PROCESS COMPLIANCE IN OPEN SOURCE SOFTWARE DEVELOPMENT – A STUDY OF PYTHON ENHANCEMENT PROPOSALS (PEPS)

Savarimuthu, Bastin Tony Roy, University of Otago, Dunedin, New Zealand, tony.savarimuthu@otago.ac.nz

Dam, Hoa Khanh, University of Wollongong, Wollongong, Australia, hoa@uow.edu.au

Licorish, Sherlock A., University of Otago, Dunedin, New Zealand, sherlock.licorish@otago.ac.nz

Keertipati, Smitha, University of Otago, Dunedin, New Zealand, smitha.keertipati@otago.ac.nz

Avery, Daniel, University of Wollongong, Wollongong, Australia, da488@uow.edu.au

Ghose, Aditya, University of Wollongong, Wollongong, Australia, aditya@uow.edu.au

Abstract

Decision-making processes in proprietary software development are often well-captured. In contrast, stakeholders participating in open source software development (OSSD) projects often do not understand the decision-making processes at work, as these are either captured poorly or remain hidden. Using Python as the case study of an OSS project where the processes are well-documented, this work 'mined' the decision-making processes used to develop Python Enhancement Proposals (PEPs), to study whether the extracted process complied with the publicly advertised process for decision-making (i.e., the prescribed process). In doing so we investigate whether the previously observed normative-descriptive dichotomy in proprietary software development, between the two theories - normative decision theory (i.e., what is expected) and descriptive decision theory (i.e., what is done) applies to the domain of OSSD. We also investigate whether the decision-making processes are provided at the right level of granularity (fine vs. coarse) for different stakeholders. Our findings confirm the lack of process compliance in Python, thus confirming dichotomy. Thus, it validates the relevance of using decision-theory to study decision-making processes in OSSD. Additionally, it confirms the inadequacy of the granularity-level of the available process. We also discuss the implication for decision-making practice and processes in the Python community.

Keywords: process compliance, decision-theory, decision-making, open source software development (OSSD), Python, Python Enhancement Proposals (PEPs)

1 Introduction

Open Source Software (OSS), including Linux[1] and Firefox[2] have become promising alternatives to Closed Source Software (CSS) as they offer low-cost, high-quality, and feature-rich solutions (Crowston et al. 2012; Fitzgerald 2011; Sen et al. 2012; Von Krogh et al. 2012). Gartner[3] predicts that "by 2016, the vast majority[4] of mainstream IT organizations will leverage non-trivial elements of OSS (directly or indirectly) in mission-critical IT solutions. Consequently, IT organizations must learn to manage hybrid portfolios that contain both OSS and CSS assets". Given the growing importance of OSS, it is crucial to understand how these software projects are governed.

Many proprietary software development firms use governance frameworks such as COBIT (Lainhart IV 2000; Sahibudin et al. 2008) to enable them to comply with legal obligations. In contrast, early OSS organizations were governed in a rather ad-hoc manner, where the governance structures evolved over time (O'Mahony and Ferraro 2007). Despite this evolution, and the well-known success stories of governance in several OSS projects (Mockus et al. 2002; Shah 2006), formal governance approaches in OSS are still held to have maturity problems (Martínez-Torres and Diaz-Fernandez 2014; Schaarschmidt et al. 2015). Governance structures often include the specification of formal processes whose executions are monitored for adherence. Decision-making processes in particular are important (Weill and Ross 2004; Xue et al. 2008) because success of organisations hinge upon the processes used to make good decisions (Dean and Sharfman 1996). Despite the 'openness' that allows flexibility, establishing decision-making processes and ensuring compliance is crucial for the success of Open Source Software Development (OSSD), due to its distributed nature. Often, large OSSD projects involve huge number of voluntary participants from different geographies performing various tasks that require coordination using well-defined processes.

Prior work has shown that the processes followed in proprietary software development deviate from the prescribed process (Spreitzer and Sonenshein 2004; Suriadi et al. 2013; Swinnen et al. 2012). However, there has been limited investigation on broader compliance issues in OSSD, with works tending to focus on code convention violations and deviant behaviour such as code forking (Kessentini et al. 2010; Kogut and Metiu 2001). In the OSSD domain, Crowston et al. (2007) have noted that successful projects have established processes and the members adhere to these processes. Other researchers have also highlighted the need for adherence to processes in various contexts in OSSD (Gacek and Arief 2004; Jørgensen 2001; Mockus et al. 2002). However, their focus was not on the compliance of decision-making processes, which have been noted to be crucial for the success of organisations (Weill and Ross 2004; Xue et al. 2008). Thus, this study bridges the gap by *studying compliance in decision-making processes* in OSSD.

In considering theoretical frameworks suitable for studying compliance in decision-making processes, decision theory (Dean and Sharfman 1996) was noted to be the most suitable as it focuses on the study of how decisions are made by individuals and groups in organisations (Hansson 1994; Peterson 2009). The two main branches of study are the normative and descriptive decision theories. While the normative decision theory studies how decisions *should be* made, and descriptive theory studies how decisions *are actually* made. Our work here scrutinizes the nature of the normative-descriptive dichotomy ('as-is' vs. 'should-be') in human-decision making (Bell et al. 1988; Hands 2012; Rapoport 2013). This dichotomy in decision-making is a result of the inherent tension between the two, where on the one hand organisations desire to follow prescribed ('should-be') processes, and on the other hand leverage the opportunity to modify those processes based on day-to-day practicalities (Rapoport

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

[2] http://tinyurl.com/bw7edaw

[3] https://www.gartner.com/doc/2829917/enterprise-open-source-coexist-hybrid

[4] The vast majority here is at least 95%.

2013). Thus, by studying process compliance, this work bridges the gap in the decision-theory literature on whether there is dichotomy in decision-making processes in OSSD, and if so, to what extent. So far this has been demonstrated only in the proprietary software development context (Suriadi et al. 2013; Swinnen et al. 2012).

Some of the key prerequisites for an effective process employed in an organisation are: (1) the process is *transparent* to the users (i.e., the process is available to the stakeholders) (Kalpic and Bernus 2002), (2) the process is *complete* (i.e., the process includes all the required details – and doesn't contain more or less details than required) (Aguilar-Saven 2004; Kalpic and Bernus 2002), and (3) the process is *adherent* to the expectations of the organisation and other external bodies (Ghose and Koliadis 2007; Sadiq et al. 2007). We investigate these prerequisites in OSSD under the umbrella of *compliance*, by studying decision-making processes in Python as a case study. Additionally, researchers have reported that the processes made available might not be at the right level of granularity (Ko et al. 2009; Rosemann and De Bruin 2005), despite meeting compliance requirements. We also investigate this issue.

The remaining sections of the paper are structured as follows. In Section 2 we present the background on process compliance in OSSD. We also provide an overview of the decision-making processes used in the development of Python Enhancement Proposals (PEPs), and present the three research questions in this section. In Section 3 we present the methodology used to extract the decision-making processes, and the relevant metrics used for assessing the results. We then present our results for each of the research questions in Section 4, before discussing our findings and outlining their implications in Section 5. Thereafter, we present our plan for future work in Section 6, before finally providing concluding remarks in Section 7.

2 Compliance with decision-making processes in OSSD

This section provides a background on the issue of process compliance in organisations, and in particular OSSD organisations. It then introduces the Python case study and presents the research questions.

2.1 Process compliance in organisations

Organisations employ well-defined processes to facilitate their success (Dean and Sharfman 1996). These processes are available in text and figure formats (process diagrams) to the stakeholders involved, and these processes are required to comply with a variety of requirements (e.g., a set of standards, policies and protocols), both internal and external to the organisation (Silverman 2008). Compliance of these processes in an organisation is managed using compliance management frameworks (El Kharbili et al. 2008; Ghose and Koliadis 2007; Sadiq et al. 2007), which monitor, detect, mitigate and also undertake reparations measures if deviance is detected. Detecting compliance violations is a key task, which is aided through the modelling of business processes using modelling notations such as (BPMN) (White 2004) and Simple Precedence Diagrams (SPD) (van Dongen et al. 2005). By comparing the process diagrams constructed using such techniques the compliance of enacted processes is validated against those prescribed (Ghose and Koliadis 2007; Sadiq et al. 2007).

2.2 Process compliance in decision-making processes in OSSD

Several researchers have investigated the nature of decision-making processes in OSSD (Akintomide et al. 2014; Jensen and Scacchi 2005; Jensen and Scacchi 2010; O'Mahony and Ferraro 2007; Schaarschmidt et al. 2015; Shah 2006). However, compliance to decision-making processes has not received attention. Our investigations of the well-known open source projects such as Linux and Firefox show that the decision making processes are not made publically available (i.e., are not made explicit), albeit, these could lie hidden in multiple sources such as mailing lists and discussion boards. To this end, the processes are not often readily accessible to various external stakeholders, and hence, compliance verification cannot be easily undertaken. An approach to make these processes explicit is to 'mine' them from the project repository data.

Given our goal to study the nature of process compliance in OSSD, we chose Python (Van Rossum and Drake Jr 1995), a well-known, widely used open-source programming language. We were particularly driven by the excellent publicly-available documentation of the Python decision-making processes, both in textual and process diagram[5] forms. Not only is the Python decision-making process transparent to the public (*process transparency*), the data about the enactment of these processes is also made public (*data transparency*) in the form of archived history logs (documents and mailing list discussions). These artefacts could be mined to construct and validate the enacted process. This enables compliance checks to be performed by independent observers (e.g., authors who are not involved in the development activities of Python). The following sub-section provides details about the decision-process investigated in the Python context.

2.3 Python Enhancement Proposals (PEPs)

In Python development, Python Enhancement Proposals (PEPs) are decisions codified in the form of documents. These documents capture all the major changes proposed to the language and the processes that should be adhered to by the developers while developing the software collaboratively. There are 363 PEPs in total (obtained at the end of 2014), that belong to three categories: process, informational and standard track PEPs. Process PEPs describe the process that should be used by developers during the process of software development. For example, PEP 10[6] describes the voting process. It specifies that voters must use +1, 0 or -1 in their email messages to indicate their approval, neutrality or disapproval respectively. Informational PEPs provide information to the development community such as the philosophical underpinnings of Python development (PEP 20[7]) and release notes for various versions of Python (e.g., PEP 398[8]). Standard track PEPs describe the changes proposed to the Python language. These include features that need to be added, removed and modified, triggering the evolution of the language. Out of 363 PEPs, 33 are process PEPs, 45 are informational PEPs and 285 are standard track PEPs.

Similar to any artefact that is collaboratively and incrementally developed by a group of people, PEP documents evolve over time. The evolution of PEP can be tracked from archived version history files. The process of PEP development (i.e., decision-making process employed) is discussed in PEP 1 (see footnote 5). The status attribute in the PEP document captures the state of the proposal. The state-transition diagram describing the different states of the PEP and their transitions as prescribed in the Python community (the *prescribed process* to be used by decision makers to indicate the state of the proposal) is given in Figure 1. The states are the rectangles and the transitions are directed, unlabelled arrows.

A simplified version of the decision-making process works as follows: when someone in the community has a new idea, it is recommended that the idea gets discussed at an appropriate forum (typically using one of the mailing lists[9]). If the idea gains positive momentum, a PEP document is drafted by the proposer(s). Once the draft is ready, it gets circulated among stakeholders (developers and users through appropriate mailing lists such as python-dev and python-list), and if the support is positive, the PEP is accepted. PEPs can be rejected by the community (developers and users) or by the creator of Python. The PEPs have to adhere to the PEP conventions and these are checked by the PEP editors before they are accepted. When there are no more changes required on a PEP, it would be finalized (i.e., no more changes are made). Also, PEPs can be withdrawn by the authors if the ideas are not worth pursuing. PEPs can be replaced by other PEPs. Some PEPs remain in the active state which represents that these PEPs always hold and they are never meant to be finalized so as to make changes

[5] https://www.python.org/dev/PEPs/PEP-0001/

[6] https://www.python.org/dev/PEPs/PEP-0010/

[7] https://www.python.org/dev/PEPs/PEP-0020/

[8] https://www.python.org/dev/PEPs/PEP-0398/

[9] https://mail.python.org/mailman/listinfo

when required. An example is the voting PEP which currently holds, but can be changed if required. Some drafted PEPs may be deferred for some time before they are redrafted to address some known issues. Figure 1 provides a simple demonstration of the process a PEP traverses. This detail is made publically available by the Python community (refer to footnote 5).

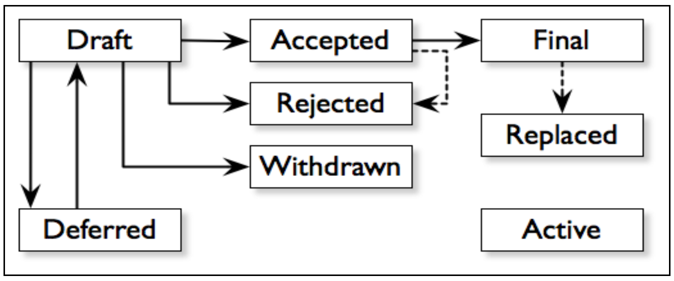

Figure 1. Prescribed process of PEPs

3 Research questions

Given the importance of decision-making processes in the success of organisations (Dean and Sharfman 1996) and their importance in OSSD (Akintomide et al. 2014; Jensen and Scacchi 2005; Jensen and Scacchi 2010; O'Mahony and Ferraro 2007; Schaarschmidt et al. 2015; Shah 2006), as discussed in Section 2, the nature of compliance (or the lack there of) of these processes needs to be investigated. Using Python as the case study domain, we investigate whether the previous findings reporting the dichotomy between normative and descriptive theories, in proprietary software development context (Spreitzer and Sonenshein 2004; Suriadi et al. 2013; Swinnen et al. 2012), indeed apply to OSSD domain. Scrutinizing this forms our first research question.

RQ1: Is the Python enacted decision-making process (i.e., extracted process) compliant to the publicly advertised process (i.e., prescribed process)?

Process compliance will be scrutinized using the three key prerequisites of an effective process (transparency, completeness and adherence), as discussed in Section 1.

Also, the adequacy of the abstraction-levels of the processes made available needs to be investigated, given prior findings that in some organisations the level of process details available to the stakeholders is inadequate (Ko et al. 2009; Rosemann and De Bruin 2005). Considering there are three types of processes - process, informational and the standard track PEPs, and their different foci, we need to scrutinize whether the details available in the prescribed and extracted processes are adequate for different stakeholders (i.e., details are presented at the right granularity-level) for day-to-day decision-making. This is because we envisage different PEP types might warrant different level of details to be made available to the stakeholders engaged in those processes. To examine these, we compare the structures of the prescribed and extracted processes to those processes extracted for three specific types of PEPs. These investigations are captured in the two research questions below that compare process similarity.

RQ2. Are the extracted PEP processes of the three types of PEPs (process PEPs, informational PEPs and standard track PEPs) similar to the prescribed process or the extracted process?

RQ3. Are the three types of PEP processes similar to each other?

RQ2 would reveal whether there is any value to the publicly available prescribed process or the ‘mined’ extracted process, and RQ3 would expose the right abstraction level for the processes used in Python. Answering these questions may have implications to practice of decision-making in OSSD.

4 Methodology

To investigate the research questions presented above, we needed a framework that can extract the enacted decision-making process used by the Python development community from the data repository. We developed a framework and operationalized it in the form of a software tool since existing tools were unsuitable for our new problem. The framework uses the methodology (a set of steps) shown in Figure 2. First, the software tool[10] retrieves the PEP diff files for all the PEP commits from the GitHub repository[11]. 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 *active* state. The diff file in this case will highlight that there was a change in status (i.e., from *draft* to *active*). Second, we used the tool to extract the status change in each diff file. This was done by searching each diff file using a regular expression to identify the status which contained the pattern "+status: X", where the text that follows the colon (X in this case) is the state[12]. Third, using the tool, we recorded the change of states from all versions of all the PEPs and stored them as state logs. A sample state for PEP 201, a standard track PEP would contain an entry 'draft-accepted-final' to indicate that the PEP was initially drafted, then accepted, then finalized. Fourth, we used process mining tool, ProM (Van Der Aalst 2011; van Dongen et al. 2005), to construct the process diagrams in the form of Simple Precedence Diagrams (SPD) (van Dongen et al. 2005). Fifth, we analysed the results using different computational techniques to provide quantitative evidence towards answering the three research questions. These techniques were also augmented by formal statistical testing for validation. The techniques used are described in the following sub-sections.

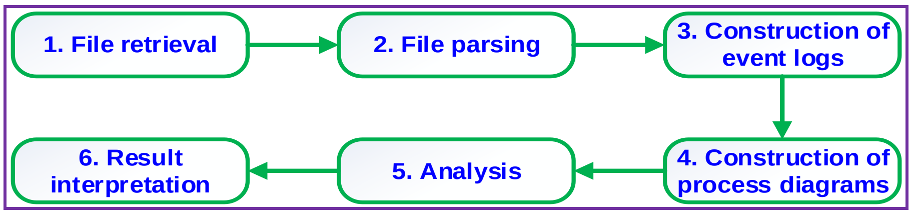

Figure 2. Methodology for normative process extraction and analysis

4.1 Compliance verification (RQ1)

The enacted decision-making process is visualised as a process diagram. A process diagram typically comprises of three units - nodes, transitions and pathways. A node in our work represents the state of the PEP (e.g., draft). The transition represents the change of state (e.g., a PEP's state changing from draft to accepted). The pathway represents the full path taken by a particular PEP (e.g., draft-accepted-final), which comprises of nodes and transitions. The process diagram shown in Figure 1 contains nodes that are represented as boxes and the transitions as directed arrows.

To investigate whether there are differences between the prescribed and extracted processes (RQ1), we compare the frequency (counts) of the three units of analysis: (1) nodes (i.e., states), (2) pairwise transitions, and (3) pathways, for these two processes. These metrics are widely used in the business process modelling and management literature (Gerth 2013; Smirnov et al. 2012). The difference between the prescribed and extracted processes for a particular unit of analysis is done using the percentage change (Kaiser 1989) metric using the formula ((E - P)/P) * 100 where E and P are the counts of the unit of analysis for the extracted and prescribed processes respectively. Graph based approaches were not considered as only approximate solutions are known for this NP-hard problem (Zeng et al. 2009).

4.2 Similarity comparison (RQs 2 and 3)

To investigate whether the extracted process of each type of PEP is similar to the overall extracted process or the overall prescribed process (RQ2), we employ the same three metrics described in Sec-

[10] The developed software tool is now available for public use. However, the URL is not provided to preserve anonymity.

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

[12] See http://tinyurl.com/qdvgqrb for an example.

tion 4.1, and the difference is computed using the same percentage change metric. To investigate the similarities between the extracted processes of the three types of PEPs (RQ3), apart from using process-level metrics (percentage change for the three units of analysis), we also investigated the instance-level metrics[13].

5 Results

5.1 Compliance verification (RQ1)

The process extracted using the methodology discussed in Section 4 is shown in Figure 3. The figure shows the unique pathways the PEP documents (all 363 of them) have followed. By comparing the structures of Figure 1 (prescribed process) and those of Figure 3 (extracted process), one can infer that there are *significant* differences between the prescribed process (Figure 1) and the extracted process (Figure 3). The extracted process is *more complex* than the prescribed process. The complexity is as a result of capturing all the state changes that have occurred (represented by the incoming and outgoing arcs to any particular state) using a data-driven approach. To understand the scale of the divergence in process compliance, let us examine the *accepted* state and its predecessors and successors both in the prescribed and the extracted processes. The prescribed process in Figure 1 captures that a PEP goes from *draft* to *accepted* state and then into the *final* state. The state has one incoming arrow and one outgoing (solid) arrow. In contrast, the extracted process shows that there are 5 incoming and 7 outgoing arrows, all referring to other states. This points us towards the severity of the (lack of) compliance problem.

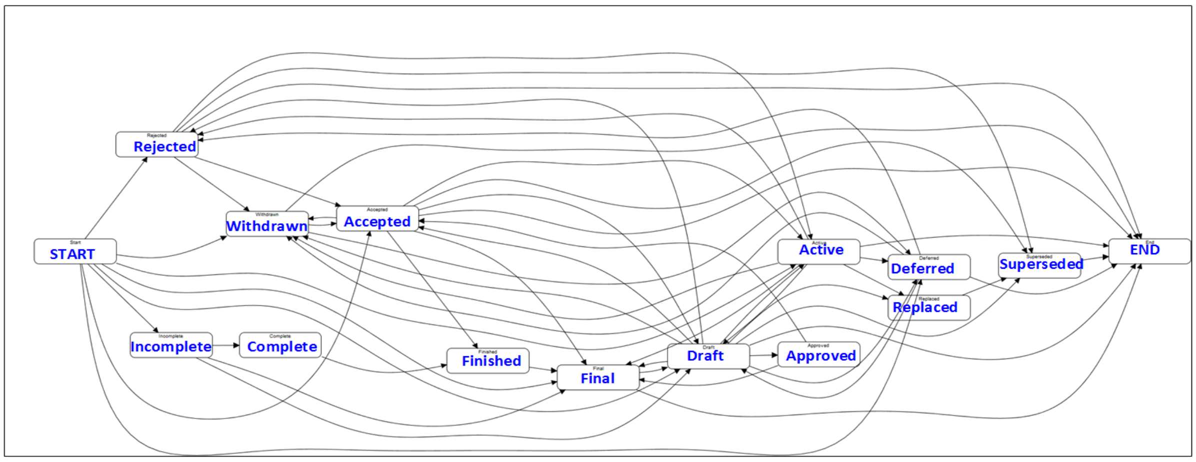

Figure 3. Extracted process of the PEPs

Table 1 shows the metrics that have been extracted from the structures of the two process diagrams (Figures 1 and 3). The counts of the three metrics (nodes, transitions and pathways) are given in columns 2 and 3 and the percentage change is shown in column 4. Based on the results of percentage change it can be inferred that there are significant differences between the prescribed and extracted processes ranging from 62% to 650% for the different units of analyses. We conducted a t-test (two-sample assuming unequal variances) to assess the significance of these differences, which reveals a significant difference in the mean of the structural metrics for the two processes (p=0.003), highlighting that there were significant divergence between what was prescribed and what was enacted by the Python community. We next present the details of each of the three types of analyses that have been conducted.

[13] The detail of each PEP (e.g., draft-accepted-final) is called an instance. While the process metrics show the metrics for a process diagram involving all the PEPs, an instance level metric shows the details of a particular instance (e.g., a PEP). The PEP draft-accepted-final comprises of 3 nodes, 2 transitions and 1 pathway.

Comparison criterion	**Count for the prescribed process (P) shown in Figure 1**	**Counts for the extracted process (E) shown in Figure 3**	**Percentage change**
Number of unique states	8	13	62%
Number of unique pairwise transitions	9	39	333%
Number of unique pathways	6	45	650%

Table 1. Comparison of counts of unique states, pairwise transitions and pathways for prescribed vs. extracted processes

Node analysis - Table 1 shows that there are five new states (13-8) in the extracted process (refer to Figure 3), which are missing in the prescribed process (see Figure 1). These are *incomplete*, *complete*, *finished*, *replaced* and *approved*. Our analysis shows that all the five states have been used at least once while the state *incomplete* appears 19 times in the state logs. While there is evidence for the state *replaced* to be renamed to *superseded*[14], the reasons for not including the other four states in the prescribed model are not obvious. Possible reasons include: (a) error of omission and (b) these states may not have been considered to be important (or even useful). Another possible hypothesis is that, since some of these states could be easily folded into others (e.g., *complete* and *finished* are synonymous to *final*, *incomplete* is closer to *draft* semantically, and *replaced* is synonymous to *superseded*), these could have been omitted. Whatever the reason for excluding these states[15], it is clear that the prescribed model does not precisely capture the underlying state-transitions, thus, demonstrating that the process isn't complete. The omission of these states may be assessed as violations of the prescribed PEP developmental process.

Transition analysis - The comparisons of pairwise transitions between the two processes showed 333% change between the prescribed and enacted process. It was interesting to observe that some transitions presented in the prescribed process never materialized in the extracted process, thus pointing to the problem of presenting inaccurate details. The dotted arrows in the prescribed process in Figure 1 show the (possible) state changes between *accepted* and *rejected* and also between *final* and *replaced*. However, these were not observed during PEP enactment, and hence, we did not identify these in the extracted process. This suggests that the dashed arrows may have perhaps been erroneously added to the diagram. Additionally, the prescribed process shows the loop between *draft* and *deferred*. However, many other loops between pairs of states (e.g., between *active* and *rejected*) are not captured, even with the exclusion of the loops involving the five new nodes identified in the previous sub-section. Again, omitting this information about the cycles present can mislead stakeholders (particularly new members).

Pathway analysis - Table 2 shows there were 45 unique pathways for a PEP in the extracted process, compared to 6 in the prescribed process (650% change). The reason for the huge increase in the number of unique pathways is because of the five states that were not captured in the prescribed model. This was exacerbated by the number of loops that haven't been considered between the states. We also conducted case-by-case investigation of the PEPs to investigate the magnitude of the deviance problem. While 64% of PEPs follow one of the 6 pathways in the prescribed process, 36% of the PEPs (132 out of 363) deviate from these pathways (i.e., 132 PEPs follow one of the 39 deviant pathways).

5.2 Similarity comparison (RQs 2 and 3)

Given there are three specific types of processes with different foci, similarity comparison enables us to answer, whether the publicly available prescribed process and the 'mined' extracted process are adequate for different Python stakeholders (i.e. details are provided at the right level of abstraction).

[14] https://mail.python.org/pipermail/docs/2013-March/013093.html

[15] We hypothesize that the Python community could have presented the simplified process so as to avoid burdening the developers with complexity.

5.2.1 Similarity between extracted processes and the overall processes (RQ2)

Using the methodology described in Section 4, we also extracted the enacted processes for each of the three types of PEPs. We studied whether the structures of these three types of PEPs were similar to the extracted process or the prescribed process, thus answering whether these two processes are adequate for the stakeholders to obtain a holistic overview. To undertake this task, we conducted two sets of comparisons of the process structures: (1) the overall prescribed process (Figure 1) and the extracted processes of the three types of PEPs, and (2) the overall extracted process (Figure 3) and the extracted processes of the three types of PEPs. The extracted processes for the process and informational PEPs are shown in Figures 4 and 5. The extracted process for the standard track PEP closely resembles the structure of the overall extracted process (Figure 3), and hence, is not shown here. *Examination of type-specific process structures (i.e. Figures 4 and 5), reveals these are different to that of the prescribed (Figure 1) and extracted (Figure 3) processes, reflecting their inherent differences*. Our statistical analysis of data from the structures of the overall prescribed process (Figure 1) and the type-specific extracted processes show that there is statistically significant differences between the two processes (p=0.049 for the t-test outcome). However, no significant difference was noted between the structures of the extracted process and the type-specific extracted processes (p=0.071 for the t-test – though this result is very close to the 95% confidence level). This outcome shows that the overall extracted process is slightly closer to the structure of the three individual PEP types, hence, a better model of the actual decision-making process.

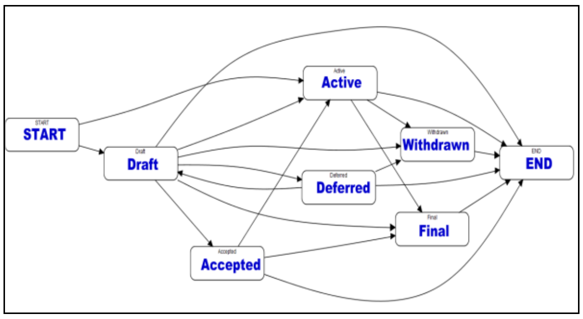

Figure 4. Extracted process of Process PEPs

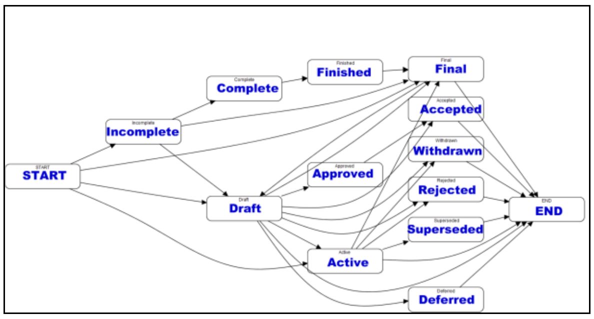

Figure 5. Extracted process of Informational PEPs

We further investigated the similarities of the overall extracted and prescribed processes to that of the extracted processes for each type of PEP process in detail so as to gain further insights. The results of the two sets of comparative analyses are provided in Table 2 for each of three comparison criteria (states, pairwise transitions and pathways). Columns 3 and 4 show the metrics (counts) for the overall prescribed process (P-All) and the overall extracted process (E-All) respectively. These values come from Table 1, and are repeated for each PEP type. Column 5 shows the counts for the extracted process for each type of PEP (E-Specific) - process, informational and standard track PEPs. Column 6 shows the percentage change between the extracted process for each type of PEP and the overall prescribed process (E-Specific vs. P-All). The values of this column are derived from the values in columns 3 and 5 using the percentage change formula presented in the previous section. Column 7 shows the percentage difference between the extracted process for each type of PEP and the overall extracted process (E-Specific vs. E-All). The values for this column are derived from columns 4 and 5. A positive value in columns 6 and 7 indicate that the specific process (E-Specific) has more entities (e.g., nodes, transitions and pathways) than the overall process model (P-All or E-All). A negative value indicates the opposite.

PEP types	**Comparisons**	**Counts for Prescribed process (P- All)**	**Counts for Extracted process (E–All)**	**Counts for Extracted process for each type of PEP (E-Specific)**	**Percentage change** **(E-Specific vs. P-All)**	**Percentage change** **(E-Specific vs. E-All)**
Process PEPs	Unique states	8	13	6	-25%	-54%
	Unique pairwise transitions	9	39	11	22%	-72%
	Unique pathways	6	45	9	50%	-80%
Informational PEPs	Unique states	8	13	13	63%	0%
	Unique pairwise transitions	9	39	19	111%	-51%
	Unique pathways	6	45	15	150%	-67%
Standard Track PEPs	Unique states	8	13	12	50%	-8%
	Unique pairwise transitions	9	39	34	278%	-13%
	Unique pathways	6	45	39	550%	-13%

Table 2. Comparison of prescribed process vs. extracted process for each of the three types of PEPs

From columns 6 and 7 of Table 2 it can be observed that there are significant differences between the type-specific process models that have been extracted and the overall process models where the values range from -25 to 550 for column 6 and -80 to 0 for column 7. *The values show that each of the three process models extracted for each type of PEP is unique* (i.e., graph structures are not the same).

Figure 6 presents the values in columns 6 and 7 of Table 2 in a graphical form. There are 9 pairs of columns (bars) in the figure. The figure shows three partitions (separated by horizontal dotted lines) and the three pairs within a partition correspond to a particular PEP type (PP, IP and ST in the diagram denotes process, informational and standard Track PEPs respectively). Within the group of three, the first pair of bars corresponds to state, the second corresponds to transition and the third corresponds to pathway. The first bars in each pair (in blue) correspond to the comparison between E-Specific and P-All. The second bars in each pair (in red) correspond to the comparison between E-Specific and E-All. For each of the three groups we compare whether the blue bars or the red bars are closer to zero. Values closer to zero indicate lower deviance between a specific PEP process and the overall process. In other words, if the values of blue bars are closer to zero than the red bars, then the specific process is closer to the overall prescribed process (Figure 1), otherwise is closer to the overall extracted process (Figure 3).

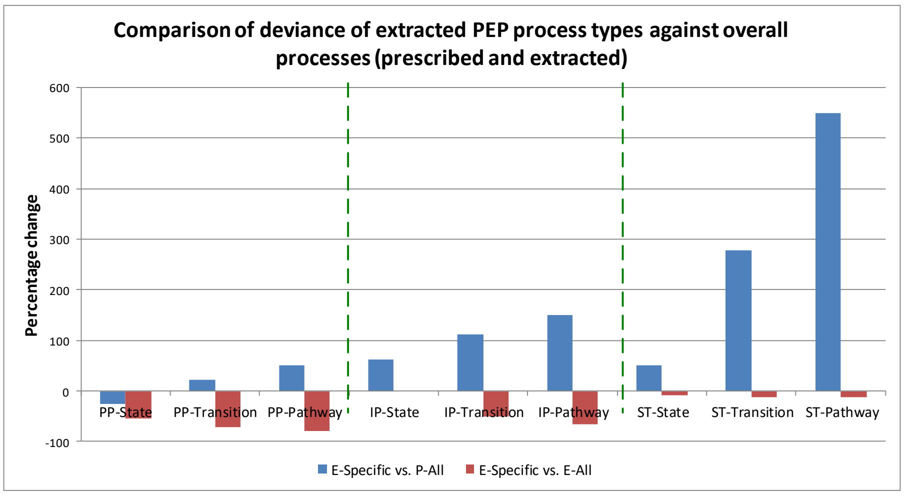

Figure 6. Comparison of each of the three extracted processes and the overall processes

From the first group, it can be observed that the deviations (percentage change) between the extracted process PEP and the overall prescribed process PEP (E-Specific vs. P-All) is mostly positive. On the other hand, the deviation between the extracted process PEP and the overall extracted process PEP (E-Specific vs. E-All) is negative. For the first group, it should be noted that the magnitude of the negative values are higher (reaching a low value -80%) and that the blue bars are closer to zero than the red bars. This shows that the structure of the process PEP (Figure 3) is closer to the structure of overall prescribed process PEP (Figure 1) than extracted process PEP (Figure 3). In contrast, using similar analysis for the second group, we note that the structure of the information PEP (Figure 4) is closer to the extracted process PEP (Figure 3) than the prescribed process PEP (Figure 1). Similarly, the result for group 3 shows that the structure of the standard track PEP is closer to the structure of the overall extracted process PEP (Figure 3) than that of overall prescribed process PEP (Figure 1). This is interesting because this points to the fact that *the prescribed process made available, though not representative of the overall enacted process, is closer to the enacted process of the process PEP (hence, argued to be useful), but is not a good fit for the informational and the standard track PEPs*. The overall extracted model is a better a better fit for those. So, both the overall processes (extracted and prescribed) are useful, although for capturing different types of PEPs.

5.2.2 Similarity between three types of PEPs (RQ3)

Given that there are three types of PEPs, we also investigated whether there are similarities between the processes of these three types of PEPs (RQ3) by comparing the structures of the three types of PEPs with one another (i.e., intra-process comparisons) using the percentage change metric (results are based on columns 3 and 5 of Table 2).The results showed significant deviations among the three. This result appears to be intuitive because it shows the unique purpose of the three types of PEPs. *This uniqueness highlights that there is a need for making fine-grained, type-specific process models to be available to the users*. Currently, the stakeholders are limited to the high-level prescribed process (Figure 1).

We noticed a close resemblance of the structure of the standard track PEP to the extracted overall process (Figure 3), hence, we scrutinized whether the other two (process and informational PEPs) are indeed similar to each other functionally, but are distinct from the standard track PEPs. To scrutinize this, we conducted analysis of instance-level details. This was done by investigating how often different states (nodes) were visited by the three types of PEPs (363 PEPs). Figure 7 shows the relative proportions of states assigned for the three types of PEPs. It can be observed that most PEPs were assigned the *draft* state (the area in Orange) most of the time (74%, 71% and 42% for the three types of PEPs). While the proportions for process and informational PEPs are close (74% and 71% respectively), the proportion for standard track PEP is comparatively small (42%). It is interesting to note that overall frequency patterns for Process and Information PEPs are similar. However, the overall pattern for the standard track does not appear to be similar to the other two. This is because standard track PEPs spent higher proportions of time in *accepted*, *final* and *rejected* states than the other two. Though differences between the three types of processes at the instance-level are visible in Figure 7, these differences were not statistically significant (i.e., ANOVA test, and the follow-up t-test did not reveal any statistically significant differences).

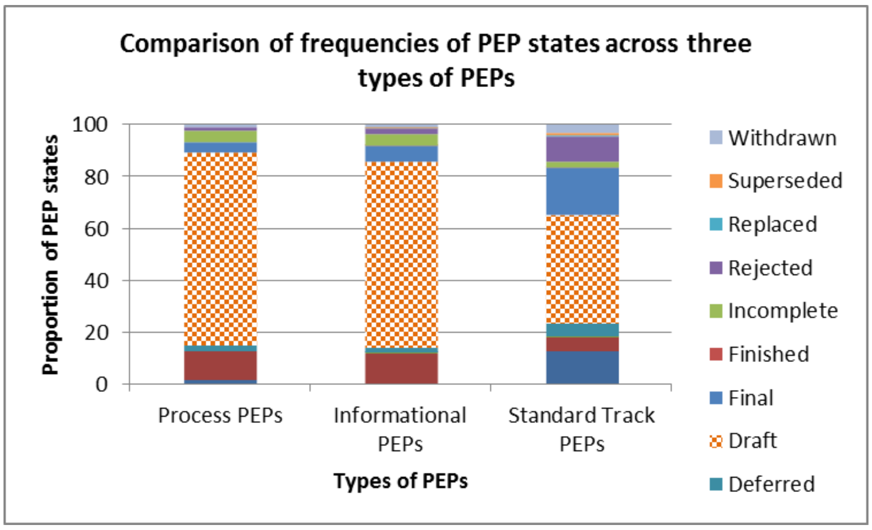

Figure 7. Frequency analysis of three types of PEPs

6 Contributions and implications

While there has been work on process compliance in proprietary software projects, the nature of compliance in decision-making processes in OSSD has not been explored. Given the importance of decision making processes to the success of organisations, and the availability of data from open source repositories, we embarked on mining artefacts related to the decision-making process of Python developers, a representative case of a matured OSS project. We looked to verify whether the decision-theory literature that reports on the dichotomy of decision-making processes in fact holds in the OSSD, thus, revealing the relevance of decision-theory in this domain. Towards this end, this work studies the decision-making process employed during the development of Python Enhancement Proposals (PEPs). We now discuss our key contributions with respect to the research questions, and outline the implications for theory and practice.

RQ1. Is the Python enacted decision-making process (i.e., extracted process) compliant to the publicly advertised process (i.e., prescribed process)? It was shown that there are significant differences between the Python enacted decision-making process and the prescribed process (62%-650% for the three types of analyses conducted), and in particular, 36% of the pathways were deviant. We believe that this finding has multiple implications for practice and theory, of which four are considered here. First, through mining publicly available data, it points to the *lack of process transparency* (i.e., the actual Python process is hidden, and had to be extracted). The processes involved with extracting this knowledge requires computational-machinery which is not at the disposal of potential Python contributors. Our finding thus supports the previous finding that transparency is one of the key problems in open source software development (Jensen and Scacchi 2010). These authors have indeed noted that this lack of transparency has prevented other community members from understanding and accepting the changes taking place within OSSD. Second, we also provide evidence for the *incompleteness* (*inaccuracy) of process details* (i.e., the prescribed process has missing details and also some extraneous details that weren't required). Given that accurate knowledge and transparent dissemination of the internal processes were deemed to be critical factors for individuals' and organisations' success (Aguilar-Saven 2004; Kalpic and Bernus 2002), the incompleteness noted in our results could be potentially detrimental to the effective day-to-day decision-making of Python developers, and those working in other OSSD projects where this phenomenon exists. Third, we observed a *lack of compliance* within Python. The extracted enacted process (Figure 3) shows that the process used is considerably different to the prescribed process (Figure 1). Apart from hindering decision-making of internal stakeholders this could result in external stakeholders being potentially misinformed about the actual decision making processes. *The results reveal the dichotomy in decision-making in OSSD, thus, affirming the relevance of decision theory to OSSD, beyond proprietary software development*. It is interesting to note that the magnitude of process deviance in OSSD is not very different to the deviance reported in proprietary development (29% deviance in processes as observed by Swinnen et al. (2012) when compared to 36% in our study). The slight increase in magnitude of deviation is of potential concern, and we suspect that there may be larger deviations in other OSSD due to its open nature, which promote deviation. Further investigation in this vein is required to assess the magnitude of the compliance deviance in other open source projects. Fourth, an implication to practice of decision-making in Python is that the *Python community is likely to benefit from the overall structure of the extracted process* to obtain the holistic bird's eye view of the actual enacted process.

RQ2. Are the extracted PEP processes of the three types of PEPs similar to the prescribed process or the extracted process? Results presented in Table 2 show that there are significant differences between the three types of PEPs and the overall process models (prescribed and extracted). The analysis further revealed that the extracted process for the process PEPs is closer to the overall prescribed process (Figure 1). However, informational and standard track PEPs are closer to the overall extracted process (Figure 3). Thus, the overall process model are still useful. However, the significant difference between the structures of the three types of PEPs suggests that the *descriptive decision-making processes (i.e. prescribed processes) are not being provided at the level of granularity that might be potentially desirable in Python*. This highlights the need for presenting more fine-grained, type-specific process models (e.g., as shown in Figures 4 and 5), as against the overall holistic processes (Figure 1 and 3). We believe that this divergence could be a result of a dilemma about the right level of granularity (fine vs. coarse) for presenting process details within an organisation (Elzinga et al. 1995). An implication of this finding for practice, and particularly to Python developers, is that the community is likely to benefit from making the fine-grained type-specific extracted processes available for each of the three types of PEPs. Such a move could be especially useful as different groups of individuals may work on different types of PEPs, requiring process knowledge corresponding to their roles and responsibilities. For example, our investigation reveals that the process and informational PEPs are worked on by the project managers and the core-developers of Python and the standard track PEPs involve a range of stakeholders (developers, patch-fixers, etc.). The availability of the overall extracted process, and those for the specific-types of processes employed in Python would enable process transparency, and also potentially speed up compliance verification (e.g., through manual visual-verification by individuals in the first instance).

RQ3. Are the three types of PEP processes similar to each other? Our analyses reveal that the process and informational PEPs are similar to each other functionally (based on instance-level data). We believe this is because of inherent similarities between the process and information PEPs (i.e., these deal with *social aspects* of software development – e.g., how do we vote and the release notes for the latest version), but the standard track PEPs are different as they deal with *technical aspects* of software development (e.g., what new features should be built). The implication of this finding relates to the decision-making practice in Python as it *raises the question on process re-engineering: should a single process describing the two (process and informational PEPs) be made available instead of two processes?*, which could streamline the internal processes into just two categories instead of three (1- *social PEPs* that now consolidate process and informational PEPs and 2- *technical PEPs* comprising the standard track PEPs).

7 Future work

There are several interesting avenues for future work. First, given that deviance has been observed, we plan to investigate the reasons for deviance and the nature of the deviance (positive or negative), using deviance theory as the lens of our investigation. We plan to conduct this study by examining the Python communication archives (e.g., emails, discussion boards and wikis). We will triangulate our findings by seeking the participation of the Python community in interviews or focus group sessions. Second, we intend to extend the study to other successful and failed OSS projects to assess the magnitude of compliance problems in decision-making processes, and also study the motivations for deviance and its impact.

8 Conclusion

Compliance verification of enacted decision-making processes with the prescribed processes in OSSD hasn’t received much attention. Using Python Enhancement Proposals (PEPs) as the case study artefacts to investigate decision-making processes, this work first proposed an approach for extracting the enacted decision-making processes. Second, by comparing the enacted process to the prescribed process, it showed that the prescribed process is incomplete (inaccurate) and that the enacted process deviates significantly. This result demonstrates that the dichotomy between normative and descriptive decision theories, originally demonstrated in proprietary software development, is also relevant to the domain of OSSD. Third, we compared the similarity between the extracted processes for three types of PEPs (process, informational and standard track) with the overall extracted process and the prescribed process. The result showed that the prescribed process is still valuable, as it is closer to the extracted process of the process PEPs, while the extracted processes for the other two PEP types are closer to the overall extracted process. Fourth, we examined the similarity between the extracted processes of the three types of PEPs, and showed that process and information PEPs are functionally closer to each other. The findings above have implication for decision-making processes in the Python project community. In particular, we believe that the Python community is likely to benefit from having a two-levels of decision making processes, the coarse grained overall extracted process and the fine-grained, type-specific extracted processes for the three PEP types.

References

Aguilar-Saven, R. S. 2004. "Business Process Modelling: Review and Framework," *International Journal of production economics* (90:2), pp. 129-149.

Akintomide, A., Ram, P., Chimariya, A., and Ahmed, L. 2014. "Chaos or Distributed Control: Governance in Ossd."

Bell, D. E., Raiffa, H., and Tversky, A. 1988. *Decision Making: Descriptive, Normative, and Prescriptive Interactions*. Cambridge University Press.

Crowston, K., Li, Q., Wei, K., Eseryel, U. Y., and Howison, J. 2007. "Self-Organization of Teams for Free/Libre Open Source Software Development," *Information and Software Technology* (49:6), pp. 564-575.

Crowston, K., Wei, K., Howison, J., and Wiggins, A. 2012. "Free/Libre Open-Source Software Development: What We Know and What We Do Not Know," *ACM Computing Surveys (CSUR)* (44:2), p. 7.

Dean, J. W., and Sharfman, M. P. 1996. "Does Decision Process Matter? A Study of Strategic Decision-Making Effectiveness," *Academy of management journal* (39:2), pp. 368-392.

El Kharbili, M., Stein, S., Markovic, I., and Pulvermüller, E. 2008. "Towards a Framework for Semantic Business Process Compliance Management," *Proceedings of GRCIS* (2008).

Elzinga, D. J., Horak, T., Lee, C.-Y., and Bruner, C. 1995. "Business Process Management: Survey and Methodology," *Engineering Management, IEEE Transactions on* (42:2), pp. 119-128.

Fitzgerald, B. 2011. "Open Source Software Adoption: Anatomy of Success and Failure," *International Journal of Open Source Software & Processes. v1 i1*), pp. 1-23.

Gacek, C., and Arief, B. 2004. "The Many Meanings of Open Source," *Software, IEEE* (21:1), pp. 34-40.

Gerth, C. 2013. "Introduction," in *Business Process Models. Change Management*. Springer, pp. 1-12.

Ghose, A., and Koliadis, G. 2007. "Auditing Business Process Compliance." Springer, pp. 169-180.

Hands, D. W. 2012. "The Positive-Normative Dichotomy and Economics," *Philosophy of economics*), pp. 219-240.

Hansson, S. O. 1994. "Decision Theory," *Stockholm: Royal Institute of Technology (KTH)*).

Jensen, C., and Scacchi, W. 2005. "Modeling Recruitment and Role Migration Processes in Ossd Projects," *ProSim'05*), p. 39.

Jensen, C., and Scacchi, W. 2010. "Governance in Open Source Software Development Projects: A Comparative Multi-Level Analysis," in *Open Source Software: New Horizons*. Springer, pp. 130-142.

Jørgensen, N. 2001. "Putting It All in the Trunk: Incremental Software Development in the Freebsd Open Source Project," *Information Systems Journal* (11:4), pp. 321-336.

Kaiser, L. 1989. "Adjusting for Baseline: Change or Percentage Change?," *Statistics in medicine* (8:10), pp. 1183-1190.

Kalpic, B., and Bernus, P. 2002. "Business Process Modelling in Industry—the Powerful Tool in Enterprise Management," *Computers in industry* (47:3), pp. 299-318.

Kessentini, M., Vaucher, S., and Sahraoui, H. 2010. "Deviance from Perfection Is a Better Criterion Than Closeness to Evil When Identifying Risky Code," *Proceedings of the IEEE/ACM international conference on Automated software engineering*: ACM, pp. 113-122.

Ko, R. K., Lee, S. S., and Wah Lee, E. 2009. "Business Process Management (Bpm) Standards: A Survey," *Business Process Management Journal* (15:5), pp. 744-791.

Kogut, B., and Metiu, A. 2001. "Open-Source Software Development and Distributed Innovation," *Oxford Review of Economic Policy* (17:2), pp. 248-264.

Lainhart IV, J. W. 2000. "Cobit™: A Methodology for Managing and Controlling Information and Information Technology Risks and Vulnerabilities," *Journal of Information Systems* (14:s-1), pp. 21-25.

Martínez-Torres, M. R., and Diaz-Fernandez, M. 2014. "Current Issues and Research Trends on Open-Source Software Communities," *Technology Analysis & Strategic Management* (26:1), pp. 55-68.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)* (11:3), pp. 309-346.

O'Mahony, S., and Ferraro, F. 2007. "The Emergence of Governance in an Open Source Community," *Academy of Management Journal* (50:5), pp. 1079-1106.

Peterson, M. 2009. *An Introduction to Decision Theory*. Cambridge University Press.

Rapoport, A. 2013. *Decision Theory and Decision Behaviour: Normative and Descriptive Approaches*. Springer Science & Business Media.

Rosemann, M., and De Bruin, T. 2005. "Application of a Holistic Model for Determining Bpm Maturity," *BP Trends*), pp. 1-21.

Sadiq, S., Governatori, G., and Namiri, K. 2007. "Modeling Control Objectives for Business Process Compliance," in *Business Process Management*. Springer, pp. 149-164.

Sahibudin, S., Sharifi, M., and Ayat, M. 2008. "Combining Itil, Cobit and Iso/Iec 27002 in Order to Design a Comprehensive It Framework in Organizations," *Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on*: IEEE, pp. 749-753.

Schaarschmidt, M., Walsh, G., and von Kortzfleisch, H. F. 2015. "How Do Firms Influence Open Source Software Communities? A Framework and Empirical Analysis of Different Governance Modes," *Information and Organization* (25:2), pp. 99-114.

Sen, R., Singh, S. S., and Borle, S. 2012. "Open Source Software Success: Measures and Analysis," *Decision Support Systems* (52:2), pp. 364-372.

Shah, S. K. 2006. "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development," *Management Science* (52:7), pp. 1000-1014.

Silverman, M. G. 2008. *Compliance Management for Public, Private, or Non-Profit Organizations*. McGraw Hill Professional.

Smirnov, S., Reijers, H. A., Weske, M., and Nugteren, T. 2012. "Business Process Model Abstraction: A Definition, Catalog, and Survey," *Distributed and Parallel Databases* (30:1), pp. 63-99.

Spreitzer, G. M., and Sonenshein, S. 2004. "Toward the Construct Definition of Positive Deviance," *American Behavioral Scientist* (47:6), pp. 828-847.

Suriadi, S., Wynn, M. T., Ouyang, C., ter Hofstede, A. H., and van Dijk, N. J. 2013. "Understanding Process Behaviours in a Large Insurance Company in Australia: A Case Study," *Advanced Information Systems Engineering*: Springer, pp. 449-464.

Swinnen, J., Depaire, B., Jans, M. J., and Vanhoof, K. 2012. "A Process Deviation Analysis–a Case Study," *Business Process Management Workshops*: Springer, pp. 87-98.

Van Der Aalst, W. 2011. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Science & Business Media.

van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H., Weijters, A., and Van Der Aalst, W. M. 2005. "The Prom Framework: A New Era in Process Mining Tool Support," in *Applications and Theory of Petri Nets 2005*. Springer, pp. 444-454.

Van Rossum, G., and Drake Jr, F. L. 1995. *Python Reference Manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Von Krogh, G., Haefliger, S., Spaeth, S., and Wallin, M. W. 2012. "Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development," *MIS Quarterly* (36:2), pp. 649-676.

Weill, P., and Ross, J. W. 2004. *It Governance: How Top Performers Manage It Decision Rights for Superior Results*. Harvard Business Press.

White, S. A. 2004. "Introduction to Bpmn," *IBM Cooperation* (2:0), p. 0.

Xue, Y., Liang, H., and Boulton, W. R. 2008. "Information Technology Governance in Information Technology Investment Decision Processes: The Impact of Investment Characteristics, External Environment, and Internal Context," *MIS Quarterly*), pp. 67-96.

Zeng, Z., Tung, A. K., Wang, J., Feng, J., and Zhou, L. 2009. "Comparing Stars: On Approximating Graph Edit Distance," *Proceedings of the VLDB Endowment* (2:1), pp. 25-36.