Revealing Software Development Work Patterns with PR-Issue Graph Topologies

CLEIDSON R. B. DE SOUZA, Universidade Federal do Pará, Brazil
EMILIE MA, The University of British Columbia, Canada
JESSE WONG, The University of British Columbia, Canada
DONGWOOK YOON, The University of British Columbia, Canada
IVAN BESCHASTNIKH, The University of British Columbia, Canada

How software developers work and collaborate, and how we can best support them is an important topic for software engineering research. One issue for developers is a limited understanding of work that has been done and is ongoing. Modern systems allow developers to create Issues and pull requests (PRs) to track and structure work. But, developers lack a coherent view that brings together related Issues and PRs.

In this paper we first report on a study of work practices of developers through Issues, PRs, and the links that connect them. Specifically, we mine graphs where the Issues and PRs are nodes, and references (links) between them are the edges. This graph-based approach provides a window into a set of collaborative software engineering practices that have not been previously described. Based on a qualitative analysis of 56 GitHub projects, we report on eight types of work practices alongside their respective PR/Issue topologies.

Next, inspired by our findings, we developed a tool called WorkflowsExplorer to help developers visualize and study workflow types in their own projects. We evaluated WorkflowsExplorer with 6 developers and report on findings from our interviews.

Overall, our work illustrates the value of embracing a topology-focused perspective to investigate collaborative work practices in software development.

CCS Concepts: • **Software and its engineering → Collaboration in software development**; • **General and reference → Empirical studies**.

Additional Key Words and Phrases: work practices, pull-based development, pull requests, issues, graphs

ACM Reference Format:
Cleidson R. B. de Souza, Emilie Ma, Jesse Wong, Dongwook Yoon, and Ivan Beschastnikh. 2018. Revealing Software Development Work Patterns with PR-Issue Graph Topologies. In *ICSE '24: 46th International Conference on Software Engineering, April, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 22 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Pull-based development is an important paradigm for software development [13], that has been adopted by sites like GitHub. Developers create Issues, or tickets, and bundle code commits into Pull Requests (PRs) that can be reviewed and discussed before being merged. However, complex development work spans multiple issues and PRs. And developers lack the tools to provide them with a coherent view of related issues and PRs. Such a view would allow them to better understand their software development processes and, consequently, make informed decisions about how to change them. In this paper, we consider two questions: first, what are the underlying work

FSE 2024, July 2024, Porto de Galinhas, Brazil
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

practices that involve multiple interlinked issues/PRs? And second, what is the best way to support developers in investigating their issues/PR work practices?

Previous work considered Issues and PRs in *isolation* [13, 22]. For instance, by automatically identifying duplicate Issues [4] and PRs [36]. However, Issues and PRs are coupled in practice: Issues are frequently resolved with PRs, and PRs are associated with Issues. Researchers have also studied *connections* (links)[1] between PRs and Issues. In this case, Li and colleagues [18] described why software developers create links, while Hirao et al. [16] show that linking information can improve code review tools. However, previous work has a limited view of the PR-Issue ecosystem because it mainly considers edges between just two nodes, when in reality, an Issue or a PR might have *several* links [35]. In aggregate, inter-linked PRs and Issues create a *PR-Issue graph.* To the best of our knowledge, we are the first to consider PR-Issue graphs and to design and evaluate a tool to reveal these graphs to developers.

In this paper, we have two goals. First, we want to understand PR-Issue graphs and the software development work practices they reveal. Second, we aim to develop a tool to help developers understand PR-Issue graphs in their projects.

To achieve this, we first characterized PR-Issue graphs and conduct a qualitative study [7] of PR-Issue graphs from 56 open-source GitHub projects; these projects were based on work by Chopra et al. [6]. We used a qualitative approach because we needed to understand the reasons why the edges (links) between two nodes (PRs and Issues) were created, and, more importantly, how larger sets of nodes were inter-connected. Inspired by our findings, we then built WorkflowsExplorer, a tool to reveal PR-Issue graphs to developers. We evaluated WorkflowsExplorer in interviews with 6 developers who have used the tool.

Contributions. Our paper makes the following three contributions:

- ★ We found that most of the links in our dataset are not the GitHub built-in *fixes* or *duplicates* links. Across the links from all projects we studied, only 12.11% of the links are *fixes* (2.32% for *duplicates*) links on average. This shows that most of the user-created links are not the built-in ones. Based on our analysis of the links from a 64-component sample, we report nine types of linking relationships (see Table 1), complementing previous work [18] with 3 new linking types.
- ★ After analyzing the 64 components (*not* links), we identify eight *workflow types* that capture distinct aspects of collaborative work in software development (see Figure 5). A workflow type is an abstraction that we contribute to represent the identified software developers' work practices. We associate these workflow types with PR-Issue *topologies.* For each workflow type, we provide a description, examples, variations (if they exist), and a topological representation. Our analysis of the workflow types reveals collaborative work practices of articulation work [34], optimization, complexity management, reuse and waste [31], with the first 3 ones not being reported in previous studies.
- ★ We developed *WorkflowsExplorer*, a tool to help developers visualize and understand workflow types in their projects or other projects. We interviewed six developers who have used WorkflowsExplorer. They reported that our tool could help them make more informed decisions about their development processes, training needs, and other aspects.

Next, we review relevant prior work.

[1]In the rest of this paper, we use *edges* and *links* interchangeably.

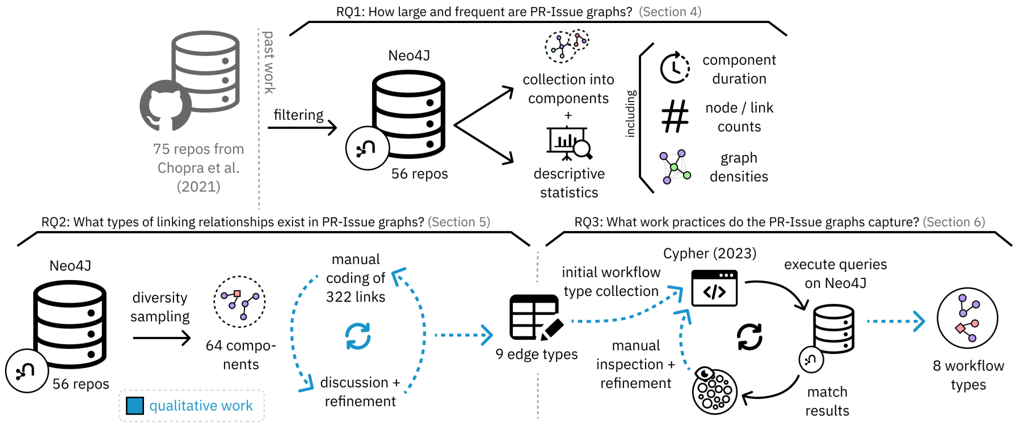
RQ1: How large and frequent are PR-Issue graphs? (Section 4)
75 repos from Chopra et al. (2021)
past work
filtering
Neo4J
56 repos
collection into components
+
descriptive statistics
including
component duration
node / link counts
graph densities
RQ2: What types of linking relationships exist in PR-Issue graphs? (Section 5)
Neo4J
56 repos
diversity sampling
64 components
manual coding of 322 links
discussion + refinement
qualitative work
9 edge types
RQ3: What work practices do the PR-Issue graphs capture? (Section 6)
initial workflow type collection
Cypher (2023)
execute queries on Neo4J
manual inspection + refinement
match results
8 workflow types

Fig. 1. Overview of our methodology to answer RQs 1-3.

2 RELATED WORK

2.1 PR and Issue Comments and Links

Li and colleagues [18] describe an empirical study of the links between PRs and Issues in 16,584 GitHub projects. They manually coded a sample of the links and identified six types of linking relationships. Using regex expressions, they automatically coded the rest of the links and combined these links with information about the degree of the nodes to analyze the graph of all the projects studied. Their results suggest that information about the graph topology could be helpful for developers (e.g., to identify high-priority/core Issues within a project).

More recently, Chopra et al. [6] studied referential behavior in pull-based development by studying 450 PRs and about 7,000 references in these PRs. They observed that developers often create references to source code elements but also reference other developers, the version control platform (branches, PRs, Issues, comments, etc), compilation and execution results (like errors and warnings), and relevant associated project, client or third-party documentation. They also observed that *merged* PRs frequently reference Issues, users, and tests.

The above work studies the relationships between *pairs* of software artifacts. By contrast, our work focuses on the *graph* comprised of PRs and Issues, and the links between them.

2.2 Software Traceability

In the context of software traceability, Rath and colleagues [29] presented an approach that uses temporal and structural relations to identify *missing* links between Issues, commits, and source code. This is important because out of the six large projects they studied, *"on average only 60% of the commits are linked to issues."* This limits the traceability between artifacts in a project. In our work, we take into account Issues and PRs (not commits or source code) and focus on *existing* links.

Nicholson et al. [26] is one of the few efforts to explore graph structures. They studied 66 open-source projects and analyzed two important network properties: (i) the distribution of links showing that Issues with higher degrees (i.e., hubs) are central to the project's requirements; and, (ii) the transitivity property of links to show that it is possible to predict a link type with 72% accuracy.

Nicholson's work differs from our work in three ways. First, they focus only on Issues. Second, their studied Issues are connected by a known set of linking relationships, which allows them to leverage this information in their analysis. In contrast, we explore (i) both Issues and PRs, and (ii) without knowing beforehand the types of links between them. We also (iii) use PR-Issue graphs to infer the collaborative work that takes place in the context of interconnected PRs and Issues.

2.3 Code Reviews

A software engineering area related to our work focuses on the analysis of the comments in modern code reviews. Some authors [2, 11] have studied these comments in isolation. However, more recent work [16, 35] considers the links between code reviews. For instance, Hirao et al. [16] analyzed 6 open-source software communities and concluded that links between reviews vary from 3% to 25% across projects. Furthermore, they found five types of linking relationships. For each of these, they created a subgraph representing files, comments, patches, codebase and the dependencies between these items. They then used five different ML techniques to classify the linking relationships between code reviews. They conclude that these links can be used to improve code review tools, for instance, by suggesting better reviewers.

Wang et al. [35] studied the correlation between review time and number of links in two large projects. Their results indicate that the more internal links, the longer the review time, even when controlling for factors that impact review time like # of added/deleted lines of code, patch size, # of files changed, # of comments, # of reviewers, etc. They also proposed a taxonomy of linking relationships.

By contrast, our work is *not* concerned with code reviews, but focuses on comments in PRs and Issues that contain links.

Some previous studies do use source code and version control information in their analysis, such as work by Hirao et al. [16]. But, we note that empirical studies that do not include this information are also important to carry out, as indicated by previous work [6, 18, 26].

3 METHODOLOGY OVERVIEW

We define a *PR-Issue graph* as a graph in which Issues and PRs are nodes, and the links between PRs/Issues are directed graph edges. For a GitHub project, a node is created for each PR and each Issue. Links between PRs/Issues can be created in three ways in GitHub. First, when a developer writes a comment on a PR or Issue and adds a link to another node. Second, when the link to another PR or Issue is inserted by the developer through GitHub's UI. Third, when the developer uses specific syntax or keywords, GitHub interprets this as a command to create a link between PRs/Issues[2]. Whenever a link to another PR or Issue is detected, we create a directed edge in the PR-Issue graph.

The goal of our study is to (1) explore PR-Issue graphs to find out how they can be used to improve our understanding of the collaborative work taking place during software development; and, (2) use our findings to prototype a useful tool for developers that reveals work practices. Specifically, address the following research questions:

- RQ1: What are the network characteristics of PR-Issue graphs?
- RQ2: What types of linking relationships exist in PR-Issue graphs?
- RQ3: What work practices do the PR-Issue graphs capture?
- RQ4: How do developers perceive PR-Issue graphs and the associated work practices?

Figure 1 overviews our methodology for RQs 1-3. The first part (detailed in Section 4.1), is focused on identifying and characterizing PR-Issue graphs. In the second part (Section 5.1), we qualitatively study a sample of PR-Issue components to identify the types of linking relationships in PR-Issue graphs. In the third part (Section 6.1), informed by these linking types, we qualitatively identify

[2]Here are the keywords adopted by GitHub: <https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/using-keywords-in-issues- and-pull-requests>

and illustrate collaborative work practices associated with pull-based software development. We address RQ4 in Section 7.

To answer RQs 1-3, we studied PR-Issue graphs from a sample of GitHub projects. Before conducting our qualitative analysis, we first mined the repositories of 56 projects to recover node (PRs and Issues) and edge (links) information to construct our sample of PR-Issue graphs. Answering RQ4 required a different methodology, which we describe in Section 7.2.

We adopted a qualitative approach to understand the comments containing links between PRs and Issues, the context in which these comments were created (e.g., other comments, authors, dates), and more importantly, the other PR and Issues that compose the graph in which the comment is embedded. Specifically, we adopted a reflexive thematic analysis approach [7]. A quantitative approach was not suitable to answer these RQs because it was necessary to understand the context (PRs, Issues, their comments, etc) in which the collaborative work took place to characterize the different link types. And, as mentioned in the previous section, prior work [16, 18] has automated the classification of *links* between *two* PRs and Issues, but we needed to classify a *graph* that spans multiple PRs and Issues, which means these automated approaches would not work for us.

The lead investigator led the qualitative analysis, while other team members periodically reviewed the results to establish consensus and resolve disagreements. We held weekly meetings to discuss preliminary results from the lead author, including quotes, codes, and themes and reach a negotiated agreement as a group [12]. The other authors thoroughly reviewed and questioned these intermediate results to identify gaps, misunderstandings, assumptions, and potential biases in the data analysis. Based on this feedback, the lead author revisited the data and refined the results accordingly. With this process, we aimed to minimize researcher bias by consistently verifying the findings. As we will show in Section 5.2, our identified linking relationships closely resemble those found by Li et al. [18], indicating that we effectively reduced potential biases.

More precisely, we used *constant comparison* during qualitative analysis [34]. This method recommends making comparisons during each qualitative analysis stage of whatever unit of data is being analyzed [5]. For instance, the lead author compared *comments* in different nodes to identify link types (RQ2). In another example, (s)he compared instances of *sets of nodes and links* to make sure they describe the same workflow type (RQ3). Furthermore, we sampled the "items" used for comparison to refine the results, and not for population representativeness, that is, we adopted an approach similar to theoretical sampling [34]. Again, the other authors compared the partial results presented by the lead author, for instance, by comparing whether the *workflow types* described similar or different work practices. As in any qualitative study, this was a reflexive and iterative process [9].

4 RQ1: PR-ISSUE GRAPHS CHARACTERISTICS

The goal of this part of the study was to mine and characterize PR-Issue graphs. To do so, we studied the PR-Issue graphs from 56 GitHub open-source projects (Figure 1). We downloaded and studied a subset of the GitHub repositories studied by Chopra et al. [6]. These authors studied references in PR discussions. For this, they selected a representative sample of 75 projects from a population of 7,000 projects. To create this sample, they used Nagappan et al.'s [24] algorithm that generates a diverse sample of projects by accounting for dimensions like number of developers, main programming language, project domain, recent activity, project age, and others[3]. In fact, these 75 projects had 40.8% of the total variance. From these 75 repositories, we excluded those that have

[3]More specifically, Nagappan et al.'s [24] introduced the notion of coverage, defined as *"the percentage of projects in a population that are similar to a given sample."* This is done by comparing projects along various dimensions. In addition, they provided the implementation of a greedy algorithm for computing the concept of coverage.

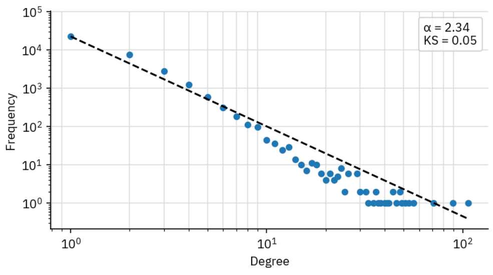

Fig. 2. Degree distribution across nodes in all studied GitHub projects. α is the power-law exponent, and KS is the Kolmogorov-Smirnov statistic.

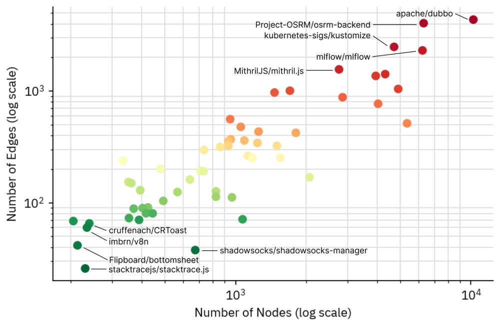

Fig. 3. Log distribution of nodes versus edges. Each point is a project; colour indicates the number of edges in a project.

been deprecated or archived. We also excluded projects with fewer than a sum total of 100 PRs and Issues. We did this to remove small PR-Issue graphs. This resulted in 56 projects.

4.1 Extracting PR-Issue Graphs

GitHub allows a user to specify two types of links. The first one is the type *fixes* or *closes*, as in this PR fixes a bug specified in a particular Issue. These are implemented in GitHub using a set of keywords. This relationship is considered prototypical on GitHub, as it is the main type that is supported by GitHub's linking feature. The second type of linking is *duplicates*, which indicates that an Issue is a duplicate.

We created a crawler to extract these link types and other information from the 56 projects. This crawler, using GitHub APIs, extracts PRs and Issues and detects and extracts the links found in the comments of the PRs and Issues. The crawled data is used to construct our PR-Issue graphs, mapping PRs and Issues to nodes and the links between them to edges.

Our crawler mapped links as being *fixes* or *duplicates*, if they were specified as such. Any links that are not any of those types were classified as *other* edges. If a particular node did not contain any edges, this produced an isolated node. The final PR-Issue graph was be the aggregation of all edges and nodes (isolated or not).

As we will discuss later in Section 5.1, each resulting PR-Issue graph is composed of a set of *connected components* [32]. In total, we studied 56 PR-Issue graphs (projects) that contained 64,581 components. These graphs were imported into Neo4J[4], a graph database, for later topological querying with Neo4J's query language, Cypher. We also imported metadata associated with each node, including `type` (`pull_request` or `issue`); `status`, (`open`, `closed`, or `merged` for pull requests); `repository`; `number` (the node identifier, the same as its GitHub ID); and the node's creation, update, and close timestamps. As mentioned, for each link, we imported its `link_type`, based on GitHub's explicit links. We also tracked the `user` who created the node or link.

4.2 Characteristics of PR-Issue Graphs

The 56 projects we studied used different programming languages, have different goals, vary in size, etc. As part of our analysis, we wanted to find out if the PR-Issue graphs from each project have similarities. Therefore, following Nicholson's approach [26], our first investigation focused on

[4]https://neo4j.com

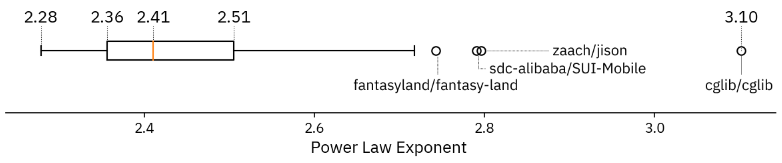

Fig. 4. Power law exponent distribution across all components. The boxplot shows the 25/75th percentiles, with whiskers extending to up to 1.5 IQR.

the degree distribution of the nodes[5]. Figure 2 shows that this degree distribution in the projects follows a power-law distribution with an exponent of 2.34. In other words, as the degree of a node increases, the proportion of nodes with that degree decreases exponentially.

Figure 4 shows the power law exponent across the projects we studied. All projects have an exponent larger than two, and one outlier project has an exponent of 3.1. This suggests that across all the projects we consider, a small number of nodes is connected to several other nodes.

A common explanation for power-law degree distributions is the concept of a scale-free network that is generated by preferential attachment: nodes with higher degrees (more edges) receive even more edges than those who have smaller degrees [3]. Intuitively, this means that some Issues or PRs (e.g., those that are more complex, critical, or controversial) will accumulate more links [26]. Therefore, these nodes will be part of components with a larger number of nodes since more nodes imply more edges, as shown in Figure 3. This figure shows how the projects we studied vary along two dimensions: the number of nodes and the number of edges. It is interesting to note that the project with the most nodes and edges (*dubbo* at top right) uses a bot that creates an Issue with links to all merged PRs every week. We will discuss the impact of this bot in our results in Section 6.9.

5 RQ2: LINKS IN PR-ISSUE GRAPHS

The *links* between PRs and Issues are an important artifact which gives rise to PR-Issue graphs. We study these links qualitatively by considering a sample of 64 components (see Figure 1). We start by detailing our sampling and qualitative approaches.

5.1 Analysing Links in PR-Issue Graphs

Figures 4 and 3 show that the 56 projects studied vary along several dimensions. So, to properly answer RQ2, we need a sample of components from these projects with different features: sizes, topologies, sparseness, and Github activity. For this, we adopted Nagappan et al. [24] sampling algorithm again. As mentioned in Section 4.1, this prior work provides an algorithm that aims to compute a sample that best covers a population according to different dimensions. We use this algorithm *not* to identify *projects* since we had already done this (see Section 4.1). Instead, we use this algorithm to select *components* from these projects. We took into account: the number of nodes (PRs and Issues) in the component, the number of GitHub comments from all of the nodes within a component, the number of GitHub users responsible for creating the nodes within the component, edge density of a component [32], and the diameter of a component [32].

The result of the algorithm was a final sample containing 64 components. For each component, we listed all links between each pair of nodes. This resulted in 322 unique links. The lead investigator qualitatively coded [7] all the links into different types of linking relationships and double-checked

[5]While the degree of a node describes the number of edges this node has, the degree distribution describes the frequency of each degree value.

all *fixes* and *duplicates* links. The coding process was conducted by component: all the links from one component were coded before moving to the next component. To identify the link type, the lead investigator read the comments associated with the PRs and Issues of that component to understand the broader context of the links. The types of nodes (PRs or Issues) being connected by the link were also taken into account during this analysis.

The coding process occurred in several rounds and whenever a new type of linking relationship was created, all previous links were re-analyzed for this new type. Similarly, old types of linking relationships were merged to create new ones, and when this happened, all instances of the old type were double-checked to ensure the new linking type was appropriate. As mentioned in Section 3, we took several actions to minimize researcher bias. In the end, we identified a set of nine types of linking relationships described in the next section.

5.2 Types of Links in PR-Issue Graphs

After coding our sample of components from PR-Issue graphs, we identified nine types of links. Table 1 lists these types and also presents quotes of comments from PRs or Issues to illustrate each relationship type. As we will discuss in Section 6.1, identifying types of linking relationships helped us to identify workflow types.

Table 1. List of types of relationships

Relationship type	**Connected nodes**	**Description**	**Example comment**	**Prevalence: N, %**
Fixes / Is fixed by	Issue ⇔ PRs	The PR provides a solution that closes the Issue, or the Issue is closed by the solution provided by the PR.	*"This PR fixes #194"*	115, 35.7%
Is contextualized by	Nodes ⇒ Nodes	The linked node provides information (e.g., an example, a limitation, additional information, etc) that is useful to understand the origin node.	*"I found this comment from* [username] *from Nov 18, 2013 'Make sure that you load the correct files. Right now, you will need around 48 GB of RAM to run the server on the planet-wide dataset.' (#802)"*	66, 20.5%
Is a duplicate of	Issues ⇒ Issues	The origin Issue is a duplicate of the target Issue.	*"Same issue is detailed here, also with a gif of desired result #385"*	33, 10.2%
Improves / Is improved by	PRs ⇔ PRs	The target (origin) PR provides an implementation that replaces, extends, follows up or is a better alternative than the origin (target) PR.	*"We added new alembic.ini config files & moved around existing ones in #1292. This PR updates wheel build logic to include the moved .ini files in the wheel - otherwise running mlflow db upgrade from a wheel built via python setup.py bdist-wheelfails while trying to find the config file"*	32, 9.9%
Is similar	Issues ⇒ Issues or PRs ⇒ PRs	The target Issue (PR) describes a problem (solution) similar to the one described by the origin Issue (PR).	*"The symptoms appear to be similar to #3711 except that they are now not caused by base-url."*	22, 6.8%
Is irrelevant	Nodes ⇒ Nodes	The content of the origin Node is irrelevant given the linked Node.	*"There is an old issue about this, with MySQL, but it is closed and supposedly solved: #1397"*	17, 5.3%
Impacts	Nodes ⇒ Nodes	The linked Node is created or reopened because of something from the origin Node.	*"Hello, I would like reopen issue #3537 because with summernote 0.8.20 the same problem occurs again."*	17, 5.3%
Uses	Issues ⇒ Issues or PRs ⇒ PRs	The origin Node tried to adopt or adopted the same solution described in the linked Node.	*"Fix the Python version to avoid differences in behavior for users with conda2/conda3, mirroring changes we made in #96"*	12, 3.8%
Blocks	PRs ⇒ PRs	The target PR requires the origin PR to work.	*"As discussed with* [username], *this change should help unblock #2367"*	8, 2.5%

In our sample, we observed that 35.7% of relationship types were *fixes*, which is expected since this is the prototypical relationship in GitHub, indicating that a PR closes an Issue. The second most common relationship type, *Is contextualized by*, with 20.5% of the studied links, describes a scenario where the link points to a linked node that contains useful information to the origin node. For instance, the link might point to a comment in a different Issue that provides information about the configuration of the test environment. Given its broad nature, it is not surprising that this relationship type was identified so often. In contrast, given the interdependent nature of software artifacts [10], it is surprising to find such a low occurrence of both *Block* (2.5%) and *Impacts* (5.3%) relationships because these link types capture dependency between software artifacts. The frequency of *Is a duplicate of* (10.2%) and *Is irrelevant* (5.3%) relationships is also notable since these reflect waste in software development [31]. Finally, the *Improves*, *Is similar*, and *Uses* relationships together account for 20.5% (9.9+6.8+3.8) of the links, indicating a non-trivial amount of reuse.

The second column in Table 1 illustrates how the link types depend on the node types. Specifically, the *Is a duplicate of* relationship was only observed between Issues, *Improves* and *Blocks* link types were only between PRs, and *Uses* and *Is similar* relationships were observed either between PRs or between Issues (but not between PRs *and* Issues). By contrast, the *Fixes* relationship was only observed between different types of nodes. Finally, *Is contextualized by*, *Is irrelevant* and *Impacts* relationships were observed between all types of nodes. The same second column indicates that two types of relationships are undirected [32]: *Fixes* and *Improves*. In these types the origin or target of the relationship is irrelevant. In all other link types, the linked node is clearly indicated in the description; therefore, the direction of the relationship is relevant. For instance, in the *Is a duplicate of* relationship, the origin Issue is the one that is the repetition of the target Issue.

Table 2 compares Li's [18] classification and distribution of link types to our results. It shows that the relationship types have a nearly identical frequency distribution (*Blocks* is an exception). We also identified three additional relationship types: *Is irrelevant*, *Impacts*, and *Uses*, which suggests that our classification allow us to identify more nuanced types of collaborative work. We discuss this further in Section 8.

Table 2. Comparison of link types and their frequencies between Li's et al. [18] and our work. Link types are ordered in decreasing frequency order of our work.

Li's et al. [18]	Frequency	**Our work**	Frequency ↓
Fixes	40.4%	*Fixes*	35.7%
Reference	16.6%	*Is contextualized by*	20.5%
Duplicates	14.6%	*Is a duplicate of*	10.2%
Enhances	7%	*Improves*	9.9%
Relevant	5.2%	*Is similar*	6.8%
N/A	0%	*Is irrelevant*	5.3%
N/A	0%	*Impacts*	5.3%
N/A	0%	*Uses*	3.8%
Dependent	6.5%	*Blocks*	2.5%
Other	8.9%	*N/A*	0%

6 RQ3: PR-ISSUE TOPOLOGIES AND WORKFLOW TYPES

Now that we have considered the linking relationships, we are ready to consider PR-Issue topologies. We qualitatively analyzed PR-Issue components to understand the underlying collaborative pull-based work practices (see Figure 1). We call these practices *workflow types*. A workflow type is presented with a description, a set of examples that illustrate its variations (e.g., same vs. different node authors, order in which nodes are created, etc), and a topological representation. This topological representation includes the minimal set of PRs and Issues required to model the work practice. Next, we further explain the methodology we adopted to identify the workflow types.

6.1 Identifying Workflow Types

As mentioned, we qualitatively analyzed all 64 PR-Issue components. We conducted our coding [7] by considering one component at a time. The coded links, and graphical component topologies, along with PR and Issue meta-data all informed our coding process. Specifically, the lead investigator identified workflow types in rounds, discussed their results with the co-authors and successively refined the types (see Figure 1). Once a workflow type was identified, the set of previously analyzed components was inspected to verify the existence of other instances of a new type. Each new workflow type was also mapped to a prototypical topology. For example, when we identified the *Decomposing Issue* workflow type, we captured it using a *closed* Issue fixed by multiple *merged* PRs. Using this topological description, we queried our PR-Issue graph database to find other instances of this topology to find more *potential* examples of this workflow type. These instances were qualitatively checked to find out whether they did or did not match our definition of the associated workflow type. In summary, we used constant comparison [7] to support, contradict or expand our definitions of workflow types and their topologies. To help find instances for comparison we issued queries against the Cypher database.

During this process, when possible, we refined the definition of workflow types by including either temporal or authorship constraints. These constraints helped us to further refine our workflow type definitions. The final list of eight workflow types is presented in the following sections. For each type, we give its definition, an example, variation(s) (when they exist), and the topology. Figure 5 visualizes all the workflow type topologies. Our supplementary materials and source code[6] include the definitions and Cypher queries used to identify instances of the topologies.

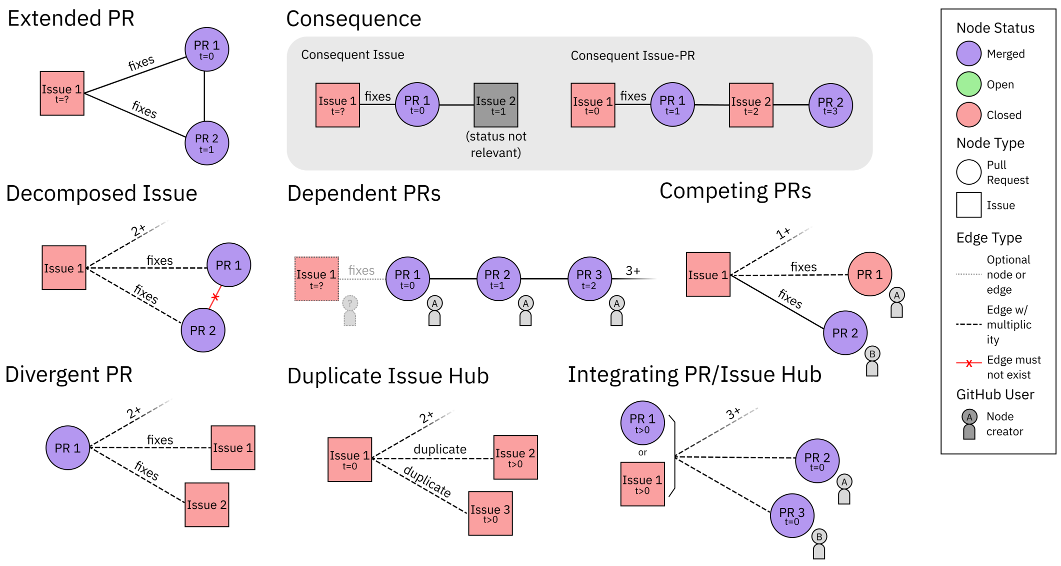

Fig. 5. Topologies of the workflow types we uncovered.

6.2 Extended PR

Definition: This workflow type describes a common aspect of collaborative work in which someone extends previous work, for instance, to correct or improve it. In pull-based development, this means

[6]https://anonymous.4open.science/r/pr-issue-topology-project-D888

that a particular implementation (a merged PR) to a problem or feature (closed Issue) is expanded with additional work (a new merged PR) while still addressing the initial Issue. The topology for this workflow (Figure 5) has two *merged* PRs fixing the same Issue, and there is a link between these PRs.

Example: This workflow appears in the *jupyterhub* project, where we identified PR 842, which corrects PR 813, (which is the first solution to Issue 812). PR 842 is necessary because PR 813 did not consider everything necessary to fix Issue 812. Indeed, while PR 813 was titled *"Add error message when generate config path does not exist"*, PR 842 is titled *"**finish** error message when generate-config path does not exist"* (our emphasis) and contains the following as its first comment:

> *"finish up #813*
> *checks directory existence*
> *exit with message on stderr if not found*
> *closes #812"*

Variation(s): An extended PR can also appear with nodes created by the same person. In project *mithril.js*, the developer created PR 2265 as an improvement to their previous PR 2250, which was in turn created to fix Issue 1986 (also with the same author). PR 2265 is a documentation update with title *"Remove a section that's 1. buggy and 2. controversial"* and contains the following in its description: *"See discussion in #2250 and #1986 for more details".* This Extended PR workflow instance suggests that even when the nodes are created by a single author, they are still embedded in a collaborative process, i.e., our workflow type can still be used to identify a pull-based work practice.

Interpretation of the Work Practice: An Extended PR can represent work that is either detrimental or beneficial. In the examples, an Extended PR represents a form of waste due to rework, i.e., "revising work that should have been done correctly" [31]. However, we also identified Extended PRs that were beneficial because the second PR improved the implementation in the first PR.

6.3 Consequence

Definition: This workflow type indicates how software artifacts influence each other [10], and by doing so, capture how software developers coordinate their work [20]. In our context, this can be translated into a situation where a solution (PR) to an Issue impacts the rest of the project in such a way that a new Issue is created after the PR (see Consequent Issue in Figure 5).

Example: In project *mlflow*, a developer made a comment on Issue 4892 about a soon-to-be-merged PR 4845: *"will add the MLServer to the MLflow project, and I think if you're looking for a fully formed server for use in K8s, it probably makes sense to use that."* In other words, according to the developers, PR 4845 can be used to implement the functionality required in Issue 4892. It should be noted that in this case, this Issue remains open, i.e., the Consequent Issue workflow does not require the Issue to be closed.

Variation(s): In the Consequence workflow, optionally, an additional merged PR might be necessary to address the newly created Issue, i.e., the Consequent PR-Issue in Figure 5. This can be observed in project *ts-node.* As requested by Issue 1229, PR 1232 implements the requirement: *"always throw ERR_REQUIRE_ESM when attempting to execute ESM as CJS, even when ESM loader is not loaded."* However, Issue 1342's author realizes that this PR creates a significant problem for projects with a particular configuration. This problem is fixed by PR 1371. Interestingly, Issue 1229, PR 1232, Issue 1342, and PR 1371 were all co-authored by the same developer. However, PR 1232 and Issue 1342 were created about three months apart, i.e., the actual consequence of the PR was only perceived later.

Interpretation of the Work Practice: The two examples above illustrate another aspect of the Consequence workflow type. The first example can be interpreted as having a *positive* impact on the project, i.e., by fixing Issue 4892, PR 4895 allows new features to be discussed and potentially implemented in *mflow*. However, this workflow can also have a *negative* impact: in the *tsnode* example, PR 1232 created a problem that was later documented in Issue 1342.

6.4 Decomposed Issue

Definition: Complex tasks are frequently divided into simpler sub-tasks. In pull-based software development, this means that the work required to close a single Issue may be distributed across multiple PRs. This workflow models this work practice and is translated into a topology in which a *closed* Issue is *fixed* by more than one *merged* PR (see Figure 5) Based on our observations, these PRs are not linked to each other. As we will discuss later, this decomposition also facilitates the process of reviewing PRs.

Example: An illustrative example of this workflow type can be found in project *osrm-backend*, which is part of the OpenStreetMap routing engine. In this example, Issue 5067 was opened to document the need to provide *"routing over barrier=kerb which might be tagged e.g. on driveways"*. This Issue is closed by two different PRs that address different transportation modes: PR 5076 implements the routing solution for *bicycle* users, while PR 5077 focuses on *walking* users. Upon inspection of the files changed in each PR, we noticed that they have identical changes but in different files. These PRs have the same developer as their author.

Another instance of this workflow type is in project *kustomize* where Issue 2761 is fixed by two different PRs: 2762 and 2769. PR 2762 includes the comment *"**Partially** fixes #2761"*(emphasis added) suggesting that the work required to close the Issue is *not* complete. In this instance, the Issue and the two PRs were created by the same developer over two days.

Interpretation of the Work Practice: This particular workflow type models a good and well-known software engineering practice in which work is divided into smaller pieces to facilitate its execution, improve documentation, decouple modules, etc.

6.5 Dependent PRs

Definition: The Dependent PR workflow describes a situation in which pull requests are connected to other pull requests in a chain. Usually, the chained PRs have the same author, or a limited set of authors. This workflow facilitates the merging process across different branches. At the high-level, this workflow type is similar to the Decomposed Issue workflow in that it models a work practice to break down tasks into smaller, more manageable, parts.

Example: The *mlflow* project contains an example of Dependent PRs. PR 5092 provided an initial implementation for a required feature. After a sequence of comments and changes, the author of this PR added a comment to this PR *"Create an example code PR: #5186"*, indicating that PR 5186 was created to provide an example to the change implemented in PR 5092. In short, the first PR implemented a new feature, while the second PR implemented an example that uses this feature.

Variation(s): The work split into two or more PRs might not be decided by the author of the first PR. For instance, in the *tiny-dnn* project, one author created PR 979 to implement a new feature. In the comments on this PR, another developer (reviewer1) suggested that the PR should be broken into two parts: @author [the changes] *"LGTM.* [looks good to me] *However, you should split the commit one for padding and the other to update the authors script."* Accordingly, PR 989 was created as indicated in the subsequent comments:

> @reviewer1 *The script for updating authors was removed.*
> *The AUTHORS file was reverted to the original version.*
> *Please consider merging this PR.*
> (...)

> @reviewer1 @reviewer2
> *Please also review PR #989.*

Interpretation of the Work Practice: It is important to note that the Dependent PRs workflow is also called Stacked, Incremental, or Chained PRs. It is regarded by many practitioners as a good software development practice: a large change is broken into smaller, easier-to-review PRs that depend on each other [1, 8]. Interestingly, to the best of our knowledge, no empirical studies have explored this work practice.

6.6 Competing PRs

Definition: The Decomposed Issues and the Dependent PRs both capture divide-and-conquer workflows. However, we also identified poor, or sub-optimal, work practices. For instance, when developers are not aware of each other's work and end up duplicating work [14]. We call one such workflow *Competing PRs*; it describes a situation where there are alternative implementations (PRs) for the same Issue. Since these PRs aim to close the same Issue, only one of them is merged. In addition, these PRs are created by different developers. Its topology is presented in Figure 5.

Example: We observed this workflow type in project *discord.py*. In this case, Issue 1874 had two competing PRs: 6453 and 6462, each one created by different developers. Interestingly, the PR accepted (6462) was created second.

While refining this workflow type, we identified a *potential* instance of this workflow type in *mlflow* where the competing PRs occurred more than a year apart. Upon inspection, we observed that this was not *not* an instance of Competing PRs. We therefore decided to constrain the creation date of the PRs so that they are all created within a week interval. Future work should explore how to better identify the most appropriate time-frame according to specific project's information.

Interpretation of the Work Practice: One way to view the Competing PRs workflow is that it enables open source projects to select the best implementation [30]. However, recent research shows that there are different reasons influencing whether contributions (PRs) are accepted [23, 28, 33]. We therefore consider Competing PRs as a form of waste in software development, potentially due to poor communication [31].

6.7 Divergent PR

Definition: This workflow type captures a situation in which a developer *merges* a PR that *closes* multiple Issues at the same time. Similarly to the Extended PR, this workflow type can be seen as both *negative* light (when the Issues are unrelated), in or *positive* light (when the closed Issues are related).

Example: PR 886 in project *grpc-web* is an example of the negative aspect of a Divergent PR. The first comment for this PR contains the following:

> *Fixes #848: Fixed a bug where we can't pass the interceptors into the client constructor because of a type mismatch.*
> *Fixes #868: Added MethodDescriptor to the exported types. Replace the deprecated MethodInfo with MethodDescriptor.*
> *Fixes #877: Fixed issue where UnaryResponse symbols are being optimized away.*
> *And in general added some missing classes to the exported types as well.*

Variation(s): It is important to mention that PR 886 and Issues 848, 868 and 877 were created by different authors. However, this is not a restriction. In fact, PR 947 in project *chai* closes Issues 916 and 923 and all these PRs and Issues have the same author.

Interpretation of the Work Practice: This workflow type abstracts a somewhat common practice in software development in which a single commit bundles multiple changes to the codebase.

6.8 Duplicate Issue Hub

Definition: This workflow type models a situation in which developers are not aware of each other's work [14]. This workflow appears when an Issue *I* is connected to several other *closed* Issues using the *is a duplicate of* link, i.e., Issue *I* is a hub. An important temporal constraint is that this hub has been created *before* the connected Issues. Since it is unlikely that developers will duplicate Issues they created, another constrain is that the duplicated Issues are created by different authors.

Example: An instance of the Duplicate Issue Hub appears in project *discord.py* where Issue 5867 is connected to 11 (eleven) duplicated Issues! These duplicated Issues are 5889, 5942, 5951, 5955, 5971, 5978, 5980, 6030, 6041, 6047, and 6175. It should be noted that the first Issue, and accordingly all others, were created because of a significant change in the project that was reported in the project change log. And, all these duplicated Issues were reported even though the Issue hub was already closed with a reference (link) to the changelog.

Ramifications for Work: We speculate that the Duplicate Issue Hub workflow is more typical in large open-source projects in which different developers want to contribute and do so by reporting Issues [25]. The problem, as we observed in the above example, is that these duplicated Issues end up wasting work [31].

Interpretation of the Work Practice: This workflow type abstracts a situation where contributors are not aware of previous work and end up reporting issues that already exist.

6.9 Integrating PR/Issue Hub

Definition: During the inspection of the components to identify workflow types (see Figure 1), a topology we often observed was a node connected to several *merged* PRs by at least two different developers, so that the node is a hub. This node was usually created after all of its connected PRs. Upon inspection, we observed that this workflow occurred when developers documented, either as an Issue or a PR, (i) a release, (ii) large changes in the codebase (e.g., a large refactoring), (iii) reports of tests that failed in important branches or (iv) bots.

Example: One example can be found in project *Polly*, where PR 373 is titled *"Merge development branch for v5.6.0"* and contains links to 10 different merged PRs. In the Apache project *dubbo*, we observed a bot that provided a weekly report of the activities for the project. This report included basic data about *"Issues & PRs show the new/closed issues/pull requests count in the passed week."* as well as a list of all PRs merged in the repository in a given week. One of the examples of the Integrating PR/Issue hub for this project is Issue 2466, which lists 24 merged pull requests between 2019-5-31 and 2019-6-7. Finally, an example of an Integrating Issue Hub being used to report tests failing in important branches includes Issue 3734 in project *jupyterhub* that is titled *"Tests are failing on main branch"* and is connected to 7 merged PRs.

Upon querying our dataset for other instances of this workflow type, we even found an instance of an Integrating with PR/Issue hub connecting 70 PRs in project *mithril.js*. The hub PR 2766 is titled *"Release v2.1.0"*.

Interpretation of the Work Practice: In this workflow type a developer links to other developers' contributions to help surface these contributions and to coordinate the development process.

6.10 Interactions between Workflow Types

So far, we presented individual workflow types. But, in practice, they occur concurrently. For instance, Figure 6 shows a PR-Issue graph from project *pundit* with two different workflow types that have common nodes.

PR 697, Issue 689, and Issue 666 indicate an instance of the Divergent PR workflow. Meanwhile, Issue 666, PR 697, and Issue 723 are an instance of the Consequent Issue workflow. In this case, Issue 723 is created as a "side-effect" of PR 694 while also allowing the Issue observed in PR 697 to be documented and discussed in Issue 723. In other words, Issue 723 was created based on the discussion in PR 697.

Workflow types can also *interact* with each other. For example, one can speculate whether the Consequent Issue above happened *because of* the Divergent PR, i.e., a PR aimed to close multiple Issues but had its quality affected, therefore leading to the creation of a new Issue. We did not explore how often workflow types co-occur and the interactions between them.

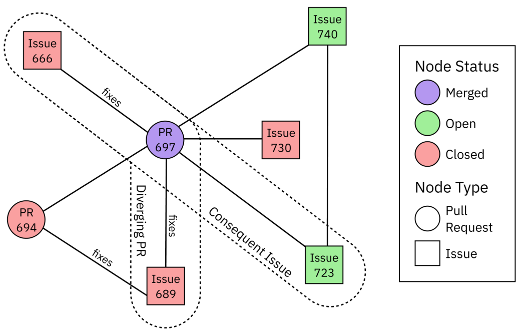

Fig. 6. Two workflow types in a PR-Issue component.

7 RQ4: PERCEPTION AND UTILITY OF PR-ISSUE GRAPHS AND WORKFLOW TYPES

In this part of our study, we wanted to answer RQ4 and understand the value of our approach for software developers. For this, we recruited and interviewed 6 developers (3 open-source and 3 closed-source) who used the tool we developed (see next section).

7.1 Visualizing Workflow Types

To validate our workflow types and enable developers to explore these types in their own projects, we built *WorkflowsExplorer*, a workflow type visualizer tool[7]. Figure 7 shows a partial screenshot of this tool.

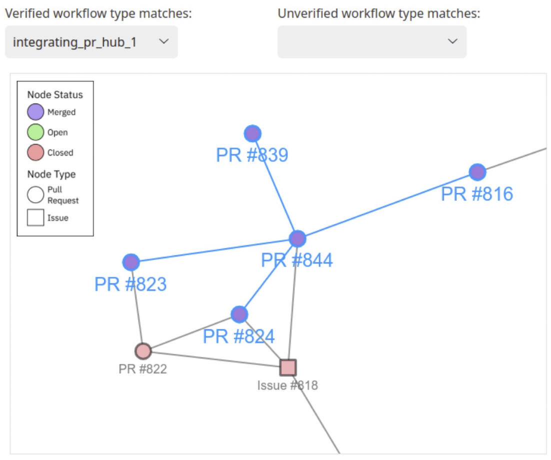

Fig. 7. WorkflowsExplorer applied to the App-vNext/Polly project.

To use WorkflowsExplorer, a user selects a project to inspect, and then chooses an extracted workflow type instance from a dropdown at the top. The tool then displays the selected workflow type instance in the context of its PR-Issue graph. Edges in the workflow type are highlighted in blue while nodes in the PR-Issue graph are color-coded depending on the status of the PR/Issue. The user can investigate the graph in an interactive spatial preview window. They can click on nodes to navigate to the corresponding PR/Issue page on GitHub. The tool also includes a brief description of the selected workflow type (not shown in Figure 7)

7.2 Interview Methodology

Open-source developers. First, we used WorkflowsExplorer to create visualizations of the workflow types for each of the projects we studied. Then, we identified and emailed the core contributors for each project a link to their respective WorkflowsExplorer instance, with an invitation to be

[7]https://anonymous.4open.science/w/pr-issue-topology-project-03EB

interviewed. Next, we selected three contributors to interview. They were compensated CAD $40. These contributors were associated with projects Discord.py, JupyterHub, and gRPC-Web[8]. More details about each project are in the Supplementary Material. In the last stage, we conducted the interviews which we recorded using Google Meet, and later transcribed.

We prepared an interview guide specific to each selected project/interviewee. This guide was divided into three parts. In the first part, we collected information about the developers' development experience; the second focused on the open-source project's usage of PRs and Issues; while in the third one, we used the tool with the interviewees to navigate the workflow types in their own projects. All interviewees had already used WorkflowsExplorer prior to the interview. Interviews lasted about 45 minutes. We focused on four of our eight workflow types: Duplicate Issue Hub, Competing PRs, Extended PR, and Divergent PR. We focused on these types because they capture inefficiencies in software development processes.

Closed-source developers. We used convenience sampling to reach professional software developers in our social networks. These professionals should be using PRs and Issues and be willing to be interviewed for about 45 minutes. We prepared an interview guide like the one for open-source developers but with examples extracted from open-source projects.

We qualitatively analyzed the six transcribed interviews. We conducted our coding [7] one interview at a time. Specifically, the lead investigator identified initial codes associated with parts of the interview and successively refined the codes so that themes were identified. As a new code was identified, the previous transcribed interviews were revisited to identify potential instances of this code. Below, we discuss the themes that we identified in the interviews.

Both open-source and closed-source developers were experienced in software development. Their professional experience ranged between 10 and 20 years. We refer to the open-source developers as Open1, Open2, Open3; and, the closed-source developers as Closed1, Closed2, and Closed3.

7.3 Interview Results

7.3.1 Usefulness of WorkflowsExplorer. All interviewees noted that WorkflowsExplorer is useful and that PR-Issue graphs and workflow types capture important information about the software development process. We probed them further about when the information surfaced by WorkflowsExplorer would be useful: all developers agreed that workflow types would help them make better decisions about different aspects of the software development process. For instance, Open3 suggested that too many Duplicated Issue Hubs might mean a change in the process is warranted:

> *"if you ... see a lot of duplication ... a percentage of, you know, issues being duplicated, then something needs to be done process-wise, you know? documentation and those things.".*

Meanwhile, Closed1 mentioned the following: *"If someone had a lot of bugs there,* [identified with the Extended PR,] *what's going on? Do they need training to improve? Do they need someone with them to do pair programming to help them improve?".* In other words, this quote describes a potential scenario in which additional training might be necessary. Closed1 also mentioned that the frequency of Extended PRs could be used, e.g. in a retrospective meeting, to understand the amount of extra work taking place at the end of a sprint. This information could also be used to find out the extent to which the estimations done at the beginning of the sprint were accurate. This is something that is not currently done in his organization.

Finally, regarding the actual visualization of the workflow types and graphs, Closed2 and Closed1 argued that this information would be more interesting to those responsible for the *business* aspects of development (e.g., a tech lead in their organizations), and not necessarily a "regular" developer. Closed1 argued that such a visualization would be interesting to surface inter-dependencies between

[8]Here are the WorkflowsExplorer links we emailed these participants: discord.py, jupyterhub, and gRPC-web.

requirements (user stories, issues, etc): *"Non-functional stories, for example, that we create to structure the project, and they end up generating an inter-dependence on each other. These need to enter; they need to be developed first, because otherwise, I won't have the project's foundation to start . . . then the interdependence of the PRs begins. This PR is related, this issue is related to another because this one needs to come before it . . . And that's precisely the point, GitHub doesn't give you that."*

7.3.2 Frequency of Workflow Types. All six developers recognized the four workflow types we presented. But, open-source and closed-source developers had different views on the frequency of these workflows. *Open-source* developers indicated that these workflows happen often in their projects. *Closed-source* developers had a different view: they agreed about the Divergent PR (*"That's quite common, but we don't recommend it, however, it does happen."*), but disagreed about the other workflow types. In closed-source development, Duplicate Issue Hubs do not happen with Issues, but instead with other tools used in their projects (e.g., an incident management tool used by Closed1). Closed2 also mentioned that this duplication happens at a team level. Most importantly, Competing PRs do not happen because developers are allocated to work on Issues at the beginning of each sprint by the Project Manager or Project Owner. Finally, Extended PRs might or might not occur, depending on the business logic relevant to the PR.

Interestingly, the Decomposed Issue workflow type was not presented to Open3, but despite that, he described it and mentioned that it does occur in his work. According to him, in a recent sprint: *"Besides needing an API here, we needed to build two more APIs to communicate with other things, so the complexity is high. This happened with us now in project [X], so the complexity was quite high, and three developers participated to finish everything in two weeks. So, the previous sprint was quite complex."*

7.3.3 Perception of Workflow Types. All developers discussed whether the workflow types were appropriate, i.e., whether, in an ideal situation, they should, or should not occur. Their answers had a range. For instance, the Extended PR was regarded by Open3 as a good software development practice (*"small, incremental PRs . . . as opposed to . . . one giant PR, that emerged and closed an issue"*), while two other workflow types (Competing PRs and Duplicate Issue Hub) represented bad practices (*"it means multiple, multiple people tried fixing the same thing, which is not very ideal"*). As mentioned, Competing PRs do not even occur in closed-source projects. In particular, the Duplicate Issue Hub was regarded as unavoidable in open-source projects (*"they happen because anybody can open issues. And . . . you know, as much as we try to emphasize 'please search for your problem', users are always going to open issues, that's unavoidable."*).

The Divergent PR was regarded as good practice if a certain condition was met: the PR must fix a set of *related* Issues; otherwise, it is *not* a good practice. Finally, Decomposed Issue was described as *necessary* given the need to deal with time constraints.

Even though we use the terms good and bad practice, Open3 emphasized that these value judgements might be inappropriate: *"I don't think you can just blindly use the data saying, 'oh, you know, you're not following the practice one way or the other', right? because, the tool, I mean, the data doesn't really know. You know, what is the nature of the issue?"*. In other words, a tool that extracts and interprets workflow types is inherently limited because it lacks the important context in which the workflow type occurred.

7.3.4 Impact of Bad Practices. As important as indicating that some workflow types might represent good or bad development practices was the interviewees' insight that these bad practices have an associated *cost*. For instance, a Duplicate Issue Hub costs project maintainers time: *"if there are a lot of duplicate issues, somehow, the project maintainers sort of act on it. . . . you need to figure out how to reduce the number of duplications, because it costs everyone time, you know, the users."*[Open3].

This comment was echoed by closed-source developers. Meanwhile, Competing PRs have a cost on the developer choosing which PR to accept, which includes their review time. According to Open1, this decision is more subtle because one does not want to lose contributors: *"you don't want to hurt other people's feelings"*.

As we already discussed, close-source developers indicated that Competing PRs are unlikely to happen. When asked to explain this, Closed1 answered:

> *"we're allocating resources that could be used for another task. We're wasting time and money, literally looking from a capitalist perspective. So when we put two people on one task, it's wasteful. . . . So I think the correct word is this, it's waste"*

8 DISCUSSION

Software development is collaborative. And, as in any collaboration, different types of work are necessary. For instance, developers need to communicate with each other by asking questions to resolve misunderstandings, make decisions about what to (not) build, when and how to do the work, etc.

Our qualitative analysis of PR-Issue graphs enabled us to identify different workflow types and their associated collaborative work practices. For instance, the *Consequence* and the *Integrating PR Hub* workflows are examples of work practices required to *coordinate* developers' contributions. However, these coordination practices are different. *Consequence* reflects the idea that coordination is about the management of dependencies between tasks or artifacts [10, 20]. The *Integrating PR Hub* workflow is about the work conducted by software developers to document a set of related PRs. CSCW[9] researchers often use the concept of *articulation work* to describe a "supra-type of work" [34], which includes both the coordination practices required to mesh individual contributions (the *Integrating PR Hub* workflow) and the work practices to manage one's task impact on other people's tasks (the *Consequence* workflow). In other words, the workflow types we presented can be used to identify instances of articulation work in a pull-based development. As we will discuss in the next section, this can help explain a project's evolution as well as allow comparison between projects.

Meanwhile, the *Decomposed Issue* and the *Dependent PR* workflows reflect work practices associated with *complexity management*, i.e., the work necessary to close an Issue is divided into smaller parts to facilitate its execution, integration, assessment, etc [1, 8]. In other words, these workflow types illustrate the work practices by which software developers use modularity [27]. Programming languages and software development methodologies realize abstractions to help manage software modularity (e.g., methods, classes, scope, modules, aspects, etc). Previous research has considered how developers use these abstractions. By contrast, our work sheds light on modularity in the context of pull-based development processes. It is interesting to notice that during the interview, Closed3 spontaneously described a situation in which his team adopted a *Decomposed Issue* to handle the complexity and time constraints of their work.

The *Divergent PR* workflow is associated with some kind of *optimization* of the work since it is a bundling of work into a single PR to resolve multiple Issues. As we noted in Section 6.7 and later confirmed in Section 7.3.3, this practice can leverage either positive or negative results depending on whether the different closed Issues are related, i.e., whether the Divergent PR adheres to the single responsibility principle [21].

In the *Extended PR* workflow there are two different PRs with the second extending the first one. This illustrates two different work practices, depending on whether the extension is beneficial or detrimental. If the second PR improves the first, then this workflow reveals the work of *reuse*, another important software engineering principle [27]. If the second PR changes the work that

[9]Computer-Supported Collaborative Work.

should have been done correctly in the first PR, then the *Extended PR* reveals a form of *waste* in software development [14, 19, 31]. This was confirmed by Closed1 (see Section 7.3.1) when they said that a developer implementing several instances of Extended PRs might need additional guidance.

Finally, our interviewees suggested additional examples of waste including the *Competing PRs* and the *Duplicate Issue Hub* workflow types. In the last quote of the previous section, Closed1 spontaneously used the work waste. According to Sedano et al. [31], *SE waste* is an activity that consumes resources without creating customer value. Adopting their classification, we see *Competing PRs* and the *Duplicate Issue Hub* workflows as *rework*, while the "negative" *Extended PR* as an instance of *ineffective communication*. The graph-based perspective we propose in this paper is an initial step towards automatically identifying waste in software development, i.e., while it is possible to identify rework (duplicate Issues), it is not as easy to identify ineffective communication [17]. This identification would help improve software development processes, as suggested by lean approaches and corroborated by our WorkflowsExplorer tool and interviews.

In summary, analyzing *links* between PRs and Issues (as done in previous studies) can reveal developers' practices associated with work that is expected to occur (via the *fixes* relationship), reuse (via *improves*, *uses*, and *is similar*), waste (via *is a duplicate of*), and coordination work (via *blocks* and *impacts*). In contrast, our proposal to analyze PR-Issue *graphs* can reveal *additional* collaborative work practices among software developers, including practices of *articulation*, *complexity management*, *optimization*, *reuse* and *waste*. More importantly, the first three sets of practices have not been previously observed in pull-based software development.

9 IMPLICATIONS

Our graph analysis perspective reveals work practices unavailable via link analysis. A topological explanation of this claim is that *link* analysis takes into account two nodes and one edge, while *graph* analysis spans multiple nodes and edges. When making suggestions to improve code review tools, Hirao et al. [16] listed several factors to improve the identification of duplicate linking relationships. Furthermore, they showed that the performance of code review tools could be improved by using the factors they identified. Similarly, we argue that the topology of the PR-Issue graph can also be used in this scenario: for instance, an Issue A with an edge to another Issue H that is part of the hub in the *Duplicate Issue Hub* workflow has a higher chance to be a duplicate when compared to another Issue B with a link to a third isolated Issue. In another example, when facing Competing PRs, a developer can prioritize reviewing a PR that is also linked to other issues, i.e., a PR might belong to a Competing PR at the same time that it is part of a Divergent PR topology[10]. Hirao et al. [16] algo suggests that duplicate Issue detection is important to avoid waste [31] and they used links to automate this identification. Our results suggest that PR-Isse graph *topologies* can also be used to explore duplicate detection and enhance review fairness and efficiency. These examples illustrate how PR-Issue graph topologies can further improve code review tools.

A similar argument can be made about the automatic identification of Good First Issues (GFIs), i.e., issues that are adequate for newcomers interested in joining an open-source project [36]. In this case, it might not be appropriate to indicate an Issue that is the hub of a *Duplicate Issue hub* as a GFI because these hubs "are more likely to have critical connections with other issues (e.g., blocks, depends upon, incorporates)" [26], i.e., they are less suited for newcomers. In this case, Xiao et al. [36] proposed an approach to automatically indicate possible GFIs by taking into account the Issue content (title, description, labels), background (project, developers), and dynamics (change in Issue states). We believe that extending Xiao's approach with topological information would yield better results.

[10]We thank one of our interviewees for this insight.

Finally, machine-learning classifiers have been built to automatically identify types of linking relationships between two nodes [16]. We believe future work should focus on creating *topology* classifiers that take into account the entire graph as well as authorship and temporal information to accurately identify and classify workflow types. These classifiers will enable additional applications of our approach. For instance, by allowing one to compare different open-source projects regarding their work practices to determine which projects better leverage reuse, generate less waste, or require less articulation work. Such a comparative approach would be similar to the one proposed by Zoller's and colleagues [37] who used PR submissions and acceptances to create a topology of open-source projects. This topology allowed them to compare projects along different dimensions, including collective identity, hierarchy, and popularity. In our case, topology information could be used to better understand, and potentially compare, the collaborative work practices in different open-source projects.

10 LIMITATIONS

An important limitation of our work is the lack of validation of all eight workflow types with practitioners. As we discussed, we presented four of eight workflow types to developers. Further work should validate the remaining four types.

To avoid researcher bias, our qualitative analysis included the constant comparison method and an approach similar to theoretical sampling [34]. This way, the lead investigator contrasted the results with new carefully chosen data, and presented intermediate results regularly for feedback to the rest of the research team to reach a negotiated agreement [12] (see Section 3). We believe this process helped us to minimize researcher bias.

Another limitation of our work is that we do not account for branches and commits because pull requests are "becoming the atomic unit of software change" [15]. This might lead to misidentifying workflow types. For instance, in the *apache dubbo* project, we identified a potential *Decomposition* involving Issue 1641, and PRs 2621 and 2631. However, on inspection, we observed that these PRs are identical except for their branches: while PR 2621 was committed into branch *apache:2.6.x*, PR 2631 was committed into branch *apache:master*. In short, this set of nodes does *not* represent decomposition. Since we do not analyze commits, our work practices do not distinguish between contributors [14] and integrators [15]. One should notice that we do *not* claim that our topological descriptions are completely accurate; they were used to facilitate our qualitative investigation of the workflow types. Future work could use machine-learning classifiers to infer the best topology descriptions.

During our analysis of the workflow types, we decided to focus on *merged* PRs and *closed* Issues because they represent units of work that have been finalized. We made this decision to avoid analyzing work that has not been concluded, because, new links could still be added to the graphs. This means that ongoing work (open PRs and Issues) is not captured by our workflow types.

11 CONCLUSION

We first conducted a qualitative study of the graphs created by PRs, Issues, and the links between them. We analyzed a sample of 56 GitHub projects and found that the links were used by developers to express nine different types of relationships. We then used these graphs to identify eight different *workflow types*, which capture work practices that have not been previously observed in pull-based software development.

Then, we built a tool called WorkflowsExplorer to visualize PR-Issue graphs. We carried out an interview study with six open-source and closed-source developers who used WorkflowsExplorer. They noted that the tool helps to explain and improve work practices. They also envisioned scenarios in which surfacing workflow types could be used to improve their software development processes.

The *topology perspective* of our work reveals new insights into how developers use Issues, PRs, and the links between them to organize their work. This means that the workflow types we uncovered can help with understanding collaborative work practices surrounding interconnected PRs and Issues.

12 DATA AVAILABILITY

The raw data is available on OSF. The source code and queries used to identify workflow types are available here. The WorkflowsExplorer tool is hosted here.

REFERENCES

[1] Timothy Andrew. 2021. A Better Model for Stacked (GitHub) Pull Requests. https://timothya.com/blog/git-stack/

[2] Alberto Bacchelli and Christian Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, 712–721.

[3] Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512. https://doi.org/10.1126/science.286.5439.509 arXiv:https://www.science.org/doi/pdf/10.1126/science.286.5439.509

[4] Vincent Boisselle and Bram Adams. 2015. The impact of cross-distribution bug duplicates, empirical study on Debian and Ubuntu. In *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 131–140. https://doi.org/10.1109/SCAM.2015.7335409

[5] Kathy Charmaz. 2014. *Constructing grounded theory*. sage.

[6] Ashish Chopra, Morgan Mo, Samuel Dodson, Ivan Beschastnikh, Sidney S. Fels, and Dongwook Yoon. 2021. "@alex, This Fixes #9": Analysis of Referencing Patterns in Pull Request Discussions. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 385 (oct 2021), 25 pages. https://doi.org/10.1145/3479529

[7] Victoria Clarke and Virginia Braun. 2013. *Successful Qualitative Research: A Practical Guide for Beginners*. Sage, London.

[8] Benjamin Congdon. 2022. In praise of stacked PRS. https://benjamincongdon.me/blog/2022/07/17/In-Praise-of-Stacked-PRs/

[9] John Creswell. 2009. *Research Design: Qualitative, Quantitative, and Mixed-Method Approaches.*

[10] Bill Curtis, Herb Krasner, and Neil Iscoe. 1988. A Field Study of the Software Design Process for Large Systems. *Commun. ACM* 31, 11 (nov 1988), 1268–1287. https://doi.org/10.1145/50087.50089

[11] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2018. Communicative Intention in Code Review Questions. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 519–523. https://doi.org/10.1109/ICSME.2018.00061

[12] D.R. Garrison, M. Cleveland-Innes, Marguerite Koole, and James Kappelman. 2006. Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. *The Internet and Higher Education* 9, 1 (2006), 1–8. https://doi.org/10.1016/j.iheduc.2005.11.001

[13] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-Based Software Development Model. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 345–355. https://doi.org/10.1145/2568225.2568260

[14] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 285–296. https://doi.org/10.1145/2884781.2884826

[15] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie van Deursen. 2015. Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 358–368. https://doi.org/10.1109/ICSE.2015.55

[16] Toshiki Hirao, Shane McIntosh, Akinori Ihara, and Kenichi Matsumoto. 2019. The Review Linkage Graph for Code Review Analytics: A Recovery Approach and Empirical Study. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 578–589. https://doi.org/10.1145/3338906.3338949

[17] Mikko Korkala and Frank Maurer. 2014. Waste identification as the means for improving communication in globally distributed agile software development. *Journal of Systems and Software* 95 (2014), 122–140. https://doi.org/10.1016/j.jss.2014.03.080

[18] Lisha Li, Zhilei Ren, Xiaochen Li, Weiqin Zou, and He Jiang. 2018. How Are Issue Units Linked? Empirical Study on the Linking Behavior in GitHub. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. 386–395. https://doi.org/10.1109/APSEC.2018.00053

[19] Zhi-Xing Li, Yue Yu, Tao Wang, Gang Yin, Xin-Jun Mao, and Huai-Min Wang. 2021. Detecting Duplicate Contributions in Pull-Based Model Combining Textual and Change Similarities. *Journal of Computer Science and Technology* 36, 1 (01 Jan 2021), 191–206. https://doi.org/10.1007/s11390-020-9935-1

[20] Thomas W. Malone and Kevin Crowston. 1994. The Interdisciplinary Study of Coordination. *ACM Comput. Surv.* 26, 1 (mar 1994), 87–119. https://doi.org/10.1145/174666.174668

[21] Robert C. Martin. 2003. *Agile software development: principles, patterns, and practices.* Prentice Hall PTR. http://dl.acm.org/citation.cfm?id=515230

[22] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian KaUstner. 2022. "Did You Miss My Comment or What?": Understanding Toxicity in Open Source Discussions. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 710–722. https://doi.org/10.1145/3510003.3510111

[23] Reza Nadri, Gema Rodriguez-Perez, and Meiyappan Nagappan. 2021. Insights Into Nonmerged Pull Requests in GitHub: Is There Evidence of Bias Based on Perceptible Race? *IEEE Software* 38, 2 (2021), 51–57. https://doi.org/10.1109/MS.2020.3036758

[24] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2013. Diversity in Software Engineering Research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (Saint Petersburg, Russia) *(ESEC/FSE 2013)*. Association for Computing Machinery, New York, NY, USA, 466–476. https://doi.org/10.1145/2491411.2491415

[25] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye. 2002. Evolution Patterns of Open-Source Software Systems and Communities. In *Proceedings of the International Workshop on Principles of Software Evolution* (Orlando, Florida) *(IWPSE '02)*. Association for Computing Machinery, New York, NY, USA, 76–85. https://doi.org/10.1145/512035.512055

[26] Alexander Nicholson, Deeksha M. Arya, and Jin L. C. Guo. 2020. Traceability Network Analysis: A Case Study of Links in Issue Tracking Systems. In *7th IEEE International Workshop on Artificial Intelligence for Requirements Engineering, AIRE@RE 2020, Zurich, Switzerland, September 1, 2020*. IEEE, 39–47. https://doi.org/10.1109/AIRE51212.2020.00013

[27] R.S. Pressman and D. Bruce R. Maxim. 2014. *Software Engineering: A Practitioner's Approach.* McGraw-Hill Education. https://books.google.ca/books?id=i8NmnAEACAAJ

[28] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and André van der Hoek. 2018. Relationship between Geographical Location and Evaluation of Developer Contributions in Github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Oulu, Finland) *(ESEM '18)*. Association for Computing Machinery, New York, NY, USA, Article 22, 8 pages. https://doi.org/10.1145/3239235.3240504

[29] Michael Rath, Jacob Rendall, Jin L. C. Guo, Jane Cleland-Huang, and Patrick Mäder. 2018. Traceability in the Wild: Automatically Augmenting Incomplete Trace Links. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 834–845. https://doi.org/10.1145/3180155.3180207

[30] Eric S. Raymond and Tim O'Reilly. 1999. *The Cathedral and the Bazaar* (1st ed.). O'Reilly Associates, Inc., USA.

[31] Todd Sedano, Paul Ralph, and Cécile Péraire. 2017. Software Development Waste. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 130–140. https://doi.org/10.1109/ICSE.2017.20

[32] Robert Sedgewick and Kevin Wayne. 2011. *Algorithms, 4th Edition.* Addison-Wesley. I–XII, 1–955 pages.

[33] Igor Steinmacher, Gustavo Pinto, Igor Scaliante Wiese, and Marco A. Gerosa. 2018. Almost There: A Study on Quasi-Contributors in Open Source Software Projects. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 256–266. https://doi.org/10.1145/3180155.3180208

[34] Anselm Strauss. 1985. WORK AND THE DIVISION OF LABOR. *The Sociological Quarterly* 26, 1 (1985), 1–19. https://doi.org/10.1111/j.1533-8525.1985.tb00212.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1533-8525.1985.tb00212.x

[35] Dong Wang, Tao Xiao, Patanamon Thongtanunam, Raula Gaikovina Kula, and Kenichi Matsumoto. 2021. Understanding shared links and their intentions to meet information needs in modern code review:. *Empirical Software Engineering* 26, 5 (08 Jul 2021), 96. https://doi.org/10.1007/s10664-021-09997-x

[36] Wenxin Xiao, Hao He, Weiwei Xu, Xin Tan, Jinhao Dong, and Minghui Zhou. 2022. Recommending Good First Issues in GitHub OSS Projects. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1830–1842. https://doi.org/10.1145/3510003.3510196

[37] Nikolas Zöller, Jonathan H. Morgan, and Tobias Schröder. 2020. A topology of groups: What GitHub can tell us about online collaboration. *Technological Forecasting and Social Change* 161 (2020), 120291. https://doi.org/10.1016/j.techfore.2020.120291

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009