Reverse Engineering Abstract Scenarios for Software Comprehension

Edna Braun Ph. D. Candidate. Supervised by Dr. Daniel Amyot and Dr. Timothy Lethbridge University of Ottawa

Abstract

Reverse-engineering software to recover behavioural models is a difficult task, further complicated by the lack of a standardized way of visualizing the extracted knowledge. In this thesis, **we study a technique for automatically extracting static and dynamic data from software, filtering and analysing the data and visualizing the behavioural model of a selected feature of the software**.

We propose a method to simplify lengthy traces by filtering out software components that are too low-level to give a high-level picture of the selected feature. We use various static information to identify and remove small and simple (low complexity index) software components from the trace. **We define a *utility* method as any element of a program designed for the convenience of the designer and implementer and intended to be accessed from multiple places within a certain scope of the program**. We identify *utility* methods by using various **combinations of selected dynamic and static variables to calculate their *utilityhood***. We use an iterative approach to detect and then remove those methods that have high *utilityhood* values. By eliminating the utilities, **we are left with a much smaller execution trace**. The resulting trace is then visualized using the Use Case Map (UCM) notation. We validate the results by showing the generated models to software developers who have knowledge of the software feature we chose to model.

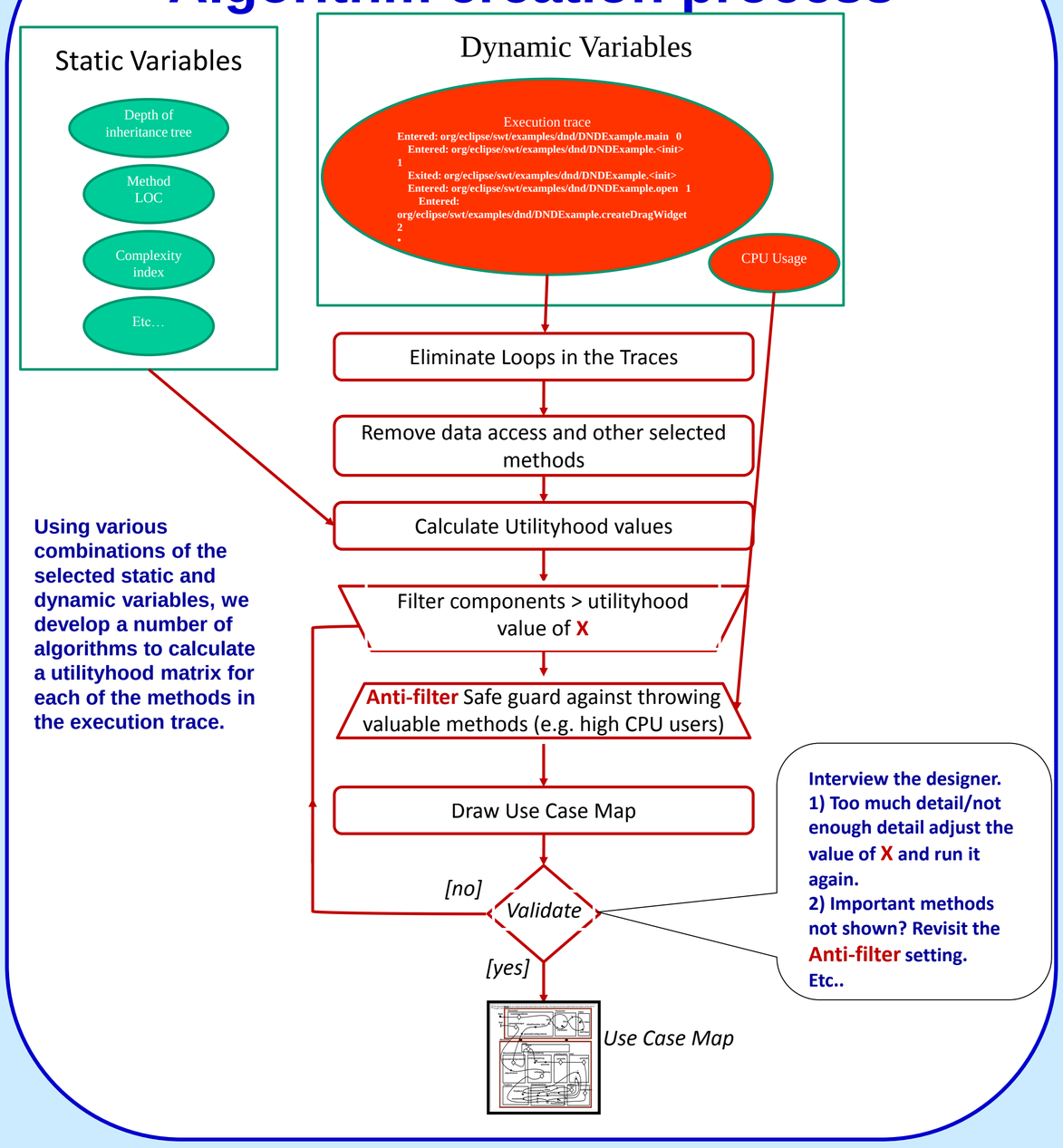
Motivation

The main motivation of this research is to reduce the time and effort spent on program comprehension.

It is very rare for a new programmer to be given a requirement for a software product and asked to implement a solution from scratch. It is much more common that a programmer has to **modify existing code to add a feature, fix a bug, or enhance an existing feature**. Computer programmers spend most of their time repairing existing software and updating it to keep it running. A programmer is more likely to be involved in software evolution and software maintenance than in new, greenfield software development.

We study a method to extract the behavioural model of one software feature at a time; given a specific set of inputs, automatically and quickly.

Algorithm creation process

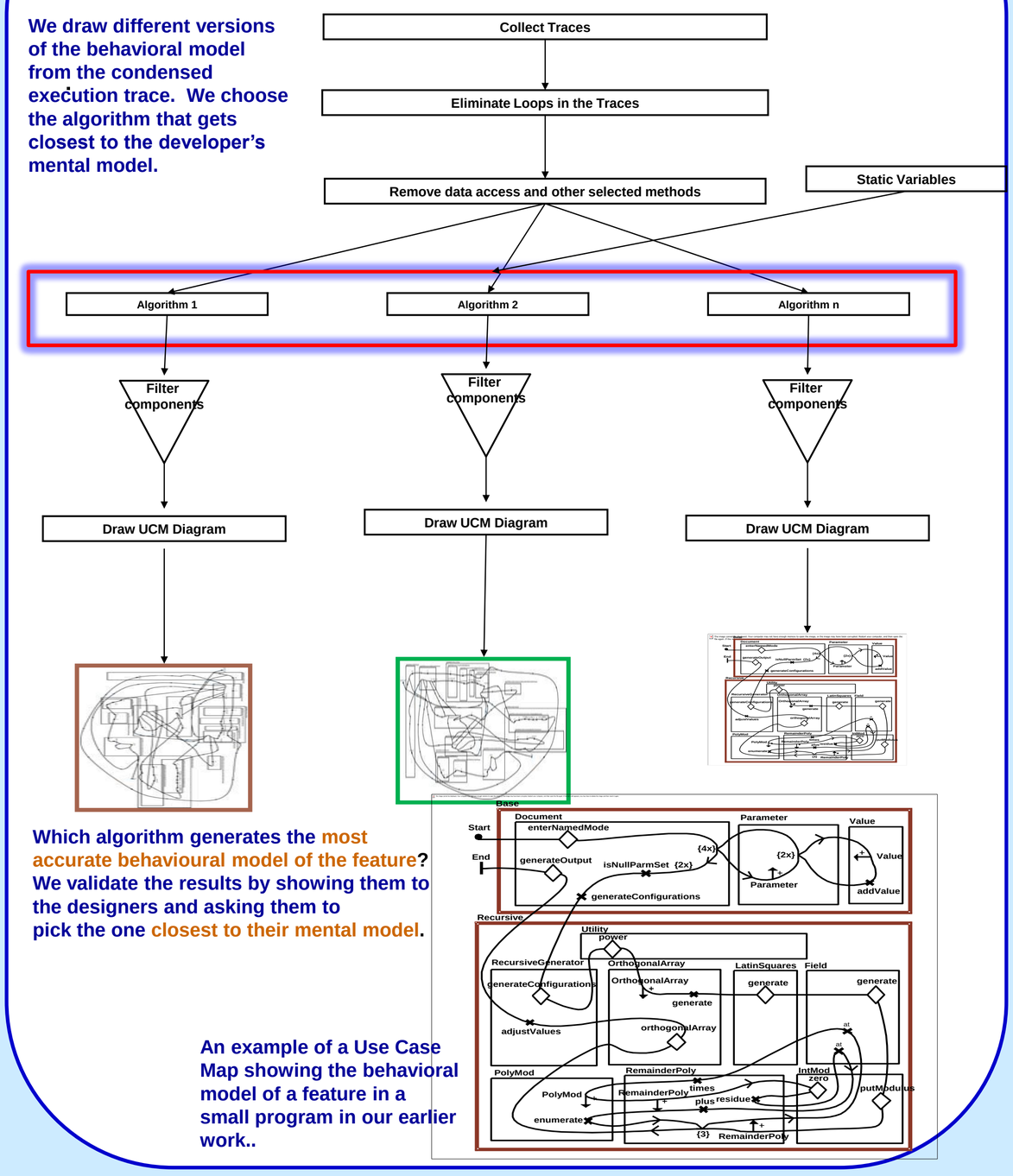


Summary of Research Methodology

Problem Definition	• Define the problem. • Define objectives and goals of the thesis. • Discuss why it is important to study the problem and who the solution could impact.
Initial literature review	• Review papers that have used static and dynamic data analysis for software comprehension. • Create a list of criteria for good software comprehension methodologies. • Critic the papers using the criteria developed.
Run Case Study 1 to determine data sets	• Select the static and dynamic data to collect . • Create algorithms to detect utilities using static and dynamic data collected. • Run the algorithm on small software and fine tune the algorithms based on the results.
Run Case Study 2 to fine tune algorithms and design of experiment	• Run one feature of jUCMNav and collect execution trace. • Collect static and dynamic data, analyse, and calculate utilityhood of each method called in the feature. • Ask participants to mark methods as utility and non-utility. • Check if answers align with utilityhood rating of any of the algorithms. • Draw UCM diagrams based on the filtered traces and show the designers to see if it matches their mental model of the feature
Analyze results from study	• Analyze the results. • Fine tune the algorithms, the questionnaire and the design of the experiment.
Run Case study 3 and validate	• Repeat the experiment on larger software (Weka) • Validate the results by surveying Weka designers online to check if the essence of the feature is captured.
Discussion and conclusion	• Compare our findings with other reviewed work using the evaluation criteria we define in Chapter 3 section 1. • Discuss the findings and future work.

Compare the Created Algorithms

We draw different versions of the behavioral model from the condensed execution trace. We choose the algorithm that gets closest to the developer's mental model.



Which algorithm generates the most accurate behavioural model of the feature? We validate the results by showing them to the designers and asking them to pick the one closest to their mental model.

An example of a Use Case Map showing the behavioral model of a feature in a small program in our earlier work..

Contribution

- A set of **utility detection algorithms** to detect utilities and exclude them from execution traces in an effort to reduce the size of execution traces.
- A **literature review** on studies that used static and dynamic data analysis for software comprehension. Summarize and compare studies that adopt the strengths of the two methods and minimize the effect of their weaknesses.
- An approach to **reduce large** amount of data contained in **execution traces** and to acquire sufficient knowledge about the system by identifying crucial program artefacts.
- An approach to **visualize condensed software execution traces**.

.