Does the Propagation of Artifact Changes across Tasks reflect Work Dependencies?

Christoph Mayr-Dorn
Johannes Kepler University
Linz, Austria
christoph.mayr-dorn@jku.at

Alexander Egyed
Johannes Kepler University
Linz, Austria
alexander.egyed@jku.at

ABSTRACT

Developers commonly define tasks to help coordinate software development efforts—whether they be feature implementation, refactoring, or bug fixes. Developers establish links between tasks to express implicit dependencies that needs explicit handling—dependencies that often require the developers responsible for a given task to assess how changes in a linked task affect their own work and vice versa (i.e., change propagation). While seemingly useful, it is unknown if change propagation indeed coincides with task links.

No study has investigated to what extent change propagation actually occurs between task pairs and whether it is able to serve as a metric for characterizing the underlying task dependency. In this paper, we study the temporal relationship between developer reading and changing of source code in relationship to task links. We identify seven situations that explain the varying correlation of change propagation with linked task pairs and find six motifs describing when change propagation occurs between non-linked task pairs. Our paper demonstrates that task links are indeed useful for recommending which artifacts to monitor for changes, which developers to involve in a task, or which tasks to inspect.

CCS CONCEPTS

• **Software and its engineering** → *Software evolution*;

KEYWORDS

task links, change propagation, bugzilla, mylyn, empirical study

ACM Reference Format:
Christoph Mayr-Dorn and Alexander Egyed. 2018. Does the Propagation of Artifact Changes across Tasks reflect Work Dependencies?. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3180155.3180185

1 INTRODUCTION

A task in software engineering defines a work item—usually for feature implementation, refactoring, or bug fixes.[1] Often, tasks are broken down into subtasks that can be solved by individuals. In such cases, tasks can be a coordination mechanism to manage software development efforts.

However, tasks may also arise out of other reasons. For example, dependencies among tasks commonly occur when closely related software artifacts are changed (e.g., methods that call one another or that share data) or when developers in one task wait for the output of another task. In such cases, the developers need to explicitly coordinate the involved tasks [**?**]. The developer responsible for a given task then has to not only understand the changes implied by the given task but also assess the impact of these changes onto dependent tasks. The developer responsible for a dependent task, in turn, studies these changes to assess the impact on his or her own work. Hence, we should expect that developers access artifacts in dependent tasks while working on their own— a practice we refer to as change propagation.

When creating tasks, developers face the challenge of identifying dependent tasks. One would assume that developers use the links offered by task management tools to make the implicit task dependencies explicit—links that may be inaccurate and incomplete at times. They then use the links to identify which developers to notify/involve about changes and which artifacts to change. Since identifying relevant engineers and artifacts for change propagation remains a significant problem [**? ? ?**], the question arises when do links actually reflect change propagation?

We believe insights into the usefulness and applicability of change propagation to identify task dependencies can provide an effective basis for novel support of developers during software change. Such insights are valuable beyond advising developers which linked tasks to monitor for changes [**?**]. They determine under which conditions change propagation metrics may detect implicit dependencies between non-linked tasks. Developers may then decide to link them, respectively monitor them for changes. Studying change propagation provides an understanding how a-posteriori analysis of change propagation may result in reclassifying existing links. This reduces a developer's effort to understand the implicit dependency between tasks and reveals further relevant tasks during future software evolution activities [**? ?**]. A concrete scenario in Section 1.1 motivates the importance of the presented research and potential benefits for developers in further detail.

To obtain these insights we need to closely investigate to which extent and under which conditions change propagation correlates with links and whether non-linked task pairs exhibit similar behavior. To the best of our knowledge, no study has investigated change propagation across tasks. Existing work typically identifies coordination needs by determining which artifacts are usually changed together [**? ? ?**], how to correctly propagate changes among artifacts [**? ?**], or how to detect inconsistencies [**? ? ?**]. Yet

[1] Throughout this paper we use the term *task* to represent any work item such as *issue*, *ticket*, *bug*, *change request*, *feature*, or *story*.

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05.
https://doi.org/10.1145/3180155.3180185

non of the approaches consider the significance of links for change propagation.

This paper analyses the temporal relationship between developer reading and changing source code on the one hand, and task links on the other hand, as found in the Mylyn data set. Mylyn is an open source task management tool for the Eclipse IDE that captures traces of developer interactions (i.e. artifact reads and writes). Mylyn developers use this tool during their work on Mylyn. Mylyn, therefore, serves as the data gathering tool as well as the system under investigation in this paper.

Ultimately, we make the following four contributions:
(1) we identify seven situations that explain when linked task pairs exhibit change propagation;
(2) we identify six motifs that explain why non-linked task pairs exhibit change propagation;
(3) we lay out the implications of the found situations and motifs on future development support tools; and
(4) we provide a data set that combines developer interactions, tasks, and task links.

Specifically, we find that 64% of linked task pairs exhibit change propagation (true positives). The remaining 36% false negatives can be explained by three situations in which developers use links to manage tasks dependencies that do not entail change propagation. Examples are task synchronization and task decomposition. We identify additional four situations that describe distinct artifact-centric task dependencies. Artifact reuse or work continuation dependencies, for example, explain why task pairs exhibit strong change propagation. Further analysis showed no change propagation for 93% of all non-linked task pairs (true negatives). We discover six motifs that enable the classification of the remaining 7% false positives as either true positives (i.e., task pairs that should have been linked) or true negatives (i.e., task pairs with irrelevant change propagation). These insights are vital for designing recommendation mechanisms that utilize change propagation, for example, to trigger change notifications between tasks, respectively, identify relevant tasks.

The remainder of this paper is structured as follows: We refine our general research hypothesis and present our study design in Section 2. We analyse the Mylyn data set quantitatively in Section 3. Section 4 and 5 detail the manual inspection of sample linked and non-linked task pairs, respectively. We interpret our findings and implications in Section 6. We discuss related work in Section 7 before concluding this paper with an outlook on future work in Section 8.

1.1 Motivating Example

We motivate the need to investigate change propagation between task pairs using an actual example task subset from the open source Mylyn project. Mylyn [**?**] allows a developer to connect to a task management tool (such as Bugzilla) for selecting tasks to work on and captures all developer read and write events within the Eclipse IDE. The tasks in our example address different mechanisms for creating a new Mylyn task. Figure 1 depicts the links among tasks as of Nov. 14, 2007. The central Task 169426 has links to five tasks. The greyed out Task 210022 has not been set up yet. Task 209892 (bold) was just created and thus no progress has been made yet.

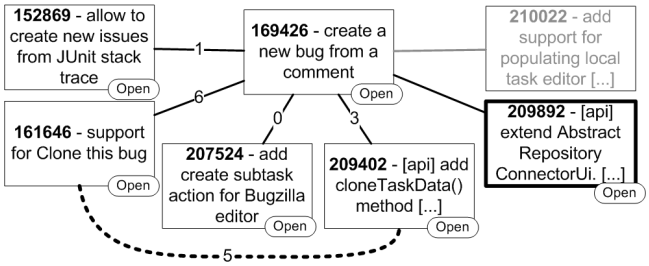

Figure 1: Example excerpt of linked task pairs. Full lines depict manually set links. The dotted line displays change propagation among non-linked task pairs. Line labels report the number of propagated, changed artifacts.

All tasks are in status "open". As the developer S.P. assigned to Task 209892 commences work, he needs to know where to look for artifacts and their (recent) changes relevant to the realization of his task. Likewise, the developers currently working on the other open tasks need to assess who they should work with and perhaps notify about changes.

Without a support tool, developers need to maintain an up-to-date view on what is going on in each linked task, a very tedious, time-consuming, and error-prone process as small details are easily missed and links may be inaccurate or incomplete. In the month prior to Nov. 14, 2007, there are 59 developers accessing ~1300 artifacts in 164 tasks. Alternatively, developers may choose to observe only the directly linked task pairs and miss important developments in other tasks. Access to change propagation information—such as displayed in Figure 1—may serve as indicator what tasks are relevant. S.P. may deduce from the change propagation values that Task 161646 shares artifacts not only with the central task, but also with Task 209402 and subsequently monitors primarily these two for changes. Yet, it is currently unclear to what extent links between tasks describe implicit task dependencies relevant to change propagation and, hence, whether one could reliably exploit them to determine where changes should be propagated to, respectively who to notify about which particular change. As we mentioned above, no study has investigated the correlation of change propagation and task links.

Understanding how change propagation occurs between linked task pairs is also important for supporting software evolution activities. Suppose we now encounter a task for fixing a bug related to incomplete cloning of data in the Mylyn task editor. The responsible developer may identify Task 209402 perhaps through keyword search, referral in the task's comments by another developer, or vaguely remembering that it once concerned data cloning. No matter how, she then needs to understand if that Task 209402 covers the problem, or if the linked Task 169426 is relevant also, or if any of the other indirectly linked tasks need inspection. Applying past change propagation to better classify the links among the tasks assists the developer to quickly narrow down the relevant task locations. Again, this requires an informed understanding how links among tasks coincide with change propagation.

2 STUDY DESIGN

2.1 Research Questions

This paper aims to answer the following research questions:

RQ1: *To what extent does change propagation occur between linked task pairs?* This question focuses on whether artifacts that are subject to a change in one task are eventually accessed in the linked task and vice versa. We analyse what proportion of changed artifacts are accessed and how much linked task pairs differ from non-linked task pairs.

Answering this question give us insights into whether linked task pairs represent a change propagation dependency, respectively whether change propagation indicates a need for coordination (i.e., creating a link between two tasks). If there is no (consistent) change propagation among linked task pairs, then developers cannot reliably use links for deciding who to involve in a task or notify about changes.

RQ2: *What are the reasons for change propagation to occur between two tasks?* This question extends the previous question and investigates under which conditions are linked task pairs showing change propagation (true positives) and when is there no change propagation occurring (false negatives). Additionally this questions studies the cause for non-linked task pairs to exhibit change propagation (false positives).

By identifying the cause for change propagation, we are able to determine more precisely when propagating changed artifacts among linked task pairs is relevant and when not—highly important for providing effective change notifications. Identifying the reasons also allows us to identify dependent but non-linked task pairs—this is potentially indicating a missing link.

2.2 Data Gathering Method

The Mylyn project[2] uses the Eclipse Bugzilla bug tracker[3] for managing task dependencies. Developers working on Mylyn attach the captured read and write events (i.e., the interaction data) to the tasks they are responsible for. Hence, for this project, we know what task a given developer was working on and what artifacts he or she was looking at or modified. In this paper we are interested in changes at the file level.

We extracted 410 tasks with attached interaction data and at least one Bugzilla *blocks*/*depends_on* link[4] - referred to as the *base set*. Those 226 tasks that have a link to another task in the *base set* then form the *linked set*. We end up with a total of 160 links in the *linked set*.

The supporting online material [**?**] provides (i) a more detailed description of the data gathering process, (ii) the set of tasks and attachments considered, (iii) the source code for collecting, filtering, and analysing the data, as well as (vi) the aggregated data underlying all figures and tables in this paper.

2.3 Temporal Data Processing

The temporal order of interaction events is important to accurately determine which changes in one task could have been accessed later—potentially by the same developer—in the linked task. Figure 2 visualizes this procedure.

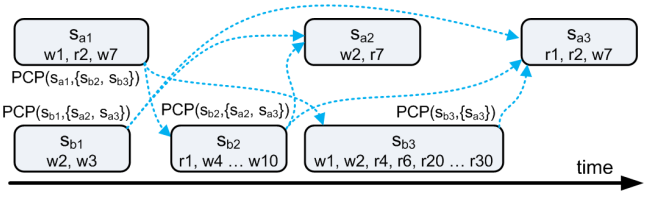

Figure 2: Example for determining change propagation. The resulting artifact sets and metrics are: $W(a) = \{1, 2, 7\}$, $W(b) = \{1...10\}$, $R(a) = \{\}$, $R(b) = \{20...30\}$, $CP_{a,b} = \{1, 7\}$, $CP_{b,a} = \{1, 2, 7\}$, $ROCP(a, b) = 2/3$, $ROCP(b, a) = 3/3$, $RACP(a, b) = 2/21$, $RACP(b, a) = 3/3$, $BiRCP(a||b) = 1.76$

We group interaction events into interaction sessions as developers tend to commit changes when they completed part of a task rather than instantly. Figure 2 depicts example task a's three sessions (s_{a1}, s_{a2}, s_{a3}) on the top and task b's three sessions on the bottom. Session duration is represented by horizontal size, with the read-only (r) and written (w) artifacts identified by number, e.g., in session $a1$ a developer changes artifacts 1 and 7 and reads artifact 2. By default, each interaction data attachment becomes one session. We split long lasting sessions whenever two events are more than 1.5 days apart.[5]

We make the assumption that a change (as recorded by a write event) in one interaction session potentially propagates to all subsequent sessions in the linked task. The dashed lines in Figure 2 highlight the potential change propagation direction. The *PCP* set contains all artifacts that a developer changes in one session of task a and which subsequently a developer accesses (i.e., read or write event) in any subsequent session of task b. For example, a change to artifact 1 in session $a1$ is read by a developer in session $b2$ who subsequently changes the artifact in $b3$. There is no propagation across parallel or partially overlapping sessions. Changes to artifact 2 in session $a2$, for example, don't propagate to task b as session $b3$ happens at the same time. The complete set of artifacts changed in task a and propagated to task b ($CP_{a,b}$) is the union of all $\bigcup_{i=1}^{n} PCP(s_i^{t_a}, S'_b)$. This guarantees that we count a propagated artifact only once, even when it is accessed in multiple sessions. Table 1 summarizes our formalization of change propagation.

The absolute number of propagated artifacts ($|CP(a, b)|$) tends to overestimate the importance of change propagation between tasks where the developer accesses a large number of artifacts (e.g., task b in Figure 2). The more artifacts are changed within a task, the more likely these are accessed later. Inversely, the more artifacts a developer accesses, the more likely these artifacts were previously changed in the linked task. The resulting high number of propagated changes as measured by absolute change propagation ($|CP_{a,b}|$), however, might be coincidental to the dependency among the two tasks. On the other hand, a developer working on a small task might be interested only in a subset of all changed artifacts (e.g.,

[2]https://www.eclipse.org/mylyn/
[3]https://bugs.eclipse.org/bugs/query.cgi
[4]The Bugzilla *duplicates* link type is irrelevant for this paper.

[5]We opted for this large window as open source software developers often work in their free time and may spread work across several days to complete a task.

Symbol	Description
$s_i^j \in S_j$	interaction session i of task j.
R_t	set of unique artifacts that were read-only in task t.
W_t	set of unique artifacts that were changed in task t, together $R_t + W_t$ make up the set of unique artifacts accessed in the scope of task t.
$PCP(s_j^a, S'_b)$	a potential change propagation (PCP) set contains all artifacts that were changed in session s_j of task a, and which subsequently were accessed in any later session of task b, where $s_j \in S_a$, $S'_b \subseteq S_b$, and $s_j.end < s'.start\ \forall s' \in S'_b$.
$CP_{a,b}$	the set of unique artifacts that were changed in task a and accessed in task b: $CP_{a,b} = \bigcup_{i=1}^{n} PCP(s_i^a, S'_b)$
$\lvert CP_{a,b} \rvert$	absolute Change Propagation from task a to task b
$BiCP(a\|b)$	Bi-directional Absolute Change Propagation: $BiCP(a\|b) = CP_{a,b} + CP_{b,a}$
$ROCP(a, b)$	Relative Observed Change Propagation: $ROCP(a, b) = \lvert CP_{a,b} \rvert / \lvert W_a \rvert$
$RACP(a, b)$	Relative Attainable Change Propagation: $RACP(a, b) = \lvert CP_{a,b} \rvert / \lvert R_b + W_b \rvert$
$RCP(a, b)$	Relative Change Propagation: $RCP(a, b) = (ROCP(a, b) + RACP(a, b))/2$
$BiRCP(a\|b)$	Bi-directional Relative Change Propagation between task a and task b: $BiRCP(a\|b) = RCP(a, b) + RCP(b, a)$

Table 1: Symbols used for describing change propagation.

task a in Figure 2). This results in low absolute change propagation and risks underestimating the relevance of the propagated changes.

We, therefore, introduce two metrics that mitigate the limitations of absolute change propagation. Specifically, we introduce the *Relative Observed Change Propagation* metric ($ROCP(a, b)$) defined as the ratio of propagated artifacts ($CP_{a,b}$) to all changed artifacts in task a ($ROCP(a, b) = |CP_{a,b}|/|W_a|$). A value of 1 indicates that the developer responsible for task b accessed all artifact changed in task a, whereas a value of 0 implies that the developer accessed none of the changed artifacts. Thus, *Relative Observed Change Propagation* ensures we consider the propagated changes only as relevant when the developers of task b accesses close to all changed artifacts, but not necessarily when accessing a lot of previously changed artifacts.

The *Relative Attainable Change Propagation* metric ($RACP(a, b)$) is defined as the ratio of propagated artifacts ($CP_{a,b}$) to all accessed artifacts in task b ($RACP(a, b) = |CP_{a,b}|/|R_b + W_b|$). A value of 1 signifies that all the artifacts a developer accesses in task b have been changed in task a before, whereas a value of 0 implies that the developer accessed only artifacts that have not been changed in task a before. Thus *Relative Attainable Change Propagation* ensures that we consider even a small set of propagated changes as relevant when a developer accesses a few artifacts, most of which have been changed in the linked task before.

We combine $ROCP(a, b)$ and $RACP(a, b)$ to obtain the *Relative Change Propagation* $RCP(a, b)$ from task a to task b: $RCP(a, b) = (ROCP(a, b) + RACP(a, b))/2$. Ultimately, the sum of $RCP(a, b)$ and $RCP(b, a)$ produces the *Bi-directional Relative Change Propagation* ($BiRCP(a\|b)$) between task a and task b. A value of 0 indicates that absolutely no change propagation occurred in either direction. A value of 2 indicates that developers accessed and changed exactly the same set of artifacts in two concurrent tasks (a very rare case).

Examples in Section 4 show that *BiRCP* is better suited for determining whether two non-linked tasks are dependent than bi-directional absolute change propagation (*BiCP*).

3 QUANTITATIVE ANALYSIS

In this section, we quantitatively analyse the Mylyn data set to provide an answer to RQ1. We calculate *BiRCP* for all linked task pairs in the *linked set*. The histogram of these *BiRCP* values (see Figure 3 (light grey), Table 2 left) shows that 58 linked task pairs (~36%) don't exhibit any change propagation. The other 102 task pairs cover the change propagation spectrum up to 1. On average linked task pairs show 21% *BiRCP* (σ = 22%, median=16%) and 1.81 *BiCP* (σ = 2.36, median=1).

We also inspect the change propagation among non-linked task pairs from the same data set in order to confirm that the extent of change propagation can be attributed to the links among tasks and not to a general feature of the data set. To this end, we calculate the same change propagation metrics (see Table 1) for all task pairs in the *base set* that are more than two link-hops apart - denoted the *non-linked set*. I.e., we exclude task pairs that have links to a common third task as we expect several of these pairs to yield similarly high change propagation as directly linked task pairs. We report the resulting *BiRCP* values in Table 2 (right) and Figure 3 (dark grey). Around 93% of the 82,129 pairs in the *non-linked set* exhibit no change propagation. On average non-linked task pairs exhibit 1.14% *BiRCP* (σ = 5.55%, median=0%) and 0.12 *BiCP* (σ = 0.66, median=0).

Comparing *BiRCP* average, standard deviation, median, and the histogram distribution in Figure 3, we conclude that change propagation is a feature of the *linked set*. We, however, find two orders of magnitude more task pairs in the *non-linked set* with non-zero change propagation (5897) compared to the *linked set* (102 pairs with $BiRCP > 0$).

We draw the following preliminary conclusions: First, based on the quantitative analysis alone, linked task pairs imply an underlying change propagation dependency (~64% true positives). Hence, a recommendation algorithm may utilize the links between two tasks as an indicator that developers in one task, for example, should be notified about changes in the linked task. Yet, from the distribution of *BiRCP* values we learn that there is a significant number of task pairs where no change propagation occurs (~36% false negatives) and thus recommendations would be irrelevant. We subsequently use the underlying quantitative data to sample task pairs for qualitatively investigating the reasons for high, respectively, low *BiRCP* in Section 4.

Second, the large number of task pairs with non-zero change propagation in the *non-linked set* (~7% false positives) entail that randomly selecting two tasks from the *base set* and detecting change propagation provides no indication whether the two tasks might indeed be dependent. In other words, the data imply that a recommendation algorithm suggesting related tasks based on change propagation alone will very likely produce a list of tasks that are not truly relevant. Reducing the number of false positives is pertinent. We, thus, use the underlying quantitative data to sample task pairs

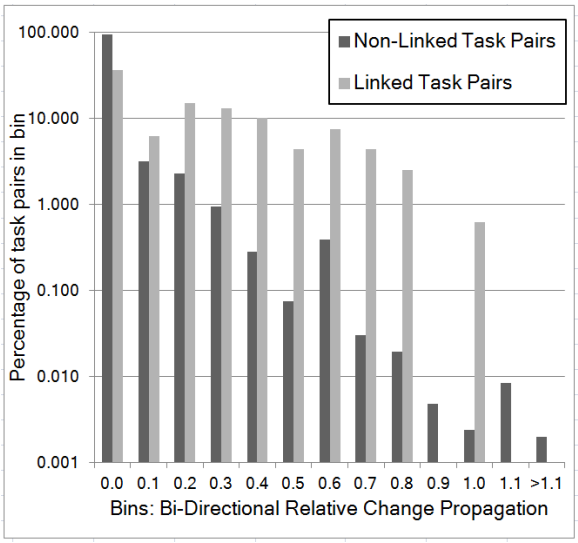

Figure 3: Histogram of the relative amount of task pairs from the *linked set* (light grey) and the *non-linked set* (dark grey) as determined by *BiRCP* bins. Note that the percentage on the log-scale y-axis is given to 3 decimal places.

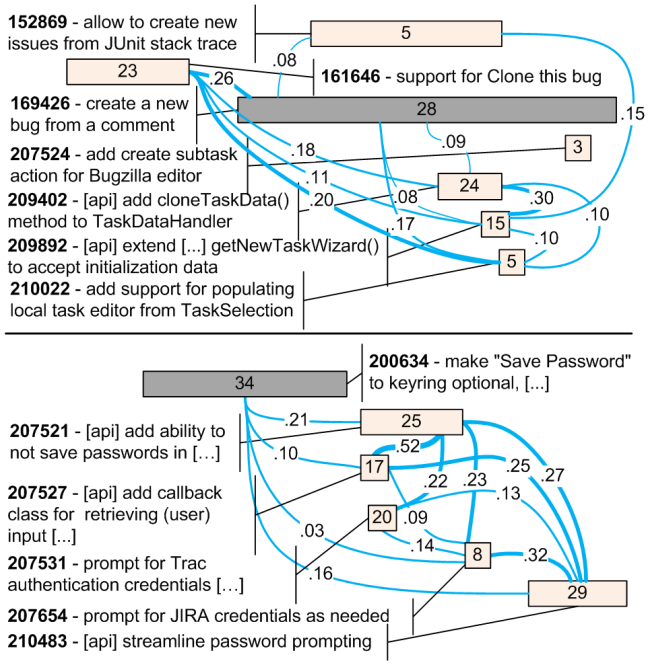

Figure 4: Artifact change propagation graphs for two selected cases: tasks are represented by boxes that report the number of changed artifacts, their size indicates work duration, time flowing from left to right. The dark shaded box identifies the central task that all other tasks link to. Lines report *BiRCP*($a \| b$).

for qualitatively investigating the reasons why some non-linked task pairs exhibit high change propagation (see Section 5).

4 QUALITATIVE ANALYSIS OF LINKED TASK PAIRS

In this section we aim to answer RQ2 with respect to determining the reasons for high and low change propagation among linked task pairs using methods from Grounded Theory [**?**]. First, we sampled 17 task pairs from the 160 links in the *linked set* (see Table 3). We applied following sampling criteria: select a combination of zero and non-zero BiRCPwith small as well as large write sets (W_t). For each task pair, we manually inspected the task details, task description changes, and comments on the Eclipse Bugzilla website.[6] We captured all information pertinent to work coordination among tasks on virtual cards in an open coding process [**?**]. Upon completing the processing on all samples, we iterated through the cards and retained those that represented common coordination concerns. Based on these remaining cards, we aimed to establish the purpose of the link between two tasks, and thus the reason for low or high change propagation.

The Tasks 169426 (Sample 3 and 6) and 200634 (Sample 10) are part of a larger graph of linked tasks for which we have interaction data. In a second round, we included all tasks in these two graphs in our analysis (increasing the manually analysed task pairs to a total of 25, i.e., ~16% of all links in the *linked set*). In total, we inspected 41 out of the 226 tasks in the *linked set*. Page restrictions limit us to a brief introduction of the two connected task graphs. We subsequently map the 25 task pairs to the identified coordination concerns (see also right most column in Table 3). When referring to a sample task pair, we indicate the task's location in Table 3 with a and b, respectively. We provide exemplary excerpts from task comments to support our conclusions.

[6]https://bugs.eclipse.org/bugs/show_bug.cgi?id=TASKID

4.1 Connected Sample Graphs

We introduced **TaskGraph 1** in the motivating scenario (Sec. 1.1). The upper three tasks represent desirable features that have been sitting dormant for several months. Work on Task 161646 becomes the basis for the development in the central Task 169426. Figure 4 (top) visualizes how the work in the remaining tasks occurs in parallel to the central task. During development, subtasks 209402 and 209892 emerge that aim to provide a reusable interface for the other tasks to program against. Task 207524 represents another related feature, limited to a subcomponent (i.e., the Bugzilla editor). Task 210022 continues the work of Task 152869 by fixing a bug introduced there.

TaskGraph 2 addresses the ability to obtain authentication details from the user. Similar to TaskGraph 1, significant up-front work is done in Task 200634 which serves as basis for refinement and extension (see Fig. 4 bottom). In contrast to TaskGraph 1, however, here the up-front work task becomes the central task, and all remaining linked tasks represent various subtasks. The first two subtasks (207521 and 207527) concentrate on the two distinct efforts to avoid storing a password and the ability to retrieve the password on demand from the user, respectively. The next two subtasks (207531 and 207654) integrate the capabilities of the prior

Bin	Linked		Non-Linked	
	Abs.	Rel.	Abs.	Rel.
0.0	58	36.3%	76,231	92.819%
0.1	10	6.3%	2,589	3.152%
0.2	24	15.0%	1,864	2.270%
0.3	21	13.1%	776	0.945%
0.4	16	10.0%	232	0.282%
0.5	7	4.4%	62	0.075%
0.6	12	7.5%	318	0.387%
0.7	7	4.4%	25	0.030%
0.8	4	2.5%	16	0.019%
0.9			4	0.005%
1.0	1	0.6%	2	0.002%
1.1			7	0.009%
>1.1			2	0.002%

Table 2: Absolute and relative number of task pairs from the *linked set* (left) and the *non-linked set* (right) per *BiRCP* bin.

Nr	$Task_a$	$Task_b$	$\|R_a\|$	$\|W_a\|$	$\|R_b\|$	$\|W_b\|$	$\|CP_{a,b}\|$	$\|CP_{b,a}\|$	$\|BiCP(a\|\|b)\|$	$BiRCP(a\|\|b)$	Sit.
1	208629	179254	1	2	35	41	1	2	3	0.61	S7
2	222833	226822	48	28	20	3	0	0	0	0.00	S1
3	161646	169426	665	14	38	25	6	0	6	0.26	S4
4	191793	202547	24	15	121	2	0	0	0	0.00	S2
5	244653	242445	43	15	33	4	1	0	1	0.05	S3
6	209402	169426	629	13	38	25	1	2	3	0.09	S2
7	244359	238038	35	0	59	12	0	4	4	0.22	S6
8	393640	386344	2	3	154	71	0	3	3	0.32	S6
9	201464	196700	6	4	1467	33	0	7	7	0.46	S4
10	207531	200634	21	15	51	28	0	0	0	0.00	S2
11	229014	176212	45	0	1	2	0	2	2	0.52	S7
12	262107	261683	3	0	1	5	0	2	2	0.53	S7
13	217694	205861	2	5	1	2	1	2	3	0.91	S6
14	210170	189313	0	1	1	2	0	1	1	0.75	S7
15	303431	199345	1	3	71	67	2	3	5	0.74	S6
16	231336	216150	22	15	21	21	8	8	16	0.66	S5
17	277191	272621	4	1	59	16	1	1	2	0.64	S6

Table 3: Sampled linked task pairs selected for qualitative analysis. The *Sit.* column lists the respective situations S1 to S7; for metric definitions, see Table 1.

two task in two different connectors to third party systems (Trac and JIRA). Work in the last task (210483) occurs several weeks after the previous tasks have been completed and undertakes rework of existing artifacts to improve usability.

4.2 Situations leading to low BiRCP

We found three main situations that lead to low Bi-directional Relative Change Propagation.

S1: Task Decomposition One task serves as parent task for multiple child tasks which refine the work described in the parent task. When the parent task is predominantly used for coordinating work and subsequently involves little coding effort and thus few changes, then there is little opportunity for change propagation.

For example, Task 226822 (*b*) in Sample 2 serves as central parent task coordinating the refactoring of attachment data handling. The linked Task 222833 is one among a few child tasks realizing the refactoring. The task pair exhibits low *BiRCP* as the parent task includes only a few, intermittent changes. See, e.g., one comment in Task 226822:

```
Remaining work:
- set defaults for the task editor or working copy
- replace TaskSelection with new API
```

S2: Separation of concerns applies to task pairs that conceptually belong together (hence the task links) but address different aspects of the same concept and thus—regardless of the number of changed artifacts—share few changed artifacts as work in the respective tasks tends to affect different artifact sets.

We find such situations in TaskGraph 1—Task 207524, 209402 (*a*) (i.e., Sample 6), and 209892 all linked to Task 169426 (*b*)—and in TaskGraph2; here in Sample 10 Task 207531(*a*) links to Task 200634(*b*). In Sample 4 both tasks are related refactorings postponed to a different release. The Tasks 191793 (*a*) focuses on code refactoring, the linked Task 202547 (*b*) focuses on feature restructuring of non-code artifacts.

S3: Synchronization aims at coordinating work among task pairs that are likely to change the same artifact and cause merge conflicts. Typically only a few central artifacts are subject to a potential write conflict and thus *BiRCP* remains low.

We find such a case in Sample 5: Work in both tasks involves changes to a central file (the only artifact changed by both tasks). The link serves as locking mechanism to avoid a conflict: i.e., first completing Task 244653 (*a*) and subsequently the linked Task 242445 (*b*). Note following excerpt from the comments of Task 242445:

```
Helen, it is probably best to wait until the patch on bug 244653 is
merged before starting on this to avoid conflicts.
```

4.3 Situations leading to high BiRCP

During our analysis we identified four situations that give rise to high *BiRCP*.

S4: Work Continuation occurs when developers in one task explicitly hand over development effort to another task. Work in the continuing task is then likely to change the same artifacts and hence lead to high BiRCP.

In Sample 9 we find developers in Task 196700 (*b*) implementing a new feature. This feature contained a bug that was subsequently corrected in the linked Task 201464 (*a*). The former task thus continued and completed the work of the latter task, hence the large *BiRCP*. See following comment in Task 196700:

```
Warning! There appears to be some corruption [...] there is another
bug to track this issue: bug 201464. I would avoid using the new task
editor feature on any kind of production xplanner server until that
bug is fixed.
```

In TaskGraph 1, work on Task 169426 (*b*) and 209892 continues on code from Task 161646 (*a*): in the comments for Task 169426 we find:

`Thanks for the patch Frank. It would be great if you could make the implementation more generic so it could be reused for bug 161646 which overlaps significantly with this bug.`

In TaskGraph 2, Task 210483 continues the work of streamlining the artifacts of preceding tasks 207521, 207527, 207531, and 207654.

S5: Artifact Reuse resembles S4: Work Continuation but differs in the explicit focus on making use of the output of a linked task, rather than continuing the work.

In TaskGraph 1, work on Task 161646 is explicitly postponed to continue after the output of linked Task 209402 is available:

`We're holding off on this until we can make use of the api that emerges from bug#209402.`

In TaskGraph 2, developers in Tasks 207521 and 207527 reuse code of the parent Task 200634 as the comment in Task 200634 highlights:

`[...] Thanks, I have used a mix of your patches as the base for my modifications.`

In Sample 16, Task 231336 (*a*) fixes a sorting problem and provides code to be used in the linked Task 216150 (*b*) that is equally concerned with sorting. A comment excerpt in Task 231336 states:

`[...] for bug#216150 the compare part is now in class TaskComparator [...]`

S6: Emerging Task Decomposition results in several tasks becoming the children of the common, central linked task (similar to S1). However, here, significant up-front work is done in the central task and the child tasks are created one by one as needed.

In Sample 7, a significant amount of work occurred in Task 238038 (*b*) (about providing time range based folding for comments) when the linked Task 244359 (*a*) was specified as a subtask to implement a sub-aspect of the parent (implement grouping strategy for task comments). Hence, the *BiRCP* among linked task pairs is high. The central task exhibits further links to other subtasks that were created step by step as the work progressed. In the emerging child Task 244359 we find following comment:

`Extract the implementation for grouping of task comments discussed on bug 238038.`

In Sample 8 the two tasks address support for test integration with third party systems. In Task 386344 (*b*), the work is done for one third party system (here Trac). In the linked Task 393640 (*a*), the work is then replicated for another third party system which explains the high *BiRCP*. In Sample 13, Task 217694 (*a*) addresses which icon to use inside a tooltip while the linked Task 205861 (*b*) coordinates several tasks on improving tooltip presentation. Similar, in Sample 15, Task 303431 (*a*) is about colors, the linked Task 199345 (*b*) ties together tasks about configuring labels. Finally, Sample 17 has Task 277191 (*a*) focusing on UI issues in the connector discovery dialog and the linked Task 272621 (*b*) coordinating the Mylyn connector discovery mechanism.

Emerging Task Decomposition is closely related to S5: Artifact Reuse. If we restructure TaskGraph 1 to set Task 161646 as the parent task, and have Tasks 169426, 209402, and 209892 as child tasks, then these three child tasks would constitute examples of emerging task composition rather than simply artifact reuse.

S7: Small Bug Fixes involve corrections to a handful of artifacts that are changed in both tasks resulting in high *BiRCP*. In Sample 11, both tasks are related bugs about hyperlink processing which were independently rectifiable. Task 176212 (*b*) constituted minor corrections to only two artifacts which were subsequently changed again during work on the linked Task 229014 (*a*). Similar, the tasks in Sample 12 are two related bugs, here, one about the size, the other about the position of a user interface element. The single changed artifact in the Task 261683 (*b*) was changed again in the linked Task 262107 (*a*). Also Sample 1 fits this situation. Task 208629 (*a*) fixes a small bug, hence only a few changes, which largely overlap with the refactoring done in the linked Task 179254 (*b*). Sample 9 could also be classified as S7: Small Bug Fix, but comments and the link to a feature development task better places it with S4: Work Continuation. Finally, in Sample 14, Task 210170 (*a*) fixes tooltip visibility, relevant for improving tooltip positioning in the scope of the linked Task 189313 (*b*).

5 QUALITATIVE ANALYSIS OF NON-LINKED TASK PAIRS

In this section we complete our answer to RQ2. In Section 3, we found that a significant amount of non-linked task pairs exhibit high *BiCP* and/or high *BiRCP*. Here we manually inspect sample non-linked task pairs to determine whether high *BiRCP* is indicative of missing links or whether we simply detect a lot of false positives. To this end, we sampled 22 non-linked task pairs from the 5897 links in the *non-linked set* that have non-zero *BiRCP*, a total of 32 tasks. We applied following sampling criteria: select a combination of small as well as large write sets (W_t), a range of 1 to highest observed *BiCP*, and a range of 0.1 to 1.5 *BiRCP*. We followed the same coding procedure of the qualitative analysis of linked task pairs (see Section 4) with a focus on the set of propagated artifacts ($CP_{a,b}$), any available links to other tasks, and comments. We cannot introduce the tasks individually due to page limits.

We identified six motifs that describe when task pairs exhibit high *BiCP* and/or high *BiRCP* and which explain when the these metrics uncover a true (albeit implicit) dependency among the tasks.

M1: Task Cluster Membership We find several cases where the tasks address the same implementation topic, e.g., hyperlink issues in Sample I2 and I14 or connector discovery in samples I4 and I15. Typically these tasks have links to other tasks that are directly linked. Recall that we excluded two-hop related task pairs in the quantitative analysis. Figures 5 and 6 (left) depict how these samples are three hops (I4, I15) or more (I2, I14) apart but still part of a cluster of related tasks.

M2: Support Cluster Membership Similar to M1, we find several tasks that focus on implementing and/or testing a particular feature or subsystem. Figure 6 (right) visualizes how change propagation among the tasks in Samples I9, I11, I16, and I18 ties together tasks on supporting a new version of Bugzilla that otherwise are not explicitly linked. Similar, Sample I6 brings together two dependent tasks: one fixing a new feature, the other provisioning the respective testing infrastructure.

M3: Mutual Access of Utility Artifacts Task pairs exhibit high *BiRCP* when they primarily read or update commonly used utility artifacts. All tasks in Samples I8 and I17 require UI features to be configurable. While the tasks focus on different UI elements (here labels, line highlighting, and search characters), they all needed to make changes to the same set of UI and preference-centric artifacts.

Nr	$Task_a$	$Task_b$	$\|R_a\|$	$\|W_a\|$	$\|R_b\|$	$\|W_b\|$	$\|CP_{a,b}\|$	$\|CP_{b,a}\|$	$\|BiCP(a\|\|b)\|$	$BiRCP(a\|\|b)$	M.
I1	208629	220688	1	2	0	1	1	1	2	1.42	M5
I2	164221	176212	2	4	1	2	2	2	4	1.25	M1
I3	206568	216677	7	10	0	1	1	1	2	1.08	M5
I4	276942	277910	3	3	1	1	1	1	2	1.00	M1
I5	167941	208629	16	19	1	2	3	1	4	0.84	M5
I6	201464	196523	6	4	2	2	3	0	3	0.75	M2
I7	219911	175922	2	4	52	17	3	3	6	0.73	M4
I8	199345	299697	71	67	6	6	5	5	10	0.68	M3
I9	252297	256045	5	9	4	3	5	0	5	0.63	M2
I10	200634	196056	51	28	7	16	1	11	12	0.45	M4
I11	226851	242480	34	28	51	52	14	3	17	0.37	M2
I12	143011	160389	52	57	150	101	1	19	20	0.19	M6
I13	220688	216677	0	1	0	1	1	0	1	1.00	M5
I14	244442	164221	10	3	2	4	2	2	4	0.83	M1
I15	276942	278331	3	3	1	3	1	2	3	0.79	M1
I16	242480	254695	51	52	0	2	1	2	3	0.77	M2
I17	199345	318045	71	67	9	4	6	2	8	0.53	M3
I18	252297	242480	5	9	51	52	8	1	9	0.53	M2
I19	272207	290465	45	61	17	30	4	13	17	0.35	M6
I20	200634	160389	51	28	150	101	0	15	15	0.17	M6
I21	238038	236690	59	12	124	55	0	14	14	0.23	M6
I22	224780	175922	183	27	52	17	0	12	12	0.38	M6

Table 4: Sample non-linked task pairs selected for qualitative analysis. The *M.* column lists the respective motifs M1 to M6; for metric definitions, see Table 1.

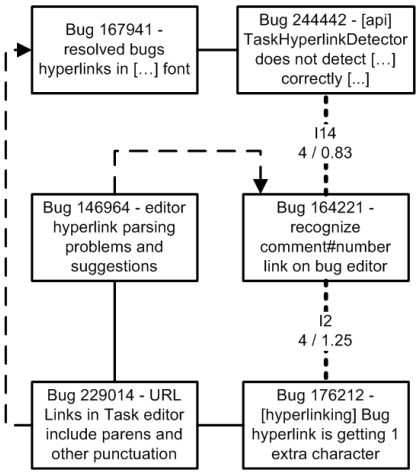

Figure 5: Examples of Task Cluster Membership for sample non-linked task pairs. Dotted lines report *BiPC* and *BiRCP*, full lines depict links among tasks, dashed lines depict task references in the comments.

M4: Orthogonal Concerns Tasks tend to access the same artifacts when these encode overlapping topics. Task *a* in Sample I7 focuses on setting certain properties when moving a bug to another product; task *b* focuses on supporting custom fields in Bugzilla. The propagated artifacts implement Bugzilla data handling functionality. Similar, both tasks in Sample I10 are related to connecting to Bugzilla for synchronizing data: task *a* addresses password acquisition, task *b* addresses data fetching.

M5: Core Artifact Access In several samples, one or both tasks access only a very small set of one to three common artifacts. This results in high *BiRCP* but no real underlying task dependency when these artifacts are at the core of the system under development and thus subject to many unrelated changes. All tasks in Samples I1, I3, I5, and I13, propagate changes of the same core artifact (`AbstractRepositoryTaskEditor`). The tasks, however, are otherwise completely unrelated.

M6: Grand Tasks We inspected task pairs with high *BiCP* to check whether absolute instead of relative change propagation is a suitable indicator for dependent tasks. Only tasks accessing many artifacts are able to yield high *BiCP* and subsequently are less likely to also yield high *BiRCP* as Table 4 shows. Task pairs with $BiCP > 11$) tend to exhibit comparatively lower *BiRCP* (≤ 0.45) and vice versa. All tasks in Samples I12, I19, I20, I21, and I22 access a high number of artifacts ($|R_t + W_t|$ ranging from ~20 to ~250). In these samples, one task is typically a refactoring effort while the other task is an unrelated, large feature implementation or bug fix.

6 DISCUSSION

Manual inspection of linked task pairs revealed seven reoccurring situations. These explain why Bi-directional Relative Change Propagation coincides only with a subset of linked task pairs. Specifically, we find medium to high *BiRCP* (i.e., >0.2) only when links represent an underlying artifact-centric task dependency. We infer that developers use links to manage control flow, data flow, task decomposition, and simultaneity dependencies [**?**]. These situations are highly relevant for the design of development coordination support tools and thus have significant potential impact on software engineering practise. Qualitative analysis shows evidence that high *BiRCP* is able to identify implicit dependencies among task pairs when additional information on artifacts (e.g., core or utility) and tasks (e.g., refactoring) is considered. Absolute bi-directional change propagation is less well suited as it identifies mostly tasks involving many artifact changes.

Situations explain why a large proportion of links show no change propagation. Yet, a lack of change propagation doesn't necessarily mean that developers don't need artifact-centric coordination. In situation S1: Task Decomposition or S2: Separation of Concerns, for example, the change propagation between in-directly linked task pairs, e.g., among child tasks of a common parent task, identifies task pairs where developers would benefit from change notification. The motivating scenario (Sec. 1.1) describes such a situation. Inversely, situations leading to high change propagation might not require detailed change propagation. In S4: Work Continuation, we only expect work to happen in the linked follow-up task once work handover occurred. Hence, developers assigned to the initial task would not need to know about changes in the follow-up

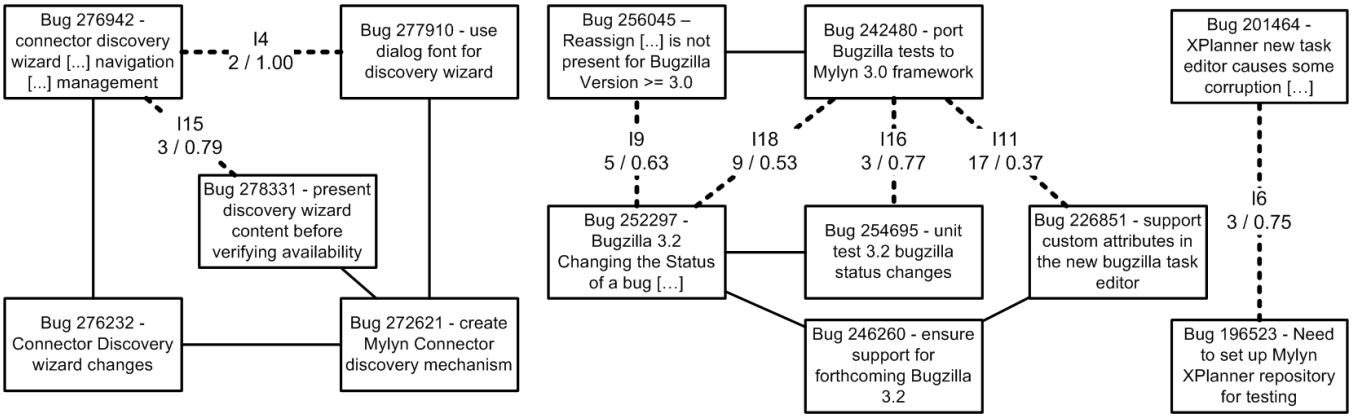

Figure 6: Examples of Task Cluster Membership (left) and Support Cluster Membership (middle, right) for sample non-linked task pairs (dotted lines, reporting *BiPC* and *BiRCP*). Full lines depict links among tasks.

task. Rather, they should receive a warning when they are about to make any changes as developers of the follow-up task are not expecting changes. Links in situations S5, S6, and S7 indicate that developers in either linked task are interested in an overlapping set of artifacts and hence would benefit from change-propagation centric coordination support.

The motifs found among the non-linked task pairs with high *BiRCP* suggest that developers would benefit from coordination support beyond directly linked task pairs. M1: Task Cluster Membership demonstrates that change propagation and multi-hop paths across existing links identify dependent task pairs that, for example, serve as input to a change notification mechanism. Even in the absence of multi-hop paths, change propagation identifies dependent tasks in M2: Support Cluster Membership and M3: Mutual Access of Utility Artifact. In the case of M3 identifying non-linked task pairs assists the developer in finding examples how to use a particular artifact, suggestions on what artifacts to change as well, and recovering design justifications from comments.

The motifs highlight that change propagation alone is insufficient to reliably determine dependent (but non-linked) task pairs. M5: Core Artifact Access points out the need to identify frequently changed artifacts (i.e., the core artifacts) and assign less importance to them when measuring change propagation. Doing so reduces the likelihood of detecting false positive task pairs. Similar, M6: Grand Tasks suggests to ignore non-linked task pairs that exhibit a high amount of accessed artifacts and high absolute *BiPC*. The non-linked samples classified as M6 motivate a *BiRCP* threshold of 0.4. This reduces the non-linked sample pairs under consideration for implicit dependencies from ~5800 down to 436. Note that this threshold is higher than the *BiRCP* values we find in situations with high change propagation among linked task pairs. Some task pairs in situations S4 and S6 exhibit *BiRCP* between 0.2 and 0.3. As we mentioned, the change propagation metrics alone are often not sufficient to reliably interpret a link. In the case of Samples 3 and 7, the manually inspected context determines their classification. Currently, the number of manually investigated samples is too low to infer the distribution of situations and motifs across the complete Mylyn data set and thus to reliably select a threshold.

Change propagation metrics are also useful for a-posteriori (re-) classification of links. Accurate and complete links are important during maintenance efforts as the basis for identifying relevant tasks to inspect. Correcting a bug requires identifying tasks that fixed a similar bug or similar location before (i.e., S7). Developers engaging in refactoring may benefit from knowing where work has continued from an initial task and studying the latter for extracting design rationale (S4 and S5). High change propagation among child tasks in situations S1 and S6 assist in identifying closely related tasks, separating them from tasks that implement independent concerns (S2).

6.1 Threats to validity

6.1.1 Internal Validity. We address researcher bias by analysing data from an open source system rather than conducting controlled experiments. The analysis focused on artifacts and tasks and was not specifically tailored to Java development in general or the Mylyn project in particular. The manual inspection during qualitative analysis showed no indication that the use of links was specifically adapted to the Mylyn development "process".

With respect to the data set quality, we noticed that occasionally interaction attachments appeared to be missing (e.g., a long interval between interaction data attachment and commit message in the comments). The impact on our results, however, is minimal as missing attachments occurred typically towards the end of a task that contained several other preceding attachments. We thus expect the missing attachments to contain little additional information.

6.1.2 External Validity. We analysed only a single data set as we are not aware of other real world projects aside from Mylyn that make a significant amount of interaction data and linked tasks available. Mylyn interaction data upload capabilities are not available by default and thus not widely used beyond the Mylyn project. Hence, we are careful to generalize our findings beyond the scope of the Mylyn project.

We can infer from other metrics, however, that the Mylyn project is similar to other projects. Thompson et al.[**?**] analyse the task links of three open sources projects including Mylyn and finds the

ratio of links classified as Specification or Problem similar. Zou and Godfrey [**?**] report on maintenance tasks and finds a median of 2 for edited files and the median of viewed-only files in a task is 4. We found a median of 3 and 6, respectively, for Mylyn. D'Ambros et al. [**?**] report similar commit transactions per class and change coupling metrics for Mylyn and ArgoUML. Heck and Zaidman [**?**] show that Mylyn has similar duplicate bug reports as other open source projects. All indications are that Mylin data are in fact transferable. We don't expect all situations and motifs to arise in other projects, nor do we claim completeness, but the identified dependencies are found beyond the software engineering domain [**?**].

The Mylyn dataset might not accurately represent (non-open source) development environments where other communication channels (such as direct messaging or face-to-face discussion) exist for conveying the impact of an artifact change [**?**]. This reduces a developer's need to inspect (i.e., read only) artifacts. We speculate that fewer observed read-only events subsequently result in lower levels of change propagation. Additional data sources and analysis are necessary to assess the effect of read-only events on change propagation in these environments.

Bugzilla limits link types to *duplicates* and *blocks*/ *depends_on* while other task management tools (e.g., JIRA, Redmine) provide diverse and customizable link types. The available link types, however, have no impact on the results as our analysis is independent of the link type semantics and direction. The identified seven situations reflect generally applicable coordination concerns that are common in software development and in no way specific to Bugzilla or the Mylyn project. We suspect that data from a project using JIRA or Redmine will show differences in change propagation for the various link types but similar results across all links.

7 RELATED WORK

Prior investigations of change propagation studied primarily the logical coupling between artifacts, i.e. which artifacts tend to co-evolve [**? ? ?**] and not the links among tasks. These approaches observe which artifacts frequently occur in the same commit (or in commits in temporal proximity) independent of the task these commits belong to. Zou et al. [**?**] apply interaction histories for detecting such coupling as they are more rich in information. They present a set of change patterns based on the temporal order of artifact reads and writes. Bantelay et al. [**?**] combine interaction histories and commit data from the Mylyn project to improve the detection of evolutionary coupling between artifacts. Kobayashi et al. [**?**] also use the interaction history to determine during an artifact change which other artifacts are accessed and what artifacts are changed in succession. The resulting graph is used to predict which artifacts to change next. Robbes et al. [**?**] record detailed artifact changes from IDE interactions via SpyWare [**?**] for predicting sequential changes.

All these approaches scope their analysis to artifact changes within a single (often implicit) task. The underlying data sets either lack links among tasks or have no association of events to explicit tasks. Our data set enables for the first time the study of change propagation across linked tasks.

Several tools utilize developers' interactions with the IDE to suggest relevant artifacts. NavTracks [**?**] supports navigation during software maintenance by suggesting related files based on the developers' IDE interaction path in previous navigation sessions. TeamTracks [**?**] aims to ease program comprehension visualizing navigation patterns. Configurable HeatMaps [**?**] capture how often a file was changed or visited. Mylyn [**?**] is the most prominent tool that associates observed developer interactions with tasks. It thus determines the relevant artifacts for the developer's underlying task. Similar, Hipikat [**?**] supports the developer in retrieving relevant artifacts from the project's overall history. Its considers documents, tasks, commits, messages, and artifact changes but not the detailed interaction history.

These tools offer assistance independent of tasks or focus on one single task context, i.e., the underlying conceptual models lack task links. We investigated how changes propagate across tasks.

Prior work studied work breakdown relationships based on task title [**?**] but didn't consider artifact changes to classify relations. Work on socio-technical congruence (STC) assesses team performance by investigating whether developers assigned to linked tasks also communicate and work on common artifacts. Initial work on STC [**?**] substituted co-evolving artifacts from commit data for explicit work dependencies. Valetto et al. [**?**], for example, propose mining software repositories to determine socio-technical congruence. Later approaches applied explicit tasks dependencies [**?**]. As our research has shown, developers use task links for managing diverse coordination needs, specifically that change propagation doesn't necessarily coincide with explicit links. A lack of artifact change propagation (or co-evolution), therefore, doesn't imply there is no dependency that needs managing and hence communication. Our work, thus, adds another challenge to measuring, understanding, and achieving social-technical congruence [**?**].

8 CONCLUSIONS AND OUTLOOK

We presented a quantitative and qualitative analysis of artifact change propagation in the Mylyn data set. We found seven situations describing how developers apply task links to manage implicit dependencies such as task synchronization and task continuation—not all of which are artifact-centric. This explains why change propagation occurs for only 64% of all linked task pairs. We identified additional six motifs that group non-linked task pairs according to either missing links or incidentally high change propagation.

We discussed the importance of our findings for development coordination support mechanisms. These rely on a classification of linked task pairs and non-linked task pairs. How exactly such mechanisms determine the respective situations and motifs automatically and reliably is subject to our future research. We intend to extend our qualitative analysis to all links in the Mylyn data set in order to identify complementary metrics based on graphs of directly linked tasks, artifact change frequency, and other similarity metrics (e.g., [**?**]) beyond change propagation.

ACKNOWLEDGMENTS

This work was supported by the Austrian Science Fund (FWF): P29415-NBL funded by the Government of Upper Austria.

REFERENCES

[] Fasil Bantelay, Motahareh Bahrami Zanjani, and Huzefa H. Kagdi. 2013. Comparing and Combining Evolutionary Couplings from Interactions and Commits. In *20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, 2013*, Ralf Lämmel, Rocco Oliveto, and Romain Robbes (Eds.). IEEE Computer Society, 311–320. https://doi.org/10.1109/WCRE.2013.6671306

[] Lionel C. Briand, Yvan Labiche, and L. O'Sullivan. 2003. Impact Analysis and Change Management of UML Models. In *19th International Conference on Software Maintenance (ICSM 2003), The Architecture of Existing Systems, 22-26 September 2003, Amsterdam, The Netherlands*. IEEE Computer Society, 256–265. https://doi.org/10.1109/ICSM.2003.1235428

[] Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. 2006. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work (CSCW '06)*. ACM, New York, NY, USA, 353–362. https://doi.org/10.1145/1180875.1180929

[] Davor Cubranic, Gail C. Murphy, Janice Singer, and Kellogg S. Booth. 2005. Hipikat: A Project Memory for Software Development. *IEEE Trans. Software Eng.* 31, 6 (2005), 446–465. https://doi.org/10.1109/TSE.2005.71

[] Marco D'Ambros, Michele Lanza, and Romain Robbes. 2009. On the Relationship Between Change Coupling and Software Defects. In *16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France*, Andy Zaidman, Giuliano Antoniol, and Stéphane Ducasse (Eds.). IEEE Computer Society, 135–144. https://doi.org/10.1109/WCRE.2009.19

[] Cleidson R. B. de Souza and David F. Redmiles. 2008. An Empirical Study of Software Developers' Management of Dependencies and Changes, See [**?**], 241–250. https://doi.org/10.1145/1368088.1368122

[] C. R. B. de Souza and D. F. Redmiles. 2011. The Awareness Network, To Whom Should I Display My Actions? And, Whose Actions Should I Monitor? *IEEE Transactions on Software Engineering* 37, 3 (May 2011), 325–340. https://doi.org/10.1109/TSE.2011.19

[] Robert DeLine, Mary Czerwinski, and George G. Robertson. 2005. Easing Program Comprehension by Sharing Navigation Data. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005), 21-24 September 2005, Dallas, TX, USA*. IEEE Computer Society, 241–248. https://doi.org/10.1109/VLHCC.2005.32

[] Alexander Egyed. 2011. Automatically Detecting and Tracking Inconsistencies in Software Design Models. *IEEE Trans. Software Eng.* 37, 2 (2011), 188–204. https://doi.org/10.1109/TSE.2010.38

[] Alexander Egyed, Emmanuel Letier, and Anthony Finkelstein. 2008. Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models. In *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy*. IEEE Computer Society, 99–108. https://doi.org/10.1109/ASE.2008.20

[] Harald C. Gall, Karin Hajek, and Mehdi Jazayeri. 1998. Detection of Logical Coupling Based on Product Release History. In *1998 International Conference on Software Maintenance, ICSM 1998, Bethesda, Maryland, USA, November 16-19, 1998*. IEEE Computer Society, 190–197. https://doi.org/10.1109/ICSM.1998.738508

[] Petra Heck and Andy Zaidman. 2013. An analysis of requirements evolution in open source projects: recommendations for issue trackers. In *13th International Workshop on Principles of Software Evolution, IWPSE 2013, Proceedings, August 19-20, 2013, Saint Petersburg, Russia*, Romain Robbes and Gregorio Robles (Eds.). ACM, 43–52. https://doi.org/10.1145/2501543.2501550

[] Li Jiang, Kathleen M. Carley, and Armin Eberlein. 2012. Assessing team performance from a socio-technical congruence perspective. In *2012 International Conference on Software and System Process, ICSSP 2012, Zurich, Switzerland, June 2-3, 2012*, D. Ross Jeffery, David Raffo, Ove Armbrust, and LiGuo Huang (Eds.). IEEE, 160–169. https://doi.org/10.1109/ICSSP.2012.6225961

[] Massila Kamalrudin, John C. Grundy, and John G. Hosking. 2010. Managing Consistency between Textual Requirements, Abstract Interactions and Essential Use Cases. In *Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2010, Seoul, Korea, 19-23 July 2010*, Sheikh Iqbal Ahamed, Doo-Hwan Bae, Sung Deok Cha, Carl K. Chang, Rajesh Subramanyan, Eric Wong, and Hen-I Yang (Eds.). IEEE Computer Society, 327–336. https://doi.org/10.1109/COMPSAC.2010.40

[] Mik Kersten and Gail C. Murphy. 2006. Using Task Context to Improve Programmer Productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006*, Michal Young and Premkumar T. Devanbu (Eds.). ACM, 1–11. https://doi.org/10.1145/1181775.1181777

[] Takashi Kobayashi, Nozomu Kato, and Kiyoshi Agusa. 2012. Interaction Histories Mining for Software Change Guide. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering, RSSE 2012, Zurich, Switzerland, June 4, 2012*, Walid Maalej, Martin P. Robillard, Robert J. Walker, and Thomas Zimmermann (Eds.). IEEE, 73–77. https://doi.org/10.1109/RSSE.2012.6233415

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

[] Christoph Mayr-Dorn. 2018. Supporting Online Material for ICSE2018 submission. (Jan 2018). https://doi.org/10.6084/m9.figshare.5346253.v1

[] Sebastian C. Muller and Thomas Fritz. 2013. Stakeholders' Information Needs for Artifacts and Their Dependencies in a Real World Context. In *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*. IEEE Computer Society, 290–299. https://doi.org/10.1109/ICSM.2013.40

[] Christian Nentwich, Wolfgang Emmerich, and Anthony Finkelstein. 2003. Consistency Management with Repair Actions. In *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, Lori A. Clarke, Laurie Dillon, and Walter F. Tichy (Eds.). IEEE Computer Society, 455–464. https://doi.org/10.1109/ICSE.2003.1201223

[] Romain Robbes and Michele Lanza. 2008. SpyWare: a Cchange-Aware Development Toolset, See [**?**], 847–850. https://doi.org/10.1145/1368088.1368219

[] Romain Robbes, Damien Pollet, and Michele Lanza. 2010. Replaying IDE Interactions to Evaluate and Improve Change Prediction Approaches. In *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, Proceedings*, Jim Whitehead and Thomas Zimmermann (Eds.). IEEE Computer Society, 161–170. https://doi.org/10.1109/MSR.2010.5463278

[] David Röthlisberger, Oscar Nierstrasz, Stéphane Ducasse, Damien Pollet, and Romain Robbes. 2009. Supporting Task-Oriented Navigation in IDEs with Configurable HeatMaps. In *The 17th IEEE International Conference on Program Comprehension, ICPC 2009, Vancouver, British Columbia, Canada, May 17-19, 2009*. IEEE Computer Society, 253–257. https://doi.org/10.1109/ICPC.2009.5090052

[] Anita Sarma, Jim Herbsleb, and André Van Der Hoek. 2008. Challenges in Measuring, Understanding, and Achieving Social-Technical Congruence. In *Proceedings of Socio-Technical Congruence Workshop, In Conjuction With the International Conference on Software Engineering*.

[] Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn (Eds.). 2008. *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*. ACM.

[] Janice Singer, Robert Elves, and Margaret-Anne D. Storey. 2005. NavTracks: Supporting Navigation in Software Maintenance. In *21st IEEE International Conference on Software Maintenance (ICSM 2005), 25-30 September 2005, Budapest, Hungary*. IEEE Computer Society, 325–334. https://doi.org/10.1109/ICSM.2005.66

[] Anselm Strauss and Juliet Corbin. 1990. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Sage Publications, Inc.

[] C. Albert Thompson, Gail C. Murphy, Marc Palyart, and Marko Gasparic. 2016. How Software Developers use Work Breakdown Relationships in Issue Repositories. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, Miryung Kim, Romain Robbes, and Christian Bird (Eds.). ACM, 281–285. https://doi.org/10.1145/2901739.2901779

[] Giuseppe Valetto, Mary Helander, Kate Ehrlich, Sunita Chulani, Mark Wegman, and Clay Williams. 2007. Using Software Repositories to Investigate Socio-Technical Congruence in Development Projects. In *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*. IEEE, 25–25.

[] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 461–470.

[] Annie T. T. Ying, Gail C. Murphy, Raymond T. Ng, and Mark Chu-Carroll. 2004. Predicting Source Code Changes by Mining Change History. *IEEE Trans. Software Eng.* 30, 9 (2004), 574–586. https://doi.org/10.1109/TSE.2004.52

[] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. 2005. Mining Version Histories to Guide Software Changes. *IEEE Transactions on Software Engineering* 31, 6 (2005), 429–445.

[] Lijie Zou and Michael W. Godfrey. 2006. An Industrial Case Study of Program Artifacts Viewed During Maintenance Tasks. In *13th Working Conference on Reverse Engineering (WCRE 2006), 23-27 October 2006, Benevento, Italy*. IEEE Computer Society, 71–82. https://doi.org/10.1109/WCRE.2006.12

[] Lijie Zou, Michael W. Godfrey, and Ahmed E. Hassan. 2007. Detecting Interaction Coupling from Task Interaction Histories. In *15th International Conference on Program Comprehension (ICPC 2007), June 26-29, 2007, Banff, Alberta, Canada*. IEEE Computer Society, 135–144. https://doi.org/10.1109/ICPC.2007.18