R&I

CODE GOVERNANCE IN DECIDIM

Jordi Cabot / **Javier Cánovas**

SOM Research Lab
Av. Carl Friedrich Gauss, 5. Building B3
08860 Castelldefels

SOM

CONTENTS

LIST OF ACRONYMS

AGPL	Affero General Public License
DSL	Domain-Specific Language
GNU AGPLv3	GNU Affero General Public License version 3
GPLv3	General Public License version 3
GSD	Global Software Development
HTML	HyperText Markup Language
ICT	Information and Communication Technology
NGO	Non-Governmental Organization
OSS	Open Source Software
SHA	Secure Hash Algorithm
SQL	Structured Query Language

1 INTRODUCTION

1.1 OPEN SOURCE SOFTWARE DEVELOPMENT

We live in a software-enabled world. Global cost of software development is estimated to be over one trillion dollars (and over 200 billion in market revenue in Europe) making it a crucial market for Europe's Information and Communication Technology (ICT) initiatives. But, at the same time, complexity of this software is also growing, forcing people to join forces and collaborate in order to develop such software.

The development of Open Source Software (OSS) usually follows a method that harnesses the power of distributed peer review and transparency. Indeed, OSS is typically developed in a collaborative manner via online hosting platforms like GitHub. However, in practice, many OSS projects are not as open as they should.

Effective collaboration requires adequate technical solutions like Git (a distributed version control system [5]) but they are not enough. Adoption of good organizational practices and development processes within the development team is also a must. This is especially true when we have distributed teams and even more when the distributed team includes external developers or subteams subcontracted to work on part of the project (Global Software Development (GSD) [7]). Crowdsourcing parts of the development [22] could be regarded as an extreme case of this.

Getting these collaboration models to work at a large scale is very challenging. That is why many software projects decide to embrace the principles of OSS. According to the Open Source initiative: *OSS development is a development method that harnesses the power of distributed peer review and transparency*. OSS is typically developed in a collaborative manner via online code hosting platforms like GitHub [5] (with more than 75 million projects hosted and 25 million of registered users at the moment). And this is the main difference with respect to proprietary software: not only the code is open (i.e., free for everybody to access and modify) but also the development is (supposedly) performed *in the open* which favours the collaboration of the whole community behind the software, including its users (that nowadays everybody mostly agrees that should be active participants of the development process, for instance, as defended by all Agile methodologies[1]).

This distinction was clearly illustrated in the well-known essay, and later a book, *The Cathedral and the Bazaar* by Eric S. Raymond [18], based on his observations of the development of Linux kernel that then the author tested and validated on his own open source project. This essay contrasts two development models: the *Cathedral model* where code is developed by a restricted set of developers and the *Bazaar model* where development is a collaborative endeavor and users are co-developers as the way to rapid code improvement, effective debugging and aligned software evolution. This *co-developer* role does not mean users always contribute code, it highlights the fact that users are key members of the software community, have a say in it and can contribute in any form or shape they are able to, e.g., submitting bug reports, feature requests or just giving feedback of any aspect of the software. This is different from end-user development approaches [23] that pretended to convert users in a kind of developers to adapt themselves the software alone.

1.2 GITHUB AS THE DEVELOPMENT PLATFORM FOR OPEN SOURCE

In the last years GitHub has become a key enabler for the collaboratively development of OSS. GitHub is a Web-based Git version control repository hosting service overpowered with extra functionalities like bug tracking, feature requests, task management, and wikis for every project. The platform has popularized a pull-based development process based on pull requests (requests from, mostly external, users to have their code

[1] http://agilemanifesto.org/

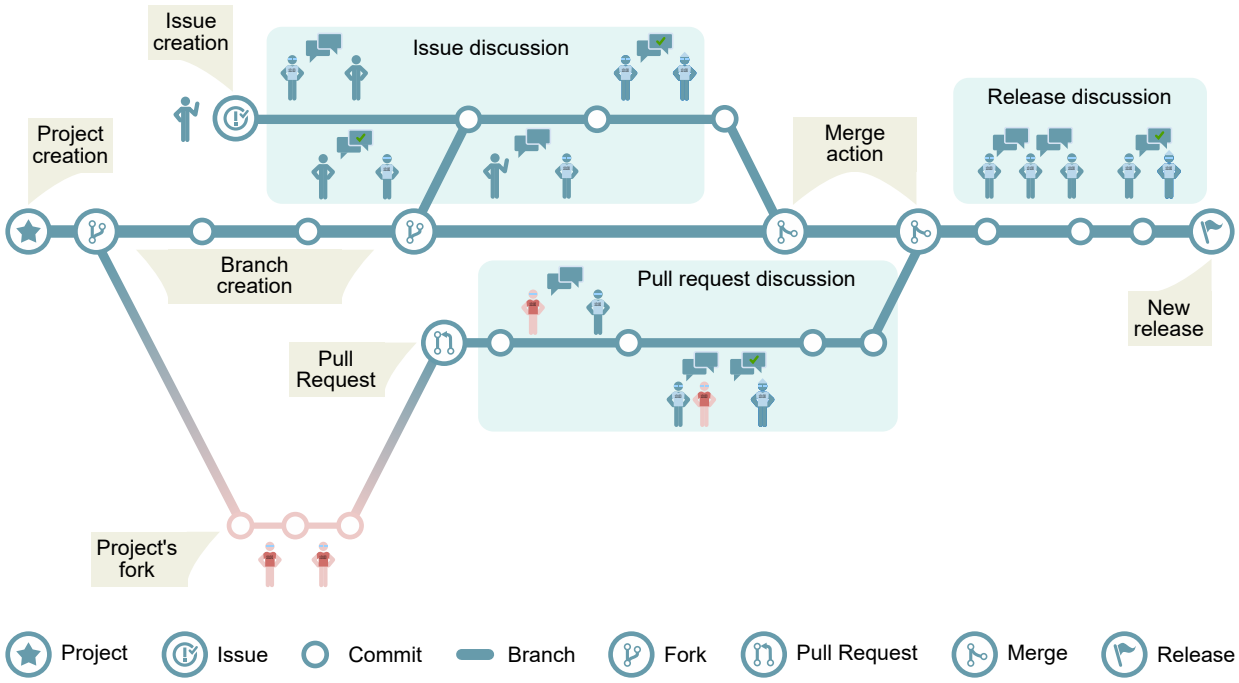


FIGURE 1: DEVELOPMENT PROCESS IN GITHUB.

added to the project), which promotes the collaboration and participation of developers in any project of the platform.

In GitHub, projects and users can be considered the main assets of the platform. A project is managed by two kinds of users: the owner and the project members, the former has full control over the repository, whereas the latter are granted full management permissions. Beyond this core users, we have all external users interested in collaborating with the project. A project is composed of one or many repositories, each one keeping track of its source code, submitted issues/bugs and pull requests (together with the corresponding associated events, e.g, creation, merging, etc.).

In the following we describe the main terminology used in Git-based repositories and the contribution models in GitHub. Figure 1 illustrates these concepts and models.

1.2.1 GIT TERMINOLOGY

The source code of a repository is hosted in an internal Git installation. Git is a popular version control system based on a master-less peer-to-peer replication where any replica of a given project can send or receive any information to or from any other replica. Although Git can theoretically work without a centralized repository, in practice there is usually a central repository that serves as the authoritative copy of the software project, from where everyone fetches and pushes the code to.

The structure of a Git repository follows a tree structure where each node is represented by a *commit* to which *references* can be assigned. A commit is uniquely identified (i.e., using a Secure Hash Algorithm (SHA)) and contains a revision of the files within the repository reflecting the state of the project at a given point in time (i.e., a snapshot). In particular, a commit stores the differences between the files that changed between two revisions (i.e., a patch). It also contains information (i.e., name and email) regarding the corresponding *author* and *committer*, where the former is the one that did the change and the latter is the one that applied the change to the repository. Furthermore, a commit includes a reference to its parent commit(s). Generally,

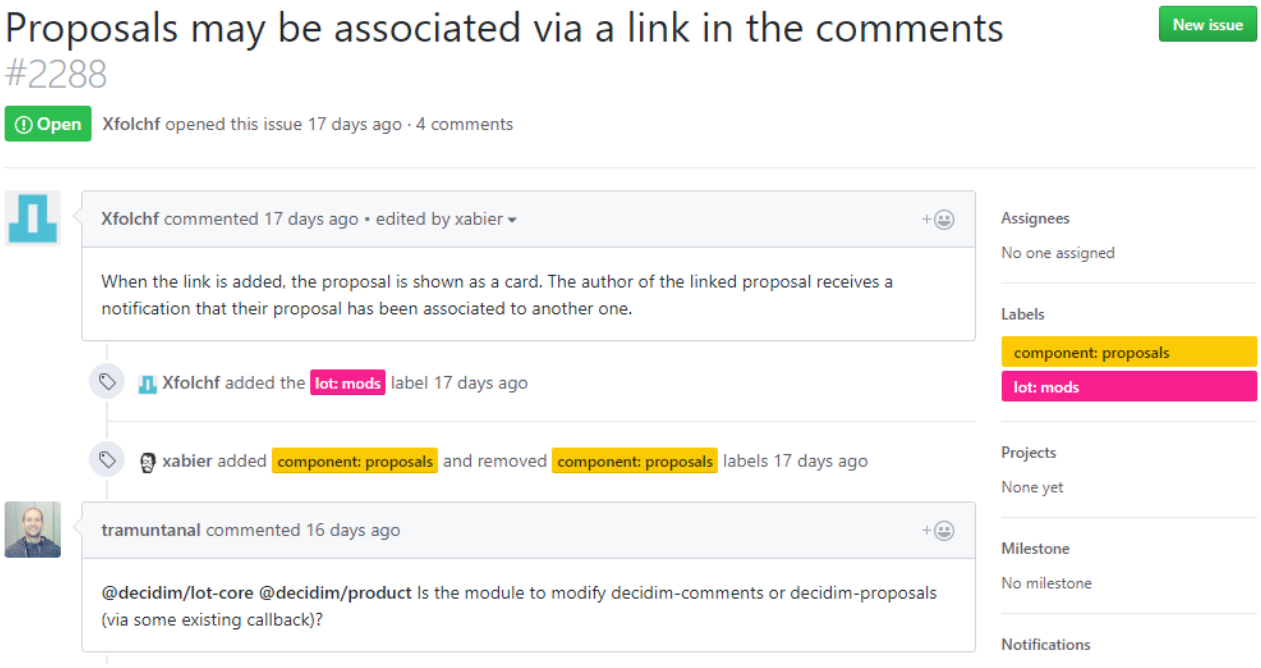


FIGURE 2: EXAMPLE OF ISSUE IN DECIDIM.

a commit has only one parent, that represents the previous state of the project; however, it can be parent-less (e.g., the commit originates the repository) or have multiple parents (when merging two or more branches).

Commits can be linked to different references, such as branches and tags. A *branch* represents a line of development that can be local if exists only in the cloned repository or remote if it belongs to the remote repository. The default branch is usually named *master*, but new ones can be created to start new separated lines of development (e.g., to work on a new feature or to fix a bug). A branch can be also be merged with another one (e.g., to make a new release when fixing a bug). A *tag* is a reference that can also be assigned to commits and it is generally used as marker for relevant events in the repository (e.g., releases, milestones).

1.2.2 CONTRIBUTION MODELS IN GITHUB

There are two basic ways to contribute to a repository: (1) by submitting an *issue* or (2) creating a *pull request*. GitHub issues usually describe a change request in natural language while pull requests include a set of commits (i.e., source code) implementing such request. Any user can create issues and pull requests to point out bugs or request/contribute new features. Examples of an issue and a pull request from the Decidim platform are shown in Figures 2 and fig:introduction:pr.

To create a pull request, the user has first to *fork* a project, then perform some changes and finally send the pull request to the original project. The pull request is evaluated (some discussions can arise as comments in the pull request) and if eventually accepted, the related code changes (i.e., commits) are merged into the original project.

Each contribution intent corresponds to a decision-making point in the project (i.e., issues and pull requests have to be accepted or rejected). Additionally, we also identify a third decision-making point, which covers the release of a version of the software product. In short, these are the main decision-making points we identify in any GitHub project:

ISSUES Submitting issues to a GitHub repository usually raises a discussion for a bug or feature request

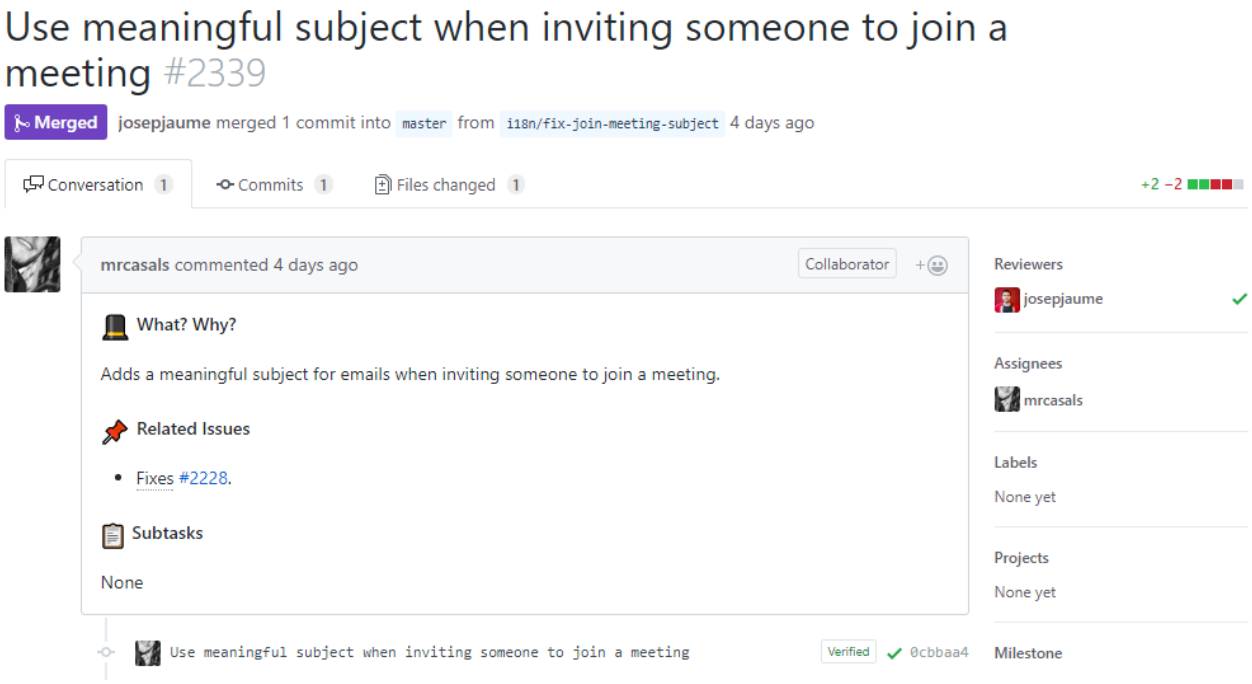


FIGURE 3: EXAMPLE OF PULL REQUEST IN DECIDIM.

which may eventually end up with the modification of the source code (by means of either a commit or a pull request).

PULL REQUESTS The contribution of pull requests involves the validation and testing of the source code to be merged in the codebase of the repository. This process usually involves discussions around the submitted code (e.g. in terms of quality or style) and the implementation of fixes to comply with repository's guidelines.

RELEASES The project roadmap usually includes periodical releases, which involve the identification of issues and pull requests that should be addressed and completed for that release.

1.3 CHALLENGES IN OPEN SOURCE SOFTWARE DEVELOPMENT

Although most of the software is (or heavily relies on) OSS, in practice, many OSS projects struggle to survive. OSS suffers from the *tragedy of the commons*: everybody wants to benefit from the software but they all hope others will chip in. In fact, we can conclude the OSS model is broken with many projects failing and getting abandoned in the very early stages (see [3] for some statistics). As a result, there is a grossly disproportionate imbalance between those consuming the software and those participating in building the software. In OSS, popular does not imply healthy, more the opposite.

A well known example is the OpenSSL cryptographic library project that, despite being used by two-thirds of the Web, was maintained by one full-time employee (as the famous heartbleed bug exposed). A more recent example is the fever around blockchain and cryptocurrencies. Despite being worth over $30bn and $65bn, there are only 15 meaningful contributors to the core Ethereum and Bitcoin projects respectively.

In theory, the collaboration proposed in OSS makes it better than proprietary software thanks to this higher community involvement. This is in principle, good news, since most of the crucial software projects for our society are OSS (like Apache Server, Firefox, Linux or WordPress). Nevertheless, in practice, many OSS

projects are not as open as they should. Yes, their source code is freely accessible but the management of the project and its leadership is not transparent and hardly ever follows any kind of democratic practices, making it extremely difficult for users and the community in general to influence the development of the project.

For instance, a manual analysis of the twenty-five most popular projects in GitHub reveals that only one (4%) explicitly described how user contributions would be managed (with another 28% giving partial hints). This means that 68% had no explicit governance model. Absolutely none of them were democratic (i.e., end-users could not vote in any way not even to elect people to represent them). In fact, the only one describing its decision-making process stated that *this project follows the timeless, highly efficient and totally unfair system known as Benevolent dictator for life*. This situation is similar in other open source platforms. For instance, in Eclipse[2] they even have established an award to the most open project to *recognize the open source project that best exemplifies the openness, transparency and diversity expected of great open source projects*. Clearly, the contribution processes for OSS projects must drastically improve to ensure their long-term sustainability.

And this is not the only problem. Most projects struggle to attract contributors. In fact, more than two thirds of all projects in GitHub have only one or two contributors [14], which of course limits a lot the interaction they may have with the users of that software specially if they have a large user base. Therefore, we can conclude the OSS model as it is now it is broken. This affects not only the long-term sustainability of the projects (with many projects failing and getting abandoned in the very early stages, see [19] for some statistics) but also their role in the growing of Europe's ICT where OSS is supposed to play an important role according to the European Software Strategy report.

Unfortunately, there are no easy solutions for this situation. When talking with the owners of some of these projects, a typical answer is that you can always fork the original project and adapt it to your needs but this is only possible for those users that are tech-savvy enough to do it and even then it is only feasible for very small projects. For larger ones, a fork is typically a group decision that ends up splitting up the community (as an example, we have recently witnessed the turmoil of the fork and later reunification around node.js[3]). Another frequent comment is that you can always contribute code to fix the bugs or implement the functionalities that you would like to see in the product. Again, this leaves out of the equation most of the users of the product that do not have the skills (or the time) to do it themselves but would still be able to at least influence the project future priorities. Instead, typically, projects work like a kind of meritocracy where only those who contribute new code can have a saying (and, as discussed above, sometimes not even that, only the founder or a very reduced group of core committers have any decisional power).

In the remaining of the report, we will take a close look on how these issues affect the Decidim project.

[2] `https://eclipse.org`
[3] `https://nodejs.org`

2 DECIDIM

Decidim, for the Catalan term *let's decide*, is a platform for participatory democracy which enables the creation of citizen participation portals [3]. The platform provides a set of components which configure different participatory spaces in the portals. For instance, Decidim incorporates components to enable the proposal of iniciatives, calling for assemblies, defining processes or calling for consultations.

> The Decidim project is a platform aimed at creating participatory portals. The platform defines a social contract for democratic guarantees and open collaboration for the community around the project. Furthermore, the development of the Decidim project is overseen by Metadecidim, another participatory portal devoted to allow anyone to contribute and discuss new features (or improvements) for the platform.

Portals created with Decidim allow any organisation (local city council, association, university, NGO, neighbourhood or cooperative) to create mass processes for strategic planning, participatory budgeting, collaborative regulatory design, urban spaces and elections. It also enables the organisation of in-person meetings, meeting invites, registrations, the publication of minutes, the structuring of government bodies or assemblies, the convening of consultations, referendums or channelling citizen or member initiatives to impact different decision making processes.

2.1 THE PLATFORM

The Decidim platform is modular, scalable, easy to configure and integrated with other tools. The platform is organized in terms of participatory spaces and components, which we briefly describe in the following.

PARTICIPATORY SPACES A space defines a participatory channel or media through which citizens or members of an organisation can process requests or coordinate proposals and make decisions. In Decidim we find four types of spaces, namely: *initiatives* (e.g., a citizen initiative), *processes* (e.g., a participatory budget), *assemblies* (e.g., a workshop to discuss a proposal) and *consultations* (e.g., a call for voting for/against a proposal).

PARTICIPATORY COMPONENTS A component defines a mechanism that enables a specific interaction between users and spaces. In Decidim we currently find sixteen types of components, namely: *meetings*, *conferences*, *conferences*, *incubators*, *proposals*, *participatory texts*, *surveys*, *discussions and debates*, *results*, *monitoring*, *votes*, *pages*, *blogs*, *observations*, *newsletters* and *search engines*.

Participatory spaces and components are used by Decidim users, which can be classified according to three groups:

VISITORS, who have access to all the content of the portal.

REGISTERED MEMBERS, who can contribute by adding comments, creating proposals, sending messages and following elements of the portal.

VERIFIED MEMBERS, who may be recognized as member of an organization, citizen or a decision-making community. Verified members enjoy the highest level of participation in the portal and, apart from the actions included in the previous group, they can also register for in-person meetings, support proposals, sign initiatives and vote for consultations.

Figure 4 illustrates the main participatory spaces and components in Decidim, together with the set of users. The Figure also includes the main dependencies between components, which we do not elaborate as it is out of the scope of this document, more information can be found in the work by Barandiaran et al. [3].

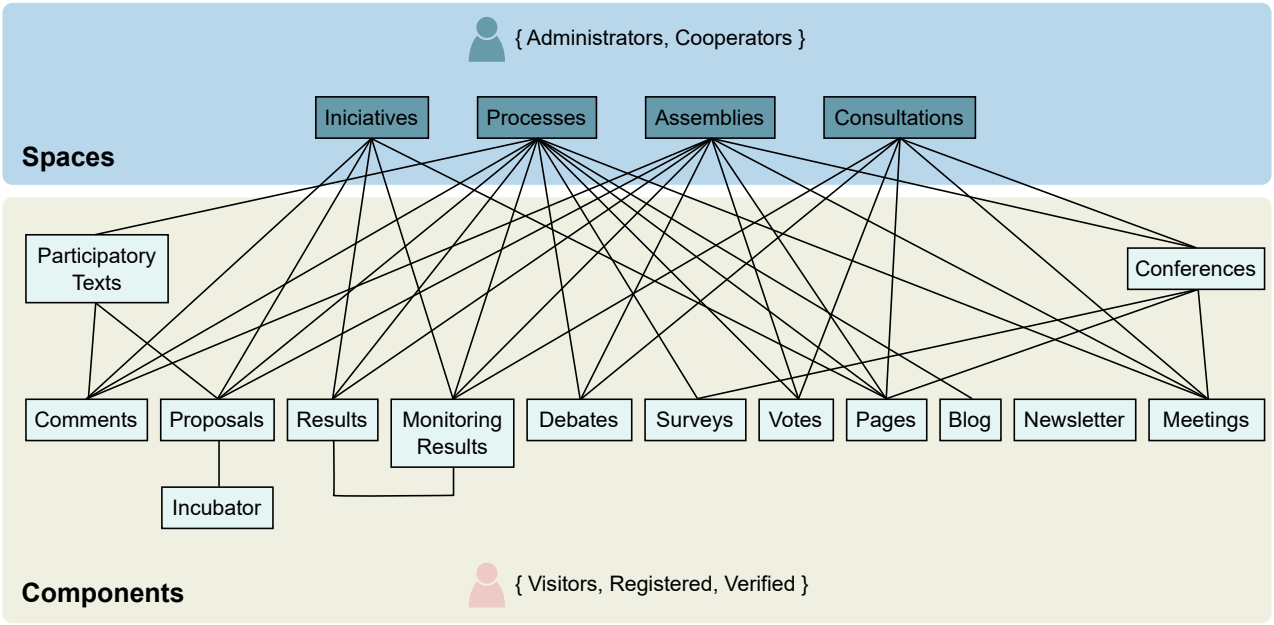


FIGURE 4: MAIN COMPONENTS OF DECIDIM.

2.2 DECIDIM SOCIAL CONTRACT

Decidim defines a social contract for democratic guarantees and open collaboration for the community around the project. Any user or developer of the Decidim platform must fully accept and commit to this social contract. The social contract covers six main topics[4]:

FREE SOFTWARE AND OPEN CONTENT The code of the platform (including modules, libraries or any other code that is developed for its functioning and deployment) will always be Free Open Source Software. Furthermore, the content, data, APIs and/or any other interfaces that are deployed to interact with any type of user must follow open and interoperable standard. Likewise, the content, text, graphics, fonts, audio, video or other design elements will be published under a Creative Commons By-SA[5].

TRANSPARENCY, TRACEABILITY AND INTEGRITY Decidim must ensure the transparency, traceability and integrity of documents, proposals, debates, decisions, or any other object, mechanism or participatory process. This means that all data related to all such participatory mechanisms and processes are available for download, analysis and treatment, following the most demanding standards and formats to share information.

EQUAL OPPORTUNITIES AND QUALITY INDICATORS The platform promises to ensure equal opportunities for all people, as well as for their proposals or other contributions the platform might host. The platform will offer equal starting opportunities to all participatory objects (proposals, debates, etc.) for them to be viewed, discussed, commented, evaluated or treated without discrimination of any kind. In this sense, the platform ensures that the digital identity of the users will always be personal and not transferable.

DATA CONFIDENCIALITY The confidentiality and privacy of the personal data that people might provide to participate in any of the functionalities and/or possibilities of participation that the platform provides shall be guaranteed at all times.

[4] Extracted from `https://decidim.org/contract/`
[5] `https://creativecommons.org/licenses/by-sa/3.0`

ACCOUNTABILITY AND RESPONSABILITY A commitment to citizens shall be taken to respond to all queries and contributions in the shortest time possible. A commitment shall also be adopted to follow-up the results of participatory processes and to respond to those demands specifically request it.

CONTINUOUS IMPROVEMENT AND INTER-INSTITUTIONAL COLABORATION Mechanisms for periodic review and evaluation will be provided in order to facilitate the continuous improvement of the platform.

2.3 DECIDIM AND METADECIDIM

Portals deployed by Decidim enable for participatory democracy in city councils, associations or universities, among others. Additionally, this participatory nature is also applied to the very own Decidim platform. Thus, the development of the Decidim project is driven by a participatory portal devoted to allow anyone to contribute and discuss new features (or improvements) for the platform. This participatory portal which oversees Decidim is called Metadecidim.

This has the huge benefit of empowering all kinds of users to participate in the future of Decidim. Nevertheless, it also introduces a duality in the governance of the project between those participating in the project via Metadecidim versus those participating directly on the GitHub side of the project. Precise rules on the governance of both layers (and on the interaction between the two) may be required to keep a consistent view of the project across all project levels.

3 GOVERNANCE IN OPEN SOURCE

Governance rules help to prioritize and manage tasks, and contribute to the long-term sustainability of the project by clarifying how developers (and end-users) should collaborate. Despite their importance, these rules are usually implicit or scattered in the project documentation/tools. In this chapter we show how to enable the explicit definition and enforcement of governance rules for OSS projects.

The development of large-scale software is a long-life process which has to cope with a huge number of development tasks consisting of either implementing new issues or fixing bugs [2]. Effective and precise prioritization of these tasks is key for the success of the project.

With this purpose, each project defines and applies its own set of governance rules, that is, a set of instructions that describe how to contribute to the project and how decisions regarding the acceptance/rejection of such contributions are going to be made. According to Conway's Law [8], this coordination has a direct impact on the product being built [12]. For instance, rules can be as simple as *team leaders decide the tasks to do* or more complex as *the task to be done will be the one most voted by the developers participating in the project*. Governance rules enable the coordination of developers in order to advance the project during its whole lifespan and, more importantly, their evolution allows the project to adopt societal changes on the way people want to collaborate, thus promoting the sustainability of the development process.

Despite their importance, in practice, governance rules are hardly ever explicitly defined, specially in the context of OSS, where it is hard to find an explicit system-level design, a project plan or list of deliverables [17, 20, 21] (e.g., browsing issue and bug trackers for any project quickly uncovers "forgotten" requests from several years ago where new users periodically comment on trying to guess if that feature/bug has been implemented/fixed or not, if it will be, and how they can help to push it up on the priority list). Moreover, the support for enforcing such governance rules is even more limited. This hampers and can even scare away the participation of new users/developers who must invest some time understanding the "culture" of the project.

To alleviate this situation, mechanisms to facilitate the communication and assignment of work are considered crucial [16, 4]. Tracking systems (i.e., bug-tracking such as Bugzilla[6] and issue-tracking systems such as Mantis[7]) 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 enforcing governance rules (e.g., automatically prioritizing tasks based on votes, easy tracking of decisions made in the project, etc.).

Therefore, we believe the explicit definition of governance rules along with the corresponding infrastructure to help developers follow them would have several benefits, including improvements in the transparency of the decision-making process (promoting the understanding of the project evolution), 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 inconsistent behaviour in the evolution of the project).

3.1 THE IMPORTANCE OF DEFINING GOVERNANCE MODELS

To assess the importance of having explicit governance rules in OSS projects we conducted a survey in 2015 [13]. Our aim was to answer the following research questions:

[6] http://www.bugzilla.org/
[7] http://www.mantisbt.org/

RQ1 How hard is it to contribute to OSS projects?,

RQ2 Is it feasible to make explicit the governance rules? if so, how useful?

To promote the participation, the survey was announced and distributed online, being freely accessible and targeting any developer willing to participate. To maximize the number of responses, we kept the number of questions as reduced as possible. The survey comprised questions on three different aspects:

QA. USER PROFILE This aspect includes questions to characterize the survey participant according to his/her experience contributing to OSS projects, the role (i.e., project leader, contributor, end-user, no expertise or bad experience), and the number of projects contributed (if any).

QB. UNDERSTANDING OF THE CONTRIBUTION PROCESS This aspect analyzes how difficult it is to understand how to contribute (e.g., providing source code, notifying bugs, proposing new features, etc.) to the project. This category comprised questions to assess the difficulty level (ranking from 1 - easy, to 5 - hard) of the main contribution activities (i.e., how difficult was to propose a change request, a bug or a patch) and understanding the governance rules applied in the process (i.e., how difficult was to know when a proposal was accepted or whether/when it will be included in the next release).

QC. GOVERNANCE RULE DEFINITION Including questions to analyze the opinion regarding the feasibility and usefulness of making governance rules explicit.

The survey was promoted through social media (i.e., twitter, facebook, etc.), remained available for one month and was answered by 51 people that had the option to remain anonymous if they wished[8]. All participants answered the full set of questions.

Figure 5 depicts the main results for the questions profiling the surveyed developers (i.e., QA questions). Note that respondents who tried but failed to contribute (i.e., QA1) sometimes considered their attempt as contribution (i.e., QA2), which will influence the remainder of the survey. As can be seen, the great majority of the respondents are contributors (see QA1 results) who have participated in 1 to 5 projects (see QA2 results).

Figure 6 shows the results regarding the difficulty of understanding the contribution process and addresses RQ1. The results reveal that, in average, contribution activities are easy to understand: proposing a change request receives an average value of 2.37 (see QB1 results), notifying a bug receives an average value of 1.80 (see QB2 results) and sending a patch receives an average value of 2.07 (see QB3 results). However, the results regarding the understanding of the application of governance rules get worse evaluations in average. Thus, knowing the status of a proposal (i.e., QB4) or its inclusion in the next release (i.e., QB5) receive an average value of 3.15 and 3.72, respectively.

Figure 7 shows the results with regard to the governance rule definition, thus answering RQ2. Respondents believe that most governance rules can be written down fully or partially (see QC1 results) and, in those cases where it is possible, they also acknowledged its positive impact to encourage new contributors to participate (see QC2 results) in the project. Furthermore, the results also show that a tool enforcing governance rules would be welcome in the context of OSS projects (see QC3 results), though with less support, which may indicate reluctance to (semi)automate the development process.

Our survey also included an open question to gather opinions about how to make the understanding of the governance easier. Among the answers, it is remarkable that a couple of developers noted that it is a common problem to put in the effort for sending patches to later realize that they are delayed or ignored with no clear reason (e.g., one of the answers was *I really wanted to work on a project and fixed few bugs. They answered immediately that they will use it, but it then took more than a year until they really did*). There were

[8] The results are available at http://goo.gl/tvzYz9

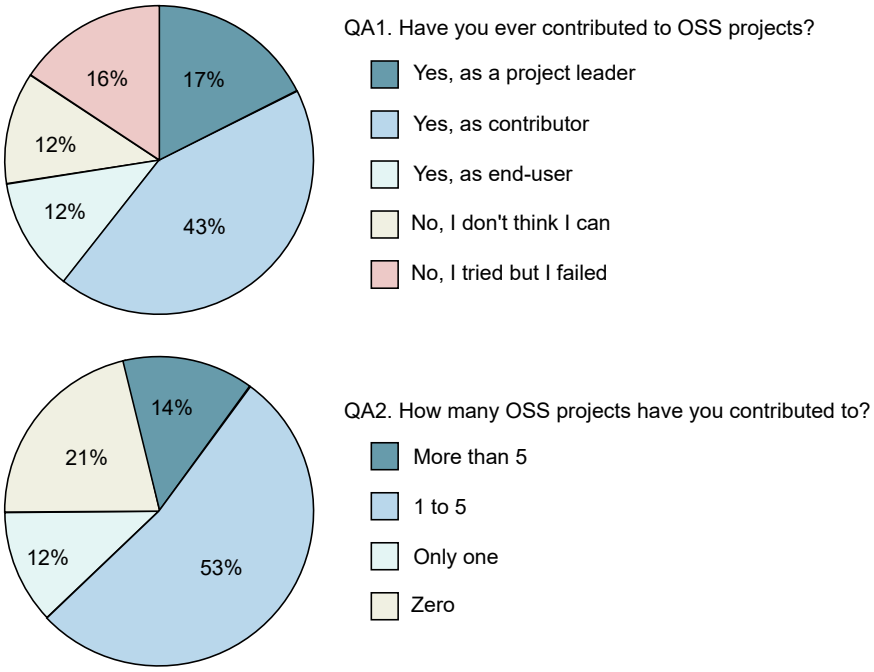


FIGURE 5: RESULTS OF THE SURVEY REGARDING THE USER PROFILE.

also other answers commenting about the need for better mechanisms to facilitate the understanding of how the project is governed, such as better user interfaces (e.g., *one for the end-user (simple, elegant, clear) and another for developers/maintainers with full information*), delegating a person to specifically care about first-time contributions or ranking the projects according to their *friendliness*. These answers reveal that developers are willing to contribute but the effort to make is still too high for some of them.

Although the results of our survey may not represent the universe of all developers in OSS projects, we believe that they are illustrative enough to motivate the need of explicitly defining the governance rules behind OSS projects.

3.2 HOW OPEN SOURCE IS CURRENTLY GOVERNED

While the survey presented in the previous section shown the need for explicitly defining the governance rules in OSS projects, in this section we analyze eight well-known OSS projects in order to empirically study how existing projects are governed, that is, the contribution models they apply (cf. Section 1.2.2). This information will help us to further confirm the survey results and to know better this domain in order to develop our solution as described in the next sections.

Most large OSS projects apply a contribution model organized in terms of tasks (either feature requests or bugs, also known as *issues* in GitHub) that are solved by means of patches (implementing new features in response to the requests or correcting the bugs, usually addressed by *commits* or *pull requests* in GitHub) which at some point can be selected to be part of the next software release. Figure 8 summarizes this workflow. The workflow includes three main decision-making points, namely: (1) task selection, (2) patch review and (3) release selection. Decisions at each point are taken based on the governance rules defined for the project.

We have studied eight OSS projects where external contributions are welcome (i.e., Android[9], GNOME[10],

[9] http://www.android.com
[10] https://www.gnome.org

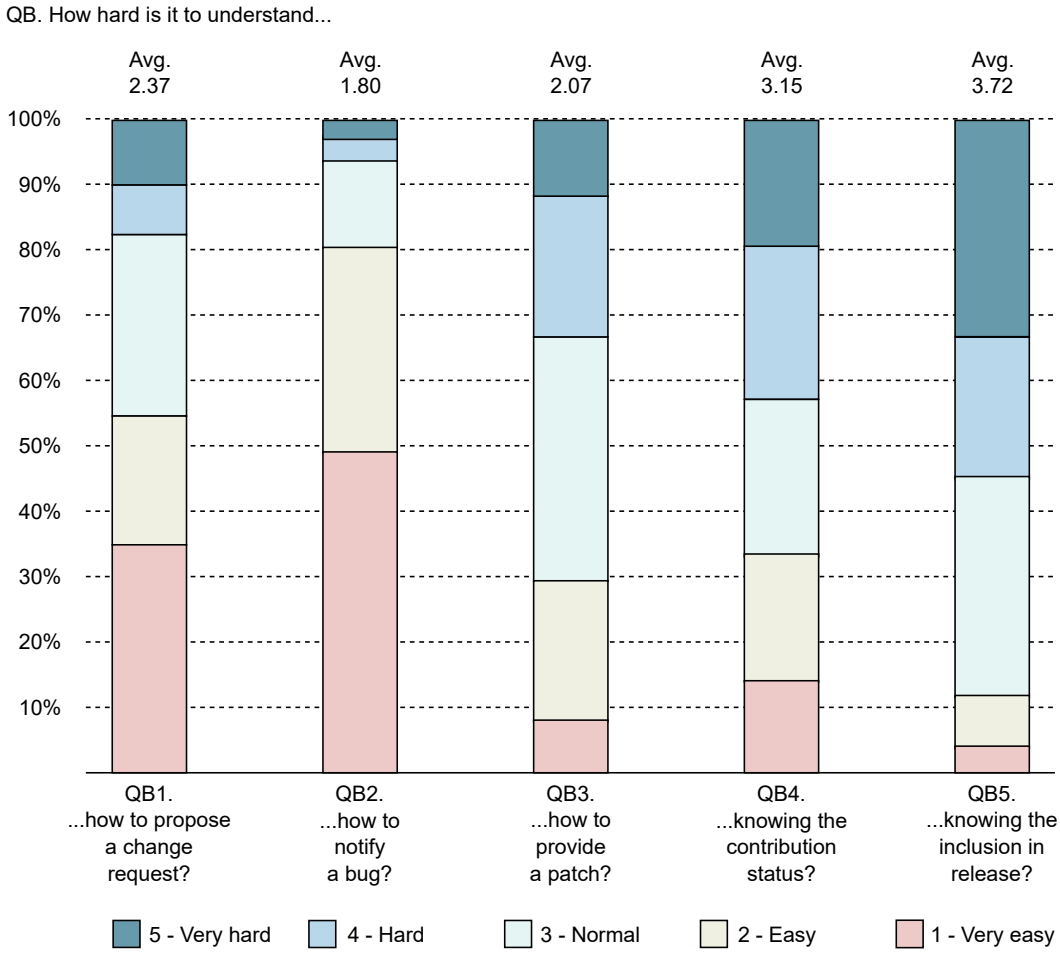


FIGURE 6: RESULTS OF THE SURVEY REGARDING THE UNDERSTANDING OF THE CONTRIBUTION PROCESS.

Apache (web server)[11], Mozilla[12], Python[13], Moodle[14], EMF[15], MoDisco[16]).

Each project was analyzed according to nine dimensions targeting three main viewpoints, specifically: organizational viewpoint, which studies how developers are organized (mainly hierarchically or not) (*organization* dimension), the main communication means (*coordination* and *tracking system* dimensions) and who can participate (*participation* dimension); development process viewpoint, which assesses how the review process is done in each one of the three main decision-making points (*task review*, *patch review* and *release decision* dimensions); and governance rule definition viewpoint, which studies where the governance rules are usually defined and who is involved (*Rules Def. / App.* and *Roles* dimensions). We believe these dimensions cover all elements that are part of the governance of an OSS project.

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

[11] https:/www.apache.org
[12] https://www.mozilla.org
[13] https://www.python.org
[14] https://moodle.org
[15] https://www.eclipse.org/emf
[16] https://www.eclipse.org/modisco

TABLE 1: COMPARISON OF HOW OSS SYSTEMS ARE GOVERNED.

PROJECT	ORG.	COORD.	T. SYST.	PART.	T. REV.	P. REV.	REL. DECISION	RULES DEF. / APP.	ROLES
Android	Hierarchy	Forum	Git Gerrit	Anyone	Yes (Approver)	Yes (Verifier)	Yes (Project Lead)	Documentation / Track system	Contributor Developer Verifier Approver Project Lead
GNOME	Hierarchy	IRC mailing-list	Bugzilla	Anyone	Yes	Yes (Bug Squad)	Yes	Documentation / Track System	Contributor Mentor
Apache Web Server	Meritocracy	Mailing-list	Bugzilla	Anyone	Yes (Voting)	Yes (Voting)	Yes (Voting)	Documentation / Mailing-list	User Developer Committer PMC Member PMC Chair
Mozilla	Hierarchy	Forum Mailing-list IRC	Bugzilla	Anyone	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	Anyone	No	Yes (Reviewer)	Yes (Core Developer)	Documentation / Track System	Contributor Reviewer Core Developer
Moodle	Hierarchy	Forum	Moodle tracker	Anyone	Yes (Component leads)	Yes (Developers)	Yes (Component Leads)	Documentation / Track System	Users Developers Component Leads Integrators Testers Manteiners
EMF	Hierarchy	Forum	Bugzilla	Anyone	Yes (Committer)	Yes (Committer)	Yes (Project Leader)	Documentation / Track System	User Contributor Committer Project Leader
MoDisco	Hierarchy	Forum	Bugzilla	Anyone	Yes (Committer)	Yes (Committer)	Yes (Committers)	Documentation / Track System	User Contributor Committer Project Leader

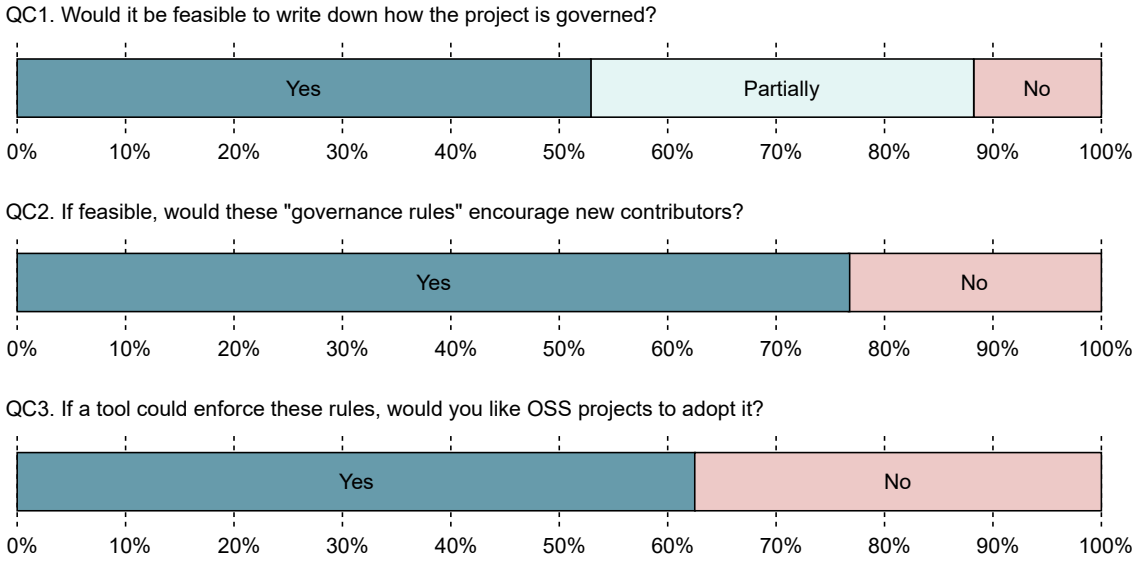


FIGURE 7: RESULTS OF THE SURVEY REGARDING THE GOVERNANCE RULE DEFINITION.

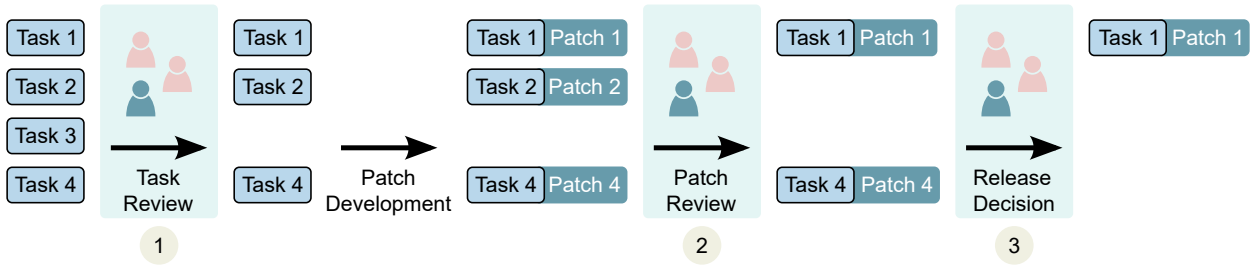


FIGURE 8: WORKFLOW FOLLOWED BY A TASK AND MAIN DECISION-MAKING POINTS.

vast majority of analyzed projects follow a hierarchical scheme, meaning that there exist several hierarchical levels of user groups collaborating in the development (e.g., leaders, contributors, users, etc.), where each group/role has different permissions and tasks. 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.

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

TRACKING SYSTEM The management of tasks is performed by tracking systems, Bugzilla being the most popular one. Interestingly enough, tracking systems are also used as a forum to discuss possibles tasks which are not mature enough to be considered as real issues. For instance, in the GNOME project, some tasks[17] are used to openly discuss pOSSible changes in the development strategy.

[17] http://felipec.wordpress.com/2011/09/23/no-gnome-doesnt-want-user-feedback-how-i-argued-in-favor-of-voting-i

PARTICIPATION In general, in OSS, anyone is able to take part in the development, i.e., it is not necessary to be an official developer to inform new feature requests or bugs.

TASK REVIEW Once a task or issue (i.e., feature request/bug) is notified, it can be reviewed to be accepted or rejected. Except for Python, where all the tasks 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 may come with the corresponding patch (i.e., commit or pull request in GitHub) implementing the improvement (if it is a feature request) or the fix (if it is a bug). Before incorporating the patch into the product, it is possible to perform a revision and test the quality of the patch. All the studied 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 analyzed projects, the decision of selecting when to perform the release and which tasks should be incorporated is normally 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.

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 to make things worse, the result of their application is scattered throughout the management systems (i.e., track systems or mailing-lists).

ROLES The group of users collaborating in the development are classified according to a set of roles. All the analyzed projects include both the role of developer (a.k.a. contributor/committer), who can commit changes into the source code repository of the software projecs, and the corresponding role for leaders (a.k.a owners/chairs).

In summary, most projects are hierarchical where 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 collaboration tools (e.g., email, mailing-lists or forums).

The development of Apache web server is the main exception. The project is governed by a voting-based decision rule where all contributors can decide which tasks should be accepted and which ones should be included in 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 process should be performed. Thus, if the task involves changes in the source code, the voting should result in an unanimous agreement in order 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 a rationale, which can therefore help to solve the disagreement.

Clearly, there is an interesting trade-off between the openness of the project and the challenges in managing the community behind it. Our feeling is that most projects start being as open as possible and that they become closer and closer as they grow as the only way they see to manage the community in an efficient way.

n-bugzilla-and-got-banned-as-a-result

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. This evidence confirms the results gathered in the survey presented in the previous section.

Moreover, the result of the application of these rules is directly updated in the tracking systems, where normally there is no traceability 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 led to that decision.

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

3.3 GOVERNANCE MODEL DEFINITION

In order to define the set of governance rules for development tasks, we propose to use a Domain-Specific Language (DSL)[18] providing specialized constructs that facilitate the specification of decision rules to be followed during the management of development tasks.

This language does not presume any specific governance model nor it does force you to follow a concrete one, its aim is just to make it explicit. Our goal is to provide a simple but precise language to make explicit such rules, thus promoting the understanding of how the project is developed and eventually encouraging developers to contribute. Based on our validation, the language is expressive enough to cover typical governance scenarios; nevertheless it could be tailored to cover more complex situations. Furthermore, it opens the door to automatically enforce those rules as discussed later on.

A DSL is defined by three main elements [15]: abstract syntax, concrete syntax and semantics. The abstract syntax defines the main concepts of the language and their relationships, and also includes well-formedness rules constraining how to use them. 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 use metamodelling techniques [6] to define the abstract syntax. Similar to grammars, metamodels restrict the possible structure of valid models for a DSL. More specifically, we have defined a governance metamodel to represent the concepts and relationships needed to specify governance rules.

Governance models conforming to this metamodel (i.e., models that can be expressed as valid instances of this metamodel) represent specific sets of governance rules of software projects. As concrete syntax, we have opted for a textual language following a typical block-based structure. Thus, each instance of a metaclass is textually represented by its keyword and a block that contains the name and value of its attributes. Containment references are represented as nested blocks while non-containment references use an identifier to refer to the referred element. Other textual syntaxes can also be considered, for instance, to verbalize a model into Spanish or English.

The abstract syntax metamodel of the language is shown in Figure 9. The concepts represented in the metamodel cover all the governance aspects identified in our study of OSS projects and *has been extended in the context of this report* to cover the definition of governance models in GitHub, including the ability to represent users and a new governance rule to cover consensus-based decision-making processes.

[18] A DSL is a language specifically designed to perform a task in a certain domain. Some examples of DSLs are: HTML in the Web domain, SQL in the database domain or Ant in the software integration domain.

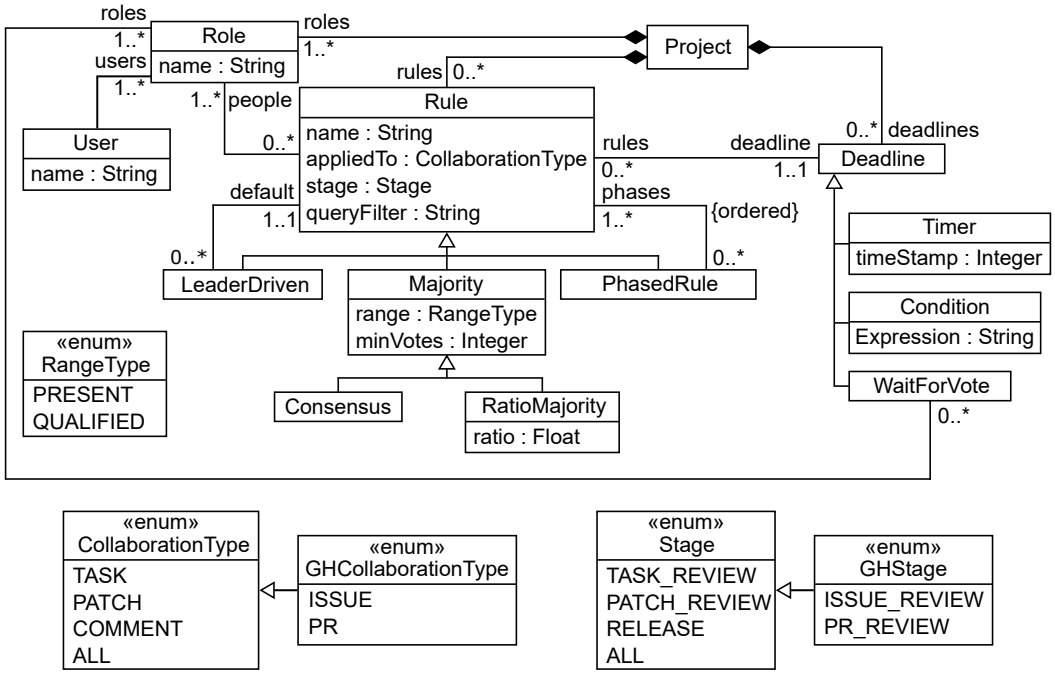


FIGURE 9: ABSTRACT SYNTAX METAMODEL OF OUR DSL TO REPRESENT GOVERNANCE RULES.

A development project (`Project` metaclass) includes a group of roles of users (`roles` reference), rules (`rules` reference) and deadlines (`deadlines` reference). A role has an identifier (`id` attribute in `Role` metaclass) and represents a group of people (`User` metaclass) in the development community who can vote. Decision rules are applied to a particular type of collaborations (`appliedTo` attribute) according to their nature in the tracking system, which can be either a generic type (i.e., task, patch or comment, see `CollaborationType`) or GitHub-specific (i.e., issue or pull request, see `GHCollaborationType`). Furthermore, decision rules are applied a specific moment (`stage` attribute) of the process (i.e., task review, patch review and release), which can be either a generic phase (see `Stage`) or GitHub-specific (see `GHStage`). Finally, the scope of the rule can also be defined (`queryFilter` attribute) (e.g., only those tagged as high priority).

We have predefined several types of decision rules (included in the hierarchy with root `Rule`), which cover the cases analyzed before:

MAJORITY RULE Under this rule (`Majority` metaclass) only those collaborations which have received a support over 50% will be selected. The way of counting votes may differ depending on who voted during the lifespan of the rule. Since different terminology is used to refer the several types of majority rules (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 the votes of those participants presented during the voting time. Otherwise, a `qualified` range is based on the votes of those participants qualified to vote (having voted or not, where abstention means disagreement). A majority rule can also require a minimum number of votes to be triggered (`minVotes` attribute).

As an example, a majority rule 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 with our textual concrete is shown in Figure 10.

A different percentage for the majority can also be set. In this case the rule will be called ratio majority

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

FIGURE 10: EXAMPLE OF MAJORITY GOVERNANCE RULE EXPRESSED IN OUR DSL.

```
Project myProject {
  Roles: Committers, ProjectLeader
  Deadlines:
    myDeadline : 7 days
  Rules:
    myRatioRule : Ratio {
      applied to Task
      when TaskReview
      people ProjectLeader
      range Present
      minVotes 0
      ratio 0.75
      deadline myDeadline
    }
}
```

FIGURE 11: EXAMPLE OF RATIO MAJORITY GOVERNANCE RULE EXPRESSED IN OUR DSL.

(`RatioMajority` metaclass) and the ratio value must be specified. For instance, this type of rule would allow implementing well-known majorities such as three-fifths or two-thirds, which may be required for changes on fundamental laws. As example of ratio majority, the rule shown in Figure 11 modifies the rule presented before in Figure 10 to be ratio-based with a ratio of acceptance of 75% and to be voted only by the project leaders.

It is also possible to define a special case of majority rule which requires a support of 100% (e.g., full agreement in the community), which will be called consensus (`Consensus` metaclass). As example of consensus, the rule shown in Figure 12 modifies the rule presented before in Figure 10 to be consesus-based.

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

An example of leader-driven rule to be applied only to tasks, with a majority default rule to be applied when the leader does not make the decision and with a deadline of seven days from the task creation date is shown in Figure 13 (note that the `myMajorityRule` rule defined before is reused as default behavior).

```
Project myProject {
  Roles: Committers
  Deadlines:
    myDeadline : 7 days
  Rules:
    myMajorityRule : Consensus {
      applied to Task
      when TaskReview
      people Committers
      range Present
      minVotes 0
      deadline myDeadline
    }
}
```

FIGURE 12: EXAMPLE OF CONSENSUS GOVERNANCE RULE EXPRESSED IN OUR DSL.

```
Project myProject {
  Roles: Committers
  Deadlines:
    myDeadline : 7 days
  Rules:
    myMajorityRule : Majority { ... }
    myLeaderDrivenRule : LeaderDriven {
      applied to Task
      when TaskReview
      default myMajorityRule
      deadline myDeadline
    }
}
```

FIGURE 13: EXAMPLE OF LEADER-DRIVEN GOVERNANCE RULE EXPRESSED IN OUR DSL.

```
Project myProject {
  Roles: Committers, ProjectLeader
  Deadlines:
    myDeadline : 7 days
  Rules:
    myMajorityRule : Majority { ... }
    myRatioRule : Ratio { ... }
    myPhasedRule : Phased {
      phases {
        myMajorityRule
        myRatioRule
      }
    }
}
```

FIGURE 14: EXAMPLE OF PHASED GOVERNANCE RULE EXPRESSED IN OUR DSL.

PHASED RULE This is a composite rule to apply several rules in a chained way. The rules are defined and applied in an ordered way (`phases` reference). Thus, a set of collaborations are selected according to the first rule, the selected ones are then voted again and filtered according to the second rule, and so on.

An example of phased rule 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 has been done, applies a leader-driven rule among the project leaders is shown in Figure 14 (note that `myMajorityRule` and `myRatioRule` are reused).

Regarding the deadlines that trigger a decision rule, we have currently defined three covering deadlines based on: (1) time (`Timer` metaclass), (2) an OCL[19] 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).

[19] http://www.omg.org/spec/OCL

4 GOVERNANCE IN DECIDIM

This section focuses on the analysis of the previous governance concepts in relation with the Decidim project. Decidim is a GitHub project which includes thirteen repositories, where one of them implements the core framework. The platform also uses development supporting tools, mainly to provide package management, perform code quality and promote communication among developers. All the analysis done in this section was performed based on a **snapshot of the Decidim GitHub project of November, 23rd 2017**.

Decidim GitHub project includes a set of repositories which we analyzed (on November, 23rd 2017) to define its governance model. Generally speaking, the governance model in Decidim mainly relies on a set of consensus-based rules where developers have to unanimously agree on the acceptance of issues and pull requests, as well as what to include in the next release.

4.1 DECIDIM IN GITHUB

As commented above, the Decidim project is being developed in GitHub, a Web-based Git version control repository hosting service overpowered with extra functionalities like bug tracking, feature requests, task management and wikis. The project includes thirteen repositories, which we briefly describe in the following:

DECIDIM This is the main repository of the project and implements the core participatory democracy framework. It also includes a generator and multiple Ruby gems made with Ruby on Rails.

METADECIDIM This repository is the code of the participatory portal to decide how to develop Decidim. The portal has been created using the Decidim framework itself.

DOCKER This repository includes some Docker containers for deploying a generic Decidim participatory portal in a Docker container server.

DECIDIM.ORG This repository contains the web source code for the `decidim.org` site. It has been developed as a static webpage which relies on MIDDLEMAN[20]

DECIDIM-INITIATIVES This repository implements the support for initiative spaces in the Decidim framework. The development of this space is currently performed separated from the main repository.

TRAINING Training curriculum that offers workshops and activities around political participation mediated by technology, digital security and privacy.

DOCS-FEATURES This repository includes the sources for the documentation regarding the Decidim features and future roadmap.

DECIDIM-RESULTS This repository implements the support for the results module in the Decidim framework. The development of this space is currently performed separated from the main repository.

DOCS.DECIDIM.ORG This repository includes the documentation of Decidim. It contains only a commit and seems to be abandoned for the moment.

DECIDIM-ACCOUNTABILITY This repository implements the support for the accountability module in the Decidim framework. The development of this space is currently performed separated from the main repository.

[20] https://middlemanapp.com/

TABLE 2: REPOSITORIES INCLUDED IN DECIDIM GITHUB PROJECT AND MAIN ACTIVITY METRICS.

REPOSITORY	COMMITS	ISSUES	P.R.	BRANCHES	RELEASES	CONT.	L.A.
decidim	1,474	236	13	51	43	28	1 day
docker	42	0	0	1	0	2	1 day
decidim.org	327	5	5	28	0	10	5 days
Metadecidim	30	0	0	1	0	1	2 days
decidim-initiatives	125	1	0	2	0	1	9 days
training	107	2	0	6	0	3	19 days
docs-features	10	0	0	3	0	2	1 month
decidim-results	12	0	0	1	0	2	1 month
docs.decidim.org	1	0	0	1	0	1	2 months
decidim-accountability	138	3	0	1	3	5	2 months
design-admin	54	7	0	3	0	5	5 months
design	829	10	0	52	0	13	5 months
RFC-DIP	9	10	0	1	0	1	7 months

Legend: P.R. = Pull Request. Cont. = Contributors. L.A. = Last Update

DESIGN This repository implements the new design and the front-end implementation of Decidim platform for the city hall of Barcelona.

DESIGN-ADMIN This repository implements the administration view of Decidim Barcelona (see previous repository).

RFC-DIP This repository implements a proposal for identity management system in Decidim. The system enables anonymous, institutionally certified, publicly verifiable and privately managed digital identity that makes use of asymmetric (private/public key) encryption and blockchain.

Table 2 shows a summary of some activity metrics of these repositories. As can be seen, `decidim` and `decidim.org` repositories are the most active ones. Others, like `design`, `design-admin` and `RFC-DIP` repositories have not shown any relevant activity in the last months.

4.1.1 DEVELOPMENT PROCESS

Decidim is deploying a custom development process which relies on (a) Metadecidim to discuss change features for Decidim and (b) GitHub to address their implementation. Next we briefly describe this development process, which is illustrated in Figure 15.

New features are first discussed in Metadecidim as proposals. Metadecidim allows the Decidim community to vote, comment and discuss on proposals, thus enabling the participatory selection of new features for the Decidim platform. Once the proposals have been voted and discussed, the Decidim team makes a final evaluation and creates a backlog to address them in the Decidim platform (see *Product Backlog*).

At this point, the backlog includes a set of development tasks to evolve the Decidim platform. Note that the identification and definition of such development tasks usually involves translating what the Decidim community discussed (usually with no technicalities) into the corresponding GitHub issues describing how the Decidim platform should be modified. Thus, a single proposal in Metadecidim can easily become a number of issues in the GitHub platform.

Once the backlog tasks have been identified, the process follows an agile methodology where sprints drive the development. In each sprint, the Decidim team first plans the set of backlog tasks (i.e., GitHub issues) to

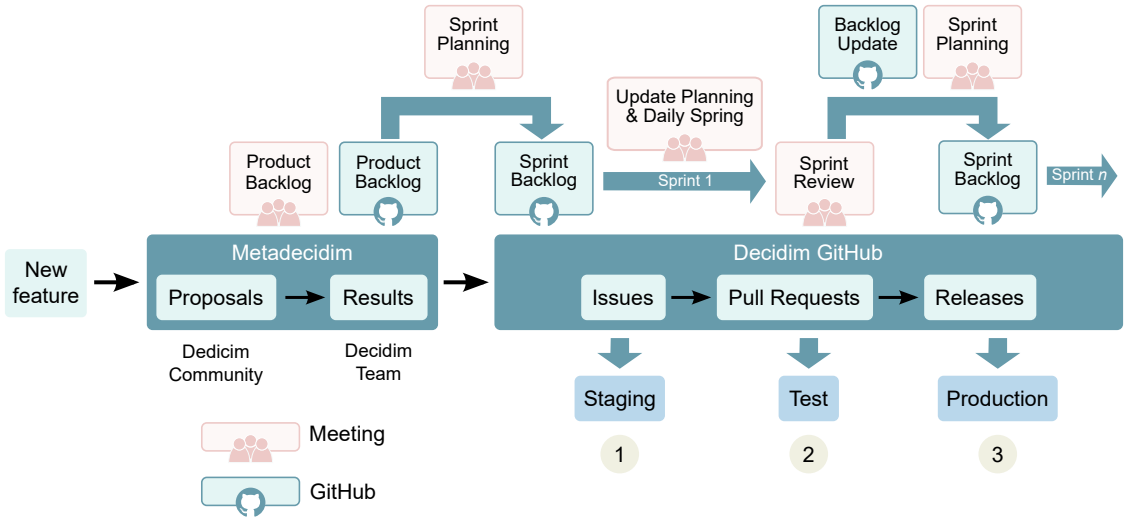


FIGURE 15: DEVELOPMENT PROCESS FOLLOWED IN DECIDIM.

address (see *Spring Planning*), which are then implemented (as pull requests in GitHub). At the end of the sprint, the Decidim team performs a spring review and starts the cycle again if there are more tasks in the backlog.

The development of Decidim follows the GitHub standard development process (cf. Section 1.2), where the issue-tracker is used to track change proposals and the pull-based development process (i.e., usage of pull requests) is applied to implement any change in the code. Thus, issues in the spring backlog are addressed in the Decidim issue-tracker, implemented as pull request in the platform and eventually merged with the main source code branch for their release.

Release procedure follows a three step process which starts by publishing the changes in a staging server and, if validated, they are published in a test server; which in turns, if validated, are finally published in the production server.

It is important to note that this process is currently being adopted by the community. At the moment of elaborating this report (i.e., November, 2017), the process mainly relies on GitHub (see right part of Figure 15) to drive the development. Thus, the role of Metadecidim in the process is still under discussion and implementation. In this section we will focus on the analysis of the governance model in the Decidim GitHub project. Some comments on the future evolution of this governance model (potentially with an key role for Metadecidim) is discussed in the next section.

4.1.2 OTHER PLATFORMS USED

Apart from GitHub, the Decidim project also uses a set of development tools aimed at tracking analytics, improving code quality checks and promoting communication. In the following, we present these platforms sorted according to their category:

PACKAGE MANAGEMENT The main project publishes the Ruby packages at RUBYGEMS[21], a package manager for the Ruby programming language that provides a standard format for distributing Ruby programs and libraries, and their documentation at RUBYDOC[22]. The project also uses GEMMNASIUM[23] to keep

[21] https://rubygems.org/gems/decidim
[22] http://www.rubydoc.info/github/decidim/decidim/master
[23] https://gemnasium.com/github.com/decidim/decidim

track of Ruby dependencies and security alerts.

CODE QUALITY The platform employs a number of code quality tools, namely: CIRCLECI[24], as continuous integration and delivery platform; CODECLIMATE[25], to perform code review for test coverage, complexity, duplication, security, and style; CODECOV[26] for code coverage; CROWDIN[27], a localization project management platfom and translation tool; INCHCI[28], to assess the documentation quality; and ROCKETVALIDATOR[29], to validate the websites included in the platform.

COMMUNICATION Decidim uses GITTER as main communication tool for contacting the developers team[30], which is an Open Source instant messaging tool specially tailored for developers and users of GitHub repositories.

4.2 ANALYSIS OF THE CURRENT DEVELOPMENT GOVERNANCE PROCESS

In this section we dig into the governance model of the Decidim GitHub project (and repositories) beyond the generic description provided above.

4.2.1 OVERVIEW

We start the analysis by having a first look at the repository to look for insights that may help to understand how the development process actually works in practice. We mainly focus on license documents, newcomer indications and code of conduct specifications. In the following we report on our findings.

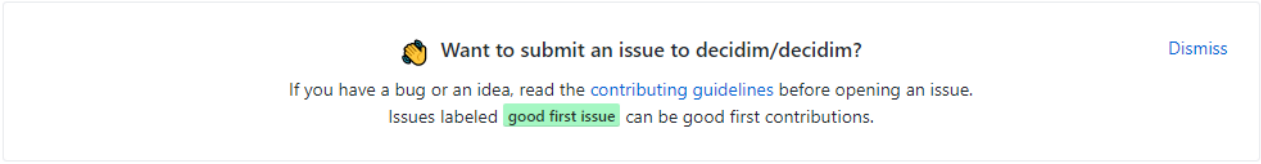
4.2.1.1 LICENSE

Software licenses are a legal instrument which govern the use or redistribution of the software. The Open Source community deeply relies on software licenses to guarantee the distribution, modification and reuse of existing code. Although it rarely happens, once a license has already been specified, Open Source projects may change their licenses to better suit the requirements of the development communities and to cope with unaddressed legal issues.

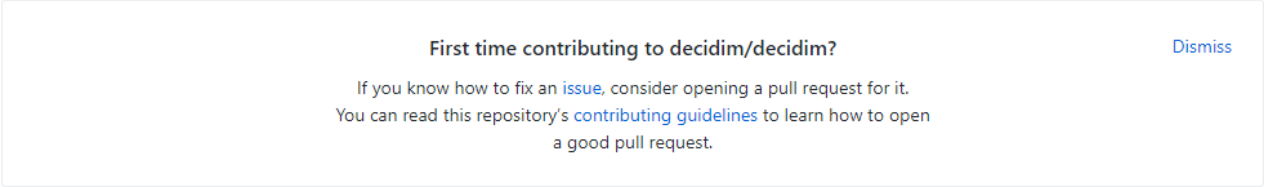
In Decidim, most of the repositories state clearly the software license, which is GNU Affero General Public License version 3 (GNU AGPLv3) [10]. GNU AGPLv3 is a free, copyleft license based on the GNU General Public License version 3 (GPLv3) [11] and the Affero General Public License (AGPL) [1]. This license protects developers' rights in two main dimensions: (1) assert copyright on the software, and (2) provide legal permission to copy, distribute and/or modify the software. Furthermore, improvements made in alternate versions of the program have to become available for other developers, including the source code. Some well-known software projects licensed as GNU AGPL are RSTUDIO, a free and open-source Integrated Development Environment (IDE) for R, a programming language for statistical computing and graphics; and MONGODB, a free and open-source cross-platform document-oriented database program.

By relying on GNU AGPL license, any developer willing to contribute to Decidim can be sure that such code will remain open even if it is used for derivative works.

[24] https://circleci.com/gh/decidim/decidim
[25] https://codeclimate.com/github/decidim/decidim/trends/technical_debt
[26] https://codecov.io/gh/decidim/decidim
[27] https://crowdin.com/project/decidim
[28] https://inch-ci.org/github/decidim/decidim
[29] https://rocketvalidator.com/s/2c6ce461-b007-4071-85d7-75e49bc768e8/v
[30] https://gitter.im/decidim/decidim



(a)INDICATIONS FOR CONTRIBUTING ISSUES.



(b)INDICATIONS FOR CONTRIBUTING PULL REQUESTS.

FIGURE 16: INDICATIONS FOR NEWCOMERS IN DECIDIM REPOSITORY.

4.2.1.2 NEWCOMER INDICATIONS

Contributions coming from external developers or newcomers are usually welcomed in any Open Source project. However, it is not always easy to find indications or guidelines to identify which parts of a software project need help. It is therefore crucial for any Open Source project aiming at capturing newcomers to provide these kind of indications.

The repositories of the Decidim project make use of a GitHub feature to clarify how new issues and pull requests (except `decidim-initiatives`) can be reported. For instance, Figure 16 shows the indications for contributing issues (see Figure 19) and pull requests (see Figure 16b) for the `decidim` repository.

These indications kindly invite the developer to read the `CONTRIBUTING.md` file of the corresponding repository (when available)[31]. This file briefly describes how to address issues and pull requests to the developer team. However, in its current state, these indications do not cover crucial governance information such as (a) how issues and pull requests are selected, (b) who is in charge of deciding that or (c) when they are going to be evaluated.

4.2.1.3 CODE OF CONDUCT

In the last years, Open Source projects have increasingly been defining codes of conduct to promote their expectations and standards of ethical behavior. A code of conduct establishes ground rules for communications between participants, outlines enforcement mechanisms for violations and tries to codify the spirit of a community, such that anyone can contribute comfortably regardless of gender, ethnicity, physical challenges or sexual orientation [24].

The Decidim GitHub project includes a document describing the code of conduct[32]. This document defines a set of rules to guarantee a harassment-free experience for everyone willing to contributes in the project.

[31] The `CONTRIBUTING.md` file for the `decidim` repository is located at `https://github.com/decidim/decidim/blob/master/CONTRIBUTING.md`

[32] Available at `https://github.com/decidim/decidim/blob/master/CODE_OF_CONDUCT.md`

TABLE 3: MAIN DEVELOPERS IN DECIDIM.

TOP 10 AUTHORS			TOP 10 COMMITTERS		
USER	**# COMMITS**	**% COMMITS**	**USER**	**# COMMITS**	**% COMMITS**
Josep Jaume Rey Peroy	585	16.18%	GitHub	872	24.12%
htmlboy	409	11.31%	Josep Jaume Rey Peroy	653	18.06%
David Rodríguez	355	9.82%	htmlboy	351	9.71%
Marta Armada	348	9.63%	Marta Armada	346	9.57%
Marc Riera	314	8.69%	beagleknight	244	6.75%
beagleknight	293	8.11%	Marc Riera	235	6.50%
decidim-bot	203	5.62%	Oriol Gual	192	5.31%
Oriol Gual	133	3.68%	David Rodríguez	132	3.65%
Xabier E. Barandiaran	130	3.60%	jsperezg	125	3.46%
jsperezg	126	3.49%	Amaia Castro	77	2.13%

4.2.2 REPOSITORY AND COMMUNITY ANALYSIS

In order to facilitate the reverse engineering of the current governance model in Decidim, we have conducted a study of both the repositories and community activity in the Decidim GitHub project.

The community involved in Decidim repositories is composed of 145 members where 35 of them (24.14%) have access to the code, 26 of them (17.93%) have contributed code at least once (by means of pull requests, thus not having access to modify the code) and, the rest, 84 of them (57.93%) have only participated in discussions of issues and pull requests.

4.2.2.1 CODE ANALYSIS

Developers mainly contribute code to Decidim by means of commits, as commented in Section 1.2. A commit has an author (the developer that contributes the code) and a committer (the developer who has permission to modify the codebase of the project. We analyzed the top 10 developers who authored and committed commits in the Decidim GitHub project. Table 3 shows the results of the study. As can be seen, the commits contributed by these developers represent more than a half of the commits of the whole community.

Note as well that some of the developers identified in this top list are actually bots (e.g., `decidim-bot` and `GitHub`), which are generally used to perform code quality checks or other automatic tasks.

In general, the author and committer of a commit are the same developer, in particular, in small Open Source projects. However, the study of this information helps to classify developers between internal/core contributors (i.e., committers) and external contributors (i.e., those developers with no access to the code base). According to the results shown in Table 3, most of the authors match with the committers in the top 10 lists. Further analysis revealed that 47.27% of the commits in Decidim come from external developers who contributed code to be validated by a committer.

We analyzed how the number of authors and committers evolved along time. Figure 17 shows this analysis, where it can be observed that at the beginning of the project only committers were contributing code to the platform. It was in August 2016 when external contributors started to participate in the development of the platform by authoring commits that were eventually accepted by the committers. Since then, the number of authors have been always higher than the number of committers, which is a good indicator of good performance to attract new contributors for the platform.

Finally, we analyzed the distribution of code, comments and blank lines in the source of Decidim. Figure 18

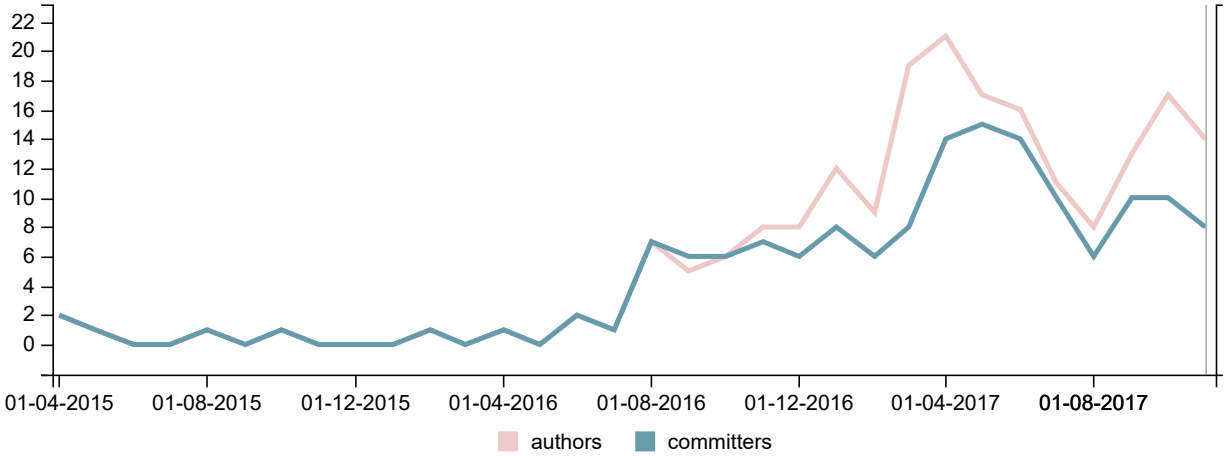


FIGURE 17: ACTIVE DEVELOPERS IN DECIDIM

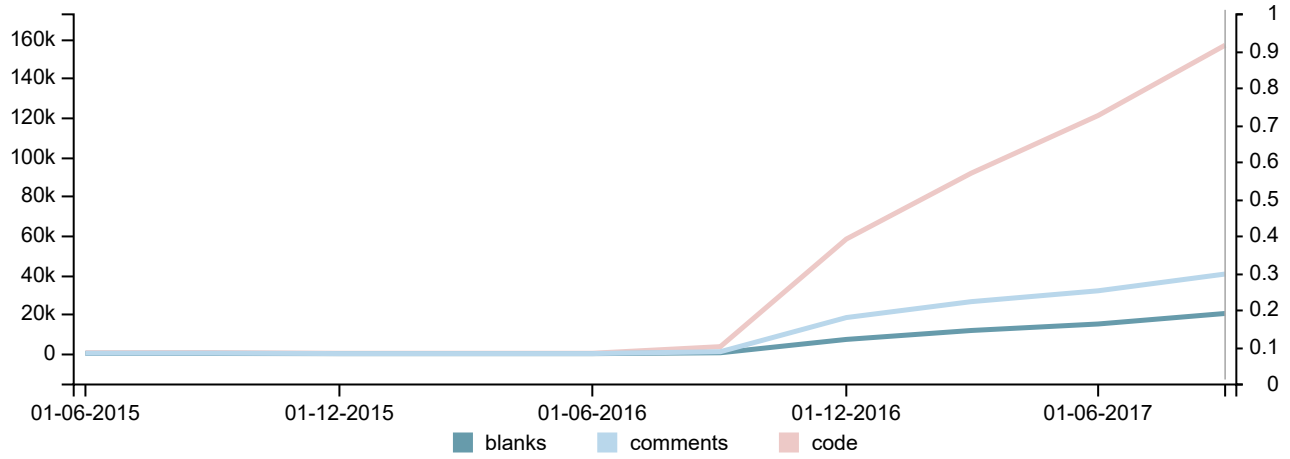


FIGURE 18: CODE EVOLUTION IN DECIDIM

shows the results of this analysis. As can be seen, the number of lines of code has been increasing since September 2016 and at higher rate than the number of lines of comments.

4.2.2.2 ISSUE AND PULL REQUEST ANALYSIS

In the previous section we analyzed the activities mainly related to code development in the Decidim GitHub project. In this section we will study how Decidim treats its issues and pull requests, which offers a clear view of how it is governed.

We first analyze the main community members participating in the issues and pull request of the project, in particular, the top 10 members opening/closing these elements. Table 4 shows the results of this analysis. As can be seen, it is easy to spot the main people participating in the issues and pull requests. The group of users *josepjaume*, *oriolgual* and *mrcasals* seem to lead the activities related to issues in Decidim. They do not only report the highest number of issues but also participate in their closing, thus meaning that they are part of the decision-making process. A similar result is obtained for pull requests, where we can also find the users *josepjaume* and *mrcasals* leading the activities of opening and closing.

TABLE 4: TOP 10 DEVELOPERS INVOLVED IN ISSUES/PULL REQUESTS IN DECIDIM.

TOP 10 ISSUE OPENERS		TOP 10 ISSUE CLOSERS		TOP 10 PR OPENERS		TOP 10 PR CLOSERS	
USER	**# ISSUES**	**USER**	**# ISSUES**	**USER**	**# PR**	**USER**	**# PR**
josepjaume	273	josepjaume	247	josepjaume	287	josepjaume	587
oriolgual	100	mrcasals	167	mrcasals	243	mrcasals	338
mrcasals	92	oriolgual	110	deivid-rodriguez	202	oriolgual	217
andreslucena	83	beagleknight	91	beagleknight	187	beagleknight	176
beagleknight	78	lastpotion	55	decidim-bot	182	htmlboy	52
virgile-dev	52	amaia	40	oriolgual	89	lastpotion	40
carolromero	50	arnaumonty	28	lastpotion	73	andreslucena	36
xabier	42	andreslucena	18	htmlboy	63	xabier	25
deivid-rodriguez	39	htmlboy	16	andreslucena	40	martuishere	16
arnaumonty	27	deivid-rodriguez	11	Dor3nz	36	arnaumonty	13

Figure 19 shows the collaboration graph for the issues and pull requests in Decidim. In this graph, nodes represent developers and node size the number of issues or pull request where they have collaborate on (the bigger the higher). On the other hand, edges represent a collaboration between two developers on a same issue or pull request, and edge thickness the number of collaborations in the same element (the thicker the higher). This collaboration graph confirms the results obtained previously identifying the top 10 contributors in issues and pull requests. Furthermore, the graph allows us to identify the main collaboration connections between collaborations.

We then calculated some metrics to study how issues and pull requests are addressed in Decidim. We defined three main metrics: (1) number of messages, which help us to measure the discursive level in project's contributions; (2) number of people participating, which reveals the involvement of the community; and (3) time to close, which provides some insights about the main priorities and interests in the project.

Figure 20 shows the results of this study; while Figures 20a to 20f show the boxplots for the metrics and Figure 20g shows the main descriptive statistics. These results reveal that the number of messages in issues and pull requests is similar (around 3 messages), thus meaning that contributions always received at least some feedback on average. There seems to be always around two different developers involved in issues and pull requests. This indicates a minimal participatory discussion but it could be that further discussion take place outsdide the GitHub platform. Finally, the time to close issues and pull requests is very different: the time to close issues (10 days and 11 minutes on average) is much higher than the time to close pull requests (1 day and 19 minutes on average). Further analysis revealed that a possible reason for this behavior could be related to clear and concise discussions in the issues, thus facilitating its implementation and eventually a quick acceptance of the corresponding pull request.

Note that all these results are not good or bad in themselves but instead they should also be regarded in relationship with the current structure of the Decidim community. For instance, if a project manages to attract many more external contributors, it will get more pull requests, which is good; but probably the time to close a pull request will get worse since it will take time for the new external contributors to get used to the project dynamics, style, etc.

Finally, we collected the labels used in issues and pull requests and generated a tag cloud, shown in Figure 21. The tag cloud allows us to identify where the effort of the development process is spent. As can be seen, the three most used labels are *in-review*, *helpwelcome* and *hacktoberfest*, being the last two clear indicators of the commitment of the project to attract and welcome new contributors.

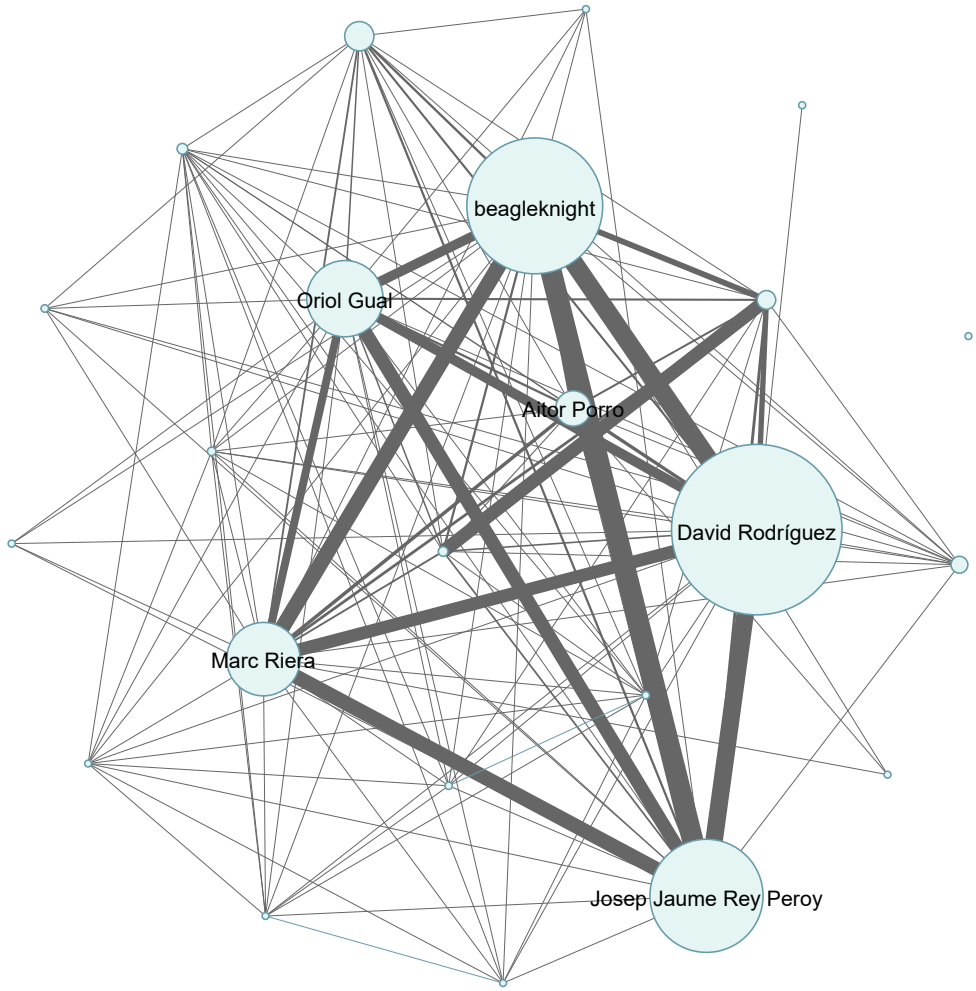
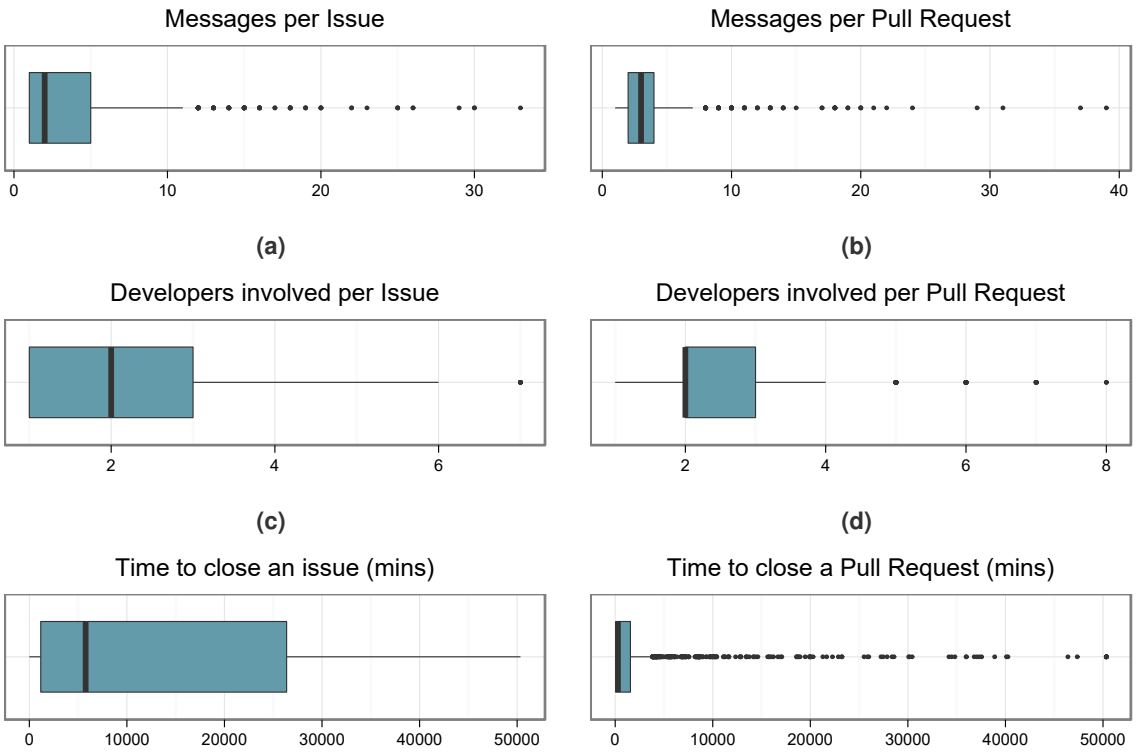


FIGURE 19: COLLABORATION GRAPH FOR THE ISSUES AND PULL REQUESTS IN THE DECIDIM REPOSITORY

4.2.2.3 BUS FACTOR ANALYSIS

Software development projects face a lot of risks (e.g., requirements inflation, poor scheduling, technical problems, etc.). Underestimating those risks may put in danger the project success. One of the most critical risks is the employee turnover, that is the risk of key personnel leaving the project. A good indicator to evaluate this risk is to measure the concentration of information in individual developers. This is also popularly known as the *bus factor* (i.e., *number of key developers who would need to be incapacitated, i.e. hit by a bus, to make a project unable to proceed*) [9] or *pony factor* (in the Apache Software Foundation context).

We have calculated the bus factor in June 2017 for Decidim repositories, obtaining a bus factor value of 3 for committers, including the developers *David Rodríguez*, *beagleknight* and *Josep Jaume Rey Peroy*. We then calculated the evolution of the bus factor along the project lifespan, including both committers and authors. Figure 22 shows the evolution of the bus factor in Decidim. As can be seen, while the bus factor for committers has remained stable (at value 3) since January 2017, the bus factor for authors has increased up to 4 since June 2017. This results reveal that the contributions coming from authors have become an important asset in the Decidim GitHub project.



METRIC	MIN.	1ST QU.	MEDIAN	MEAN	3RD QU.	MAX	ST. DEV.
Messages per Issue	1.000	1.000	2.000	3.906	5.000	33.000	4.411
Developers involved per Issue	1.000	1.000	2.000	2.118	3.000	7.000	1.253
Time to close an issue (mins)	0.067	1,172.000	5,773.000	15,050.000	26,370.000	50,340.000	18,230.583
Messages per Pull Request	1.000	2.000	3.000	3.313	4.000	39.000	2.996
Developers involved per Pull Request	1.000	2.000	2.000	2.504	3.000	8.000	1.009
Time to close a Pull Request (mins)	0.117	56.900	283.100	2,562.000	1,561.000	50,340.000	6,480.195

(g)

FIGURE 20: ISSUE AND PULL REQUEST ANALYSIS.

4.2.3 SOM AND JAM MEETINGS

We attended three meetings to survey, interview and validate the advances of our analysis of the governance of Decidim. These meetings included two SOM and one JAM session, which we describe and present the main insights we gathered in the following.

4.2.3.1 SOM SESSIONS

SOM sessions are hold periodically to discuss about Decidim. The call for session is published in the Metadecidim platform and usually include five axis dealing with (1) research (called *LAB axis*), narrative (called *COM axis*), technology (called *TECH axis*), participant experience (called *PX axis*) and governance (called *GOV axis*). The minutes and main decisions made during the sessions are published online in the Metadecidim platform.

FIGURE 21: TAG CLOUD FOR THE LABELS USED IN ISSUES AND PULL REQUESTS IN DECIDIM.

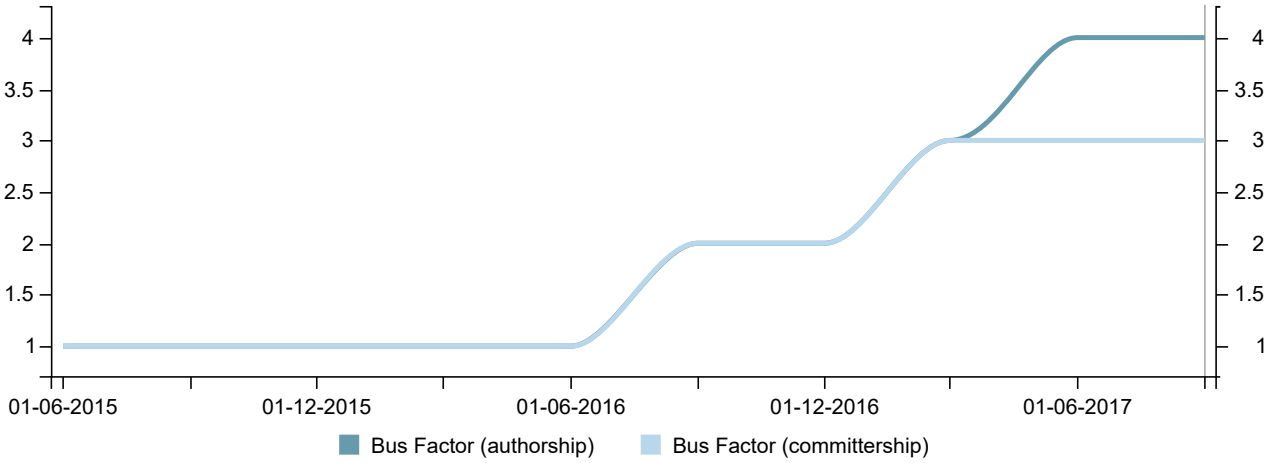


FIGURE 22: BUS FACTOR ANALYSIS FOR DECIDIM.

We attended the *GOV axis* on October, 5^{th} 2017 and on November 20^{th} 2017. In October we had the opportunity to learn and discuss the main architecture of Decidim and Metadecidim platforms. The input gathered in this session helped us to elaborate this document, mainly Chapter 2 and parts of Chapter 4. In November, we provided a first draft of this document and discussed it with the main development teams of the platform with the aim to validate our findings in Chapter 4.

4.2.3.2 JAM CONFERENCE

Once a year, the Decidim platform celebrates the Metadecidim Annual Conference (*Jornadas Anuales de Metadecidim*, JAM, in Spanish), where all members of the community, institutions and users are invited to participate. The objective of the JAM is to design and discuss about participative democracy and how the Decidim platform can help to achieve it.

We attended the JAM'17 conference on November, 26^{th}-28^{th} 2017. This year the conference included several discussion panels, keynotes and parallel sessions. In particular, it featured a parallel session per axis identified in the SOM sessions (cf. Section 4.2.3.1). We attended to the *GOV axis* session, where we presented

an introduction and conducted a workshop on governance in Open Source and Decidim [33].

The workshop was intended to discuss with the participants the way GitHub address the main decision-making points and which governance model (or governance rules) would be the most suitable for Decidim. To this aim, we invited both end-users and developers of the platform to the workshop, thus helping us to collect a rich and varied set of opinions. The discussion was divided into three main categories, corresponding to the main decision-making points in GitHub, namely: (1) issues, (2) pull requests and (3) release. Next we summarize the main insights we gathered in the workshop.

ISSUES The discussion on issues covered two main topics: (1) the need to differentiate between bug and feature, which would help to use the GitHub platform for the former and the Metadecidim platform for the latter; and (2) the deployment of prioritization mechanisms, to help developers to identify the most urgent bugs/features (including sponsorship-based or contract-based solutions).

PULL REQUESTS The discussion about how to govern pull request revealed that most important contributions to the platform are currently coming from the committers of Decidim. In the discussion, three main points arose: (1) it may be needed to define different governance models for issues and pull requests, which is mainly motivated by the fact that bugs and feature requests were managed only in GitHub at that time; (2) the deployment of developer positions to be able to assign tasks (instead of individuals); and (3) the need of publishing better good practices documents. Furthermore, the idea of better prioritization mechanisms was also raised.

RELEASE When discussing about how to make decisions on Decidim releases, participants mainly pointed that it is crucial to differentiate between was is achieved in GitHub and Metadecidim and how to vote each element in these platforms. Two possible mechanisms were discussed to vote: (1) by distributing votes and (2) by relying on some kind of sponsorship solution (as also commented for addressing issues).

The feedback collected in the workshop influenced and refined the governance model presented in this report. In particular, ideas raised for issues and pull requests were incorporated in the governance model (cf. Section 4.3, while insights related to releases were identified as further work (cf. Sections 5 and 6), as they may involve organization changes that go beyond the governance model definition.

4.3 GOVERNANCE MODEL FOR DECIDIM

The analysis of the Decidim GitHub project plus the meetings hold with the Decidim community allowed us to identify the current governance model of the project.

We have formalized the Decidim governance model (see Figure 23) using our DSL (cf. 3.3) (see Figure 23a) and also generated the corresponding verbalization in English (see Figure 23b).

The discovered governance model includes three types of groups of users (i.e., *Developers*, *Product* and *End-users*), for which we also indicate the main individuals. To define these groups and assign people we used the results obtained in the analysis of the GitHub project presented before (cf. Sections 4.2.1 and 4.2.2) plus the feedback obtained in the conducted meetings (cf. Section 4.2.3).

The model includes four governance rules to define how to deal with the main decision-making points in any GitHub project (i.e., issues, pull requests and releases). According to the analysis presented previously, all the rules use a Consensus rule, which means that decisions are made only when full agreement is reached. A special case appears for governance rules dealing with issues, which includes a rule for issues created by the product group (i.e., product owners) of the project (see rule *issuesRuleFromLeaders*) and another one for those issues coming from the rest of users (see rule *issuesRuleFromAnyone*) Furthermore, the deadline set for

[33] All the materials of the session are available at `https://bcnparticipa.cat/decidimjam/es/materials.php`

```
Project Decidim {
  Roles: Committers, Leaders, End-users
  Users:
    Product: XavierE, ArnauM, AndresP, CarolR,
             JosanF
    Developers: JosepJ, OriolG, MarcC, DavidM,
                AndresP, AitorP, JuanS, GenisM,
                DavidR, DanielH
    End-users: anyone
  Deadlines:
    issueDeadline       : 10 days
    pullRequestDeadline : 1 day
  Rules:
    issuesRuleFromLeaders : Consensus {
      applied to Issue (from Product)
      when IssueReview
      people Product
      range Present
      minVotes 1
      deadline waitFor Product
    }
    issuesRuleFromAnyone : Consensus {
      applied to Issue (from not Product)
      when IssueReview
      people Developers
      range Present
      minVotes 2
      deadline issueDeadline
    }
    pullRequestRule : Consensus {
      applied to PullRequest
      when PullRequest
      people Developers
      range Present
      minVotes 2
      deadline pullRequestDeadline
    }
    releaseRule : Consensus {
      applied to PullRequest, Issue
      when Release
      people Developers, Product
      range Present
      minVotes 2
      deadline waitFor Product
    }
}
```

(a)

```
At Decidim, there are three main groups of users: (1)
  product, (2) developers and (3) end-users.
Product group includes XavierE, ArnauM, AndresP,
  CarolR and JosanF, who are involved in the
  decision-making process of releases and issues
  coming from this group.
Committers group includes JosepJ, OriolG, MarcC,
  DavidM, AndresP, AitorP, JuanS, GenisM, DavidR and
  DanielH, who are in the decision-making process of
  issues (not proposed by Product group) and pull
  requests.
End-users group includes anyone willing to contribute
  to the project.

Issues coming from Product group will be accepted by
  themselves, by consensus, with at least one vote
  and as soon as they cast their vote.
Issues coming from others will be reviewed, in 10
  days approximately, by the group of developers, who
  will decide, by consensus, if it deserves to be
  considered in the project. At least two votes have
  to be cast.
Your pull request will be reviewed, in 1 day
  approximately, by the group of developers, who will
  decide, by consensus, if it deserves to be merged
  in the project's codebase. At least two votes have
  to be cast.
The decision to include pull requests in the next
  releases is made by developers and product groups
  of Decidim by consensus. Decisions are firm when
  all product group members have cast their vote.
```

(b)

FIGURE 23: DISCOVERED GOVERNANCE MODEL FOR DECIDIM, EXPRESSED IN (A) OUR DSL AND (B) ENGLISH VERBALIZATION.

issues and pull requests rules has been set to the data reverse engineered previously. The same happens for the minimum number of votes for these rules. The rule regarding the decision-making involved in the release procedure has been configured according to the information gathered in meetings with members of Decidim.

By publishing this governance model in the GitHub repositories anyone willing to contribute will get a clear idea about the development process, the people and the timing of the project. Besides, making public the model also contributes with the Decidim social contract, which advocates for transparency and open content.

5 GOVERNANCE MODEL 2.0

The governance model for GitHub activity described in the previous chapter has been created as a result of the analysis of the Decidim GitHub project and community. Although it describes the current governance model of the project, it has to be validated by the whole Decidim community, which may involve the modification and/or removal of elements. In this chapter we present a methodology to validate the governance model with the Decidim community relying in the Metadecidim platform. Once validated, we also discuss some scenarios to use Metadecidim in the decision-making process of the Decidim platform development.

> The discovered governance model presented in the previous chapter has been obtained after analyzing the activity of the Decidim GitHub project and holding several meeting with the community. The community as a whole should decide whether this is the governance model they want moving forward or it should be changed as part of participatory process.

5.1 GOVERNANCE MODEL 2.0

To validate the current governance model, adapt it or propose a completely new one, the Decidim team will launch a participatory process starting on the Metadecidim platform, which is specially aimed at collaboratively discuss about the Decidim platform development.

In particular, we will propose to discuss the governance model according to the three main decision-making points identified in GitHub projects, namelly: (1) issues, (2) pull requests and (3) releases, thus facilitating the organization of the discussions and the easy engagement of community members, who can participate in those elements where they feel more interested.

The full process will be composed of four phases, which we describe in the following. Figure 24 illustrates these phases.

PHASE 1. DIAGNOSTIC This first phase presents the results of this report and describes the governance model and its rules. The objective of the phase is to introduce a starting point to facilitate the discussing in the community.

PHASE 2. PROPOSALS Proposals are organized according to the decision-making points commented before (i.e., issues, pull requests and release). Each proposal will include a short introduction and then a description of the governance rule.

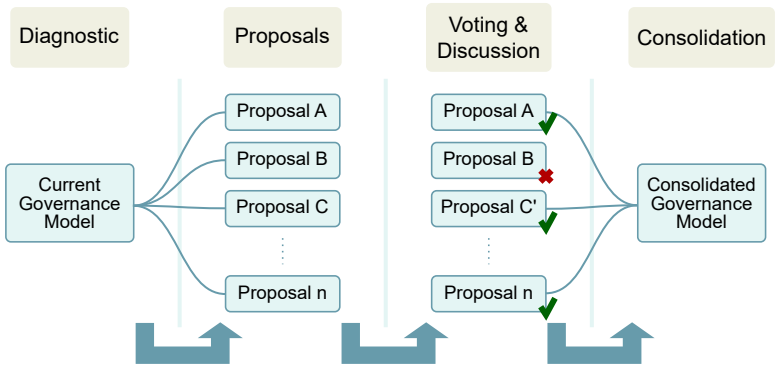


FIGURE 24: PROCESS TO VALIDATE THE GOVERNANCE MODEL IN DECIDIM USING METADECIDIM.

TABLE 5: PROS AND CONS OF THE IDENTIFIED SCENARIOS.

SCENARIO	PROS	CONS
Full serial process	• Every change (issue and pull request) is tracked • Clear traceability between the two platforms	• Overload of the process, as everything (even small bugs) has to pass by Metadecidim • It may cause pollution in Metadecidim • It may discourage external contributors
Parallel process	• Developers can report bugs and fixes directly in GitHub • Easier to attract occasional contributors	• Different governance models (for both the Metadecidim and Decidim platforms) • Risk to introduce inconsistencies between decisions on both platforms

PHASE 3. VOTING AND DISCUSSION Metadecidim community members can vote and discuss on the proposals, which can therefore evolve and incorporate the recommendations made.

PHASE 4. CONSOLIDATION Once the proposals have been validated by the Metadecidim community, they become the final governance model of the Decidim platform. At the end of this phase, the community can be sure that the governance model fulfills the end-users' needs.

5.2 USING METADECIDIM IN THE DEVELOPMENT PROCESS

It is important to keep in mind that the governance model for the Decidim GitHub project is only one part of the whole Governance model for Decidim, which also involves the governance of the discussions on the Metadecidim platform (where similar decisions must be made regarding who, how and when makes decisions on the discussions taking place there). And both models must be aligned to agree on how decisions on one are then transferred to the other one.

We foresee two main scenarios to enable this (the pros and cons of each scenario are shown in Table 5, obviously, between these two "extreme" options, many intermediate approaches are also possible):

FULL SERIAL PROCESS In this scenario, any change request (including bugs and improvements) has to be first notified via Metadecidim platform. Once they have been validated and selected, they pass it on to GitHub platform order to be implemented. No new issues or bugs or proposals can ever be directly created on GitHub.

PARALLEL PROCESS There are two entry points to send requests to the Decidim GitHub project. On the one hand, change requests and proposals are notified via Metadecidim platform and eventually transformed into GitHub issues to be implemented in the Decidim platform. On the other hand, bugs and corrections are notified via GitHub issues to be discussed and considered only at the GitHub level.

5.3 ADDITIONAL CONSIDERATIONS REGARDING THE GOVERNANCE MODEL

An additional option discussed with the Decidim community (cf. Section 4.2.3) was to introduce the role of sponsor entities in the previous scenarios. These entities could push for the development of specific features (or bugs) and therefore bypass any decision-making process proposed before. The main advantage would include the attraction of entities to use the tool, but its main disadvantages could involve a loss of control in the development process (as only those ones with funds can influence it).

Moreover, the question of who can participate and vote needs also some clarification. This is even more important with a sponsorship model but relevant nonetheless. It is important to decide: (1) the number and

requirements of the membership levels (a sponsor is not the same as an individual); and (2) the need of vote privileges, not only to restrict who can (or cannot) vote, but also to specify the weight of the vote according to the role of the user in the platform (e.g., sponsors may have more privileges to vote but also city halls or large organizations could have more power than individual end users).

6 NEXT STEPS

This report focused on the identification and definition of the governance model for Decidim code repositories in GitHub. This section wraps it up by identifing some further work related to (1) improve the integration of Decidim with Metadecidim and (2) the enforcement of governance models. For the realization of both lines of work we propose the deployment of *bots*, software applications specially tailored to run automated tasks ranging from analysis, validation to reporting. Some of these bots could be deployed as chatbots, i.e., bots offering a conversational interface able to talk with users in natural language. This could be especially useful to help non-technical users navigate some of the complexities of the platforms.

> Beyond the definition of the governance model of Decidim GitHub, a better integration at the technical level between Decidim and Metadecidim is still missing no matter how the final governance model defines the relationship between the two. Also, the possibility to enforce the governance model has to be explored and discussed. We propose to use bots to facilitate these tasks.

6.1 METADECIDIM AND DECIDIM INTEGRATION

Further work should be focused on developing the required integration infrastructure between the Decidim GitHub project and Metadecidim, with the main goal of keeping consistency and providing a fluent workflow between the two platforms.

We propose to define and incorporate automatic mechanisms to keep the consistency and fluency of the workflow. These mechanisms would consider the deployment of bots, for instance, a bot in Decidim may open issues in GitHub automatically after agreements at the MetaDecidim level are reached. Also it may bring over also all the related information and the ownership of such proposals so that authors get credit on GitHub as well.

6.2 DEPLOYMENT AND ENFORCEMENT OF GOVERNANCE MODELS

While the current definition of the governance model enables anyone to understand how the project is governed, developers may also want to use the specification of the rules to automatically manage the project itself. In order to do that we would need to: (1) provide support for recording all information regarding the community interactions (e.g., proposals, votes, etc.), and (2) implement a decision engine that can use this information and the governance rules to make the corresponding decisions and update the project status accordingly.

Thus, additionally to our DSL, it would be possible to provide the needed infrastructure to “execute” the DSL specifications so that, beyond improving the understanding of the project internal organization, they can also be enforced to support and guide the collaboration among the project participants.

Figure 25 illustrates how the automatic enforcement of the DSL rules would change the group dynamics at a particular decision point. Given a set of tasks to be discussed (i.e., to accept them, accept a patch for them or include them in a release), users can vote for/against them (step 1), then a decision engine analyzes the votes according to the governances rules defined (e.g., total agreement, simple majority, etc.) (step 2) and, as a result, the status of the tasks is changed based on the decisions taken by the engine (step 3).

In this setting, we believe that bots could also be deployed, for instance, to help to check that rules are being followed, warn the community when they are being skipped or encourage the Decidim community to follow the rules.

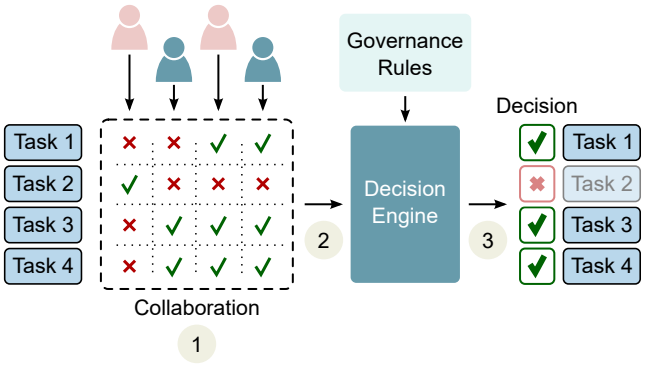


FIGURE 25: DECISION PROCESS PROPOSED.

6.3 MANAGING THE COMMUNITY

Beyond the enforcement of the governance model, bots can also be used in many other community management scenarios. Just to mention a few:

DIVERSITY BOTS, which could be used to detect contribution attempts from newcomers or people from underrepresented groups and tag those contributions accordingly to make sure they get an special attention.

CONTRIBUTION BOTS, which could help people when writing their feature requests or bug report making sure they provide all the relevant information. Incomplete bug reports are typically rejected (since, for instance, they cannot be reproduced) so a good bug report maximizes the chances of getting the attention it deserves.

POLITENESS BOTS, which could use sentiment analysis techniques to identify users that are being aggressive to other users and other similar violations of the project code of conduct and automatically ban them from the platform.

These are just examples but they would definitely help in making Decidim a friendly place to collaborate and make sure Decidim has a healthy community backing it up.

REFERENCES

[1] Affero. Affero General Public License. URL: http://www.affero.org/oagpl.html, last accessed Nov. 2017.

[2] Jorge Aranda and Gina Venolia. The Secret Life of Bugs: Going Past the Errors and Omissions in Software Repositories. In *Int. Conf. on Software Engineering*, pages 298–308, 2009.

[3] X.E. Barandiaran and C. Romero. Funcionalidades y características de decidim. v.1.0, 2017.

[4] Dane Bertram and Amy Voida. Communication, Collaboration, and Bugs: the Social Nature of Issue Tracking in Small, Collocated Teams. In *Int. Conf. on Computer-Supported Cooperative Work*, pages 291–300, 2010.

[5] John D Blischak, Emily R Davenport, and Greg Wilson. A Quick Introduction to Version Control with Git and GitHub. *PLoS computational biology*, 12(1):e1004668, 2016.

[6] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012.

[7] Eoin Ó Conchúir, Pär J. Ågerfalk, Helena H. Olsson, and Brian Fitzgerald. Global Software Development. *Communications of the ACM*, 52(8):127, 2009.

[8] M. E. Conway. How Do Committess Invent? *Datamation*, 14(4):28–31, 1968.

[9] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Assessing the Bus Factor of Git Repositories. In *Int. Conf. on Software Analysis, Evolution, and Reengineering*, pages 499–503, 2015.

[10] Free Software Foundation. GNU Affero General Public License. URL: https://www.gnu.org/licenses/agpl-3.0.en.html, last accessed Nov. 2017.

[11] Free Software Foundation. GNU General Public License. URL: https://www.gnu.org/licenses/gpl.html, last accessed Nov. 2017.

[12] James D. Herbsleb and Rebecca E. Grinter. Splitting the Organization and Integrating the Code: Conway's Law Revisited. In *Int. Conf. on Software Engineering*, pages 85–95, 1999.

[13] Javier Luis Cánovas Izquierdo and Jordi Cabot. Enabling the Definition and Enforcement of Governance Rules in Open Source Systems. In *International Conference on Software Engineering*, pages 505–514, 2015.

[14] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. An in-depth Study of the Promises and Perils of Mining GitHub. *Empirical Software Engineering*, 21(5):2035–2071, 2016.

[15] A. Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison Wesley, 2008.

[16] A Mockus. A Case Study of Open Source Software Development: the Apache Server. In *Int. Conf. on Software Engineering*, pages 263–272, 2000.

[17] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Trans. Soft. Eng. Method.*, 11(3):309–346, 2002.

[18] Eric. S Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 2001.

[19] Charles M Schweik and Robert C English. *Internet Success: A Study of Open-Source Software Commons*. The MIT Press, 2012.

[20] Bianca Shibuya and Tetsuo Tamai. Understanding the Process of Participating in Open Source Communities. In *Int. Conf. on Free/Libre/Open Source Software*, pages 1–6, 2009.

[21] Vandana Singh, DM Nichols, and MB Twidale. Users of Open Source Software: How do They Get Help? In *Hawaii International Conference on System Sciences*, pages 1–10, 2009.

[22] Klaas-Jan Stol and Brian Fitzgerald. Two's Company, Three's a Crowd: a Case Study of Crowdsourcing Software Development. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, pages 187–198. ACM Press, 2014.

[23] Alistair Sutcliffe and Nikolay Mehandjiev. End-user Development: Tools that Empower Users to Create their own Software Solutions. *Communications of the ACM*, 47(9):31, 2004.

[24] Parastou Tourani, Bram Adams, and Alexander Serebrenik. Code of Conduct in Open Source Projects. In *Int. Conf. on Software Analysis, Evolution and Reengineering*, pages 24–33, 2017.

SOM

R&I