# DAO Simulation: Governance & Operations

Understanding Stakeholders, Decision Flow, and Voting

Yohannes Taye

# Open-source software DAO

- The OSS-Gov DAO is a prototype demonstrating a Decentralized Autonomous Organization specifically designed for governing open-source software projects. The project implements a governance model that includes various stakeholders, a defined decision flow, and Quadratic Voting as the primary voting mechanism.

# DAO Overview

- A system managing users, proposals, and a voting process.
- Features a treasury for funding approved initiatives.
- Employs a Quadratic Voting (QV) mechanism to balance influence.
- Cycle: Users - Proposals - Voting - Execution - Treasury Impact

# Who is Involved? The Stakeholders

- Users: The core participants in the DAO.
- Each user possesses a unique ID and voice_credits.
- voice_credits represent a user's potential influence or stake in the DAO.
- User Roles (from user.py): CommunityMember, Contributor, Maintainer.
- Proposers: Any registered user can submit proposals.

# From Idea to Action: The Proposal Lifecycle

- Creation: A registered user submits a proposal.
- Includes a description.
- Can be standard or Funding Proposal (requests funds).
- Status: PENDING - Awaiting voting commencement.

# Making Decisions: Voting Mechanism

- Opening Vote: Proposal moves from PENDING to VOTING.
- Casting Votes: Only one vote per user per proposal.
- Users commit X credits to a vote (X > 0).
- Vote weight = sqrt(X), Cost = X^2 voice_credits.

# Determining the Outcome

- Closing Vote: Voting period ends.
- Quorum Check: Minimum unique voters required.
- If not met - REJECTED.
- If met - Calculate total QV Score (sum of sqrt(X)).
- Compare to approval_threshold.
- If score >= threshold - APPROVED, else REJECTED.

# Actioning Decisions: Proposal Execution

- Only APPROVED proposals proceed to execution.
- Standard Proposals: Marked as EXECUTED.
- Funding Proposals:
- - If funds sufficient: disburse & EXECUTED.
- - If not: remains APPROVED but not executed (FAILED_EXECUTION state).

# Under the Hood: Simulating Smart Contracts

- DAO logic encapsulated in Python classes: DAO, User, Proposal.
- Simulates on-chain smart contract behavior.
- Key Functions:
- - add_user(), create_proposal(), cast_vote(),
- - close_voting_and_tally(), execute_proposal().
- Benefits: Automation, transparency, less need for intermediaries.

# DAO Configuration & Key Parameters

- Approval Threshold: QV score needed for approval.
- Quorum Minimum Voters: Voter threshold for validity.
- Initial Treasury Funds: Starting capital.
- User Voice Credits: Individual influence metric.
- Proposal Types: Standard vs. Funding.
- Voting: Quadratic Voting (weight = sqrt(X), cost = X^2).