

MOSAICO

D4.2: Common Policy Languages Survey

March, 2026

Funded by the European Union

Funded by the European Union under the Grant Agreement No 101189664. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HADEA). Neither the European Union nor the granting authority can be held responsible for them.

D4.2 Common Policy Languages Survey

Work package	WP4 AI Community Governance and Supervision
Task	T4.2 Templates of common governance models
Task contributors	ECL, LIST
Due date	31/03/2026
Submission date	30/03/2026
Type of deliverable	Report
Dissemination Level	Public
Deliverable lead	UY
Version	1.0.0
Authors	UY
Reviewers	IMT, ECL, LIST
Keywords	governance, policy, domain-specific language, open-source

Document Revision History

Version	Date	Description of change	List of contributor(s)
v0.0.1	24/01/2026	Initial creation	Antonio GARCIA
v0.1.0	03/03/2026	Draft for review	Kostas BARMPIS
v1.0.0	30/03/2026	Final version	Kostas BARMPIS and Gwendal JOUNEAUX

Partners

























Funding

Funded by
the European Union

Call	Digital and emerging technologies for competitiveness and fit for the Green Deal (HORIZON-CL4-2024-DIGITAL-EMERGING-01)
Topic	HORIZON-CL4-2024-DIGITAL-EMERGING-01-22: Fundamentals of Software Engineering
Type of action	Research and Innovation Action (RIA)

Disclaimer

Funded by the European Union under the Grant Agreement No 101189664. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HADEA). Neither the European Union nor the granting authority can be held responsible for them.

Copyright Notice

© MOSAICO Consortium, 2026

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.

Executive Summary

This report provides a review of current governance models used in software development and an overview of popular governance language used in industry. It then compares these languages with the MOSAICO governance language, positioning it against them with respect to its expressiveness, giving examples of how their concepts would be expressed in it.

More concretely, we present a synthesis of a recent governance survey, taking the top 30 GitHub repositories identified in it and discussing their governance models. We identify four primary ways that software governance is done: direct democracy, meritocracy, oligarchy, and benevolent dictatorship. Various projects use a combination of these approaches for different parts of their governance, such as direct democracy for PRs, but oligarchy for member approval or removal.

Then, we present concise overviews of seven governance languages used in industry today, giving examples of their abstract syntax as well as short snippets of a policy written in that language. We look at PolicyKit, GitRules, Open Policy Agent, Kyverno, Pulumi, Cloud Custodian and the Software Bill of Materials languages in this report.

Finally, we compare how our MOSAICO governance language fares with respect to these languages, with respect to its expressiveness. We translate the snippets presented in each language summary into the MOSAICO governance language, discussing any missing features that this current version of our language may have.

Based on this comparison with these existing approaches, we argue that the MOSAICO governance language provides solid abstractions to define policies and workflows of policies with fine-grained scoping and role management. Nevertheless, we also highlight in our translation of various policies, the need to extend the language in terms of processes and platforms, specifically in order to allow it to handle scenarios in which legal processes should be triggered, or when artifacts are not only produced but also deployed.

Table of Contents

1 Introduction . . . 9

2 Common governance models . . . 10
2.1 Survey of common Governance Models . . . 10
2.2 The Eclipse Development Process (EDP) . . . 12

3 Common governance languages . . . 14
3.1 PolicyKit and Pika . . . 14
3.2 GitRules (formerly Gov4Git) . . . 15
3.3 Open Policy Agent (OPA) . . . 18
3.4 Kyverno . . . 19
3.5 Pulumi . . . 21
3.6 Cloud Custodian . . . 23
3.7 Software Bill of Materials (SBOM) . . . 25

4 Comparison with MOSAICO Governance Language . . . 26
4.1 PolicyKit and Pika . . . 26
4.2 GitRules . . . 28
4.3 Open Policy Agent and Rego . . . 30
4.4 Kyverno, Pulumi and Cloud Custodian . . . 31
4.5 Software Bill of Materials . . . 33

5 Conclusion . . . 34

List of Figures

1 Use of Pika for defining the Slack channel rename policy in PolicyKit, by Wang et. al.[10] 17
2 GitRules voting on an issue[11] 18
3 GitRules PR being accepted and eligible for rewards[11] 19

List of Tables

1 Governance Models 11
2 Tasks explicitly mentioned in the governance file 11
3 Decision-making process followed 12
4 Summary of repositories surveyed 13
5 Comparison of policy languages based on usability and management of the decision process 27

Abbreviations

CI	Continuous Integration
ECA	Eclipse Contributor Agreement
EDP	Eclipse Development Process
LLM	Large Language Model
OPA	Open Policy Agent
OSS	Open-Source Software
PaC	Policy-as-Code
PR	Pull Request

1 Introduction

Large-scale software development is a collaborative activity, where multiple participants contribute to the evolution of a codebase. The collaborative nature of software development, especially in Open-Source Software (OSS), requires proper governance to ensure effective decision-making and accountability. Governance in software projects defines the roles, responsibilities, and processes for managing contributions, resolving conflicts, and maintaining the integrity of software projects (Cánovas Izquierdo et al., 2023). Governance policies formalize decision rights and procedures so that responsibilities are transparent to all participants. Yet many OSS projects either lack such policies or document them only superficially, obfuscating how decisions are taken (Cánovas Izquierdo et al., 2023). Furthermore, even when documented, policies are often scattered across project resources, making it difficult for contributors to discover and follow the project's rules and procedures (Cánovas Izquierdo et al., 2015). Foundations like the Eclipse Foundation [1] can help mitigate this by providing unified processes, such as the Eclipse Development Process (EDP); This ensures that all relevant projects follow a consistent and discoverable governance baseline, regardless of project size.

At the same time, teams increasingly rely on automation (a.k.a. bots) (Santhanam et al., 2022; Wessel et al., 2018) and AI-powered agents that participate in development workflows, introducing new governance challenges around transparency, accountability, and human-agent collaboration (Rasheed et al., 2024). Thanks to advancements in AI, agents, powered by Large Language Models (LLMs), are becoming more capable of performing complex tasks, collaboration, and even decision-making (Xi et al., 2025). Moreover, human diversity is increasingly recognized as a key factor for effective governance, as diverse teams bring different perspectives and experiences that can lead to better decision-making (Yang et al., 2022), while also benefiting end-users (Gunatilake et al., 2024). Although human-agent collaboration and greater human diversity can drive innovation and productivity, they simultaneously expose a key challenge: the absence of governance mechanisms that explicitly and comprehensively address such heterogeneous participation.

The previous D4.1 deliverable discussed the current state of governance in open-source, surveyed MOSAICO partners on their expectations for governance policies, and proposed a domain-specific language to formalise governance policies that addressed participant diversity (in terms of roles and human/AI actors). The present deliverable complements D4.1 by offering an additional independent survey of the most commonly used governance models and languages in the open-source community. Our results show that the most common models are democratic and use voting as part of their decision-making process, covering aspects like member nomination/removal and acceptance of pull requests. Regarding languages, we noticed that there is a mix of general-purpose governance (like OPA) as well as domain-specific governance for aspects like collaboration (GitRules) or cloud deployment (Kyverno, Pulumi, Cloud Custodian) being used.

The rest of the document is structured as follows. Section 2 investigates which governance models are commonly used in open-source projects, by surveying the available literature and examining a subset of a well-known dataset of governance policies. Section 3 focuses instead on the policy-as-code languages that are currently being used by open-source projects to support the definition and enforcement of their governance. Section 4 compares

[1] https://www.eclipse.org/

the expressivity of the MOSAICO Governance Language presented in D4.1 with that of the various policy-as-code languages. Section 5 concludes this report with a summary of the findings and how they will influence further work in WP4.

2 Common governance models

Governance models in software projects draw from organizational and political theory, where various typologies have been established. Traditional models include *representative democracy*, where elected representatives make decisions on behalf of constituents; *direct democracy*, enabling all members to vote on decisions; *meritocracy*, where authority derives from demonstrated competence; *oligarchy*, where power concentrates in a small elite group; and *autocracy* or *benevolent dictatorship*, concentrating decision power in a single leader (Dahl, 2008; Held, 2006; Mintzberg, 1979). OSS projects often adapt and blend these models, such as using meritocratic principles for technical decisions while applying consensus-based approaches for community matters (De Noni et al., 2013; Markus, 2007).

Due to the open nature of OSS development, OSS projects often face challenges in decision-making and management, such as the participation inequality problem: everybody uses OSS, but few contribute back to its development (Gasparini et al., 2020; Marsan et al., 2019). The lack of key governance information deters potential contributors from joining the project, as they are unsure about the project's decision-making processes and how to effectively contribute (Cánovas Izquierdo et al., 2023). Documentation is therefore key for OSS projects to communicate about the goals, organization, and processes of the project. Recently, formal governance systems have been increasingly adopted by communities (Chakraborti et al., 2024). The `GOVERNANCE.md` file[2] aims to provide a standardized format for documenting governance in OSS projects.

The rest of the section presents an independent analysis of a recent survey on governance in open-source projects and presents the governance model of the Eclipse Foundation, a partner in the MOSAICO project.

2.1 Survey of common Governance Models

Yan et al. (Yan et al., 2023) analysed 710 `GOVERNANCE.md` files from GitHub repositories, revealing that while adoption is growing, many projects still lack comprehensive governance documentation.

Among the entries in Yan et al.'s dataset, Ait et al. (Ait et al., 2025) selected the top 30 projects with the most stars (as a proxy of their relative importance in the OSS community) in their work. We took this curated set and labelled them based on which of the traditional models they were closest to, while also identifying any cases where they blended approaches for different matters.

Table 1 shows a summary of various common governance models found and how many times they appeared in this curated set. Note that some projects may have had a combination of governance models, resulting in the overall counts being greater than the number of use-cases (this applies to all three tables discussed in this section).

[2] https://governance.md/

The results show the majority of projects using direct democracy at least as part of their governance model, with oligarchy having a substantial percentage as well. In one of the cases, the policy was so vague that a governance model could not be inferred at all.

Table 1: Governance Models

Model	Count	%
direct democracy	18	62
oligarchy	11	38
autocracy (benevolent dictatorship)	4	14
meritocracy	3	10
other	1	3
unknown	1	3

Next, we looked at which tasks these repositories included in their governance files, summarised in Table 2.

The results show that the majority of policies include nomination of new members, with removal of existing members, as well as policies on submitting/accepting pull requests and patches also having a substantial inclusion. Some projects included high-level tasks like "decision making" or "contribution", which can be too vague or broad for a governance policy.

Table 2: Tasks explicitly mentioned in the governance file

Task	Count	%
nomination	15	52
removal	10	34
PR/patch	10	34
generic decision making	7	24
GH issues/proposal submission	4	14
contribution	2	7

Finally, we looked at how decisions were being made in these projects, summarised in Table 3. The majority of projects use voting as part of their decision-making process, some of which require an absolute majority, or have weighted voting. Many projects also use

reviews or rely on lazy consensus for accepting changes (such as PRs). Relating these to Table 1: voting, review, consensus, absolute majority and lazy consensus are mostly present in democratic models, weighted voting in meritocratic models and veto and owner decision in benevolent dictatorships. Table 4 shows a summary of all the repositories, their governance model(s), as well as the tasks and roles those models encompass.

Table 3: Decision-making process followed

Decision	Count	%
voting	16	55
review	6	21
consensus	5	17
absolute majority	3	10
veto/lazy consensus	2	7
weighted voting	1	3
owner decision	1	3

2.2 The Eclipse Development Process (EDP)

The Eclipse Development Process[3] (EDP) is a governance model for open-source software (and its broader ecosystem), which we will use as an example of a widely adopted industry-standard model. EDP operationalises three open-source pillars:

- Openness - Everyone is provided the same opportunity and everyone can participate with the same rules. Importantly, vendor neutrality allows anyone, such as direct competitors in the market, to be a part.
- Transparency - Project discussions, minutes, deliberations, project plans, etc. are open, public, and easily accessible.
- Meritocracy - Eclipse follows a meritocratic system. The more someone contributes, the more responsibility they will earn. Leadership roles in Eclipse itself are merit-based and earned by peer acclaim.

Eclipse projects have a development team led by their project leaders. This development team is composed of committers and contributors. contributors are those who contribute code, fixes, etc. to the project. Committers have write access to the project's resources (such as the source code repository, the website, etc.) and are expected to directly influence the project's development. Contributors who earn the trust of the project's committers can,

[3] https://www.eclipse.org/projects/dev_process/

Table 4: Summary of repositories surveyed

Repository	Model	Tasks	Roles	Decisions
changendog86b-bfe	meritocracy	nomination, removal	maintainer, seniorMaintainer { vote value: 2.0 }	weighted voting
emissary-ingress-community	direct democracy	nomination, removal	maintainer	voting
nodejs-web-server-frameworks	direct democracy	nomination	teamMember	voting
chakra-ui-chakra-ui	direct democracy	nomination, removal, PR	contributor, collaborator, maintainer, leadMaintainer	voting
oam-dev-spec	direct democracy	nomination, removal	maintainer	voting
consensus-orchestrate-quick-start	direct democracy	decisionMaking	approver, benevolentDictator	veto
skydive-project-skydive	other	nomination	collaborator	consensus
assaultcube-ac	direct democracy, meritocracy	nomination	committeeMember	review
finos-FDC3	direct democracy, oligarchy	appealPR	maintainer (other roles: editor, participant, discussionGroup)	voting
microsoft-PowerShellForGitHub	oligarchy	nomination	powerShellForGitHubCommittee (other roles: projectLead. . .)	absolute majority
qir-alliance-.github	direct democracy	working group decisions	Maintainer, Editor, Participant	dominant view
rustwasm-team	direct democracy, oligarchy	nomination, TeamConsensus, PR	Member, Team, CoreTeam, Collaborator	absolute majority
submarine-io-submariner	direct democracy, autocracy	nomination, removal, PR	ProjectOwners, Committers, CNCF	voting, owner decision
telepresenceio-telepresence	direct democracy	nomination, removal, GH issues	Maintainers	review, voting
ConsenSys-validator-smart-contracts	democracy, autocracy	Decision making	BenevolentDictator, Approvers, Contributors	voting
vmware-tanzu-tanzu-framework	oligarchy	Proposal submission, PR	Users, Contributors, Maintainers	lazy consensus
ConsenSys–besu-plugins	oligarchy, autocracy	Decision making	BenevolentDictator, Approvers, Contributors	voting
vmware-tanzu-besu-plugins	oligarchy, autocracy	Decision making	BenevolentDictator, Approvers, Contributors	lazy consensus
AcademySoftwareFoundation-OpenColorIO	oligarchy	nomination, removal, PR	TSC, Committers (other roles: Contributors, Emeriti)	veto
wrig-docs	oligarchy	ContributionTask	Contributors, Maintainers (other roles: Owners)	majority
getdok-governance	direct democracy	nomination, removal	TABMembers (other roles: Facilitator, CommunityMembers)	vote
govex-open-data-getting-started	direct democracy	nomination, removal	Chair, CommitteeMembers (other roles: CommunityMembers)	vote
Chem4Word-Version2	direct democracy, oligarchy	Patches	Committers, PMC (other roles: Chairperson, Users, Contributors)	lazy consensus
dnivernments-dlmmntdns-ccoreDNS	direct democracy, meritocracy	nomination, removal	ProjectLead, Maintainers (other roles: Contributors)	consensus
interpretml-ebm2onnx	direct democracy	nomination, removal, PR	Maintainers, Contributors, TSC	majority
kubedge-kubedge	unknown	SIG management	SIGChairs, SIGMembers, Contributors, Maintainers	review
open-quantum-safe-oqs-demos	direct democracy	MembershipTask	Members (other roles: Contributors, Sponsors)	review
SaferBus-SaferTest	direct democracy	nomination, PR	Contributors, Contributors (other roles: ProjectLead)	review
mapitoblio-governance	oligarchy, autocracy	Contribution and participation	Contributors, MDT, NumFOCUS, Users	veto, voting

through election, be promoted to committer for that project. This is done through an election process and will subsequently allow those individuals, should they become committers, to then be part of future election processes and decision-making in the project. The removal of inactive or disruptive committers is the preview of the project leaders.

3 Common governance languages

The need to make governance explicit, machine-checkable, and enforceable has motivated research on governance-as-code systems for online communities and Git-based ecosystems. For instance, PolicyKit (Zhang et al., 2020) encodes roles, proposals, and procedural rules to make community governance executable, while also providing Pika (Wang et al., 2024), a visual programming interface to facilitate creating governance policies for non-programmers. Gov4Git[4] (now GitRules), stores governance state and protocols in-repo to enable auditable, verifiable processes.

In practice, projects operationalize governance through policy-as-code and platform controls: (1) legal sign-off such as ECA (Eclipse Contributor Agreement) and DCO (Developer Certificate of Origin); (2) repository rules such as CODEOWNERS[5] and branch protection or required reviews; (3) rules engines like Mergify[6] and Policy-Bot[7] that declaratively specify merge gates and approval constraints; and (4) general policy frameworks (e.g., Open Policy Agent or OPA[8]) integrated into CI to enforce organization-wide policies on configuration and process.

A recent study (Foalem et al., 2026) identified 28 unique Policy-as-Code (PaC) tools and analysed 399 OSS projects using one of these tools, finding that among the nine tools found in those projects, the top four tools (OPA, Kyverno, Pulumi, and Cloud Custodian) accounted for 82% of the repositories. Among those 399 projects, 32% were reusable toolkits (e.g. libraries or frameworks), 37% were documentation (e.g. tutorials or demos), 17% were dedicated to automate the provisioning of software (DevOps), and 12% were applications: only 6 were found to automate machine learning lifecycles (MLOps), and 1 was dedicated to AI techniques. The study utilised a combination of AI-driven labelling with manual cross-checking by multiple researchers to derive a taxonomy of typical uses of PaC languages, with the most common categories being security governance (65.4% of PaC files) and compliance governance (21.3%).

In the rest of the section, we will provide high-level descriptions of a selection of the above governance languages, largely divided into those intended to manage communities (e.g. PolicyKit), and the PaC languages dedicated to codifying specific concerns (e.g. OPA and Kyverno).

3.1 PolicyKit and Pika

PolicyKit is an open-source tool for evolving governance in online communities (such as Slack channels). It describes governance as a series of actions and policies, written as scripts. It

[4] https://gov4git.org/
[5] https://docs.gitlab.com/user/project/codeowners/
[6] https://mergify.com/
[7] https://github.com/palantir/policy-bot
[8] https://www.openpolicyagent.org/

uses a custom engine for evaluating policies[9].

Each policy contains *Action Types* which specify its scope. Various base policies are provided that can be used as fallback policies when more specific ones are not yet defined.

Policy code is divided into discrete blocks: filter, initialize, check, notify, pass, and fail; scope is not shared across blocks:

- Filter - allows to further filter down the scope of the policy - for example, the community has decided that Friday evenings are a free-for-all in a particular channel
- Initialize - setup code that must be completed once at the start of the policy
- Check - returns PROPOSED, PASSED, or FAILED - for example, it may test whether a vote has reached a quorum, or whether an elected individual has responded to the proposal. As long as an action is still PROPOSED, check will run periodically until it returns PASSED or FAILED
- Notify - code for notifying members (soliciting user input)
- Pass - if an action is passed (e.g. check returned PASSED) - i.e. post-action clean-up tasks
- Fail - if the action fails to pass the policy - i.e. invoking fall-back actions due to failure

An example of a simple policy is seen in Listing 1.

This is a policy for calling for a vote when a channel rename is proposed. The policy reverts the channel rename until the vote is held, and then re-applies it only if the vote passes.

PolicyKit offers Pika, a GUI for defining these policies. For example, the Slack channel rename policy can be created using Pika as seen in Figure 1[10].

3.2 GitRules (formerly Gov4Git)

GitRules adds collective governance to pre-existing collaboration environments, such as GitHub, GitLab, Jira or Linear. It is an open-source protocol that uses a blockchain-like structure where a ledger of credits is mirrored by the Git repositories of all its members.

GitRules has two roles: collaborators (also called participants or members) and organizers (also called maintainers). In GitRules:

- Maintainers can issue credits to anyone.
- Collaborators create project issues and PRs freely.
- Collaborators can vote on managed issues and PRs.
- PRs with positive votes are considered “eligible”.
- An eligible PR will freeze an issue until the PR is accepted or rejected.
- Maintainers accept a PR by merging it.
- Accepting a PR triggers distribution of awards and bounties.

Every project in GitRules has an associated virtual currency, called credits. Credits are owned by collaborators and the project’s treasury. Credits represent equity in the project, and Credits entitle their owners to participate in project governance. Credits can be earned for code contributions and for participation in governance processes (such as voting) that lead to real positive outcomes. Members participate in governance by staking a desired

[9] https://policykit.readthedocs.io/en/latest/design_overview.html#policy-engine-evaluation-loop
[10] https://arxiv.org/html/2310.04329v2

Listing 1: Defining a policy for voting on Slack channel renames

```
{
    "policy_kind": "Platform",
    "action_types": [
        "slackrenameconversation"
    ],
    "name": "Vote on renaming Slack channels",
    "description": "When a channel is renamed in Slack, revert the change
       and perform a vote. If the vote passes, execute the rename.",
    "is_bundled": false,
    "filter": "return True\n\n",
    "initialize": "pass\n\n",
    "check": "yes_votes = proposal.get_yes_votes().count()\nno_votes =
       proposal.get_no_votes().count()\nlogger.debug(f\"{yes_votes} for, {
       no_votes} against\")\nif yes_votes >= 1:\n  return PASSED\nelif
       no_votes >= 1:\n  return FAILED\n\nlogger.debug(\"No votes yet....\")
       \nreturn PROPOSED\n",
    "notify": "message = f\"Should this channel be renamed to #{action.name}
       ? Vote with :thumbsup: or :thumbsdown: on this post.\"\nslack.
       initiate_vote(text=message)",
    "success": "text = f\"Proposal to rename this channel to #{action.name}
       passed.\"\nslack.post_message(text=text, channel=action.channel,
       thread_ts=proposal.vote_post_id)",
    "fail": "text = f\"Proposal to rename this channel to #{action.name}
       failed.\"\nslack.post_message(text=text, channel=action.channel,
       thread_ts=proposal.vote_post_id)"
}
```

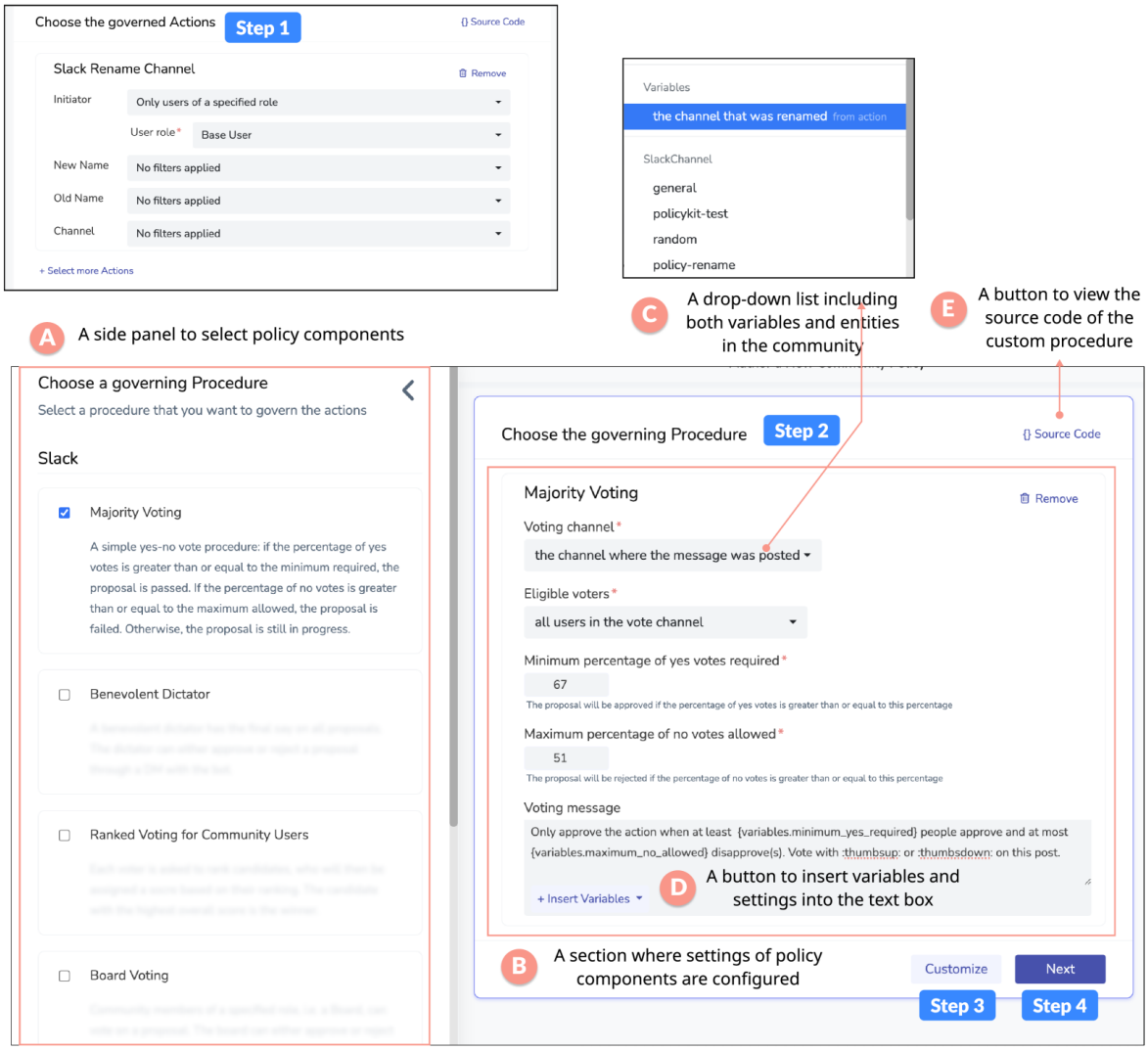


Figure 1: Use of Pika for defining the Slack channel rename policy in PolicyKit, by Wang et. al.[10]

amount of credits to an issue or a PR, which entitles them to exercise a weighted vote “for” or “against”. Votes can be adjusted during the lifetime of an issue or a PR. Stakes are returned to their owners when the issue or PR is closed.

Every issue is associated with a priority score, continuously computed from the votes “for” and “against” it When a PR that resolves the issue is approved (and merged), the PR author is awarded a contribution reward of newly minted credits in an amount proportional to the resolved issue’s priority score. Due to this reward mechanism, the priority score of an issue can be viewed as the value that the community assigns to the issue being resolved. This mechanism incentivizes contributions for higher priority and higher value issues.

Every PR has a claim on resolving one or more issues. Every PR is associated with an approval score, continuously computed from the votes “for” and “against” it When a PR’s approval score reaches a desired threshold, it can be merged. Maintainers control merge decisions and can overwrite the approval mechanism.

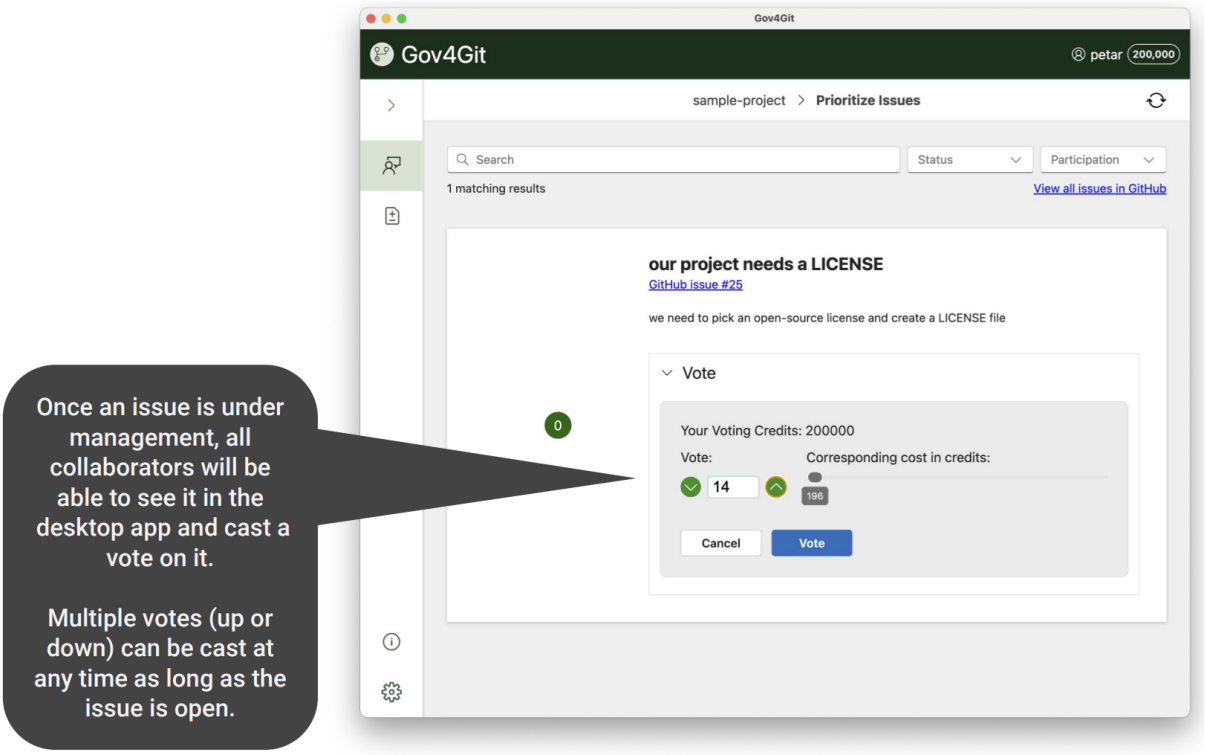


Figure 2: GitRules voting on an issue[11]

When an issue is closed due to being resolved by a PR, participants who voted for the issue are awarded a participation reward in proportion to their votes When an issue is cancelled, participants who voted against it are rewarded analogously. Similarly, when a PR is approved, participants who voted for are awarded a participation reward in proportion to their votes. When a PR is rejected, participants who voted against are rewarded analogously.

The total supply of credits always equals the value of all merged PRs, plus any credits issued by the treasury, minus any credits burned by it. Maintainers can (transparently) accept or reject new member applications, issue or burn credits, transfer credits to and from any accounts, or overwrite PR approval decisions (possibly against the popular vote).

Figures 2 and 3 from a GitRules walkthrough[11] show the GitRules UI for issue voting and a PR being approved respectively.

3.3 Open Policy Agent (OPA)

OPA is a general-purpose policy engine that offers policy enforcement across the stack. Its provides a high-level declarative language Rego[12], that lets you specify policies for a wide range of use cases. For example, OPA can enforce policies in applications, proxies, Kubernetes, CI/CD pipelines, API gateways etc.

Rego defines policies, which contain rules[13]. A rule is made up of:

- keywords - i.e. contains, every, if, not

[11] https://assets.super.so/6d85bda8-80fa-4a98-a31f-c2c0cba4df6a/files/40fe3fc6-d12c-42d5-8780-847fc43ce781/DEPLOY-MANAGE-COLLABORATE-DETAILED.pdf
[12] https://www.openpolicyagent.org/docs/policy-language
[13] https://www.openpolicyagent.org/docs/policy-reference

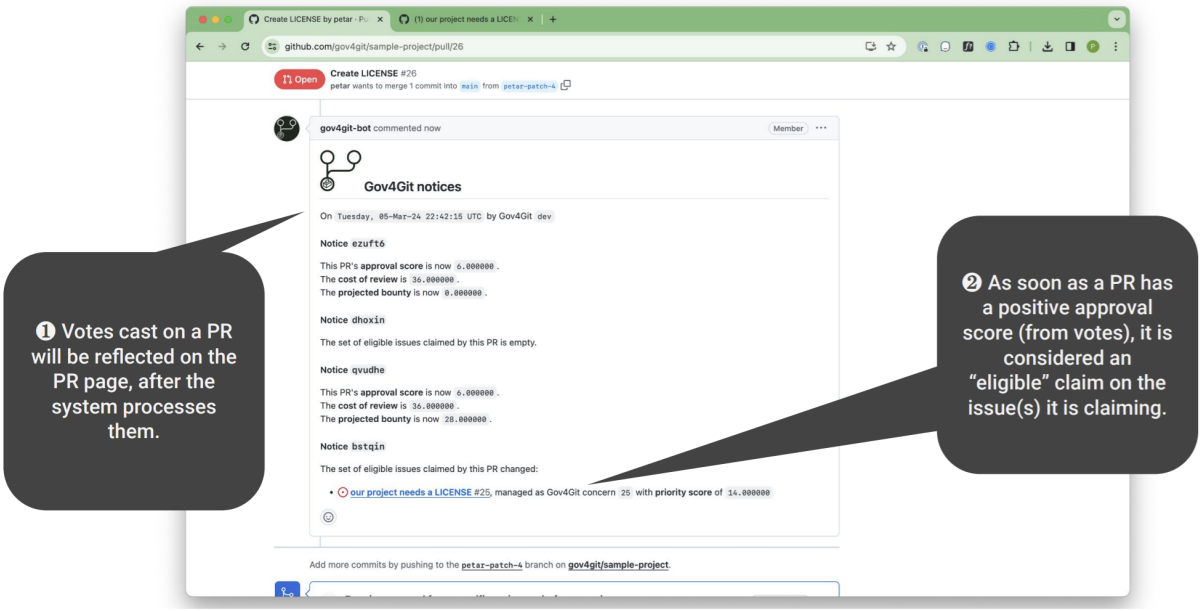


Figure 3: GitRules PR being accepted and eligible for rewards[11]

- scalars - strings, composite values
- arrays (ordered values), objects (unordered key-value), sets (unordered values)
- variables - can be used both as input (when a value is given) and output (when a value is not given)
- expressions - i.e. for assignment and comparison, logical operations like for all and exists
- builtin functions - i.e. aggregate functions, comparison functions, regex functions
- user defined functions

A simple example of a policy written in Rego can be seen in Listing 2. In this listing, the input is checked for three things, before the payment is allowed: the account is open, the account holder has low or medium risk, and the transaction amount is 1000 or less.

Listing 2: Policy on account payments

```
package payments

default allow := false

allow if {
        input.account.state == "open"
        input.user.risk_score in ["low", "medium"]
        input.transaction.amount <= 1000
}
```

3.4 Kyverno

Kyverno is a cloud native policy engine. Originally built for Kubernetes, it can now also be used outside of Kubernetes clusters as a unified policy language. On installation, Kyverno

runs as a dynamic admission controller in a Kubernetes cluster. It receives validating and mutating admission webhook HTTP callbacks from the Kubernetes API server and applies matching policies to return results that enforce admission policies or reject requests. Kyverno supports the following policy types[14]:

- ValidatingPolicy - Validate Kubernetes resources or JSON payloads
- MutatingPolicy - Mutate new or existing resources
- GeneratingPolicy - Create or clone resources based on flexible triggers (declarative language)
- DeletingPolicy - Deletes matching resources based on a schedule
- ImageValidatingPolicy - Verify container image signatures and attestations

The **ValidatingPolicy** type extends the Kubernetes ValidatingAdmissionPolicy type for policy evaluations. A ValidatingPolicy is a superset of a ValidatingAdmissionPolicy and contains additional fields for Kyverno specific features:

- spec.evaluation - defines how the policy is applied and how the payload is processed. It can be used to enable or disable admission request processing and background processing for a policy.
- spec.webhookConfiguration - defines properties used to manage the Kyverno admission controller webhook settings.
- spec.autogen - defines policy auto-generation behaviours, to automatically generate policies for pod controllers and generate ValidatingAdmissionPolicy types for Kubernetes API server execution.

The **MutatingPolicy** type extends the Kubernetes MutatingAdmissionPolicy type for mutation operations. Similarly to ValidatingPolicy, a MutatingPolicy is a superset of a MutatingAdmissionPolicy and contains additional fields for Kyverno specific features:

- spec.evaluation - defines how the policy is applied and how the payload is processed. It can be used to enable or disable admission request processing and existing resource mutation for a policy.
- admission.enabled - when set to true, the policy is applied during admission requests (CREATE, UPDATE operations). This is the primary use case for mutating resources as they are being created or updated.
- mutateExisting.enabled - when set to true, the policy is applied to existing resources in the cluster through background processing. This allows you to mutate resources that already exist without requiring them to be recreated or updated.
- spec.webhookConfiguration - defines properties used to manage the Kyverno admission controller webhook settings.
- spec.autogen - defines policy auto-generation behaviours, to automatically generate policies for pod controllers and generate MutatingAdmissionPolicy types for Kubernetes API server execution.

The **GeneratingPolicy** type enables the creation of Kubernetes resources using Kubernetes' Common Expression Language (CEL) expressions. It provides the same core functionality as Kyverno's generate rules, but is designed with a CEL-first approach for alignment with Kubernetes' policy standards.

The **DeletingPolicy** type allows cluster administrators to automatically delete Kubernetes resources matching specified criteria, based on a cron schedule. This policy is helpful for implementing lifecycle management, garbage collection, or enforcing retention policies.

[14] https://kyverno.io/docs/policy-types/overview/

Unlike admission policies that react to API requests, DeletingPolicy runs periodically at scheduled times, evaluates existing resources in the cluster and deletes resources when matching rules and conditions are satisfied.

The **ImageValidatingPolicy** type is for verifying container image signatures and attestations. It extends the Kyverno ValidatingPolicy with the following additional fields for image verification features:

- images - when Kubernetes resources are evaluated, images for pods and pod templates are automatically extracted for processing. For custom resources, or for JSON payloads, this field can be used to declare CEL expressions that extract images from the payload.
- spec.matchImageReferences - defines rules for matching container images. It allows specifying glob patterns or CEL expressions that specify which images the policy should match.
- attestors - declares trusted signing authorities, such as keys or certificates.
- cosign attestors - use public keys, keyless signing, transparency logs, certificates, or TUF-based metadata for image validation.
- notary attestors - use certificates and optional Time Stamp Authority (TSA) certificates for image signature verification.
- validationConfigurations - defines settings for mutating image tags to digests, verifying that images are using digests, and enforcing image validation requirements across policies.
- credentials - specify the authentication information required to securely access and interact with a registry.

An example of a simple ValidatingPolicy can be seen in Listing 3[15]. In this example, the deployments (line 18) are checked that they have labels (line 21).

3.5 Pulumi

Pulumi offers cloud governance through a combination of ready-to-use policy packs for common compliance frameworks (like CIS, PCI DSS, HITRUST, and NIST) and user-defined policy-as-code[16].

Policies are individual rules that validate infrastructure configuration (e.g., “S3 buckets must be private” or “VMs must use approved instance types”). Policy packs are versioned collections of related policies that are published and managed together. Custom packs can be written in a variety of languages like TypeScript, JavaScript, or Python. OPA is also supported as an experimental language for defining policies. Policy groups apply policy packs to specific stacks or cloud accounts, allowing stricter policies in production and more permissive policies in development environments.

Pulumi has two modes of policy enforcement:

- preventative - validates Pulumi stack resources during “pulumi preview” and “pulumi up”, blocking deployments when violations are detected, to prevent non-compliant resources from being created.
- audit - continuously scans resources discovered through Insights Discovery, to identify violations across all infrastructure, to provide visibility without blocking operations.

[15] https://kyverno.io/docs/policy-types/validating-policy/
[16] https://www.pulumi.com/docs/insights/policy/

Listing 3: ValidatingPolicy for checking deployment labels[15]

```
{
apiVersion: policies.kyverno.io/v1
kind: ValidatingPolicy
metadata:
  name: check-deployment-labels
  annotations:
    policies.kyverno.io/title: Check Deployment Labels
    policies.kyverno.io/category: Other
    policies.kyverno.io/severity: medium
spec:
  validationActions:
    - Audit
  matchConstraints:
    resourceRules:
      - apiGroups: [apps]
        apiVersions: [v1]
        operations: [CREATE, UPDATE]
        resources: [deployments]
  validations:
    - expression: >-
        'environment' in object.metadata.labels
      messageExpression: >-
        "Deployment " + object.metadata.name + " is missing required label '
           environment'"
      message: Deployment is missing required label
}
```

An example of a simple TypeScript policy can be seen in Listing 4. This policy ensures that every S3 bucket name is prefixed by the company name, in this case “mycompany-” (line 20).

Listing 4: Policy to check S3 bucket prefixes

```
1
2  import * as aws from "@pulumi/aws";
3  import { PolicyPack, validateResourceOfType } from "@pulumi/policy";
4
5  // Create a new policy pack.
6  new PolicyPack("policy-pack-typescript", {
7      // Specify the policies in the policy pack.
8      policies: [{
9          // The name for the policy must be unique within the pack.
10         name: "s3-bucket-prefix",
11
12         // The description should document what the policy does.
13         description: "Ensures S3 buckets use the required naming prefix.",
14
15         // The enforcement level can be "advisory", "mandatory", or "
               disabled".
16         enforcementLevel: "mandatory",
17
18         // The validateResourceOfType function allows you to filter
               resources. In this case, the rule only applies to S3 buckets and
               reports a violation if the bucket prefix doesn't match the
               required prefix.
19         validateResource: validateResourceOfType(aws.s3.Bucket, (bucket,
               args, reportViolation) => {
20             const requiredPrefix = "mycompany-";
21             const bucketPrefix = bucket.bucketPrefix || "";
22             if (!bucketPrefix.startsWith(requiredPrefix)) {
23                 reportViolation(
24                     `S3 bucket must use '${requiredPrefix}' prefix. Current
                           prefix: '${bucketPrefix}'`);
25             }
26         }),
27     }],
28 });
```

3.6 Cloud Custodian

Cloud Custodian offers cloud governance, offering a domain language for defining policies to run on specific cloud vendors. These policy definitions, written in YAML, contain the following common aspects (across all vendors):

- filters - i.e. values, list, events, reduce

- actions - webhook action (with information about your resources)

The remainder of the policy definition expressiveness is coupled to the specific vendor, like AWS. Cloud Custodian currently supports AWS, Azure, GCP, OCI and Tencent, as well as having Kubernetes-specific features[17].

For AWS, vendor specific functionality related to Lambdas, CloudWatch, S3 Logs, Security Hub etc. are available. A simple policy leveraging Lambdas for real-time monitoring can be seen in Listing 5[18]. In this listing, the policy will automatically create a CloudWatch Event Rule triggered Lambda function in the account and region, which will be triggered anytime a user logs in from an invalid IP address (lines 14-18).

Listing 5: Policy to check for a valid login IP address

```
1  policies:
2
3    - name: invalid-ip-address-login-detected
4      resource: account
5      description: |
6        Notifies on invalid external IP console logins
7      mode:
8         type: cloudtrail
9         events:
10           - ConsoleLogin
11     filters:
12       - not:
13           - type: event
14             key: 'detail.sourceIPAddress'
15             value: |
16                '^((158\.103\.|142\.179\.|187\.39\.)
                      ([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])
17                \.([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]))
                      |(12\.([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])
18                \.([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])
                      \.([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]))$'
19             op: regex
20     actions:
21       - type: notify
22         template: default.html
23         priority_header: 1
24         subject: "Login From Invalid IP Detected - [custodian {{ account }}
               - {{ region }}]"
25         violation_desc: "A User Has Logged In Externally From A Invalid IP
               Address Outside The Company's Range:"
26         action_desc: |
27             "Please investigate and revoke the invalid session along
28             with any other restrictive actions if appropriate"
29         to:
```

[17] https://cloudcustodian.io/docs/quickstart/index.html
[18] https://cloudcustodian.io/docs/aws/examples/index.html

```
30          - CloudAdmins@Company.com
31          - SecurityTeam@Company.com
32        transport:
33          type: sqs
34          queue: https://sqs.us-east-1.amazonaws.com/12345678900/cloud-
              custodian-mailer
35          region: us-east-1
```

3.7 Software Bill of Materials (SBOM)

A software bill of materials (SBOM) lists every component that software depends upon/uses. It includes information like:

- Component name: log4j
- Version used: log4j 2.14.0
- Supplier name: Apache Software Foundation
- License type: Apache 2.0
- Dependency relationships: How components rely on one another
- Unique identifiers: Elements such as SWID tags or Package URLs (purl)

These details are structured using open standards like SPDX[19], CycloneDX[20] or SWID[21], to keep them machine-readable and easy to integrate with various tools such as security vulnerability scanners. Organisations are encouraged by regulators or agencies like NIST[22] to use SBOMs.

For example, the Eclipse Foundation supports the use of SBOMs through tools like Eclipse Dash for license vetting and Eclipse SW360 for component management. These tools enable the use of SBOMs as policy-as-code, to enforce license compatibility and security throughout the lifecycle of projects hosted by Eclipse.

An example of a component dependency (log4j) written in CycloneDX can be seen in Listing 6. Line 3 defines this component as a library used by our software (which is identified by the group "com.example" in line 4). Lines 5 and 6 detail the name and version of the component, while line 7 has its purl. Finally, lines 8-15 expose the ancestors of this component, which are detailed in the same way as the dependency itself (with their type, group, name, version and purl).

Listing 6: Example SBOM excerpt in CycloneDX of a component dependency

```
1 "components": [
2 {
3    "type": "library",
4    "group": "com.example",
5    "name": "log4j-core",
6    "version": "2.14.0",
7    "purl": "pkg:maven/com.example/log4j-core@2.14.0?repository_url=registry.
        example.com",
```

[19] http://spdx.dev/
[20] https://cyclonedx.org/
[21] https://nvd.nist.gov/products/swid
[22] https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity/software-security-supply-chains-software-1

```
   "pedigree": {
   "ancestors": [
    {
     "type": "library",
     "group": "org.apache.logging.log4j",
     "name": "log4j-core",
     "version": "2.14.0",
     "purl": "pkg:maven/org.apache.logging.log4j/log4j-core@2.14.0"
    }
   ]
  }
 }
]
```

4 Comparison with MOSAICO Gov. Language

In this section, we compare the presented policy languages to the governance language created for MOSAICO as part of deliverable 4.1.

In MOSAICO, the objective of the governance language is to provide a way to describe policies detailing the process of selecting a solution agent output. This selection is based on human opinion and on the evaluation by the supervision agents of the proposed solutions. The governance process described is inherently a decision process between supervision agents and humans to select the appropriate result to the initial request.

On the other hand, these policies will generally be implemented by people with little expertise in the inner working of the MOSAICO platform. For this reason, the MOSAICO governance language should provide the abstractions to describe context, participants, policies and actions enacting the decision made. In the broader sense, the usability of the governance language will be a key factor to streamline the use of the MOSAICO platform.

For these reason, we compare the MOSAICO governance language to the seven languages previously detailed. Table 5 details for each language: (1) the provided abstractions and the presence of a GUI to represent its usability, and (2) the policies that can be described, workflows that can be specified, roles that can be used by stakeholders, and how votes are characterized. In the following sections, we details the differences between each language and MOSAICO governance language and provide a translation of a policy in the compared language to the MOSAICO governance language. As most of these language involve platform specific constraint and/or actions, we created these translation assuming an extension of the MOSAICO governance language exists for this platform.

4.1 PolicyKit and Pika

The core differences between the MOSAICO governance language and PolicyKit are the use case and built-in defined behaviours. PolicyKit was designed to manage governance in the context of online communities. Thus, it provides dedicated abstractions such as notification mechanisms and action types based on the Slack platform, while the MOSAICO governance

Table 5: Comparison of policy languages based on usability and management of the decision process

Policy Language	Usability		Decision Process			
	Abstractions	GUI	Policies	Workflow	Roles	Votes
PolicyKit	Scoping, checks, notification, decision actions. Behaviour in Python	Yes	Custom, implemented manually in Python	Possible using side effects on the platform	No	Same value
GitRules	Credits, Votes, PR approval	Yes	Fixed voting policy	Fixed workflow	Fixed	Weighted, credit-based
OPA	Generic : expressions, functions, values	No	None, only checks	Conjunction and disjunction of checks	No	None
Kyverno	Cloud-oriented, scoping, logging, Python expressions	No	None, only checks	Through the admission controller webhooks	No	None
Pulumi	Cloud-oriented, enforcement level, policy groups. Behaviour in TypeScript	No	None, only checks	No relation between policies at runtime	No	None
Cloud Custodian	Cloud-oriented, scoping, messaging, platform dependant	No	None, only checks	Possible using side effects on the platform	No	None
SBoM	Component inclusion, dependencies	Vary	None, only checks	None	No	None
MOSAICO Governance Language	Role-oriented and scoped policy workflows, Extensible	Yes	Voting, Dictatorship, Consensus	Supported by the language	User-defined	Weighted, user-based

language needs to be extended to support slack interactions to trigger policies and for policies to be enacted. Furthermore, to enact the decisions, PolicyKit relies on Python code. This choice allows more freedom in defining specific checks or specific actions to be performed. However, it also require understanding of the used API to describe policies.

Listing 7 present a translation of Listing 1 in the MOSAICO governance language.

Listing 7: Implementation of Listing 1 with MOSAICO Governance Language

```
Scopes:
    Projects:
        SlackServer from Slack {
            Activities :
                ChannelsManagement {
                    Tasks :
                        Rename : NewName {
                            Action: rename
                        }
                }
        }
Participants:
    Roles : Everyone, Commitee

MajorityPolicy RenameChannel {
    Scope: Rename
    DecisionType as BooleanDecision
    Participant list : Everyone
    Conditions :
        Deadline : 2 days
    Parameters:
        ratio : 0
}
```

Considering an extension dedicated to integrate Slack as a platform, we observe that we can express a similar voting mechanism aimed at renaming a channel. However, Listing 1 is missing important information related to the overall process such as: after how many votes/how much time the decision is made or what triggers the evaluation of the check (i.e. is it triggered after each vote?). For this reason, our translation defines a deadline of two days, implying that votes will be collected at that time and the decision will be made and enacted just after.

4.2 GitRules

GitRules is the most similar approach to the MOSAICO governance language as it provides a streamlined way to vote on issues priorization and merge in the repository. A translation of the workflow proposed by GitRules using our language is proposed in listing 8.

Listing 8: Implementation of GitRules with MOSAICO Governance Language

```
Scopes:
```

```
    Projects:
        GithubProject from Github : org/repo {
            Activities :
                PRManagement {
                    Tasks :
                        Merge : Pull Request{
                            Action: merge
                        }
                }
                IssueManagement {
                    Tasks :
                        Importance : Issue{
                            Action: review
                        }
                }
        }
Participants:
    Roles : Maintainers, Collaborators
    Individuals :
        collaborator1 { vote value : 0.5,  role: Collaborators},
        collaborator2 { vote value : 0.75, role: Collaborators},
        collaborator3 { vote value : 1.0,  role: Collaborators, Maintainers}


ComposedPolicy issue_importance {
    Scope: Importance
    Order :
        Execution : parallel
        RequireAll : false
        CarryOver : false
    Phases {
        MajorityPolicy importance_assessment {
            DecisionType as BooleanDecision
            Participant list : Collaborators
            Conditions:
                Deadline : 2 days
            Parameters:
                ratio : 0.5
        }
        MajorityPolicy validation {
            DecisionType as BooleanDecision
            Participant list : Maintainers
            Conditions:
                Deadline : 2 days
            Parameters:
                ratio : 1.0
        }
```

```
    }
}


ComposedPolicy pr_merge {
    Scope: Merge
    Order :
        Execution : sequential
        RequireAll : true
        CarryOver : true
    Phases {
        MajorityPolicy eligibility {
            DecisionType as BooleanDecision
            Participant list : Collaborators
            Conditions:
                Deadline : 2 days
                LabelCondition post: eligible
            Parameters:
                ratio : 0.5
        }
        MajorityPolicy phase_2 {
            DecisionType as BooleanDecision
            Participant list : Maintainers
            Conditions:
                Deadline : 2 days
            Parameters:
                ratio : 0.0
        }
    }
}
```

In this example, we reproduce the voting mechanisms for both the issue prioritization and pull request merge. The main differences with the original approach are the weighted votes and linking of issues with associated pull requests. In GitRules, votes are weighted based on a credit value managed in the tool, while in our language, weights can only be defined at the user level. Furthermore, we currently cannot automatically close issues if a pull request solving this issue is merged.

On the other hand, GitRules only use voting-based policies, and cannot address scenarios such as benevolent dictatorship or consensus-based decision-making.

4.3 Open Policy Agent and Rego

Rego's genericity allows the definition of policies across the entire stack. To enforce the policies, OPA receives data about the task at stake, and computes the policy based on the provided data. This mechanism allows the definition of complex constraints on those data. In contrast, in our governance language, the constraints are predetermined as concepts of the language, allowing less flexibility. For this reason, we could not translate Listing 2 to the

MOSAICO governance language.

On the other hand, the second major difference between the two approaches is the definition of decision processes. Indeed, Rego can only define constraints that must hold for some action to be performed. Decision processes such as voting or consensus cannot be created using Rego.

4.4 Kyverno, Pulumi and Cloud Custodian

Kyverno, Pulumi and Cloud Custodian are three similar approaches providing abstractions to define governance in cloud-oriented contexts. While they differ from one another, they build on the same main concepts, allowing to define constraints on images deployments. Similarly to OPA and Rego, these approaches do not allow the definition of decision processes and remain limited to constraint checking.

Listings 9, 10, and 11 present translations of the examples provided for Kyverno, Pulumi and Cloud Custodian in Sections 3.4, 3.5, and 3.6 respectivelly.

Listing 9: Implementation of Listing 3 with MOSAICO Governance Language

```
Scopes:
    Projects:
        Deployment from Kubernetes {
            Activities :
                Validation {
                    Tasks :
                        CheckLabel : Audit {
                            apiGroups: [apps]
                            apiVersions: [v1]
                            operations: [CREATE, UPDATE]
                            resources: [deployments]
                        }
                }
        }
Participants:
    Roles : Everyone, Commitee, NoOne

LazyConsensusPolicy EnvironmentLabelCheck {
    Scope: CheckLabel
    DecisionType as BooleanDecision
    Participant list : NoOne
    Conditions :
        Deadline : 0 days
        LabelCondition include: environment
}
```

Listing 10: Implementation of Listing 4 with MOSAICO Governance Language

```
Scopes:
    Projects:
```

```
        Deployment from AWS {
            Activities :
                Validation {
                    Tasks :
                        BucketValidation : S3Bucket {}
                }
        }
Participants:
    Roles : Everyone, Commitee, NoOne

LazyConsensusPolicy S3BucketPrefix {
    Scope: BucketValidation
    DecisionType as BooleanDecision
    Participant list : NoOne
    Conditions :
        Deadline : 0 days
        NamingCondition prefix : "mycompany-"
}
```

Listing 11: Implementation of Listing 5 with MOSAICO Governance Language

```
Scopes:
    Projects:
        Cloud from AWS {
            Activities :
                Security {
                    Tasks :
                        CheckLoginIP : Login {
                            Action : warn
                            To: CloudAdmins@Company.com,
                                SecurityTeam@Company.com
                        }
                }
        }
Participants:
    Roles : Everyone, NoOne

LazyConsensusPolicy InvalidIP {
    Scope: CheckLoginIP
    DecisionType as BooleanDecision
    Participant list : NoOne
    Conditions :
        Deadline : 0 days
        IP not : "^((158\.103\.|142\.179\.|187\.39\.)
                    ([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])
                    17 \.([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]))
                    |(12\.([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])
                    18 \.([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])
```

```
                    \.([01]?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]))$"
}
```

In performing the translation of the policies, we observe the necessity to extend the MOSAICO governance language (as for PolicyKit) to introduce relevant platform concepts such as conditions on IPs or metadata of the deployment process (as in the Kyverno example).

4.5 Software Bill of Materials

Software Bill of Materials and their use as policy-as-code can be seen as legal constraint checking. In this sense, SBOM do not provide abstractions to define decision process on which library should be used, but a constraint on which library can be used.

Listing 12 provide the definition of a license checking policy in the MOSAICO governance language. This policy make use of a potential extension of the language allowing to react to dependency changes and trigger a license check with a dedicated condition checking the compatibility with a given license. This policy also allows the licensing expert to oppose the inclusion, allowing further analysis among inherited dependencies.

Listing 12: Implementation of a license checking policy with MOSAICO Governance Language

```
Scopes:
    Projects:
        GitHubProject from GitHub : org/repo {
            Activities :
                Legal {
                    Tasks :
                        LicenceCheck : DependencyChange {
                            files : pom.xml
                        }
                }
        }
Participants:
    Roles : LicenseExperts, NoOne

LazyConsensusPolicy CheckLicenseValidity {
    Scope: LicenceCheck
    DecisionType as BooleanDecision
    Participant list : LicenseExperts
    Conditions :
        Deadline : 2 days
        LicenceCompatibility : MIT
}
```

5 Conclusion

This report presented an overview of the state-of-the-art with respect to governance in software development. It reviewed common governance models, using a peer-reviewed survey from literature, as well as common governance languages used in industry. The models reviewed identified four primary ways that software governance is done: direct democracy, meritocracy, oligarchy, and benevolent dictatorship. Various projects use a combination of these approaches, for different parts of their governance, such as direct democracy for PRs, but oligarchy for member approval or removal. The languages identified and presented in our review were then compared with the MOSAICO governance language, in order to better understand the capabilities and limitations of its first iteration from D4.1.

Based on the comparison with existing approaches in the domain of Policy-as-Code, we argue that the MOSAICO governance language provides solid abstractions to define policies and workflows of policies with fine-grained scoping and role management. However, the translation of various policies highlighted the need to extend the language in terms of processes and platforms, to handle scenarios in which legal processes should be triggered (e.g., license compatibility) or artifacts are not only produced but also deployed (e.g., deploying on cloud infrastructure, or publishing Docker images).

References

Ait, Adem, Gwendal Jouneaux, Javier Luis Cánovas Izquierdo, and Jordi Cabot (2025), "Towards Automated Governance: A DSL for Human-Agent Collaboration in Software Projects", *in*: *40th IEEE/ACM Int Conf on Automated Software Engineering, ASE 2025*. [Cited on page 10]

Cánovas Izquierdo, Javier Luis and Jordi Cabot (2015), "Enabling the Definition and Enforcement of Governance Rules in Open Source Systems", *in*: *Int. Conf. on Software Engineering*, pp. 505–514. [Cited on page 9]

— (2023), "For a More Transparent Governance of Open Source", *in*: *Commun. ACM* 66.8, pp. 32–34. [Cited on pages 9, 10]

Chakraborti, Mahasweta, Curtis Atkisson, Stefan Stanciulescu, Vladimir Filkov, and Seth Frey (2024), "Do We Run How We Say We Run? Formalization and Practice of Governance in OSS Communities", *in*: *Conf. on Human Factors in Computing Systems*, 923:1–923:26. [Cited on page 10]

Dahl, Robert A (2008), *Democracy and its Critics*, Yale university press. [Cited on page 10]

De Noni, Ivan, Andrea Ganzaroli, and Luigi Orsi (2013), "The evolution of OSS governance: a dimensional comparative analysis", *in*: *Scand. J. Manag.* 29.3, pp. 247–263. [Cited on page 10]

Foalem, Patrick Loic, Foutse Khomh, Leuson Da Silva, and Ettore Merlo (Jan. 2026), *An Empirical Study of Policy-as-Code Adoption in Open-Source Software Projects*, arXiv:2601.05555 [cs], doi: 10.48550/arXiv.2601.05555, url: http://arxiv.org/abs/2601.05555 (visited on 01/22/2026). [Cited on page 14]

Gasparini, Mattia, Robert Clarisó, Marco Brambilla, and Jordi Cabot (2020), "Participation Inequality and the 90-9-1 Principle in Open Source", *in*: *Int. Symposium on Open Collaboration*, 6:1–6:7. [Cited on page 10]

Gunatilake, Hashini, John C. Grundy, Rashina Hoda, and Ingo Mueller (2024), "The impact of human aspects on the interactions between software developers and end-users in software engineering: A systematic literature review", *in*: *Inf. Softw. Technol.* 173, p. 107489. [Cited on page 9]

Held, David (2006), *Models of democracy*, Polity. [Cited on page 10]

Markus, M Lynne (2007), "The governance of free/open source software projects: monolithic, multidimensional, or configurational?", *in*: *J. Manag. Gov.* 11.2, pp. 151–163. [Cited on page 10]

Marsan, Josianne, Mathieu Templier, Patrick Marois, Bram Adams, Kevin Carillo, and Georgia Leida Mopenza (2019), "Toward Solving Social and Technical Problems in Open Source Software Ecosystems: Using Cause-and-Effect Analysis to Disentangle the Causes of Complex Problems", *in*: *IEEE Softw.* 36.1, pp. 34–41. [Cited on page 10]

Mintzberg, Henry (1979), "The structuring of organizations", *in*: *Readings in strategic management*, Springer, pp. 322–352. [Cited on page 10]

Rasheed, Zeeshan, Muhammad Waseem, Malik Abdul Sami, Kai-Kristian Kemell, Aakash Ahmad, Anh Nguyen-Duc, Kari Systä, and Pekka Abrahamsson (2024), "Autonomous Agents in Software Development: A Vision Paper", *in*: *Int. Conf. on Agile Software Development*, pp. 15–23. [Cited on page 9]

Santhanam, Sivasurya, Tobias Hecking, Andreas Schreiber, and Stefan Wagner (2022), "Bots in software engineering: a systematic mapping study", *in*: *PeerJ Comput. Sci.* 8, e866. [Cited on page 9]

Wang, Leijie, Nicholas Vincent, Julija Rukanskaite, and Amy Xian Zhang (2024), "Pika: Empowering Non-Programmers to Author Executable Governance Policies in Online Communities", *in*: *Conf. on Human Factors in Computing Systems*, 925:1–925:18. [Cited on page 14]

Wessel, Mairieli Santos, Bruno Mendes de Souza, Igor Steinmacher, Igor Scaliante Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco Aurélio Gerosa (2018), "The Power of Bots: Characterizing and Understanding Bots in OSS Projects", *in*: *Hum. Comput. Interact.* 2.CSCW, 182:1–182:19. [Cited on page 9]

Xi, Zhiheng, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, Qi Zhang, and Tao Gui (2025), "The rise and potential of large language model based agents: a survey", *in*: *Sci. China Inf. Sci.* 68.2. [Cited on page 9]

Yan, Yibo, Seth Frey, Amy X. Zhang, Vladimir Filkov, and Likang Yin (2023), "GitHub OSS Governance File Dataset", *in*: *Int. Conf. on Mining Software Repositories*, pp. 630–634. [Cited on page 10]

Yang, Yang, Tanya Y. Tian, Teresa K. Woodruff, Benjamin F. Jones, and Brian Uzzi (2022), "Gender-diverse teams produce more novel and higher-impact scientific ideas", *in*: *Proc. Natl. Acad. Sci. U.S.A.* 119.36, e2200841119. [Cited on page 9]

Zhang, Amy X., Grant Hugh, and Michael S. Bernstein (2020), "PolicyKit: Building Governance in Online Communities", *in*: *ACM Symposium on User Interface Software and Technology*, pp. 365–378. [Cited on page 14]