Applying Aspect Mining Techniques to Understand an Existing Program

Fernanda Campos Yadran Eterovic
Pontificia Universidad Católica de Chile
{mfcampos,yadran}@ing.puc.cl

Abstract

Research in magnetic resonance imaging requires reprogramming the software that controls the scanner -a difficult task. We have successfully applied three aspect mining techniques to obtain information about several properties of this complex, non object oriented program.

Categories and Subject Descriptors D.2.7 [*Software*]: SOFTWARE ENGINEERING—Distribution, Maintenance, and Enhancement

Keywords Program understanding, Aspect Mining, Execution relations, Event traces, Concept analysis, Random Walks.

1. Introduction

Program comprehension is a practical field in software engineering concerned with understanding existing code for reuse, maintenance and refactoring. Research in magnetic resonance imaging (MRI), requires reprogramming the software that controls the scanner, which in turn requires identifying the functions and parameters that must be changed.

Our focus is understanding the program used to control a Philips scanner; a large, complex, non object oriented, almost not modularized and poorly documented code. It contains over 300,000 lines of code written in GOAL-C, a special-purpose programming language. A typical execution trace obtained by executing a pulse sequence in the scanner may contain more than 30,000 function calls. Therefore, identifying the changes that have to be made, usually takes a very long time. We hypothesized that by applying aspect mining to the program we could obtain information about its properties (not necessarily it's aspects), which could help us perform the changes required by MRI researchers

SPLASH'12, October 19–26, 2012, Tucson, Arizona, USA.
ACM 978-1-4503-1563-0/12/10.

2. Aspect Mining

Aspect mining is a reverse engineering process, which consists in finding crosscutting concerns hidden in existing software. Techniques have been tested mainly on small scale experiments in object oriented languages. We are interested in the application of these techniques to real, non object oriented code, where they may help understand the program's organization, behavior and important features.

We chose three techniques, that give us different information about the program. **Event traces** finds recurring execution traces, indicating general execution patterns in the code. **Formal concept analysis** looks for those functions from the same module that are present in all traces, i.e., scattered code. It computes the list of functions only present in a certain trace, i.e., what functions are more related to a specific trace type. **Modified random walks** identifies the most popular and less significant functions; it can also point out functions that crosscut the system. Since to obtain the crosscutting concerns we need to compute the popularity and significance of each function, this will also give us the list of most significant functions.

3. Applying diferent techniques

To apply **Event traces**, we search for specific relations between functions [1] in the traces. The *outside before* relation: for an execution of function *a*, function *b* is executed just before *a*; the *inside first* relation: for execution of function *a*, the first function called inside it is function *b*; *outside after* and *inside last* relations, with the obvious meanings. Two properties are defined over these relations, a relation is *uniform*, if it exists always in the same composition and *crosscutting*, if it occurs in more than one calling context in the trace. In the original paper execution relations that are both *uniform* and *crosscutting* are considered *aspect candidates*. For our purposes, these candidates show us information regarding the order in which functions call each other.

By applying this technique to different traces, we obtained a list of relations between functions. We see multiple repetitions of some functions in the relations, called from different modules, from at least 20 files, and from many other functions. This suggests the code is coupled.

Multiple types of validations are found among the most common relations in all traces; with this technique we can obtain which functions are common to the system validations, ie: functions called in various validation phases. We also found functions belonging to the scan simulation interface, called when a user selects a protocol page in the simulator; for instance, *MPIEX_softkey_screen* masks enabled parameters during validation. Thus, given that the program consists of a parameter definition part (PDF) and a measurement part (MPF), this information shows which functions belong to the PDF part of the program: those concerned with the setup and validation of the parameters. What is interesting is that most relations found in most traces are from this part of the program. Only four relations were found which belong to the MPF.

Formal concept analysis technique [2] consists in obtaining a concept lattice from traces. The use case associated with an execution trace is an object whose attributes are the functions in the trace. A concept is the object plus its attributes. Aspect candidates are obtained when different functions from the same module are executed for more than one concept or when a concept has functions from different modules (although this is not sufficient to identify a *crosscutting concern*).

This technique gives us a list of functions that appear on all traces. This means each candidate function is executed at least once in each trace. The fact that it repeats in different executions indicates scattered code, but also refers to functions that are part of the base of the program, like user interface or validation functions. We find among the aspect candidates functions related to the specific absorption rate restrictions; this makes sense since it is a factor that should always be computed before a scan.

Some common functions that execute scans may be considered part of the core of these traces, since they perform the measurements necessary for obtaining the sequences (MPF). For example *MPIMN_scan_exec* is part of these candidates. If we take a closer look we see it is in charge of executing the PDF for the scan, after the "start scan" button is pressed and before the scan starts it calls certain initializations and sets up some parameters for the scan.

We executed heart scan traces and brain scan traces, comparing the results from applying the technique only to *heart traces* with those from applying it to *brain traces*, we obtain candidates that appear in one of the types of sequences and not in the other. This identifies functions that are specific to certain procedure types.

Modified random walks, is the application of a page rank algorithm to compute possible *concerns* by determining which elements are more popular and significant [3]. We modified the original technique by performing the calculations over actual traces. *Popularity* is the number of times an element is visited from different elements. *Significance* is the number of distinct elements an element visits. These ranks help us obtain potential *crosscutting concerns* in two ways. *Homogeneous crosscutting:* popularity is higher than significance by a certain threshold. Other possible concerns are *Heterogeneous crosscutting:* elements with much higher significance than popularity.

Analyzing the results we found that some *concerns* are functions that are also concerns in previous techniques. Thus, these functions are not only crosscutting and uniform relations but they are also the most popular and least significant functions. Only one *concern* is unique to each type of trace. For the heart scans, *MPICOIL_get_app_sar_limit*, a function that gets the maximum amount of specific absorption rate allowed. For the brain scans, *MPGGEO_m_matrix-_to_sg_matrix*, a function that stores a matrix structure.

All candidates for traces of a specific sequence type are the same, and even between different type of traces most of them are repeated. Since they don't vary with the sequence, and since they are called multiple times, they can be considered part of the application's core. Because this technique requires the candidates to have low significance, the results are very simple functions, much like getters and setters.

4. Conclusions

The application of all three techniques, combined with the scarce documentation available, allowed us to understand different properties of the program's execution. Event traces reveals function call sequences found in all traces. Results from concept analysis gives us a list of the functions most likely to be related to general parts of the execution, such as parameter setup and validations. Finally random walks identifies more general functions that are called for both types of traces.

The combined application of several aspect mining techniques, based on dynamic analysis, can reveal important information about the execution of a complex program. Whether this information is related to actual aspects is not as important for us, as is understanding the code: relative execution order, function significance and popularity, function roles and their relationship to specific sequence types.

Overall the information obtained regarding the programs execution is very valuable to improve our understanding of the program. While as software engineers we interpret this information syntactically, an MRI researcher can infer at least part of the programs functionalities.

References

[1] S. Breu and J. Krinke. Aspect mining using event traces. In *Proc. of the 19th IEEE int. conf. on Automated software engineering*, 2004. ISBN 0-7695-2131-2.

[2] P. Tonella and M. Ceccato. Aspect mining through the formal concept analysis of execution traces. In *Proc. 11th Working Conf. on Reverse Engineering*, 2004. ISBN 0-7695-2243-2.

[3] C. Zhang and H.-A. Jacobsen. Efficiently mining crosscutting concerns through random walks, 2007.