ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/303515571

Exploring decision-making processes in Python

Conference Paper · June 2016

DOI: 10.1145/2915970.2915993

CITATIONS
16

READS
1,234

3 authors, including:

Sherlock A. Licorish
University of Otago

130 PUBLICATIONS **1,843** CITATIONS

SEE PROFILE

Bastin Tony Roy Savarimuthu
University of Otago

170 PUBLICATIONS **1,731** CITATIONS

SEE PROFILE

All content following this page was uploaded by Sherlock A. Licorish on 31 December 2017.

The user has requested enhancement of the downloaded file.

Full citation: Keertipati, S., Licorish, S. A. and Savarimuthu, B. T. R. 2016. Exploring decision-making processes in Python, in Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE 2016) (Limerick, Ireland, June 1-3, 2016). ACM, 1-10. 10.1145/2915970.2915993.

Exploring Decision-Making Processes in Python

Smitha Keertipati, Sherlock A. Licorish, Bastin Tony Roy Savarimuthu
Department of Information Science
University of Otago
PO Box 56, Dunedin 9016, New Zealand
{smitha.keertipati, sherlock.licorish, tony.savarimuthu}@otago.ac.nz

ABSTRACT
The process by which norms are developed to become policies, the normative decision-making process, is not often explicit to stakeholders of Open Source Software (OSS) projects. Understanding the normative decision-making process is crucial for members if such projects are to evolve and succeed. In this paper, we investigated aspects of the normative decision-making processes of OSS development through the use of Python Enhancement Proposals (PEPs). We compared extracted process models with those that are advertised by the Python community to evaluate the extent to which those processes overlap. In addition, we assess members' involvement and contribution to these processes. Our work used structural and behavioral analysis techniques, and social network analysis metrics. We found that there were differences between the extracted processes and Python's advertised process, with the extracted processes being significantly more complex. These differences also extended to granular models used for managing social and technical aspects of the Python project. Furthermore, some key members were largely responsible for PEPs' success. Our extracted models could go a far way in helping the Python community to quickly understand decision-making processes in Python.

Keywords
Open source software development; Governance; Norms; Normative decision-making processes; Developers' involvement; Social network analysis; Python enhancement proposals

1. INTRODUCTION
OSS such as Linux (linux.com) and Mozilla Firefox (mozilla.org) are promising alternatives to Closed Source Software (CSS) as they offer low-cost, high-quality, and feature-rich solutions [1,2]. Thus, it is important to keep users interested in contributing to OSS projects. However, new members wanting to join and contribute to OSS communities may find it difficult to do so because information on how to join and successfully contribute to such projects, and how decisions are made regarding new functionality to be developed, are seldom available. By studying the process by which decisions are made, we can advance the current knowledge of how norms are developed and used in OSS projects, particularly in the governance of software development processes.

In considering the normative decision-making process, *norms* are used to characterize typical or customary behaviors [3]. They are expectations of behavior in society and can be adopted in a number of different ways [4]. Norms can emerge formally where behavioral expectations are explicitly described and implemented by groups, such as OSS communities [5]. In fact, "a community is said to have a particular norm, if a behavior is expected of the individual members of the community and there are approvals and disapprovals for norm abidance and violations, respectively" [6]. Researchers have argued that norms are the basis by which modern socio-technical systems should be governed [7]. Open Source Software Development (OSSD) is an example of a socio-technical system [8] where individuals interact socially with others in the context of software development and also with the software system (technical system) that is being developed. Thus, in OSSD, there are two types of norms, *social norms*[1], which specify the standard practice agreed upon for an issue at hand (how individuals interact), and *technical norms*[2] which are related to the functionality of modules (how software is developed).

Beyond OSSD, norms are important in an organizational setting since they facilitate and maintain social order [9,10], and reduce the amount of individual computation [11]. In the context of OSSD, as norms emerge and become more embedded within an OSS project they are more likely to influence the development process of that project [12]. Thus, knowing the normative decision-making process can allow stakeholders of a project to make informed decisions regarding development initiatives. While previous work has investigated what types of norms exist in OSS through norm mining approaches [5], seldom have the normative decision-making processes been evaluated. In fact, OSS advertised processes may not be the same as the processes that are actually enacted; thus, misleading a community of members and negatively impacting project productivity. Knowledge of how norms are actually created, and by which individuals and groups, could also be useful for OSS communities in terms of aiding new members that are willing to assist.

This work has contributed towards the understanding of the normative decision-making process in OSSD, using Python[3] as a case study. We examine PEPs[4], design documents which provide

[1] For example, a voting procedure where individuals vote on whether a new module should be developed (choice indicated using +1), should not be implemented (using -1), or if they are unsure (using 0).

[2] For example, module X must not use the recently deprecated API.

[3] https://www.python.org

[4] https://www.python.org/dev/peps

information about new features or processes, in comparing Python's extracted processes with those that are advertised by the Python community. We evaluate the extent to which those processes overlap, and how members' contribute to these processes.

The remaining sections of this paper are structured as follows: we provide the study background and research questions in Section 2. We present our study setting in Section 3, and results in Section 4. We then discuss our findings in Section 5, and outline their implications in Section 6. We consider threats to our outcomes in Section 7, and finally, provide concluding remarks in Section 8.

2. BACKGROUND

2.1 OSS Project Governance

A governance model in OSSD aims to provide a social framework for collaboration based on a set of commonly agreed practices that brings disparate individuals together towards achieving a common goal. There are several types of governance models discussed in OSSD literature [13,14]. One is the *benevolent dictatorship* model, which is a hierarchy-based system where a benevolent dictator makes the final decisions and acts as a facilitator between various stakeholders. The dictator could be either permanent or can be changed on a rotation basis (called the *rotating dictatorship* model). An example of a benevolent permanent dictator is Linus Torvalds of the Linux kernel project [15]. Perl is said to employ the rotating dictatorship model [16], though Larry Wall is sometimes assessed as a benevolent dictator. Another model is the *meritocracy* model, which is based on the pure merits of those contributing to the project, allowing them to work their way up and gain more responsibilities. This model is used by the Apache Software Foundation [17]. Exploring these models of governance could provide insights into the normative decision-making processes in OSS.

Our observations of the well-known OSS projects such as Linux[5] and Eclipse[6], show that the process of how decisions are made regarding new functionality to be developed (i.e., technical norms) and the social processes (i.e., social norms) surrounding the development of software are not made public by these projects. Though data is available for these OSS projects, there are gaps in the data such as lack of details about who did what (i.e., who made what decision), and when. The data that is available for Python, however, does not contain any gaps[7]. For this reason, we have selected Python as our case study. In addition, the Python community makes the prescribed process very clear and publicly available. We pursue two directions in our investigation of Python as considered in the remaining subsections: (1) normative decision-making processes, and (2) developers' involvement in the normative decision-making process.

2.2 Decision-Making in OSS

Our first objective in this work is to determine whether the extracted decision-making process is compliant with the prescribed decision-making process, in the context of PEPs. By decision-making processes, we are referring to the processes by which decisions are made. Our work here relates to the literature on decision theory [18,19], which has two main branches, normative decision theory, which studies how decisions *should be* made, and descriptive decision theory, which studies how decisions *are actually* made. This work provides direct evidence on whether discrepancies exist in the real-world between '*should be*' and '*as is*' decision-making processes in a new context (i.e., the decision-making processes in OSS, as evident in repositories). In this paper we investigate the difference between normative decision-making (i.e., the prescribed processes) and descriptive decision-making (i.e., the extracted processes).

The process of decision-making in organizations, where the stakes are considerable and the impact is widespread, is complicated and very important [20]. Decision-making plays an important role in the functioning of an organization [21], where individual, groups, teams and committees must work together to deliver solutions [22]. There are many benefits of group decision-making, including the availability of more knowledge and expertise to solve the problem at hand, the exploration of a greater number of alternative solutions, better understanding and acceptance of the final decision by all group members, and more commitment among all group members to make the final decision work [23]. However, there are also many downsides to group decision-making, including groupthink [24] and group polarization [25].

While other work has investigated what types of norms exist in OSS through norm mining approaches [5], seldom have the normative decision-making processes been studied. In this work, we study these processes using data obtained from the Python repository in answering the first research question:

***RQ1**. Do Python extracted decision-making processes comply with their advertised decision-making process?*

2.3 Developers' Involvement

Our second objective is to extract knowledge about how norms are actually created, and by which individuals and groups. In achieving this objective we aim to compare differences in developers' involvement in the normative decision-making process to identify individuals and groups that are more influential than others in creating decision-making processes.

Studies have considered the involvement of software developers in group decision-making and team dynamics. Crowston et al. [26] examined the work of the developers of five small OSS projects and found that the core groups of developers comprised only a small number of those contributing to the projects. The related study by Crowston and Howison [27] found some OSS projects to be highly centralized (with just a few members communicating), and this pattern was especially pronounced for smaller projects. Additionally, it was revealed that most OSS projects had a hierarchical social structure [27]. Licorish and MacDonell [28] studied team dynamics by investigating how the contribution of members in global software development affected their teams' knowledge diffusion process, and how their personality profiles related to their dominant presence. While also observing few members to dominate group work, they found that the members who exhibited more openness to experience, agreeableness, and extroversion were more inclined to be the influential members of their teams. Their prior work also progressed that of Crowston and Howison [27], by examining the true role of core members, finding that they contribute towards both social and task-related aspects during development [29]. These works show that regardless of the OSS project, few members tend to occupy the core of team interactions. Given the three models that are typically observed in OSS projects (c.f., *benevolent dictatorship, rotating dictatorship* and *meritocracy)*,

[5] http://www.linuxfoundation.org/content/how-participate-linux-community

[6] http://eclipse.org/eclipse/development/

[7] In terms of the authors, the states, etc., of PEPs from when they are first proposed to their end state.

we are interested in understanding if this central/core pattern also exists in the shaping of team norms and proposals.

Given the pattern above for core members, and particularly, its replication in multiple OSS projects [26,27], we anticipate that a specific group of members may shape team norms (and decision-making) in such an environment. To this end, new members joining an OSS project may benefit from coming into contact with these individuals. Although several other works have studied developers' involvement in the decision-making process (e.g., [29]), previous work did not consider members' involvement in normative decision-making processes. We answer our second research question in addressing the above objective:

RQ2*. Who are the most influential and successful Python decision-makers?*

3. STUDY SETTING

As noted above, we examine PEPs taken from the Python repository. A total of 363 proposals were extracted covering three forms of PEPs, Process, Informational, and Standards Track. The basic details of the PEP include the PEP *id*, *title* (description of what the PEP is about), *authors*, *status* (draft, accepted, finalized etc.), *type* (Process, Informational, Standard Track), *creation date* and *modification history*.

To investigate processes compliance (RQ1), a data-driven bottom up approach was used in order to extract the normative decision-making processes from these publicly available Python artifacts. These processes were extracted by using process mining techniques [30]. Process mining is used for Business Activity Monitoring (BAM) and Business (Process) Intelligence (BPI) [31]. It aims at extracting processes within an organization based on logs available in different forms such as process execution logs (e.g., event logs, state logs) that capture who did what, and when, in a particular context (e.g., handling an insurance claim).

Our data for inferring the decision-making process was sourced from the publicly archived versions of the PEP document files as they underwent changes over a period of time (starting with the creation of the first PEP on 13th June 2000, to 31st December 2014). Extracting the knowledge from PEP document files involved a number of steps. The PEP diff files[8] for all the PEP commits were first retrieved from the GitHub repository[9] (step 1). Thereafter, in step 2 the status changes in each diff file were extracted by parsing regular expressions. The new status had the pattern "+status: X", where the text that follows the colon (X in this case) is the state (or event). In step 3 the status changes of all versions of all the PEPs were recorded and stored as event logs. Sample event logs are shown in Table 1; for example, PEP 201[10] was initially drafted, then accepted, and then finalized. In the fourth step the process mining tool Disco[11] was used to construct process diagrams from the imported log files. These diagrams used Simple Precedence Diagram (SPD) notation [32]. In step 5 we analyzed our results and computed frequency, proportions, and percentage difference, which was followed by formal statistical analysis. Finally, these results were then interpreted.

[8] A diff file shows the difference between two text files. For example, a newly created PEP document might be in the *draft* state. After discussions, the status might be changed to the *active* state. The diff file in this case will highlight that there was a change in status (i.e., from *draft* to *active*).

[9] https://github.com/python/PEPs

[10] https://www.python.org/dev/peps/pep-0201/

[11] https://fluxicon.com/disco/

To answer the second research question (RQ2), we employed social network analysis techniques, using PEPs authors' contribution information and modification history. We outline our specific measures in the following two subsections.

Table 1. Sample event logs corresponding to various PEPs

PEP id	PEP type	Event-Transition log
2	Process	Draft-Deferred-Draft-Active-Final
160	Informational	Incomplete-Complete-Finished-Final
201	Standards Track	Draft-Accepted-Final

3.1 Measuring Compliance (RQ1)

As noted above, we used the process mining tool Disco to produce SPDs. Disco allowed us to automatically create smart flow diagrams (or process maps) of processes using event logs. Using Disco, we produced four SPDs, one for each of (1) the overall extracted process, (2) the extracted process for Process PEPs, (3) the extracted process for Informational PEPs, and (4) the extracted process for Standards Track PEPs. We then manually assessed a sample of these outputs for accuracy, which confirmed that the tool functioned correctly. When creating these SPDs, we considered only those PEPs that were initially proposed after October 29, 2005, as the prescribed process model was made publicly available by the Python community[12] at this time. We measured process compliance by comparing the extracted processes against the prescribed process. We examined the differences between the extracted process for all PEPs and the extracted process for each of the three types of PEPs (Process, Informational and Standards). Comparisons were done using two types of analyses, *structural analysis*, which aims at comparing the structures of processes using *comparative structural analysis* to study differences in structures (e.g., if a network diagram has 5 nodes and 4 links connecting the nodes, and another diagram has 7 nodes and 3 links; comparing the number of nodes and the links between the nodes in the diagrams constitutes structural analysis), and *behavioral analysis*, which aims at investigating the patterns of behavior as exhibited by process instances (i.e., the process models) [34].

While structural analysis compares static structures, behavioral analysis focuses on the dynamic aspects. Behavioral analysis facilitates the identification of key elements in the process model (e.g., nodes, pathways or loops), based on the data from all process instances. To study the dynamic behavior, we conducted *frequency analysis* (i.e., we studied the number of times a particular node or state was visited in order to see what nodes were visited most frequently). We also examined the completion times of PEPs traversing different pathways.

Comparative structural analysis as used in this work is an approach widely used in biology and chemistry to compare and analyze structures of two or more entities such as viruses and enzymes [33]. It is also used in the business process management community to compare the structures of business processes [34]. As mentioned above, this type of analysis allows us to study the differences in the structures of processes. In computing such differences domain specific metrics or heuristics such as SPDs have been widely used. The normative decision-making process models in our approach have also been created using SPDs [32], a variant of precedence networks [35], which are commonly used in project scheduling, an important aspect in project management. We used these techniques in our evaluation of

[12] https://github.com/python/peps/blob/master/pep-0001-1.png

compliance. Our outcomes from these analyses are provided in Section 4.1.

3.2 Measuring Involvement (RQ2)

We used social network analysis (SNA) to model developers' involvement in the normative decision-making process, in the context of PEPs. Social network analysis allows us to understand specific workflows, as well as the specific types of individuals and the roles they play in the decision-making process and successful development of PEPs [28,38]. We investigated the roles of individual authors, such as those who are involved in the managing of the Python project or those who are involved in contributing and writing code. The roles of these individuals can range from being the primary contributor of a PEP proposal to revising a PEP proposal that has been proposed by another individual or group of authors. We used NodeXL[13], a free open source network analysis and visualization software package to visualize the network of Python authors, in providing preliminary understanding of members' involvement.

In large graphs, such as the network of Python authors, graph clustering analysis provides a utility for meaningful scientific data analysis [36]. This technique divides graphs into groups, called clusters, whose vertices are highly connected inside each cluster. By using graph clustering, we can discover the structures and representative examples present in the raw graph data. Using NodeXL, we thus cluster Python members based on their participation on PEPs using Clauset-Newman-Moore (CNM) algorithm [37]. We used the CNM algorithm as it is a greedy modularity-based[14] algorithm, which obtains a modularity gain after merging a pair of nodes, and uses nested heap structures of modularity gain for all pairs of nodes. It iteratively selects and merges the best pair of nodes, which has the largest modularity gain, from the heap until no pairs improve the modularity [36,37]. This process allowed us to measure members' influence in the network.

We further studied the influence of certain individuals by looking at their *degree centrality* measure. Network centrality is used in the analysis of structural characteristics of social networks, and can determine the relative importance of nodes in a network [38]. This measure was fittingly used here as it allowed us to model the number of links a node has to other nodes [28], in further probing influence. We used the *degree centrality* measure to identify the individuals who have contributed to the most number of PEPs. We assessed successful members based on the degree to which the PEPs they were involved with reached the *final* state. Given that the direction of contribution was not relevant in this study we used undirected graphs. The results for this aspect of our analyses are presented in Section 4.2.

4. RESULTS

In this section, we discuss the results for the two research questions presented in Sections 2.2 and 2.3. We present our findings on normative decision-making process compliance in the context of PEPs, and show that there are indeed differences between the extracted normative decision-making processes and the advertised (or prescribed) normative decision-making process (RQ1). We next present our findings on developers' involvement in the normative decision-making process, and show that there are individuals and groups who are more influential and successful in Python decision-making than others (RQ2).

[13] http://nodexl.codeplex.com/

[14] Modularity evaluates the density of edges within clusters as compared to edges between clusters. Higher modularity scores result in better clustering results.

4.1 Process Compliance (RQ1)

First, we provide evidence to determine the extent to which the overall extracted process for PEPs is compliant with the advertised process. This comparison allows us to answer RQ1; however, we go one step further in conducting additional analyses aimed at understanding the PEP processes. We investigate whether there are differences between the normative decision-making processes of the three types of PEPs, Process, Informational, and Standards Track. Thereafter, we compare the pathways and completion times of PEPs. Given that our analysis was performed on PEPs proposed after October 29, 2005, our dataset for this phase of analysis comprised 190 PEPs. Overall, there were a total of 33 Process, 45 Informational, and 285 Standards Track PEPs (363 altogether). By considering only those PEPs proposed after the date mentioned, our new dataset comprised 18, 21, and 151 PEPs respectively (and 190 in total). As noted above, we considered a successful PEP to be one whose end state was *final*, and a failed (or unsuccessful) PEP to be one whose end state was either *rejected* or *withdrawn*.

Extracted versus Prescribed Process

The overall extracted process model for PEPs is shown in Figure 1. The numbers in the boxes (or states) indicate the number of times that specific state was visited (e.g., the state *final* was visited 95 times). Also, the numbers beside the arrows indicate the number of times that specific transition (i.e., from one state to another) occurred (e.g., the transition *draft→active* occurred 8 times). Finally, the thickness of the arrows indicates their weight on a log scale (e.g., the transition *draft→accepted* occurred 83 times, and the most, so it has the thickest arrow between the two states).

The prescribed process model is shown in Figure 2. This is the model that Python developers have made available to the community as the one that is being followed during PEP development (refer to: python.org/dev/peps/pep-0001). Looking at Figure 2, it can be seen that there are differences between this model and the extracted process model, as the extracted process captures aspects that are missing in the prescribed process. The extracted process model appears to be more complex than the prescribed process model. It not only contains more nodes (i.e., *approved*, *finished*, and *superseded*) than the prescribed process (with a percentage difference of 31.6%), but the extracted process also contains more pathways (120% difference). This comparison is shown in Table 2. The higher number of nodes and pathways in the extracted process indicates that there are more paths that a proposed change to the Python language can take before it reaches its end state, than the ones prescribed by the Python community. There are also many loops (9 loops as shown in Table 2, with a percentage difference of 160%) that can be seen in the extracted process (for example, the pathway *final→draft→final*), between pairs of states. This differs from the prescribed process which shows only one possible loop (i.e., the loop between *draft* and *deferred*). It can also be seen in Figure 1 that once a PEP has moved to the next state (i.e., from *final* to *deferred*), it can go back and forth between these states or any combination of two states (e.g., *draft→deferred→draft→active→final*). This is not represented in the prescribed process in Figure 2. Also, the purpose of the dashed arrows in the prescribed process is unclear as neither of these transitions (i.e., *accepted→rejected* and *final→replaced*) appear in the extracted process. Overall, comparing Figures 1 and 2, there are several differences in the visual representations of the two processes. The prescribed process appears to be only a simplified model of the inferred decision-making process.

We formally tested the metric with the least percentage difference in Table 2 (i.e., nodes) to see if these differences were statistically significant. We anticipated that if there were statistically significant differences in the measures for prescribed and extracted nodes, then a similar pattern of result would obtain for prescribed and extracted pathways and loops (given the 120% and 160% divergence for these metrics, compared to just 31.6% for nodes in Table 2). We first evaluated normality. Our standardized Skewness and Kurtosis coefficients were both within the boundaries of normally distributed data (i.e., -3 to +3). Thus, the parametric independent sample *t*-test was conducted to test the mean nodes for significant differences, which revealed statistically significant difference ($p < 0.01$).

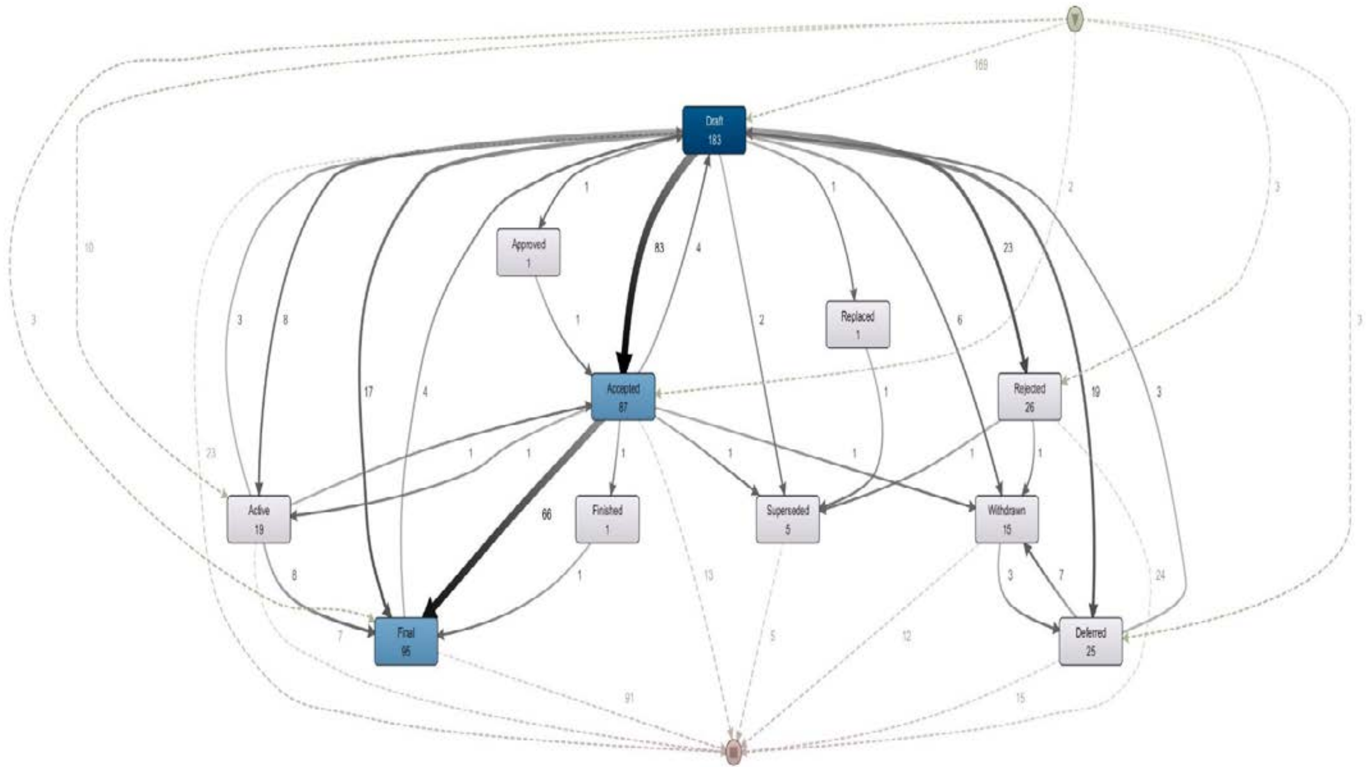

Figure 1. The overall extracted process model for PEPs

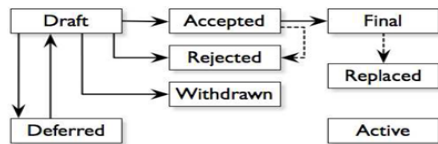

Figure 2. The prescribed process model for PEPs

Table 2. Number of nodes, pathways and loops for prescribed and extracted processes for PEPs

	Prescribed process	Extracted process	Percentage difference
Nodes	8	11	31.6
Pathways	9	36	120
Loops	1	9	160

We next constructed the process models (i.e., the extracted processes) for the three individual types of PEPs in comparing these to the overall extracted process (Figure 1). In the overall extracted model (Figure 1), it is shown that there are some nodes that are visited more frequently than others (e.g., *draft*). This is consistent in the extracted models for Process PEPs (shown in Figure 3), Informational PEPs (shown in Figure 4), and Standards Tracks PEPs (shown in Figure 5). In the overall extracted process model, there are also some pathways that are taken more frequently than others (e.g., *draft→accepted→final*). This is the same for Standards Track PEPs. However, this is different for Process and Informational PEPs as the most frequent pathway for both of these types of PEPs is *draft→active→final*. Though they are not the same, these two pathways are very similar as they have the same start node (i.e., *draft*) and end node (i.e., *final*).

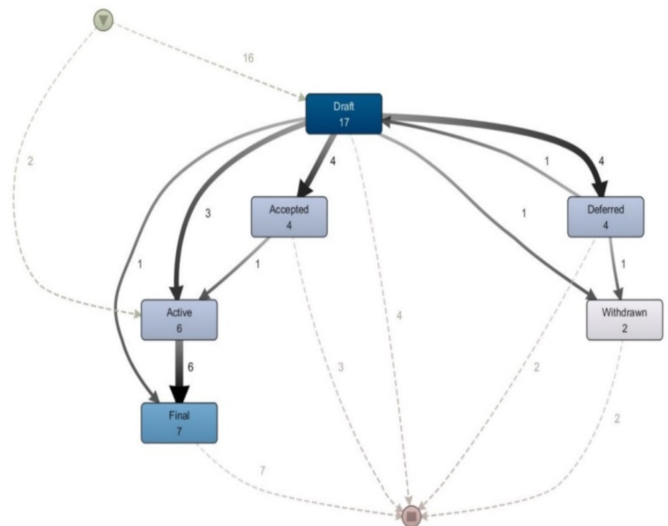

Figure 3. The extracted process model for Process PEPs

Looking at Figure 4, we can also see that no Informational PEPs that reach the *accepted* state have ever reached the *final* state. That said, structurally, the overall extracted process is more like the model for Standards Track PEPs, than the models for the Process and Informational PEPs. This comparison can be seen in Table 3. In terms of the number of nodes, pathways, and loops, the overall extracted process and the extracted process for Standards Track PEPs are similar, while the extracted process for Process PEPs and the extracted process for Informational PEPs are similar to each other. One reason for this may be that Process and Informational PEPs relate to the social aspects of Python (e.g., deciding how to vote for a module), while Standards Track PEPs relate to the technical aspects (e.g., deciding on a technology for implementation).

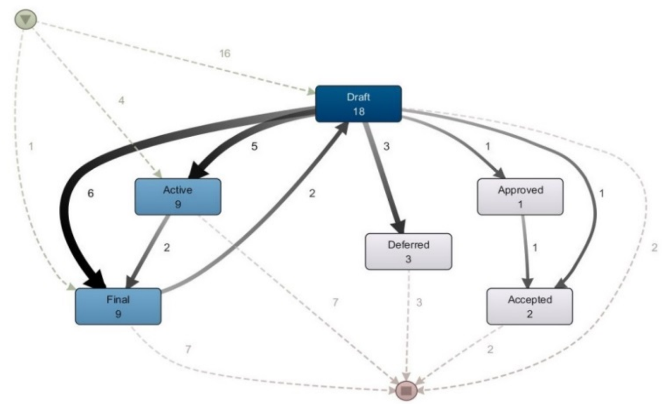

Figure 4. The extracted process model for Informational PEPs

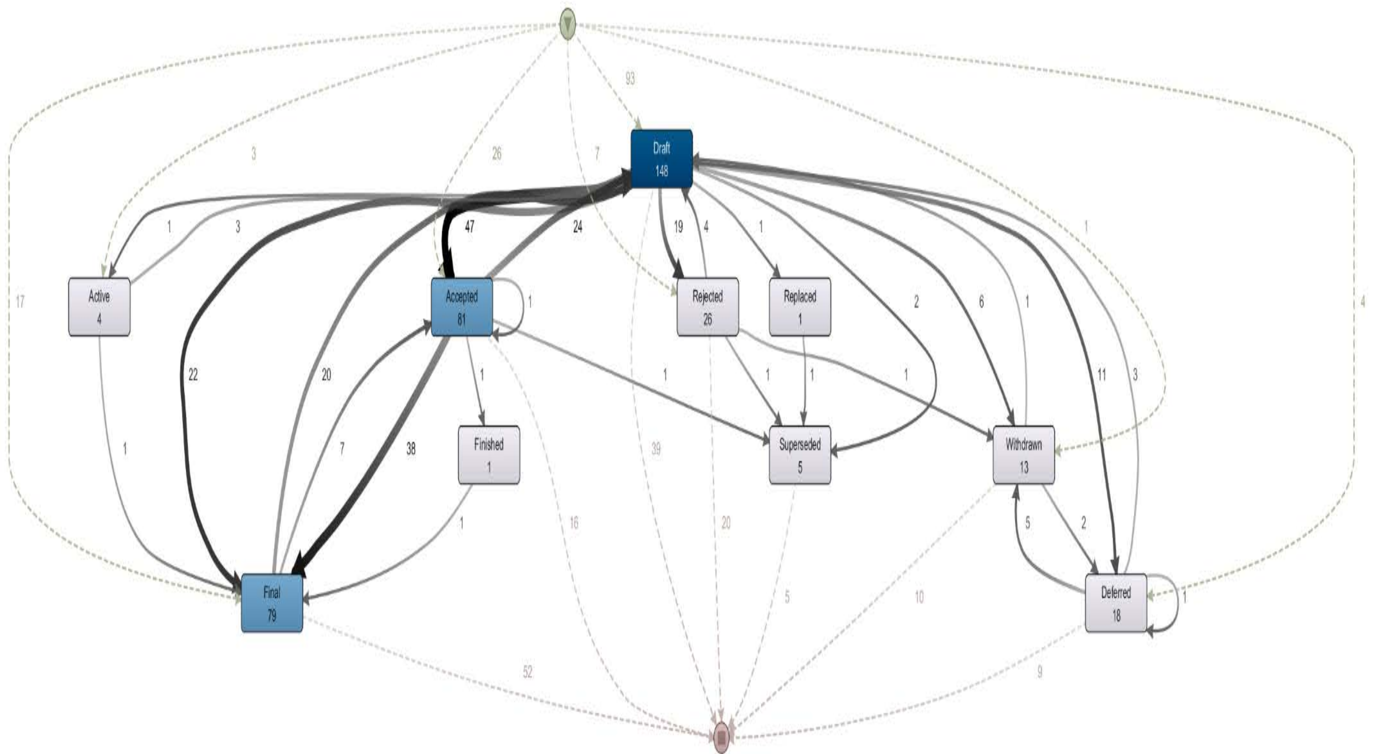

Figure 5. The extracted process model for Standards Track PEPs

Table 3. Number of nodes, pathways and loops for the overall extracted process and the extracted processes for the three types of PEPs

	Process PEPs	Informational PEPs	Standards Track PEPs	Overall extracted process
Nodes	6	6	10	11
Pathways	10	10	27	36
Loops	1	1	7	9

Unique Pathways and Completion Times

We next examined the frequency of the unique pathways of all PEPs, which revealed that some pathways were taken (or visited) more frequently than others. However, most of the pathways were taken only once or twice, especially the ones that go through multiple states (e.g., *draft→rejected→withdrawn*). The top three pathways that were widely used are *draft→accepted→final* (with a count of 76), *draft→rejected* (a count of 60), and *draft→final* (a count of 47). The pathways that were taken more frequently appear to consist of similar states or nodes, such as *draft*, while the pathways that are taken less frequently possess the *incomplete* and *withdrawn* nodes.

We compare the completion time of pathways resulting in a negative outcome (i.e., the end state being *rejected* or *withdrawn*) and the pathways resulting in a positive outcome (i.e., the end state being *final*) in Figure 6. Here it is shown that pathways resulting in a negative outcome take, on average, more time than pathways resulting in a positive outcome (ignoring the outliers in both the negative and positive outcome pathways). Formal statistical testing confirmed (*t*-test result) statistically significant difference ($p < 0.01$), suggesting that *perhaps* negative outcome PEPs needed to be revised more thoroughly before a final decision was made. In examining such outcomes for the individual PEPs, we observed that two of the pathways (i.e.,

draft→accepted and *draft→deferred*), reach completion faster in Process PEPs than in Informational or Standards Track PEPs. Similarly, two more of the pathways (i.e., *active→final* and *draft→active→final*) reach their end state faster in Standards Track PEPs than they did for the other two types of PEPs. It was also observed that one pathway (i.e., *draft→final*) had a shorter mean completion time in Informational PEPs than in Process or Standards Track PEPs. These observations can arise for a number of reasons, including the fact that these pathways are more frequent in one type of PEP than in the others. This would affect the average time taken for that pathway in one of the three types of PEPs. This is true for the *draft→active→final* and *draft→deferred* pathways for Process and Standards Track PEPs, respectively.

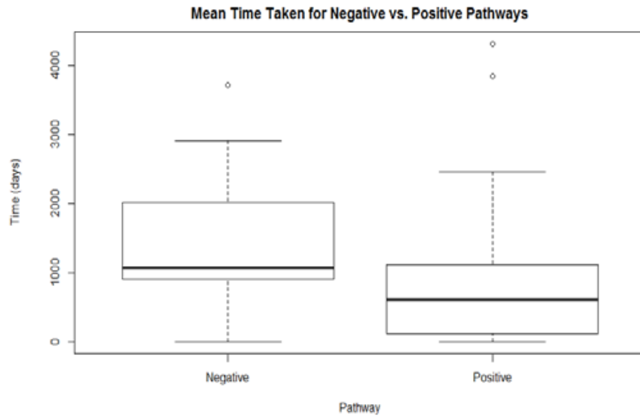

Figure 6. Average time taken for negative (i.e., *rejected*, *withdrawn*) and positive outcomes (i.e., *final*)

4.2 Members' Involvement (RQ2)

Altogether, we identified 142 unique authors (or contributors); 46 authors were involved in Process and Informational PEPs (i.e., the social aspects of Python), while the majority of developers (133 authors) were involved in Standards Track PEPs (i.e., the technical aspects). In fact, some authors who were involved in Standards Track PEPs were also involved in Process and Informational PEPs. There were three authors who were only involved in Process PEPs, and six authors who were only involved in Informational PEPs. We also identified nine authors who were involved in all three types of PEPs.

In investigating influential authors in the Python community we visualized and clustered all the authors into groups using the CNM clustering algorithm [37]. Figure 7 reveals eight clusters of membership, with the bottom right cluster comprising only two members. These members made very little contribution to the PEPs. We have thus focused on the seven densely connected clusters here in our results. We observe that in each of these seven clusters, there was an individual who stood out, or played the hub (or core member). Further analysis revealed that five of these seven authors were involved in all three types of PEPs. To determine the extent to which these seven individuals were influential, we identified the total number of PEPs they were involved in (i.e., contributed to). These results are shown in Table 4. Here it is shown that C1 (Guido van Rossum) was the most influential individual. He contributed to the most number of PEPs (95 out of 363 PEPs in total). This is not surprising as he is the original creator of, and the final design authority for the Python programming language. Other significant contributors to PEPs were C2, C3, C4, and C6. These members are also occupying central position in the network in Figure 7. We observe that C1, C3, C6, and C8 are indeed PEP editors, and so their pronounced presence is fitting. PEP editors are individuals who are responsible for managing the administrative and editorial aspects of the PEP workflow (e.g., assigning PEP numbers and changing their status).

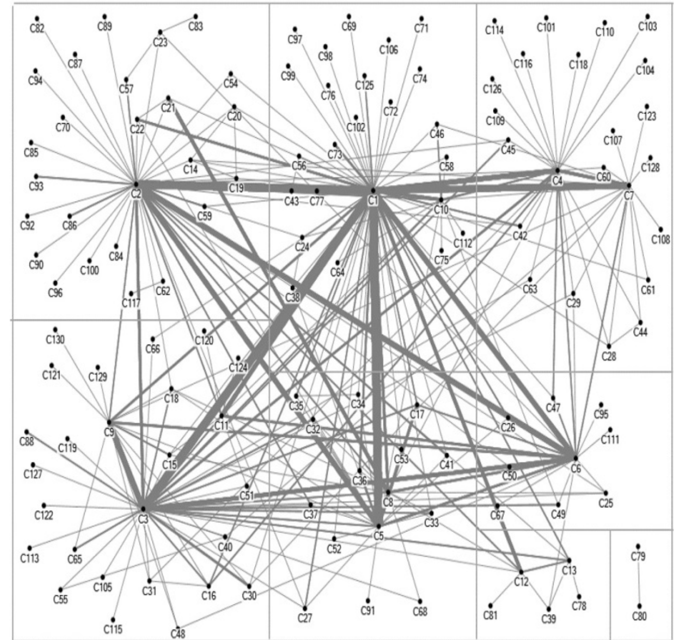

Figure 7. CNM cluster of Python members

Table 4. Degrees centrality of Python seven key members

Author	Degree centrality
C1	95
C2	75
C3	73
C4	33
C6	30
C8	23
C13	11

For the seven individuals identified as most influential in the network by the CNM clustering algorithm (shown in Figure 7), we found the number of successful and unsuccessful PEPs each individual was involved in (i.e., contributed towards), as either the *primary contributor*[15] or a *secondary contributor*[16]. We also calculated the proportion of success for each individual. These results are shown in Table 5. Here it is observed that, though C1 had the highest number of successful PEPs, his proportion of success is relatively low (i.e., just over 50%) when compared to that of the others. The individual with the highest proportion of success is C8 (with 17 successful PEPs out of 23 PEPs in total). This does not necessarily indicate that C8 is the most influential individual in terms of success. To get a more accurate picture we further examined the outcomes for only the top three authors in terms of the number of PEPs they have been involved in, C1, C3, and C2, and found that C3 had the highest proportion of success (with 46 successful PEPs out of 73 PEPs in total), followed by

[15] A primary contributor is characterized as the first author (i.e., initially proposed the PEP), the only author involved in the PEP, the author who contributed the most to the PEP, or any combination of these.

[16] A secondary contributor is characterized as an author who joined the PEP at a later stage, and does not fit the criteria of a primary contributor.

C1 (51 out of 95), and then C2 (34 out of 75). We considered only these three authors as they were involved in a relatively similar number of PEPs. Overall, C4 was the only author, out of the seven, who was involved in more unsuccessful PEPs (with a count of 18 out of 33) than successful PEPs (count of 13).

Table 5. Key members' presence on successful and unsuccessful PEPs

Author	Successful PEPs	Unsuccessful PEPs	Proportion of Success[17]
C1	51	30	0.54
C3	46	13	0.63
C2	34	12	0.45
C8	17	3	0.74
C6	17	11	0.57
C4	13	18	0.39
C13	7	1	0.64

5. DISCUSSION

We revisit our research questions in this section. We first analyze our outcomes in answering RQ1 in Section 5.1, before considering our results in answering RQ2 in Section 5.2.

5.1 Process Compliance (RQ1)

RQ1. Do Python extracted decision-making processes comply with their advertised decision-making process? Our results confirmed that there were divergence in the extracted normative decision-making processes and the prescribed normative decision-making process as publicized by the Python community. We observed that these differences also existed at a more granular level, for the different types of python enhancement proposals. **Thus, in the Python project there is a difference between "as-is" versus "should-be", when considering decision-making processes.** One potential reason for this is that it is extremely difficult to capture software development decision-making processes, and present them in a simple and accurate representation [2]. This is because there would be several decision-making processes involved in the software development process that are not succinctly specified at project conception, owing to the evolving nature of the software development process [39]. Such processes would tend to change depending on the realities of the software ecosystem. Thus, capturing such processes fully would require that the processes are documented in a way that there are no gaps in the information. Therein lays the opportunity for mechanisms to then update process models without human intervention. Automated tools could aid this process, and may be particularly necessary for OSS environment where individuals volunteer [1], and may find little incentive in recreating decision-making models. That said, such models are necessary as the divergence noted could have implication for the performance of those involved in the community, whose expectation could be violated in this context. Such violations could lead to frustration and members quitting a software project. In addition, delays may also result in members having to verify those incorrect processes that are publicized. Furthermore, new members wanting to contribute may find it difficult to do so.

Our outcomes here advance those that previously assess the types of norms that are prevalent in OSS projects [5]. In fact, investigating the prescribed versus extracted (or enacted) process is important, not only for OSS project contributors, but also for the project management or team leadership to comprehend the complexity of the real process (i.e., extracted process) and the dissonance between the extracted and prescribed processes. Insights from such validations would be beneficial for both OSS and CSS environments, in informing process (re)engineering activities.

We observed several paths that a proposed change to the Python language can take before it reaches its end state. Notwithstanding differences in the number of observations in Section 4, we also noticed that Process and Informational PEPs (those related to the social aspects of Python) differed to those of a Standards track PEPs (technical aspects). This evidence may indicate that PEPs governing technical aspects of Python possessed much more complexity than social aspects, which were agreed upon much sooner. So, for example, 'agreements on a proposal around how long members of a technical review team should take to respond to a newly integrated software function' (a social aspect) was reached much faster say than a 'proposal specifying the actual platform to be used for executing builds' (a technical aspect). This pattern is fitting, as previous work noted that such (technical) mechanisms aimed at automating repetitive tasks, while being more demanding in the beginning, reduce the need for collaboration at a later stage [11]. That said, socio-technical systems (and social and technical norms) need effective governance if software projects are to succeed [7].

We observed that PEPs that had a positive outcome had a smaller lifecycle than those that had a negative outcome (or, were rejected). We believe that rejection only came after critical review, and hence, the process was delayed. In fact, such rejection would need justification. It is interesting to observe such patterns in an OSS context, where it is believed that ad hoc processes are enacted. Our observation here suggests however that Python had a strong steering committee, notwithstanding the divergence in extracted and prescribed processes. We examine the workings of these important members further in the next subsection.

5.2 Members' Involvement (RQ2)

RQ2. Who are the most influential and successful Python decision-makers? Our results in the previous section established that specific individuals (and group) are more involved in the development of PEPs than others. We have previously observed this pattern when studying actual code changes and communication logs, where a specific subset of developers tended to dominate [28,29]. Here we see a similar pattern for the normative decision-making processes, where seven members were extremely pronounced in the clusters noted. In fact, these members tended to also operate as bridges to other clusters or groups.

Looking at the wider space of evidence, Crowston and Howison [27] found that most OSS projects had a hierarchical social structure, consistent with our investigation of Python as we also noted the core developers' syndrome. Crowston et al. [26] examined the work of the developers of five small OSS projects and also found that core groups of developers comprised only a small number of those contributing to the projects. These core developers are said to be the elite contributors. In the current study such members were overseen by Guido van Rossum (C1). This confirmation of previous evidence seems to suggest that, **OSS projects activities are driven by core groups, with others supporting these members, regardless of the actual task being performed (i.e., whether planning and scoping policies or coding software functionalities).**

[17] Computed by dividing members' successful PEPs by the total PEPs they were involved in (i.e., contributed to); e.g., 51/95=0.54 for C1.

We also observed that some individuals involved in the development of PEPs were more influential to their success than others. That said, we are cautious that the success of these authors can vary based on the different types of PEPs; whether Process, Informational, and Standards Track. Such differences may also be influenced by the complexity and size of the PEPs. Thus, to get a more accurate picture of the results presented here, we plan to take these additional factors into consideration. In fact, we found that different members were involved in different forms of PEPs, although, there were also some levels of overlap in membership. This suggests that the OSS community attracts different types of people (i.e., people involved in the structuring of the social aspects of software development (Process and Informational PEPs), and people who are more technical-minded (Standards Track PEPs). Previous work has indeed promoted this notion, where it is suggested that a balancing of roles are necessary for projects to succeed. Social, task driven and critique roles were also shown to exist among IBM Jazz practitioners [40]. While we did not study this issue as such, we observed in this study that members tended to cut across the social and technical lines, with less performing both roles. These, and the other findings above, have implication for practice and theory. We consider this issue next.

6. IMPLICATIONS

We consider the implications of our findings for practice and theory in this section. In terms of practice, we observed that there were divergence in the extracted normative decision-making processes and the prescribed normative decision-making process as publicized by the Python community. There were also differences in the processes for different forms of PEPs, Process, Informational, and Standards Track. Based on these findings, perhaps the Python community should make the updated processes available to the community. In addition, a mechanism to update these processes should they change may also ensure currency. Such an approach may be automated to reduce the burden on team members. We anticipate that updated processes would benefit new members wanting to join and contribute to the OSS community, as well as existing members, as they would have a clear understanding of the decision-making processes that are involved in the creation of PEPs. Efforts aimed at keeping normative decision-making processes current may also return similar benefits to other OSS or CSS projects.

We observed that PEPs governing technical aspects of Python possessed much more complexity than social aspects, which were agreed upon much sooner. Python members should plan for these delays. These members may also preempt rejection should there be delays in the approval of PEPs. We observed that specific individuals (and group) were more involved in the development of PEPs than others. Generally, knowing who the most influential individuals in decision-making processes are could be beneficial for those needing help. Beyond general advice, such key members may even offer recommendations on the potential of proposals succeeding, or may inform the design and setup of proposals. We also noticed that specific members were involved in different forms of PEPs, although, there were some levels of overlap in membership. Accordingly, leveraging these members' specific strength could go a far way in Python's project governance.

In terms of the implication for theory, our work may be extended to other successful OSS projects such as Linux and Eclipse to evaluate if the patterns noted here would be replicated for these projects. Replicating this pattern would inform guidelines that may serve more generally for OSS projects. In fact, studying the normative decision-making process of other successful OSS projects in the GitHub repository could also inform such a knowledge base. We also anticipate that considering factors other than developers' contribution to PEPs and the number of successful PEPs they were involved in (e.g., the type of PEP, the size and complexity of the PEP, and the number of authors involved in the PEP), could provide fruitful extensions of the work that is performed here. We also plan to validate our findings with the Python community.

7. THREATS

Our work has studied artifacts of only one OSS project, which limits its generalizability. In addition, our data was gathered from a single source (i.e., the GitHub repository of Python), potentially limiting our observations, and particularly when considering members' involvement. Developers may engage about PEPs in other mediums such as mailing lists, blogs, and discussion boards where richer details about members' involvement in the normative decision-making processes is likely to be present. That said, given our replication of patterns found previously [26-29], we believe that our outcomes may apply to other OSS contexts. Finally, we did not consider factors such as the type and size of the PEP individuals were involved in during this study, which may have also impacted the pattern of results noted.

8. CONCLUSION

OSS solutions have over the years delivered noteworthy alternatives to those offered by CSS, in terms of providing low-cost, high-quality, and feature-rich systems. Thus, it is pertinent to understand the mechanisms that are likely to keep such projects going. In contributing towards this cause we explored the normative decision-making process in OSSD, using Python's PEPs as a case study. In addition, we assess members' involvement and contribution to these processes. Among our findings, we observed that the advertised process for creating PEPs is incomplete. In addition, divergence also exists at a more granular level in terms of different types of PEPs (Process, Informational, and Standards Track). Furthermore, we observed that PEPs governing technical aspects of Python took longer to be agreed upon than social aspects. We observed that specific individuals (and group) were more influential and successful in the development of PEPs than others. Based on these findings we propose that the Python community should make the updated processes available to the community. In addition, a mechanism to update these processes should they change may also ensure currency. Python members should plan for PEP delays; and furthermore, we anticipate that influential individuals would hold key insights into the decision-making processes, which could be beneficial for those needing help.

9. REFERENCES

[1] Crowston, K., Wei, K., Howison, J., and Wiggins, A., Free/Libre Open-Source Software Development: What We Know and What We Do Not Know. ACM Computing Surveys, 44, 2 (2012).

[2] Fitzgerald, B., Open Source Software Adoption: Anatomy of Success and Failure. International Journal of Open Source Software & Processes, 1, 1 (2011), 1-23.

[3] Stroll, A., Norms. Dialectica, 41, 1, (1987), 7-22.

[4] Ullmann-Margalit, E., The Emergence of Norms. OUP Catalogue (2015).

[5] Dam, H. K., Savarimuthu, B. T. R., Avery, D., and Ghose, A., Mining Software Repositories for Social Norms. In 37th ICSE (Florence, Italy, 2015), IEEE, 627-630.

[6] Savarimuthu, B. T. R., and Dam, H. K., Towards Mining Norms in Open Source Software Repositories. in Agents and Data Mining Interaction. Springer Berlin Heidelberg (2014), 26-39.

[7] Singh, M. P., Norms as a Basis for Governing Sociotechnical Systems. ACM Trans. on Intel. Sys.& Tech., 5, 1 (2013), 21.

[8] Baxter, G., and Sommerville, I., Socio-Technical Systems: From Design Methods to Systems Engineering. Interacting with Computers, 23, 1 (2011), 4-17.

[9] Andrighetto, G., Villatoro. D., and Conte, R., Norm Dynamics Within the Mind. in Computational Social Sciences. Springer International Publishing Switzerland (2014), 141-160.

[10] Axelrod, R., An Evolutionary Approach to Norms. American Political Science Review, 80, 4 (1986), 1095–1111.

[11] Epstein, J. M., Learning to Be Thoughtless: Social Norms and Individual Computation. Computational Economics, 18, 1 (2001), 9-24.

[12] Conley, C. A., and Sproull, L., Design for Quality: The Case of Open Source Software Development. Ph.D Dissertation. New York University, Grad. Sch. of Bus. Admin. (2008).

[13] Jensen, C., and Scacchi, W., Modeling Recruitment and Role Migration Processes in OSSD Projects. ProSim05, (2005), 39.

[14] Jensen, C., and Scacchi, W., Governance in Open Source Software Development Projects: A Comparative Multi-Level Analysis. Open Source Software: New Horizons. Springer Berlin Heidelberg (2010), 130-142.

[15] Murray, P., Governance in Open Source Software Projects. Lyrass, Available from: http://web.archive.org/web/20111007034152/http://www.lyrasis.org/Resources/Articles/Governance-in-Open-Source-Software-Projects.aspx

[16] Ljungberg, J., Open Source Movements as a Model for Organising. European Journal of Information Systems, 9, 4 (2000), 208-216.

[17] The Apache Software Foundation., How the ASF Works. (2011), Available from: http://web.archive.org/web/20111006032712/http://www.apache.org/foundation/how-it-works.html

[18] Hansson, S. O., Decision Theory: A Brief Introduction. Department of Philosophy and the History of Technology, Royal Institute of Technology (KTH), Stockholm (1994).

[19] Rapoport, A., Problems of Normative and Descriptive Decision Theories. Mathematical Social Sciences, 27, 1 (1994), 31-47.

[20] Greenberg, J. Behavior in Organizations. Upper Saddle River, NJ: Prentice Hall, 2011.

[21] Mintzberg, H. The Nature of Managerial Work. Harper and Row, New York, 1973.

[22] Bonito, J. Interaction and Influence in Small Group Decision Making. New York, NY: Routledge, 2012.

[23] Schermerhorn, J. R., Hunt, J. G., and Osborn, R. N., Organizational Behavior. New York, NY: Wiley, 2011.

[24] Janis, I. L., Groupthink and Group Dynamics: A Social Psychological Analysis of Defective Policy Decisions. Policy Studies Journal, 2, 1 (1973), 19-25.

[25] Bordley, R. F., A Bayesian Model of Group Polarization. Organizational Behavior and Human Performance, 32 (1983), 262-274.

[26] Crowston, K., Wei, K., Li, Q., Howison, J., Core and Periphery in Free/Libre and Open Source Software Team Communications. In 39th HICSS (Hawaii, USA, 2006), IEEE, 118.1.

[27] Crowston, K., and Howison, J., Hierarchy and Centralization in Free and Open Source Software Team Communications. Knowledge, Technology & Policy, 18, 4 (2006), 65–85.

[28] Licorish, S. A., and MacDonell, S. G., Communication and Personality Profiles of Global Software Developers. Information and Science Technology, 64 (2015), 113-131.

[29] Licorish, S. A. and MacDonell, S. G., The true role of active communicators: an empirical study of Jazz core developers. In 17th EASE2013 (Porto de Galinhas, Brazil, 2013). ACM, 228-239.

[30] van der Aalst, W. Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Berlin Heidelberg, 2011.

[31] van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H., Weijters, A., and van der Aalst, W. M., The ProM Framework: A New Era in Process Mining Tool Support. Applications and Theory of Petri Nets, Springer (2005), 444-454.

[32] van Dongen, B. F., and Adriansyah, A., Process Mining: Fuzzy Clustering and Performance Visualization. Business Process Management Workshops, Springer-Verlag Berlin Heidelberg (2010), 158-169.

[33] Eisenberg, R., de Leon, M. P., and Cohen, G., Comparative Structural Analysis of Glycoprotein Gd of Herpes Simplex Virus Types 1 and 2. Journal of Virology, 35, 2 (1980), 428-435.

[34] Liu, Y., Muller, S., and Xu, Ke., A Static Compliance-Checking Framework for Business Process Models. IBM Systems Journal, 46, 2 (2007), 335-361.

[35] Burman, P. J. Precedence Networks for Project Planning and Control. London: McGraw-Hill, 1972.

[36] Shiokawa, H., Fujiwara, Y., and Onizuka, M., Fast Algorithm for Modularity-Based Graph Clustering. In 27th AAAI (Bellevue, Washington, 2013), AAAI Digital Library, 1170-1176.

[37] Clauset, A, Newman, M. E., Moore, C., Finding Community Structure in very Large Networks. Physical Review, 70, 6 (2014).

[38] Sabidussi, G., The Centrality of a Graph. Psychometrika, 31, 4 (1966), 581-603.

[39] Licorish, S. A., Philpott, A. and MacDonell, S. G. Supporting agile team composition: A prototype tool for identifying personality (In)compatibilities. In ICSE CHASE 2009, (Vancouver, Canada, 2009). IEEE Computer Society, 66-73.

[40] Licorish, S. A. and MacDonell, S. G. Self-organising Roles in Agile Globally Distributed Teams. In 24th ACIS 2013, (Melbourne, Australia, 2013). ACIS, 1-11.

View publication stats