CPSC 448 Report: A Quantitative Approach to Characterizing Issue-PR Reference Graphs

Emilie Ma
University of British Columbia
Vancouver, Canada
kewbish@student.ubc.ca

1 Introduction

Open-source software development is inherently collaborative, requiring coordinated work to develop a product in public. GitHub is a popular platform for this organization. On GitHub, development is typically broken up into "issues" and "pull requests" (PRs). Issues represent future work to be addressed, like bug reports or feature ideas, and PRs contain submitted code and code reviews. Users can link between issues and PRs to express context: for example, two distinct issues reporting the same bug may be linked and marked as "duplicate". Issues, PRs, and the links between them thus capture development practices in graph topologies.

Previous work on collaborative development on GitHub has studied links from issues to issues [13], from PRs to PRs [9], and to some extent between issues and PRs [1]. However, analyzing whole graph topologies of issues and PRs grants greater context than individual link analysis alone: it allows us to identify the causes and collaboration behind emergent workflow types.

We discussed several of these workflow types in the paper "Revealing Work Practices in Pull-Based Software Development through Issue-PR Graph Topologies", submitted to ICSE 2024 [6]. Through classifying workflow types and link relationships and examining empirical occurrences, we drew insights from the types of collaborative practices that brought forth certain topologies.

The paper's qualitative focus was enabled and accelerated by quantitative analysis of the graphs. Quantitative tools, like graph database queries and an image generation module, allowed us to quickly sample instances of workflow types and visualize matches in context. These tools supported the manual coding process and facilitated an iterative approach to workflow type definitions. In this report, I focus on the quantitative results of our work, their implementation, and their optimizations.

We aimed to understand how repositories apply workflow types: what factors affect their adoption, and how varied are workflow patterns? Specifically, I analyzed the following:

- **RQ1:** How can workflow type matches be identified and characterized in large issue-PR graphs?
- **RQ2:** How representative are our topologies of the entire dataset? Are occurrence frequencies the same across projects?
- **RQ3:** How frequently do projects adopt workflow types when an opportunity arises?
- **RQ4:** How do characteristics of connected components change as their size increases? What characteristics differentiate small (<10 nodes) and large (>100 nodes) connected components?
- **RQ5:** Is there a link between the most common component isomorphisms and workflow types?

Overall, we found that the frequency and structures of matches varied widely across workflow types, with some types appearing to be more naturally adopted in development work. We noted that most repositories contained a workflow type match, illustrating fair coverage of our workflow types. A positive correlation between repository size and the number of matches demonstrated that more mature projects tend to have a higher adoption of workflow types. Projects also do not adopt workflow types frequently when given an opportunity, indicating workflow types are intentionally applied. Small and large connected components were found to significantly differ in a number of characteristics, like issue/PR ratio and component duration, which implies they make up separate categories of work within a project. As well, we observed a mapping between some common connected component isomorphisms and our workflow type definitions, supporting that our definitions reflect common structures of work.

The main contributions of this work are reusable queries for identifying workflow types, a flexible image generation module for visualizing query results, and a library of statistics scripts for characterizing issue-PR graphs. These serve to automate workflow type detection for further qualitative software development research and open-source governance.

2 Related Work

Previous work has concentrated on links between two nodes[1] (issue-issue, issue-PR, or PR-PR), particularly their prevalence and classification. Existing studies have shown that linking behaviour is sparse on GitHub, even with built-in support for the explicit "fixes" and "duplicate" link types. L. Li et al.

[1] A *node* is an issue or PR.

notably found that only 27.48% of nodes were linked in their sample of over 16k GitHub repositories [13]. Chopra et al. also found that while built-in links were used frequently for discussing GitHub-related activity, only 15.8% of all sampled semantic links were marked explicitly [4].

Because of the low usage of explicit link types on GitHub, methods to categorize implicit links have also been analyzed. Using natural-language processing (NLP) tools, Z. Li et al. detected textually similar PRs and issues as duplicates (as cited in L. Li et al. [13]). Rath et al. also examined several NLP approaches to recover links from specific commits to issues [15]. L. Li et al. explored issue-PR links by applying keyword matching in the bodies of PRs and comments to classify links automatically [13].

To facilitate discussion of link type semantics, several taxonomies of link types and their implications for collaborative development have been proposed. L. Li et al. classified their links into 6 link types, including "dependency", "duplicate", "relevance", "referenced", "fix", and "enhancement" relationships [13]. In addition, Wang et al. [18], Hirao et al. [9], and Nicholson et al. [14] have independently coded their two-node links into similar categories as Li et al. The nine-category link classification system developed in de Souza et al. is also comparable [6].

As well, some work has been done on analyzing issues or PRs from a graph perspective. Hirao et al. examined linked code reviews on PRs via a reconstruction of the PR review graph [9]. They provided prototypical examples of common link intents and explain each link type's semantics. Nicholson et al. studied the traceability of software development via links to issues, identifying properties of key "hub" issues to which many development artifacts are semantically connected [14].

These findings highlight a well-explored precedent in link classification types, which our work can build on by establishing topological pattern types. Prior work in automated link detection shows that the semantics of node connections are of interest and can be practically applied to recover software development patterns. Our topological perspective on graphs containing both issues and PRs is also novel, as no existing work examines the higher-level semantics of multiple connections in topologies.

3 Methods

The methods for the full project, including the methods for qualitative analysis, are available in de Souza et al. [6]. Here, I will focus on the tool-specific techniques used to develop the quantitative findings.

The 56 GitHub repositories sampled for this project had previously been scraped [19]. They were saved as JSON output and raw Python pickle files. Each repository was associated with one "graph" JSON file and one "structure" JSON file. The "graph" JSON files contained metadata for every node in a project, as well as linking them to other nodes in its connected component. The "structure" files grouped together connected components by size [20]. Each repository had an associated collection of raw pickle files, which were parsed when specific link metadata, such as type or creator, was needed.

Statistics were primarily computed via loading NetworkX [7] object representations of each project, retrieving all connected components, and iterating through individual node metadata. We used Python's `multiprocessing` library with `cpu_count() / 2` cores to parallelize individual graph creation for all statistics scripts.

Where possible, we directly read metadata from the structure and graph JSON files, as loading and deserializing JSON is much less expensive than building up the equivalent NetworkX representation. For example, statistics for the number of nodes in 1- and 2-node topologies read from fields in the structure JSON files. However, not all data was available in these JSON files. In particular, comment and link metadata was only available from the raw pickle files.

3.1 RQ1: Characterizing Workflow Types

Sections 3.1.1 and 3.1.2 detail how the issue-PR graphs were imported into the graph analysis tools used, NetworkX and Neo4j. Section 3.1.3 covers the particulars of building the topological queries: Section 3.1.3.1 explains a typical query structure, and Sections 3.1.3.2 to 3.1.3.7 contain various implementation details.

3.1.1 Importing to NetworkX

NetworkX is a popular Python package for network analysis, used to compute many of the statistics for this report. The previously implemented `NetworkVisCreator` and `BarChartCreator` classes help transform the raw scraped data into directed NetworkX graphs by extracting explicit GitHub links from comment text and computing connected components [20]. In these graphs, each node retains the following metadata:

- a node ID string, used to uniquely identify the node in NetworkX. The ID has the format `<owner/repo#node_identifier>`. The `node_identifier` is the same number used to reference the node within the repository on GitHub (i.e. issue #31 had `node_identifier` 31.
- the node's type, which was one of `pull_request` and `issue`
- the repository the node belongs to
- the node's identifier from GitHub
- the node's creation and update times

Links contained just their link type, which was one of "fixes", "duplicate", or "other" to represent a relationship not supported by GitHub's built-in explicit links.

3.1.2 Importing to Neo4j for Topological Querying

Implementing query matching with NetworkX alone would be difficult, as it only implements matching for isomorphic subgraphs. To be able to query for larger topological patterns within the graph, we imported our repository graphs into Neo4j, a graph database management software [11]. Its query language is called Cypher: it supports queries for retrieving specific links, multi-node topologies, and connected components [10].

To import to Neo4j, we re-parsed the raw Python pickle files to add additional metadata to the NetworkX graph objects. We then dumped each repository into the GraphML file format using NetworkX's `write_graphml()` function, and imported the files into Neo4j via Cypher's `apoc.import.graphml()` procedure.

Cypher supports a construct called node "labels", which allows nodes and links[2] to be grouped into larger categories. We use `issue` or `pull_request` for node labels, and `fixes`, `duplicate`, or `other` for link labels.

Note that by default, NetworkX's `write_graphml()` will not correctly import node labels and other node attributes: `named_key_ids=True` must be passed. As well, `apoc.import.graphml()` must be called with `{readLabels: true}` in order to correctly read the node labels.

Beyond the metadata discussed above, the projects were imported into Neo4j as directed graphs with the following additional metadata:

- the node status: one of "merged", "closed", or "open"
- the node's close time, if it was closed
- the GitHub username and URL of the user that created the node (or for edges, the comment linking the two nodes)
- the URL to node's or link's GitHub page
- the node's or edge's label

3.1.3 Matching Workflow Types with Topological Queries

In de Souza et al., we identified a number of workflow types, which were each represented by a topology [6]. These topologies specified their constituent nodes, their expected metadata, and the relationships and restrictions between nodes. See Figure 1 for examples of the identified topologies.

These topologies were refined through an iterative process of finding examples within a sample of connected components, creating a query to match the characteristics of an observed workflow type, and finding counterexamples or additional constraints within the returned matches. These topologies served to support and validate the qualitative analysis of the paper with empirically observed occurrences.

The full topology queries can be found in the project repository here.

3.1.3.1 An Example Cypher Workflow Type Query

Each workflow type's topology query began with a subquery call to match the topology and return IDs of any node in a topology match. This was used to compute the match opportunity statistics in Section 3.3. This call is identical to the one used in the main body of the function, but must be performed beforehand to correctly aggregate all IDs. We do not compute the whole connected component in this section of the call to reduce repeated work.

[2]Neo4j uses slightly different language for edges or links, calling them relationships. All three terms are used interchangeably in this paper.

The second part of the query, its main body, computes the match as in the first subquery call, but also computes the connected component the match was found in and some other statistics.

The topology match itself is typically structured with an initial `match` line to specify the nodes, their statuses, their types, their relationships, and their relationships types. Listing 1 shows an example of this initial match line, querying for a closed issue connected to two PRs of any status via a "fixes" link.

```
match (i:issue {status: "closed"})-[r {labels
    : "fixes"}]-(pr:pull_request), (i)-[r2 {
    labels: "fixes"}]-(pr_2:pull_request)
```

Listing 1. Initial topology match from Decomposed Issue query.

This is followed by one or more `where` clauses specifying other metadata or constraints on the topology match nodes. Listing 2 shows one such clause from the Consequent Issue query. It requires that the second issue must be created after the PR, and that the two issues are distinct[3].

```
where i2.creation_date > p.creation_date and
    i.number <> i2.number
```

Listing 2. Topology filtering from Consequent Issue query.

In some cases, additional `with ... collect` clauses are needed to aggregate all nodes with a relationship of an arbitrary multiplicity. These are often used to apply authorship or temporal constraints, or to specify constraints on relationships with arbitrary multiplicities. These aggregation relationships are also used to collect all the edges matched in the topology for later highlighting.

```
with i, collect(distinct pr) as pull_requests
    , collect (distinct pr.user) as users,
    collect(r) as match_relationships, max(pr
    .creation_date) as max_date, min(pr.
    creation_date) as min_date [...]
where size([p_r in pull_requests where p_r.
    status="merged"]) = 1 and size([p_r in
    pull_requests where p_r.status="closed"])
     >= 1 and size(pull_requests) >= 2 and
    size(users) > 1 and max_date - min_date
    <= 604800
```

Listing 3. Additional aggregation clause from Competing PRs query.

Listing 3 is an excerpt from the Competing PRs query that shows several uses of this aggregation and filtering. The `collect(distinct pr) as pull_requests` in the first line is filtered in the second to check that only one of the PRs attached to the issue is merged (`size([p_r in pull_requests where p_r.status="merged"]) = 1`). We similarly check that there are at least 2 total PRs attached to the node (`size(pull_requests) >= 2` created by more

[3]Cypher uses <> to express inequality.

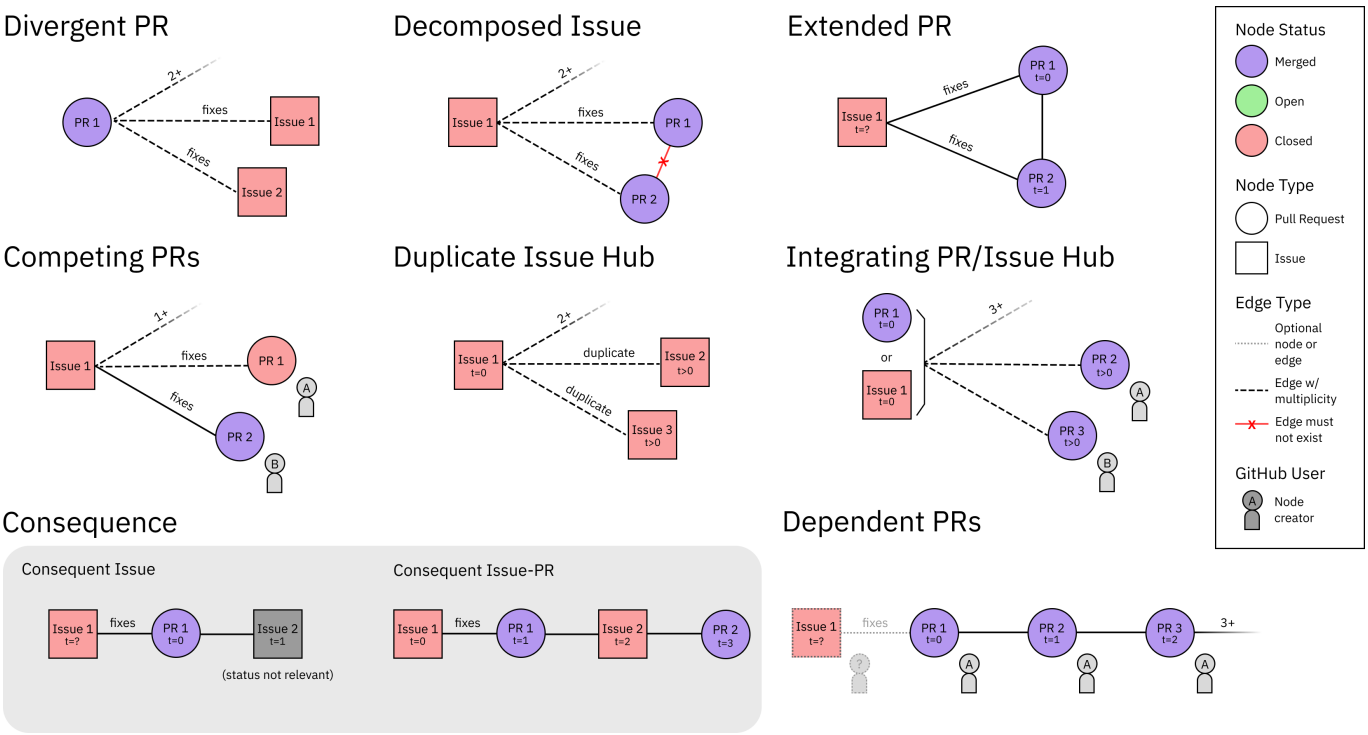


Figure 1. Topologies of the workflow types.

than 1 user (`size(users) > 1`). We use the aggregating functions `min` and `max` to get the maximum and minimum creation dates in any of the matched PRs and can use those results to check that their delta is within a specified threshold (`max_date - min_date <= 604800`).

After the match has executed, we then want to collect the connected component that the match took place in, and return the results. We collect the connected component to provide additional context for visualizing and examining the topology match result. The final query results include the nodes and edges in the match, along with all their available metadata, and the nodes and edges in the connected component the match was found in. Listing 4 shows a basic example of how this is done for the Extended PR query.

```
call apoc.path.subgraphAll(i, {limit: 50, bfs
    :true})
yield nodes, relationships [...]
return i, pr1, pr2, nodes, relationships,
    match_relationships
```

Listing 4. Collecting and returning the connected component for the Extended PR query.

Note that Neo4j will return matches separately for multiple topologies within the same connected component. For example, if a query is for an issue linked to a PR, and there is one issue linking to many PRs, each issue and PR match is returned separately. Our scripts parse each match result separately.

Neo4j will also match greedily. If a query is for an issue linked to any number of PRs and there is an issue linking to three PRs, for example, it will return a single match including one issue and all three PRs. This simplifies the matching process as matches would otherwise have to be unioned manually, saving time during later analysis and qualitative coding.

Neo4j returns duplicate nodes by default when running a `collect` aggregation of results, so we have ensured `collect(distinct ...)` is used instead to exclude duplicate matches.

3.1.3.2 Applying Constraints to Avoid False Positives

Besides the topological shape of the query pattern, we applied up to four main constraints to match nodes: differentiating a node's status, type, authorship, and temporal metadata. These constraints served to avoid false positives in the returned matches and tedious manual verification.

For example, the authorship constraint was developed to exclude non-collaborative work from some workflow types, which returned false positives that did not match the workflow type's semantics. For instance, the Competing PRs topology could otherwise be mistakenly matched when a person creates a PR to fix an issue, then abandons it to create another similar PR.

The temporal constraint was applied to several topologies due to similar observations of false positives when during manual examination. The temporal constraint is also used across topologies as a heuristic for "consequence", since a node X that was created after a node Y and is linked to node Y likely builds on or references node Y's work.

After manual coding of the sampled links, we noted that link direction typically did not appear to change the semantics of a node link. This also reflects the bidirectionality of GitHub links: the GitHub UI will show a mention in the linked PR as well as in the linking PR. None of our queries specified edge direction, save for the Dependent PRs workflow type. In this case, we use link direction as an additional constraint and heuristic in place of Git branch dependencies, which is how Dependent PRs are typically identified.

3.1.3.3 Weakening Constraints to Avoid False Negatives

In several topologies, we initially enforced a strict constraint across all nodes within a topology, but decided to loosen it to apply to at least half the nodes in the topology. This avoided false negatives where a sampled component was a topological candidate and a good semantic match for a workflow type, but a single node did not follow the constraint.

For example, in the Decomposed Issue query, we previously enforced that all pull requests attached to the issue needed to be merged. However, upon manual inspection, we found several counterexamples and thus believed it was appropriate to weaken the restriction to require that *at least half* of the pull requests were merged.

To achieve this dynamically, we compared the amount of merged pull requests to the number of pull requests attached to the issue. If the number was at least $\frac{1}{2}$, the match continued. This dynamic comparison requires an aggregation to be performed on all pull requests first. Listing 5 shows how this was done in Cypher.

```
where size(pull_requests) > 1 and size([p in
    pull_requests where p.status = "merged"])
     >= toFloat(size(pull_requests))/ toFloat
    (2)
```

Listing 5. Dynamic constraint checking in Decomposed Issue query.

3.1.3.4 Excluding Topologies to Avoid Double-Matching

Some topologies require other topologies to be excluded before they can be matched to avoid miscounting occurrences or false positives. For example, matches of the Consequent Issue-PR workflow type will all contain matches for Consequent Issues, as a Consequent Issue-PR is a special case of a Consequent Issue containing an extra final PR. Consequent Issue-PRs therefore need to be excluded before querying for Consequent Issues to ensure their results are mutually exclusive.

We achieved this by collecting a list of node IDs (Neo4j-assigned) for all known matches of a topology. Because of the way Neo4j's `collect` works, the query for the excluded topology must also be called in the query for the topology to be matched (e.g. the query for Consequent Issue-PR must be called in the query for Consequent Issue).

- First, run the match for the excluded topology and collect all matched node IDs (i.e. `return collect(distinct id(i))+collect(distinct id(p)) as exclude_ids`)
- Then, run the match for the topology to be matched, and assert none of its node IDs are in the list of `exclude_ids`, (i.e. `where id(i) not in exclude_ids`)

This technique is used in several queries, including for excluding Dependent PRs from Integrating PR Hubs and excluding Integrating PR Hubs from Extended PRs. Note how the pattern exclusions "chain" into queries: the query for exclusion of Dependent PRs from Integrating PR Hubs must also be executed in the Extended PRs query.

3.1.3.5 Retrieving Connected Components for Context

We often wished to analyze the resulting topology matches in context, so before returning each result, we also queried for the connected component the topology match was in. We used a built-in subgraph collection function in Cypher to retrieve the nodes in the connected component around a central node: `apoc.path.subgraphAll(central_node, limit: 50, bfs: true)`.

There were performance limitations due to some large components that did not allow us to select the entire connected component. We imposed an arbitrary limit of returning a maximum of fifty nodes closest to the central node or the number of nodes in the topology match, whichever was larger. The connected component retrieval function iterates in a breadth-first manner. It is possible that there were more nodes in the connected component outside of what was returned, but we found that for our analysis, the fifty closest nodes provided enough semantic and topological context.

3.1.3.6 Generating Dependent PR Query Variants

The Dependent PR topology query is unique in that Neo4j does not have a feature to match a path of unspecified length consisting only of nodes with certain properties. Queries for each size of Dependent PR must instead be generated and executed with a script.

The same query generation function is used to statically fetch all IDs of nodes in any Dependent PR topology. This makes the Integrating PR Hub and Extended PR queries more efficient because these topologies require excluding the Dependent PR topology before matching. The list of Dependent PR node IDs can be loaded statically in these parent queries to avoid running many iterations of the Dependent PR subquery within a single parent query match.

3.1.3.7 Managing Result Cardinality for Performance

According to Cypher documentation, Cypher operations create result streams for each clause in the query, which are then inputted row-by-row into the next clause [3]. The number of rows streamed between clauses is referred to as cardinality. The aggregation function `collect` was used frequently to reduce the cardinality of the data: `collect`ing all connected nodes of arbitrary multiplicities first avoided returning each of the node relationships as its own row, thereby decreasing cardinality.

Aggregations and filtering were also performed as early as possible in the query as recommended by Neo4j documentation. This helped to reduce cardinality early on, increasing overall efficiency.

The `profile` keyword was also used intermittently during query development to ensure a reasonable limit on the amount of reads performed.

3.2 RQ2: Determining Project Representativeness

To validate the coverage of our project sample and to investigate the properties of the entire cross-project network, we calculated the number of matches for each topology in each project and the distribution of workflow type matches across all repositories.

These scripts relied on the Cypher queries discussed in 3.1.3. After executing the Cypher query, they mapped repositories to topology match count. These scripts did not utilize `multiprocessing` because the majority of the computation was already performed in Neo4j.

3.3 RQ3: Calculating Topology Occurrence Opportunities

We computed a metric called Match Topology Repository Opportunities (MTRO) for repositories containing a workflow type match in order to determine how individual projects adopted workflow types.

MTRO describes the number of opportunities where a workflow type could have occurred in a project but did not, over the total number of nodes in the topology. This represents how likely a repository adopts a workflow given an opportunity to apply it. MTROs of 0 indicate that there were no opportunities for a match to have occurred within the project (if all its nodes were of a different node type or status, for example). Low MTROs indicate that most of the source nodes of a topology in a repository are part of matches, and high MTROs represent more unfulfilled opportunities for matches to have occurred.

Computing MTRO requires a "source" node type and status to be identified for each topology. This source node is typically the central hub node in a topology with a relationship with arbitrary multiplicity, or the first node in a topology with a temporal constraint.

After executing each topology match query, we aggregate a list of all nodes matched by the workflow type topology and in a particular repository. We then aggregate a list of all nodes that have the same type (issue or PR) and status as the "source" node of the topology: call these nodes "candidates". These candidates excluded nodes in connected components of size ≤ 2, because each of our workflow type topologies required a minimum of 3 nodes.

We additionally define "opportunities" as candidates that are not in a topology match (i.e. nodes that *could* have started a topology match but did not). MTRO is then computed by taking $\frac{\text{\# of opportunities in project}}{\text{\# of candidates in project}}$. We use the number of candidate nodes as the denominator instead of using the number of nodes in the graph to normalize the result and avoid bias with extremely large and small repositories.

To calculate this, we first executed the workflow type's Cypher query. For each returned match, we loaded the graph JSON file of the associated repository and iterated over all nodes to determine the candidates. We then iterated over all returned matches for the workflow type to build a set of matched nodes, and subtracted this from the set of candidates to obtain the final set of opportunities.

3.4 RQ4: Computing Statistics across Connected Component Size

We aimed to contrast small and large connected components to see if there were any statistically significant patterns in metadata differentiating the two. We arbitrarily defined small connected components as those with < 10 nodes, and large connected components as those with > 100 nodes. We also wished to examine trends across connected component sizes over our entire dataset.

We computed descriptive statistics for all connected components, including the distribution of issues and PRs, as well as node status.

3.4.1 Connected Component Duration

One heuristic we examined was component duration, or the time delta between the first node creation in a component and the last node update event.

We did not cut off node updates when a node was marked as closed, as in cases where a node is closed and reopened multiple times, because this was at times followed up by additional meaningful conversation. Our statistics take the final node update in the component as its last update time regardless of the update type (i.e. close, comment, mention).

We included nodes that were not updated or commented on after their creation. If these nodes were isolated (in a connected component of size 1), their component duration would be 0.

3.5 RQ5: Identifying the Most Frequent Connected Component Isomorphisms

We aimed to find the most frequent topologies of each size across all projects in order to generate potential new workflow type definitions and validate our existing definitions. To do this, we find all connected components of a given size, group them into isomorphic equivalence classes, and find the frequency of each isomorphism out of all isomorphisms of that size.

Isomorphism depends only on node type (issue v.s. pull request) and edge type (e.g. "fixes"), though the module is capable of adding constraints on identical node statuses and edge direction. These aspects are not taken into consideration because they led to many permutations of a semantically similar topology, and we wished to limit the number of isomorphisms returned for ease of manual analysis.

Computing these components can take some time due to the cost of checking for an isomorphism. To optimize, we use multiprocessing with `cpu_count() / 2` cores. As well, we avoided repeatedly calling `nx.compose` as this causes the entire graph (of all projects) to be re-copied at the end of each iteration. Instead, we used `g = nx.compose_all(graph_iterable)` at the end of isomorphism checking.

Enforcing additional constraints on node status and edge direction could have also increased the efficiency of the isomorphism checking. NetworkX uses the VF2++ algorithm under the hood to perform isomorphism verification, which supports pruning the search space significantly when node metadata doesn't match [8].

3.6 RQ1 & RQ5: Visualizing Topology Matches

We generated images of topology matches and connected components to aid in manual qualitative coding. For images depicting a workflow type topology match, we first executed a Neo4j query match and retrieved its result via the Neo4j Python driver. We converted the result into a NetworkX graph object. Images were rendered with NetworkX's built-in `draw` functions.

Images visualize node status, node type, node number, edge direction, and edge type. A square represents an issue, and a circle represents a PR. Red nodes represent closed nodes, green represents open, and purple represents merged, in accordance with the GitHub UI. Nodes are labelled with their type ("I" for issue or "PR" for pull request) and GitHub identifier. "Fixes" or "duplicate" edges are labelled; edges of type "other" are not explicitly labelled.

Image generation relies on the presence of several specially named return variables in the query. Each query must return:

- `nodes`: the nodes in the connected component, usually retrieved with `apoc.path.subgraphAll(node, {limit: 50, bfs: true})`
- `relationships`: the relationships of the connected component, also returned by `apoc.path.subgraphAll()`
- `match_relationships`: the relationships within the component matched by the actual topology; edges in `match_relationships` may exist in `relationships`

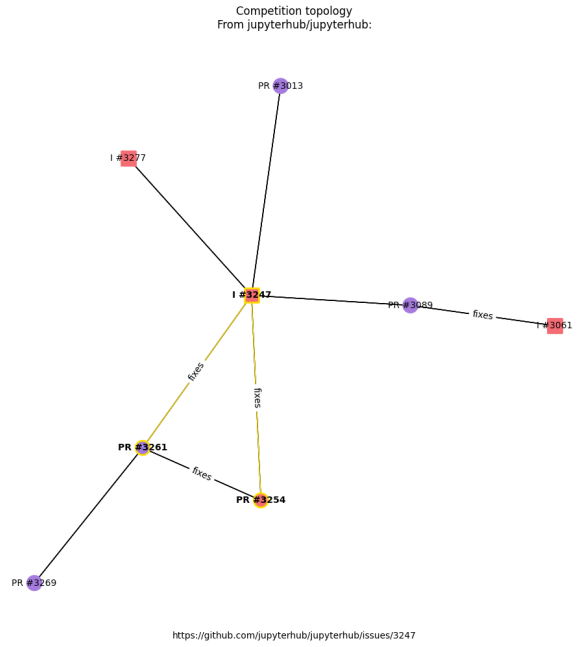
Rendering images can be a bottleneck for processing, especially if the topologies are large. To optimize, the image generation scripts utilized multiprocessing on `cpu_count() / 2` cores to speed up rendering. The image generation also uses the non-interactive `agg` Matplotlib graphics backend and the NetworkX `graphviz_layout` option for faster rendering. We found that avoiding repeated calls to `nx.draw()` also provided significant optimization. We opted instead to aggregate node and edge attributes, like colour, into lists to render with a single call.

3.6.1 Single Topology Match Images

To visualize workflow type match topologies, we executed the Neo4j query, retrieved the topology matches in context with their surrounding component, reconstructed it into a NetworkX object, and rendered the image. Figure 2 is an example of a visualization result, highlighting a match of the Competing PRs topology in yellow.

These topologies may not show the full component a topology match takes part in because the number of nodes was limited to a maximum of fifty nodes or the number of nodes in the topology match, whichever was greater. This was done to avoid rendering large components, which were often only linked to the workflow type match several degrees of separation away.

Figure 2. Topology image rendered from Neo4j with a single topology match.



A random sample of topology matches have their images generated. By default, we generated `min(# records / 2, 20)` images per topology.

3.6.2 Sequential Topology Match Images

A "sequential" topology match is defined as a topology match in which one of its nodes is also part of another match for another topology. These topologies are rendered according to the component size restriction above. In these images, any nodes matched in both topologies will be additionally labelled with a star. These stars represent the "intersection" of the two topologies.

See Figure 3 for an example result. Issue #1986 and PR #2350 are parts of both the Divergent PR (red) and Extended PR (orange) matches. Issue #1775 was only part of the Divergent PR match, as indicated by its red highlight, and PR #2265 was only part of the Extended PR match. The edges are coloured according to the topology match they are contained in.

3.6.3 Sampled Connected Component Images

Component visualizations for the 64 sampled components generated via diversity sampling were generated by reading a CSV of sampled components, reading the repository's graph JSON file, reconstructing the component into a NetworkX object, and rendering the image.

The links in these images were manually coded, so they may differ from the built-in GitHub link types and those that actually appear in GitHub UI. Several links were manually marked as "ignore" and do not appear in rendered images.

Figure 3. Topology with both Divergent PR and Extended PR topologies.

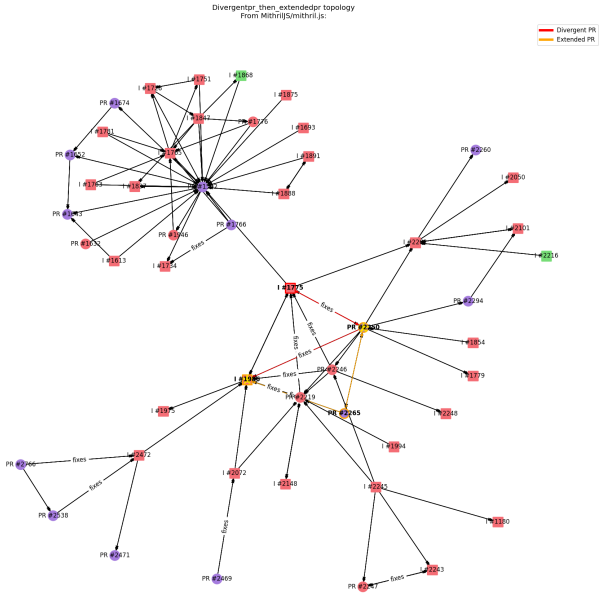
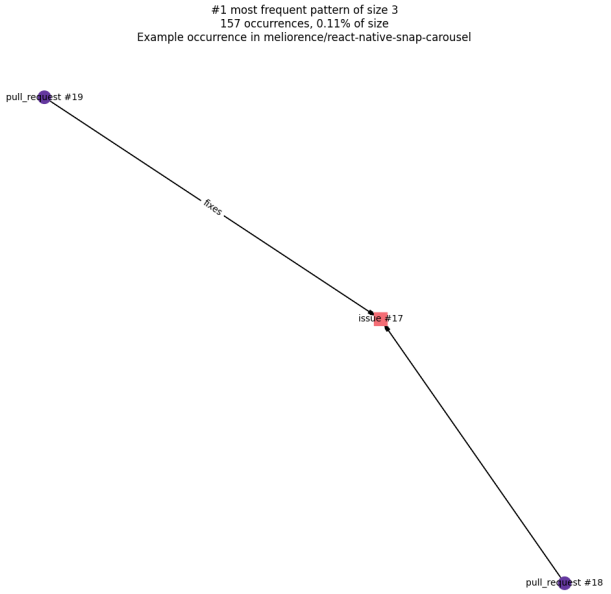


Figure 4. Most frequent connected component isomorphism with 3 nodes.



3.6.4 Most Frequent Connected Component Isomorphism Images

We also created a script to visualize the most frequently occurring isomorphisms of connected components. It reads graph data from local pickle files in `raw_data/`, finds the most frequent topology isomorphisms, and renders an example isomorphism. See Figure 4 for one such isomorphism image, depicting the most frequent isomorphism of size 3.

These images are an *example* of the most frequent connected component of that size. Other isomorphic connected components will count towards that isomorphism's frequency but not be visualized separately.

For more information on the matching of these connected components, see Section 3.5.

3.7 Fitting Power-Law and Degree Distribution

In de Souza et al., we aimed to characterize the overall issue-PR Collaboration Graph [6]. We plotted the power-law exponent of each project subgraph to verify that they possessed the characteristics of scale-free networks and the degree distribution of all projects to ensure it also followed a power-law distribution. NetworkX has built-in helpers to calculate the degree of each node, but does not provide any tools to describe these distributions.

For fitting a power-law distribution to the degree distribution, we used the `powerlaw` Python library to calculate the exponent and minimum coefficients [2]. `powerlaw` uses a method prescribed by Clauset et al., from whom we adapted our visualization script [5]. Note that the α value returned by `powerlaw` is one less than the actual exponent used to plot the degree distribution graph, as per Clauset et al.

4 Results

4.1 RQ1: Workflow Type Characterization

As discussed in de Souza et al., we identified eight workflow types, which were represented in topological queries and sampled for image generation [6]. The resulting visualizations were qualitatively examined by the first researcher to further refine the workflow type definitions and queries, as well as their semantic implications.

By using these queries, we were able to collect and examine types of work with more context than analyzing single links. While these queries sometimes fail to match semantically similar work to a workflow type, they capture work practices well when adequate temporal and structural metadata is provided.

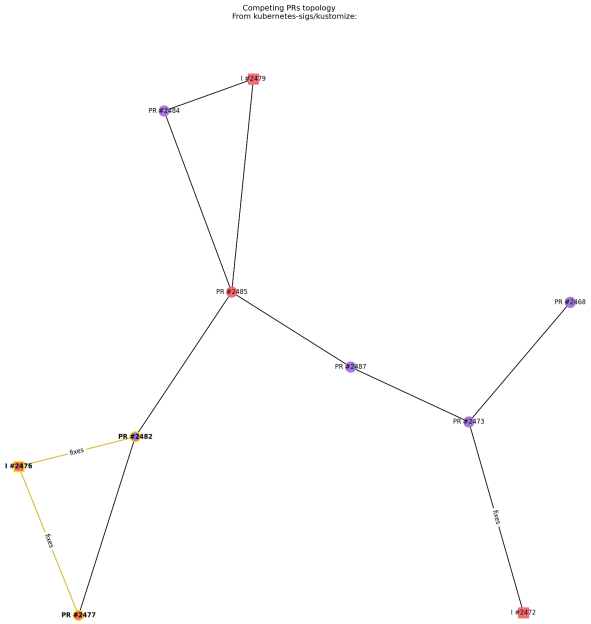
These queries also allow us to quantitatively characterize workflow type matches through workflow type match counts and individual match sizes. We noticed that some workflow types appear much more frequently than others, implying that these workflows are often naturally adopted in open-source development[4]. By examining the matches of a specific workflow type, we can also compare how each of its applications differs in size or structure. Workflow types where matches are highly variable indicate significant flexibility of the work practice.

The workflow type definitions can be found in Figure 1. Example images of each topology match are shown in Figures 5 and 6.

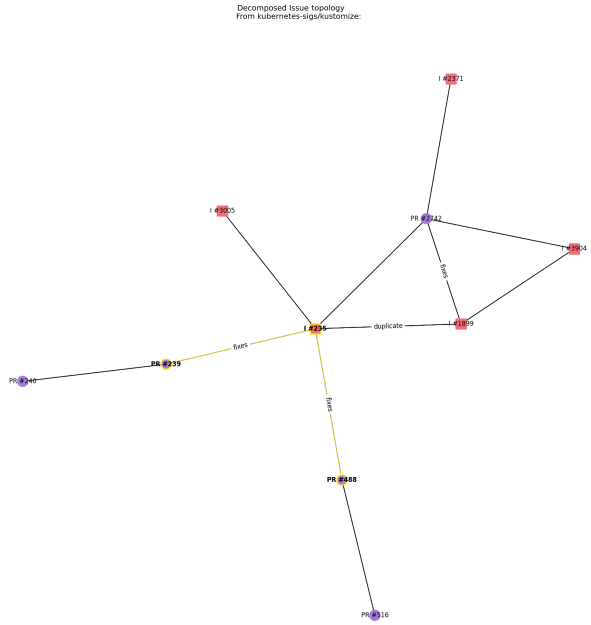
The number of matches, along with the repository with the most matches of that workflow type, is presented in Table 1.

[4]When discussing workflow type adoption, we can only *suggest* high adoption when there are many matches, as we do not know the false positive rate of the queries.

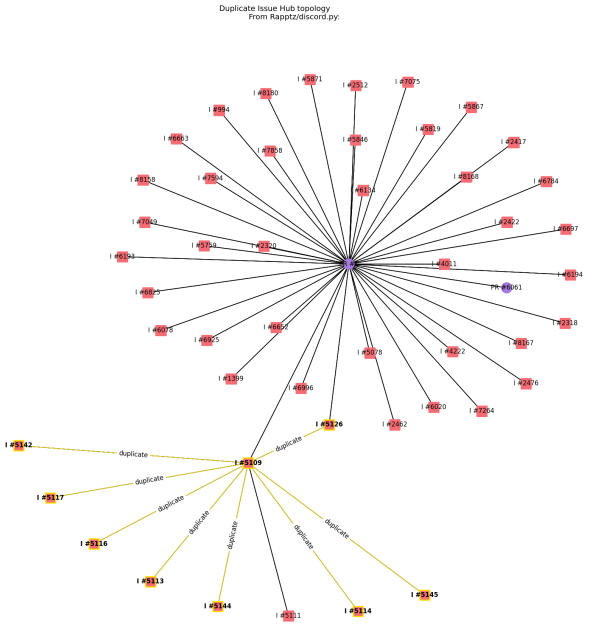
Figure 5. Topology match examples. (Pt. 1)



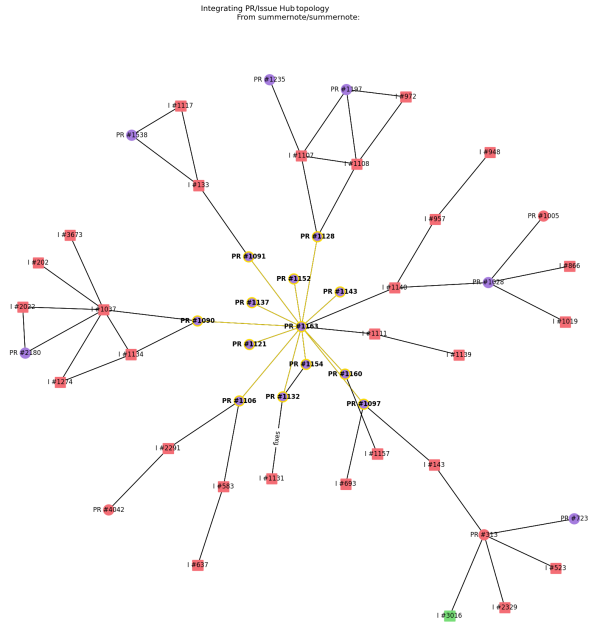
https://github.com/kubernetes-sigs/kustomize/issues/2476



https://github.com/kubernetes-sigs/kustomize/issues/235

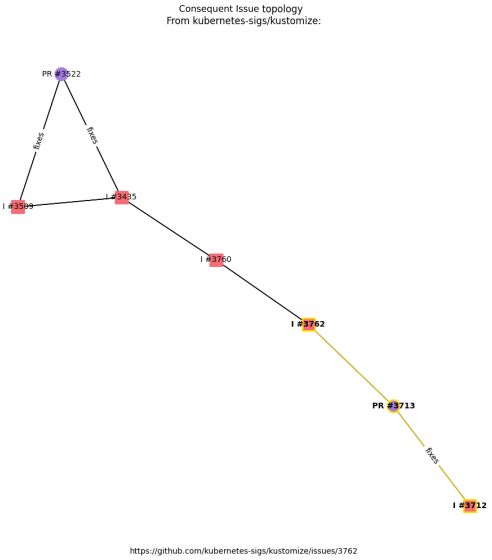


https://github.com/Rapptz/discord.py/issues/5109

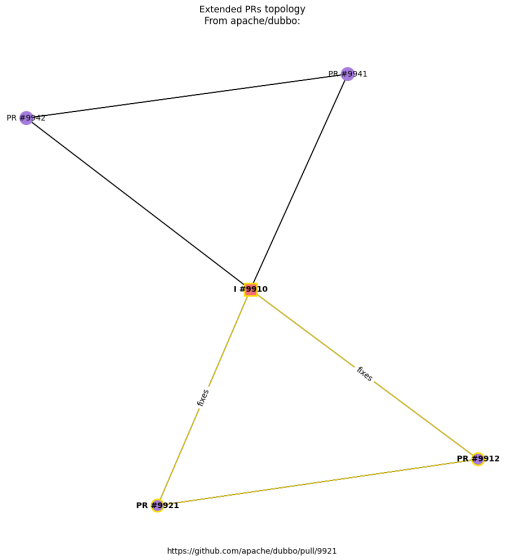


https://github.com/summernote/summernote/pull/1163

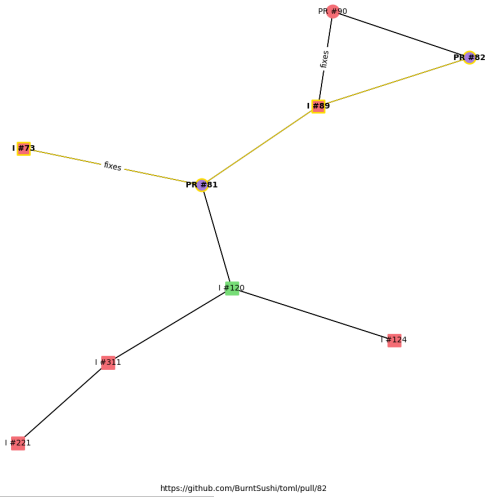
Figure 6. Topology match examples. (Pt. 2)



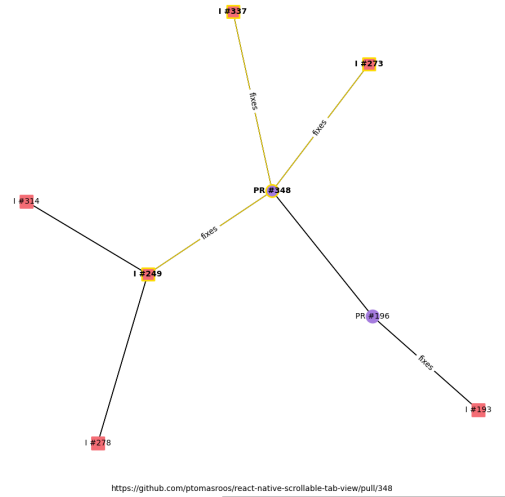
https://github.com/kubernetes-sigs/kustomize/issues/3762



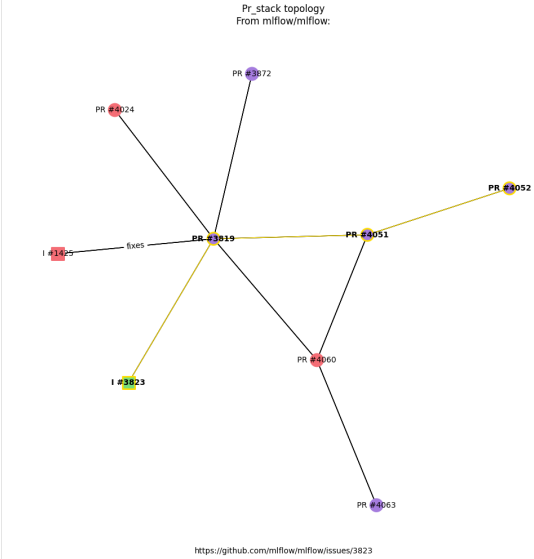
https://github.com/apache/dubbo/pull/9921



https://github.com/BurntSushi/toml/pull/82



https://github.com/ptomasroos/react-native-scrollable-tab-view/pull/348



https://github.com/mlflow/mlflow/issues/3823

Table 1. Workflow types and their number of matches.

Workflow Type	Num. Matches	Max. Matches In Single Repository (Repository Name)
Competing PRs	15	4 (mlflow/mlflow)
Consequent Issue	210	29 (TypeStrong/ts-node)
Consequent Issue-PR	471	132 (App-vNext/Polly)
Decomposed Issue	91	40 (apache/dubbo)
Dependent PRs	52	11 (mlflow/mlflow)
Divergent PR	118	13 (kubernetes-sigs/kustomize, TypeStrong/ts-node, MithrilJS/mithril.js)
Duplicate Issue Hub	45	17 (Rapptz/discord.py)
Extended PR	14	5 (apache/dubbo)
Integrating PR/Issue Hub	94	34 (apache/dubbo)

We define a workflow type as having a "hub topology" when it includes a relationship with an arbitrary multiplicity (e.g. Competing PRs, where there is an arbitrary number of PRs fixing one issue). The match size distribution of workflow types with hub topologies is shown in Table 2.

Table 2. Topology size distributions for workflow types with hub topologies.

Workflow Type	Average Size	Min. Size	Max. Size	Median Size	Size STDev
Competing PRs	3	3	3	3	0
Decomposed Issue	3.84	3	6	3	1
Dependent PRs	3	3	3	3	0
Divergent PR	3.48	3	14	3	1.36
Duplicate Issue Hub	4.07	3	20	3	2.99
Integrating PR/ Issue Hub	11.35	4	71	6	10.48

4.2 RQ2: Project Representativeness

We examined the percentage of nodes in a topology match to determine the overall coverage of our workflow type definitions. We then compared the distribution of a workflow type's matches across all repositories to reveal how well topologies represent work done in those projects.

Overall, our topologies are a good representation of the data, capturing around half of significant collaborative work across all repositories. Some repositories adopt certain topologies much more frequently than others, which may indicate a more formalized adoption of a development workflow, especially if a repository is larger and thus more mature. Our topologies fail to represent some repositories well, though this likely demonstrates a lack of repository organization or activity.

Out of the 91126 nodes present in the issue-PR graphs, 3044 nodes were in at least one topology (3.34%). There were 5776 nodes in a connected component of size ≥ 3 containing an explicitly labelled edge (i.e. "fixes" or "duplicate" relationship) or consisting of all merged PRs (to allow for Dependent PRs and Integrating PR/Issue Hubs). These nodes have an adequate size and contain the characteristics required by our workflow type definitions, so they are are the best approximation for nodes that could possibly be in a workflow type match. Out of these candidates, the number of nodes in at least one topology increases to 52.70%.

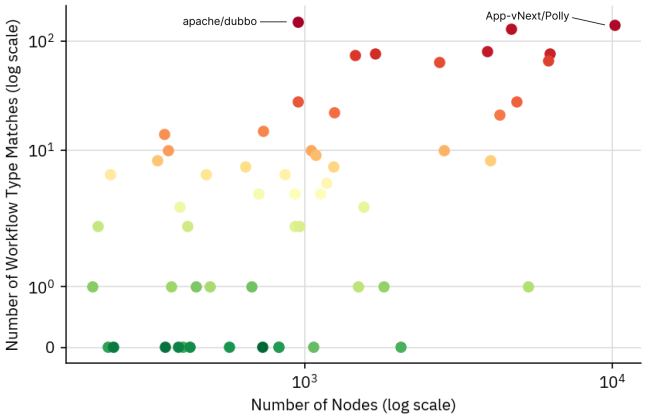
Tables 4 to 12 show the distribution of workflow type matches across all repositories. Repositories with no matches for a particular workflow type are not shown or included in the Z-score calculation.

Some repositories contained a significant percentage of the matches for a workflow type. For example, `App-vNext/Polly` contained 28.03% of all Consequent Issue-PR matches. `apache/dubbo` contained 43.96% of all Decomposed Issue matches and 36.17% of Integrating PR/Issue Hub matches.

`App-vNext/Polly` contains the largest proportion of topology matches (13.51% of all topology matches), and `apache/dubbo` makes up 12.61% of all topology matches. It would appear that large repositories have more topology matches due to having more nodes and thus more opportunities for workflow types to be utilized. The `apache/dubbo` project indeed contains a large proportion of the total nodes, with 10222 nodes (11.22%). There is also a moderate positive correlation (0.622) between repository size and number of topology matches.

However, there are several outliers with high numbers of matches and relatively fewer nodes. `App-vNext/Polly`, which contains the highest number of topology matches, only contains 949 nodes (1.04% of total nodes). Several other repositories in Figure 7 have a low node count but relatively high numbers of topology matches (i.e. the left-most orange markers).

Figure 7. Number of topology matches over repository size for all projects.



Twelve repositories did not contain matches of any topology. They can be found in Table 13.

4.3 RQ3: Topology Occurrence Opportunities

In Section 4.2, we computed the distribution of workflow type matches across repositories. In this section, we examine the distribution of matches compared to the number of opportunities in the repository for such a match to occur.

Overall, projects rarely adopt workflow types, indicating that work practices are typically a result of intentional one-off coordination. When repositories frequently adopt workflow types when opportunities arise, they have likely adopted the workflow type, either formally or informally. This appears to be the case especially when the repository is small.

We use the MTRO proportion for determining how frequently a project adopts a workflow type given an opportunity to do so. This heuristic allows us to identify repositories that had an unusually high or low number of "missed" opportunities to apply a workflow type.

Tables 14 to 22 show the MTROs of each workflow type, computed by repository. These tables do not include repositories with no matches, because their MTRO would be 100% (any workflow type candidates would not be in a topology match). Table 3 summarizes the mean and STDev of the MTRO of each workflow type.

Table 3. Workflow type MTRO mean and standard deviation.

Workflow Type	Mean MTRO	MTRO STDev
Competing PRs	98.17%	1.65%
Consequent Issue	79.09%	17.05%
Consequent Issue-PR	80.85%	16.19%
Decomposed Issue	86.73%	12.93%
Dependent PRs	95.43%	5.99%
Divergent PR	82.67%	13.44%
Duplicate Issue Hub	85.19%	16.78%
Extended PR	91.21%	14.07%
Integrating PR/Issue Hub	82.74%	10.04%

Several workflow types have repositories with MTROs <50%. Most opportunities (i.e. most of the nodes of that source node type) are therefore part of workflow types.

The topologies with more total matches (e.g. Consequent Issue) and with larger average sizes (e.g. Integrating PR/Issue Hub) had relatively lower MTROs.

Some repositories are among the extremes of multiple workflow type MTROs. For example, `deployphp/deployer` has the highest MTRO per repository for 5 workflow types: Consequent Issue, Consequent Issue-PR, Decomposed Issue, Dependent PRs, and Divergent PR. `jupyterhub/jupyterhub` has the highest MTRO of both the Duplicate Issue Hub and Extended PR workflow types.

`App-vNext/Polly` tends to have low MTROs for the workflow types it contains. The project is in the bottom five repositories for MTRO in 4 workflow types: Extended PR, Divergent PR, Decomposed Issue, and Consequent Issue-PR. For the Extended PR workflow, the project has a relatively low number of matches. However, for the other three workflow types, the project has an above average number of matches. In particular, for Consequent Issue-PR, it has the highest amount of matches yet the lowest MTRO.

`Rapptz/discord.py` shows that a project can have a high MTRO for one workflow type, and a low MTRO for another workflow type. For example, it has a MTRO of 66.46% for the Duplicate Issue Hub workflow, but has the most matches. However, it has the highest MTRO for the Competing PRs workflow type, with only one match.

4.4 RQ4: Connected Component Characteristic Trends

In de Souza et al., we aimed to characterize the whole issue-PR graph by explaining the power-law distribution of connected component sizes we observed [6]. Here, we examine the trends in connected component characteristics as their size increases.

We propose that small and large components represent different levels of activity of work, as we found that small and large connected components were significantly different in many aspects. For example, large components have more issues, a higher proportion of merged PRs, and a longer component duration as a result of more coordination and active development. They represent work that evolves to require high levels of context, which occurs much more rarely than the less complex work represented by small components.

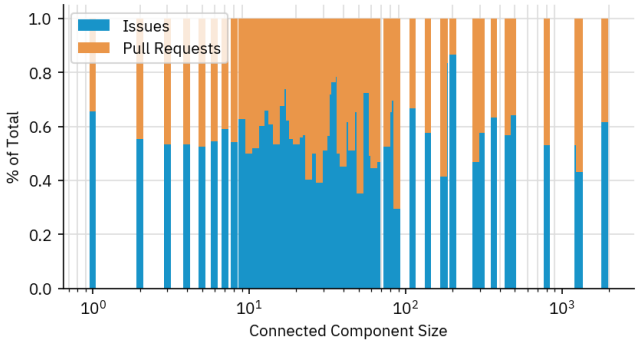
We calculated the distribution between issues and PRs, the distribution between node statuses, and the durations of connected components, bucketed by size. There were no clear trends in issue/PR distribution, node status distribution, or connected component duration observed across the all connected component sizes, with no or slight correlations observed.

The results in this section include all connected components from all projects.

4.4.1 *Issue / PR Distribution*

Figure 8 shows the distribution of issues and PRs across connected component sizes.

Figure 8. Distribution of Issues and PRs across connected component sizes.



No linear correlation (0.069) between issue component percentage and size was found. Comparing the extremes of small connected components (size < 10) and large connected components (size > 100), 69.26% of nodes in small connected components were issues, dropping to 58.05% of nodes in large connected components. The Chi-square statistic is 145.11, thus this difference in issue / PR percentage across component sizes is statistically significant ($p < 0.00001$).

4.4.2 *Node Status Distribution*

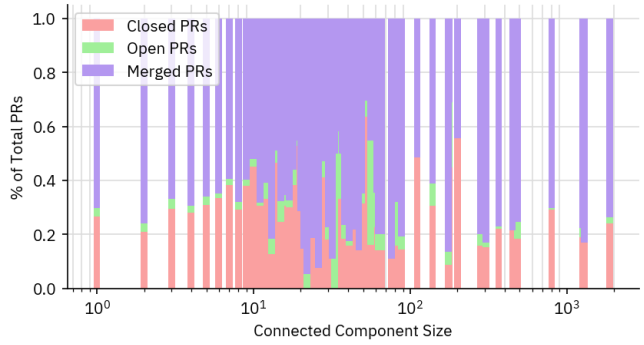
Figure 9 shows the distribution of issue statuses (i.e. open / closed) and Figure 10 shows the distribution of PR statuses (i.e. open / closed / merged) across connected component sizes.

No linear correlation (0.084) was observed for open issues with increasing connected component size. Figure 9 also does not immediately indicate any clear trend in issue status with increasing connected component size.

Figure 9. Distribution of issue statuses across connected component sizes.

A comparison of the extremes of small and large connected component yields a statistically significant difference. In small connected components, 41.94% of issues are open, and 58.06% are closed. In large connected components, 13.85% are open, and 86.15% are closed. The Chi-square statistic here is 538.23, with $p < 0.00001$.

Figure 10. Distribution of PR statuses across connected component sizes.



Both closed and open PR statuses had very slight negative correlations (-0.132 for closed PRs, and -0.099 for open PRs) with increasing connected component size. Merged PR statuses had a slight positive correlation with increasing size (0.180).

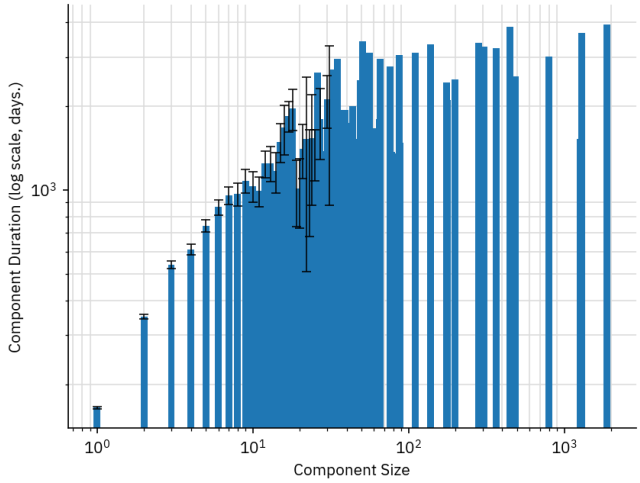
Comparing extremes of small and large connected components yields a statistically significant difference. In small connected components, 12.20% of PRs are open, 24.40% are closed, and 63.40% are merged. Large connected components have 3.83% of their PRs open, 19.58% closed, and 76.59% merged. The Chi-square statistic is 108.35, with $p < 0.00001$.

4.4.3 Connected Component Duration

We observed an initial increasing trend in component duration (strong positive correlation of 0.954) until around component size 15. For larger sizes, there appeared to be a plateau of component duration even with increasing component sizes (moderate positive correlation of 0.467). Over all connected component sizes, the correlation was moderate, at 0.490. Due to a low number of samples for some mid-size component sizes, there is a large standard error of the mean for some sizes.

For small connected components, the average duration was 730 days, and for large connected components, the average duration was 3095. The t-test statistic is 9.88, with a p-value < 0.0001.

Figure 11. Connected component duration by connected component size across all projects. Error bars mark standard error of the mean.



4.5 RQ5: Most Frequent Connected Component Isomorphisms

We calculated the top twenty most frequent connected component isomorphisms of each size up to fifteen. Several of our subgraph isomorphisms map closely to the topological structures of workflow types, supporting that our workflow types reflect common work structures. Select examples of these component isomorphisms are shown below.

Figure 12 shows the most frequent connected component isomorphisms for sizes three through six. At sizes above six, the most frequent isomorphism had only one or two matches.

Several of the other most frequent isomorphisms also appear structurally similar to our workflow types, lending some support to our definitions. For example, Figure 13 shows a common isomorphism similar to the Extended PR workflow type topology and Figure 14 shows another isomorphism similar to the Competing PRs topology.

The most frequent connected component isomorphisms usually do not contain many explicit link types.

The most frequent isomorphism of a connected component of size two was a typical issue-PR link, making up 39.85% of size two isomorphisms. The next most frequent size two isomorphism was an issue-issue link, making up 24.54%, and a PR-PR link, making up 12.67%. Figure 15 illustrates these isomorphisms.

5 Discussion

In this project, we have explored the concept of workflow type topologies and proposed a topological perspective on issue-PR graphs in repositories. This perspective enables contextual analysis impossible by examining individual links.

Figure 12. Most frequent connected component isomorphism for sizes three to six.

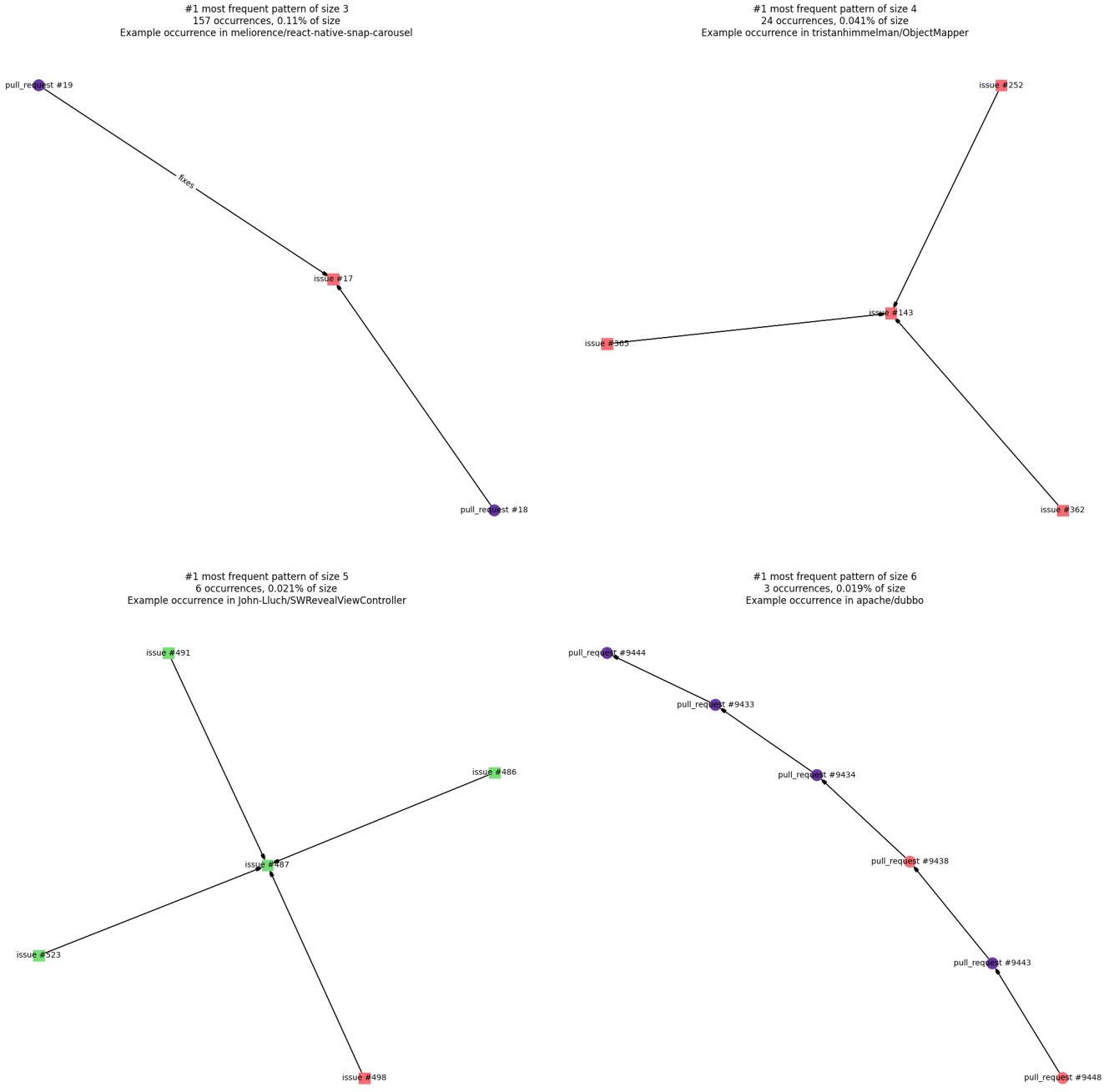


Figure 13. Seventh most frequent connected component isomorphism of size three. It closely resembles the Extended PR workflow type.

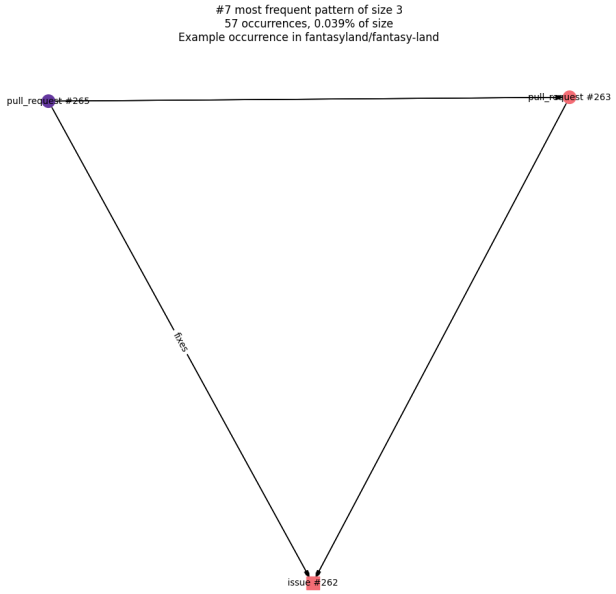


Figure 14. Ninth most frequent connected component isomorphism of size four. It closely resembles the Competing PRs workflow type.

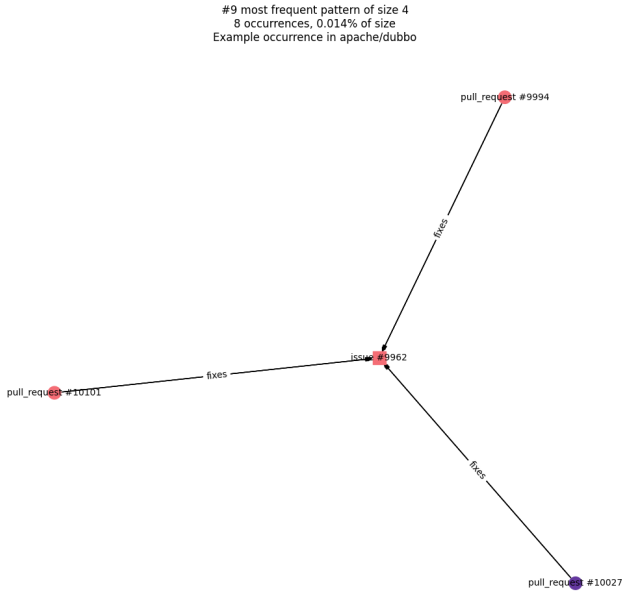
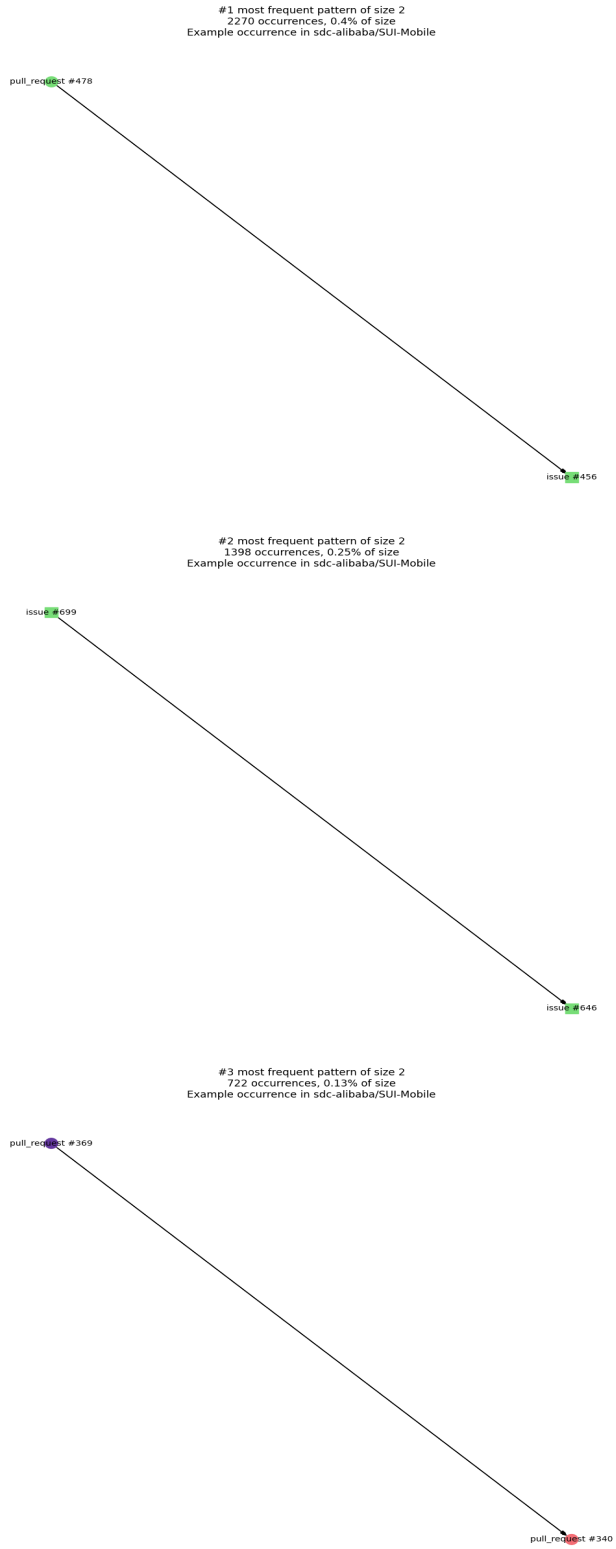


Figure 15. Top three most frequent isomorphisms of size two, ordered from top to bottom in decreasing number of matches.



5.1 RQ1: Workflow Type Characterization

The use of topological queries to search for workflow types allowed us to examine a sampling of each workflow type. Understanding how frequently workflow types are matched, and the structures of matches, also allows us to compare workflow types and further discern how they are applied. From there, we were able to draw conclusions on the general applications of each workflow type and the limitations of our query-based approach.

Our topological perspective on workflow type identification enables matching of workflow types especially well when topologies utilize explicit GitHub links and effective metadata heuristics. For example, we know for certain that the matches of the Duplicate Issue Hub workflow type, which uses the GitHub-native "duplicate" relationship in its topology, are true positives. Similarly, we can be quite confident that matches of the Integrating PR/Issue Hub topology are true positives, because of the effectiveness of the temporal heuristic we apply (all PRs connected to the central PR must have been created before the central PR). During the iterative process of developing these two workflow type queries, no false positives were matched for the Duplicate Issue Hub workflow type, and few were matched for the Integrating PR/Issue Hub.

The Integrating PR/Issue Hub workflow type topology expresses a unique "consequence" relationship, where the central node was created after all other PRs to document a release. As well, the topology indicates that all "spoke" PRs extending from the central node share some common trait that led them to be grouped into an integrating hub. Depending on the type of integration, we can infer that these spoke PRs are temporally or topically similar. This is a strength of our topological perspective, as this type of context across multiple nodes cannot be derived from a single link classification alone.

Figure 1 show that some workflow types are more frequently matched than others, which likely reflects how often they are adopted by open-source developers[5].

The Consequent Issue and Consequent Issue-PR had by far the highest number of matches out of all the defined topologies. This may be because of the less restrictive topology definition: the link between the first PR and the second issue (and optionally between the second issue and second PR) do not have any specified link type. This increases the opportunities for a match to occur, as explicit links make up a small number of the total links in the data set. This may also imply that Consequent Issue workflow types arise naturally in pull-based development, where PRs and issues build sequentially off one another.

In contrast, the Competing PRs and Extended PR workflow types had few matches (fifteen and fourteen respectively). This is unexpected for the Competing PRs workflow, since in open-source development it seems likely that wasted efforts in PR competition to create the best solution would occur, as proposed by Raymond [16]. The low number of matches may reflect overly strong constraints, but a more likely explanation is that competition is avoided in the surveyed repositories by effective communication [17].

In the Extended PR case, the low number of matches may also be due to the additional structural constraint that the PRs were connected. Decomposed Issue, a similar topology requiring that there is no link between PRs, has a higher number of matches. One hypothesis for this behaviour is that when multiple PRs address the same issue, they tend to address parts of it separately. By not linking between PRs, this shows that the contents of the PRs likely do not depend on each other or require context from the other PR, so the PRs' work is distinct. This would explain why Decomposed Issue is more often matched than Extended PRs.

The Consequent Issue-PR topology extends and is a special case of the Consequent Issue type. We therefore expected fewer matches of the Consequent Issue-PR topology, as it requires an extra PR addressing the second issue. However, there are 471 matches of the Consequent Issue-PR workflow type, and 210 of the Consequent Issue type: more than double. This indicates that in the studied repositories, addressing follow-up issues is a more common development practice than leaving them stale, even though it requires additional collaborative work.

In the Consequent Issue and Issue-PR, Competing PRs, and Extended PR workflow types, we have discussed a structural constraint as a potential reason why a topology is frequently or infrequently matched. These constraints may be causing false results from our queries, because we lack rich enough metadata to more effectively target matches. For example, the Competings PR workflow type currently uses a temporal constraint to ensure all PRs are "truly" competing with each other instead of being left stale, in which case there is no current competition. However, there is no "competes with" link type on GitHub that we could use to disambiguate this. For workflow types like the Duplicate Issue Hub, where we utilize one of GitHub's explicitly supported links, we can be more confident that our topology captures the workflow type well. A richer body of link types would thus help make queries more specific, and in Section 7, we discuss potential approaches for doing so.

The characterization of hub topologies is also important to highlight the variability and flexibility within prototypical workflow types. The Integrating PR/Issue Hubs had high average size, and a high size standard deviation as well. This represents the wide variability of releases across projects: while some may link many PRs in a release, others may aim to keep releases rolling and small. This highlights the flexibility of the workflow type, where projects can adapt its structure to their needs while retaining the same semantics.

The Competing PRs topology only had matches of size three, meaning no topologies with more than two closed PRs were found. This is likely due to the temporal constraint on the PRs having been created within a week of each other. Though manual inspection revealed that this constraint limited false positives effectively, it may have also excluded some valid

[5]Because we have not yet characterized queries' false positive rate, we cannot say for certain that this *is* how often they are adopted.

matches with PRs created more than one week apart[6]. Alternatively, it may indicate that open-source maintainers communicate effectively about current work being done on issues, so few people submit competing PRs.

Similarly, the Dependent PRs topology had a maximum of three stacked PRs across all matches. This may also be due to a constraint, as the authorship of the PRs is restricted to having one individual creating at least half of the PRs. However, this may instead reflect a natural complexity limit in dealing with Dependent PRs. When the number of PRs depending on each other grows, it becomes harder to bubble down changes made in earlier PRs and Git operations become more convoluted. This is consistent with the restricted match sizes and demonstrates the inflexibility of the workflow type.

The other topologies with multiplicity had average sizes close to their minimums, tending not to include additional relationships beyond what was required for their topology definition. These topologies have little structural variation, so are also relatively inflexible. This could be due to status or edge type constraints making it unlikely that any additional nodes would be included in the match.

5.2 RQ2: Project Representativeness

The distribution of workflow types across projects is important to ensure our workflows represent the projects well. Around half of the nodes meeting basic criteria for a workflow type match were matched as part of a topology, and almost all projects included at least one workflow type. This indicates that we have a fair amount of coverage in workflow types of larger repositories. As well, we found that the repositories with no topology matches were small, implying that a lack of maturity in a project leads to fewer applications of workflow types.

A moderate positive correlation (0.622) between workflow type matches in a repository and its number of nodes appears to hold across all projects: there is also a general upwards trend in Figure 7. It is possible that these repositories simply contain a large proportion of nodes, and thus relatively more opportunities for workflow types to have been matched, even by "chance" development. This is explored further in Section 5.3.

`App-vNext/Polly` is an outlier in this regard, with a relatively small number of nodes but the highest overall number of topology matches. This seems to indicate that the project frequently structures its work in one of the workflow types, implying it has an organized open-source contribution workflow.

We instead propose that the repositories that included a large percentage of matches for a workflow type are likely the repositories that adopt the workflow type, both implicitly and explicitly. For example, `apache/dubbo` contained 36.17% of Integrating PR/Issue Hub matches. The project deploys a weekly report bot that opens an issue listing all PRs merged in the last week, formally applying the Integrating PR/Issue Hub topology to increase situational awareness in the repository. As well, `Rapptz/discord.py` contained 37.78% of all Duplicate Issue Hub matches. While the project does not appear to have a formal requirement to explicitly mark issues as "duplicate" through the GitHub platform, as opposed to simply linking the two issues, the project's maintainers seem to frequently do so to avoid repeating support work and to organize addressed issues.

The projects with no workflow type matches, shown in in Table 13, make up 8.88% of the total nodes, and are generally quite small with only several hundred nodes each. These projects would therefore have fewer opportunities for workflow types to arise. As well, these projects are likely less mature and less active projects compared to the larger repositories. There is likely a lower number of core contributors and less coordination between them, so it is unsurprising that these repositories contain no workflow type matches.

5.3 RQ3: Topology Occurrence Opportunities

Match Topology Repository Opportunities, or MTRO, is a ratio assessing how frequently a project utilizes a workflow when able to do so. While comparing raw match counts may give a skewed result for repositories with higher node counts, MTRO normalizes across the number of candidate nodes of a workflow type in each repository. A high MTRO represents that there are many candidate nodes in a repository that could have started a workflow type match but ultimately did not evolve into the workflow types. The MTROs observed were >75% for all projects, indicating that a large proportion of the candidate nodes for a workflow type in each repository were not part of a match.

The lower MTROs (between 75-85%) were observed in topologies with more matches (e.g. Consequent Issue) and with larger average sizes (e.g. Integrating PR/Issue Hub). This is as expected, due to more of the candidates in the repository being part of a topology.

MTRO is affected by the number of matches in a project, which implicitly decreases the number of candidate nodes not in a topology. Some repositories with low numbers of matches therefore had high MTROs. For example, `deployphp/deployer` had the highest MTRO across 5 workflow types because it also had relatively fewer matches of the topologies it matched (e.g. Consequent Issue-PR, Dependent PRs) compared to other repositories.

While MTRO is impacted by the amount of matches in a project, it is also not necessarily a simple mapping due to the normalization over the number of candidates in the repository. We observed that `App-vNext/Polly` has the highest number of Consequent Issue-PR matches, but also has the lowest MTRO for the workflow type. This indicates that most of the closed issues (source node for Consequent Issue-PR topology) in this repository were a part of a match. Further extension of issues likely occurs frequently for closed issues in this project.

See also `Rapptz/discord.py`, which has a low MTRO for the Duplicate Issue Hub workflow type and the highest MTRO for Competing PRs. This also shows topologies with the

[6]The initial sampled component from prior work that motivated the definition of the Competing PRs topology was not matched due to this constraint.

same source node can also have different MTROs in the same project, highlighting their relative frequencies of adoption. Though the source node for both of these topologies is a closed issue, fewer opportunities evolve into Competing PRs compared to Duplicate Hubs. This is also reflected by their workflow types matches: `Rapptz/discord.py` has far more Duplicate Hub matches than Competing PRs.

MTRO can be used as a heuristic to assess if workflow type matches could be attributed to chance development events. If so, we would expect close to half of candidate topologies to contain a match. However, there are few workflow types with MTROs close to 50%. Almost all repositories have high MTROs > 90% for all workflow types, indicating there is likely intentional effort behind the occasional adoption of these workflow types. The existence of outlier repositories with low MTROs < 50% also supports this, indicating that those projects frequently apply those workflow types when given the opportunity. This suggests a coordinated effort to adopt the work practices associated with those workflow type.

5.4 RQ4: Connected Component Characteristic Trends

Trends in connected components across sizes imply that different types of work are done in components of different sizes. We found that very small connected components (those with size < 10 nodes) had more issues and shorter component durations than large connected components. Just over half (58.06%) of issues were closed, and most PRs (63.50%) were merged. In contrast, very large connected components (those with size > 100 nodes) had fewer issues and longer durations. A large proportion of issues were closed (86.15%). The same is true of PRs, with 76.59% merged.

There is no clear trend in issue / PR type, issue status, or PR status across all connected component sizes. This may be due to the limited number of samples of connected components of some sizes.

However, the extreme groups of small and large connected components are significantly different in all of these characteristics. For example, larger connected components tended to have a higher proportion of merged PRs. This is unexpected, as we believed that large connected components were artifacts of highly complex and competitive work where many PRs would be closed instead of merged. However, this does not appear to be the case: connected components of large sizes have comparatively more merged PRs, indicating higher proportions of accepted work.

This implies that small and large components comprise different types of related work. Large connected components can be interpreted as more "active" areas of a project, with work spanning longer time intervals and more work (PRs) being accepted than closed. These components may represent more mature areas of work in a project, with more context enabling additional work to be done effectively. These areas may tend to address existing historical issues rather than create new ones, explaining the higher proportion of closed issues in large components.

On the other hand, small connected components may represent less complex work or still-nascent work. Small features likely don't require the past context that larger initiatives within a project do, so they may tend not to link to other nodes. In these cases, it may be more likely that new issues are created frequently and left open as current works-in-progress, supported by the higher proportion of open issues and PRs in small components.

Under this theory, it appears logical that small connected components have lower component durations, and larger components have higher durations, scaling with the amount of existing work done in the area. The component size duration has a clear upwards trend, supporting this initial hypothesis. Besides having large numbers of nodes with more discussion, these larger components are usually comprised of several smaller clusters of nodes connected together. There are therefore more opportunities for a recent comment on a new node within one cluster of the component to drastically increase the overall component duration. This is consistent with the interpretation that large components represent more active and mature initiatives within a project, which would likely have recently updated nodes as well as older ones.

It is interesting to note that even relatively small connected components have durations of over a year: the average component duration of a component of size ten is around two years, and just under a year and a half for components of size five. Even isolated nodes are last updated more than half a year later on average. This contrasts with the results of Kikas et al., who observed that stale issues in their data set had a maximum duration of about 100 days [12]. This may be due to different repositories being studied, but may also reflect the evolution of the open source maintenance efforts of our studied projects. For example, when a project is small, issues and PRs may be left stale or unlabelled. This strategy fails to scale as more contributions come in. The update events taking place long after node creations may reflect organization efforts being retroactively applied to nodes. As well, contributors to newer issues may frequently be referred to past solutions and similar bug reports, which would lead to late updates.

5.5 RQ5: Most Frequent Connected Component Isomorphisms

Our analysis of isomorphic connected components is also interesting because a mapping from frequent isomorphic connected components to workflow types would further validate our workflow type definitions as examples of common development work. We observed structural similarities between the most common isomorphisms of sizes three through six, lending some support to our definitions

The most frequent isomorphisms for sizes greater than six had only one or two matches, indicating high variability in topology structure. This is as expected, as there are more permutations of components with additional nodes. Even with smaller topologies, there are fewer than a couple hundred matches for the most frequent isomorphism, also due to the isomorphism constraints.

The frequently identified connected components tended not to have explicit edge types, which is as expected, as only 10.87% of the links in the issue-PR graphs were explicitly labelled

"fixes" or "duplicate". This lack of links makes it more difficult to find isomorphisms similar to workflow types among the most frequent isomorphisms. Several of the most frequent isomorphisms in Figure 4 are structurally similar to a workflow type: Size 3 looks like a Consequent Issue, Size 5 could be a Duplicate Hub, and Size 6 looks like a Dependent PRs match. While this gives credence to our workflow type definitions, our definitions also depend on the existence of explicitly labelled links. We therefore cannot say for certain that our workflow types are among the most frequent isomorphisms.

The most frequent isomorphism of a connected component of size two was a typical issue-PR link. We expected this, as this link represents the prototypical pull-based development model. However, the next most frequent isomorphisms were an issue-issue link, making up 24.54% of two-node connected components, and a PR-PR link, making up 12.67%. None of these isomorphisms had explicit link labelling, which indicates that their link semantics are likely among the alternative cases covered by our nine edge type definitions [6]. Because these unlabelled two-node links are common, this supports the need for additional relationship classification to distinguish these isomorphic links.

6 Limitations

There are two main limitations of the above findings. For one, we were unable to take into account additional GitHub metadata besides what was previously scraped. For example, we were unable to accurately track the number of seasoned contributors across connected component sizes because we had not initially scraped this data and there was no way to access the number of contributors at a previous point in time. As a result, we relied on heuristics for several of our topologies. For example, the clearest way to identify a Dependent PRs workflow type is to compare the source and destination branches of the PR to determine if there is a dependency "stack". This data was not available without further scraping, so we resorted to using an authorship heuristic and checking link direction instead. This lack of continual scraping also drove us to focus on closed and merged nodes in our topologies, because they represented work that had already "finished" as of data collection.

As well, though workflow type topologies were first informed and further iterated on by qualitative coding, no semantic analysis on the returned workflow type matches was done. Because there was a large number of components returned by the queries and due to time constraints, we were unable to manually examine them all. We therefore cannot verify the true positive rate of our queries.

7 Future Work

One crucial aspect of future work is manual coding of a sample of the returned workflow type matches to determine the false positive rate of the queries. We have noted that the lack of thorough manual verification of these matches is a limitation of our study, as we have no objective measure of our queries' accuracy and cannot make conclusive statements about how frequently work practices are applied.

Future work could also augment our existing workflow type definitions with more specific constraints, particularly on the link types. As discussed in Section 5.1, the availability of explicit GitHub metadata to use in query constraints, like querying on the "duplicate" link type, increased the effectiveness of certain workflow type queries. Additional GitHub metadata, like comment, branch, and author data, could be used to create an automatic link classifier according to the nine edge link types described in de Souza et al. [6]. Link types and constraints in our workflow types queries could then be amended appropriately. This would decrease false positive rate and more accurately target workflow types.

In addition, further development of a project explorer tool that integrates the workflow type topologies discussed here could aid in further issue-PR graph research. While a skeleton module has been completed during this project to generate basic queries from a provided graph JSON file, integration with an interactive visualization remains to be done. This tool would also be useful for open-source project governance and maintenance to simplify the identification and querying of workflow types.

In Section 3.6.2, we also explored the concept of workflow type sequences, where a match of one workflow type would lead into a match of another workflow type. Due to time constraints and computational limits, we were unable to fully pursue this idea, but we believe this could lead into interesting findings around high-level meta-patterns of work.

Finally, we have considered conducting interviews with open-source project contributors, particularly those that have participated in a workflow topology, to assess how they perceive these topologies. This would externally validate the accuracy of our workflow type definitions and potentially provide insight into additional workflow types.

8 Conclusion

In this project, we studied issue-PR graphs to understand how projects apply software development practices from a topological perspective. Our work expands on prior research on individual links to uncover larger patterns of collaboration. For my Directed Studies report, I focused on the quantitative analysis supporting the qualitative findings in de Souza et al.[6].

We defined and optimized 8 workflow type patterns, finding that their occurrences were unevenly distributed. We also found that some workflow types frequently included more nodes than others, highlighting their flexibility. In addition, we observed that different repositories applied workflow types at varying frequencies, with larger repositories correlated to a higher frequency of matches. This shows that more mature projects tend to adopt workflow types more frequently. We further supported this with a heuristic that indicates that repositories rarely adopted a workflow type given an opportunity to do so, suggesting the usage of workflow types is likely an intentional step. When comparing small and large connected components, we also found that they differ in component duration and node type and status composition, indicating that

they comprise different categories of work. Finally, we examined isomorphism detection among connected components across all projects, noting that several of the most frequent isomorphisms closely reflected our workflow type structures.

This work contributes novel tools for topology identification and visualization, for use in further qualitative research or open-source maintenance. It opens up the door for further automated analysis of workflow types. For example, in de Souza et al., we have identified some workflow types, like Competing PRs, Duplicate Issue Hubs, and some cases of Extended PRs, as potentially negative [6]. The quantitative tools described in this report can aid in the identification of "harmful" workflow patterns on a repository level. Open-source projects can then tailor maintenance efforts to facilitate more productive work, even in large repositories that would not be possible to otherwise examine.

We believe that a topological perspective on software development can clarify complex collaboration patterns and provide concrete implications for a project's work. The additional context from analyzing whole topologies, both qualitatively and quantitatively, can improve our understanding of how collaborative development empirically takes place.

9 Acknowledgments

I would like to sincerely thank Professor de Souza, Professor Yoon, and Professor Beschastnikh for supporting me throughout this Directed Studies project and for introducing me hands-on to the world of software research. I greatly appreciate the opportunity to explore a project so closely aligned to my interests and in such an encouraging environment. This Directed Studies experience has been the highlight of my academic journey so far and it wouldn't have been possible without them.

I would also like to thank my family, friends, and other professors who have guided me along the path I now pursue. Thank you in particular to those who have let me bounce ideas off them and stuck with me as I worked on this report.

10 References

[1] Zakarea Alshara, Anas Shatnawi, Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, and Maad Shatnawi. 2022. PI-Link: A Ground-Truth Dataset of Links Between Pull-Requests and Issues in GitHub. *IEEE Access* 11 (2022), 697–710. DOI: http://dx.doi.org/10.1109/ACCESS.2022.3232982

[2] Jeff Alstott, Ed Bullmore, and Dietmar Plenz. 2014. powerlaw: A Python Package for Analysis of Heavy-Tailed Distributions. *PLoS ONE* 9, 1 (Jan 2014), e85777. DOI: http://dx.doi.org/10.1371/journal.pone.0085777

[3] Andrew Bowman. n.d. Tuning Cypher queries by understanding cardinality. (n.d.). https://neo4j.com/developer/kb/understanding-cypher-cardinality/

[4] 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. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (Oct 2021), 1–25. DOI:http://dx.doi.org/10.1145/3479529

[5] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. 2009. Power-Law Distributions in Empirical Data. *SIAM Rev.* 51, 4 (Nov 2009), 661–703. DOI: http://dx.doi.org/10.1137/070710111

[6] Cleidson R. B. de Souza, Jesse Wong, Emilie Ma, Dongwook Yoon, and Ivan Beschastnikh. 2023. Revealing Work Practices in Pull-Based Software Development through Issue-PR Graph Topologies. (Mar 2023). Submitted to ICSE 2024.

[7] NetworkX Developers. n.d. NetworkX — NetworkX documentation. (n.d.). https://networkx.org/

[8] NetworkX developers. n.d. VF2 Algorithm — NetworkX 3.0 documentation. (n.d.). https://networkx.org/documentation/stable/reference/algorithms/isomorphism.vf2.html

[9] 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*. ACM, Tallinn Estonia, 578–589. DOI:http://dx.doi.org/10.1145/3338906.3338949

[10] Neo4j Inc. n.d.a. Cypher Query Language - Developer Guides. (n.d.). https://neo4j.com/developer/cypher/

[11] Neo4j Inc. n.d.b. Neo4j graph data platform – the leader in graph databases. (n.d.). https://neo4j.com/

[12] Riivo Kikas, Marlon Dumas, and Dietmar Pfahl. 2015. *Issue Dynamics in Github Projects*. Vol. 9459. Springer International Publishing, Cham, 295–310. DOI: http://dx.doi.org/10.1007/978-3-319-26844-6_22

[13] 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)*. IEEE, Nara, Japan, 386–395. DOI: http://dx.doi.org/10.1109/APSEC.2018.00053

[14] Alexander Nicholson, Deeksha M. Arya, and Jin L.C. Guo. 2020. Traceability Network Analysis: A Case Study of Links in Issue Tracking Systems. In *2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. IEEE, Zurich, Switzerland, 39–47. DOI: http://dx.doi.org/10.1109/AIRE51212.2020.00013

[15] 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*. ACM, Gothenburg

Sweden, 834–845. DOI: http://dx.doi.org/10.1145/3180155.3180207

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

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

[18] 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:: A case study of the OpenStack and Qt projects. *Empirical Software Engineering* 26, 5 (Sep 2021), 96. DOI: http://dx.doi.org/10.1007/s10664-021-09997-x

[19] Jesse Wong. 2022a. *An investigation into the topologies formed by user created links between Github issues and PR nodes: A detailed report*. Directed Studies Report. University of British Columbia. https://docs.google.com/document/d/1MWDp3d1xirGBRDDQPGLe2bq1639bEWLQTMzbUCWDdOo/view

[20] Jesse Wong. 2022b. Yank Solicitation Project Crawler and Visualiser. private communication. (2022). https://github.com/Frozemint/pr_project_all_file

APPENDIX
RQ2: Project Representativeness

Table 4. Competing PRs workflow type distribution across all repositories.

Repository	Matches	Z-Score
mlflow/mlflow	4	1.65
jupyterhub/jupyterhub	3	0.76
apache/dubbo	3	0.76
kubernetes-sigs/kustomize	2	-0.13
rematch/rematch	1	-1.02
Rapptz/discord.py	1	-1.02
MithrilJS/mithril.js	1	-1.02

Table 5. Consequent Issue workflow type distribution across all repositories.

Repository	Matches	Z-Score
TypeStrong/ts-node	29	2.42
kubernetes-sigs/kustomize	28	2.31
apache/dubbo	25	1.96
mlflow/mlflow	24	1.84
MithrilJS/mithril.js	14	0.69
jupyterhub/jupyterhub	13	0.57
varvet/pundit	11	0.34
fantasyland/fantasy-land	8	-0.01
summernote/summernote	7	-0.12
grpc/grpc-web	7	-0.12
Project-OSRM/osrm-backend	6	-0.24
chaijs/chai	5	-0.36
nytimes/NYTPhotoViewer	5	-0.36
deployphp/deployer	4	-0.47
casesandberg/react-color	4	-0.47
BurntSushi/toml	3	-0.59
redis/redis-rb	3	-0.59
Rapptz/discord.py	3	-0.59
rematch/rematch	2	-0.70
tristanhimmelman/ObjectMapper	2	-0.70
App-vNext/Polly	2	-0.70
MagicStack/uvloop	1	-0.82
rauchg/slackin	1	-0.82
iron/iron	1	-0.82
transitive-bullshit/create-react-library	1	-0.82
jonschlinkert/remarkable	1	-0.82

Table 6. Consequent Issue-PR workflow type distribution across all repositories.

Repository	Matches	Z-Score
App-vNext/Polly	132	3.97
kubernetes-sigs/kustomize	62	1.53
chaijs/chai	52	1.18
Project-OSRM/osrm-backend	43	0.87
jupyterhub/jupyterhub	42	0.83
TypeStrong/ts-node	31	0.45
apache/dubbo	22	0.14
MithrilJS/mithril.js	22	0.14
rematch/rematch	16	-0.07
grpc/grpc-web	7	-0.39
nytimes/NYTPhotoViewer	6	-0.42
tiny-dnn/tiny-dnn	6	-0.42
imbrn/v8n	4	-0.49
BurntSushi/toml	3	-0.53
summernote/summernote	3	-0.53
mlflow/mlflow	3	-0.53
meliorence/react-native-snap-carousel	2	-0.56
redis/redis-rb	2	-0.56
MagicStack/uvloop	2	-0.56
Activiti/Activiti	2	-0.56
iron/iron	2	-0.56
Rapptz/discord.py	2	-0.56
duo-labs/cloudmapper	2	-0.56
deployphp/deployer	1	-0.60
casesandberg/react-color	1	-0.60
jhen0409/react-native-debugger	1	-0.60

Table 7. Decomposed Issue workflow type distribution across all repositories.

Repository	Matches	Z-Score
apache/dubbo	40	4.43
kubernetes-sigs/kustomize	10	0.72
mlflow/mlflow	6	0.23
App-vNext/Polly	5	0.11
rematch/rematch	3	-0.14
jupyterhub/jupyterhub	3	-0.14
Rapptz/discord.py	3	-0.14
chaijs/chai	2	-0.26
duo-labs/cloudmapper	2	-0.26
MithrilJS/mithril.js	2	-0.26
Flipboard/bottomsheet	2	-0.26
grpc/grpc-web	2	-0.26
Project-OSRM/osrm-backend	2	-0.26
BurntSushi/toml	1	-0.39
ag-grid/ag-grid	1	-0.39
MagicStack/uvloop	1	-0.39
Activiti/Activiti	1	-0.39
nytimes/NYTPhotoViewer	1	-0.39
deployphp/deployer	1	-0.39
casesandberg/react-color	1	-0.39
jonschlinkert/remarkable	1	-0.39
shadowsocks/shadowsocks-manager	1	-0.39

Table 8. Dependent PRs workflow type distribution across all repositories.

Repository	Matches	Z-Score
mlflow/mlflow	11	2.52
Project-OSRM/osrm-backend	9	1.85
kubernetes-sigs/kustomize	6	0.85
apache/dubbo	4	0.18
Activiti/Activiti	4	0.18
MithrilJS/mithril.js	4	0.18
redis/redis-rb	3	-0.16
tiny-dnn/tiny-dnn	3	-0.16
chaijs/chai	2	-0.49
pagekit/pagekit	1	-0.83
summernote/summernote	1	-0.83
tensorpack/tensorpack	1	-0.83
deployphp/deployer	1	-0.83
ptomasroos/react-native-scrollable-tab-view	1	-0.83
varvet/pundit	1	-0.83

Table 9. Divergent PR workflow type distribution across all repositories.

Repository	Matches	Z-Score
kubernetes-sigs/kustomize	13	2.07
TypeStrong/ts-node	13	2.07
MithrilJS/mithril.js	13	2.07
jupyterhub/jupyterhub	12	1.82
apache/dubbo	7	0.57
App-vNext/Polly	7	0.57
mlflow/mlflow	7	0.57
rematch/rematch	6	0.32
grpc/grpc-web	5	0.07
chaijs/chai	4	-0.18
iron/iron	4	-0.18
ptomasroos/react-native-scrollable-tab-view	4	-0.18
Project-OSRM/osrm-backend	4	-0.18
BurntSushi/toml	3	-0.43
summernote/summernote	2	-0.68
MagicStack/uvloop	2	-0.68
deployphp/deployer	2	-0.68
Rapptz/discord.py	2	-0.68
varvet/pundit	2	-0.68
redis/redis-rb	1	-0.93
rauchg/slackin	1	-0.93
imbrn/v8n	1	-0.93
marko-js/marko	1	-0.93
nytimes/NYTPhotoViewer	1	-0.93
tiny-dnn/tiny-dnn	1	-0.93

Table 10. Duplicate Issue Hub workflow type distribution across all repositories.

Repository	Matches	Z-Score
Rapptz/discord.py	17	3.57
metafizzy/flickity	7	1.05
kubernetes-sigs/kustomize	4	0.30
tensorpack/tensorpack	2	-0.20
MithrilJS/mithril.js	2	-0.20
mlflow/mlflow	2	-0.20
jhen0409/react-native-debugger	2	-0.20
jupyterhub/jupyterhub	1	-0.46
pagekit/pagekit	1	-0.46
chaijs/chai	1	-0.46
mgonto/restangular	1	-0.46
rlidwka/sinopia	1	-0.46
TypeStrong/ts-node	1	-0.46
nytimes/NYTPhotoViewer	1	-0.46
cglib/cglib	1	-0.46
Project-OSRM/osrm-backend	1	-0.46

Table 11. Extended PR workflow type distribution across all repositories.

Repository	Matches	Z-Score
apache/dubbo	5	2.12
mlflow/mlflow	3	0.71
MithrilJS/mithril.js	2	0.00
jupyterhub/jupyterhub	1	-0.71
TypeStrong/ts-node	1	-0.71
App-vNext/Polly	1	-0.71
imbrn/v8n	1	-0.71

Table 12. Integrating PR/Issue Hub workflow type distribution across all repositories.

Repository	Matches	Z-Score
apache/dubbo	34	3.77
Project-OSRM/osrm-backend	12	0.89
chaijs/chai	8	0.36
summernote/summernote	8	0.36
mlflow/mlflow	6	0.10
jupyterhub/jupyterhub	5	-0.03
kubernetes-sigs/kustomize	4	-0.16
MithrilJS/mithril.js	4	-0.16
App-vNext/Polly	3	-0.29
tristanhimmelman/ObjectMapper	2	-0.42
rauchg/slackin	1	-0.55
Activiti/Activiti	1	-0.55
TypeStrong/ts-node	1	-0.55
deployphp/deployer	1	-0.55
SwipeCellKit/SwipeCellKit	1	-0.55
varvet/pundit	1	-0.55
grpc/grpc-web	1	-0.55
jhen0409/react-native-debugger	1	-0.55

Table 13. Repositories containing no workflow type matches.

Repository
sdc-alibaba/SUI-Mobile
John-Lluch/SWRevealViewController
deepinsight/insightface
zaach/jison
stacktracejs/stacktrace.js
volatilityfoundation/volatility
sosedoff/pgweb
tsayen/dom-to-image
amphp/amp
cruffenach/CRToast
roboguice/roboguice
go-chi/chi

RQ3: Topology Occurrence Probability

Table 14. Competing PRs workflow type MTRO per repository

Repository	MTRO	Matches
Rapptz/discord.py	99.59%	1
kubernetes-sigs/kustomize	99.54%	2
apache/dubbo	99.10%	3
jupyterhub/jupyterhub	98.95%	3
rematch/rematch	97.85%	1
mlflow/mlflow	97.64%	4
MithrilJS/mithril.js	94.53%	1

Table 15. Consequent Issue workflow type MTRO per repository

Repository	MTRO	Matches
deployphp/deployer	98.16%	4
Rapptz/discord.py	98.16%	3
rauchg/slackin	95.45%	1
redis/redis-rb	95.15%	3
summernote/summernote	93.44%	7
Project-OSRM/osrm-backend	92.85%	6
rematch/rematch	91.40%	2
apache/dubbo	90.15%	25
tristanhimmelman/ObjectMapper	90.00%	2
chaijs/chai	89.88%	5
grpc/grpc-web	88.31%	7
jupyterhub/jupyterhub	86.79%	13
iron/iron	86.00%	1
mlflow/mlflow	84.57%	24
casesandberg/react-color	83.81%	4
App-vNext/Polly	82.04%	2
jonschlinkert/remarkable	76.00%	1
MithrilJS/mithril.js	75.47%	14
kubernetes-sigs/kustomize	72.69%	28
TypeStrong/ts-node	68.61%	29
MagicStack/uvloop	65.45%	1
BurntSushi/toml	61.11%	3
varvet/pundit	60.00%	11
transitive-bullshit/create-react-library	57.89%	1
nytimes/NYTPhotoViewer	38.64%	5
fantasyland/fantasy-land	34.25%	8

Table 16. Consequent Issue-PR workflow type MTRO per repository

Repository	MTRO	Matches
deployphp/deployer	99.48%	1
mlflow/mlflow	99.27%	3
Activiti/Activiti	99.21%	2
Rapptz/discord.py	99.18%	2
meliorence/react-native-snap-carousel	98.60%	2
duo-labs/cloudmapper	98.36%	2
jhen0409/react-native-debugger	97.47%	1
summernote/summernote	95.31%	3
redis/redis-rb	94.17%	2
apache/dubbo	90.28%	22
casesandberg/react-color	89.52%	1
Project-OSRM/osrm-backend	86.47%	43
iron/iron	86.00%	2
jupyterhub/jupyterhub	81.55%	42
BurntSushi/toml	79.63%	3
tiny-dnn/tiny-dnn	76.04%	6
chaijs/chai	75.60%	52
MithrilJS/mithril.js	72.64%	22
kubernetes-sigs/kustomize	68.29%	62
grpc/grpc-web	66.88%	7
MagicStack/uvloop	65.45%	2
TypeStrong/ts-node	64.48%	31
rematch/rematch	63.44%	16
imbrn/v8n	58.33%	4
nytimes/NYTPhotoViewer	52.27%	6
App-vNext/Polly	44.17%	132

Table 17. Decomposed Issue workflow type MTRO per repository

Repository	MTRO	Matches
deployphp/deployer	99.74%	1
ag-grid/ag-grid	99.62%	1
Activiti/Activiti	99.61%	1
MithrilJS/mithril.js	99.43%	2
Rapptz/discord.py	99.18%	3
Project-OSRM/osrm-backend	97.92%	2
mlflow/mlflow	97.28%	6
duo-labs/cloudmapper	95.90%	2
rematch/rematch	95.70%	3
jupyterhub/jupyterhub	95.18%	3
chaijs/chai	91.96%	2
casesandberg/react-color	89.52%	1
shadowsocks/shadowsocks-manager	86.67%	1
apache/dubbo	84.64%	40
grpc/grpc-web	83.12%	2
kubernetes-sigs/kustomize	81.94%	10
BurntSushi/toml	79.63%	1
jonschlinkert/remarkable	76.00%	1
Flipboard/bottomsheet	66.67%	2
MagicStack/uvloop	65.45%	1
nytimes/NYTPhotoViewer	63.64%	1
App-vNext/Polly	59.22%	5

Table 18. Dependent PRs workflow type MTRO per repository

Repository	MTRO	Matches
deployphp/deployer	99.64%	1
summernote/summernote	99.53%	1
ptomasroos/react-native-scrollable-tab-view	99.43%	1
varvet/pundit	99.35%	1
Activiti/Activiti	98.89%	4
tensorpack/tensorpack	98.27%	1
apache/dubbo	97.39%	4
kubernetes-sigs/kustomize	97.27%	6
Project-OSRM/osrm-backend	96.76%	9
mlflow/mlflow	96.50%	11
pagekit/pagekit	95.31%	1
MithrilJS/mithril.js	94.38%	4
redis/redis-rb	92.63%	3
chaijs/chai	91.04%	2
tiny-dnn/tiny-dnn	75.09%	3

Table 19. Divergent PR workflow type MTRO per repository

Repository	MTRO	Matches
deployphp/deployer	98.89%	2
mlflow/mlflow	97.16%	7
marko-js/marko	96.97%	1
Rapptz/discord.py	95.60%	2
apache/dubbo	95.41%	7
rauchg/slackin	95.00%	1
redis/redis-rb	93.86%	1
summernote/summernote	93.81%	2
varvet/pundit	93.22%	2
Project-OSRM/osrm-backend	92.88%	4
tiny-dnn/tiny-dnn	90.53%	1
rematch/rematch	87.76%	6
kubernetes-sigs/kustomize	85.54%	13
BurntSushi/toml	84.38%	3
chaijs/chai	83.00%	4
ptomasroos/react-native-scrollable-tab-view	80.65%	4
grpc/grpc-web	78.40%	5
jupyterhub/jupyterhub	76.96%	12
TypeStrong/ts-node	69.85%	13
MithrilJS/mithril.js	69.72%	13
imbrn/v8n	66.67%	1
nytimes/NYTPhotoViewer	65.12%	1
App-vNext/Polly	62.41%	7
iron/iron	61.90%	4
MagicStack/uvloop	51.02%	2

Table 20. Duplicate Issue Hub workflow type MTRO per repository

Repository	MTRO	Matches
jupyterhub/jupyterhub	97.69%	1
Project-OSRM/osrm-backend	97.00%	1
pagekit/pagekit	94.64%	1
mlflow/mlflow	94.37%	2
mgonto/restangular	94.19%	1
kubernetes-sigs/kustomize	93.29%	4
TypeStrong/ts-node	92.21%	1
chaijs/chai	91.96%	1
rlidwka/sinopia	91.89%	1
tensorpack/tensorpack	90.85%	2
jhen0409/react-native-debugger	88.61%	2
MithrilJS/mithril.js	87.92%	2
nytimes/NYTPhotoViewer	86.36%	1
Rapptz/discord.py	66.46%	17
metafizzy/flickity	64.37%	7
cglib/cglib	31.25%	1

Table 21. Extended PR workflow type MTRO per repository

Repository	MTRO	Matches
jupyterhub/jupyterhub	99.79%	1
TypeStrong/ts-node	99.76%	1
apache/dubbo	99.62%	5
mlflow/mlflow	99.09%	3
MithrilJS/mithril.js	94.53%	2
App-vNext/Polly	87.38%	1
imbrn/v8n	58.33%	1

Table 22. Integrating PR/Issue Hub workflow type MTRO per repository

Repository	MTRO	Matches
Activiti/Activiti	99.11%	1
TypeStrong/ts-node	92.46%	1
mlflow/mlflow	92.24%	6
deployphp/deployer	91.98%	1
kubernetes-sigs/kustomize	90.50%	4
jupyterhub/jupyterhub	89.58%	5
jhen0409/react-native-debugger	89.43%	1
grpc/grpc-web	87.46%	1
Project-OSRM/osrm-backend	87.34%	12
SwipeCellKit/SwipeCellKit	86.84%	1
rauchg/slackin	80.95%	1
App-vNext/Polly	77.81%	3
MithrilJS/mithril.js	77.75%	4
summernote/summernote	75.71%	8
varvet/pundit	72.73%	1
tristanhimmelman/ObjectMapper	66.84%	2
apache/dubbo	65.89%	34
chaijs/chai	64.74%	8