Software Engineering

Unit 5

Agile Development and DevOps

Agile Development: Agile Teams - Team and Scrum - Branches and Pull Requests - Reviews - Integration - Agile Iterations - Reporting and fixing bugs ; DevOps - From development to deployment - Three - tier responsiveness - Service-level objectives - Apdex - Releases and feature flags - monitoring and finding bottlenecks - Improving rendering and database performance with caching - security - defending customer data in application

* Agile Software Development

* Issues with Traditional Software Development Methods

→ Systems are regularly delivered late or over budget

→ Customer hardly works with developers after the requirements phase.

→ Not possible to gather stable and unchanging customer requirements.

→ Customer is not always sure ~~of~~ what he wants

* Agile Method - a method of software development that aims for customer satisfaction through early and continuous delivery of useful software components.

Key differences between Traditional vs. Agile Development :

(i) Agile methods are adaptive rather than predictive

(ii) Agile methods are people oriented than process oriented.

(x) Agile Process Philosophy / 4 core values of agile manifesto

A. Individuals & Interactions over processes & tools

B. Working software over comprehensive documentation

C. Customer collaboration over contract negotiations

D. Responding to change over following a plan

A Individuals and Interactions

→ face to face meetings for faster transfer of ideas & quicker responses

→ can implement pair programming

→ Individuals are considered an irreplaceable part of the system

→ Team takes technical decisions together

B. Working Software

→ source code is the most important document

→ A working model is easier to understand than documentations

→ Maintainers go through the source code first before the documentation.

C. Customer Collaboration

→ Give preference to fixed price contracts

→ Customer is on site

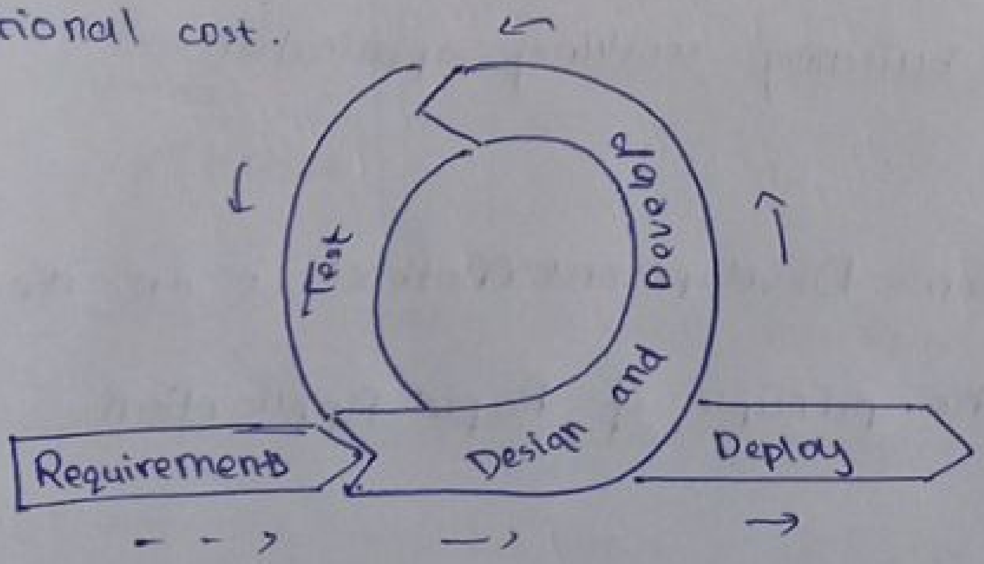
→ Customer has finer control over the project.

D. Responding to change

→ Respond to change rather than following laid out plan

→ Use a more iterative approach

→ Eliminate the difficulties of mapping new requirements to additional cost.



← Agile Iterations

* Methods of Agile Development

① Kanban - every individual in the team is allowed to make decisions related to the project

② Scrum - an iterative and incremental process to offer value to the customers

③ Lean - optimizes production and assembly lines by reducing waste

④ XP - Extreme programming collects the business requirements from user stories

⑤ Crystal - Project members are the key entity and a project flow adapts around them

⑥ FDD - Feature driven development is best for long term projects - it focuses on building working applications

⑦ DSDM - Dynamic Systems Development Method - is an iterative method based on the principle of Rapid Application Development (RAD).

* Agile Teams - Two Pizza and Scrum

Two Pizza → A two pizza team, is a team of size that can be fed by two pizzas in a meeting. The typical team size varies from 4-9 people.

**

Scrum → Mainly involves frequent short meetings everyday where each team member disscusses, what they had done since yesterday, what they are planning to do today, & if there are any impediments or stumbling blocks.

→ The benefit of those daily scrums is that by understanding what each team member was doing, the team can identify work that would helps others make progress that is more rapid.

→ Scrum uses what is called as sprints (short bursts of high speed activity).

→ A scrum has three main roles

① Team - A two-pizza size team that delivers the software

② ScrumMaster - A team member who acts as a buffer between the Team and external distractions, keeps the team focused on the task at hand, enforces team rules, and removes impediments.

(3) Product Owner - A team member (not the Scrum Master) who represents the voice of the customer and prioritizes user stories.

→ Scrum relies on self-organizaton and team members often rotate through different roles. (Changes usually happen every iteraton / sprint)

* Branches, Pull Requests and Reviews

A. Branches

→ allows one to take snapshots of work

→ can serve as backups

→ can manage multiple versions of a project's code base simultaneously, to allow each part of the team to work on an experimental new feature without disrupting working code, or to fix bugs in previously released code.

Creating Branches from Repos

→ When a repo is first made, by default it contains only a single branch called the main branch, on which a linear sequence of commits is made.

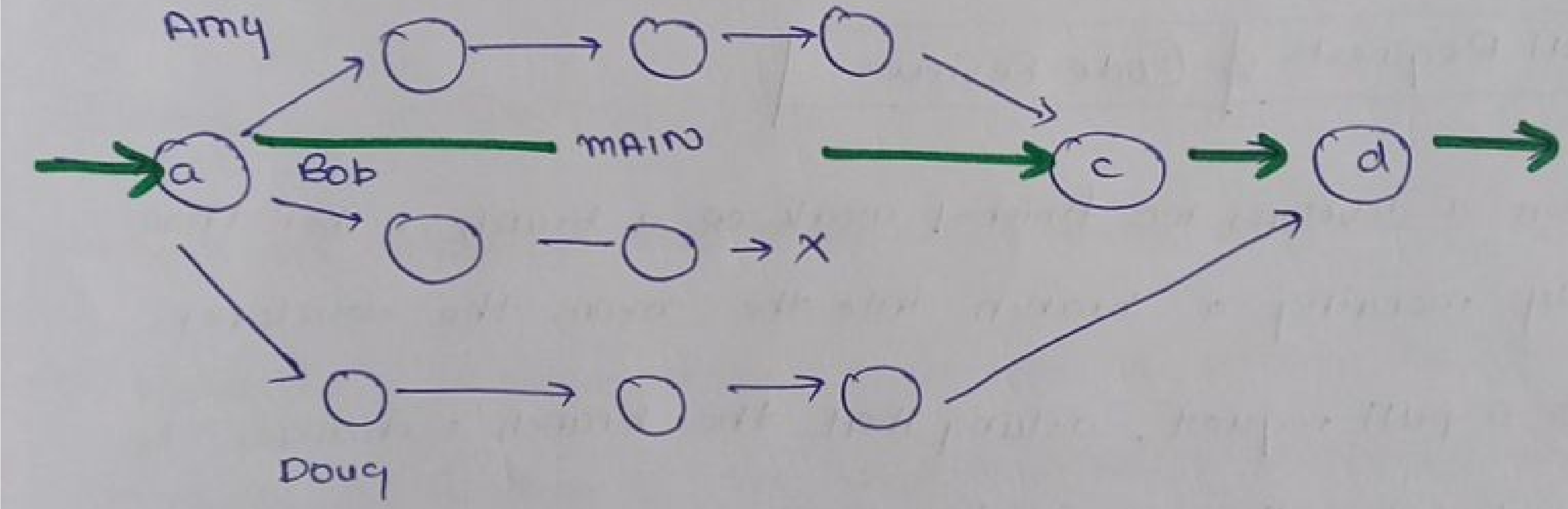
→ At any point in time, a new branch can be created that splits off from any commit of an existing branch, creating a copy.

→ As soon as a branch is created, the branch & the one from which it was split are separate - commits to one branch don't affect the other.

→ A repo branch can be merged back into another branch letter.

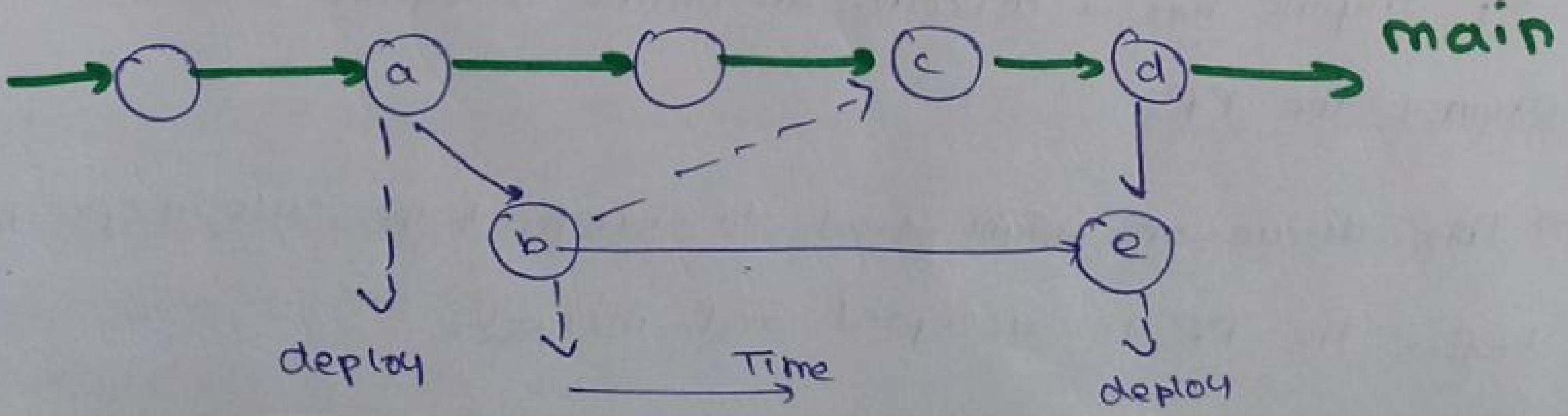
* Types of branches

A. Feature branch - allows a developer or a sub-team to make the changes necessary to implement a particular feature w/o affecting the main branch until the changes are complete and tested.



B. Release branch - used to fix branches found in a specific release

→ used when the releases are far apart enough that the main branch differs substantially (eg. Linux release versions)



* Git commands for branching?

git branch - list branches

git checkout branch - go to existing branch

git branch name - go to branch if it exists, otherwise make a new branch

git push [repo] [branch] - push changes on branch to remote repo

git pull [repo] [branch] - fetches and merges commits branch on remote repo to local repo's current branch.

B. Pull Requests & Code Reviews

→ When a developer has finished work on a branch, rather than directly merging a branch into the main, the developer makes a pull request, asking that the branch's changes be merged into the main branch.

→ All developers sharing the repo see the pull request and each has the responsibility to check how merging the changes might affect their own code.

→ If anyone has a concern, an online discussion coalesces around the PR.

→ This discussion might lead to changes to the code in question before the PR is accepted and merged.

→ The PR process is a kind of code-review, and since many PRs typically occur each day, these mini-reviews occur continuously, so there is no need for special meetings

→ A PR shouldn't be opened until a developer is confident that their code is ready. Some preparations must be made including

(i) All tests on new code shoud pass

(ii) documentation should be updated

(iii) eliminate or minimize merge conflicts.

* Rebasing vs. Merging

→ both are strategies used to eliminate merge conflicts

→ Rebasing - is an operation by means of which one tells Git to make it look as if the code had been branched off of a much later commit (change history)

→ Merging - updates one's clone of the repo ~~fro~~ with the OG repo, then merges any changes from the main branch with one's feature branch.

→ Compared to rebasing, this method is non-destructive because it doesn't rewrite history, just adds a lot of more commits to the feature branch.

→ Merging is done with 'git pull origin main'

* Continuous Integration

→ Continoous integration minimizes the time between when changes are made from a feature branch, and when those changes are merged into the main branch and deployed for customer review.

→ There are two methods that can be used for CI:

A. CI Services

→ includes services like Travis

→ CI services are responsible for running the entire test suite.

→ The rationale is that while one is continuously the code for the feature one is developing, one may not take the time to run the full test suite, which may take several minutes.

→ Most CI services can be connected directly to a GitHub repo and automatically run CI every time code is pushed to any branch in that repository?

B. Staging Server

→ A server configured as similarly to the production server as possible, but usually much smaller in scale.

→ The purpose of a staging server is to provide a safe place to deploy new featuros for customer review before they are deployed in product

→ A staging server usually has its own copy of the database containing test data

* Reporting and Fixing Bugs

→ There are 5Rs involved in reporting and fixing bugs:

1. Reporting a Bug
2. Reproducing the bug, or Reclassifying it as not a bug
3. Creating a Regression test that demonstrates the bug
4. Repairing the bug
5. Releasing the repaired code

① Reporting a Bug - Any stakeholder may find and report a bug in server-side or client-side code

② Reproducing the bug or reclassifying it as not a bug — A member of the development team must reproduce the bug, documenting the environment and the steps necessary to trigger it.

→ This process may result in reclassification as not a bug due to:

(i) not a bug ~~as~~, but rather a request for a change / enhancement in an existing feature

(ii) bug in code that is no longer supported.

(iii) bug only happens in unsupported user environment

→ Once a bug is confirmed, it is entered into a Bug Management System

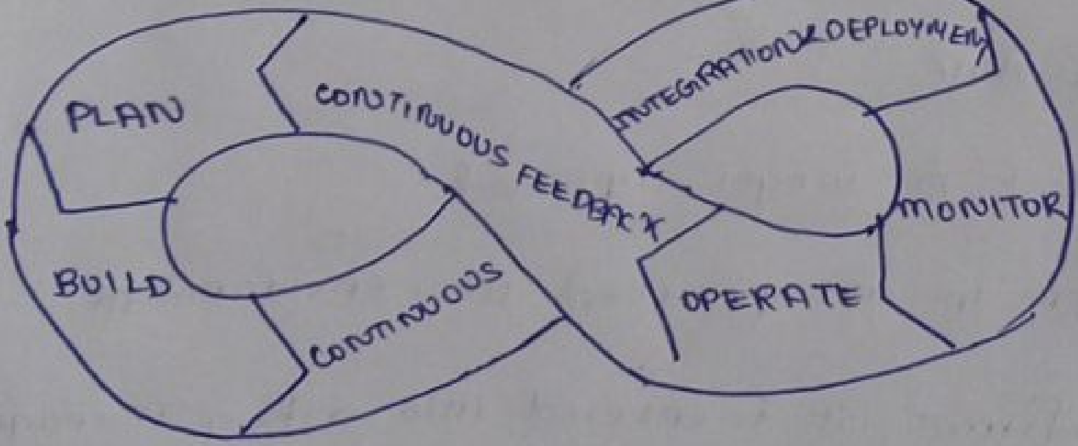
③ Creating a Regression Test - Create the simplest possible automated test that fails in the presence of the bug

④ Repairing the bug - Change code to make the regression test pass. The regression test is added to regular regression suite to ensare that the bug doesn't happen again

⑤ Releasing the repaired code - Depending on the team protocol and the bug management system that is in use, the bug may be closed out either immediately, by noting which release will contain the fix or after the release actually occurs.

* DevOps

→ DevOps = a combination of two words, development and operations. DevOps is a set of practices and tools designed to shorten the life cycle of a software development process.



* DevOps Process

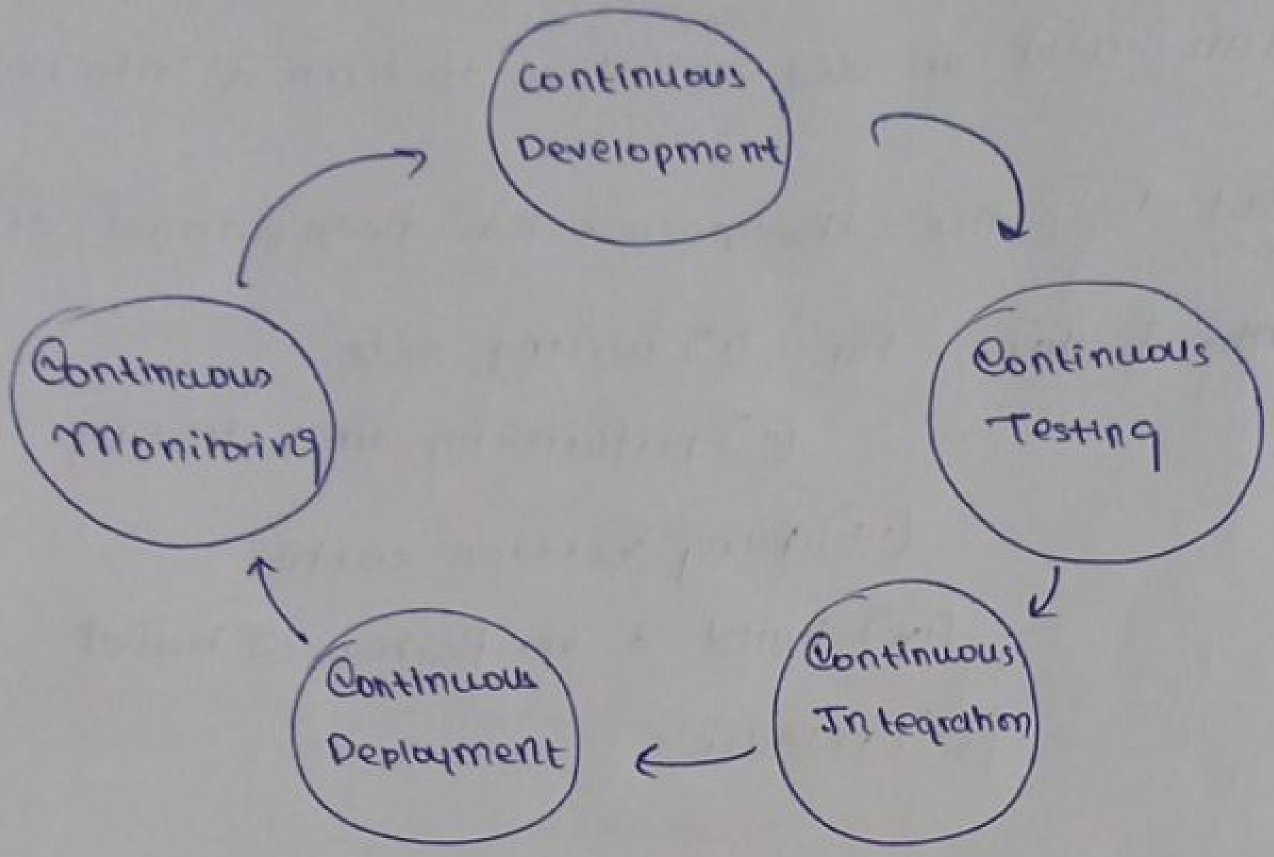
1. Plan - start with an idea and how to turn it into reality

2. Develop and test - once the project has been planned, developer turn ideas into feature by (i) writing code
   (ii) performing unit testing
   (iii) doing version control
   (iv) build & verification of build
   (v) release

3. Release - when all tests pass, the build is deployed to testing environments for each stage in the release process. This process includes : (i) automated functional environment testing
   (ii) integration testing & load testing
   (iii) moving to pre-production env.
   (iv) moving to staging env.
   (v) monitor & learn

4. Monitor - Learn and understand how users use one's applications, ~~and~~ how it reacts and quickly fix issues & bugs.
   Monitor, listen to user feedback & plan the next iteration

DevOps Architecture



* DevOp Tools

A. Source Control Management (SCM) -
- Git
- GitHub
- Subversion

B. Software Build Tools - automate the build process & an executable application -
- Maven
- Gradle
- Ant
- Grunt

C. CMT & Deployment Tools - for the deployment & operations phase :
- Jenkins
- AWS CodeDeploy
- Ansible

D. Monitoring Tools - monitor system performs and productivity
- eliminate downtime

- Nagios

E Containerization Tools - package an application with its required libraries, frameworks 2 config. files

- Docker
- Kubernetes

F. Testing Tools - tools for continuously checking for bugs

- Selenium
- TestiNG

G Integration Tools - for making CI / CD pipelines into a .exe format

- Jenkins

* Benefits of DevOps

- (i) reliability
- (ii) scalability
- (iii) security
- (iv) rapid delivery
- (v) improved collaboration
- (vi) speed

* Three - Tier Architecture

→ Software applications generally follow a three - tier architecture Logical boundaries separate these tiers - whenever one runs a development server on one's own computer or a cloud facility.

Presentation Tier

Tier-1 - consists of a HTTP server (web server)

- it accept HTTP requests from the outside world and handles static assets like image, style sheets, Javascript code.

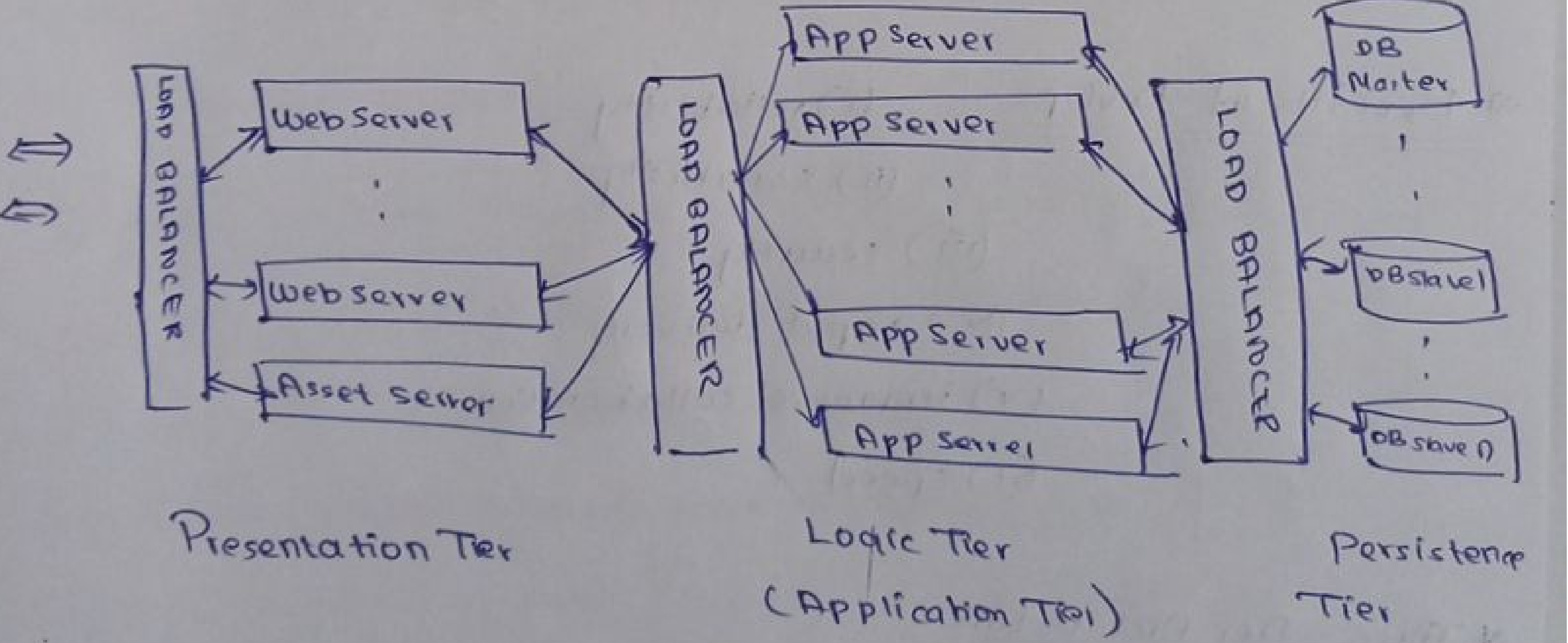
Tier-2 - Logic Tier - where the application actually runs

- receives input from Tier 1, and is supported by an application server

- That is, with Python the application server could be Django or Flask, and with Javascript it could be Node

Tier-3 - Peristence Tier, - application data must remain stored across HTTP requests such as users' login

- can be stored in databases (MySQL, PostgreSQL)



→ On a site with little content & low traffic, the software in all 3 tiers might run on a single physical computer. In production, it is common for each tier to span or more physical computers w/ specialized software.

* Responsiveness, Service Level Objectives and Apdex

→ Resposiveness is the perceived delay between when the user takes an action such as clicking a link and when the user perceives a response, such as new content appearing on the screen.

→ Responsiveness has 2 components - latency: the initial delay to start receiving new content, and throughput: the time it takes for all the content to be delivered.

→ However, the problem with these metrics is that not every user interaction takes the same amount of time - so evaluating performance requires characterizing a distribution of response times.

→ Two definitions are used to measure latency in a way that makes it impossible to ignore the bad experiences of even a small number of users:

A. Service Level Objectives (SLO) - A quantitative statement about the quantiles of the latency distribution over a time window of a given width. For eg. "95% of requests in any 5-minute window should have a latency below 100 ms"

i.e the 95th quantile should not exceed 100ms

B. APDEX Score (Application Performance Index) - an open standard that computes a simplified SLO as a number between 0 & 1 (inclusive) representing the fraction of satisfied users.

Given a user satisfaction threshold latency T

$\leq T \Rightarrow$ satisfactory

$T < x \leq 4T \Rightarrow$ tolerable

otherwise $\Rightarrow$ unsatisfactx.

The Apdex score is calculated as: $\dfrac{\text{Satisfactory} + 0.5(\text{Tolerable})}{\text{no. of samples}}$

Examples Consider a site on which 8 out of 10 requests complete in 100ms, 1 out of 10 completes in 250ms and the remaining complete in 850ms. Consider the satisfaction threshold as 200ms, find the Apdex score

Ans

$$\text{Apdex} = \frac{\text{satisfactory} + 0.5(\text{Tolerable})}{\text{no. of samples}}$$

$$= \frac{8 + (0.5)(1)}{10} = \underline{\underline{0.85}}$$

* Releases and Feature Flags

→ In Agile development, making releases a non-event requires complete automation, so that typing one command triggers all the actions to deploy the new version. It should also have the provision to abort in case something goes wrong.

→ Deployment of a non-event also requires meeting 2 challenges:

(i) Deployment testing - account for diff. in OG & production environments

- test the app in ways it should not be used, check malware handling

(ii) Feature roll-out - may require several commits especially if there are DB schema changes

There are 2 methods of implementing feature roll outs

A. Atomic Migration - Take the service offline, apply the migration for the schema change - bring the service back online. This approach is simple but causes disruptions.

★★

B. Feature Flags - a configuration variable whose value can be changed while the app is still running to control which code paths are executed. This allows at least some portion of the application to keep running. (Make 2 copies, one copy w/ a feature flag, make schema changes to the other copy while the first is still running).

* Monitoring and Finding Bottlenecks

→ It is necessary to identify which parts of an app require our attention by means of monitoring.

→ Monitoring consists of collecting app performance data for analysis and visualization. This involves monitoring Key Performance Indicators that directly impact business value.

Monitoring can be classified on three axes:

A. Active vs. Passive Monitoring:

Active - deliberately apply an external stimulus to the app to ensure it is working

Passive - no monitoring is done unless some external user asks the app to do something

B. External vs. Internal Monitoring

External - only from user's POV, like seeing response time
- a separate site might make live requests

Internal - can hook into the code & the app / app server
- Sometimes requires installing additional software
- Today, Paas, Ruby, Rails etc. is used which allows internal monitoring w/o installing external software
- Internal monitoring during development is called profiling

c. App Performance vs. User Performance

→ might want to determine what fraction of users added an item to their cart, and what fraction actually purchased them.

→ helps understand behaviour like clickstreams, think times & abandonement (measure using Google Analytics)

* → Once the monitoring is done, stress testing or longevity testing is done on a staging server to identify bottlenecks, (at what level of demands do requests become bottleneck)

* Improving Rendering and database performance with caching

→ The idea behind caching is that information that hasn't changed since the last time it was requested can simply be requrgitated rather than recomputed.

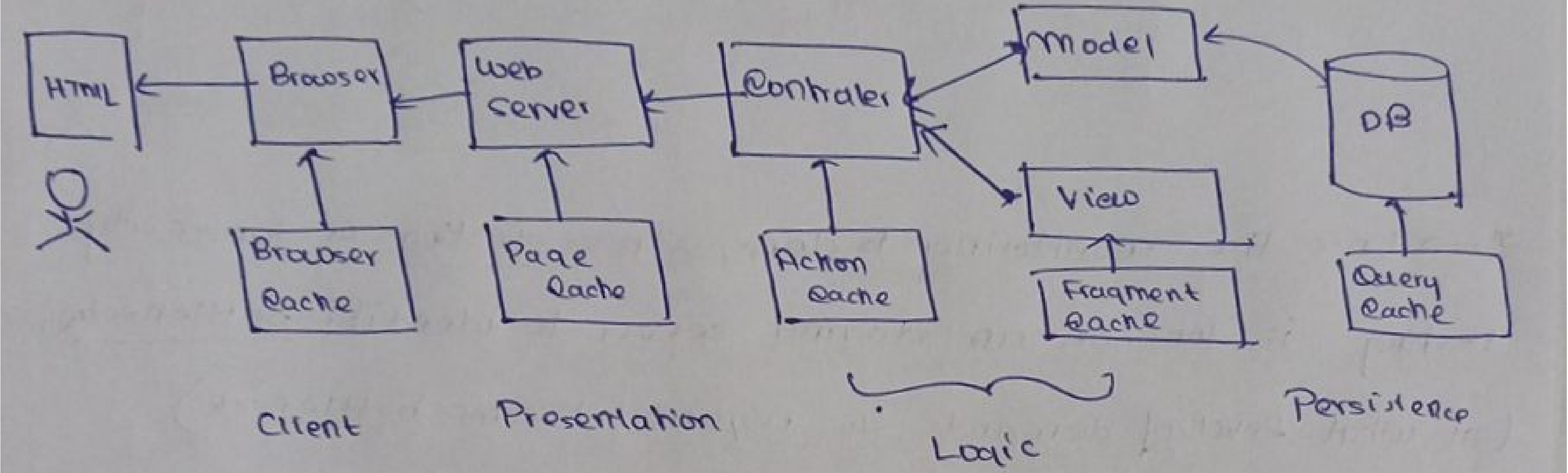
→ In the case of applications it means that we can avoid querying the database, and re-rendering HTML pages.

→ There are 2 issues that must be addressed:

A. Naming – How do you know that the result of some computation must be cached for later use, and also name it in such a way that it is used only when the exact same computation is called for?

B. Expiration - How do we detect when the cached version is out of date (stale) because the info on which it depends has changed? How is it removed from the cache?

The caching done at each level in the architecture is shown below:



There are 2 kinds of caching - page and fragment level caching

Page Level Caching → store the entire HTML output of a page, so that the same page can be used w/o regeneration

→ does not work if the page is dynamic.

Fragment Level Caching → cache certain parts / fragments of a page. For eg. If there is a sidebar that doesn't change, cache that alone. Main content can be regenerated at every request.

* Security: Defending Customer Data in Applications

* There are 3 pinciples that developers must follow while making their applications.

① **Principle of Least Privelege** - a user or a software component should be given no more privelege than what is necessary to perform its assigned task.

② **Principle of Fail Safe Defaults** - unless a user or a software component is given explicit access to an object, it should be denied access. That is, the default is denial of access.

③ **Principle of Psychological Acceptability** - the protection mechanism should not make the app harder to use than if there were no protection.

* There are 6 specific security vulnerabilities that are particularly relevant for SaaS applications:

① **Protection of Data using Encryption**

→ can encrypt all HTTP traffic by transforming it using cryptographic methods - in Transport Layer Security (TLS) and SSL (Secure Sockets Layer)

→ Running HTTP over a secure connection is HTTPS.

→ Public key cryptography like RSA can be used, where a msg. encrypted w/ a private key can only be decrypted using a public key & vice-versa. There is no way to deduce the private key from the public key, and vice versa.

→ Symmetric key cryptography (w/ just private keys can be used)

② Cross Site Request Forgery (CSRF)

→ Involves tricking the user's browser into visiting a different web site for which the user has a valid cookie, and performing an illicit action on that site as a user.

③ SQL Injection and Cross-Site Scripting

A. SQL Injection - a type of attack where an attacker injects malicious SQL code into a query. This happens when user input is not properly sanitized, and might lead to unauthorized access to the DB.

B. Cross-Site Scripting - a vulnerability that allows attacker to inject malicious scripts into web pages viewed by other viewers

④ Clickjacking

→ also called UI redress attacks

→ aimed at getting the user to take a UI action they normally wouldn't take.

→ has bait buttons on the web page

5) Prohibiting calls to private controller methods

→ Controllers may use sensitive helper functions that aren't supposed to be used / handled by end users

→ The 'protected' keyword should be used before any such non public controller action

6) Self DOS

→ A malicious denial-of-service attack seeks to keep a server busy ~~us~~ by doing useless tasks, preventing access by legitimate users.

→ One can leave oneself vulnerable to those attacks if you allow arbitrary users to perform actions to add a lot of load to the server (Instead, expensive actions should be handled by a separate background process)