Creating Task-Based Concern Maps by Merging Concern Fragments

Sukanya Ratanotayanon
Dept. of Informatics
University of California, Irvine
sratanot@uci.edu

Susan Elliott Sim
Dept. of Informatics
University of California, Irvine
sesim@uci.edu

Abstract

On any project, it is not possible to have complete and accurate concern maps for all possible tasks. We present an approach to creating concern maps from available secondary software work artifacts from common software tools, such as revisions control. We mined and indexed concern fragments from repositories of those tools. Developers can search the index for an initial set of relevant fragments. To create a final concern map, the members from the initial set of fragments are validated, merged and expanded using a call graph. Members of the final concern map members are also ranked to guide developers to more relevant sections of the code.

1. Introduction

When professional developers work on maintenance tasks, they want to pay attention only at the code related to their task. An accurate concern map that links a concern to its implementation in the code can help programmers to find all and only the parts of the code relevant to their work. However, it is rare to have an accurate and complete concern maps for all concerns in the software. Like other kinds of documentation, developer effort is required to create and maintain concern maps. As well, details may be left out of the concern maps, intentionally or inadvertently, resulting in only partial concern maps, or concern fragments. Moreover, it is not possible to anticipate the need for a particular concern maps in the future, so we will often encounter situations where no concern map is available. Therefore, instead of relying on availability of a complete concern map, we aim to create it in an ad hoc manner for a particular task.

Previous work addressed this issue by reverse engineering concern maps using information retrieval {{Marcus,Andrian 2003;}}, machine learning {{261 Zimmermann,Thomas 2004;}}, and program analysis techniques {{ 264 Zhao,Wei 2006; }} to find relationships between code and sections of high-level documents, such as requirement specifications. However, difficulties in using these techniques include the need for reliable documentation and insufficient control over which concern maps are created.

We draw on this previous work to create an approach that builds on their strengths while minimizing their weaknesses. Our approach aims to interactively re-construct a concern map using available secondary work artifacts such as commit transactions, task context and defect reports. We can obtain concern fragments from the secondary artifacts as they link together high-level description and source code. A concern map for a novel task can be created by combining members of relevant concern fragments.

To use concern fragments to locate sections of code relevant to a task, we need to address the following three challenges. One, there needs to be a mechanism for finding concern fragments that are relevant to the programmers' task. Two, there needs to be a mechanism to locate concern members that are relevant, but missing from the concern fragments. Three, there needs to be a mechanism to identify the most relevant concern members, so these can be investigated first.

2. Creating Task-Based Concern Maps

We use secondary artifacts because they are commonly produced as a part of existing work practices. Therefore, require no additional effort to create and maintain. In addition, they contain conceptual-level information that is difficult to find in the source code, such as feature descriptions, task descriptions, and links to related code. For example, revision control systems record high-level comments with what have been changes; bug tracking systems record changesets that are associated with a specific bug and bug description; context management tools, such as Mylyn, record which parts of code are examined during a task. Using this information, we can obtain a concern map from theses secondary artifacts. But concern maps from these tools are often incomplete, in another word they are concern fragments. For example, a concern fragment obtained from commit transaction may miss program units that were indirectly related to the task, but not modified or may contain only a facet of the concern that was touched by the task. To obtain a more accurate concern map, instead of using only one concern fragments, our approach combine a group of concern fragments that are conceptually related to a programmer's task to create a final concern map for the task.

We combine techniques from Information Retrieval (IR), program analysis, and data mining version histories to address the challenges in reconstructing concern maps using concern fragments. Our approach is shown in Figure 1.

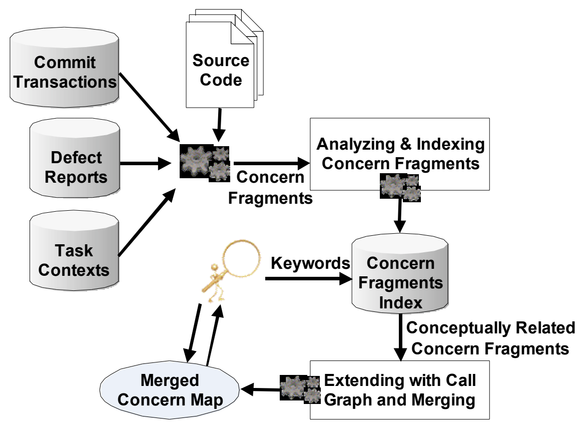

Figure 1: Approach to Re-constructing Concern Fragments

We populate the concern fragment repository with concern map fragments inferred from repositories of secondary artifacts. We represent concern maps by explicitly listing their members, because this extensional representation makes it easier to combine fragments from different tools. The smallest unit of concern members in our representation is unique identifier such as fields and methods for Java code and files for non-Java artifacts.

To enable a programmer to search for concern fragments, we created a repository with concern fragments and their metadata. The information that we indexed in the repository is: conceptual description of the concern, author, creation date, and program elements that are members of the concern fragments. Users can retrieve an initial set of relevant concern fragments in an ad hoc manger by querying with both program elements and domain-level vocabularies as keywords.

To improve the ad hoc concern map, we validate, extend, and combine the members of concern fragments that are in this initial set. When using concern fragments from historical records, we may encounter elements that no longer exist. Therefore, we validate whether a member exists in the current version of the code, before performing further analysis. We deal with the problem of under-reporting of concern members using a static call graph to discover members missing from a concern fragments. This technique reduces false negatives, by extending a concern fragment with program elements that are in the same sub-tree of the call graph or are used by the current members of the concern fragment. Lastly, all the existing and inferred members of concern fragments are combined to construct a concern map.

To deal with the problem of knowing which concern members are more important to the task, we rank members of a final concern map using a relevance metric. This metric is based on whether the concern member was actual or inferred, how frequently a concern member appeared in the merged fragment, and whether the user had shown interest in the returned fragments. This technique reduces the impact of false positives. Later, to fine tune the concern map, it is possible to add or remove concern fragments from the set.

3. Future Work

We have already implemented our approach in a prototype Eclipse plug-in, called Kayley. Our future work includes improving the precision and recall of the returned concern fragments. We also plan to improve the tool's ability to infer the relevance of a concern fragment during the initial search. As well, we need to improve our metric for ranking; currently, some marginally relevant members such as utility methods are currently receiving high score, because they are used frequently. Lastly, we need to evaluate the performance of this approach with comparable industrial software.

4. References

[1] A. Lakhotia, "Understanding Someone Else's Code: