This is a post-peer-review, pre-copyedit version of this article. The final authenticated version is available online at: https://doi.org/10.1145/2501535.2501536

Uncovering Critical Coordination Requirements through Content Analysis

Kelly Blincoe
Computer Science Department
Drexel University
Philadelphia, PA, USA
kelly.blincoe@drexel.edu

Giuseppe Valetto
Computer Science Department
Drexel University
Philadelphia, PA, USA
valetto@cs.drexel.edu

Daniela Damian
Software Engineering Global Interaction Lab
University of Victoria
Victoria, BC, Canada
danielad@cs.uvic.ca

ABSTRACT
In this paper, we describe a way to identify the critical coordination needs that exist in a software development project through post-mortem content analysis and manual coding of task pairs. Our coding scheme provides guidelines on how to score the strength of the relationship of task pairs based on four characteristics. Such a method and coding scheme has the potential to become a research tool that can be used within the community of researchers and practitioners interested in the socio-technical aspects of software development to identify coordination needs for their analysis in future studies. We seek community feedback to help improve the proposed coding scheme.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management – *productivity, programming teams.*

General Terms
Management, Performance, Human Factors.

Keywords
Task Dependencies; Proximity; Coordination Requirements; Awareness; Collaborative Software Development; Machine Learning; Content Analysis; Manual Coding

1. MOTIVATION
Large, complex software projects are often designed to streamline the technical dependencies between modules as a way to maximize task parallelism [6]. However, it is not possible to eliminate all dependencies between modules, and those dependencies often result in coordination needs between project tasks and the corresponding developers [4]. Cataldo et al. found that when coordination activities focus on the tasks where dependencies exist, productivity is likely to improve and the chance for errors is reduced [4]. If developers are made aware of their coordination needs early, more effective coordination can occur while the development tasks are still underway. To accomplish this, dependencies could be garnered from architecture design documents or from a syntactic analysis of the code. However, dependencies often change during the lifecycle of the project, and these traditional methods may not capture all types of dependencies or pick them up in a timely way.

In our recent work, we introduced a quantitative measure, called Proximity, for detecting and quantifying the need to coordinate between software developers [3]. Proximity uses electronic traces of artifact consultations and edits within a developer's IDE to identify coordination needs in near real-time. Proximity has been shown empirically to provide an accurate view of coordination needs and provides more timely awareness than other techniques. However, Proximity and other methods are limited in that they detect coordination requirements only between pairs of developers and do not provide additional context. Developers may work on multiple tasks at the same time, so a coordination requirement between two developers may encompass the work dependencies of many tasks. We extended the Proximity method to detect coordination needs between pairs of tasks instead but found that, at this level of granularity, Proximity tends to introduce too much noise and to list too many candidate task pairs with coordination needs [2].

To resolve this concern, we used machine learning techniques to supplement the Proximity metric and identify only the most critical coordination needs between pairs of tasks [2]. We defined critical coordination needs as those that can cause the most disruption to task duration when left unmanaged. However, a reliable way of capturing critical coordination requirements is not currently recorded in any existing software repositories. In an empirical study of our machine learning approach, we considered the dependencies established by the team within the task records as a ground truth for coordination needs. However, we found that these identified dependencies are often incomplete and inaccurate for use in determining coordination needs [2]. This is in line with a recent study by Aranda and Venolia [1] that found errors and omissions in repositories like Bugzilla. Thus, we were not able to validate our results against any form of ground truth.

Without a complete source of ground truth of coordination requirements to measure against, it is difficult for researchers interested in performing analysis of teamwork, coordination and awareness in software engineering to evaluate their algorithms and techniques. To account for this shortfall, we propose performing content analysis [5] of task pairs to manually code their likeliness of having or not having coordination requirements.

Methods and procedures for manual coding are well established in other research fields. To perform such an analysis, a coding scheme must be developed which details the characteristics of interest along with scoring criteria. Manual coding can be quite time intensive, so the approach is not recommended as a project management strategy that should be applied to all development

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SSE'13, August 18, 2013, Saint Petersburg, Russia
Copyright 2013 ACM 978-1-4503-2313-0/13/08... $15.00.

task pairs in a project. Instead, the manual coding scheme we contribute in this paper can be applied on a subset of task pairs to develop a better understanding of critical coordination needs. Heuristics like this are important because they can become research instruments that can be used in future studies of coordination.

In the remainder of the paper, we propose a set of task pair characteristics that should be included in a coding scheme for this type of content analysis (Section 2) and present preliminary results from coding a small set of task pairs (Section 3). We conclude with a discussion on future work (Section 4) and a summary of our contribution (Section 5).

2. CHARACTERISTICS OF TASK PAIRS INDICATIVE OF COORDINATION REQUIREMENTS

Some issue tracking repositories, like Bugzilla, include a way to mark dependencies between tasks. However, we found that the dependencies captured in this field are not always representative of actual coordination needs [2]. Project members may be unaware of some of their coordination needs, and, therefore, those will not be captured. In addition, project members may use this dependency relationship to capture other types of dependencies that are not necessarily indicative of coordination needs. For example, an umbrella task may be created along with a set of subtasks that the umbrella task depends on as a way of managing complex tasks. In this case, coordination requirements may actually exist among some of the subtasks rather than between each subtask and the umbrella task as captured through the dependency relationship. In addition, these relationships do not distinguish between critical and trivial dependencies, and coordination may not be required for trivial dependencies. For these reasons, we look for other ways of capturing the critical coordination needs.

We propose a set of characteristics of task pairs that may be indicative of coordination needs. Each characteristic is meant to indicate, on a scale, the strength of the relationship between the tasks. The stronger the relationship the higher the indication for coordination. Those characteristics along with the rationale for their selection are described in the following sections. The coding guidelines are shown in Table 1.

Task Summary Similarity: Task summaries have been used successfully in identifying duplicate bug reports with natural language processing [7]. This implies that developers are providing appropriate keywords in the task summaries to allow for automatic detection of duplicates. If duplicates can be detected in this way, it possible that similar summaries may also be an indicator for coordination needs.

Task Discussion Similarity: Task summaries are submitted when a bug report is initially created. As development work is underway, developers may discuss the task and provide more details once they gain a better understanding of the task. We, therefore, also include discussion similarity as a characteristic in our coding scheme.

Table 1. Coding Guidelines

No Coordination Need ⟶ *Critical Coordination Need*

Characteristic	No	Somewhat	Very
Task Summary Similarity	The two task summaries do not appear similar in any way.	A small portion of the main keywords overlap in the two task summaries.	A majority of the main keywords overlap in the two task summaries.
Task Discussion Similarity	The discussions of the two tasks do not share any of the same concepts.	The two task discussions refer to common aspects of the system from the perspective of EITHER the user (system functions) or the system architecture (specific reference to code, modules, etc.) OR The two task discussions indicate that the problems may be occurring in the same area of the code.	The two task discussions refer to common aspects of the system from the perspective of BOTH the user (system functions) and the system architecture (specific reference to code, modules, etc.) OR The two task discussions refer to the same or similar problems.
Evidence of Task Conflict	The discussion in the two tasks does not seem to indicate that the two tasks were conflicting in any way.	The discussion in one of the tasks does not explicitly mention a conflict between the two tasks. However, based on reviewing the timing of the tasks and their discussions, it seems there may have been a conflict between the two tasks that the team may not have been not aware of at the time.	It is apparent based on the timing of the tasks and the discussion thread that there was a conflict between the pair of tasks. The conflict is clearly discussed and may or may not explicitly link the two tasks by ID.
Artifact Overlap	The two tasks have no common files in their working sets (artifacts edited or consulted).	At least one, but no more than 30%, of the union of the files is shared across the two tasks' working sets.	More than 30% of the union of the files is shared across the two tasks' working sets.

Evidence of Task Conflict: Task conflict is the epitome of a coordination need. This can be seen as a more specific case of discussion similarity, but it may be useful to distinguish between criticality of coordination needs.

Artifact Overlap: Developers who are working on the same artifacts at the same time may need to coordinate their work. We found that looking at overlap between artifact consultations and edits was indicative of a need to coordinate between developers [3].

To apply this heuristic, one or more coders must analyze the task pairs and score each pair using the coding guidelines outlined in Table 1. Considering the subjective nature of this activity, higher confidence can be obtained by having multiple coders perform the content analysis and coding independently. As a way to calibrate amongst the coders and prevent discrepancies in the coder output, the coders should compare their findings and discuss any differences after an initial small subset of the coding activity [5].

Determining a threshold for when coordination needs exist based on the scores of each task pair may be dependent on each data set. There are a number of possibilities for this threshold. For example, one must decide whether a coordination need exists if any of the characteristics show similarity or if all of the characteristics show similarity. Also, do coordination requirements exist when characteristics are ranked as somewhat similar or must they be very similar? This is likely dependent on individual data sets and the habits of the development team.

3. PRELIMINARY RESULTS

We carried out a preliminary study on the Mylyn open source project. We collected all Bugzilla change requests (tasks) from releases 3.1 and 3.2. We selected a set of 248 task pairs. Of those 248 pairs, 124 were selected as potential critical coordination needs and 124 that were likely not coordination needs. We selected the potential critical coordination needs if the pairs met any of following criteria: the tasks had a high Proximity score [3] where high is calculated as mean + (2 x stddev,) of Proximity scores over all pairs, the tasks were marked as dependent or duplicate within their Bugzilla records, the tasks were cross-referenced in their discussions, the tasks shared the same umbrella task, the tasks were marked with the same tag. These can be seen as the pool of all potential coordination needs of which we wish to identify the most critical. Two external coders familiar with software development practices independently performed the content analysis and coding of the task pairs. For each characteristic, the coders have a high level of agreement (Table 2). We considered for each of the characteristics, a score of somewhat or very as a positive identification of a coordination need. For our evaluation, we look only at pairs where there was at least a binary agreement between the coders, that is, the coders agreed on a positive or negative response. However, for the positive cases, the coders may have selected different strengths of relationship.

Table 2. Coder Agreement

Characteristic	% Agree	% Agree Binary
Task Summary Similarity	91%	91%
Task Discussion Similarity	94%	94%
Evidence of Task Conflict	93%	93%
Artifact Overlap	91%	96%

For each of the characteristics, the coders identified positives for only task pairs from the pool of 124 suggested potential coordination requirements. The 124 cases that were selected as not likely coordination requirements were rated negatively for each characteristic by the coders.

Task Summary Similarity: The two coders found 11 pairs with task summary similarity. Of these, only three had been identified by the developers as dependent tasks. The remaining were not explicitly identified by the team, but were included as potential coordination needs in our pool due to either their high Proximity or shared tags. A review of the remaining eight task pairs shows that task summary similarity, at least in this data set, does not seem to be an accurate indicator of critical coordination needs. Many of these task pairs shared keywords, but they do not appear to be solving the same problems or conflicting in any way with the exception of one task pair. That pair is also identified by each of the other characteristics and will be discussed later. The three pairs that were identified by the team are also captured by additional characteristics of the coding scheme. We conclude that including summary similarity as a characteristic may not be necessary.

Task Discussion Similarity: Only six pairs were identified with discussion similarity by the coders. Three of these pairs had been marked as dependent by the development team and two additional pairs had been cross-referenced in their discussions. The remaining task pair had high Proximity, but it does not appear to have been marked by the team in any way as a dependency. The two tasks both addressed bugs with the way hyperlinks were working and they had significant overlap in the artifacts that were involved.

Evidence of Task Conflict: The coders found only five pairs with evidence of task conflicts. Three of these pairs overlap with those identified with task discussion similarity including the case discussed above. The remaining two had both been identified by the development team; one through discussion cross-reference and one as an explicitly marked dependency.

Artifact Overlap: The highest number of pairs (33) was identified by this characteristic; 22 of those were not identified using any of the other characteristics. Only one of the 22 had been identified by the development team through cross-reference in the discussion. The remaining pairs were selected as potential dependencies in our data set because of their high Proximity or shared tags. Since it is possible that artifact overlap exists for even trivial coordination requirements, we postulate that this characteristic may be better used to confirm rather than predict critical coordination needs Further investigation is needed. It is also possible, for future studies, to automatically calculate this characteristic rather than relying on manual coding.

In our results, some overlap exists between the four characteristics as illustrated in Figure 1. As mentioned, task summary similarity does not seem to be necessary due to its high overlap with the other characteristics. Those captured by summary similarity alone do not appear to be critical coordination needs. The remaining three characteristics each capture at least one critical coordination need not identified by the other two characteristics.

If we consider any task pairs that were scored positively for at least one of the three characteristics that we identified as good measures of detecting critical coordination needs (task discussion similarity, evidence of task conflict, and artifact overlap), 35 unique task pairs were coded as true coordination needs of the 124 potential coordination requirements. All of the dependencies that were explicitly marked by the Mylyn development team are

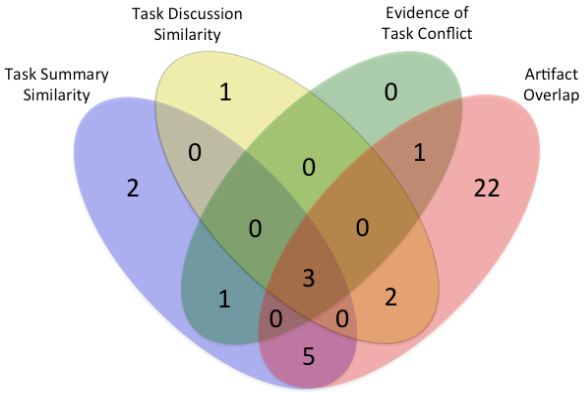

Figure 1. Scoring Overlap between Characteristics.

included in this set, as well as additional pairs that may have been missed by the team. The various indicators that we used to choose the 124 potential coordination requirements pool for this preliminary study may not always be indicative of critical coordination needs, so it is not surprising that only a subset of these were identified by the coders.

To begin to evaluate the criticality of the 35 task pairs identified by the manual coding, we examined the task durations of tasks involved in the critical coordination needs. The tasks involved in these 35 pairs have a different performance profile when compared to the tasks from the remaining 89 pairs that were not selected as critical coordination needs. The tasks with critical coordination needs have a mean duration of 12.5 days compared to 9.4 days for the other tasks. A Mann-Whitney test shows the task durations have significantly different distributions ($W = 894$, $p = 0.008$). Since we defined critical coordination needs as those most likely to cause disruption to task durations, these results suggest that our approach is identifying the most critical coordination needs. Further validation can be done through consultation with the Mylyn development team to ensure that additional critical coordination needs were not missed.

4. FUTURE WORK

Our preliminary results show promise for a research tool like the one proposed to detect critical coordination needs. The high level of agreement of the coders shows the feasibility of using such a coding scheme. We seek feedback from the SSE workshop community to help improve the proposed coding scheme. Community feedback is critical to develop a well-rounded and widely accepted research instrument that has the potential to be used again in multiple studies across the social software engineering community. After incorporating that feedback, we plan to validate the coding scheme with developers.

Once the coding scheme has been enhanced and validated, we plan to use it for our future studies of coordination needs on open-source projects. We intend to apply it to additional task pairs to perform a thorough analysis of our most recent work, which focuses on automatically identifying critical coordination needs in software projects through machine learning techniques. We will use the output of this content analysis and manual coding to both train the machine learning algorithm and validate its results.

5. CONCLUSION

Dependencies between tasks may not always require coordination. We have proposed a way to identify the critical coordination needs through content analysis and manual coding of task pairs. Such a method and coding scheme could become a research tool that could be used within the community to help identify coordination needs for analysis in future studies.

6. ACKNOWLEDGMENTS

Special thanks to Sean Goggins and Nora McDonald for dedicating their time to perform the manual coding described in this paper and for their feedback and suggestions. This work was partially supported by the NSF through grant no. VOSS OCI-1221254.

7. REFERENCES

[1] Aranda, J., & Venolia, G. (2009, May). The secret life of bugs: Going past the errors and omissions in software repositories. In Proceedings of the 31st International Conference on Software Engineering (pp. 298-308). IEEE Computer Society.

[2] Blincoe, K., Valetto, G. & Damian, D. (2013, August). Do All Task Dependencies Require Coordination? The Role of Task Properties in Identifying Critical Coordination Needs in Software Projects. To Appear in ESEC/FSE 2013.

[3] Blincoe, K., Valetto, G., & Goggins, S. (2012, February). Proximity: a measure to quantify the need for developers' coordination. In Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work (pp. 1351-1360). ACM.

[4] Cataldo, M., Herbsleb, J. D., & Carley, K. M. (2008, October). Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (pp. 2-11). ACM.

[5] Krippendorff, K. (2012). Content analysis: An introduction to its methodology. SAGE Publications, Incorporated.

[6] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), 1053-1058.

[7] Runeson, P., Alexandersson, M., & Nyholm, O. (2007, May). Detection of duplicate defect reports using natural language processing. In Software Engineering, 2007. ICSE 2007. 29th International Conference on (pp. 499-510). IEEE.