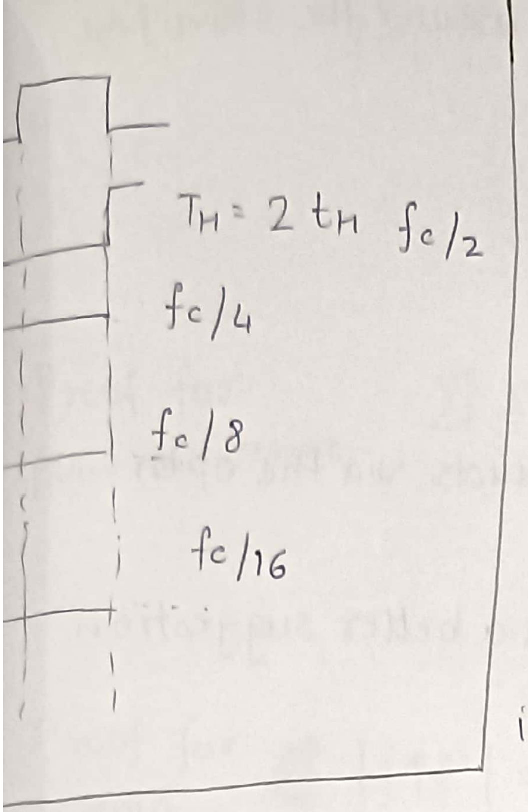
- Dev is open to community. Code is freely available

Biggest issue: licence (redistribution)

↳ If you bring some change, then you have to distribute your code

This was the initial practice & governed by GNU General Public License

Then new license came: Debian, MIT, BSD. Main thing among them is that you don't have to redistribute the code.



One more big problem was Monetization
The licenses made open source a bit more Sustainable by making commercial companies attracted.
Monetising Open Source ~~by mak~~ / Made it Sustainable : First company → Red hat
Red hat distributed Linux for free, but whatever support you use, you have to pay for it.
↳ Another version
↳ New feature
initial / first version was free by Red hat.

My SQL also tried but failed.
The model created by Red hat was successful.

Another Model: Sponsorship
↳ Open Source devs made their profile & based on their profile Companies sponsored them. Like an internship. Every commit they made in this period would have the name of the company.

Another Model: SaaS Software as a Service.
Eg: Google Docs
Takes a small fees for the features.
Display Advertisments.
This model sustained the open source movement.

What is the imp of Redistribution? → May be asked.
What is Copy right?
Copy left → ~~The~~ If I make some edits on the open source, then I have to make the code available.

Support Options were limited in open source since many people didn't know what they talk about.

Open Source & Agile Dev were progressing around the same time both trying to break the norms.

Open Source: Making Soft. cheap.

Agile Dev: Rejecting waterfall model.

Altruism → trying to do good for others.

A company which develops some of their products via the open source route → IBM

Netflix Challenge? Whoever came out with a better suggestion system will be given $1M.

~~NIH~~ NIH Syndrome → Not Invented Here

CS-204 DAA

Residual Graph

Given $G = (V, E), f$ $G_f \{V, E_f)$

$E_f = \{ (v_1, v_2) \in V \times V \mid$ ... $, v_2) >$

$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{...) } \in E \\ f(v,u) & \text{...u) } \in E \\ 0 & \text{...se} \end{cases}$

Lemma → $f \uparrow f'$ is a flow ... $\uparrow f'$ ... f'|

$f \uparrow f'(u,v) = f($ ... $+ f'(u,$ ...

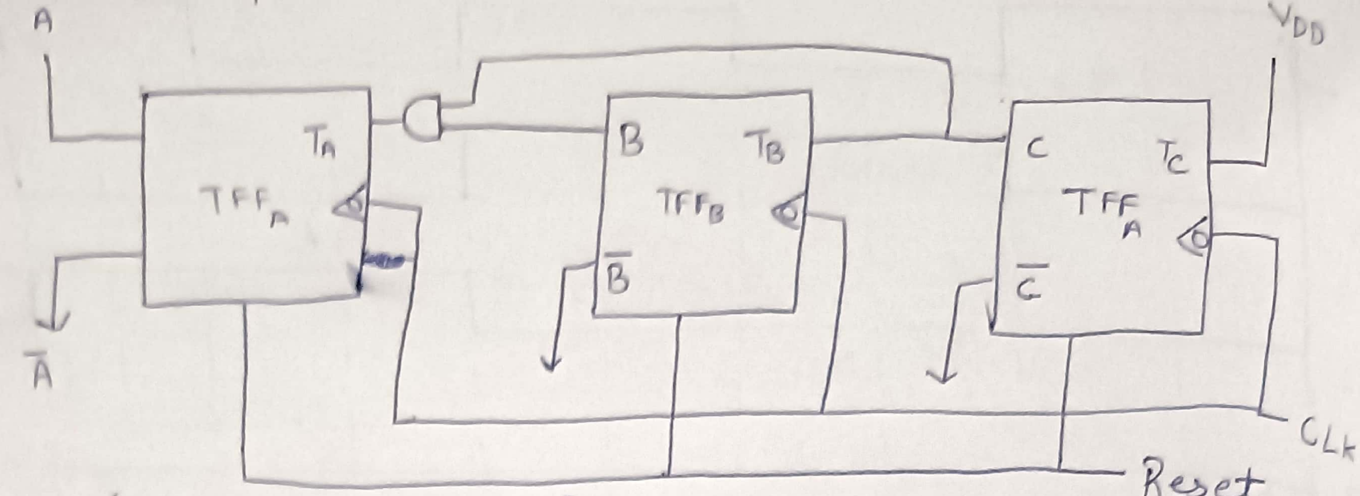
~~Lemma~~ cap... constrai... $u,v)$

flow constraint ... $\in V \setminus \{s,t\}$

Proof:- $f \uparrow f'(u,v$ ...

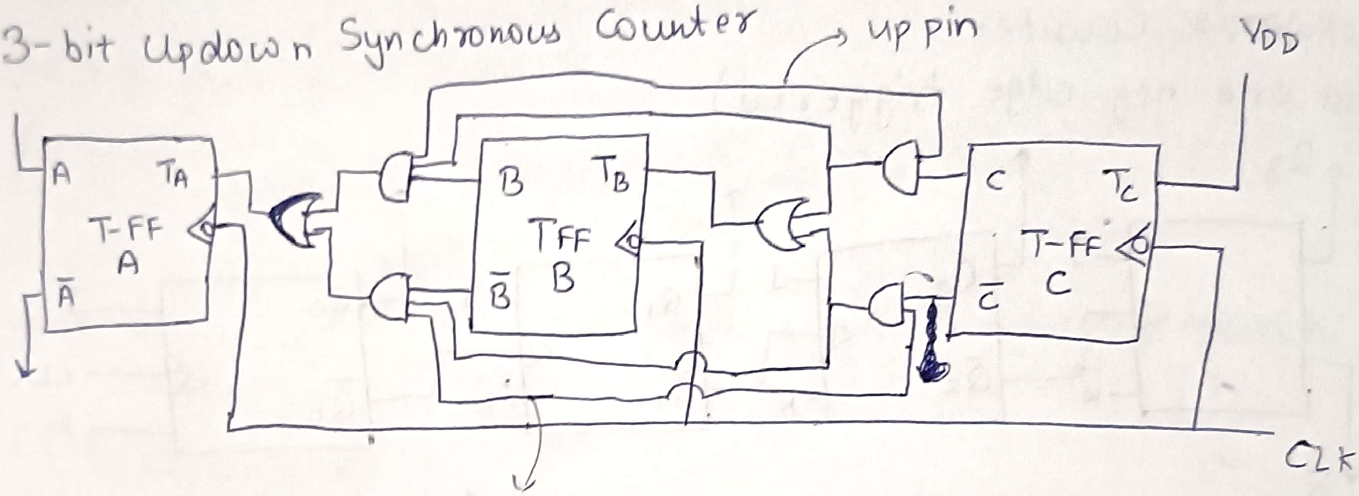
capacity constraint $f(u,$ ... (u,v)

3 bit Upcounter



(Synchronous counter)

3-bit Updown Synchronous Counter



CS-208 Software Engineering

How does Open Source die?

1) Forking

forking becomes the main app

Eg: Netscape Navigator → Chrome

2) Leader

He maybe not interested / unlike orgs.

3) Lack of Updates

→ Outlives its utility

Eg: ERP or the cloud (pay when you need)

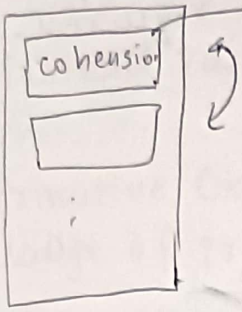
In-house ERP → you can't

ORACLE offers ERP on cloud.

Problems with Incremental Development

1) Poor Code (Maintaince Issues)
2) Structure → Needs high quality
3) Software → Must be modular → high cohesion
→ low coupling

→ similar func^n close together



min coupling

Change in one module shouldn't require much of it in others.

with more changes coupling ↑ cohesion ↓ (or increments)

Refactoring

Set of rules to gain simplification, quality & modules.

Find red flags, and do what works out the best
↳ bad smells

Navigate through the structure, find "bad smell"
→ locally.

you want to make the code clean, as the dev process is adaptive & correct ive.
you also want to make modular as it is easy to change & add features.

Since many ppl code in open source, some people may code in a bad manner
Refactoring → improve the code without changing func^{n}tionality.
↳ kent beck (bad smell)

When to refactor?
→ when you want to add new functionality
→ to find bugs
→ make code understandable

When not ?
→ When you have to completely re write it.

meant for object oriented language OOP

Bad smells

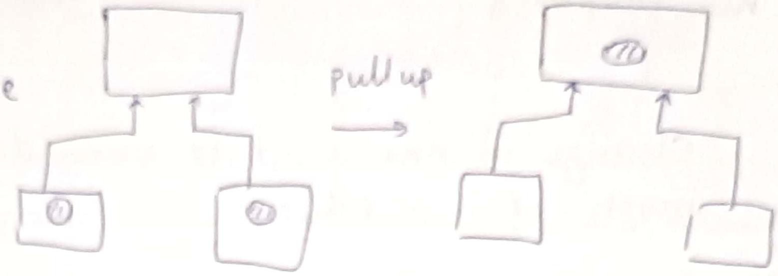
1) Duplicated Code

→ Apply Extract code, Pull up Method

Sibling subclass:

Extract Method: create a new func^n & call it whenever required.

Pull up Method:
transfers the code to parent class



2) Long Method

→ break down the methods into smaller functions.

→ Extract Method can be applied.

3) Large Class

→ Apply Extract Class to factor out some set of variables

4) Long Parameter List.

5) Divergent Change

→ A change is some part of code will ~~not~~ result in changes all over the code. Opposite of shotgun surgery.

→ Extract Class: create a new class

coupling: A lot of interaction between different modules. We want coupling to be low.

cohesion: closely encapsulated. want class to be highly cohesion and cater to one functionality.

→ divergent change means high coupling.

6) Shotgun Surgery

→ A lot of changes happening result in affecting a single class.

→ Reason: high coupling.

→ Apply Move method & move field. (move the code into the affected class)

→ Inline class: merge classes.

7) Feature Envy

→ A class is extremely interested / dependent on variables of other class. Single directional.

→ move method

8) Data Clumps

→ instead passing diff primitive classes, create object to contain them.

9) Primitive Obsession

→ Usage of primitive datatypes excessively

10) Lazy Class

→ class doing nothing much or significant.

→ Collapse Hierarchy or Inline class → fix it

↓

destroy the inheritances.

11) Speculative Generality

→ Abstract class? virtual func^n? pure virtual func^n?

→ Create a class which isn't doing anything currently but ~~still~~ we speculate that it will come handy someday.

12) Middle Man

13) Inappropriate Intimacy

→ ~~A class~~ Two classes using each others variables excessively.

→ feature envy in both directions.

14) Alt. Class with diff Interfaces.

15) Data Class.

→ dump all data into a single class & other func^n use this class. for their functionality.

→ encapsulate data

→ data class makes data separate & functions separate & make them interact together

16) Refused Bequest

→ Subclass doesn't require something sent by the super class.

→ Bequest: the things which are passed on when someone dies.

17) Comments

int a; // age ⟶ int age;

What is testing? Software testing?
S/W testing is a negative activity → start with the hope of finding vulnerability.
If you do testing & find no errors → testor gets blamed.
testing gets done by third party. Developers don't know who the people are who will be testing. Devs can't do testing.

Testing V/S Review & Inspection
In agile, not much focus to testing. But in strict process there is importance given.
Agile only does review & inspection

Testing requires test cases.
What are test cases?
We always ~~write~~ check input domain before testing.

get representative test cases from the input domain (its subset)
the test cases should cover all combinations.

the O/Ps of the test case are checked.

How do you know the correct O/P? → Oracle
Oracle can be a person, client, developed S/W etc.

Testing is done in phases:
1) Unit testing
2) Integration / Component testing
3) System testing.

Unit testing → testing a single unit like class, func^n etc.
tested by the developer at the time of dev.
testing: Driver → S/W module developed. This module calls the particular unit which is to be tested and also feeds the data along.
Stubs → When a func^n gets called by another func^n. Then

Component testing: group of modules. Incrementely integrate the units

Retification of errors → debugging. } difference
testing → helps you find errors.

System testing: integrate enternal systems & everything.

After all testing & dee then deliver the software to the third party testing. The 3rd party gives a % on how accurate the product is.

SLA → service level argeement.

CS-204 DAA

Max flow / Min Cut theorem

$|f \uparrow f'|$ is a flow

1) $|f \uparrow f'| = f(u,v) + f'(u,v) - f'(v,u) \quad \forall\ u,v \in E > |f|$
2) $|f| = f(S,T)$ for any cut (S,T) in G st $s \in S$ $t \in T$
3) $|f| \leq c(S,T)$

Max flow / Min. Cut theorem:
the following 3 statements are equivalent:
1) $|f| = c(S,T)$ for some cut S and T of the network for $s \in S$ $t \in T$
2) f is the maximum flow
3) f admits no augmented paths.

$1 \Rightarrow 2$
$2 \Rightarrow 3$
$3 \Rightarrow 1$

$3 \Rightarrow 2$ tells correctness of ford f... algo.

Max flow achieved when min cut $1 \Leftrightarrow 2$