MOSAICO

D4.1: Formalization of a Policy Language

September, 2025

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.

D4.1 Formalization of a Policy Language

Work package	WP4 AI community governance and supervision
Task	T4.1 Definition of a policy language to define the governance rules for reaching a consensus
Due date	30/09/2025
Submission date	29/09/2025
Type of deliverable	R - Document, Report
Dissemination Level	Public
Deliverable lead	LIST
Version	1.0.0
Authors	Gwendal Jouneaux (LIST), Jordi Cabot (LIST), Adem Ait
Reviewers	IMT, UDA, UY
Keywords	Domain-Specific Language, Governance, AI Agents

Document Revision History

Version	Date	Description of change	List of contributor(s)
v0.1.0	28/08/2025	First complete version	Gwendal Jouneaux, Jordi Cabot, Adem Ait
v0.2.0	10/09/2025	Addressing UY comments	Gwendal Jouneaux
v0.3.0	19/09/2025	Addressing UDA comments	Gwendal Jouneaux
v0.4.0	25/09/2025	Addressing IMT comments	Gwendal Jouneaux, Adem Ait
v1.0.0	26/09/2025	Final version	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

Executive Summary

The stakeholders involved in software development are becoming increasingly diverse, with both human contributors from varied backgrounds and AI-powered agents collaborating together in the process. This situation presents unique governance challenges when explicit governance policies to decide how decisions are taken (e.g. by majority voting? by consensus? depending on what?) are lacking or unclear. Moreover, diversity is usually overlooked in the policies increasings the risks for minorities of being overlooked. Indeed, while diversity can lead to more innovative solutions, it also requires robust governance mechanisms to ensure equitable representation and ethical decision-making. This report presents a novel Domain-Specific Language (DSL) designed to define and enforce governance policies in systems involving diverse stakeholders, including AI agents. We propose the core principles for such a DSL, emphasizing the explicit modeling of diverse participant profiles (both human and agent), their unique characteristics, roles, and adaptable decision-making processes, with a central focus on fostering fairness and transparency. This conception offers a pathway towards more robust, adaptable, and ultimately automated governance, paving the way for more effective and ethical collaboration.

Table of Contents

List of Figures

List of Listings

Abbreviations

SE	Software Engineering
DSL	Domain-Specific Language
OSS	Open-Source Software
LLM	Large Language Model
AISP	AI Server Protocol
GHAs	GitHub Actions

1 Introduction

Large Language Models (LLMs) have emerged as powerful tools capable of performing complex tasks such as natural language comprehension and generation, reasoning, among others (Chang et al., 2024). Thanks to these capabilities, LLMs can power all types of agents, which are autonomous entities that can perceive their environment and respond to it, interact with other agents, and take actions to achieve specific goals (Wooldridge et al., 1995).

An escalation in the number of agents allows creating multi-agent systems, or as recently named agentic systems, showing collective intelligence that allow them to be used in a wide range of applications (T. Guo et al., 2024). However, despite their potential, they still pose significant challenges (e.g., hallucinations, biases, and ethical concerns) that need to be addressed before they can be safely deployed in real-world scenarios (Mao et al., 2024).

In the MOSAICO project, we envision the future of Software Engineering (SE) as a reliable application of generative AI to SE tasks, enabled by the coordinated and supervised collaboration of – a possibly large number of – different LLM-based AI agents, that we call a Community of AIs (or AI-agent community). The project aims at providing a dedicated platform for the engineering and operation of communities of AIs across the SE life-cycle, handling communication, orchestration, governance, quality assessment, benchmarking and reuse of AI agents.

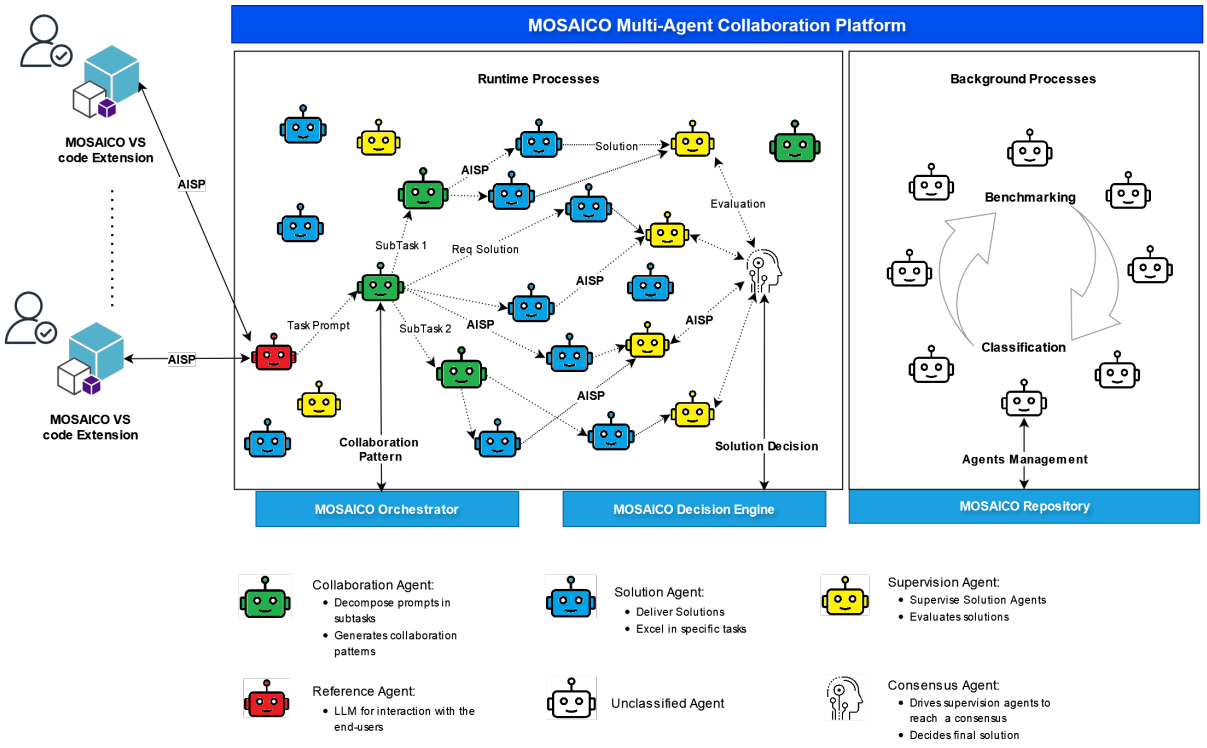


Figure 1: Overview of the MOSAICO Platform

Figure 1 presents an overview of the envisioned MOSAICO platform. On the left, the end-user is presented as using the VS Code editor with the MOSAICO extension allowing the use of the platform. On the right, the MOSAICO repository is in charge of the benchmarking, classification, and management of agents for their (re)use in the platform. In the middle, the runtime processes of the platform take the form of a community of agents communicating using a dedicated protocol, the AI Server Protocol (AISP). Among those agents, we find five main types of

agents. The reference agent (in red) is in charge of the communication with the end-users and communicate the prompts to the appropriate agent. Often, this agent will be a collaboration agent (in green) decomposing the task in subtasks and assigning them to other agents through the generation of collaboration patterns using MOSAICO Orchestrator. The agents responsible for performing the subtasks are solution agents (in blue), excelling in specific domains they deliver solutions to the expected task/subtask. These solutions are then evaluated by supervision agents (in yellow) supervising the work of solution agents. These evaluations are then collected by the consensus agent to decide the final solution to be adopted. To do so, the consensus agent relies on the MOSAICO Decision Engine that applies policies specified using a dedicated governance language. In some cases, these policies can ask for a consensus among the stakeholders (users and agents), requiring the consensus agent to drive discussions among them to reach this consensus.

In this report, we present our first contribution to this vision in the form of a governance language specification. This language will allow specifying policies to describe the decision-making process to enact depending on the context. Then, these policies will be used by the MOSAICO Decision Engine and related consensus agent to enforce the decision at run-time.

Governance is a key aspect of software development, where the collaborative nature of development requires clear guidelines and policies to ensure effective decision-making and accountability. Governance policies help to establish roles, responsibilities, and processes for managing contributions, resolving conflicts, and maintaining the integrity in software projects (Cánovas Izquierdo et al., 2023). Such policies make explicit the decision-making process, ensuring that all the involved stakeholders are aware of how decisions are made and who is responsible for them. However, most projects do not explicitly define governance policies, and those that do are often poorly described and lack clarity (Cánovas Izquierdo et al., 2023). Furthermore, when described, they are usually scattered across different project resources, making it difficult for contributors to understand the rules and procedures that govern the project (Cánovas Izquierdo et al., 2015).

The rapid proliferation of AI-powered agents participating in development tasks, coupled with a growing recognition of the critical role of diverse human backgrounds, presents unprecedented challenges to these established governance paradigms (Bjørn et al., 2023; Rasheed et al., 2024; Rodríguez-Pérez et al., 2021; M. S. Wessel et al., 2018). AI agents, powered by advancements in Large Language Models (LLMs), are moving beyond simple automation to become active participants capable of complex communication, collaboration, and even decision-making (T. Guo et al., 2024). On the other hand, human diversity is also beneficial for software development, as it can lead to more innovative solutions and better decision-making (Yang, Tanya Y Tian, et al., 2022a), while also benefiting end-users (Gunatilake et al., 2024). While this human-agent collaboration and broader human diversity promise significant benefits in innovation and productivity, they also expose a critical gap: the lack of governance frameworks designed to explicitly and holistically manage such multifaceted participation.

Recently, the governance of AI, and in consequence communities of AIs, has gained attention in the scientific literature (Chesterman et al., 2024). Effective governance ensures that agents operate within predefined constraints, maintain accountability, and produce results that meet quality standards. The introduction of agents in existing systems requires a shift in the way we think about governance, as traditional governance models may not be sufficient to address the unique challenges posed by these systems. In the context of MOSAICO, we decided to tackle this issue by providing a new governance Domain-Specific Language (DSL) designed to define and enforce governance policies in systems involving a diverse set of stakeholders, including communities of AIs.

In the rest of the report, we illustrate the application of the DSL in the context of OSS projects. We chose OSS project as they involve many stakeholders with different expertise and potentially include bots or agents in their process, creating a decision process similar to the Human/AI multi-agent system at the core of MOSAICO. The proposed DSL provides a structured approach for expressing governance rules, roles, responsibilities, and decision-making processes. It encompasses various dimensions of governance, including decision-making procedures, positive and negative discrimination, and other methods to ensure a fair representation. We envision our DSL to be field-agnostic, extensible, and adaptable to different contexts, allowing for the inclusion of various stakeholders and their specific needs.

The rest of this report is structured as follows. Section 2 provides background information on DSLs, the notion of diversity with AI agents, and the current state of governance in OSS as example. Section 3 details the result of a preliminary survey on governance policies in the context of AI agents. Section 4 presents the design and implementation of the DSL. Section 5 discusses the coverage and expressivity of the DSL with a case study. Section 6 provides the related work. Finally, Section 7 concludes the paper and outlines future work.

2 State of the Art

2.1 Domain-Specific Languages

Domain-Specific Languages (DSLs) are languages specially designed to help to solve a problem in a particular domain. A DSL is composed of three main elements (Wasowski et al., 2023): (1) abstract syntax, which defines the concepts and relationships of the domain where the language is applied; (2) concrete syntax, which defines the notation of the language (e.g., textual, diagram-based, etc.); and (3) semantics, which defines the meaning of the language constructs. Furthermore, DSLs can be classified into external DSLs, which are generally defined by a grammar; and internal DSLs, which are embedded in a general-purpose programming language (known as host language). By using a DSL, the developer can use domain-specific constructs and therefore address the problem more efficiently (Fowler, 2011).

In the context of governance, a DSL can serve as a formal framework for specifying policies, roles, and decision-making procedures, thereby enhancing the transparency and reliability of agent (and human) interactions. Furthermore, by implementing an engine to execute such DSL, the governance policies can be automatically enforced, reducing the risk of human error and ensuring that the systems operate within the defined constraints.

2.2 Cognitive Diversity of AI Agents

Software engineering teams are becoming increasingly diverse across multiple dimensions. Recent research has highlighted the importance of diversity within software development processes (Bjørn et al., 2023; Rodríguez-Pérez et al., 2021). Diversity is being studied in terms of gender, race or ethnicity, but also in terms of cognitive diversity (Dutta et al., 2023; Giner-Miguelez et al., 2025). Studies have shown that diverse teams tend to produce more innovative solutions (Catolino et al., 2019; Yang, Tanya Y. Tian, et al., 2022b), and impact positively in software productivity (Vasilescu et al., 2015).

Adding to this human diversity, software projects now include a growing number of non-human participants in the form of bots and AI-powered agents (Rasheed et al., 2024; M. S. Wessel et al., 2018). These automated participants have traditionally performed routine tasks

like continuous integration, code quality checks, and dependency management (Kinsman et al., 2021). In the context of OSS, the use of automation mechanisms has become widespread, with GitHub Actions (GHAs) being adopted by approximately 30% of the 5,000 most popular repositories (M. Wessel et al., 2023). These automated tools have begun to play more significant roles in project governance by assisting with issue triaging, code review processes, and community management tasks such as welcoming new contributors and ensuring compliance with Contributor License Agreements (Decan et al., 2022), or management of codes of conduct (Cobos et al., 2025).

However, the rise of LLMs is transforming these agents from mere automation tools into sophisticated collaborators capable of reasoning, communication, and participation in complex decision-making (T. Guo et al., 2024; Xi et al., 2025) – introducing cognitive diversity among AI agents. An example of such change can be seen in the use of GitHub Copilot to manage issues, propose a fix, and iterate with maintainers through pull request comments to iterate on the proposed solution.[1] This profound shift introduces an unprecedented dimension of diversity, presenting novel and urgent conceptual challenges for governance that existing frameworks are missing.

Existing AI governance frameworks, while valuable, predominantly address AI systems as products or services, focusing on principles like fairness, accountability, and transparency from an external or societal viewpoint (Chesterman et al., 2024). However, they fall short in providing concrete mechanisms for integrating these principles into practical governance policies for diverse software teams that include both human and agent participants. In our proposal, we incorporate some identified indicators for AI: (1) autonomy level (Rahwan et al., 2019), to represent how independent the agent can make decisions; (2) explainability (Arrieta et al., 2020), usually understood for understandability, interpretability, explainability, and transparency of the AI system; and (3) human oversight (Shneiderman, 2020), as whether the AI system is under human control or not.

2.3 The Example of Governance in OSS

A previous study illustrated the need for explicit governance policies in OSS (Cánovas Izquierdo et al., 2015) but lacked the mechanisms to address the challenges introduced by participant diversity, particularly the inclusion of AI agents in decision-making processes. Our current proposal aims to fill this gap by incorporating diversity-aware governance mechanisms that acknowledge the different characteristics of participants and ensure equitable representation in decision-making processes.

While the landscape of participants is evolving, the explicitness and nature of governance policies in practice, particularly in OSS, remain a concern. To understand this current state, we replicated the study conducted in (Cánovas Izquierdo et al., 2023), which analyzed the top 25 starred software projects and found that most OSS projects do not have explicit governance policies, and those that do are often poorly defined and lack clarity. The governance evidence is analyzed from the explicit indications of four dimensions: (1) whether there is some workflow to contribute, different from typical pull-based development process; (2) who makes the decisions to accept code, and how; (3) how long it takes to review or accept a contribution; and (4) how to become a contributor.

We found that 68% of repositories reported at least one of the four dimensions, 24% reported none, and 8% reported all four dimensions. The situation is showing slight improve-

[1] https://docs.github.com/en/copilot/how-tos/use-copilot-agents/coding-agent/assign-copilot-to-an-issue

ments from the previous study. In comparison, we have seen an increase in the number of projects that partially discussed governance, from 32% to 68%, mainly because of the report of the contribution process. However, how long it takes to review or accept a contribution is only defined in 16% of the projects, how to become a contributor is only defined in 20% of the projects, and who makes the decisions, and how, to accept code is only defined in 24% of the projects. Only 2 projects (8%) provided a full description of the policies governing them. Note no project included a governance.md file, but reported them in the contributing.md file or in their documentation website. None of the analyzed projects adopted a DSL to specify their governance policies. The use of a DSL could help in the definition, enforcement, and automation of the governance policies (Cánovas Izquierdo et al., 2023).

3 Expectations on Governance Policies

To better understand the user needs regarding governance of software projects, we conducted a preliminary study in the form of a survey. The survey was built as three sections of multiple choices questions preceded by questions to characterize the development context of the participants. Among the seven answers to this survey, five were from industrials, one from academia, and one from the open-source community. We received answers for both small (1-50 employees) and big (250+ employees) companies. The sectors of activity of the companies involved include Digital platform development (in the domain of Internet of Things), mixed/virtual/augmented reality, and AI solutions / software development tools. In terms of software development activities, the primary focus of the surveyed panel revolves around research and development of new technologies, followed by the development of commercial software and internal tools and systems.

3.1 Project Dynamics and Challenges

The first goal of this survey was to understand project dynamics and related challenges. Our first question in this direction was: *What are some common challenging situations or recurring complex aspects you experience in your software development life-cycle?* The most selected answer (71.4%) was about the effective on-boarding of new team member, followed by the automation of development steps and the design of the software architecture.

To better understand the current place of governance in the software development process, we asked the panel which aspects of this process require explicit written policies? For most of the participants, release management, feature prioritization, issue tracking and dependencies management appear as the processes usually demanding explicit governance rules.

Finally, we focused on the decision-making process with three question revolving around the people involved, the forms of participation, and what seemed important to the surveyed people when it comes to decision-making in software projects. For the first two, the answers show that the decisions are typically decided by leaders and senior developers, while usually clients and AI agents are not allowed to participate. Consequently, we observe that the formal voting mechanism is often disregarded in the profit of leader-based, expert opinion, or consensus-building process. For the last question, the most important aspects mainly revolve around transparency, accountability, and clarity of the decision process.

3.2 AI in Project Processes

The second goal of this survey was to understand the place of AI and in particular AI agents in the software development process. We first asked the participants if they were currently using AI agents in any part of their software development process, to which five answered yes.

As a follow-up question, agent users were asked to characterize the role of these agents, while non-users were asked what was the reason to not use agents. The main uses of AI agents reported are code contribution and code analysis, while decision-making/guidance was reported only once. On the other hand, non-users report as the main reason of their choice to be the absence of mature or available agents for their needs. Other notable reasons include integration, development and maintenance costs, explainability, and security concerns.

3.3 Language and Automation for Governance

In the last part of the survey, we asked the participants the envisioned benefits, drawbacks and challenges of, first, a specialized language for governance policies, and then, an automation tool to interpret and enact those policies.

Regarding the governance policy language, participants reported the benefits of increased clarity, as well as consistent application of the policies, easier on-boarding, and better documentation of governance practices. However, participants also foresee an overhead in defining and maintaining those policies, and the risk of the policies to become too rigid.

Considering automation tooling to enact policies, participants mainly reported the benefits of improved compliance and objectivity of data/metrics provided. On the other hand, the participants also foresee security and fault-tolerance concerns, as well as the over-reliance on the automation provided.

3.4 Discussion

In this section, we will discuss the answers provided in the three main sections of the survey. Bar charts of the provided answers as well as percentage can be found in the appendix.

In the first section, the three main challenges identified revolve around the understanding (on-boarding), automation, and decision mechanisms (design choices) of the project processes. These challenges call for an explicit, clear, and actionable description of the processes.

Furthermore, written policies appear to mainly be used in decision-making processes (release management, feature prioritization) and maintenance activities (issue tracking, dependency management). We learned in the second section that AI agents can be used for code analysis and contribution, including maintenance scenarios. However, these agents come with multiple issues that would require human supervision – *e.g.,* explainability, security concerns. In this context, we argue that such verifications and acceptance/rejection of the AI contribution is a decision-making process. This means that such governance scenarios calls for written policies governing decision-making processes, including both human contributors and AI agents.

Moreover, the questions about decision-making processes in the first section of the survey reveal a variety of different forms of participation in this process. Leader-based (or small group based) and expert opinion based mechanisms align with the current leader/expert centric nature of the participants involved in the decision. Yet, the most popular alternative remains the consensus among all the stakeholders. In a context where AI agents can contribute to and analyze the software at stake, their potential involvement in the decision process aligns with the previously depicted governance scenario.

Finally, the third section of the survey explores the benefits and challenges in view of a structured language to specify governance policies and the associated tooling to enact them. The answers show that such language is seen as a mean to provide increased clarity and explicitness, as well as an easier on-boarding for new team members addressing the challenges identified in the first section of the survey when coupled to its automation tooling. However, challenges for this language should be addressed, such as the overhead to define and maintain policies and the expressivity and flexibility of the language to avoid bureaucratic processes and express nuanced governance scenarios.

4 Defining Governance Policies

In the context of governance, a DSL can serve as a formal framework for specifying policies, roles, and decision-making procedures, thereby enhancing the transparency and reliability of the interactions. By using a DSL, the developer can use domain-specific constructs and therefore address the problem more efficiently (Fowler, 2011), which can be further automated and enforced.

To the best of our knowledge, there is no DSL for defining governance policies in systems that address the diversity of participants. With our approach, governance policies can be defined in a more structured way, allowing for the specification of the participants involved, the scope of the policy, and the conditions that must be fulfilled. This is particularly interesting in systems that include agents, as it allows for a clearer understanding of how these agents fit into the overall governance framework.

As mentioned before, a DSL is composed of three main elements (Wasowski et al., 2023): (1) abstract syntax, which defines the concepts and relationships of the domain where the language is applied; (2) concrete syntax, which defines the notation of the language (e.g., textual, diagram-based, etc.); and (3) semantics, which defines the meaning of the language constructs. In the following, we describe the main elements of our DSL.

4.1 Abstract Syntax

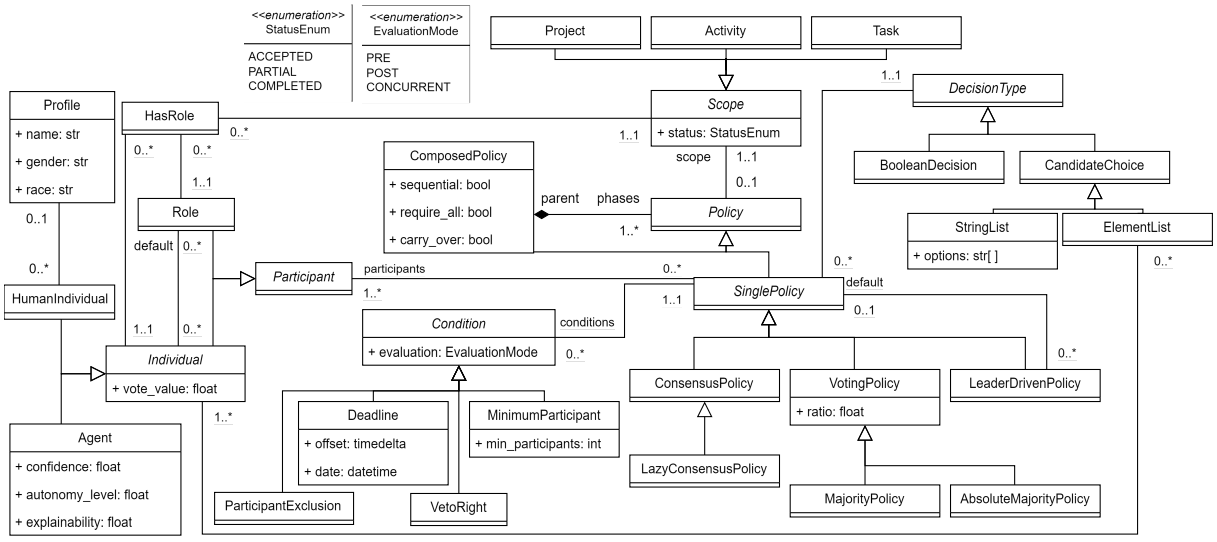


Figure 2: Abstract syntax metamodel of our DSL.

The abstract syntax of our DSL is defined via the metamodel shown in Figure 2. Metamodels restrict the structure of valid models (i.e., DSL instances, in our case, concrete policies) and define the relationships between the different elements of the language (i.e., the different concepts that can be used to define a policy and how these concepts can be linked to each other) (Brambilla et al., 2017). Governance models conforming to this metamodel represent specific sets of governance policies.

Mainly, the metamodel is aimed at representing four main aspects: (1) scope (i.e., where the policy is applied); (2) participants (i.e., who is involved in the policy), (3) policies (i.e., what the policy is about), and (4) conditions (i.e., when it should be applied). Next we describe the main metaclasses of the metamodel according to these four aspects.

Scope. To represent the first aspect, we propose the `Scope` generalization. We envisioned a three-layered structure consisting of the: (1) `Project` metaclass representing the whole system; (2) `Activity` metaclass representing a subset of a project (e.g., development activities); and (3) `Task` metaclass representing the atomic part of a project (e.g., a pull request in the context of OSS projects). This structure allows the extension of the metamodel to include other scopes in the future. Each metaclass represents one level in the atomicity of the scope, thus they can be associated with each other. We also defined a `StatusEnum` enumeration to represent the state, which can be accepted, meaning the policy instance itself is accepted for application; partial, meaning the process is ongoing; or completed, meaning the process is finished. This attribute is optional, but it can be useful to define the status of the scope in the decision engine.

Participants. We define the `Participant` abstract metaclass, which is a generalization of the `Role` and `Individual` metaclasses. The former is intended to represent a group of individuals that share a common role in the project (e.g., developers, maintainers, etc.), while the latter is intended to represent a specific individual (e.g., an owner). To ensure fair representation, we define an attribute to favor contributions from certain individuals in a decision (see `vote_value`). These attributes are optional, but they can be used to calculate the vote value of an individual and favor certain minorities. To represent humans involved in the decision process, we extend the `Individual` metaclass with a `HumanIndividual` metaclass that can be associated to a given `Profile` describing its gender or race. On the other hand, to represent agents (see `Agent` metaclass) in the project, we extend the `Individual` metaclass. We defined a set of attributes aforementioned discussed (see Section 2.2). These quantitative indicators (i.e., `autonomy_level`, `explainability`, and `confidence`) can be used to assess properly the participation of agents in the decision-making. Additionally, we defined a ternary association between the `Individual`, the `Scope` and the `Role` metaclasses (see `HasRole` metaclass) to represent the particular role an individual might have on a specific scope (e.g., a user might be contributor for development, but reviewer for documentation).

Policies. To define policies, we define the policy hierarchy, which allows defining two kinds of policies: single policies and composed policies, which consist of a set of policies (either single or composed). Depending on the method apply for resolution, the policy can be based on: (1) voting, where each participant selects one (or more) candidates among different possibilities; (2) consensus, which is focused on a cooperative dynamic where all participants work together to make the best possible decision for the group; or (3) leader-driven, where one participant (usually the owner) takes the decision.

In our proposal, we generalized the evaluation of voting policies in two kinds: (1) by the number of participants involved in the voting process (see `AbsoluteMajority` metaclass) or (2) by the number of participants that voted (see `Majority` metaclass). In short, the decision is made by assessing whether the most voted option surpasses a threshold based on the number of potential voters or the number of actual voters. For instance, in a voting policy with 10 participants, with a threshold of 50%, if 4 of them vote for approval and 2 for rejection, the

voting policy will be approved with a majority policy, but it will not be approved with an absolute majority policy. This threshold is defined by the `ratio` attribute and is set to 50% by default.

Consensus policies are based on the idea that all participants should agree on a decision before it is made. This is particularly useful in situations where the decision has a significant impact on the project or the community. We provided support for lazy consensus [2], which is a form of consensus where a decision is made if no one objects. It is, as mentioned in the APACHE Foundation, one of the most preferred consensus methods.

Leader-driven policies represent the case where a participant, playing the role of a leader, makes the decision for the group. Because of this decision-making process relies on a single individual, the leader can define a fallback policy (see `default` attribute) in case the decision is not made by the deadline.

In order to express what the policy is selecting, we defined a set of decision types (see `DecisionType` hierarchy). A policy must have one decision type. Policies looking for an acceptance or rejection of a proposal can be represented as a `BooleanDecision` (e.g., a pull request review). Policies looking for a selection among a set of options can be represented as a `CandidateChoice` (e.g., who will become the new leader of a community). Thus, the set of options can be represented as a list of elements of our metamodel (see `ElementList` metaclass) or as a set of strings (see `StringList` metaclass).

These policies can be combined to create more complex policies. Composed policies (see `ComposedPolicy` metaclass) are defined by a set of policies (either single or composed). The evaluation of the composed policy is based on the evaluation of its sub-policies. For this, we propose a set of boolean attributes to define the relationship between the sub-policies: (1) `sequential`, sub-policies are evaluated by order of declaration; (2) `require_all`, all policies must be satisfied; and (3) `carry_over`, the outcome of the first policy is used as input for the next one (this requires `sequential` attribute to be true).

Conditions. we defined the condition hierarchy (see `Condition` metaclass) to express the determinants of the policy (e.g., a policy to be resolved before two weeks) and the policy hierarchy (`Policy` metaclass) to define the method in which to resolve the policy (e.g., absolute majority). The condition hierarchy is composed of several conditionals, and structural properties that define the policy (see, for instance, `ParticipantExclusion` metaclass). Conditionals can be evaluated before or after the policy is executed (see `pre` and `post` attributes). If they are fulfilled, the engine calculates the outcome. For instance, in a policy that requires three participants (see `MinimumParticipant` metaclass), it will not be evaluated until this condition is fulfilled (which means that by the end of the deadline, if exists, the policy will be rejected in case the condition is not fulfilled). Structural properties are considerations that will be taken in the evaluation of the policy. For instance, in a voting policy, we can define an exclusion of certain participants (see `ParticipantExclusion`) to avoid biases in their approval (e.g., excluding the one proposing the change). Similarly, a policy can have a deadline (see `Deadline` metaclass), defined by a fixed date or an offset (e.g., two weeks).

4.2 Concrete Syntax

As concrete syntax, we use a textual language following a typical block-based structure. Each instance of the metaclass is textually represented by its keyword and a block that contains the properties of the instance. Containment references are represented as nested blocks, while non-containment references use an identifier to refer to the target element. To specify this textual representation, we used an ANTLR grammar, allowing us to generate the associated

[2] `https://www.apache.org/foundation/glossary.html#LazyConsensus`

parser for our governance DSL. The complete grammar (Listing 4) of the governance language can be found in the appendix.

Listing 1 presents a simplified excerpt of the language grammar written in the EBNF format. At the root of the grammar, the `governance` rule describes the overall governance file starting with the scopes and participants, followed by the list of governance policies. Scopes are represented as a list of projects, activities and tasks (l.4) in which tasks can be nested in activities and activities nested in projects. Participants definition involve the definition of roles, individuals and profiles (l.5) with the idea of linking roles to individuals and profiles to human individuals.

Defined policies can be of two types, single or composed (l.8), reflecting what is defined in the abstract syntax. Single policies (l.9) are characterized by a policy type (l.13), allowing the parser to instantiate the correct type, and an ID to define its name (for logging and explainability of the decision process). A single policy has references to both a scope defined in the scope section and members (can be individuals or roles) defined in the participant section. It also defines the decision type (l.16), conditions (l.20) and parameters for the rule (l.23).

While composed policies still define an ID and refer to a defined scope, their composition differs. Composed policies are, as the name suggest, composed of other policies called phases. To correctly instantiate a composed policy, we first parse its `order` (l.26) representing its execution type (l.27), requirement for all policies to be approved (l.29), and the "carrying over" nature of the vote (l.30). Then, we parse its phases (l.31) as a list of policy (single or composed).

```
1  \\ Entry point of the grammar
2  governance    := scopes participants policy+ ;
3
4  scopes        := 'Scopes' ':' ( projects | activities | tasks)+ ;
5  participants  := 'Participants' ':' (roles | individuals | profiles)+ ;
6
7  \\ Policy definition
8  policy        := single | composed ;
9  single := policyType ID '{' scopeID decisonType membersID conds params '}' ;
10 composed      := 'ComposedPolicy' ID '{' scopeID order? phases '}' ;
11
12 \\ SinglePolicy elements
13 policyType := 'VotingPolicy' | 'MajorityPolicy' | 'AbsoluteMajorityPolicy' |
14            'LeaderDrivenPolicy' | 'ConsensusPolicy' | 'LazyConsensusPolicy' ;
15
16 decisonType := 'DecisionType' 'as' ('BooleanDecision'              |
17                                    'StringList'  ':' ID (',' ID)* |
18                                    'ElementList' ':' ID (',' ID)* ) ;
19
20 conds := 'Conditions' ':'  deadline? exclusion? minParticipant? vetoRight?
21                           passedTests? labelsCondition* ;
22
23 params := 'Parameters' ':' (('ratio' ':' FLOAT) | ('default' ':' policy)) ;
24
25 \\ ComposedPolicy Elements
26 order := 'Order' ':' ( orderType orderMode carryOver ? ) ;
27 orderType := 'Execution' ':' orderTypeValue ;
28 orderTypeValue := 'sequential' | 'parallel' ;
29 orderMode := 'RequireAll' ':' booleanValue ;
30 carryOver := 'CarryOver' ':' booleanValue ;
31 phases := 'Phases' '{' policy+ '}' ;
```

Listing 1: Simplified excerpt of the DSL grammar in EBNF format

```
1  Scopes:
2      Project myProject {
3          activities :
4              myActivity {
5                  tasks : myTask
6              }
7      }
8  Participants:
9      Profiles :
10         profile1 {
11             gender : male
12             race : hispanic
13         }
14     Roles : Maintainer
15     Individuals :
16         Joe {
17             vote value : 0.7
18             profile : profile1
19             role: Maintainer
20         },
21         (Agent) Mike {
22             confidence : 0.8
23         }
24 MajorityPolicy TestPolicy {
25     Scope: myTask
26     DecisionType as BooleanDecision
27     Participant list : Maintainer, Joe as Maintainer, Mike as Maintainer
28     Conditions:
29         Deadline reviewDeadline : 10 days
30         ParticipantExclusion : Mike
31         MinParticipants : 3
32     Parameters:
33         ratio : 0.4
34 }
```

Listing 2: Example of a governance policy.

Listing 2 shows a simple governance policy that requires the approval of three participants to accept a task. The policy is evaluated following the majority method (l.25) evaluated as a `BooleanDecision` (l.27), and it has a deadline of ten days (l.30), among other conditions.

Next, we declare the possible participants, in our case a `Maintainer` role and two individuals (l.28), `Joe` and `Mike` with their attributes (l.17-24). We can also define further the profile of the individuals (l.11-14). In the third block, we define the policy object. The first keyword is the policy type, in this case it is a `MajorityPolicy`, followed by its name (l.25). Inside the policy block, we declare the relationships to the scope and the participants as a list. Note that inside the participant list we can define the role an individual might have inside a policy with the `as` keyword followed by the role defined before (l.28). A policy might have certain conditions, these are defined with the `Conditions` keyword (l.29-32). At last, any parameter the policy might have, we declare it in the `Parameters` attribute (l.33), followed by the set of parameters, which in our case is the `ratio` set at 40% (l.34).

4.3 Extensibility

To allow for a wide range of application cases, we designed our DSL in an extensible way. As our current application scenarios involve policies from open-source software development, we provide a `github` extension to our metamodel, tailored by the analyzed projects.

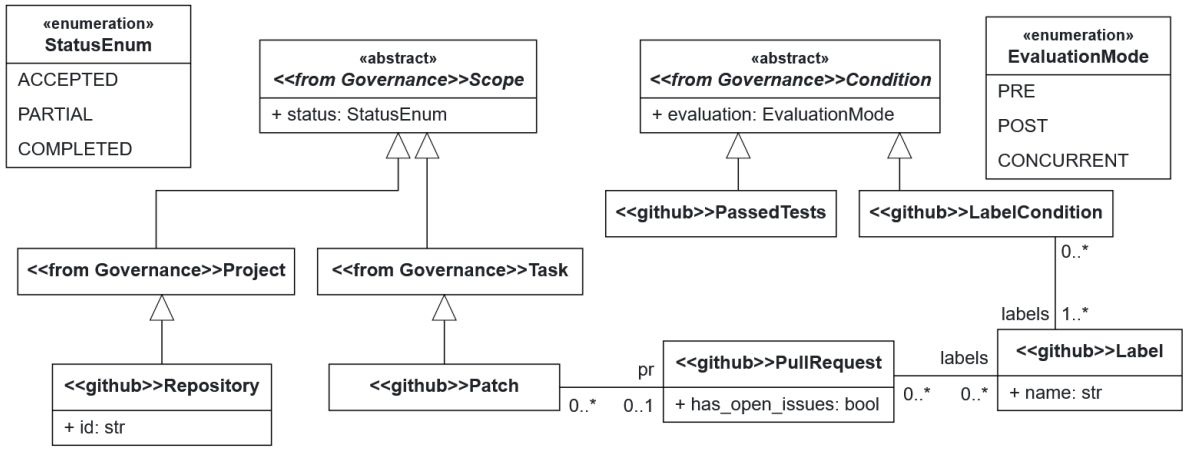


Figure 3: GitHub extension for our governance metamodel.

Figure 3 shows the extended classes, identified with the `github` stereotype. We defined a repository (see `Repository` metaclass), a patch that is composed of at least one pull request (see `Patch` and `PullRequest` metaclasses, respectively). Additionally, we defined labels (see `Label` metaclass) and specific conditions, such as a condition that require all tests defined in the pull request to be passed (see `PassedTests` metaclass), and a condition that enforces the inclusion or exclusion of particular labels (see `LabelCondition` metaclass). Note that we provided a "light" extension for GitHub elements for this representation, but a more thorough extension could be implemented leveraging on existing proposals (Ait et al., 2023).

In the future, a similar extension will be defined to address MOSAICO architecture, reifying the concepts of scope (Project/Activity/Task) and condition, based on collaboration patterns and agents KPIs respectivelly.

5 Coverage and Expressivity of the DSL

In order to evaluate the expressivity of our DSL, we identified all the governance policies from the projects analyzed in Section 2 and tried to translate them to the syntax of our DSL.

We provide the governance policies identified and their translation in our tool repository. In this report, we present one the translations. Listing 3 presents the governance policy defined in `kubernetes` repository[3]. The policy aims to explicit the decision-making on accepting pull requests.

In particular, it says that for the acceptance of a pull request (l.4 and l.13), first it is required to choose approvers and, at least, two reviewers (l.22). Once the reviewers are chosen, they have to look at the code and set a "look good to me" (lgtm). This can be understood as a vote for the acceptance of the pull request. Although it is not explicitly stated, we can assume that we require the vote of at least two reviewers (l.22), As they say that there have to be two

[3] `https://github.com/kubernetes/kubernetes`

```
1  Scopes:
2  Project K8sProject from GitHub : kubernetes/kubernetes {
3      activities : TestActivity {
4              tasks : TestTask : Pull request {
5                      Action : merge
6                  }
7          }
8  }
9  Participants:
10     Roles : Reviewers, Approvers
11     Individuals : (Agent) k8s-ci-robot
12 ComposedPolicy pr_merge {
13     Scope: TestTask
14     Order :
15         Execution : sequential
16         RequireAll : true
17         CarryOver : true
18     Phases {
19         MajorityPolicy phase_1 {
20             Participant list : Reviewers
21             Conditions:
22                 Reviewers/approvers PRassigned? pre
23                 ParticipantExclusion : PRAuthor
24                 MinParticipants : 2
25                 LabelCondition post : lgtm
26             Parameters:
27                 ratio 1.0
28         }
29         AbsoluteMajorityPolicy phase_2 {
30             Participant list : Approvers
31             Conditions:
32                 LabelCondition post: approved
33             Parameters:
34                 ratio 1.0
35         }
36         LeaderDrivenPolicy phase_3 {
37             Participant list : k8s-ci-robot
38             Conditions:
39                 LabelCondition pre: lgtm, approved
40                 LabelCondition pre not: do-not-merge/hold, needs-rebase
41                 jobs?
42         }
43     }
44 }
```

Listing 3: Governance policy for accepting pull requests in the `kubernetes` repository.

reviewers assigned. In this step, anyone but the author of the pull request can vote (l.23). Once the reviewers have voted, an automation sets the label “lgtm” to the pull request. This can be represented as a post-evaluated label condition (l.25), so that the future engine can set the label.

In the next phase, the assigned approvers must approve the pull request. The author of the pull request can be an approver. All assigned approvers must vote for the acceptance of the pull request. Thus, we represent this as an absolute majority (l.29) with a ratio of 100% (i.e., all approvers must approve the changes) – l.34. Once approved, an automation merges the pull request under certain conditions. The pull request must have the labels “lgtm” and “approved” (l.39), and cannot have the label “do-not-merge/hold” nor “needs-rebase” (l.40). Additionally,

one of the following must be true: "*there are no presubmit "prow" jobs configured for this repo*" or "*there are presubmit prow jobs configured for this repo, and they all pass after automatically being re-run one last time*" (l.41).

Some considerations must be made regarding the translation of this policy. The set of reviewers and approvers have to be extracted from the `OWNERS` file. In our case, we facilitate the participants block to define them. The kubernetes organization has an automation[4] that helps them assigning reviewers and approvers, setting the labels, and merging the pull requests. Our future decision engine should also be able to do so, but if we see a preconfigured automation (i.e., the name of the agent is part of the repository), we assume that it is already implemented. At last, they set a condition that checks for jobs.

5.1 Results and Discussion

Out of the 68% (17) of projects that reported any of the governance dimensions, we successfully translated the policies of 7 of them. The ones that could not be translated were mainly because they only provided incomplete and ambiguous information on some aspects, this suggest that even the attempt to formalize the governance can be useful for project owners that may quickly realize their descriptions are not precise enough to be useful in practice.

Based on the translated policies, we argue that our DSL is expressive enough to address real world scenarios and is flexible enough to address complex governance scenarios such as the one of Kubernetes, addressing one of the challenges described in Section 3.4. The second challenge identified was about the overhead to define and maintain such specifications. We argue that the definition overhead, while present, is allowing a clear and unambiguous definition of the policies, which was still a problem for 10 of the 17 project studied. As for the maintenance overhead, most of what will affect the policies revolve around update of scopes and change in the participant. Those two part are clearly separated from the policy definition to allow an easier maintenance.

6 Related Work

Several studies have been conducted to analyze the governance of software projects (Chulani et al., 2008; Herbsleb, 2016), particularly in the context of OSS projects (Cánovas Izquierdo et al., 2015; Keertipati et al., 2016; Pelt et al., 2021). These studies have analyzed the decision models behind different OSS projects. We leverage from their findings to propose the different decision models that can be used in our DSL. In particular, Cánovas and Cabot (Cánovas Izquierdo et al., 2015) assessed the need for explicit governance rules via a survey to developers and an analysis of eight well-known OSS projects. From their analysis, they proposed a DSL covering the main dimensions to define and enforce governance rules in OSS projects. However, their proposal did not provide support to assess the diversity of the participants involved in the decision-making process. Our DSL extends their proposal by providing a more expressive syntax to define the profile of participants involved in the decision-making process and their impact on the decision.

Within the diversity, we acknowledge the presence of agents in the decision process. In this direction, recent works (Dütting et al., 2024; Eigner et al., 2024) explore the use of different mechanisms on LLMs, to aggregate their results, such as incentives or consensus practices[5].

[4] `https://github.com/k8s-ci-robot/`

[5] `https://github.com/irthomasthomas/llm-consortium`

Similarly, the field of automated negotiation (Memon et al., 2025) has also explored the use of different decision mechanisms to reach consensus among agents. Our paper provides the identified attributes from these studies to be used in these decision mechanisms, if desired.

On the other hand, studies on the impact of diversity in decision-making processes (Jackson et al., 1995), and in particular in the context of software engineering (Feng et al., 2023; Giner-Miguelez et al., 2025; Weeraddana et al., 2023), provide an identification of the different dimensions that determine the diversity of participants in the contributions. The work of Giner et al. (Giner-Miguelez et al., 2025) provide a Software Diversity Card to represent a detailed profile of the different stakeholders of a software project. In our DSL, we provided a reduced approach to define the profile of the participants in the decision-making, but could be further extended to incorporate support for reading the diversity card.

7 Conclusion and Future Work

In this deliverable, we have presented a DSL for the explicit definition of governance policies in software projects, with a particular focus on supporting diversity among participants, including both human contributors and AI-powered agents. Our DSL extends previous work by enabling the specification of participant profiles and agent attributes, thus facilitating more equitable and transparent decision-making processes and allowing for automation in the future. Through the analysis and translation of governance policies from real-world projects, we have demonstrated the expressivity and adaptability of our approach, as well as its applicability to contemporary collaborative software development environments.

As future work for the MOSAICO project, we plan the definition of the DSL extension to align scopes and conditions with respect to the common governance policy strategies identified in deliverable 4.2. In deliverable 4.3, we will build a rule based decision engine representing the operational semantics of this DSL allowing the evaluation and automatic enforcement of the defined policies. This addition will induce changes in the DSL such as making non-mandatory the initial definition of the participants, that can be delegated to the collaboration agent – developed in deliverable 3.3 – as part of the collaboration pattern creation. This decision engine will then be enhanced as an agent-based runtime in deliverable 4.4, allowing complex interaction and discussion among the agents involved in the collaboration.

References

Ait, Adem, Javier Luis Cánovas Izquierdo, and Jordi Cabot (2023), "A Tool for the Definition and Deployment of Platform-Independent Bots on Open Source Projects", *in*: *Int. Conf. on Software Language Engineering*, pp. 214–219. [Cited on page 19]

Arrieta, Alejandro Barredo, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera (2020), "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI", *in*: *Inf. Fusion* 58, pp. 82–115. [Cited on page 11]

Bjørn, Pernille, Maria Menendez-Blanco, and Valeria Borsotti (2023), *Diversity in computer science: Design artefacts for equity and inclusion*, Springer Nature. [Cited on pages 9, 10]

Brambilla, Marco, Jordi Cabot, and Manuel Wimmer (2017), *Model-Driven Software Engineering in Practice, Second Edition*, Synthesis Lectures on Software Engineering, Morgan & Claypool Publishers. [Cited on page 15]

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 pages 9, 11, 21]

— (2023), "For a More Transparent Governance of Open Source", *in*: *Commun. ACM* 66.8, pp. 32–34. [Cited on pages 9, 11, 12]

Catolino, Gemma, Fabio Palomba, Damian A. Tamburri, Alexander Serebrenik, and Filomena Ferrucci (2019), "Gender diversity and women in software teams: how do they affect community smells?", *in*: *Int. Conf. on Software Engineering: Software Engineering in Society*, pp. 11–20. [Cited on page 10]

Chang, Yupeng, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie (2024), "A Survey on Evaluation of Large Language Models", *in*: *Trans. Intell. Syst. Technol.* 15.3, 39:1–39:45. [Cited on page 8]

Chesterman, Simon, Yuting Gao, Jungpil Hahn, and Valerie Sticher (2024), "The Evolution of AI Governance", *in*: *Computer* 57.9, pp. 80–92. [Cited on pages 9, 11]

Chulani, Sunita, Clay Williams, and Avi Yaeli (2008), "Software development governance and its concerns", *in*: *Int. workshop on Software development governance*, pp. 3–6. [Cited on page 21]

Cobos, Sergio and Javier Luis Cánovas Izquierdo (2025), "A Bot-based Approach to Manage Codes of Conduct in Open-Source Projects", *in*: *Int. Conf. on Software Engineering: Software Engineering in Society*, pp. 3–12. [Cited on page 11]

Decan, Alexandre, Tom Mens, Pooya Rostami Mazrae, and Mehdi Golzadeh (2022), "On the Use of GitHub Actions in Software Development Repositories", *in*: *Int. Conf. on Software Maintenance and Evolution*, pp. 235–245. [Cited on page 11]

Dutta, Riya, Diego Elias Costa, Emad Shihab, and Tanja Tajmel (2023), "Diversity Awareness in Software Engineering Participant Research", *in*: *Int. Conf. on Software Engineering: Software Engineering in Society*, pp. 120–131. [Cited on page 10]

Dütting, Paul, Vahab Mirrokni, Renato Paes Leme, Haifeng Xu, and Song Zuo (2024), "Mechanism Design for Large Language Models", *in*: *Web Conference*, pp. 144–155. [Cited on page 21]

Eigner, Eva and Thorsten Händler (2024), "Determinants of LLM-assisted Decision-Making", *in*: *CoRR* abs/2402.17385. [Cited on page 21]

Feng, Zixuan, Mariam Guizani, Marco Aurélio Gerosa, and Anita Sarma (2023), "The State of Diversity and Inclusion in Apache: A Pulse Check", *in*: *Int. Conf. on Cooperative and Human Aspects of Software Engineering*, pp. 150–160. [Cited on page 22]

Fowler, Martin (2011), *Domain-Specific Languages*, Addison-Wesley. [Cited on pages 10, 14]

Giner-Miguelez, Joan, Sergio Morales, Sergio Cobos, Javier Luis Cánovas Izquierdo, Robert Clarisó, and Jordi Cabot (2025), "The Software Diversity Card: A Framework for Reporting Diversity in Software Projects", *in*: *CoRR* abs/2503.05470. [Cited on pages 10, 22]

Gunatilake, Hashini, John 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.*, p. 107489. [Cited on page 9]

Guo, Taicheng, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang (2024), "Large Language Model Based Multi-agents: A Survey of Progress and Challenges", *in*: *Int. Joint Conf. on Artificial Intelligence*, pp. 8048–8057. [Cited on pages 8, 9, 11]

Herbsleb, James D. (2016), "Building a socio-technical theory of coordination: why and how (outstanding research award)", *in*: *Int. Symposium on Foundations of Software Engineering*, pp. 2–10. [Cited on page 21]

Jackson, Susan, Karen May, and Kristina Whitney (Jan. 1995), "Understanding the Dynamics of Diversity in Decision Making Teams", *in*: Jossey-Bass, pp. 7–261. [Cited on page 22]

Keertipati, Smitha, Sherlock A. Licorish, and Bastin Tony Roy Savarimuthu (2016), "Exploring decision-making processes in Python", *in*: *Int. Conf. on Evaluation and Assessment in Software Engineering*, 43:1–43:10. [Cited on page 21]

Kinsman, Timothy, Mairieli Santos Wessel, Marco Aurélio Gerosa, and Christoph Treude (2021), "How Do Software Developers Use GitHub Actions to Automate Their Workflows?", *in*: *Int. Conf. on Mining Software Repositories*, pp. 420–431. [Cited on page 11]

Mao, Rui, Guanyi Chen, Xulang Zhang, Frank Guerin, and Erik Cambria (2024), "GPTEval: A Survey on Assessments of ChatGPT and GPT-4", *in*: *Joint Int. Conf. on Computational Linguistics, Language Resources and Evaluation*, pp. 7844–7866. [Cited on page 8]

Memon, Mashal Afzal, Gino Luca Scoccia, and Marco Autili (2025), "A systematic mapping study on automated negotiation for autonomous intelligent systems", *in*: *Autom. Softw. Eng.* 32.1, p. 44. [Cited on page 22]

Pelt, Rowan van, Slinger Jansen, Djuri Baars, and Sietse Overbeek (2021), "Defining Blockchain Governance: A Framework for Analysis and Comparison", *in*: *Inf. Syst. Manag.* 38.1, pp. 21–41. [Cited on page 21]

Rahwan, Iyad, Manuel Cebrián, Nick Obradovich, Josh C. Bongard, Jean-François Bonnefon, Cynthia Breazeal, Jacob W. Crandall, Nicholas A. Christakis, Iain D. Couzin, Matthew O. Jackson, Nicholas R. Jennings, Ece Kamar, Isabel M. Kloumann, Hugo Larochelle, David Lazer, Richard McElreath, Alan Mislove, David C. Parkes, Alex 'Sandy' Pentland, Margaret E. Roberts, Azim Shariff, Joshua B. Tenenbaum, and Michael P. Wellman (2019), "Machine behaviour", *in*: *Nat.* 568.7753, pp. 477–486. [Cited on page 11]

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 pages 9, 10]

Rodríguez-Pérez, Gema, Reza Nadri, and Meiyappan Nagappan (2021), "Perceived diversity in software engineering: a systematic literature review", *in*: *Empir. Softw. Eng.* 26.5, p. 102. [Cited on pages 9, 10]

Shneiderman, Ben (2020), “Human-Centered Artificial Intelligence: Reliable, Safe & Trustworthy”, *in*: *Int. J. Hum. Comput. Interact.* 36.6, pp. 495–504. [Cited on page 11]

Vasilescu, Bogdan, Daryl Posnett, Baishakhi Ray, Mark G. J. van den Brand, Alexander Serebrenik, Premkumar T. Devanbu, and Vladimir Filkov (2015), “Gender and Tenure Diversity in GitHub Teams”, *in*: *Conf. on Human Factors in Computing Systems*, pp. 3789–3798. [Cited on page 10]

Wasowski, Andrzej and Thorsten Berger (2023), *Domain-Specific Languages - Effective Modeling, Automation, and Reuse*, Springer. [Cited on pages 10, 14]

Weeraddana, Nimmi Rashinika, Xiaoyan Xu, Mahmoud Alfadel, Shane McIntosh, and Meiyappan Nagappan (2023), “An empirical comparison of ethnic and gender diversity of DevOps and non-DevOps contributions to open-source projects”, *in*: *Empir. Softw. Eng.* 28.6, p. 150. [Cited on page 22]

Wessel, Mairieli, Joseph Vargovich, Marco Aurélio Gerosa, and Christoph Treude (2023), “GitHub Actions: The Impact on the Pull Request Process”, *in*: *Empir. Softw. Eng.* 28.6, p. 131. [Cited on page 11]

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 pages 9, 10]

Wooldridge, Michael J. and Nicholas R. Jennings (1995), “Intelligent agents: theory and practice”, *in*: *Knowl. Eng. Rev.* 10.2, pp. 115–152. [Cited on page 8]

Xi, Zhiheng, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. (2025), “The rise and potential of large language model based agents: A survey”, *in*: *Science China Information Sciences* 68.2, p. 121101. [Cited on page 11]

Yang, Yang, Tanya Y Tian, Teresa K Woodruff, Benjamin F Jones, and Brian Uzzi (2022a), “Gender-diverse teams produce more novel and higher-impact scientific ideas”, *in*: *Proceedings of the National Academy of Sciences* 119.36, e2200841119. [Cited on page 9]

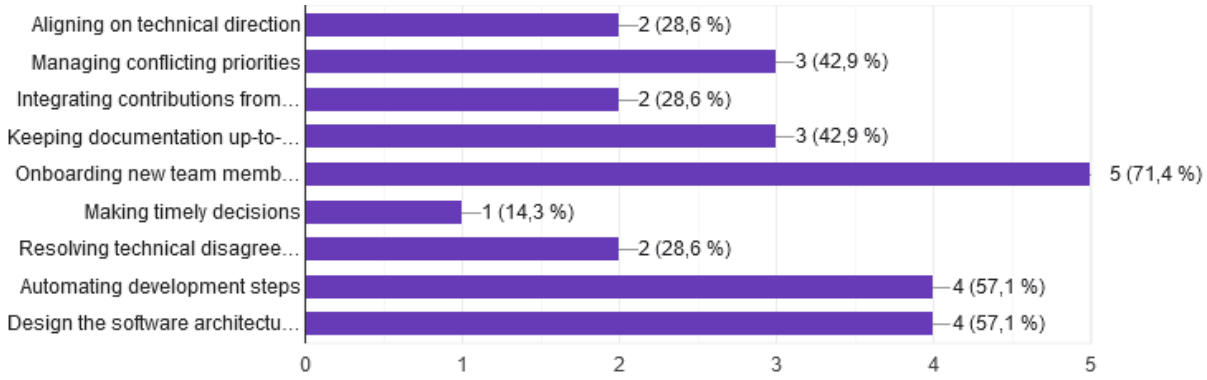
— (2022b), “Gender-diverse teams produce more novel and higher-impact scientific ideas”, *in*: *Proceedings of the National Academy of Sciences* 119.36, e2200841119. [Cited on page 10]

Annexes

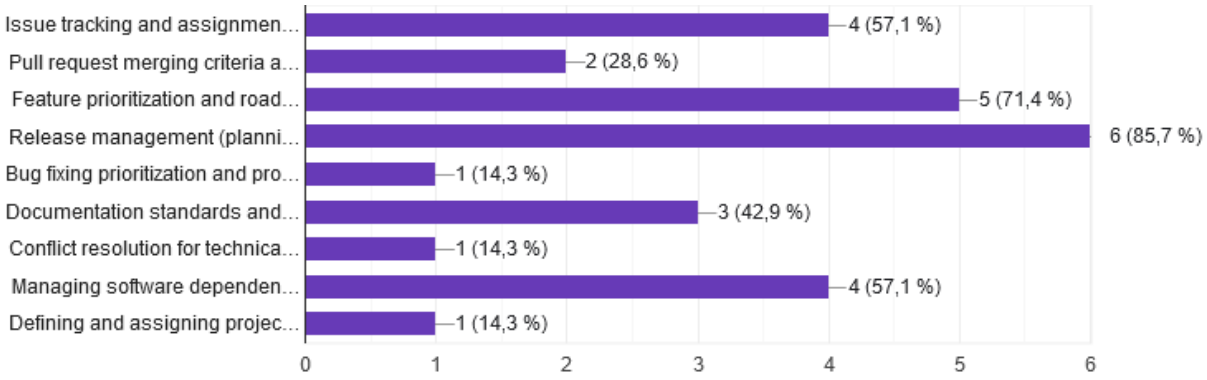
A Results of the survey

A.1 Understanding Project Dynamics and Challenges

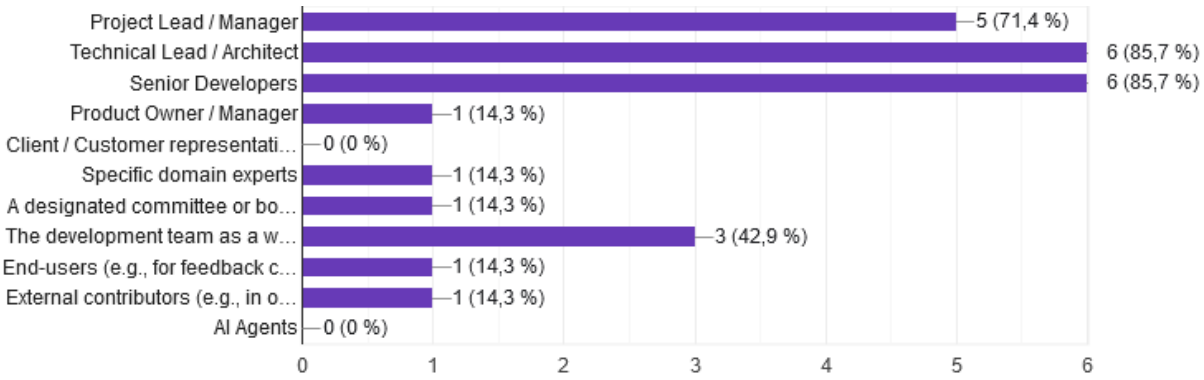
Question 1: What are some common challenging situations or recurring complex aspects you experience in your software development lifecycle?



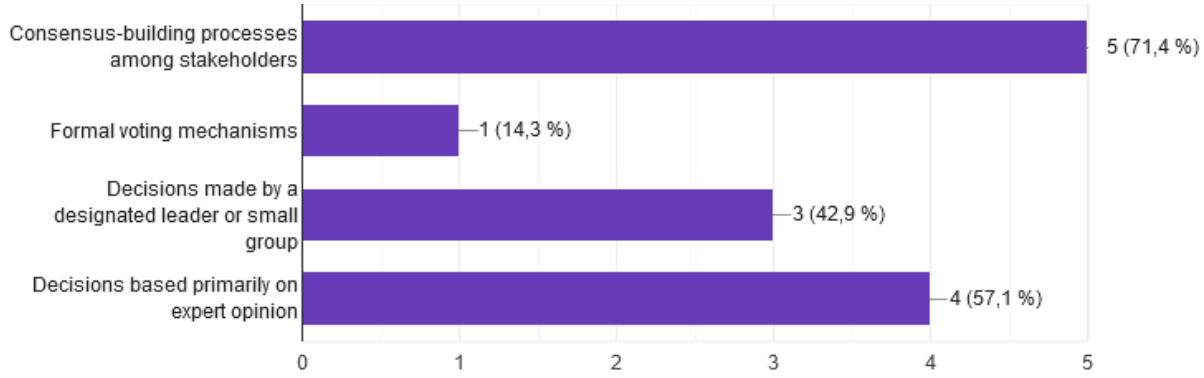
Question 2: Which of the following aspects of your software development process require explicit written policies (i.e., governance processes) in your projects to make sure a good decision is reached?



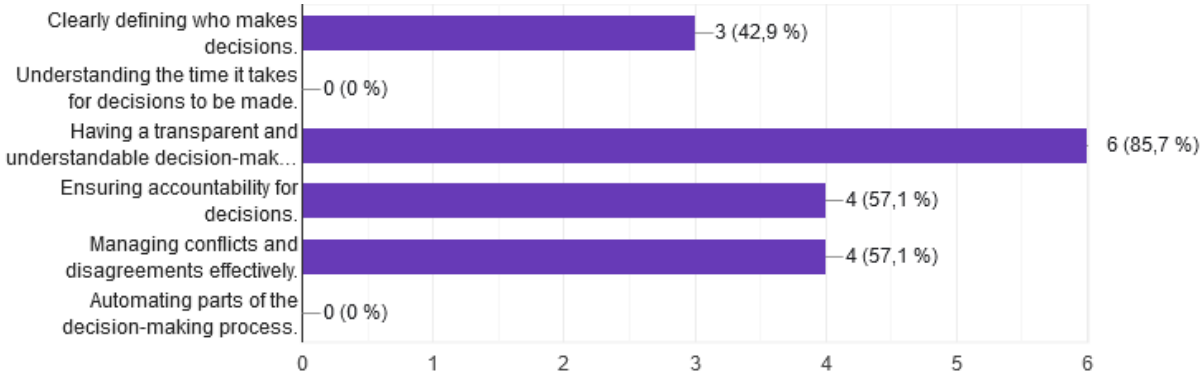
Question 3: Who is typically involved in these decision-making processes? (in any of the previous processes)



Question 4: What forms of participation or decision-making mechanisms are commonly used for the decision-making process?

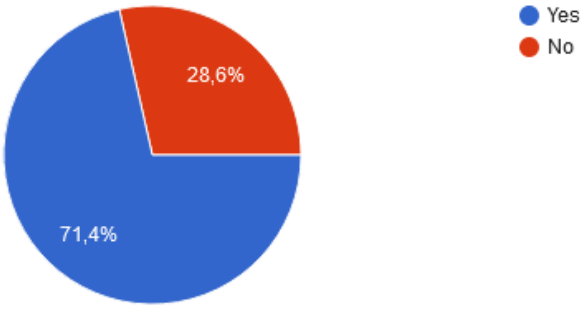


Question 5: What aspects are most important to you when it comes to decision-making in software projects?

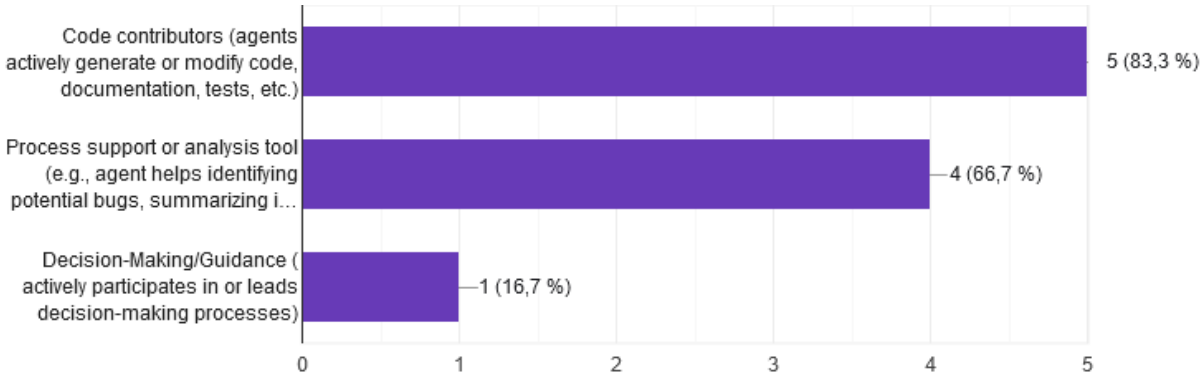


A.2 AI in Project Processes

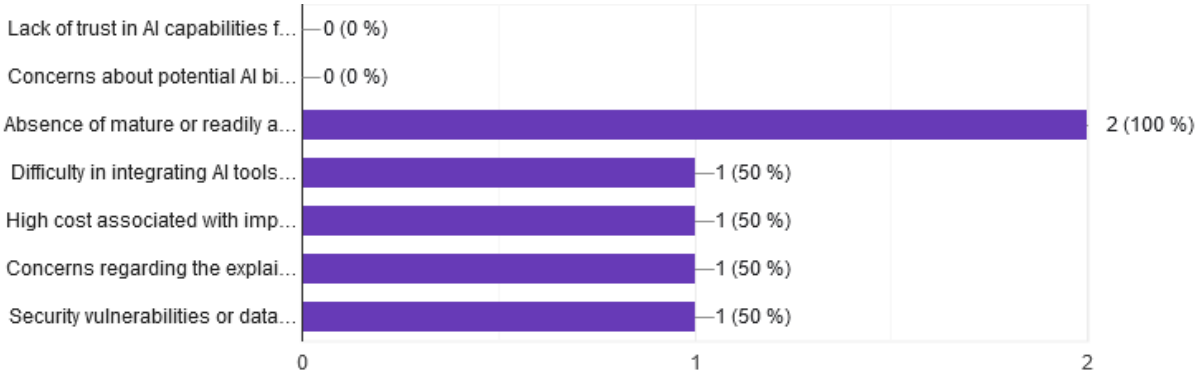
Question 6: Does your company currently utilize AI agents in any part of your software development processes?



Question 7: If so, how would you characterize the role of these AI agents?

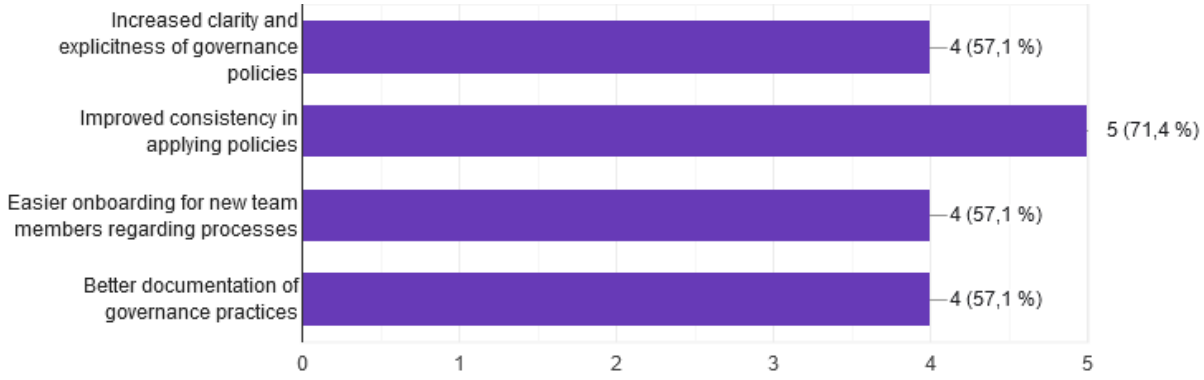


Question 8: If you do not currently use AI agents in your development decision-making or active processes, what are the primary reasons?

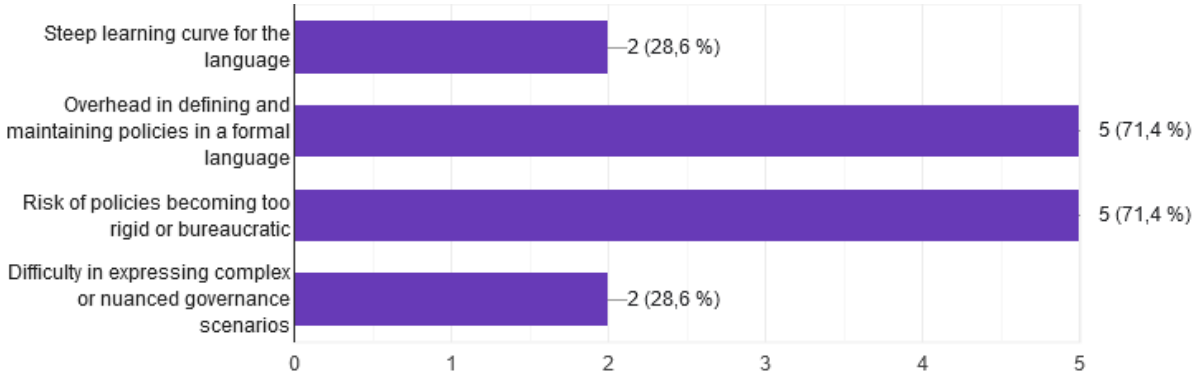


A.3 Language and Automation for Governance

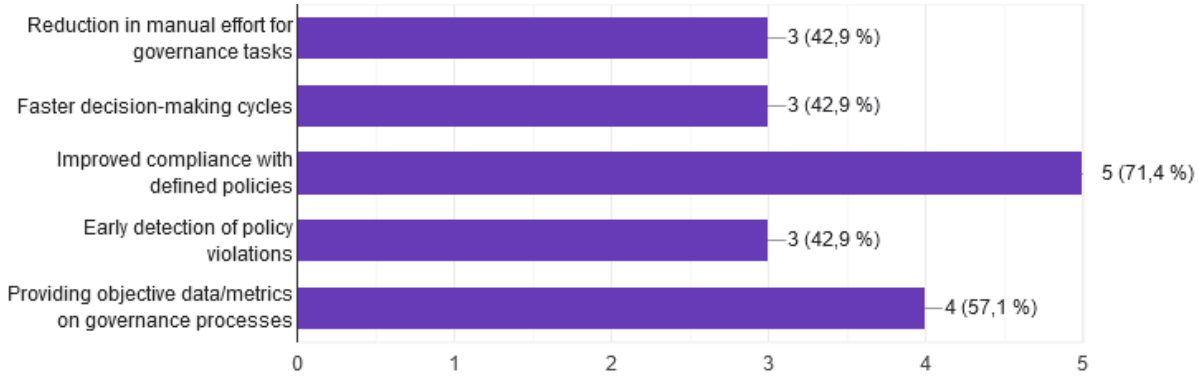
Question 9: Considering a specialized language to precisely define governance policies for software projects. What potential benefits do you see?



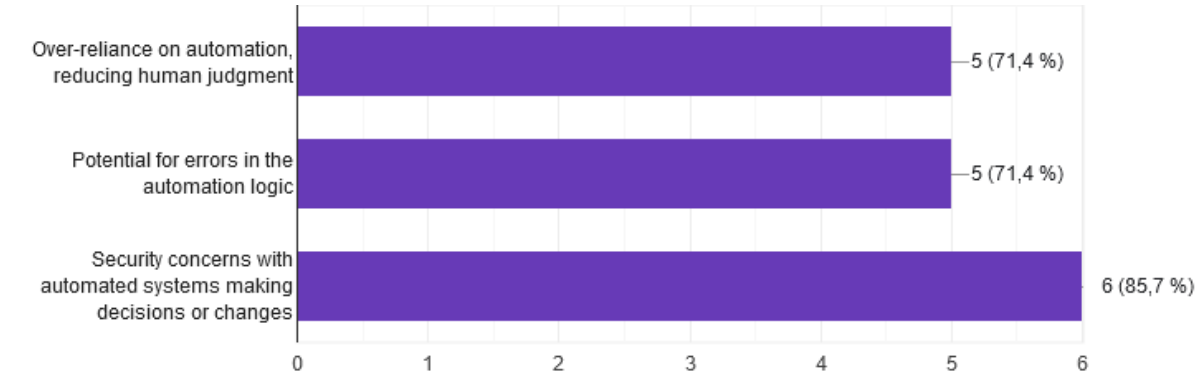
Question 10: What potential drawbacks or challenges do you foresee?



Question 11: Considering automation tooling that could interpret and help enact governance policies (potentially defined in such a language), what potential benefits do you see?



Question 12: What potential drawbacks or challenges do you foresee?



B Grammar of the Governance DSL

Listing 4: Complete ANTLR grammar of the Governance DSL.

```
grammar govdsl;

governance      : (scopes participants policy+) EOF ;
policy          : (topLevelSP | topLevelCP) ;

// Top-level policies (with scope)
topLevelSP      : policyType ID '{'
                   scope decisionType policyMembers? conditions? parameters?
                  '}' ;
topLevelCP      : 'ComposedPolicy' ID '{' scope order? phases '}' ;

// Nested policies (no scope)
nestedSP        : policyType ID '{'
                      decisionType policyMembers? conditions? parameters?
                  '}' ;
nestedCP        : 'ComposedPolicy' ID '{' order? phases '}' ;

policyType : 'VotingPolicy' | 'MajorityPolicy' | 'AbsoluteMajorityPolicy' |
          'LeaderDrivenPolicy' | 'ConsensusPolicy' | 'LazyConsensusPolicy' ;

// Scope definition
scopes          : 'Scopes' ':' ( projects | activities | tasks)+ ;
scope           : 'Scope' ':' ID ; // reference from policy
projects        : 'Projects' ':' project+ ;
project         : ID ('from' platform ':' repoID)?
                     ('{' 'Activities' ':' activity+ '}')? ;
platform        : 'GitHub' ;
repoID          : ID ('/' ID)? ; // owner/repo
activities      : 'Activities' ':' activity+ ;
activity        : ID ('{' 'Tasks' ':' task+ '}')? ;
tasks           : 'Tasks' ':' task+ ;
task            : ID (':' taskType)? ('{' taskContent '}')? ;
taskType        : 'Issue' | 'Pull request' | 'All' ;
taskContent     : status | action | actionWithLabels ;
actionWithLabels: action labels ;
status          : 'Status' ':' statusEnum ;
statusEnum      : 'completed' | 'accepted' | 'partial' ;
action          : 'Action' ':' actionEnum ;
actionEnum      : 'merge' | 'review' | 'release' ;
labels          : 'Labels' ':' ID (',' ID)* ;

// Decision type
decisionType    : 'DecisionType' 'as'
                                  (booleanDecision | stringList | elementList) ;
booleanDecision : 'BooleanDecision' ;
stringList      : 'StringList' ':' ID (',' ID)* ;
elementList     : 'ElementList' ':' ID (',' ID)* ;

// Participants group
participants    : 'Participants' ':' (roles | individuals | profiles)+ ;
roles           : 'Roles' ':' ID (',' ID)* ;
individuals     : 'Individuals' ':' individualEntry (',' individualEntry)* ;
individualEntry : individual | agent ;
individual      : ID
                ('{' voteValue? ','? withProfile? ','? withRole? '}')? ;
```

```
voteValue       : 'vote value' ':' FLOAT ;
withProfile     : 'profile' ':' ID ;
withRole        : 'role' ':' ID ;
agent           : '(Agent)' ID ('{' voteValue? ','? confidence? ','?
                   autonomyLevel? ','? explainability? ','? withRole? '}')? ;
confidence      : 'confidence' ':' FLOAT ;
autonomyLevel   : 'autonomy level' ':' FLOAT ;
explainability  : 'explainability' ':' FLOAT ;
profiles        : 'Profiles' ':' profile (',' profile)* ;
profile         : ID '{' (gender (',' race)? | race (',' gender)?) '}';
gender          : 'gender' ':' ID ;
race            : 'race' ':' ID ;
policyMembers   : 'Participant list' ':' partID (',' partID)* ;
partID          : ID hasRole? ;
hasRole         : 'as' ID ;

// Conditions group
conditions      : 'Conditions' ':'  deadline? exclusion? minParticipant?
                                    vetoRight? passedTests? labelsCondition* ;
deadline        : 'Deadline' deadlineID ':'
                                       ( offset | date | (offset ',' date) ) ;
offset          : SIGNED_INT timeUnit ;
deadlineID      : ID ;
timeUnit        : 'days' | 'weeks' | 'months' | 'years' ;
date            : SIGNED_INT '/' SIGNED_INT '/' SIGNED_INT ; // DD/MM/YYYY
exclusion       : 'ParticipantExclusion' ':' ID (',' ID)* ;
minParticipant  : 'MinParticipants' ':' SIGNED_INT ;
vetoRight       : 'VetoRight' ':' ID (',' ID)* ;
passedTests     : 'PassedTests' evaluationMode? ':' booleanValue ;
evaluationMode  : ( 'pre' | 'post' | 'concurrent' ) ;
labelsCondition : 'LabelCondition' evaluationMode? include?':' ID (',' ID)* ;
include         : 'include' | 'not';

// Parameters group
parameters      : 'Parameters' ':' (votParams | default) ;
votParams       :  ratio ;
ratio           : 'ratio' ':' FLOAT ;
default         : 'default' ':' nestedPolicy ;

// Phased policy
order           : 'Order' ':' ( orderType orderMode carryOver? ) ;
orderType       : 'Execution' ':' orderTypeValue ;
orderTypeValue  : 'sequential' | 'parallel' ;
orderMode       : 'RequireAll' ':' booleanValue ;
carryOver       : 'CarryOver' ':' booleanValue ;
booleanValue    : 'true' | 'false' ;
phases          : 'Phases' '{' nestedPolicy+ '}' ;
nestedPolicy    : nestedSP | nestedCP ;

// Lexer rules
ID              : [a-zA-Z_][a-zA-Z0-9_-]* ;
SIGNED_INT      : '-'? [0-9]+ ;
FLOAT           : '-'? [0-9]+ '.' [0-9]+ ;
WS              : (' ' | '\t' | '\r'? '\n')+ -> skip ;
```