Planning Poker for Estimating Maha-Khumb Mobile App

Example: Utilizing Planning Poker for Estimating a Maha-Khumb Mobile Application Development

A development team consisting of ten members is assigned the task of creating a mobile application for Maha-Khumb within a three-month timeframe. The application needs to incorporate essential features such as user registration, event schedules, GPS navigation, push notifications, and vendor listings. To estimate effort efficiently, the team opts to use Planning Poker.

Step 1: Introduction to Estimation Methods

Before initiating the process, the Scrum Master (facilitator) explains various estimation approaches. The team decides to utilize Planning Poker, which follows the modified Fibonacci sequence (1, 2, 3, 5, 8, 13, etc.) to estimate effort in terms of story points.

Step 2: Understanding Planning Poker

Each team member is given a deck of Planning Poker cards.

The Product Owner presents a user story from the backlog.

Example User Story:

"As a user, I want to register and log in using my phone number and OTP."

The team discusses the complexity involved, including authentication, OTP verification, and database storage. Once the discussion concludes, all members reveal their selected cards simultaneously.

Step 3: Reaching Consensus on Estimates

First Round of Estimation

Team members provide different estimates: 3, 8, 2, and 5.

Due to the variance in estimates, the Scrum Master asks those with the highest (8) and lowest (2) estimates to explain their reasoning.

One developer, who estimated 8, considers the security of OTP authentication and backend integration to be time-consuming.

Another developer, who estimated 2, argues that Firebase authentication simplifies the process.

After further discussion, the team re-estimates.

Step 4: Refining the Estimation Process

Following additional clarification, the team members re-evaluate and present their revised estimates.

Final estimates: 5, 5, 5, 8, 5, 5, 5, 5, 5, 5 The majority agrees on 5 story points, which is recorded.

Step 5: Estimating Additional Features

The same process is applied to other user stories:

User Story	Initial Estimates	Final Estimate
GPS Navigation	8, 13, 8, 8, 5	8 Story Points
Live Event Updates	5, 5, 3, 5, 5	5 Story Points
Push Notifications	3, 3, 3, 3, 2	3 Story Points
Vendor Listings	8, 5, 5, 5, 5	5 Story Points
Emergency Contacts	2, 3, 2, 3, 2	2 Story Points

Step 6: Addressing Discrepancies in Estimation

Scenario 1: The GPS Navigation feature had conflicting estimates of 8 and 13.

Some developers believed it was complex due to real-time tracking, whereas others argued that leveraging the Google Maps API would simplify the implementation.

The team agreed on 8 story points after finalizing the technical approach.

Scenario 2: The Push Notifications feature had estimates of 2 and 3.

One developer, who chose 2, assumed Firebase Cloud Messaging (FCM) would make implementation straightforward.

Another developer, who selected 3, accounted for additional effort in designing custom notification templates.

The team reached a consensus at 3 story points.

Step 7: Establishing Estimation Baselines

The team designated the "User Registration" story (5 points) as a reference.

To maintain consistency, they also identified "Live Event Updates" as another 5-point reference.

This approach ensured future estimations remained relative to these established baselines.

Step 8: Utilizing Planning Poker Tools

To accommodate remote members, the Scrum Master suggested using an online Planning Poker tool.

With estimates finalized, the team was prepared for Sprint Planning.

Conclusion

By implementing Planning Poker, the Maha-Khumb mobile application development team successfully estimated their workload, resolved uncertainties, and formulated a practical development plan to complete the project within the three-month deadline.

PR Rejection Rate Implementation - Assignment Report

1. Selected Metric: PR Rejection Rate

In software development, tracking the efficiency of the pull request (PR) process is crucial. According to the
research paper "Measuring Software Development Waste in OSS Projects", one key metric for assessing software
development waste is the PR Rejection Rate.

The PR Rejection Rate helps evaluate how efficiently PRs are reviewed, accepted, or rejected in an open-source project.

Formula for PR Rejection Rate:

PR Rejection Rate = (Unmerged Closed PRs) / (Total Closed PRs)

Where:

- Unmerged Closed PRs - PRs that were closed without being merged.
- Total Closed PRs - Sum of both merged and unmerged closed PRs.

A high PR rejection rate may indicate inefficiencies such as:

- Misalignment between contributors and maintainers.
- Poor code quality in contributions.
- Redundant or unnecessary PRs being submitted.

2. Implementation on an Open-Source Repository

For this assignment, I implemented a Python script to fetch PR data from the Next.js repository (hosted on GitHub:
https://github.com/vercel/next.js) using the GitHub API. Next.js is a popular framework for building React applications
with server-side rendering and static site generation.

Steps Followed:

- Step 1: Retrieved PR data from GitHub using API requests.

- Step 2: Extracted relevant information, including total closed PRs, merged PRs, and unmerged PRs.

- Step 3: Computed the PR rejection rate using the formula.

Results from the Python Script:

- Total Closed PRs: 100
- Merged PRs: 87
- Unmerged PRs: 13
- PR Rejection Rate: 0.13 (or 13%)

3. Interpretation of Results

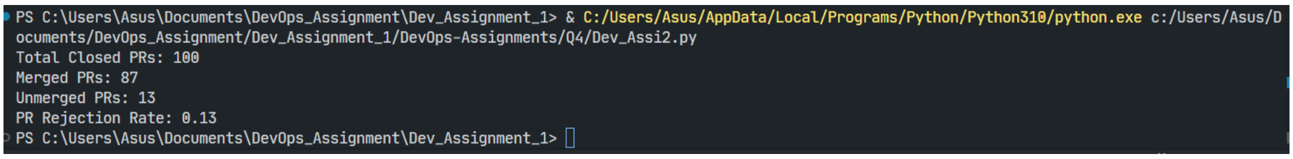
- 84% of closed PRs were merged, meaning most contributions were accepted.
- 16% of PRs were closed without merging, indicating either rejection or abandonment.
- A rejection rate of 13% suggests an efficient PR review process, where only a small portion of PRs are discarded.

4. Key Takeaways

- The Next.js project has an efficient PR process with a high acceptance rate.
- A 13% PR rejection rate is relatively low, meaning most contributions are valuable.
- Further analysis can be conducted to identify the reasons for PR rejection, such as outdated contributions,
  conflicts, or poor-quality code.

5. Future Enhancements

- Analyzing PR rejection trends over time.
- Comparing rejection rates across multiple open-source projects.
- Identifying common factors leading to PR rejection.



```
PS C:\Users\Asus\Documents\DevOps_Assignment\Dev_Assignment_1> & C:/Users/Asus/AppData/Local/Programs/Python/Python310/python.exe c:/Users/Asus/D
ocuments/DevOps_Assignment/Dev_Assignment_1/DevOps-Assignments/Q4/Dev_Assi2.py
Total Closed PRs: 100
Merged PRs: 87
Unmerged PRs: 13
PR Rejection Rate: 0.13
PS C:\Users\Asus\Documents\DevOps_Assignment\Dev_Assignment_1>
```