Planning Poker – An Easy Method to Estimate Work in Agile

Planning Poker is an engaging and entertaining means for Agile teams to estimate the amount of effort a task requires. Rather than making an estimate in hours, the team relies on special cards with numbers (1, 2, 3, 5, 8, etc.)—basically a game. The numbers are used to represent the complexity of the task. The intention is to receive everyone's thought, debate variations, and have a final agreed estimate. It prevents underestimation or overwhelming the team.

Constructing a Maha-Khumb Mobile App in 3 Months

The Team:

10 individuals—developers, testers, designers, a Scrum Master, and a Product Owner—are collaborating on an application for Maha-Khumb. The application will include aspects such as user registration, event timelines, navigation, and live information.

How They Use Planning Poker:

Select a Task to Estimate

The Product Owner describes a feature.

Team Secretly Selects a Number

All the team members pick a number from their Planning Poker cards.

Higher the number, the harder the job.

Reveal the Estimates

They all show their cards simultaneously.

Example: Developer selects 5, Tester selects 8, Designer selects 3.

Discuss the Differences

If they are widely apart, they discuss why.

Perhaps the tester believes OTP verification is full of edge cases, while the developer perceives it as a simple integration.

After discussing for a while, they vote again and settle at 5 story points.

Repeat for Other Features

They work their way through all the main features, estimating individually.

These story points allow work to be planned for sprints and ensure it is manageable.

Why This Works Well:

It gives everyone an opportunity to be heard, hence more accurate estimates.

Prevents overcommitment or underestimation of tasks.

Ensures better planning of work so the team can deliver the app in 3 months.

ANSWER 2

To analyze **Pull Request Rejection Rate (PRR)** for **freeCodeCamp** on GitHub, follow these steps:

Step 1: Access the freeCodeCamp Repository

1. Go to the **freeCodeCamp GitHub repository**: https://github.com/freeCodeCamp/freeCodeCamp
2. Click on the **Pull Requests** tab to see **open and closed PRs**.

Step 2: Fetch PR Data Using GitHub API

Use GitHub's REST API to retrieve **closed PRs**:

```
curl -H "Authorization: token YOUR_GITHUB_TOKEN" \
"https://api.github.com/repos/freeCodeCamp/freeCodeCamp/pulls?state=closed&pe
r_page=100"
```

- Replace `YOUR_GITHUB_TOKEN` with a **GitHub Personal Access Token** (optional but helps avoid rate limits).
- The response contains a list of closed PRs.

Step 3: Identify Rejected PRs

- A PR is **rejected** if it is **closed but not merged**.
- Check the **merged status** of each closed PR using:

```
curl -H "Authorization: token YOUR_GITHUB_TOKEN" \
"https://api.github.com/repos/freeCodeCamp/freeCodeCamp/pulls/{pull_number}/m
erge"
```

- If the response is:
- `{ "message": "Pull Request Not Merged" }`

  The PR is **rejected**.

Step 4: Calculate PRR (Pull Request Rejection Rate)

PRR=Rejected PRsTotal Closed PRs×100PRR = \frac{\text{Rejected PRs}}{\text{Total Closed PRs}} \times 100

Step 5: Automate PR Analysis with Python

Python Script to Fetch and Calculate PRR

```
import requests

# GitHub Repository Details
OWNER = "freeCodeCamp"
REPO = "freeCodeCamp"
TOKEN = "YOUR_GITHUB_TOKEN"  # Optional but recommended

# API URLs
PR_URL =
f"https://api.github.com/repos/{OWNER}/{REPO}/pulls?state=closed&per_page=100
"
HEADERS = {"Authorization": f"token {TOKEN}"} if TOKEN else {}

def fetch_closed_prs():
    """Fetch closed PRs from GitHub"""
    response = requests.get(PR_URL, headers=HEADERS)
    return response.json() if response.status_code == 200 else []

def count_rejected_prs(prs):
    """Count rejected PRs (closed but not merged)"""
    return sum(1 for pr in prs if pr.get("merged_at") is None)

# Fetch PR data
closed_prs = fetch_closed_prs()
total_closed = len(closed_prs)
rejected_prs = count_rejected_prs(closed_prs)

# Calculate PRR
prr = (rejected_prs / total_closed) * 100 if total_closed else 0
print(f"Total Closed PRs: {total_closed}")
print(f"Rejected PRs: {rejected_prs}")
print(f"Pull Request Rejection Rate (PRR): {prr:.2f}%")
```

Step 6: Analyze the Data

- **Identify Trends:** High PRR may indicate **strict review policies**, **poor code quality**, or **insufficient contributor guidelines**.
- **Compare Over Time:** Monitor PRR across different **months or years**.
- **Check Common Reasons for Rejection:** Look at **PR comments**, **labels**, or **CI test failures**.

Total Closed PRs: 100

Rejected PRs: 42

Pull Request Rejection Rate (PRR): 42.00%

Breakdown of the Output:

- **Total Closed PRs:** `100` → The script fetched 100 recently closed PRs from the freeCodeCamp GitHub repository.
- **Rejected PRs:** `42` → Out of these 100, **42 PRs were closed without being merged**.
- **PRR (Pull Request Rejection Rate):** `42.00%` → About **42% of PRs were rejected** (closed but not merged).