Recommending a Starting Point for a Programming Task: An Initial Investigation

C. Albert Thompson
Department of Computer Science
University of British Columbia, Canada
leetcat@cs.ubc.ca

Gail C. Murphy
Department of Computer Science
University of British Columbia, Canada
murphy@cs.ubc.ca

ABSTRACT

When starting a new task, a software developer must typically find one or more starting points amongst many resources (e.g., source code and other files) forming the software system. In this paper, we consider how we might recommend one resource as an initial starting point, saving the developer the effort of having to search or use other means to find the point. Using data from the open source Eclipse Mylyn project, we investigate whether resources considered and changed for other tasks may be used to recommend a starting point for a current task.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments; K.6.3 [**Software Management**]: Software development

General Terms

Algorithms

Keywords

program history, repository mining, task similarity

1. INTRODUCTION

Performing a task often requires consulting or changing more than one resource (e.g., source file) [13]. For many tasks, several resources must be located, considered and sometimes changed. Finding the resources necessary to reason about and change can be time consuming [5]. Ideally, a recommender could suggest all of the resources needed to complete a task so that the software developer can focus their attention on understanding those resources and changing them correctly to complete the task. In this paper, we consider a first step towards this ideal goal by investigating whether it is possible to recommend a resource as potential starting point to the developer working on the task.

We consider whether it is possible to construct a recommender that uses the similarity of task descriptions and the overlap of considered and changed resources associated with the similar tasks as the basis for recommending a potential starting resource. To test this hypothesis, we used data from the Eclipse Mylyn[1] open source project that includes both resources considered and resources changed per task within its repositories. We found that a simple approach resulted in a precision of 0.21 for recommending a single starting point. Our initial results are encouraging but also indicate the need for substantial improvement before a useful technique can be provided.

We begin by comparing our approach to existing work (Section 2). We then consider whether there is sufficient overlap in the data in the repositories of Eclipse Mylyn to make recommendations possible (Section 3) and report on the results of applying one recommendation algorithm to this problem (Section 4). We conclude with a discussion of a few of the many open issues (Section 5) before summarizing the results (Section 6).

2. RELATED WORK

Finding starting points for a task is similar to several other problems that have been tackled in software engineering research, including determining features (e.g., [2, 14, 1]), locating concepts or aspects (e.g., [7, 3, 11]), and finding the next code to change as part of a task (e.g., [16, 8, 15]), amongst others. The approaches taken to these problems typically use one of four different approaches: dynamic analysis, static analysis, textual analysis, or data mining. The approach we take in this paper is most similar to those used in textual analysis and data mining.

Empirical studies have shown that developers typically attempt to find starting points using search terms related to the task about to be performed (e.g., [5]). Various textual analysis approaches have been taken to help streamline this search process. For example, Marcus and colleagues retrieve concepts from a corpus of externally defined concepts and enable developers to use that as a base to find information relevant to those concepts in the code of interest [7]. As another example, Topic *XP* extracts topics related to resources and uses those extracted topics to recommend other similarly related resources [11]. While these style of approaches help streamline searching by providing a concept-based search, they still require the developer to formulate a search. We are interested in streamlining this further by providing recommended resources without requiring a developer to formulate a query.

The approach we consider in this paper is based on data mining. A number of existing tools take such an approach. An early example is Hipikat which brought information from multiple repositories into a central schema, allowing recommendations to be made of such information as which change

[1] `eclipse.org/mylyn`, verified 31/01/14.

RSSE '14, June 3, 2014, Hyderabad, India

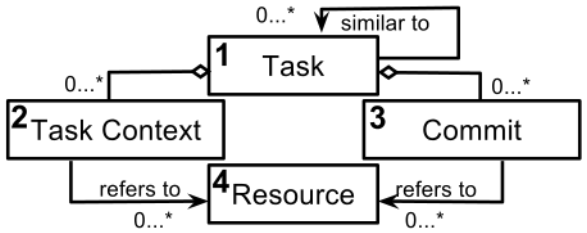

Figure 1: The relationship between objects we use to create our algorithms

done in the past and which files changed in the past are similar to a current change [1]. In this use, Hipikat could provide a recommendation of starting points associated with the most similar task done in the past. We hope to improve the accuracy of such a recommendation by considering how multiple past tasks were completed. Other works have considered how to recommend to a developer the next steps to take in a task. For example, Zimmermann and colleagues mined version control information to recommend other places in the code to edit based on how changes in the past touching similar code were completed [16]. The approach we consider in this paper attempts to take the first step in this process by providing the first resource or file to consider.

3. ASSESSING THE DATA

To be able to make a helpful starting point recommendation, we need to determine if the resources considered and changed as part of tasks have any overlap. If there is no overlap, the approach of using how other tasks have been completed is not likely to be useful. If there is overlap, it is worthwhile to investigate algorithms based on this data.

The data we assess is drawn from the repositories associated with the Eclipse Mylyn open source project. When a new feature is worked on or a bug is solved, it is the custom in this project that task contexts [4] representing the work performed on the task are attached to the issue describing the feature or bug. A task context includes all resources selected or edited as part of working on the task weighted by the degree to which they were interesting—worked on frequently or recently—with respect to the task. More than one task context may be associated with a task if work has proceeded at different times. The data we consider includes the task contexts for issues as well as commits associated with the issue representing the resources actually changed as part of the work on the task.

Figure 1 depicts the data and its relationships available from the Mylyn repositories. From Mylyn's Bugzilla repository, which records tasks, we extracted 9,484 tasks (box 1 from figure 1). Each task has zero or more task context attached; we extracted a total of 3,902 task context (box 2 from figure 1) from the attached resources to each task in the Mylyn's Bugzilla repository. We used the task id that was added to each git commit message and used pattern matching to connect the commit to the task id to get a total of 9,982 commits matched to a task id (box 3 from figure 1). We also extracted 45,824 resources from both matched git commits and contexts (box 4 from figure 1).

We would expect that the programming elements recorded as part of a task context are a superset of the programming elements recorded as part of a commit since presumably a programmer would need to touch an element before it is committed. We found that even though task contexts are supposed to keep track of all resources touched or changed there were still resources committed that were not in associated task context. On average 83% all committed resources were also in the task context. The lack of capture of all resources by task contexts means one of the following: Mylyn was not used for parts of a task, resources are changed outside of the scope of Mylyn, or resources are changed automatically. Thus we cannot solely rely on using task context to make recommendations because other resources may be necessary to complete a task and we should also include commits.

To assess the usefulness of this data for starting point recommendations, we also needed a way to determine similarity between task (or issue) descriptions. We use the standard information retrieval approach of using TD-IDF [10] values for each task. We computed TD-IDF for each task using the title, description, and comments for that task. We then determined the similarity between tasks using the cosine similarity of the TF-IDF values (e.g., [9, 6]). We formed a test set of tasks consisting of 4206 tasks from the Mylyn task repository that each have at least one attached commit or task context. Our test set of tasks had on average 1.1 task contexts and 2.8 commits.

For each task in the test set, we form the largest possible set of resources considered or changed as part of work on that task consisting of all resources related to the task , whether the code was named in a task context, or in a commit. We repeat this same process for each task to form an oracle for the resources associated for every task.

To assess the data, for each file in the test set of 4206 tasks, we compute the intersection of the resources in the oracle with a union of the total possible resources that were considered or changed in the top 10 tasks using the cosine similarity approach to the task being processed. We found an average overlap of 47% ($\pm$ 35%) between the oracle for a task and the possible set of data from which to recommend. This overlap lends evidence that there is similarity between work performed on similar tasks making a recommendation of a starting point possible.

4. ALGORITHM

The algorithm that we investigate to find a starting point recommends a resource by finding the intersecting resources of similar tasks. We use cosine similarity to find and order ten tasks ($T_1 \ldots T_{10}$) that are most similar to the task for which we want to find a starting point (T_0). We then compute the intersection of all resources between the two top ranked similar tasks (e.g., T_1 and T_2). If there are one or more common resource we return one random resource. If there are no resources in common, the algorithm proceeds by comparing the resources of T_1 and T_2, selecting a common resource to return if one exists. The algorithm continues in this manner until a resource is found to recommend.

Applying this algorithm to the 4206 tasks for Mylyn that have at least one context or commit, we return one resource for each task with a precision of 0.21. We focus on precision because we want to see if our one recommendation is correct; recall is not of interest to our problem at this time.

The algorithm performs better when the task for which the recommendation is being given (T_0 above) has more than 100 files in the task contexts and commits associated

with the task. In the 424 tasks in this category, the precision rises to 0.55. This rise in precision is not surprising given the large set of potential recommendations to make correctly. The algorithm performs worse when there is only one resource associated with the task of interest. In these cases, the algorithm has a precision of 0.08. There were 409 cases in this category from our test set of tasks.

5. DISCUSSION

The algorithm we investigate in this paper is preliminary. Many open questions remain.

Order of tasks

Our testing of the algorithm does not consider the sequence in which tasks are performed. As a result, in the precision number reported, resources may have been recommended based on tasks completed after the task of interest. This choice increases the corpus of tasks which we test and may inflate precision. A realistic recommender must consider the sequence in which tasks are performed.

What is a starting point?

Our definition of a starting point in this paper is a resource that was considered or changed as part of a task. It may be that a resource we are recommending is not a suitable starting point as it may be difficult from that resource to understand why it is being recommended or to find relevant code from that point. It may also be a resource that was a red herring for the task. Empirical study is likely necessary to understand what developers consider reasonable starting points. One possibility to better assess starting points are whether there is overlap in the text of the task description with the names and comments in the resource as developers often use search to find starting points. Another possibility is that a starting point should be closely related, perhaps structurally or textually, to one of the resources changed as part of the task completion.

Other approaches

Moving forward we want to look at different approaches to recommending a starting point. Another way we could recommend is to extract features from the comments in each task in a repository using natural language processing and create a database of features. Similar work is done for classifying action items in emails [12]. Then based on an new task we can extract features from the description and use the database and retrieve similar features to use to recommend resources.

Another possibility approach to building a starting point recommender is to create an algorithm using machine learning to cluster tasks based on the resources they change. From the cluster information, an algorithm can infer what resources are necessary to complete a type of task. A recommender can then compare a new task to existing clusters to make recommendations based on the resources in the clustered tasks.

6. SUMMARY

This paper provides evidence that it may be possible to build a recommender to help a software developer know on which resource (e.g., source code file) work might start on, for a new task to be performed. An assessment of the Eclipse Mylyn open source project data showed that there is substantial overlap in the resources considered and changed as part of similar tasks when task similarity is based on cosine similarity of TF-IDF scores for the tasks. Preliminary analysis of a straightforward algorithm based on recommending a resource used in tasks similar to the task of interest can return a starting point with a precision of 0.21.

ACKNOWLEDGMENTS

This work was funded in part by IBM and in part by NSERC through the NECSIS project.

7. REFERENCES

[1] D. Cubranic, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *TSE'05*, 31(6):446–465.

[2] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *TSE'03*, 29(3):210–224.

[3] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. von Staa. Modularizing design patterns with aspects: a quantitative study. In *Transactions on Aspect-Oriented Software Development I*, pages 36–74. Springer, 2006.

[4] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT FSE'06*, pages 1–11. ACM.

[5] A. J. Ko, H. H. Aung, and B. A. Myers. Eliciting design requirements for maintenance-oriented ides: a detailed study of corrective and perfective maintenance tasks. In *ICSE'05*, pages 126–135. IEEE.

[6] G. Kumaran and J. Allan. Text classification and named entities for new event detection. In *SIGIR'04*, pages 297–304. ACM.

[7] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *Proceedings. 11th WCRE'04*, pages 214–223. IEEE, 2004.

[8] M. P. Robillard, W. Coelho, and G. C. Murphy. How effective developers investigate source code: An exploratory study. *TSE'04*, 30(12):889–903, 2004.

[9] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW 06*, pages 377–386. ACM.

[10] G. Salton and M. J. McGill. Introduction to modern information retrieval. 1986.

[11] T. Savage, B. Dit, M. Gethers, and D. Poshyvanyk. Topic *XP*: Exploring topics in source code using latent dirichlet allocation. In *ICSM'10*, pages 1–6. IEEE.

[12] S. Scerri, G. Gossen, B. Davis, and S. Handschuh. Classifying action items for semantic email. In *LREC'10*.

[13] N. Thomas and G. Murphy. How effective is modularization? *Making Software: What Really Works, and Why We Believe It*, page 373, 2010.

[14] N. Wilde and M. C. Scully. Software reconnaissance: mapping program features to code. *Journal of Software Maintenance: Research and Practice*, 7(1):49–62, 1995.

[15] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting source code changes by mining change history. *TSE'04*, 30(9):574–586.

[16] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *TSE'05*, 31(6):429–445.