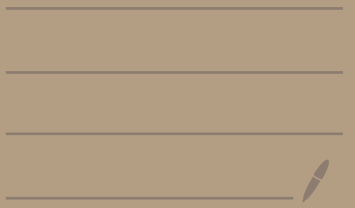
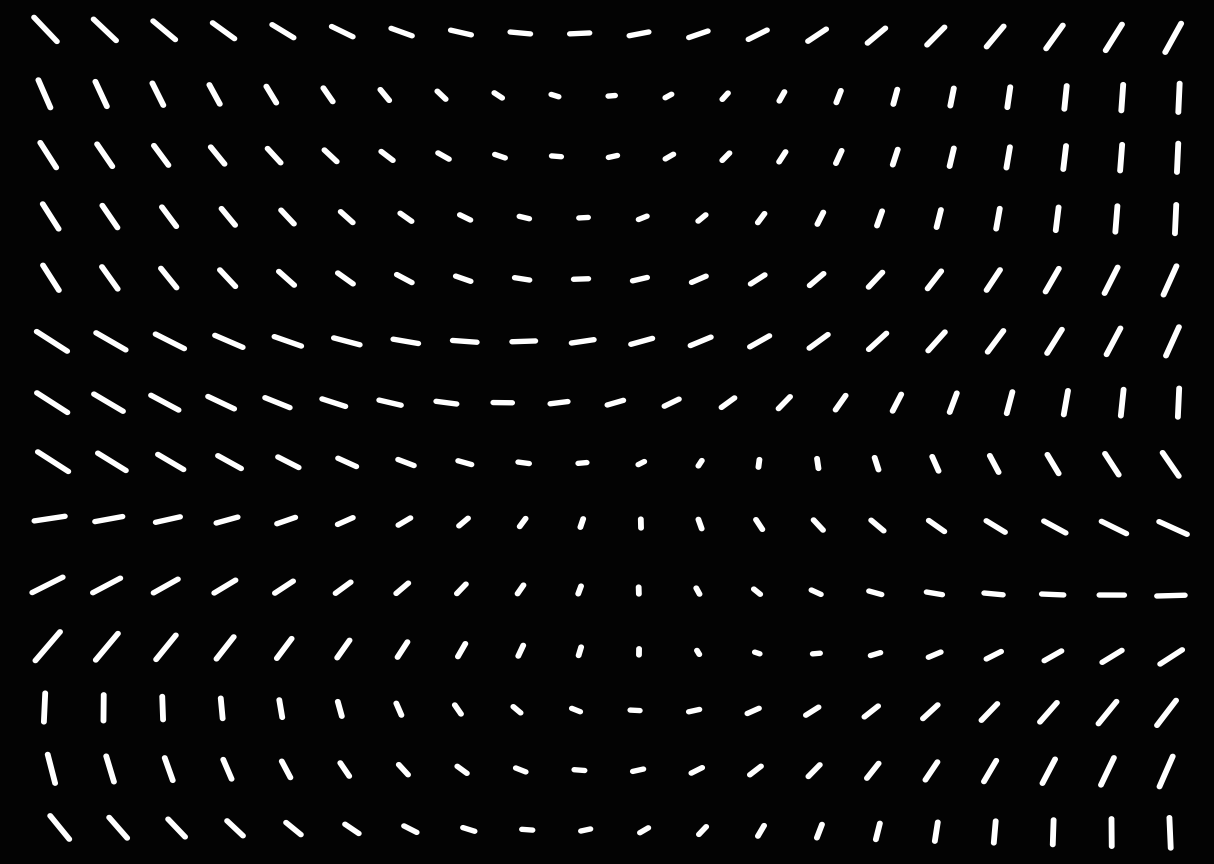
CISC 322



Lecture Notes (condensed)



Week 1: Introduction to Software Architecture

- Understand architecture of large systems and how to design, organize, maintain and evolve them.

- Topics
  - ↳ Challenges of Large Systems
  - ↳ Architectural Views
  - ↳ Evaluation Methods

Modern Software Systems are Large

- Photoshop, heavy applications have millions of lines of code
- Softwares are very large, hundreds millions LOC
- growth overtime

System Diagram

- Shows actual model code
- Configuration, Data Collection, Resource Management
- Most code is not in model, supporting ecosystem makes it usable

IEEE Definition

- Architecture. fundamental organization
  - ↳ Components
    - Relationships
    - Environment
    - Principles

A System which holds a collection of functions

Kruchten has a definition of Architecture. Which is mostly Significant decisions

Reference Architecture - generic template for an app domain

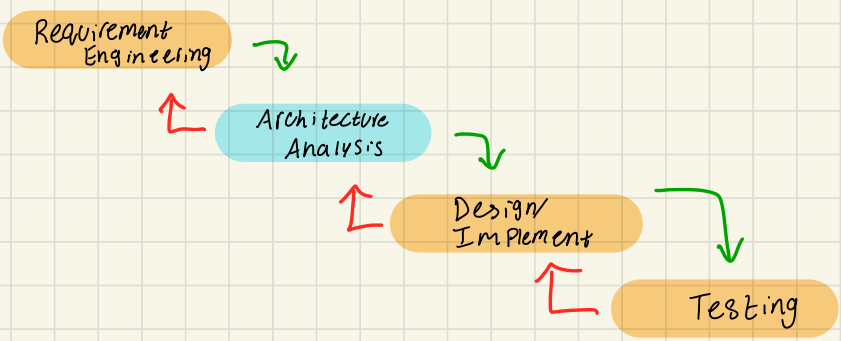
Product Line Architecture (PLA) - Reuses architecture for Similiar family of products.

Architecture vs Design

- Structure of system
  - ↳ High level and hard to change
  - ↳ Both technical and non technical
  - ↳ very early in life cycle

Design:
- ↳ Inner Structure
- ↳ Low level, interfaces help it change
- ↳ Late in life cycle

Software Architecture - High Level



Week 1: UML Slide Show

- UML (unified modelling language) is the standard way to visualize, specify, document and communicate. (Blueprints)

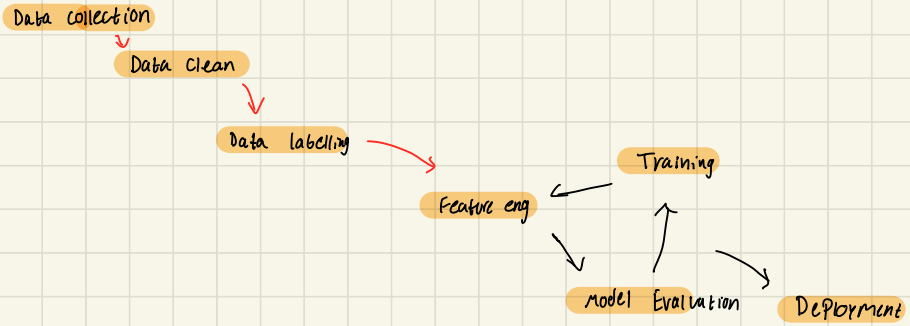
- 2 Main Types: Use case Diagrams
  - ↓
  - What system does
  - who interacts

- Sequence Diagrams
  - ↓
  - how components interact

Box and Line Diagrams:
- ↳ Show components and dependencies
- ↳ Arrows show flow of control or data
- ↳ Unidirectional arrows

Few Diagrams

Box and Line Diagram for Machine Learning Project



#2)

Consider a mobile Banking App. In this app, a customer can check their account balances, can transfer funds between accounts, and can make payments (either from a saving or checking account). Of course, the bank would only authorize a transaction if there is sufficient money in the account. It is also possible to apply for a credit card, or to sign up for a new account.

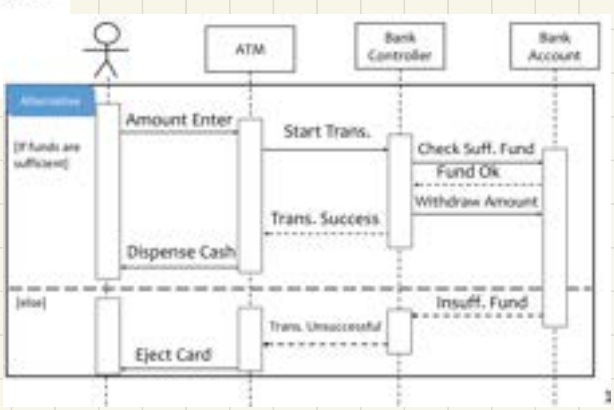
What would be a good use case diagram of this system?

#3)

Suppose you are designing an ATM transaction system. In this system, a person goes to an ATM machine to get money using their debit card. We know the ATM machine communicates with a Bank Controller to obtain access to the person's Bank Account. Please draw the sequence diagram of this system.

Actors?

Objects?



Use Case Diagrams

- what system does (use cases) and who interacts with it
- 3 major components:
  - 1) System → outer box
  - 2) Actor → stick person
  - 3) Use case → oval
- Each actor interacts with each use case atleast once.

Sequence Diagrams

- Shows interactions between components
- In right order
- Behaviour of system

2 Types of Messages

- → Synchronous, sender waits for reply
- → Asynchronous, solid arrow open head, no wait
- <- - - Reply, sent message other way

Conditional flow = If, else

parts

- Actor always outside scope
- Object (box)
- Lifeline (dotted box) (seavential)
- Activation Box (how long) an object takes to complete a task
- Message - communication between objects.

Week 2: Requirement Analysis

- understand what user wants and system before build begins
- Requirement engineering → SRS → Architecture Analysis
- Errors in requirements amplify downstream

- Fixing bugs is Cheaper/faster in requirement Stage
- Software Specification: Bridges real world ↔ technical system
- must watch for ambiguity and edge cases

Stakeholders

- Provide input, different interests, be specific, measurable, testable
- People have different interpretations of requirements

Types of Requirements

- Functional → what system does
- Non Functional → constraints

- System perspective Diagrams
  ↳ Shows whats in system and whats out.
  Example constraints slide *

Non Functional Requirements (NFR)

- Quality attributes, define how well system performs functionality
- Functional = What it does
- Non Functional = How it does

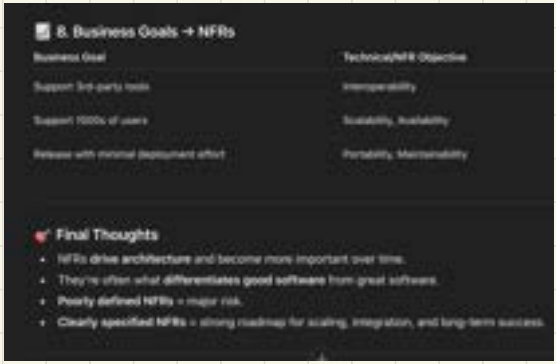
Examples of NFRs:

Category	Example
Performance	95% of searches return results in < 2 sec
Accuracy	ML model must predict > 90% correctly
Portability	Code should run across platforms or reusable libraries
Interoperability	All data must be in XML / support SQL
Usability	Must be usable for non-tech users (e.g., < 1 hr to learn)
Availability	System must be available 99.999% (a.k.a. five-nines) uptime

NFR's Should be
- Clear, Concise, Measurable
  ↳ Impacts:
  Code Structure
  Subsystem Seperation
  Framework

ICDE - Case Study

- Information Capture and Dissemination Environment
  - originally note taking app
  - now supports goals and NFR, Plan for multi-user, large scalable deployments.



Flowershop Examples

performance

- Should be able to handle a lot of online traffic
- information should be almost live
- prices should be accurate
- no bugs

Scalability

- Data, handling sensitive info like credit card numbers
- Scale to many users, bigger database, new customers
- Deployment thru parent company

Other NFRS

- Response Time
- Low lateney
- Scale for high/peaking throughput on holidays

Week 4 / Module 4 - Concrete vs Conceptual

- Conceptual Architecture: how developers think the system is structured (blueprint)
- Concrete Architecture: what the src code actually implements.
- Rarely perfect match, which is where reflexion analysis comes from

- usually mismatch is caused by missing relations in conceptual, added functionality over time, developer expediancy.

- Conceptual architectures captures functional view, high level abstraction and only shows meaningfull relations

Concrete architecture reauires static analysis of source code
↳ keep control flow, data flow

Concrete Top-Level Architecture: has way more arrows (dependencies) compared to conceptual

File Systems

- concrete file systems deal directly with real hardware, disk blocks
  ↳ coupling means many parts depend on it
- Logical file systems interact with many subsystems
- Device driver = deep integration

5 Reasons concrete ≠ conceptual

Week 6/Module 5: Reference Architecture

What is Reference Architecture

generic blueprint for entire domain

- A reference architecture is a template architecture defines common, components and relation between them
- Compilers and operating systems have good ref architecture
- Reference Architecture is reusable, Product architecture is a specific instance of this template adapted to Products requirements.

Benefits of Reference Architectures

- Documents well proven designs
- common vocabulary
- improves Code reuse

Web Servers

- Web servers like Apache, Jigsaw, ILS differ in details but share common domain-level components
- Perfect for reference architecture

Core Web Browser

- UI, Browser engine
- Networking, XML Parser
- display backend

How to derive a reference Architecture

1.) get conceptual architecture
2.) get concrete architecture
3.) compare them (link common components, etc)
4.) Derive reference architecture
5.) map each system against the reference Architecture

Web Server Reference Architecture

web server vs Browser?

- reception, resource handler
- request analysis, access control
- 3 web servers map cleanly (some split subsystems/others merge)
- In general components are splitted/merged/renamed but have same functionality

Week 7 - Reflexion Model (+ important concepts)

Reflexion Model

- lightweight technique to compare Ideal architecture with the actual implementation
- maps code-level dependencies to conceptual components
- where conceptual architecture is correct
- where code unexpectedly deviates
- where conceptual expectations are not implemented

1.) Propose

- build or assume conceptual architecture
  - ↳ comes from domain knowledge

2.) Compare

- extract concrete architecture from source code
  - ↳ • Convergence (usually not concern)
  - • Divergences (must investigate dependencies)
  - • Absences (rarely occurs in large systems)

3) Investigate

- Why do these differences exist
- Modify conceptual/concrete architecture
- Iterate

Absences

- In conceptual, not in actual
- Causes: feature removed, refactoring, dead code

Absence ≠ missing implementation

Convergences

- a dependency exists in the conceptual architecture and is also found in concrete code
- Implementation matches Design

Divergences

- Actual code differs than what you thought

Reasons: developer shortcuts, extra features, incorrect conceptual

- Divergence ≠ always bad

Sticky Notes vs Git Blame

- methods for investigating mismatches in reflexion analysis

Sticky Notes method (where)

→ code line level commits, should apply to each commit then compare and filter

- manual investigation technique
  - ↳ print dependency reports, mark specific source lines with sticky notes
  - ↳ tracks responsibility for each dependency

→ This is very accurate and helps understand why dependencies exists

→ But, Its time consuming and hard on large systems.

* Historical Symbol Table

Git Blame (why)

- git blame (version history)
  - ↳ can see specificly who added what and commit message
  - ↳ Strengths: fast, easily available
  - ↳ weakness: not always reliable, format changes, refactors rewrite blame history

---

When to modify Conceptual vs Concrete Architecture

Modify Concrete Architecture
↳ when the code is wrong
Ex. Undesired coupling, violates intended design, accidentally added
Fixes: moving functions, refactoring modules, introduces interfaces

Modify the Conceptual Architecture
↳ when conceptual model is outdated
Ex. System evolved and new dependencies are legit
Fixes: Adding new components

Mapping Views

Conceptual → Concrete → Deployment → code

We need multiple views because a single architectural diagram can't capture everything.

Conceptual views, code view, connector view, deployment view

Drift vs Erosion

Drift - Conceptual architecture is not updated when changes occur but implementation conforms to architectural principles.

↳ new subsystem added but not reflected in diagrams

Idea is documentation lags behind reality

Conceptual view becomes unreliable, but still works architecturally

Erosion - violates conceptual architectures constraints

Ex - UI accesses database

- cycles formed between subsystems

The architecture is breaking down

Consequence: hard to maintain, inconsistent behavior

Subsystem Boundaries

- subsystem is a cohesive group of responsibilities
- good subsystems: high cohesion, low coupling, clear interfaces
- bad boundaries: responsabilities unrelated, depend too heavily on each other

Responsibility Allocation

- must be allocated based on:
  - what changes together
  - who needs access to what data
  - what NFR's matter

---

unexpected Dependecy

get from week 7 ✱

Table

- stick Notes: → limitations:
  - First Note — availity of comments and commit messages
  - Last Note
  - All Notes

Module 7 / Week 8 - Project Scheduling

How to schedule tasks in Software project using formal techniques (PERT, CPM)

Why Schedule

- Projects have tasks with dependencies
- Resources are limited
- task order matters, earliest finish time, critical path

Task Graph

- Directed Acyclic graph (DAG)
  - ↳ Nodes = tasks
    Edges = dependencies
  - Each task has duration

Ex. A → C → D
B → C
This means C needs A, B to be done first.

Earliest Start and Earliest Finish

- (ES) = max(EF of all predecessors)

If a task has no predecessors → ES = 0

- (EF) = ES + duration

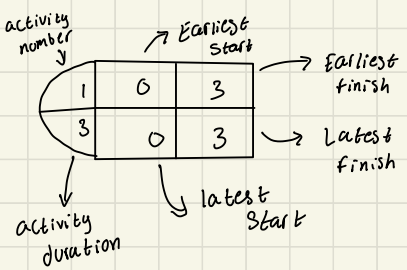
Determines how soon tasks can be done

Latest Start and Latest Finish (LF)

- Performed with backward Pass
- LF = min(LS of all successors) → if last task in Project. LF = EF of final task
- LS = LF - duration
  - ↳ tells you how late tasks can start

Slack = LS - ES = LF - EF

* Tasks with 0 slack = Critical Path tasks



Critical Path Method

- Critical Path = longest path through activity graph
- Slack = 0
- delay on any task → whole project delayed
- Often multiple Critical Paths
- To invest money into a project, normally you should invest in tasks that speed up overall Project

If Slack = 0, your on Crit Path

activity slack = LS - ES = LF - EF

PERT Estimates (Three Point Estimation)

- Expected Duration: $(te) = (O + 4M + P)/6$
  - O = optimic estimate
  - M = most likely estimate
  - P = pessimistic estimate
- Variance = $((P-O)/6)^2$
  - ↳ used for risk analysis

Common Risks

- If two tasks require same person
- resource leveling
- late discovery of hidden dependencies → delays
- wrong assumptions

Triple Constraints of Project Planning

- Time
- cost
- Scope (functionality)

} can optimize 2, but not all 3

Never make promises in early stages of Project, error rate 4x

How to fix delayed Project

- add resources
- reduce scope
- Extend schedule

Work Breakdown Structure (WBS)

- full list of activities
- Break activities into sub-activities

Problems: too many / little levels

3 Standard ways to make WBS

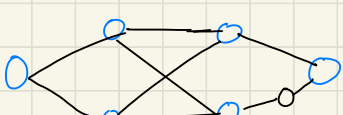
- Phase Based
- Product Based
- Hybrid Based

Gantt Charts

- horizontal timeline
- Bars show task durations
  - ↳ helps visualize parallel tasks

Activity Float

- time allowed for an activity to delay
- Total Float (without affecting project completion)
- Free float (without affecting, next activity)



A: 9
B: 8
C: 8
D: 7

} min amt of time needed to finish the Project

Week 10 / Module 9 - AI Powered Systems

AI Models

- Black box function maps inputs to outputs
- Harder to implement
- Many capabilities such as perception, cognition, learning

LLM Basics

- Encoder learns structure/patterns from inputs
- Decoder generates outputs token by token
- These make transformer style architecture

Software Progression

1.0 - Traditional coding

2.0 - Neural networks (datasets + architecture)

3.0 - API's, prompts

4.0 - Agent systems (autonomous workflows built on LLM's)

Risks/challenges

- Bias
- Hallucinations
- Safety failures

3 Pipelines

1.) Code Pipeline

2.) Data Pipeline

3.) Model Pipeline

New NFR's

- Safety
- Explainability
- Reliability

Model-centric vs Data Centric

- Model centric (focus on improving model)
- data centric (get better data) (usually better)
- data flywheel: feed data to model, find errors, then fix errors, then repeat

Feature Store

- Centralized repo for managing and versioning ML features
- Supports: high throughput API and low latency API

Week 10 / Module 9 - AI Powered Systems

- Idea: AI systems are surrounded by a lot of code infrastructure and don't have much code themselves

Where do Models Live?

- Commercial remote model (OpenAI, Gemini)
- Open Model Deployed Locally
- Open Model Fine-tune In house

Architectures* for inference Layer

Most Details on Cheat Sheet

- Model as a dependency
- Model as a Service
- Hybrid local + remote
- Offline batch prediction
- Online learning + Streaming
- Machine Learning as a Service

Week 11: Advanced Architecture for AI Systems

LLM's need Advanced Architecture

- LLM's introduce many new problems like accuracy, hallucinations
- LLM's are non deterministic (prompts matter)
  - ↳ need guardrails

Retrieval-Augmented Generation (RAG)

- adding static/domain specific knowledge to an LLM
- uses embedding model, vector store, chunking

Prompt Engineering

- zero shot / few shot (give examples to LLM)
- could increase cost, prompt length

Caching Patterns

- Why cache outputs; many queries repeat, reduce GPU cost
- Cache challenges: cached answers become stale

Cache aside pattern

- Check if saved, if not save it, return answer

- FrugalGPT is a way to save money and speed up responses by using cheap model
  ↳ ask cheap model, if no good ask mid model, if not good ask good model

  This is Confidence based routing

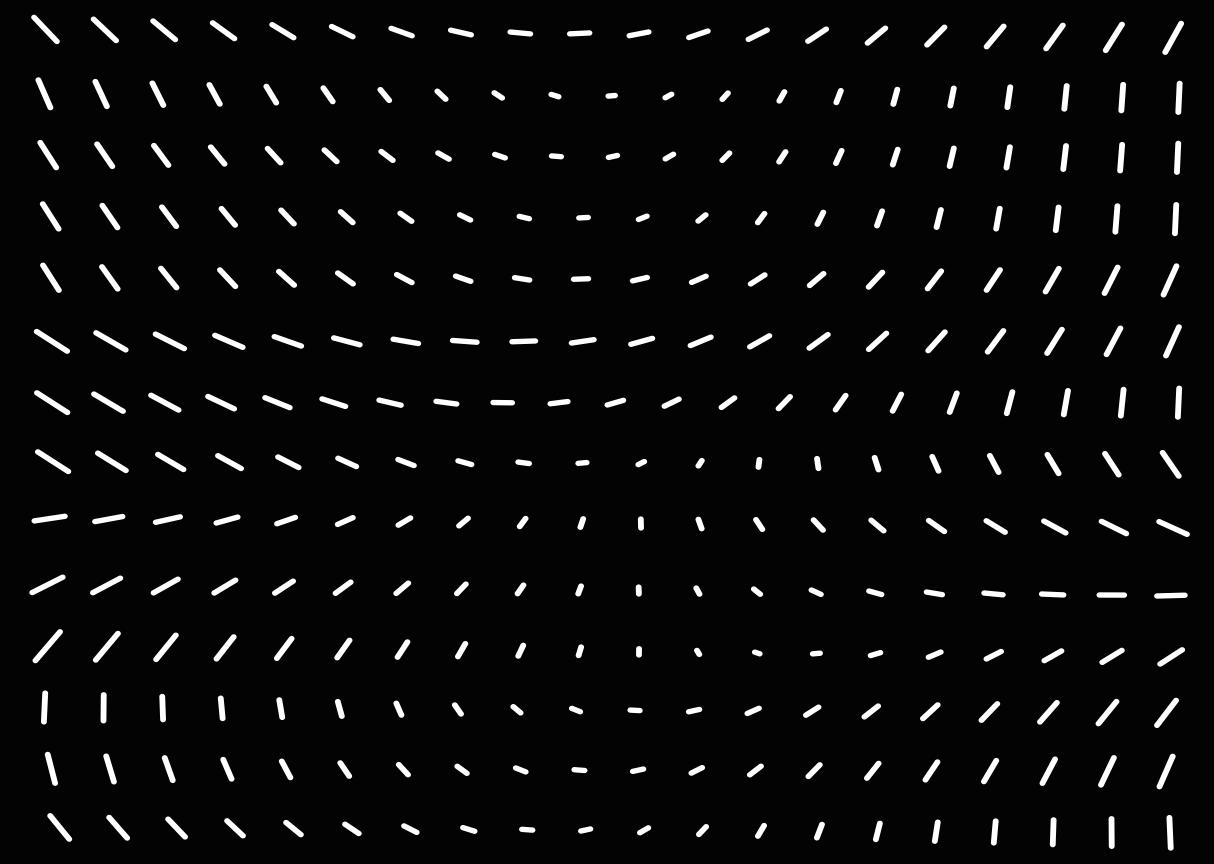
LLM System metrics

- latency, throughput, TPOT → system
- Accuracy, Hallucination rate → model
- GPU memory, cost → resources

★ Model Drift means model gets worse overtime

- Circuit Breaker is a Safety Switch for the LLM
- States: closed, open, half open
- Model migration: To switch to new version of LLM, tests wit 1-5% of users first and check for bugs
- Types of guardrails: input, output, policy and workflow

Midterm Review



Winter 2023 - Practise Test

1) a) This type of NFR is scalability and being able to withstand more users and the fact that there is not sufficient resources to allocate distributed bandwidth. The system can't take the load. (Also related to availability)

b) To address the scalability / availability issue, 2 typical approaches are horizontal scaling and reallocation of resources. (Basically scaling up and out)
Scaling up - stronger CPU, GPU, compute, RAM.
Scaling out - more servers, more nodes,

c) In the case of ChatGPT, they should scale out, get more processing, more servers and have it so that more users can work in parallel.

d) In the context of Chat it's impossible to focus on every NFR with equal priority because of the compute, maintenance, ever evolving technological models, increased work demands and the versatility required. Availability, takes priority and such others. (Trade OFF's between NFR's)
(cost and resource constraints)

2)

a) wrong, this is not specific/measurable enough. I don't know the currency, average vs peak? Amazon could sell more than that.

b) True, in object oriented they communicate closely between each other. Implicit invocation is event based

c) True, "environment" is everything affecting the system. Including deployment platforms legal constraints, policies.

d) False, should cover all views for more well-rounded software

e) True, All 3 layers (UI, logic, DB) can come from one machine

3)
a) operational view, affects runtime and deployment behaviour
b) Information view, Structure/flow of data
c) Development view, automate, execute
d) Information view, Structure of database machine
e) functional view, overall system

4)
a) Repository Architecture Style is heavyweight, overkill for this app. It allows us to manipulate a complex data model.

b) pub-sub style can be used only for events because its good for notification. Regular functionality of consulting todays job.

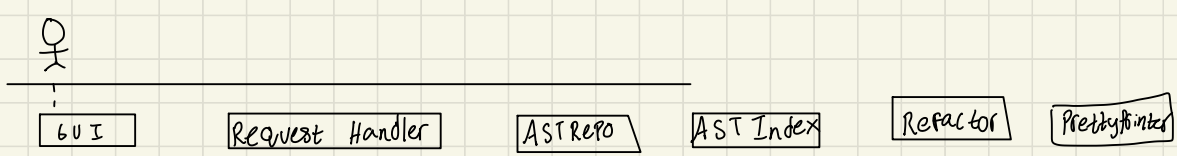
---

5) a) Response time, IDE should be updated in real time
Accuracy, should have all function names change along with reference
(Throughput / Integration) also works

b) Client/Server Architecture IDE = Client
Repository Architecture: AST (code structure) is manipulated centrally
↳ Interpreter applies refactoring commands on AST

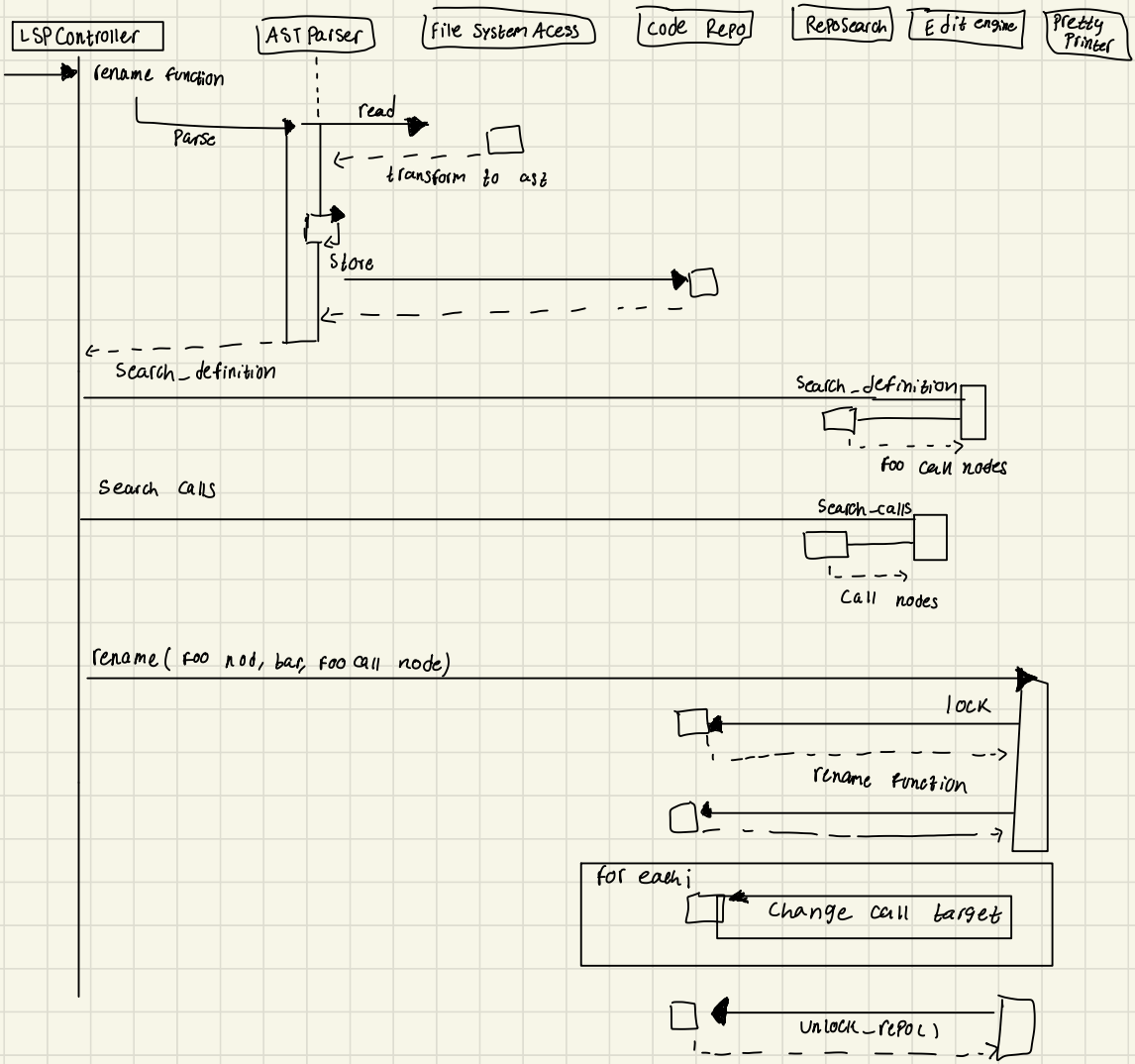
Seaverce Diagram

- Shows interaction over time, Showcase Seavences of messages
- Use Case: Select a function, rename it in the IDE, clangd updates all reference in your code

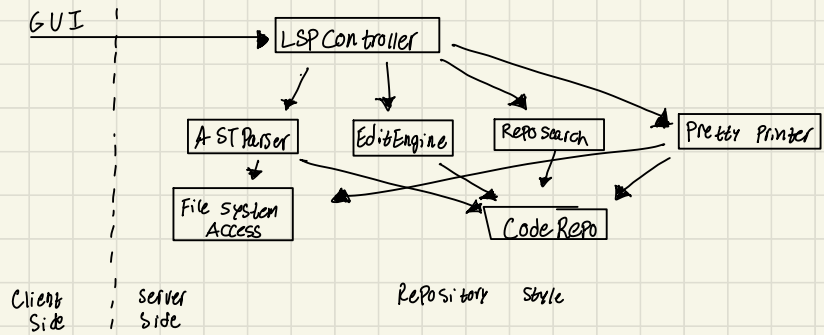


Seavence Diagram Real

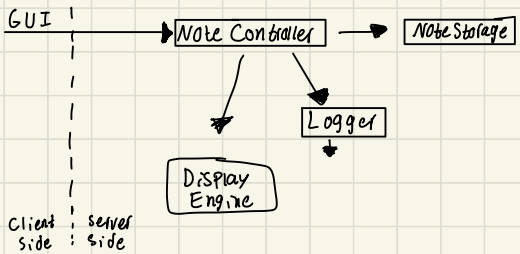
- Rename Functions Use Case



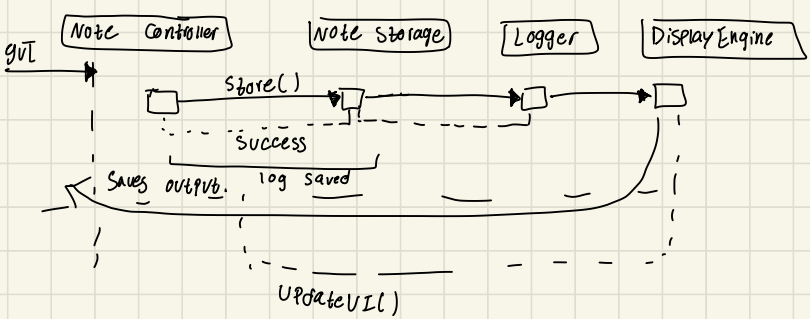
Box Diagram



Box Diagram Example:



Sequence Diagram



Practise Quiz Fall 2024

Part 1: Short Answers

a) This description relates to Scalability requirements as they rapidly gain users and need to allocate more resources and upscale their current services ✓

b) The major observation is that all application servers are staying the same and the TPS is not changing much. (TPS, decreases slightly)

c) The best way to solve this issue would be distributing application servers geographically and leverage cloud technologies to ensure they can handle increase no. of clients. (load balancing)

d) Non repudiation is relevant for deny/confirming if a user has sent or has not sent the message. Can be useful for investigation purposes

---

Part 2 - Right or Wrong?

a) False, the software requirement specifications document should not contain any version control/edit logs. It should have relevant info about the software. Also, they should know git/github, or a dif robust way ?

b) False, it's not enough to choose a style of architecture you must choose the style and functional elements, responsabilities etc.

c) True, $\frac{7}{9} > \frac{2}{9}$ of the time to be available but other aspects could be sacrificed such as security, or speed to make it more available.

d) Wrong, the repository style does not enforce execution order when order matters you want styles like Pipe/filter

e) True, functional requirements are easier to assign directly to a component, NFR are cutting concerns because there is a hard balance between diff NFRS

Section 1.3 - Architectural Views

a) Functional, explaining notification functionality.

b) Functional, hidden from students, enforcing behaviour

c) Development, built on Python, Config files, built organized

d) Informational,

e) Deployment, where and how things run.

Section 1.4

a) multi tier architecture, is better for scalability and maintainability

b) Both, object oriented style is when the people communicate with each other. modular and Client and Server network

c) Peer to peer is better for security, no central server.
Making sure system stays available

d) Integration, Scalability → Encrypt data locally

e) only makes sense with client server because there are multiple layers communicating with each other.

Part 2: Design Exercise

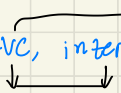
- Cross Platform, sound file
- Rythm is drawing App
- Rythm stops at each point
- musicians, new schools
- AI component, drawings → nice format

2.1

- integration, usability
  - Drawing music notation should be easy to use, UI/Ux too complex. Musicians will be turned off.
  - Portability due to cross platform neighbour. Real time audio application
  - Performance + Usability

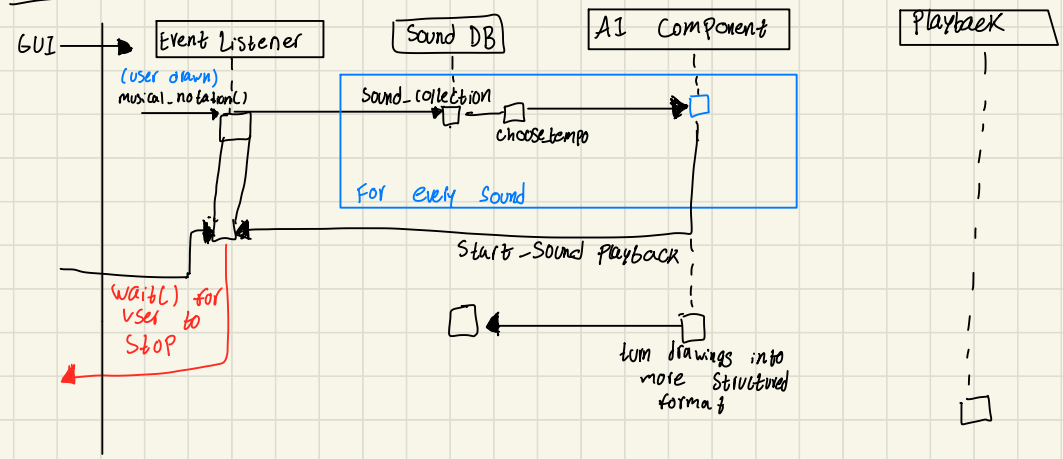
2.2

Repo, MVC, interpretor → UI focused best for web apps or OO to reuse tempos, drums,

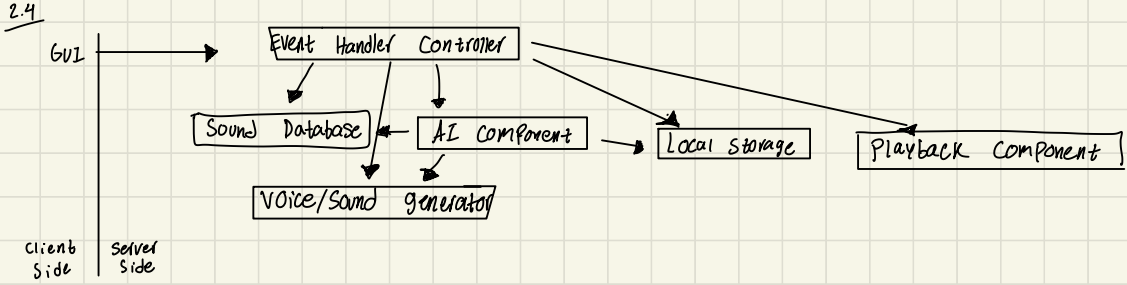


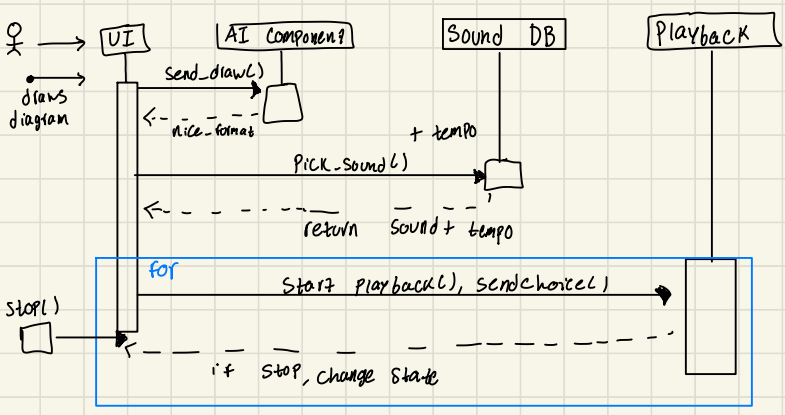
→ interpreter is easy to extend and modify the behaviour

2.3


GUI
Event Listener
Sound DB
AI Component
Playback
(user drawn) musical_notation()
Sound_collection
chooseTempo
For every Sound
Start_Sound playback
wait() for user to Stop
turn drawings into more structured format

2.4


GUI
Event Handler Controller
Sound Database
AI Component
Local Storage
Playback Component
Voice/Sound generator


Client Side
Server Side
UI
AI Component?
Sound DB
Playback
draws diagram
send_draw()
nice_format
+ tempo
Pick_Sound()
return Sound + tempo
for
Start playback(), sendchoice()
stop()
if Stop, change state

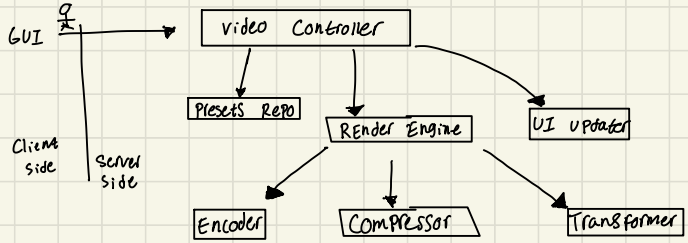
Design Exercise: Video Editing

1) Performance
modifiability
USability

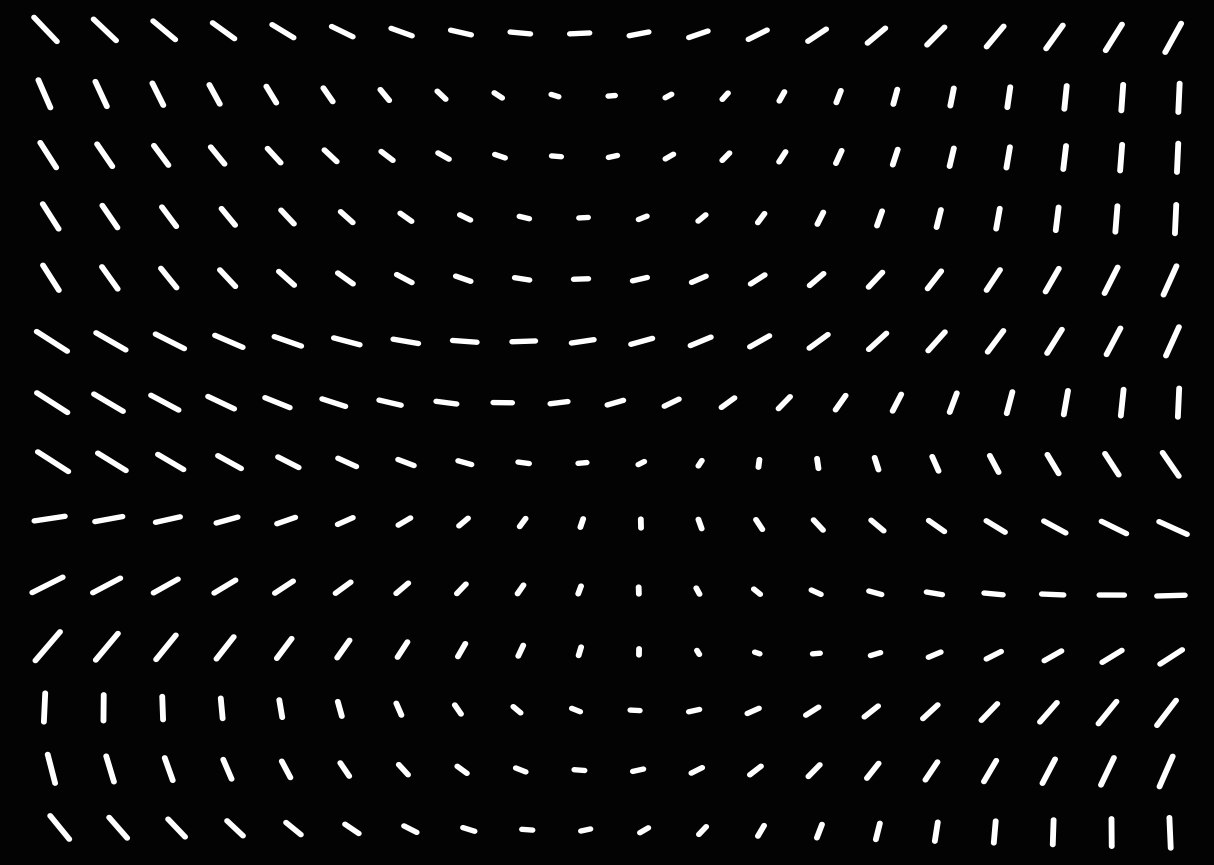
I would say the most important is USeability. in video editing, theres a lot of room to be creative and having a software that dynamically allows users to easily add affects and previews are essential

The second most important is performance as you should be able to handle video types, special effects and the performance should not tank

2) Pub-Sub
Object oriented
Pipe and filter



Final Exam Review



Final Exam Content

- Module 4 (conceptual / concrete architecture)
- Module 5 (reference architecture)
- Module 6 (reflexion analysis/models)
- Module 7 (project planning)
- Module 8 and 9 (AI powered systems)

} 3 hours

- Core Concepts:
  - reflexion analysis
  - sticky note vs git blame
  - reference architectures
  - Project planning and scheduling
  - Architects decision tradeoffs
  - use case conceptual architecture
  - AI-powered systems

All Rough work →

would reccommend practise Quizzes / Exams.

Project Scheduling Example - Rough Notes

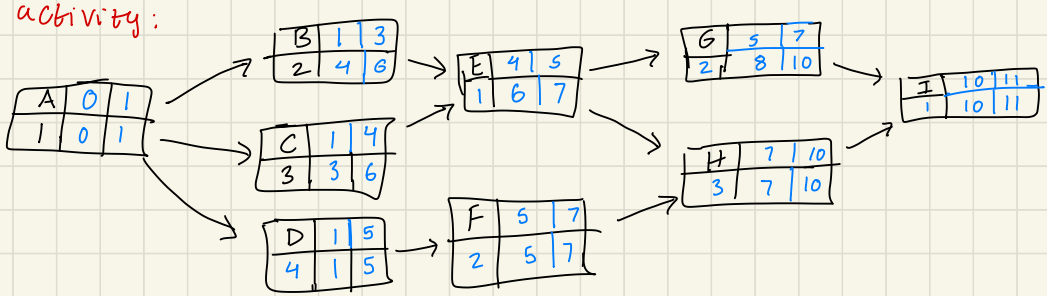
activity ID	a	m	b	depends on	t	σ²
A	1	1	1		1	0
B	1	2	3	A	2	1/9
C	1	3	5	A	3	4/9
D	2	2	14	A	4	4
E	1	1	1	B,C	1	0
F	1	1	7	D	2	1
G	1	2	3	E	2	1/9
H	1	1	13	D, E F	3	4
I	1	1	1	G, H	1	0

a - optimistic
m - most likely
b - worst case

$$t = \frac{a + 4m + b}{6}$$

$$\sigma^2 = \left(\frac{b-a}{6}\right)^2$$

1) Expected time for each activity:



1 + 4 + 2 + 3 + 1 = 11 weeks

Critical path would be A → D → F → H → I

a) Expected duration would be 11 weeks

b) Variance along Critical path activities is $\sigma = 0 + 4 + 1 + 4 + 0 = 9 \rightarrow \sigma = \sqrt{9}$

∴ Standard deviation is 3 weeks

c) $\mu = 11$, $\sigma = 3$ $\quad z = \frac{13-11}{3} = \frac{2}{3} \simeq 0.67$

$P(T \leq 13) = \quad z \simeq 0.67 \rightarrow$

$= 0.7486$

3.4) The act of speeding up a given

a) activity is Project Crashing

↳ would require more resources / time / money

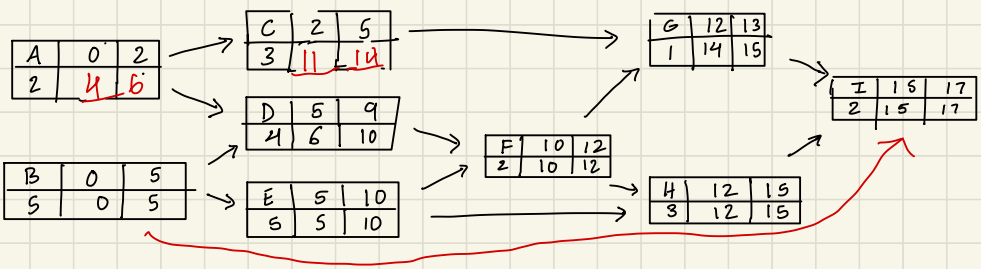
b) Yes the interns right, Shortening time of D, would Shorten the Project length

c) NO, new crit Path (Need to shorten C/E)

d) Delaying G, doesn't do anything

Winter 2022 - Quiz

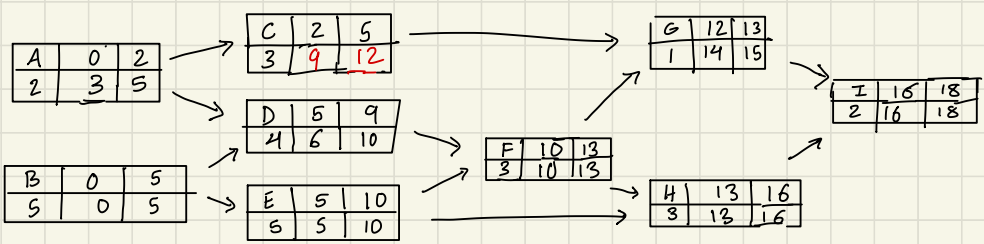
Activity ID	Duration	dependencies
A	2	
B	5	
C	3	A
D	4	A, B
E	5	B
F	2	D, E
G	1	C, F
H	3	E, F
I	2	G, H



Critical path: B → E → F → H → I

B → E → H → I

C 1
D 1
F $(4/3)^2$
G 0



B + E + F + H + I

$\sigma_B^2 = ?$

$\sigma_E^2 = ?$

$F = (\frac{4}{3})^2$

$\mu = 5 + 5 + 3 + 3 + 2$

$\mu = 18$

$\sigma^2 = (4/3)^2$

$\sigma^2 =$

$\sigma = \frac{4}{3}$

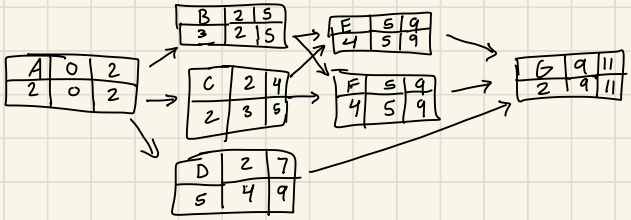
$Z = \frac{x - \mu}{\sigma}$

$Z = \frac{10 - 18}{\frac{4}{3}}$

$Z = -3.75$

Very small probability

Ex



A → B → E/F → G

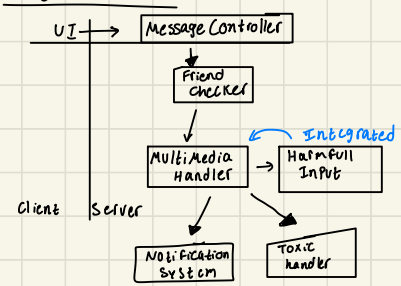
Should try to speed up any of these tasks

For Project crashing, ensure critical path does not change

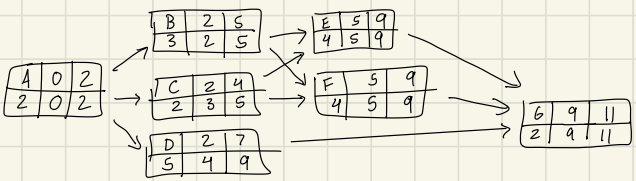
Time Cost Tradeoff *

Optimal crashing

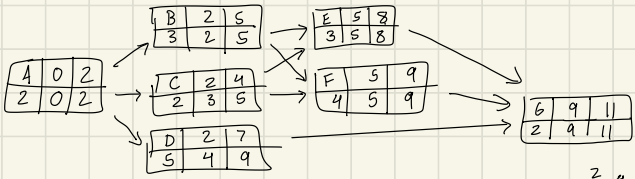
Rough work:



Winter 2021 Quiz - Scheduling



Critical Paths: A → B → E → G
A → B → F → G
} Important tasks, 11 weeks



Crit Path: A → B → F → G

$\sigma^2 = \frac{4}{9}$

$z = \frac{x - \mu}{\sigma}$

$\mu = 11$

$\sigma^2 = 1 + 4/9 + 1 + 1 = 3\,4/9$

$\sigma = 1.84$

$z = \frac{12 - 11}{1.84}$

$z = 0.54$

$\therefore$ 70%

$\sigma = 0.66$

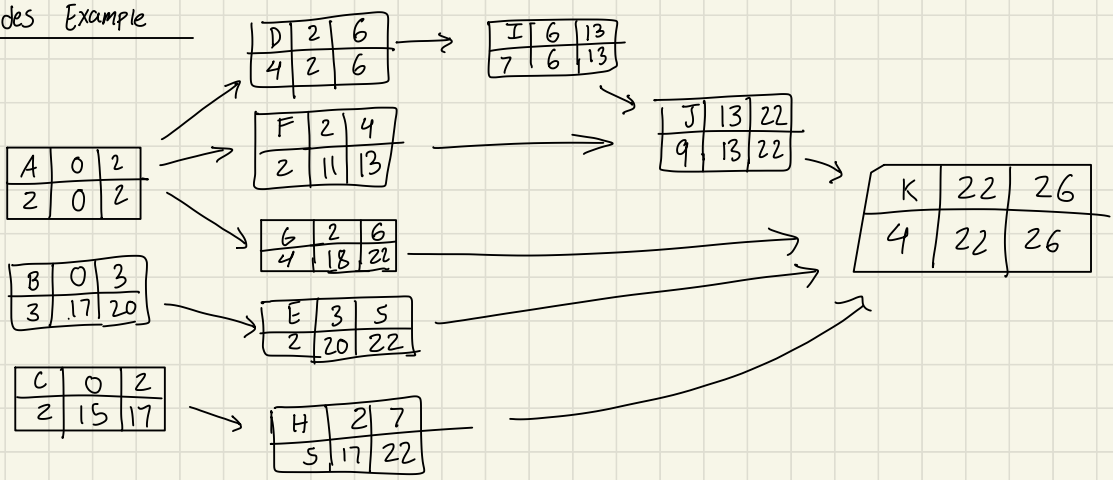
$z = \frac{12 - 11}{0.66}$

$z = 1.51$

$= 93\%$

No, it would not change Probability D.

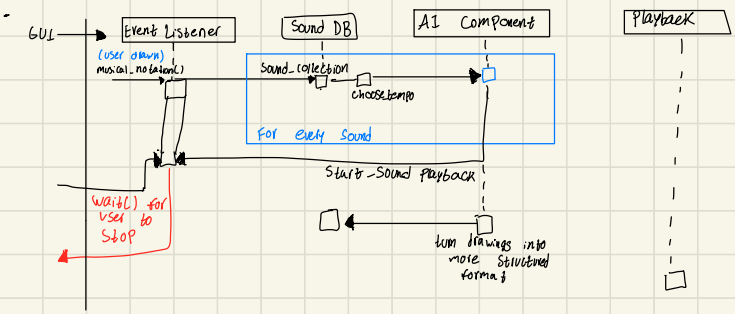
Slides Example

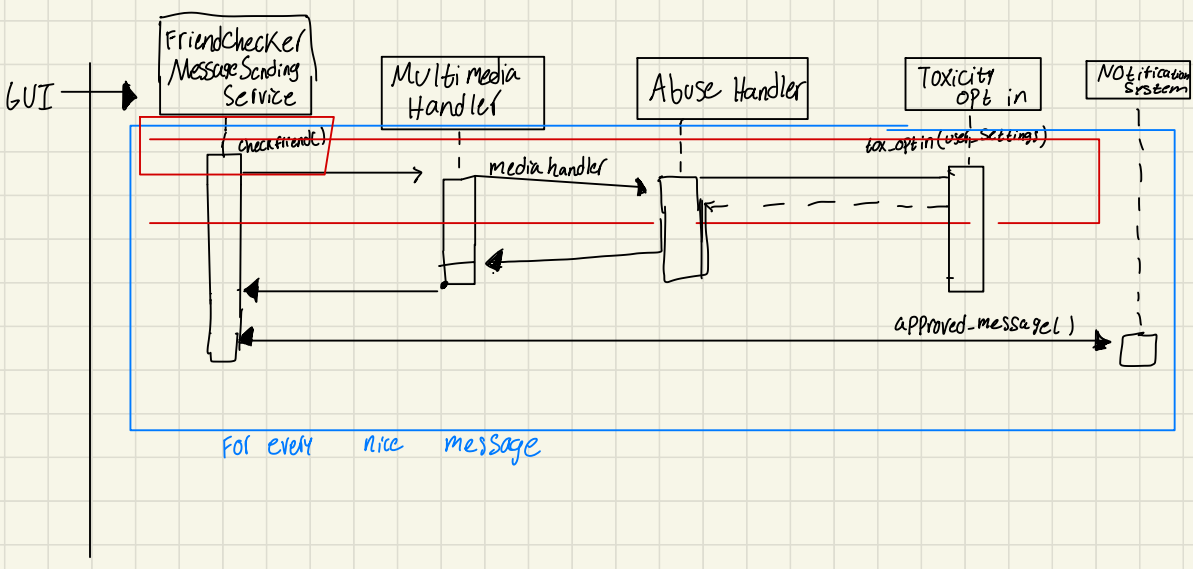


Critical path: A → D → I → J → K

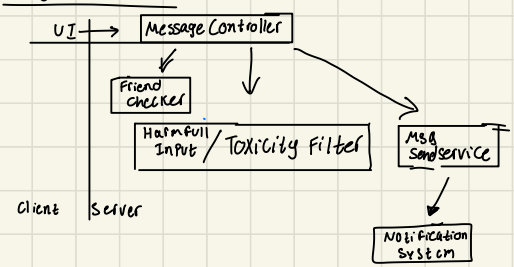
Winter 2023 - Diagram

- integrate messenger → Facebook
- only send messages to "friends"
- wide range of media
- Harmful content checker
- Opt in, discard, toxicity
- once user, gets, message, notification

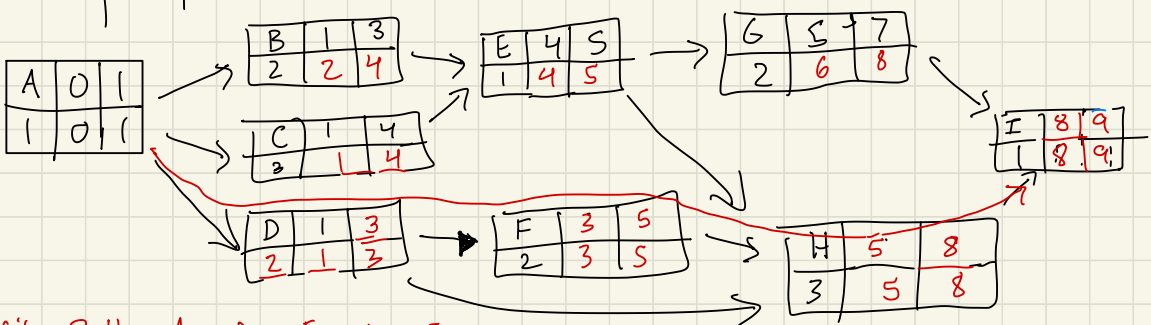




Rough work:



activity ID	t	σ^2
A	1	0
B	2	1/9
C	3	4/9
D	4	4
E	1	0
F	2	1
G	2	1/9
H	3	4
I	1	0



Crit Path: A → D → F → H → I

Expected Duration: 11 weeks

ACEHI } 2 crit paths
ADFHI }

σ^2: 0 + 4 + 1 + 4 + 0

σ^2: 9

$\sigma = 3$

$z = \frac{x - \mu}{\sigma}$

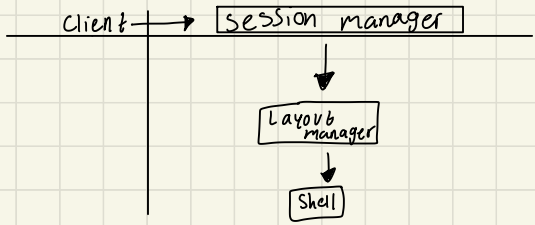
$z = \frac{13 - 11}{3}$

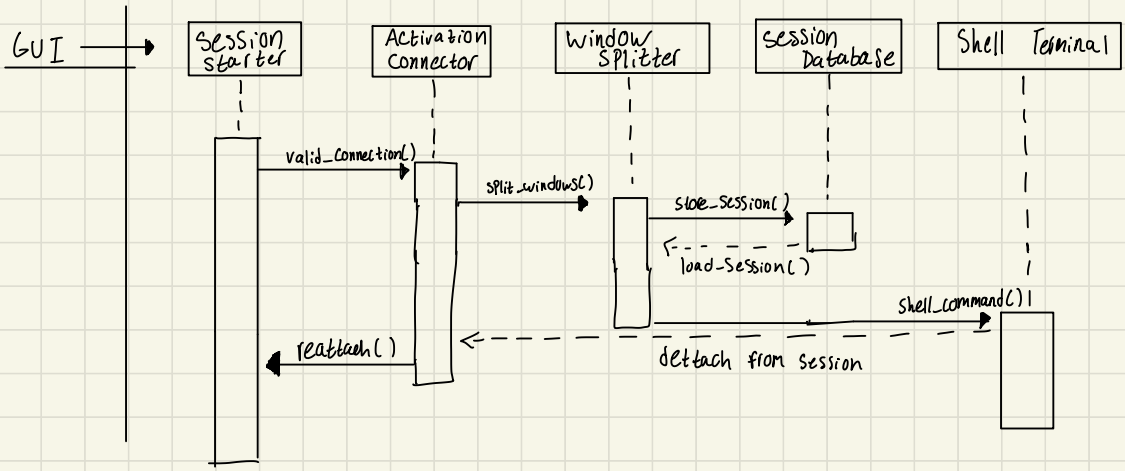
$z = \frac{2}{3}$

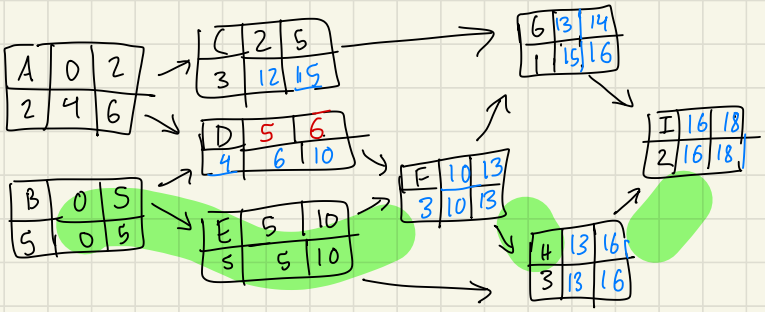
Probability = 0.7486

∴ 74% chance of finishing on task

Important Stuff
- window manager (physical terminal)
- split window layout (using command line tools)
- storing layout
- multiple virtual layout
- detatch from other sessions
- network risse







Crit path: B → E → F → H → I

Duration: 17

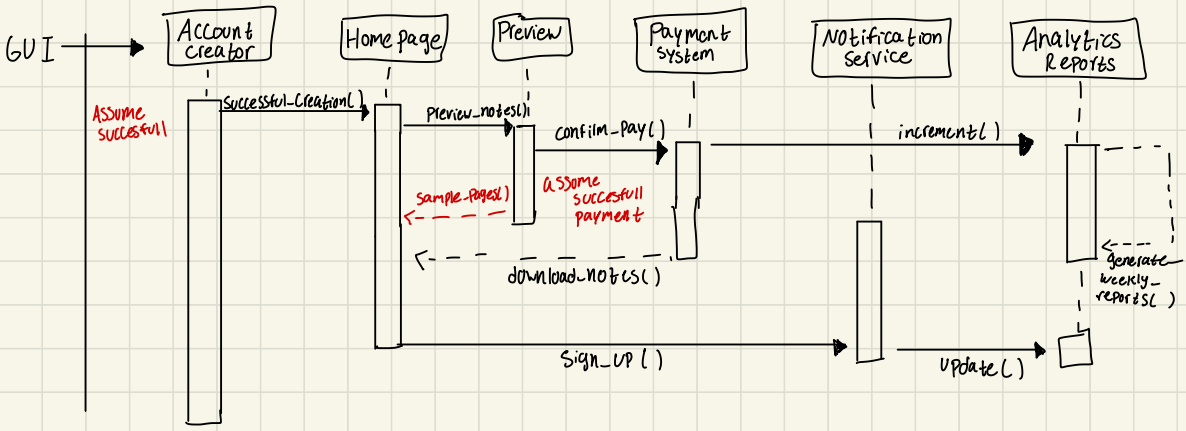
Dont speed up D, that shi dont matter lol

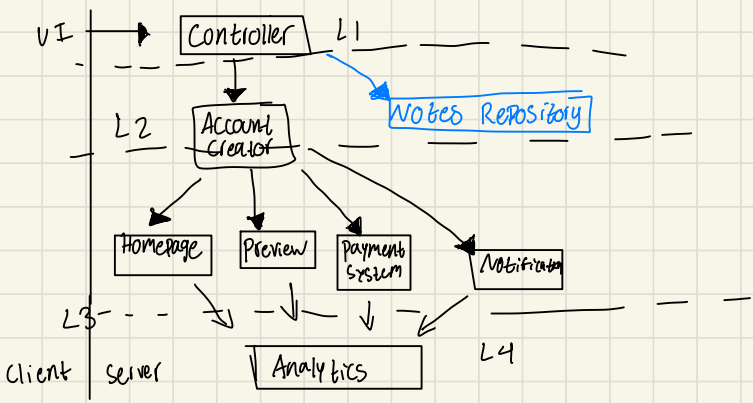
Probability($X \leq 10$): new crit path: B → E → F → H → I $t = 18$

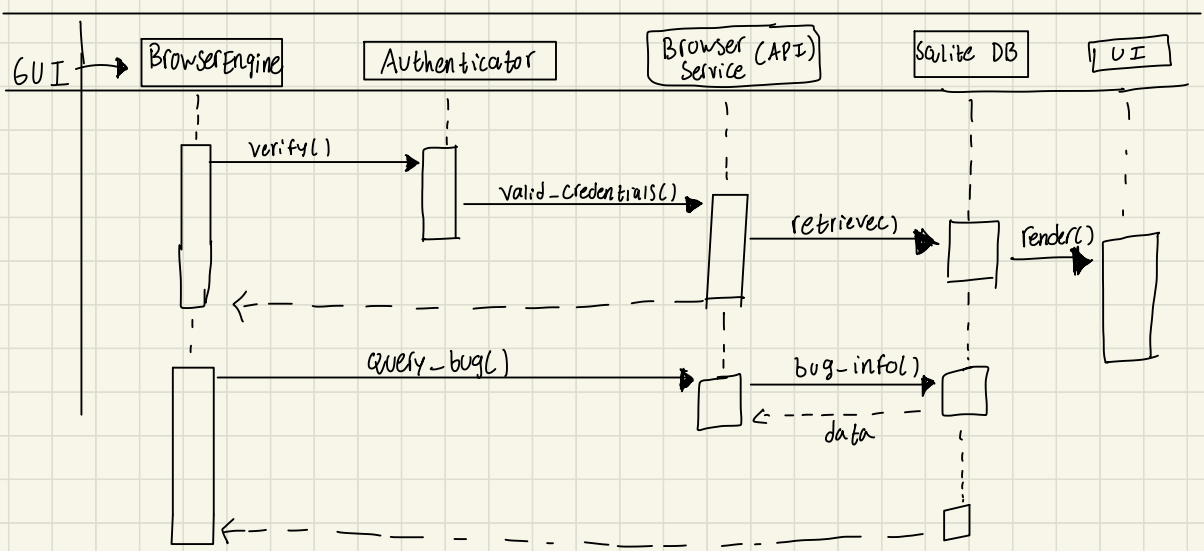
$$z = \frac{10 - 18}{\left(\frac{4}{3}\right)} \qquad z = -6$$

∴ no chance dawg

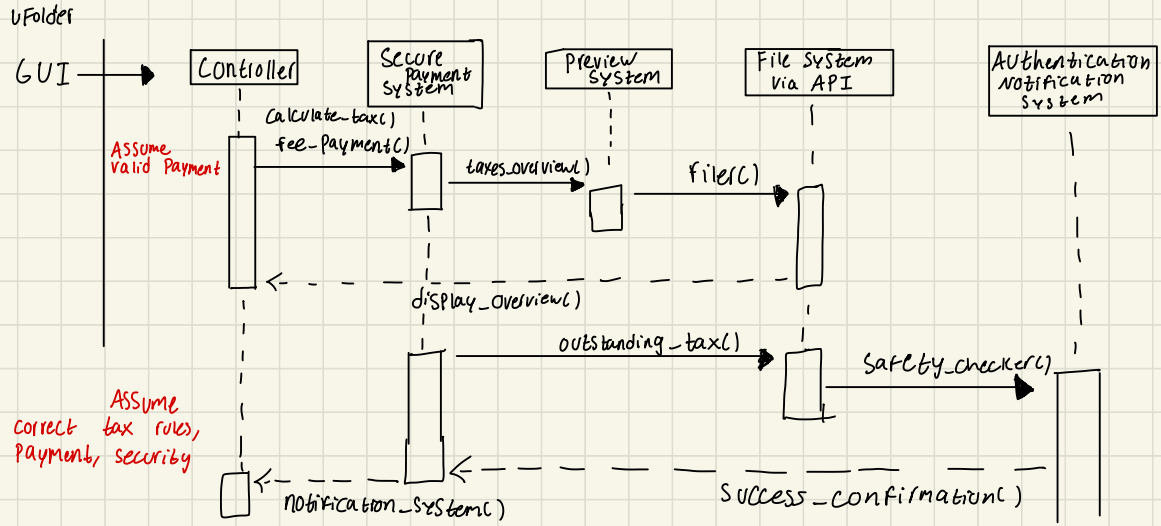
ur cooked even if u clash

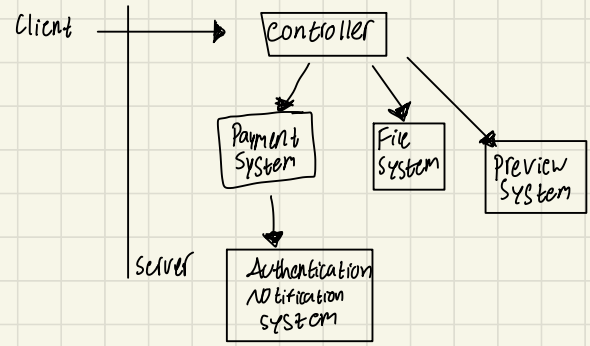

GUI
Account creator
Homepage
Preview
Payment system
Notification service
Analytics Reports
Assume successful
successful_creation()
preview_notes()
confirm_pay()
increment()
sample_pages()
Assume successful payment
download_notes()
generate weekly_reports()
sign_up()
update()


UI
Controller
L1
Notes Repository
L2
Account Creator
Homepage
Preview
Payment System
Notification
L3
Client
Server
Analytics
L4


GUI
BrowserEngine
Authenticator
Browser Service (API)
Sqlite DB
UI
verify()
valid_credentials()
retrieve()
render()
query_bug()
bug_info()
data

uFolder


GUI
Controller
Secure Payment System
Preview System
File System via API
Authentication Notification System
calculate_tax()
fee_payment()
taxes_overview()
file()
Assume valid Payment
display_overview()
outstanding_tax()
safety_checker()
Assume correct tax rules, payment, security
notification_system()
success_confirmation()


Client
controller
Payment System
File System
Preview System
Server
Authentication Notification system