DevOps HW 1

Name : Thumma Dinesh

Roll: 205224025

Branch : M.Tech Data Analytics

Q1: Read about "Planning Poker" - Agile estimation technique and illustrate an example with a Development Team of 10 who are tasked to develop a mobile app for Maha Khumb in 3 months

1. Overview of Planning Poker

• Definition: Planning Poker (also referred to as Scrum Poker) is a technique for estimation in Agile applied by development teams. Each team member makes an independent estimate (usually on a numeric scale such as the Fibonacci sequence) to a User Story or task.

• Objective: Reach agreement on effort estimates using the combined experience of the entire team, minimizing bias and anchoring effects.

2. Setting the Stage for the Example

• **Project**: Creating a mobile application for Maha-Khumb (a mass-scale event with unique requirements for crowd management, live updates, route navigation, etc.).

• Duration: 3 months (roughly 12 weeks of actual development).

• Team Structure (9 members):

1. DevOps Engineer (1 member)
2. Product Owner (1 member)
3. Scrum Master / Agile Coach (1 member)
4. Developers (4 members)
5. UX/UI Designer (1 member)
6. QA / Tester (1 member)

(Though the total is 9, usually only developers + QA + UX participate in Planning Poker for estimates. The Product Owner comes to clear requirements, and the Scrum Master coordinates.)

3. Process Outline

1. Prepare User Stories

The Product Owner provides a backlog of user stories for the mobile application. Sample user stories could be:

- As a visitor, I want to see real-time crowd density maps so that I can avoid congested areas.
- As a participant, I'd like to sign up for events through the app so that I can organize my schedule.

2. Introduce the Stories

The team goes through the acceptance criteria and clarifications:

- What data sources are required for real-time crowd density?
- Are there external APIs to be added for location services?

3. Distribute Estimation Cards

Mostly, the Fibonacci series is utilized in estimates (for example, 1, 2, 3, 5, 8, 13, 20, 40). All team members have a "deck" of cards.

4. Estimation Round

- Step 1: Scrum Master reads the user story aloud.
- Step 2: Team members independently come up with an estimate in terms of factors like complexity, risk, and level of effort.
- Step 3: All simultaneously show the selected card (e.g., 5, 8, 13, etc.).

5. Discuss Differences

- If the majority of estimates are grouped around one value but one member gives a much higher or lower value, the team discusses why.

For instance, a QA engineer may envision possible complexities in cross-platform testing those others did not. After discussion, the team re-estimates if necessary.

6. Settle on a Consensus

- After consensus, note the final story point estimate for each story.
- The Product Owner then prioritizes the stories both on business value and estimated effort.

Example of an Estimation Session

Suppose one of the key stories is:

"As a user, I want to get a push notification with safety alerts (e.g., over-crowding, route diversions) so I can be informed in real time."

- Developer A estimates 8 story points
- Developer B estimates 5 story points
- Developer C estimates 13 story points
- QA estimates 8 story points
- UX estimates 5 story points

Once cards are revealed:

- Developer C states push notifications would perhaps involve intricate integration with a third-party messaging service, particularly if tens of thousands of users potentially could be using the app at one time.
- UX/Developers B think it's easy if they just use a generic push notification provider.
- The team elects to include extra testing on high load.

They reallot and vote on 8 story points following a discussion clarifying the process to deal with concurrency and load.

5. Utilizing the Estimates for the 3-Month Schedule

- The team sums these up into a sprint plan.
- Every two-week sprint aims for a specific quantity of story points. If historically the team's velocity is near 20 story points per sprint, they can estimate how many sprints (and hence how many stories) could be completed in 3 months.
- This creates a rough blueprint: e.g., 5 sprints × 20 points = 100 total story points the team could realistically complete.

Primary Advantage: Planning Poker elicits everyone's opinions, encourages debate about hidden complexity, and generates more accurate estimates than one person's estimate

Q2: Read Paper – Measuring Software Development Waste in OSS Projects - https://arxiv.org/pdf/2409.19107. Pick one measure from this paper and apply it on any open-source repository. Share results.

1. Overview of the Concept of the Paper

The paper, "Measuring Software Development Waste in OSS Projects," suggests quantifying types of "waste" in software development. Some potential examples of "waste" might include:

- Idle Issues: Issues that stay open or unresolved for an extended duration.
- Abandoned Pull Requests: Suggested changes that stay unreviewed or unmerged.
- Rework / Code Churn: Ongoing rewriting of the same code, or high churn in a particular module showing inefficiencies.

2. Selecting One Measure

We will select "Issue Stagnation Rate" as a sample measure. This measure could be defined as:

Issue Stagnation Rate = (Number of issues remaining open after a threshold period) / (Total issues opened during the same period).

Why is this a good measure of "waste"?

- Inert issues also signal blocked or abandoned parts of the project.
- They will build up and swamp the backlog, leading to confusion and management overhead.

3. Bringing This to an Open-Source Repository

Take the "httpie" project (CLI tool for HTTP access) on GitHub (github.com/httpie/httpie). You may select any active, well-known open-source repository you wish.

1. Set a threshold for inactivity

- Consider an issue "stagnant" if it is over 60 days old with no discernible progress (i.e., no new commits or comments on the issue).

2. Collect data

- Utilize the GitHub API or the repository's Insights/Issues page.
- Filter issues opened within a time frame (e.g., the last 6 months).
- Then determine how many of those have been open for 60+ days with no activity.

3. Compute

- If in the past 6 months, there were 120 new issues. Of which 30 are still open over 60 days with no new comments/commits.

4. Compute Issue Stagnation Rate

Issue stagnation rate = (30/120) * 100% = 25%

5. Interpretation

- A 25% stagnation rate may indicate a quarter of the issues remain unaddressed for long times.
- It may indicate the maintainers are overwhelmed, or the issues are too vaguely worded, or triage is not occurring rapidly.
- In a culture of Agile/DevOps, regular backlog grooming and enhanced triage mechanisms could decrease this number.

6. Solutions

- Rotate the maintainers an "Issue Triage".
- Automark stale issues with a bot and request community or maintainer action.
- Archive/close stale issues.

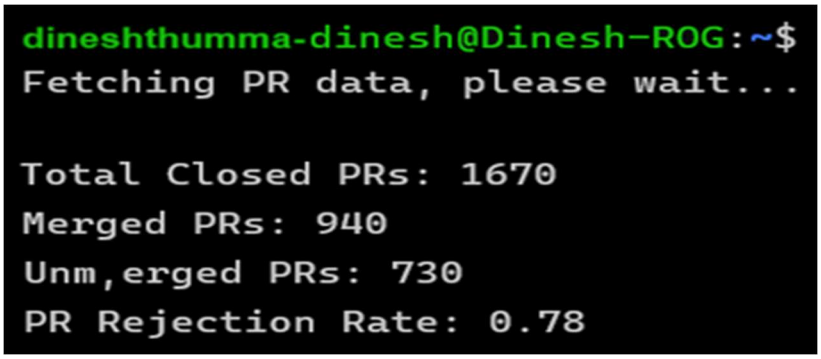
4. Sharing the Results

Example result: "We noticed in the httpie repository over the past 6 months that Issue Stagnation Rate was approximately 25%. This means that 1 out of every 4 problems might not be getting timely attention. Based on these results, we suggest a routine triage procedure and issue template clarity to minimize the stagnation rate."

Example 2:

2. Repository Analysed:

- Repository: axios/axios on GitHub (https://github.com/axios/axios).
- Method: Used GitHub API and Python to fetch PR data.



A PRR of **0.78** suggests that for every merged PR, approximately **0.78 PRs** were closed without merging. This indicates a **moderate level of waste** in terms of unmerged contributions. The PR Rejection Rate of 0.78 for the axios repository highlights a need for improved backlog management to reduce unused contributions.