Master-Thesis: Incremental Analysis of Software Product Lines

Moritz Flöter

June 2018

floeter@uni-hildesheim.de

Matrikel-Nr: 236278

Supervisor:

Prof. Dr. Klaus Schmid

Christian Köher

Contents

1 Architectural Approaches **1**
1.1 General Approach 1
1.2 One Model Approach 1
1.2.1 How does this work? 1
1.2.2 Advantages of this Approach 2
1.2.3 Disadvantages of this Approach 2
1.3 Multiple Models Approach 2
1.3.1 How does this work? 3
1.3.2 Advantages of this Approach 3
1.3.3 Disadvantages of this Approach 3

1 Architectural Approaches

In this section different approaches to the architecure will be presented and discussed. Those architectures evolve around KernelHaven using the exisiting infrastructure and implementing additional functionality as extensions or plug-ins. For remainder of this work, the term "modelset" refers to a set of instances of the Variability-Model, Code-Model and Build-Model.

1.1 General Approach

1.2 One Model Approach

This approach stores one version of the modelset. The modelset resulting of the extraction will be based on the version of the modelset that is available from the last run of the analysis. Because unchanged parts of the modelset may be reused from the previous modelset, the extraction might be less costly than a full extraction of the modelset from scratch.

1.2.1 How does this work?

The analysis may be divided into three main phases. In the first phase (preparation-phase), the input for the extractors and analysis is defined while in the second phase (extraction-phase) the extractors run and update the modelset. Finally in the third phase (analysis-phase) the analysis itself is performed.

The results of the extraction phase depend on the granularity to which files are analyzed in this phase. Looking at files for the Code-Model only, we could for define the input as one of the following:

- all changed *.c and *.h files
- all changed *.c and *.h files along with other files affected by changes in those files (eg. a class using one of the changed classes)
- only those *.c and *.h files where the code-model did change
- only those *.c and *.h files where the code-model did change along with other files affected by changes in those files

The selection of an option that includes indirectly affected files aswell depends on the extractor in use while the option for ignoring files where the variability-information did not change is expected to yield perfomance improvements for any type of extractor. The latter is especially relevant for commits that change artifact specific information only as no extractor exectutions or analyses will be required.

For the Variablity-Model and Build-Model the existing extractors impose restrictions on the options described for the Code-Model. Those extractors only allow for execution on the entire set of input-files representing the model and do not allow to specify a subset of those files as input. Therefore we will have to

run the extraction on the entire set of files as soon as the variability information changed in one of them.

So far the proposed options for filtering input files were agnostic on the analyis that is to be performed after the extraction.

In addition to those options one could consider to also filter out variability-changes that are not expected to result in changes for the analysis. For example renaming or deleting a feature would not result in additional dead-code blocks for the dead-code-analysis. The same could be true for the introduction of a new feature that does not depend on other features.

After extraction the analysis considers the changes that were not filtered out during the preparation-phase. Therefore the diff used as input gets modified during the preparation-phase so that it only contains the files that were considered to contain relevant changes to the modelset.

1.2.2 Advantages of this Approach

The advantage of this approach is that it maintains and operates on a single modelset. It is therefore not as storage-intensive as the approach described in subsection 1.3 and does not rely on costly diffs between models but may instead operate based only on the diff provided as input.

1.2.3 Disadvantages of this Approach

By integrating elements which are not agnostic of the analysis into the preparation phase, a coupling between preparation and analysis is introduced. The analysis-agnostic filtering already depends on the type of input-files. Through adding additional source-file specific filtering-logic for the preparation-phase, we further intensify those dependencies.

Assuming an extractor that generates models from a less common type of input files, we would need to adjust the filtering process to be able to consider the nature of changes within the source-files.

1.3 Multiple Models Approach

The Multiple Models Approach stores two or more versions of the modelset. As an incremental analysis will only consider changes between the previous and the current revision of source-code, only the two most recent models will be relevant for our analyses. Storing more models might however make sense for archiving purposes or for doing analyses on the evolution of the software.

Because multiple revisions of the model get stored, we have access to two diffs that are potentially relvant for the analysis. First, we have the diff of the source-code used as input for the incremental analysis. Secondly, we now may also find differences between the models after the extraction.

1.3.1 How does this work?

Following the ideas outlined in subsection 1.2, we also divide the process into three phases. The preparation-phase now only contains generic filters for the source-files - for example the analyisis-agnostic filters described for the One Model Approach.

However because multiple sets of models are stored, additional filters may be implemented for a later stage. Assuming deterministic extractors operating on the entire set of source-files without any filtering happening in the preparation-phase, the analysis now has access to the diff of the modelset after the extraction-phase.

Therefore we can introduce a filterstep in the analysis-phase that gets performed before the actual analysis is run. In this step, we can now determine whether models changed and also investigate the nature of those changes (eg. deletion/renaming/addition of feature) without looking at source-files directly.

1.3.2 Advantages of this Approach

The advantage of this approach are reduced dependencies between the elements of the three phases. While we can still perform everything proposed in the One Phase Approach, filters may now also be moved to the analyis-phase. We could for example use the same analyis for different extractors running on all changed files and then later identify the differences in the resulting models. Different extractors might be able to handle different types of source-files while the filtering method used is fairly generic. Therfore the possibilities for reuse of an incremental analysis are improved.

Given that KernelHaven creates abstractions of the models represented by source-files through its extractors, pushing analyis-related tasks towards the analysis-phase also falls more in line with the conceptual background of KernelHaven.

Because we can now determine changes in the build and variability model it might also be possible to include further optimizations such as incremental SAT-solvers for the Dead-Code analysis that operate based on differences within the modelset. For that purpose however the analyis might need to store artifacts of its previous execution such as intermediate or final results of the SAT-solver.

1.3.3 Disadvantages of this Approach

Storing two or more versions of a modelset increases the requirements for storage-space. By using generic or no filters during the preparation phase, the extraction-phase gets more costly. Furthermore storing multiple version of the modelset also takes time. After the extraction is finished, a diff between the current and previous model is created which again is a costly process.

References