

Project

“Software optimization is the process of modifying a software system to make some aspect of it work more efficiently. In general, a computer program may be optimized so that it executes more rapidly. Optimization may include finding a bottleneck, a critical part of the code that is the primary consumer of the needed resource.” — Wikipedia

The goal of the project is to give you a taste of what it means to build a software analysis tool. You will use and combine the techniques learnt during the course: static analysis, dynamic analysis, and visualisation. The project is to find the most time-consuming methods in a given Java system.

Project Roadmap

The project will be graded with a maximum of 30 points. The points will be distributed across the following steps:

1. Static analysis
 - (a) Statically create a call-graph of a given Java project, using the CHA algorithm and Moose
 - (b) Visualise the call-graph using Roassal, for example, as in Figure 1
 - (c) Produce a list of methods from the system at hand, as in Table 1

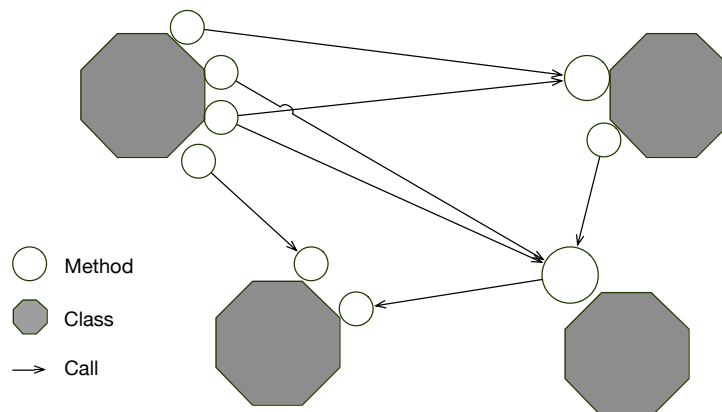


Figure 1: Call-graph example

2. Dynamic analysis

- (a) Dynamically create a call-graph of a given Java project. Using any instrumentation framework (like Javassist), collect the necessary data to build up the call graph. In other words, for each method invocation, you need the invoked method signature, the invoking method signature, and their containing classes. Also you should get for each method m the full execution time T_m , what is the time spent within the method from beginning to end. You should also report for each method m the time spent locally, T'_m , namely the time spent *just* inside that

Table 1: Example of statically-produced list of methods. This list is ordered descendingly based on LOC*NOIC

Method Full Name	Number of Lines of Code (LOC)	Number of Incoming Calls (NOIC)	LOC * NOIC
scg.parsing.Parser.parse(String)	213	5	1065
scg.utilities.Dispatcher.spawn(Thread)	75	10	750
.			
.			
.			
scg.main.MainClass.main(String[])	17	0	0

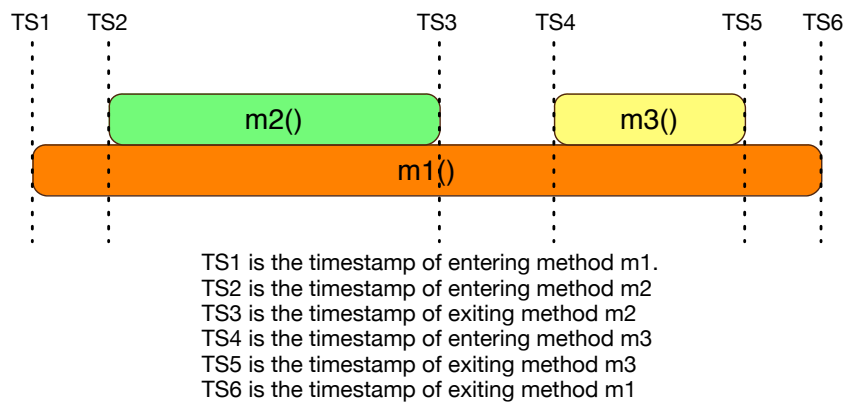
method without considering the time spent inside called methods, as in Figure 2.

$$T_{m1} = TS6 - TS1$$

$$T'_{m1} = TS6 - TS1 - T_{m2} - T_{m3}$$

$$T_{m2} = TS3 - TS2$$

$$T_{m3} = TS5 - TS4$$

Figure 2: the methods $m2()$ and $m3()$ are invoked from the method $m1()$

- (b) Refine the statically produced call graph with the dynamically produced information.
- (c) Produce a list of methods from the system at hand as in Table 2 using the collected data from your instrumentation.
- (d) Produce the same list of methods using an off-the shelf profiler (hprof, xprof, profile).

3. Result analysis

- (a) Compare the lists of methods using Spearman Correlation (Statically-produced list, Dynamically-produced list, and the profiler-produced list)

Table 2: Example of dynamically-produced list of methods. This list is ordered descendingly based on T'

Method Full Name	T'_m (m.s.)	T_m (m.s.)	Number of Outgoing Calls
scg.parsing.Parser.parse(String)	350	500	3
scg.utilities.Dispatcher.spawn(Thread)	200	230	2
.			
.			
.			
scg.main.MainClass.main(String[])	14	3000	2

- (b) Make sense of the results by comparing the types of analysis and by diving into the source code when necessary. For instance, if a method appears in the top 10 in one list and it does not in the other list, see why does that happen in terms of code.

Final Report

You should provide a brief description (1-3 pages) in which you report:

- Design decisions.
- Interesting problems and solutions.
- Limitations.
- Any extra information you want us to know.