Specification and Application of Governance Rules in Software Development[*]

Javier Luis Cánovas Izquierdo, Jordi Cabot

AtlanMod, École des Mines de Nantes – INRIA – LINA, Nantes, France
{javier.canovas,jordi.cabot}@inria.fr

Abstract. Any software development project is decomposed into a series of tasks that must be properly prioritized and managed for the success of the project. Unfortunately, the rules governing the development process (e.g., describing who can implement a task or deciding which tasks should be completed first) are usually implicit or appear scattered in the project documentation or tools (e.g., tracking-systems, forums, etc.). In this paper we propose to enable the explicit definition of governance rules for development tasks along with complementary support for their application. Making explicit governance rules will provide several important benefits, including improvements in the transparency and sustainability of the project. Furthermore, the support for the rule application will promote traceability (why a decision was made) and the automation of the governance process (i.e., liberating developers from applying the rules manually and avoiding potential inconsistencies). We created a domain-specific language to easily define governance rules, which is illustrated with real open source projects, whereas their application is supported by a tool implemented on top of Mylyn, a project management plug-in for Eclipse, which supports most tracking-systems, thus allowing the use of our approach in most development projects.

Keywords: Decision making, Governance, Collaborative Software Development

1 Introduction

The development of large-scale software has to cope with a huge number of tasks consisting of either implementing new issues or fixing bugs [1]. Thus, effective and precise prioritization of these tasks is key for the success of the project. For this purpose, each project defines and applies its own set of governance rules, which also define how to contribute in the project and how decisions are made (e.g., which issues have to be implemented or when the next milestone will be). Thus, a governance model enables the coordination of developers in order to advance the project. According to Conway's Law [2], this coordination has a direct impact on the product being built [3].

Governance models makes particularly sense in the context of Open Source System (OSS) development projects [4], where external developers can participate actively and need to know how they can contribute. However, these projects usually have a methodology very loosely defined (if any) [5] and it is hard to find an explicit system-level

[*] This work has been supported by the European Commission under the ICT Policy Support Programme, grant no. 317859.

design, a project plan, schedule or list of deliverables [6]. Thus, despite their importance, in practice governance models are hardly ever explicitly defined. This hampers the integration of new developers[1] in the team who must spend some time understanding the "culture" of the project [7].

The governance model being missing, mechanisms to facilitate the communication and the assignment of work are considered crucial for the success of the development [8, 9]. Tracking systems (i.e., bug-traking such as Bugzilla[2] and issue-tracking systems such as Mantis[3]) are broadly used to manage the tasks to be performed. Other collaborative tools such as mailing-lists or forums are also used to coordinate the developers involved in the project. While these tools provide a convenient compartmentalization of work and effective means of communication, they fall short in providing adequate support for specifying and applying a governance model (e.g., supporting the voting of tasks, easy tracking of decisions made in the project, etc.).

Therefore, the explicit definition of a governance model would have several benefits, including improvements in the transparency and sustainability [4]. Moreover, we believe the support for not only the definition but also the application of the governance model would also promote traceability (being able to track why a decision was made and who decided it) and the automation of the governance process (e.g., liberating developers from having to be aware and follow the rules manually, minimizing the risk of inconsitent behaviour in the evolution of the project).

This paper tackles both aspects by providing a new domain-specific language (DSL) to let project managers easily define the governance rules of their projects and a collaborative infrastructure to be able to "execute" those rules as part of the project evolution, though the use of the latter is up to the project managers. The DSL is defined based on our analysis of the (implicit) rules used in the governance of a set of OSS development projects. The collaborative infrastructure has been implemented on top of Mylyn, a project management plugin for the Eclipse platform with connectors for most popular tracking systems. Thanks to these connectors our approach can be used together with the existing tracking systems already in place in software development projects.

The rest of the paper is structured as follows. In the next section, we report on the lack of explicit support for defining governance models in current OSS development projects and the various governance strategies used therein. We then present our proposal consisting in a DSL to describe these rules, the infrastructure needed to apply them and an illustrative case study. Finally, we describe our tool support, compare our approach with related work and draw some final conclusions and future work.

2 How Software Projects are Governed

Many projects (specially most large open-source projects) are organized in terms of tasks (either issues or bugs) that are solved by means of patches (implementing new features in response to the issues or correcting the bugs) which at some point can be

[1] E.g., searching for *how to contribute* in *Stackoverflow* returns more than four thousand results (i.e., $\sim$8% of the total number of questions)

[2] http://www.bugzilla.org/

[3] http://www.mantisbt.org/

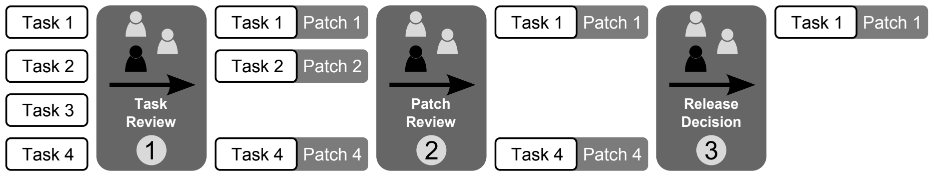


Fig. 1. Main workflow followed by a task and the main points where decisions are made.

selected to be part of the next software release. Figure 1 summarizes this workflow. The workflow includes three main decision points, namely: (1) task review, (2) patch review and (3) release insertion. These decisions (plus the right to be able to contribute and how to do so) are taken based on the governance rules defined for the project. In this section we explore the rules employed by a set of popular OSS projects and report on a lack of explicit definition and application of such rules as motivation for our approach.

We have studied nine OSS projects where the participation is open to anyone willing to contribute, namely: Android[4], GNOME[5], Apache (web server) [6], Mozilla[7], Python[8], Moodle[9], EMF[10], MoDisco[11]. Table 1 shows a summary of the data we gathered. Next we describe the results in more detail.

Organization. The organization followed by a project summarizes its main development philosophy. The great majority of analyzed projects follow a hierarchical scheme, thus meaning that there exist several hierarchical levels in the groups of users collaborating in the development (e.g., leaders, contributors, users, etc.). Thus, there are several roles (see the corresponding feature below) which a user can belong to. A good example is the Android project, where each role has assigned a particular function in the process (e.g., *approvers* approve tasks, *verifiers* review patches, etc.) supporting the project leader. On the other hand, Apache differs from the others by using a meritocracy organization where users can gain merits as they contribute to the project.

Coordination. This feature includes the main tools used to help users to collaborate during the development process. As can be seen, mailing-list and forums are the main tools used. Note that even though these tools are useful to keep in touch with the rest of developers, they are normally used to apply some governance rules as well, which goes beyond the original purpose of those tools. This is the case of Apache, which uses a mailing-list to vote which issues/bugs should be implemented/fixed next.

Tracking system. The managament of tasks is performed by tracking systems, Bugzilla being the most popular one. Interestingly enough, tracking systems are also used as a forum-like tool to discuss possibles tasks which are not mature enough to be considered

[4] http://www.android.com
[5] https://www.gnome.org
[6] https://www.apache.org
[7] https://www.mozilla.org
[8] https://www.python.org
[9] https://moodle.org
[10] https://www.eclipse.org/emf
[11] https://www.eclipse.org/modisco

Project	Organization	Coordination	Tracking system	Task Review	Patch Review	Release Decision	Rules Def. / App.	Roles
Android	Hierarchy	Forum	Git Gerrit	Yes (Approver)	Yes (Verifier)	Yes (Project Lead)	Documentation / Track system	Contributor Developer Verifier Approver Project Lead
GNOME	Hierarchy	IRC mailing-list	Bugzilla	Yes	Yes (Bug Squad)	Yes	Documentation / Track System	Contributor Mentor
Apache Web Server	Meritocracy	Mailing-list	Bugzilla	Yes (Voting)	Yes (Voting)	Yes (Voting)	Documentation / Mailing-list	User Developer Committer PMC Member PMC Chair
Mozilla	Hierarchy	Forum Mailing-list IRC	Bugzilla	Yes (Module Owner)	Yes (Module Owner & Super-reviewers)	Yes (Designated Group)	Documentation / Track System & Mailing-list	User Committer Module Owner Super-reviewer
Python	Hierarchy	Mailing-list IRC Blogs	Mercurial Roundup	No	Yes (Reviewer)	Yes (Core Developer)	Documentation / Track System	Contributor Reviewer Core Developer
Moodle	Hierarchy	Forum	Moodle tracker	Yes (Component leads)	Yes (Developers)	Yes (Component Leads)	Documentation / Track System	Users Developers Component Leads Integrators Testers Manteiners
EMF	Hierarchy	Forum	Bugzilla	Yes (Committer)	Yes (Committer)	Yes (Project Leader)	Documentation / Track System	User Contributor Committer Project Leader
MoDisco	Hierarchy	Forum	Bugzilla	Yes (Committer)	Yes (Committer)	Yes (Committers)	Documentation / Track System	User Contributor Committer Project Leader

Table 1. Comparison of how OSS systems are governed.

as real issues. For instance, in the GNOME project, some tasks[12] in Bugzilla are used to openly discuss possible changes in the development strategy.

Task review. Once a task (i.e., issue/bug) is notified, it can be reviewed to be accepted or rejected. Except for Python, where all the task are accepted and only reviewed if they include the corresponding patch, the rest of the projects have a decision process to accept or reject the task proposal. In general, the decision process is made by either the leader (e.g., component leaders in Moodle) or by unanimous agreement if there are several leaders (e.g., in big Eclipse-based projects such as EMF).

Patch review. Tasks have attached the corresponding patch implementing the improvement (if it is an issue) or the fix (if it is a bug). Before incorporating the patch into the product, it is possible to perform a revision and test to check the quality of the patch. All the analyzed projects include a review process which analyzes the patch and eventually decides its acceptance or rejection.

Release Decision. As the tasks and the corresponding patches are accepted, new software product releases have to be published. In the vast majority of analyzed projects, the decision of selecting when to perform the release and which tasks should be incorporated normally is taken by the product/component leader or by unanimous agreement when there are several leaders. The only exception is the decision process for Apache, where a ballot takes place.

[12] http://felipec.wordpress.com/2011/09/23/no-gnome-doesnt-want-user-feedback-how-i-argued-in-favor-of-voting-in-bugzilla-and-got-banned-as-a-result

Governance rule definition and application. This feature shows how governance rules are defined and applied. In general, none of the projects explictly defines these rules, and the result of their application is scattered througout the management systems (i.e., track systems or mailing-lists).

Roles. The users collaborating in the development are classified according to a set of roles. All the analyzed projects include both the role of developer (also known as contributor or commiter), who can commit changes into the source code repository of the project and the corresponding role for leaders (also known as owners or chairs).

In summary, most projects are hierarchical where a group of contributors are driven by a leader (or a set of leaders) who normally decides which tasks should be completed first. Patches are normally reviewed and tested by contributors while the product release is decided by a leader group as well. Thus, the resulting governance rules are mainly controlled by the group of leaders, who can decide how the software product should evolve. When there is a group of leaders, the decision is normally made by unanimous agreement and normally using tools such as email, mailing-lists or forums.

The development of Apache web server is the main exception. The project is governed by a vote-based strategy where all the contributors can decide which tasks should be accepted and which ones should be included into the final release. It is important to note that although anyone can vote, only the votes from contributors are binding, the rest are helpful to see the general opinion of the community. The project also establishes how the voting should be performed. Thus, if the task involves changes in the source code, the voting should unanimously agree to make the change and at least three votes must be casted. Otherwise, if the change does not involve code, a majority of positives votes (and at least three) are required. Moreover, any negative vote must include the corresponding rationale, which can therefore help to solve the disagreement.

Interestingly enough, in all the analyzed projects it was not trivial for us to discover the governance rules being applied since the available information was scarce and normally scattered among the documentation of the project. In fact, some projects such as GNOME recommend to be patient since it can take a long time to become a contributor. Potential contributors are therefore required to observe existing mailing lists, conversations on IRC, etc. to discover the way of working in the project.

Moreover, the result of the application of these rules is directly updated in the tracking systems, where normally there is no tracking information that helps to clarify later on why that decision was taken nor the possible discussion threads (maybe taken place outside the tracking systems, e.g., by chat or email as it is the case for smaller projects such as MoDisco) among the leaders that lead to that decision.

Clearly, making explicit the governance rules followed in a project could help the developer community to understand and apply those rules as part of their daily development activities. Next sections describe our proposal to tackle this problem.

3 Overview of Our Approach

Our proposal consists in two main components. First, to make the governance rules explicit, we propose to use a Domain Specific Language (DSL) including concrete constructs to define which decision rules must be followed to act on a set of tasks/patches.

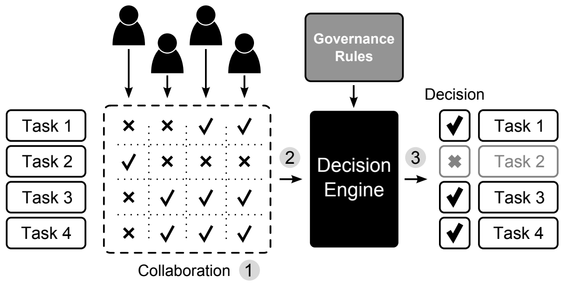


Fig. 2. Decision process proposed.

Secondly, we propose to enable the "execution" of the DSL specifications so that, beyond improving the understanding of the project internal organization, they can also be used to support and guide the collaboration among the project participants.

We believe our DSL would be useful in all the decision points identified before. Figure 2 illustrates how the application of our DSL would change the group dynamics in a particular decision point. As can be seen, given a set of tasks to be discussed in a decision point (i.e., either to accept them, accept a patch for them or include them in a release), users can vote for/against them (step 1), a decision engine analyzes the votes according to the governances rules defined (e.g., total agreement, simple majority, etc.) (step 2). As a result, the status of the tasks is changed based on the decisions taken by the engine (step 3).

In what follows we define the DSL and the infrastructure required to enable its use in practice in more detail.

3.1 A DSL to Define Governance Rules

A DSL is defined by three main elements [10]: abstract syntax, concrete syntax and semantics. The abstract syntax defines the main concepts and their relationships, and also include well-formedness rules constraining how the concepts and relationships have to be used. The concrete syntax defines the language notation (textual, graphical or hybrid) and a translational approach is normally used to provide semantics.

Our DSL has been defined in the context of Model-Driven Engineering, thus we used metamodelling techniques [11] to define the abstract syntax. A metamodel formally describes the structure of the models which are used to represent an aspect of a system at some abstraction level. In our case, we have defined a metamodel to represent the concepts and relationships needed to define governance rules. A model is an instance of a metamodel, and the *conforms to* term is normally used to express this *instance of* relationship between models and metamodels. Thus, models of our language conform to the language abstract syntax metamodel and represent specific instances of governance rules. As concrete syntax, we have opted for a textual language following a typical block-based structure. Thus, each instance of a metaclass is represented by its keyword and a block which contains the name and value of its attributes. Containment references are represented as nested blocks while non-containment references use an identifier to refer the referred element. In the following we will show some examples to illustrate the syntax.

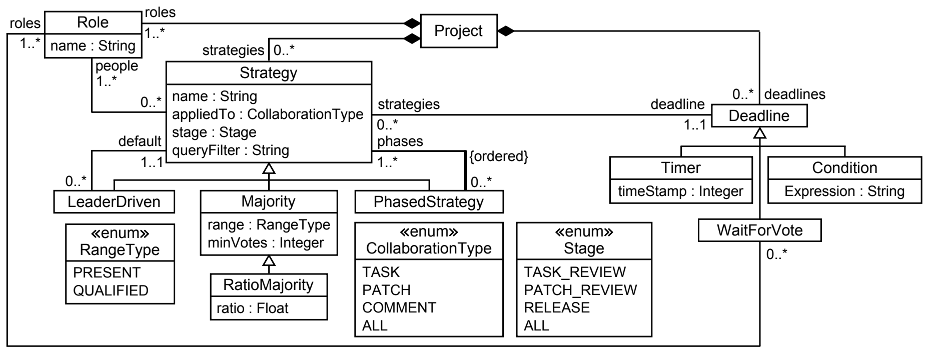


Fig. 3. Abstract syntax metamodel of our DSL to represent governance rules (expressed as a UML class diagram).

The abstract syntax metamodel of the language is shown in Figure 3. The concepts represented in the metamodel covered all the governance aspects identified in our study of OSS projets (Section 2). In particular, a development project (`Project` metaclass) includes a group of roles of users (`roles` reference), strategies (`strategies` reference) and deadlines (`deadlines` reference). A role has an identifier (`id` attribute in `Role` metaclass) and represents a group of people in the development community who can vote. Decision strategies are applied to a particular type of collaborations (`appliedTo` attribute) according to their nature in the track system (i.e., task, patch or comment) and at a concrete moment (`stage` attribute) of the process (i.e., task review, patch review, and release). The scope of the strategy can also be defined (`queryFilter` attribute) (e.g., only those ones which are tagged as high priority). We have defined several decision strategies (included in the hierarchy with root `Strategy`), which cover the following:

Majority Strategy. This decision strategy (`Majority` metaclass) accounts for the number of votes received by a collaboration element. Thus, only those elements which have received a support greater than 50% will be selected. The way of counting votes may differ depending on who is available in the moment of the votation. Since different terminology is used to refer the several types of majority (e.g., *plurality* or *relative majority* used in North America is called *simple majority* in Europe), we will use the neutral term *majority range* to determine exactly how the votes should be counted. Thus, if the range is `present`, the majority is based on those participants that are present in the moment of voting. Otherwise, a `qualified` range is based on those participants qualified to vote (presently available or not). A majority strategy can also require a minimum number of votes to be triggered (`minVotes` attribute).

As an example, a majority strategy to be applied only to tasks, voted by the committers if they are presently available, with no need of minimum votes and with a deadline of seven days from the task creation date would be specified as shown in Figure 4a.

A different percentage for the majority can also be set. In this case the strategy will be called ratio majority (`RatioMajority` metaclass) and the ratio value must be specified. For instance, this type of strategy would allow implementing well-known majorities such as three-fifths or two-thirds, which may be required for changes on fun-

```
Project myProject {
  Roles: Committers
  Deadlines:
    myDeadline : 7 days
  Strategies:
    myMajorityStrategy : Majority {
      applied to Task
      when TaskReview
      people Committers
      range Present
      minVotes 0
      deadline myDeadline
    }
}
```

(a)

```
Project myProject {
  Roles: Committers, ProjectLeader
  Deadlines:
    myDeadline : 7 days
  Strategies:
    myRatioStrategy : Ratio {
      applied to Task
      when TaskReview
      people ProjectLeader
      range Present
      minVotes 0
      ratio 75.00
      deadline myDeadline
    }
}
```

(b)

```
Project myProject {
  Roles: Committers
  Deadlines:
    myDeadline : Weekly
  Strategies:
    myMajorityStrategy : Majority { ... }
    myLeaderDrivenStrategy : LeaderDriven {
      applied to Task
      when TaskReview
      default myMajorityStrategy
      deadline myDeadline
    }
}
```

(c)

```
Project myProject {
  Roles: Committers, ProjectLeader
  Deadlines:
    myDeadline : 7 days
  Strategies:
    myMajorityStrategy : Majority { ... }
    myRatioStrategy : Ratio { ... }
    myPhasedStrategy : Phased {
      phases {
        myMajorityStrategy
        myRatioStrategy
      }
    }
}
```

(d)

Fig. 4. Examples of decision strategies: (a) Majority strategy, (b) Majority strategy using ratio (c) Leader-Driven Strategy, (d) Phased Strategy.

damental laws. As example of ratio majority, the definition shown in Figure 4b modifies the strategy presented before to be ratio-based with a ratio of acceptance of 75% and to be voted only by the project leaders.

Leader-Driven Strategy. When the decision of accepting a collaboration element relies on a user playing the role of leader (e.g., component or project leader), a leader-driven strategy (`LeaderDriven` metaclass) is followed. Thus, this decision strategy relies on the leader of a collaboration element to decide its acceptance. Next section describes how the leader is represented. The leader must also define a default strategy (`default` reference) where to delegate the decision in case the leader is not available (i.e. the leader doesn't vote before the deadline)

Figure 4c shows an example of leader-driven strategy to be applied only to tasks, with a majority default strategy to be applied when the leader does not make the decision and with a deadline of seven days from the task creation date would be (note that the `myMajorityStrategy` strategy defined before is reused as default behavior).

Phased Strategy. This is a composite strategy which allows applying several strategies in a chained way. The strategies are defined and applied in an ordered way (`phases` reference). Thus, a set of collaboration elements are selected according to the first strategy, the selected ones are then voted again and filtered according to the second strategy, and so on.

An example of phased strategy composed of two phases where the first one applies a majority among the committers and with a deadline of seven days from the task creation date, and the second phase, which has a deadline of seven days after the first phase

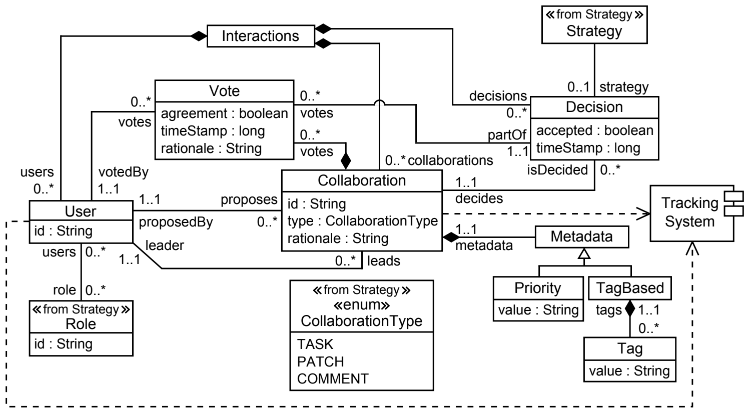


Fig. 5. Metamodel to represent collaborations.

has been done, applies a leader-driven strategy among the project leaders would be (note that `mymajorityStrategy` and `myRatioStrategy` are reused) is shown in Figure 4d.

Regarding the deadlines that trigger a decision strategy, we have currently defined three covering deadlines based on: (1) time (`Timer` metaclass), (2) a condition to be fulfilled in the collaboration (`Condition` metaclass) (e.g., the change of a tag in the collaboration model) and (3) a set of users have voted (`WaitUserVote` metaclass).

3.2 Applying Governance Rules

To be able to use the governance rules to manage the project we need to: (1) provide support for the recording of the information generated by all the participants interactions (e.g., proposals, votes, etc.), and (2) implement a decision engine that can use this information and apply the governance rules to make the corresponding decisions.

To record the participant's interactions we use the schema shown in Figure 5. A particular collaboration (`Collaboration` metaclass) is represented by an identifier (`id` attribute) and a rationale explaining the collaboration (`rationale` attribute). There are three types of collaboration elements (`type` attribute in `Collaboration` metaclass): tasks, which represent a request for an issue or a bug; patches, which represent the solution for a task; and comments, which represent comments that users can make to other collaborations. A collaboration element can also have some metadata information (`metadata` reference) and has a proposing user (`isProposed` reference) and a leader (`leader` reference). During the collaboration, a user (`User` metaclass) is identified by an identifier (`id` attribute) and belongs to a one or more roles (`roles` reference). Note also that both collaboration elements and users are linked to the tracking system (see `Tracking System` component) to retrieve the information of tasks/patches/comments and users, respectively. The metamodel also allows representing votes (`Vote` metaclass) from the users, which can agree/disagree with a collaboration (`agreement` attribute). A vote has a timestamp (`timeStamp` attribute) and a rationale (`rationale` attribute) explaining the reason for the positive/negative vote.

```
Project Apache {
  Roles : Developer, Committer,
          PMCMember, PMCChair
  Deadlines :
    twoDays : 2 days
  Strategies :
    codeStrategy : Ratio {              noCodeStrategy : Majority {
      applied to Task (tag = code)        applied to Task (tag != code)
      when TaskReview                     when TaskReview
      people Developer, Committer,        people Developer, Committer,
             PMCMember, PMCChair                 PMCMember, PMCChair
      range Present                       range Present
      minVotes 3                          minVotes 3
      ratio 100.00                        deadline twoDays
      deadline twoDays                  }
    }                               }
```

Fig. 6. Apache governance rules.

When the deadline for a governance rule passes, the decision engine receives the collaboration model as input and follows the rule instructions to process the data and take a decision. The decision updates the collaboration information to create the new decisions (`Decision` metaclass) and trace them back to the affected collaboration elements (`isDecided` reference). A decision can accept/reject a collaboration element (`accepted` attribute), includes a timestamp of the moment when the decision was made (`timeStamp` attribute) and refers to the strategy applied (`strategy` reference). A collaboration element has usually only decision element assigned except when a phased strategy is used, in this case the decisions for each phase and the final decision are also stored in the model. The decision engine is provided as part of our prototype tool implementation (see Section 5).

4 Case Study: Apache

We have used our approach to model the decision process applied in the Apache project (the choice of Apache was motivated by the fact that it has a more complex governance process than other analyzed projects).

We first represent the governance rules used to select tasks to accomplish. As described before, there are two types of voting procedures depending on whether the task to accomplish involves source code changes or not. The DSL defining the Apache governance rules is listed in Figure 6.

The example defines four roles and two strategies called `codeStrategy` and `noCodeStrategy`. The former is a ratio majority strategy which is applied to those tasks including the tag `code` (i.e., tasks involving changes in the code). All people belonging to the roles linked to the strategy (see tag `people`) and presently available (see tag `range`) can vote. It is required a ratio of agreement of 100% (i.e., meaning that everybody has to agree) and at least three votes. Finally, the strategy defines a deadline of two days, although the precise length is not specified in the project documentation. On the other hand, `noCodeStrategy` is a majority strategy which is applied to those tasks not including the tag `code`. Like the previous strategy, it is required at least three votes to make a decision and the deadline is two days after the creation of the task.

As an example of applying the previous rules, Figure 7 shows a possible collaboration model representing how developers made the decision about two task proposals

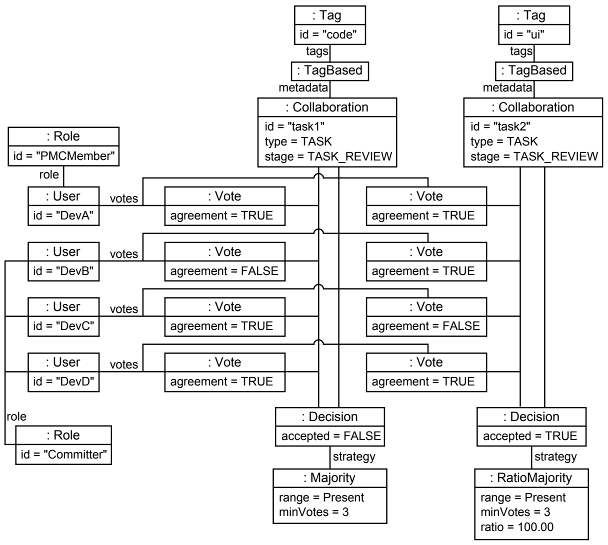


Fig. 7. Example of collaboration in Apache.

with identifier `task1` and `task2`. The former includes the tag `code` while the latter does not, thus allowing illustrating the application of both governance rules (i.e., `codeStrategy` rule for the first task and `noCodeStrategy` rule for the second one). For the sake of the clarity and conciseness, the `rationale` and `timeStamp` attributes have been removed in the Figure. As can be seen, after voting, although the first task has received more than three votes, it it is rejected because there is at least one vote negative. On the other hand, the second task is accepted because, although there is a negative vote, the majority of them agrees with the change.

Note that instead of the current process (based on mail communications and manual counting of the votes) our approach (and related tool support) automatically takes the decisions based on the rules and the interactions taking place between the developers. Moreover, both the collaboration and decisions are stored in along with the element decided (i.e., the task status change), thus avoiding scattering information along the project management tools (i.e., the mailing-list in this case) and improving the transparency of the process.

5 Implementation

Our approach has been implemented as an Eclipse plug-in [13] providing a collaborative infrastructure allowing developers and project managers to define and apply governance rules.

Figure 8 summarizes the architecture of our prototype tool. We rely on Mylyn[14] for accessing and updating the tasks of the development project we are collaborating on.

[13] The prototype can be downloaded from https://code.google.com/a/eclipselabs.org/p/governance-rules-mylyn/

[14] http://www.eclipse.org/mylyn

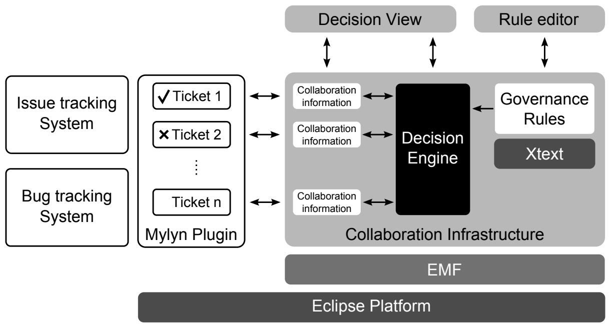


Fig. 8. Architecture of the developed tool.

Mylyn incoporates several connectors covering a great variety of tracking systems[15]. The collaboration models storing all the information about the ongoing discussions, votes, etc. are implemented on top the EMF [16] Eclipse component.

Our DSL to define governance rules has been developed in Xtext[17], a tool to create textual DSLs in Eclipse. In Xtext, DSLs are described by an annotated grammar from which the required tooling is generated (e.g., specific editor, syntax highlighting, auto-completion, content assistance, etc.). We defined the grammar for our DSL and generated the corresponding DSL tooling. The plugin therefore provides a specific editor to define governance rules (see *Rule editor*).

Once the rules are defined, the collaboration information is collected in a dedicated view (see *Decision View*) and, via Mylyn, linked to the project tasks as metadata. The rules plus this information are then interpreted by an engine developed in Java (see *Decision Engine*) that updates the tasks status according to the governance rules defined.

Figure 9 shows a snapshot of the platform including both the *Decision view* (on the left bottom part) and the *Rules editor* (on the right top part) for the Apache example presented before. Five tasks from the Apache repository (project `apache-httpd-2`) have been selected for the sake of conciseness. As can be seen, these tasks are shown in the *Task List* view provided by Mylyn and in the *Decision view* provided by our tool, which allows users to vote for/against these tasks. The view also includes buttons to configure the strategy (opening the *Rule editor* shown in the Figure), to apply it, to login and to refresh the view. As an example, the rules have been triggered using sample votes, thus resulting in the acceptance of the first two tasks and rejection of the rest.

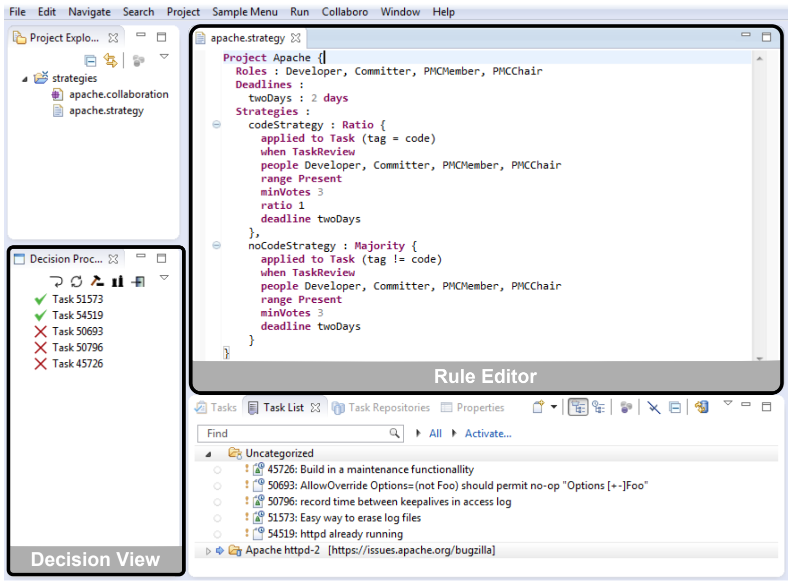
6 Related Work

The study of how people coordinate to develop a software system has been a research topic for a long time [3, 12–15], including the broader topic of IT governance [16–19]. In the field of software governance in OSS, several works have tried to classify how

[15] The complete list of supported tools can be found at http://wiki.eclipse.org/index.php/Mylyn_Extensions

[16] http://www.eclipse.org/emf

[17] http://www.eclipse.org/Xtext



```
Project Apache {
  Roles : Developer, Committer, PMCMember, PMCChair
  Deadlines :
    twoDays : 2 days
  Strategies :
    codeStrategy : Ratio {
      applied to Task (tag = code)
      when TaskReview
      people Developer, Committer, PMCMember, PMCChair
      range Present
      minVotes 3
      ratio 1
      deadline twoDays
    },
    noCodeStrategy : Majority {
      applied to Task (tag != code)
      when TaskReview
      people Developer, Committer, PMCMember, PMCChair
      range Present
      minVotes 3
      deadline twoDays
    }
}
```

- ✓ Task 51573
- ✓ Task 54519
- ✗ Task 50693
- ✗ Task 50796
- ✗ Task 45726

Uncategorized
- 45726: Build in a maintenance functionallity
- 50693: AllowOverride Options=(not Foo) should permit no-op "Options [+-]Foo"
- 50796: record time between keepalives in access log
- 51573: Easy way to erase log files
- 54519: httpd already running

Apache httpd-2 [https://issues.apache.org/bugzilla]

Fig. 9. Snapshot of the developed tool.

these projects are managed [20, 15, 21]. However, to the best of our knowledge, little attention has been paid to the precise definition and support for the governance rules in specific projects.

Some works report on concrete case studies. For instance, [1] reports on the coordination activities performed to solve bugs (e.g., how bugs are detected and closed, means used to coordinate the work, etc.). In [22] authors present how developers use mailing list during the development of FLOSS systems (e.g., which are the main topics discussed or the level or participation of the different user roles). In [4] some indications are provided to describe governance models in natural language.

Other approaches focus on leveraging collaboration information obtained by mining software repositories to infer coordination relationships and structures. The work presented in [23] describes an approach to discover potential social structures from the threads created in the mailing list of the project. In [24] visualization techniques are applied to easily discover communication patterns from Github repository metadata (e.g., the effect of geographic distance among developers, influence among cities, etc.). The tool called CrowdWeaver is present in [25], which allows coping with the complexity of managing crowdsourced projects. These tools could be adapted to facilitate the discovery of governance rules using our approach.

Concrete methodologies for collaboration strategies have also been proposed. For instance, [26] and [27] present approaches to support collaborative processes in groupware systems and online creative projects, respectively. However, they do not provide mechanisms to define and apply the governance rules to apply in each project.

Collaboration has also been the focus on two other DSLs. The approach presented in [28] describes a DSL to represent collaboration workflows that can appea in modeling tools (e.g., the steps needed to create a class diagram when several users are collaborating). In our previous work [29] we presented a DSL-based tool to collaborativelly develop DSLs, thus allowing representing the collaborations arisen in the process. Again, they do not provide any support for the governance of such collaborations.

7 Conclusion

In this paper we have presented an approach that opens the possibility to explicitly define and automatically aply the rules that govern a software development project. Our approach includes a new DSL for specifying the rules, a collaboration infrastructure to record the interactions (and to be able to trace them back) and a decision engine to automatically update the tasks status based on that. A prototype tool implemented as an Eclipse plug-in and relying on the Mylyn component for the connection with the most popular tracking systems has been developed. Note that although our approach supports both the definition and application of governance rules, they can be used independently and project leaders can therefore decide using only the former, which would improve the transparency and sustainability of governance model.

As further work, we would like to provide support for evolution in governance rules as they depend on social interactions it is common to find temporal exception, ambiguities and changes over time. Adding privacy concerns is also under evaluation (some projects may require anonimity in the votation phase, or keep private some discussions to all people with less privileges). Finally, we would like to mine existing software repositories to infer and study the governance rules they are using. It would be interesting to see if the extracted rules correspond to the perception of (external) participants (e.g., whether they have evolved through time or whether their application is full of exceptions) and how they correlate with other metrics of the projects.

References

1. Aranda, J., Venolia, G.: The secret life of bugs: Going past the errors and omissions in software repositories. In: International Conference on Software Engineering (ICSE), IEEE (2009) 298–308
2. Conway, M.E.: How Do Committess Invent? Datamation **14**(4) (1968) 28–31
3. Herbsleb, J.D., Grinter, R.E.: Splitting the organization and integrating the code: Conway's law revisited, IEEE (1999) 85–95
4. OSS Watch: Governance Models (2013)
5. McConnell, S.: Open-source methodology: Ready for prime time. IEEE software (August) (1999) 6–11
6. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: Apache and mozilla. ACM Transactions on Software Engineering and Methodology (TOSEM) **11**(3) (2002) 309–346
7. Shibuya, B., Tamai, T.: Understanding the process of participating in open source communities. In: ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS), IEEE (2009) 1–6
8. Mockus, A.: A case study of open source software development: the Apache server. International Conference on Software Engineering (ICSE) (2000) 263–272
9. Crowston, K., Wei, K., Howison, J., Wiggins, A.: Free/Libre open-source software development. ACM Computing Surveys **44**(2) (2012) 1–35

10. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison Wesley (2008)
11. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers (2012)
12. Kraut, R., Streeter, L.: Coordination in Software Development. Communications of the ACM **28**(3) (1995) 69–81
13. Parnas, D.L.: On the Criteria to be Used in Decomposing Systems into Modules. Communications of the ACM **15**(12) (1972) 1053–1058
14. Crowston, K., Wei, K., Li, Q., Eseryel, U.Y., Howison, J.: Coordination of free/libre open source software development. In: International Conference on Information Systems (ICIS). (2005)
15. Markus, M.L.: The governance of free/open source software projects: monolithic, multidimensional, or configurational? Journal of Management & Governance **11**(2) (2007) 151–163
16. Chulani, S., Williams, C., Yaeli, A.: Software Development Governance and Its Concerns. In: International Workshop on Software Development Governance (SDG), ACM (2008) 3–6
17. Ramasubbu, N., Balan, R.K.: Towards governance schemes for distributed software development projects. In: International Workshop on Software Development Governance (SDG), ACM (2008) 11–14
18. Van Grembergen, W.: Strategies for information technology governance. IGI Publishing (2003)
19. Webb, P., Pollard, C., Ridley, G.: Attempting to define it governance: Wisdom or folly? In: Hawaii International Conference on System Sciences (HICSS), IEEE (2006) 194.1–
20. Laat, P.B.: Governance of open source software: state of the art. Journal of Management & Governance **11**(2) (2007) 165–177
21. De Noni, I., Ganzaroli, A., Orsi, L.: The evolution of OSS governance: a dimensional comparative analysis. Scandinavian Journal of Management **29**(3) (2013) 247–263
22. Guzzi, A., Bacchelli, A., Lanza, M., Pinzger, M., Deursen, A.V.: Communication in Open Source Software Development Mailing Lists. In: Working Conference on Mining Software Repositories (MSR). (2013) 277–286
23. Bird, C., Pattison, D., Souza, R.D., Filkov, V., Devanbu, P.: Latent Social Structure in Open Source Projects Categories and Subject Descriptors. In: ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE). (2008) 24–35
24. Heller, B., Marschner, E.: Visualizing collaboration and influence in the open-source software community. In: Working Conference on Mining Software Repositories (MSR), ACM Press (2011) 223–226
25. Kittur, A., Khamkar, S., André, P., Kraut, R.: CrowdWeaver: visually managing complex crowd work. In: Computer Supported Cooperative Work and Social Computing (CSCW). (2012) 1033–1036
26. Duque, R., Rodríguez, M.L., Hurtado, M.V., Bravo, C., Rodríguez-Domínguez, C.: Integration of collaboration and interaction analysis mechanisms in a concern-based architecture for groupware systems. Science of Computer Programming **77**(1) (2012) 29–45
27. Luther, K., Fiesler, C., Bruckman, A.: Redistributing leadership in online creative collaboration. In: Computer Supported Cooperative Work and Social Computing (CSCW). (2013) 1007
28. Gallardo, J., Bravo, C., Redondo, M.a., de Lara, J.: Modeling collaboration protocols for collaborative modeling tools: Experiences and applications. Journal of Visual Languages & Computing (2012) 1–14
29. Cánovas Izquierdo, J.L., Cabot, J.: Enabling the collaborative definition of DSMLs. In: International Conference on Advanced Information Systems Engineering (CAiSE), To appear (2013)