HOME-WORK 1

Effort Estimation and Software Waste

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.

Answer:-

1. Overview of Planning Poker

- **Definition**: Planning Poker (also called Scrum Poker) is an Agile estimation technique used by development teams. Each team member independently assigns an estimate (often using a numeric scale like the Fibonacci sequence) to a User Story or task.
- **Objective**: Achieve consensus on effort estimates by leveraging the collective experience of the whole team, while mitigating bias and anchoring effects.

2. Setting the Stage for the Example

- **Project**: Developing a mobile app for Maha-Khumb (a large-scale event with special needs for crowd management, live updates, route navigation, etc.).
- **Duration**: 3 months (approximately 12 weeks of active development).
- **Team Composition** (10 people):
    1. **Product Owner** (1 person)
    2. **Scrum Master / Agile Coach** (1 person)
    3. **Developers** (5 people)
    4. **UX/UI Designer** (1 person)
    5. **QA / Tester** (1 person)
    6. **DevOps Engineer** (1 person)

(Although the total is 10, typically only developers + QA + UX join Planning Poker for estimates. The Product Owner attends to clarify requirements, and the Scrum Master facilitates.)

3. Process Outline

1. **Prepare User Stories**
   The Product Owner presents a backlog of user stories for the mobile app. Example user stories might include:
    - *“As a visitor, I want to see real-time crowd density maps so that I can avoid congested areas.”*
    - *“As a participant, I want to register for events via the app so that I can manage my schedule.”*
2. **Introduce the Stories**
   The team discusses the acceptance criteria and clarifications:
    - Which data sources are needed for real-time crowd density?

    - Are there any external APIs to be integrated for location services?

3. **Distribute Estimation Cards**
   Typically, the Fibonacci sequence is used for estimates (e.g., 1, 2, 3, 5, 8, 13, 20, 40). Each team member has a "deck" of cards.

4. **Estimation Round**
    - *Step 1*: The Scrum Master reads aloud the user story.
    - *Step 2*: Team members privately decide on an estimate based on factors such as complexity, risk, and amount of effort.
    - *Step 3*: Everyone simultaneously reveals their chosen card (e.g., 5, 8, 13, etc.).

5. **Discuss Differences**
    - If most estimates cluster around a single value but one member assigns a significantly higher or lower value, the team discusses the reasoning.
    - For example, a QA engineer might see potential complexities in cross-platform testing that others missed.
    - After discussion, the team re-estimates if needed.

6. **Settle on a Consensus**
    - Once consensus is reached, record the final story point estimate for each story.
    - The Product Owner can then prioritize the stories based on both business value and estimated effort.

4. Example of an Estimation Session

Let's assume one of the crucial stories is:

"As a user, I want to receive a push notification with safety alerts (e.g., over-crowding, route diversions) so I can stay 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

After revealing cards:

- Developer C explains that push notifications might require complex integration with a third-party messaging service, especially if tens of thousands of people could be using the app simultaneously.
- UX/Developers B feel it's straightforward if they use a standard push notification provider.
- The team decides to factor in additional testing under high load.

They revote and settle on **8** story points after clarifying the approach to handle concurrency and load.

5. Using the Estimates for the 3-Month Timeline

- The team aggregates these estimates into a sprint plan.

- Each two-week sprint targets a certain number of story points. If the team velocity is historically around 20 story points per sprint, they can forecast how many sprints (and thus how many stories) might fit into 3 months.
- This forms a rough roadmap: e.g., 5 sprints × 20 points = 100 total story points the team might realistically complete.

Key Benefit: Planning Poker gets everyone's input, fosters discussion on hidden complexities, and produces more reliable estimates than a single person's guess.

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.

Answer:-

1. Overview of the Paper's Concept

The referenced paper, "Measuring Software Development Waste in OSS Projects," proposes methods for quantifying forms of "waste" that occur during software development. Examples of "waste" could include:

- **Idle Issues**: Issues that remain open or unaddressed for extended periods.
- **Abandoned Pull Requests**: Proposed changes that remain unreviewed or unmerged.
- **Rework / Code Churn**: Frequent rewrite of the same code, or high churn in a particular module indicating inefficiencies.

2. Choosing One Measure

Let's pick **"Issue Stagnation Rate"** as an example measure. This measure might be defined as:

Issue Stagnation Rate = (Number of issues that remain open beyond a threshold period) / (Total number of issues created over the same period).

Why is this a good indicator of "waste"?

- Stagnant issues often indicate blocked or neglected areas of the project.
- They can accumulate and clutter the backlog, increasing confusion and administrative overhead.

3. Applying This to an Open-Source Repository

Suppose we choose the **"httpie"** project (an HTTP client CLI tool) on GitHub (github.com/httpie/httpie). You can pick any active, popular open-source repository of your choice.

1. **Define a threshold for stagnation**
   - Let's say we consider an issue "stagnant" if it has been open for more than **60 days** without significant updates (i.e., no new comments or commits referencing the issue).
2. **Gather data**
   - Use the GitHub API or the repository's Insights/Issues tab.
   - Filter for issues created in a given time window (e.g., the past 6 months).
   - Then count how many of those have been open for 60+ days with no recent activity.
3. **Calculate**
   - Suppose in the last 6 months, there were **120 new issues**.
   - Out of these, **30** remain open longer than 60 days with no new comments/commits.
4. **Compute Issue Stagnation Rate**

$$\text{Issue Stagnation Rate} = \frac{30}{120} \times 100\% = 25\%$$

5. **Interpretation**
    - A 25% stagnation rate might suggest a quarter of the issues linger unaddressed for extended periods.
    - This could mean the maintainers are overloaded, or the issues are too vaguely described, or triage is not happening quickly.
    - In an Agile/DevOps context, continuous backlog grooming and improved triage processes may help reduce this number.
6. **Potential Remedies**
    - Introduce an "Issue Triage" rotation among maintainers.
    - Label stale issues automatically with a bot and prompt the community or maintainers to act.
    - Close or archive issues that are no longer relevant.

4. Sharing the Results

Sample result: "We found that in the httpie repository, the Issue Stagnation Rate over the last 6 months was around 25%. This indicates that 1 in 4 issues may not be receiving timely attention. Based on these findings, we recommend a regular triage process and clarity in issue templates to reduce the stagnation rate."