CA6C1 – DevOps - National Institute of Technology, Trichy

HOME-WORK 1

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.

Planning Poker: Agile Estimation Technique

Planning Poker is a technique for estimation in Agile employed by development teams to gauge the effort to finish tasks within a project. Planning Poker facilitates collaboration among the team, minimizes biases, and provides more accurate and consensual estimation. The method is employed extensively in Scrum and other Agile methods for estimating story points, ideal days, or any other unit of work.

Poker Estimation working:

1. **Setup:** All participants are handed a deck of cards with predetermined values, normally an altered Fibonacci sequence (0, 1, 2, 3, 5, 8, 13, 20, 40, 100), where higher values represent more complexity and effort.

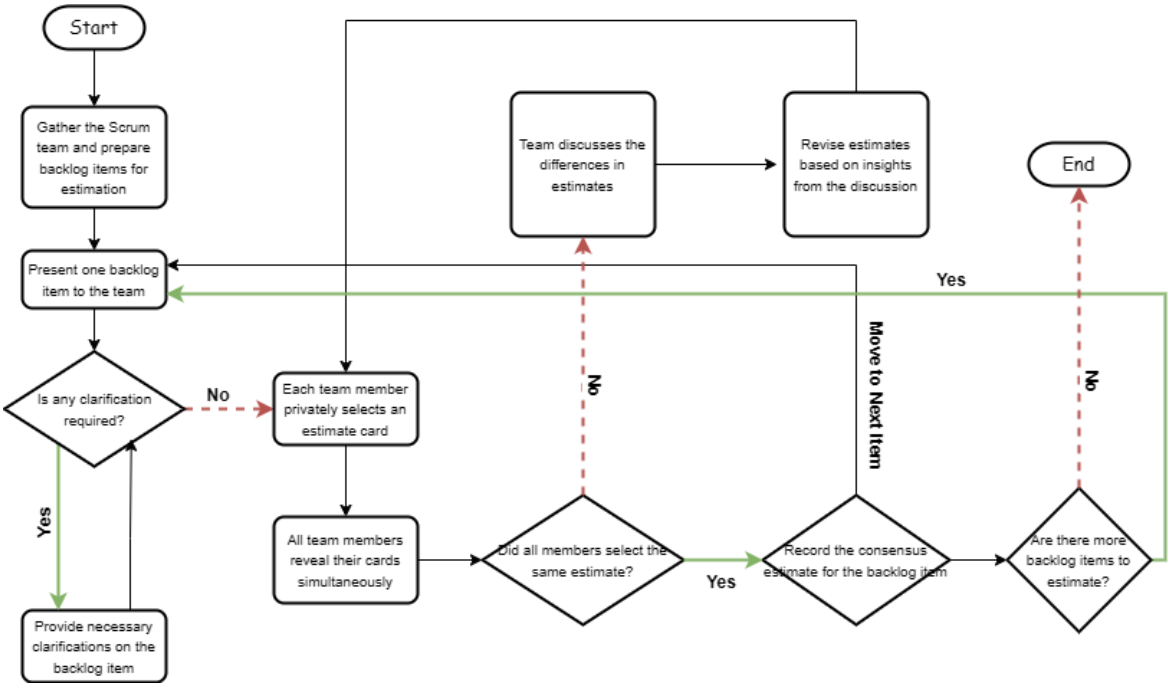
2. **Story Presentation:** The Scrum Master or Product Owner reads out a user story, including its scope, requirements, and acceptance criteria.

3. **Individual Estimation:** Each member of the team secretly picks a card that corresponds to their estimate of effort.

4. **Simultaneous Reveal:** All team members expose their cards simultaneously to prevent anchoring bias.

5. **Discussion:** If the estimates are far apart, the team discusses why they made the choices they did, taking into account technical complexity, dependencies, risks, and uncertainties.

6. **Consensus:** The cycle is repeated until the team comes to a consensus on a reasonable estimate.



Development of a Mobile App for Maha-Khumb which include Poker Estimation

Overview

A 10-member development team is assigned the job of designing a feature-laden mobile application for Maha-Khumb within three months. Since there is a very tight timeline, the team practices Agile methodology with two-weekly sprints so that consistent progress is maintained. The app will feature functionalities including:

- **Event Schedules:** A detailed calendar of all Maha-Khumb events.
- **GPS-enabled Navigation:** Real-time location tracking along with event venue maps.
- **Emergency Alerts:** Immediate notifications for crowd management and safety.
- **Live Updates:** Social media and news feed integration.
- **Multilingual Support:** Multilingual support for a wide user base.

Planning Poker in Action

Step 1: Breaking Down User Stories

The project is broken down by the development team into several user stories, which are small enough to be completed within a sprint cycle. These include:

1. *As a user, I want to view an event schedule in detail so that I can organize my visit.*
2. *As an attendee, I want to have a GPS map so that I can find my way around the event with ease.*
3. *As an organizer, I would like to send emergency notifications to all users in the event of an emergency.*
4. As a user, I would like to have multilingual support so that I can use the app in my native language.

5. As an admin, I would like to effectively manage live updates so that the users remain updated.

Step 2: Planning Poker Effort Estimation

- The Scrum Master leads the Planning Poker exercise.
- For the user story "View event schedule," the team members secretly pick their effort estimation cards.
- Cards played: 3, 5, 5, 3, 8, 5, 5, 3, 5, 3.
- Discussion includes:
    - One developer who picked 8 discusses issues with combining real-time updates.
    - After discussion, the team comes to an agreement on 5 story points.
    - The team repeats the process for the user story "GPS-enabled navigation."
    - Cards uncovered: 8, 8, 13, 5, 8, 8, 5, 8, 8, 8.
- Discussion:
    - The challenge of real-time tracking and integration of maps is talked about.
    - There is an agreement on 8 story points.

Step 3: Sprint Planning & Execution

- According to estimates, the team allocates user stories and schedules tasks for every sprint.
- With a 3-month timeline, the project is broken down into six 2-week sprints such that development, testing, and deployment can fit within the deadline.
- Tasks are allocated to members based on skills, thus preventing workload overload.
- There are continuous tests and feedback loops in place to enable early identification of issues.

Technical Considerations

- **Tech Stack:** React Native for cross-platform development, Firebase for real-time database and push notifications, Google Maps API for location services.
- **CI/CD Pipeline:** Jenkins or GitHub Actions for continuous integration and deployment.
- **Database Selection:** PostgreSQL for structured data, Redis for caching real-time updates.
- **Security Measures:** OAuth for authentication, data encryption for sensitive user information.

Advantages of Planning Poker

- Enhanced Accuracy: Promotes discussion and sharing of knowledge to make better estimates.
- Consensus-Based: Avoids one-way decision-making and fosters team consensus.
- Participation & Collaboration: Includes the entire team, enhancing commitment and morale.
- Early Risk Exposure: Identifies technical issues and possible roadblocks.

Conclusion

Planning Poker allows for a well-structured but agile estimation approach, enhancing precision and team collaboration in Agile teams. Using this method, the development team is able to provide realistic estimates and enhanced project execution. Through an organized Agile method, the team of 10 developers will be able to deliver a secure and user-friendly Maha-Khumb mobile application within the 3-month period.

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.

We can compute the Backlog Inversion Index (BII) for an open-source GitHub repository with the help of the GitHub API. This is the step-by-step procedure:

Step 1: Define the Approach

1. Retrieve Issues from GitHub – Retrieve issue information from an open-source repository (pallets/flask).
2. Assign Priorities – Label issues according to tags (for example, "bug" as high, "enhancement" as medium, "documentation" as low).
3. Examine Resolution Order – Identify instances in which lower-priority problems were solved prior to higher-priority ones.

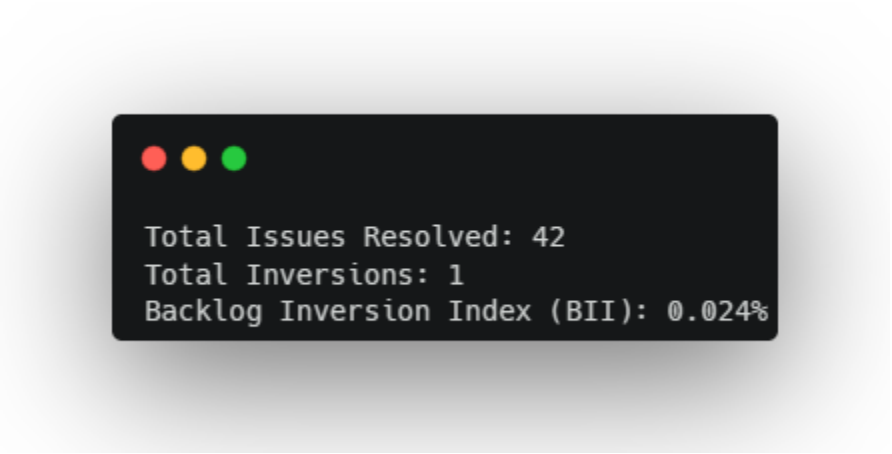
$$BII = \left(\frac{\text{Number of Inversions}}{\text{Total Number of Issues Resolved}}\right) \times 100$$

Step 2: Fetch Data & Analyze

Below is the Python script to fetch issue data and compute BII.

```python
import requests
import pandas as pd

GITHUB_REPO = "pallets/flask"
GITHUB_TOKEN = "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"

headers = {"Authorization": f"token {GITHUB_TOKEN}"}

def fetch_issues(repo, state="closed", per_page=100):
    url = f"https://api.github.com/repos/{repo}/issues?state={state}&per_page=
{per_page}"
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.json()
    else:
        print("Error fetching issues:", response.status_code)
        return []


priority_map = {"bug": 3, "enhancement": 2, "documentation": 1}  # High to Low

issues = fetch_issues(GITHUB_REPO)
processed_issues = []

for issue in issues:
    if "pull_request" in issue:
        continue

    labels = [label["name"] for label in issue.get("labels", [])]
    priority = max([priority_map.get(label, 0) for label in labels], default=0)

    processed_issues.append({
        "id": issue["id"],
        "title": issue["title"],
        "priority": priority,
        "created_at": issue["created_at"],
        "closed_at": issue["closed_at"],
    })

df = pd.DataFrame(processed_issues)
df = df.sort_values(by="closed_at")

inversions = 0
for i in range(len(df)):
    for j in range(i+1, len(df)):
        if df.iloc[i]["priority"] > df.iloc[j]["priority"]:
            inversions += 1

total_issues_resolved = len(df)
bii = (inversions / total_issues_resolved) * 100 if total_issues_resolved > 0 else 0

print(f"Total Issues Resolved: {total_issues_resolved}")
print(f"Total Inversions: {inversions}")
print(f"Backlog Inversion Index (BII): {bii:.2f}%")
```

Output :



- A high BII suggests inefficiencies in issue prioritization.
- A low BII indicates that higher-priority tasks are resolved first.