

**National University of Singapore**

**School of Computing**

**CS3202: Software Engineering Project II**

**TEAM 05: Flying Cockroach**

Semester 1, AY2014/2015

|  |  |  |  |
| --- | --- | --- | --- |
| **Matriculation Number** | **HP Number** | **Student Name** | **Email** |
| **Group-PQL:** | | | |
| A0099214B | 8518 2707 | Adinda Ayu Savitri | savitri.adinda@gmail.com |
| A0098139R | 9082 0864 | Hisyam Nursaid Indrakesuma | indrakesuma.hisyam@gmail.com |
| A0103494J | 9620 7018 | Lacie Fan Yuxin | lacie.jolene.fan@gmail.com |
| **Group-PKB:** | | | |
| A0101286N | 9833 2474 | Ipsita Mohapatra | ipsita@nus.edu.sg |
| A0080415N | 9148 6248 | Steven Kester Yuwono | a0080415@nus.edu.sg |
| A0099768Y | 9178 6540 | Yohanes Lim | yohaneslim93@gmail.com |

Consultation Day/Hour: Monday 6-6.30pm

Contents

[1. SPA 3](#_Toc400895570)

[1.1. Architecture 3](#_Toc400895571)

[1.2. Development Plan 4](#_Toc400895572)

[1.2.1. For Whole Project 4](#_Toc400895573)

[1.2.2. For Iteration 3 5](#_Toc400895574)

[2. Components 6](#_Toc400895575)

[2.1. Code Parser 6](#_Toc400895576)

[2.2. PKB 7](#_Toc400895577)

[2.3. Design Extractor 8](#_Toc400895578)

[2.4. Query Processor 9](#_Toc400895579)

[2.4.1. Query Processor 9](#_Toc400895580)

[2.4.2. Query Parser 9](#_Toc400895581)

[2.4.3. Query Evaluator 9](#_Toc400895582)

[3. Testing 10](#_Toc400895583)

[3.1. Testing Plan For Iteration 3 11](#_Toc400895584)

[3.2. System Testing 11](#_Toc400895585)

[4. API 11](#_Toc400895586)

# 1. SPA

Static Program Analyser (SPA) is a program to answer queries about an input SIMPLE program. In this report, we will be describing the design and implementation decisions made during the development of the basic SPA and any bonus features.

## Architecture

The architecture consists of 3 main components: the Code Parser, the PKB and the Query Processor. Code Parser parses the code and stores design abstractions in each of the 8 tables in the PKB. After Query Parser has parsed the query, the Query Evaluator consults the PKB API to answer queries.

Both the Code Parser and the Query Processor are dependent on PKB but not dependent on each other. This ensures low coupling between components within SPA. At the same time there is high cohesion within each component, as described in the subsequent parts.



Figure 1

## 1.2. Development Plan

### 1.2.1. For Whole Project

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Iteration 1** | | | | | **Iteration 2** | | | |
| Team member | **Implement Calls in PKB** | **Implement Modifies and Uses in PKB** | **Extend Query Parser to support with clause** | **Extend Query Evaluator to support with clause** | **Write system test cases for the enhancement** | **Implement Next in PKB** | **Extend Query Parser to support multiple clauses** | **Extend Query Evaluator to support multiple clauses** | **Write system test cases for the enhancement** |
| Adinda |  |  |  |  |  |  |  |  |  |
| Lacie |  |  |  |  |  |  |  |  |  |
| Hisyam |  |  |  |  |  |  |  |  |  |
| Steven |  |  |  |  |  |  |  |  |  |
| Ipsita |  |  |  |  |  |  |  |  |  |
| Yohanes |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Iteration 3** | | | |
| Team member | **Implement Affects in PKB** | **Extend Query Parser to support Affects and tuple results** | **Extend Query Evaluator to support Affects and tuple results** | **Write system test cases for the enhancement** |
| Adinda |  |  |  |  |
| Lacie |  |  |  |  |
| Hisyam |  |  |  |  |
| Steven |  |  |  |  |
| Ipsita |  |  |  |  |
| Yohanes |  |  |  |  |

### 1.2.2. For Iteration 3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Iteration 3** |  |  |  |  |  |  |  |  |
| Team member | **Data hiding for PKB components and method simplification** | **Revamp PKB data structures to improve performance** | **Next Star optimization by implementing nextPair relationship** | **Unit testing for nextPair** | **Extend Query Parser to support Affects and tuple results** | **Implementsemantic check and select statement filter** | **Extend Query Evaluator to support Affects and Affects\*** | **Extend Query Evaluator to support query tuple processing and projection** | **Write system test cases for the enhancement** |
| Adinda |  |  |  |  |  |  |  |  |  |
| Lacie |  |  |  |  |  |  |  |  |  |
| Hisyam |  |  |  |  |  |  |  |  |  |
| Steven |  |  |  |  |  |  |  |  |  |
| Ipsita |  |  |  |  |  |  |  |  |  |
| Yohanes |  |  |  |  |  |  |  |  |  |

# 2. Components

## 2.1. Code Parser

## 2.2. PKB

### 2.2.1. Design Decisions

In this iteration, the main focus of PKB is the data hiding, method simplifications and revamping the data structure.

Data hiding

Previously, other methods can get the individual classes inside PKB (such as Parent class) and utilise their method. This interaction has been changed and now other classes can only use the methods that are provided in the PKB API. They will not be able to get the individual classes anymore.

Method Simplifications

Previously, there were many methods in PKB that were superfluous and outside of the scope of the PKB. This will not only introduce unnecessary bug but also make it harder for the system to be optimised. As a result, a lot of methods have been trimmed down. Please refer to the PKB API to check the comprehensive methods list that PKB will now provide.

Revamping of Data Structure

The main focus of the revamping of data structure is speed. Therefore, PKB will now mainly use vector and bit array where the searching is mostly O(1).

|  |  |
| --- | --- |
| **PKB – Design Abstractions** | |
| **Tables** | **Data Structures** |
| ProcTable | vector<PROCNAME> and map<PROCNAME,PROCINDEX> |
| ConstTable | vector<CONSTVALUE> |
| TypeTable | vector<SynType> and map<SynType,vector<STMTNUM>> |
| VarTable | vector< VARNAME > and map< VARNAME, VARINDEX > |
| **Relationships** | **Data Structures** |
| Follows | vector<STMTNUM> |
| Parent | vector<vector<int64\_t>> and vector<STMTNUM> |
| Uses | vector<vector<int64\_t>> |
| Modifies | vector<vector<int64\_t>> |
| Calls | vector<vector<int64\_t>> |
| Next | vector<vector<STMTNUM>> and vector<vector<pair<STMTNUM,STMTNUM>>> |

For the tables, getting the value by index will be in O(1). Meanwhile, getting value by the name will be done in reverse mapping using map in O(log n). Map is used to avoid the worst case scenario of unordered\_map which is O(n).

For the relationships, here are the design decisions.

1. Follows

Follows will only need to record two statement numbers in each entry. One forward mapping and one reverse mapping are all that is required which can be achieved using vector. All search operations can be done in O(1).

1. Parent

For the parent to children mapping, we will use bit array. This application of bit array is exploited with the fact that the children will always be after the parent. Therefore the bit array will store the number after the parent’s statement number. In terms of storage, the number of int64\_t that will be stored is dependent on = (last children statement number – parent statement number) / 63. In most cases, it will be mostly one which is space efficient. This application will make the searching of specific parent and children combination much faster at O(1). The speed of the rest of the operations will be the same if we were to use vector. In addition, the reverse mapping of children to parent is also provided using vector which can be done in O(1).

1. Uses, Modifies and Calls

Uses, Modifies and Calls will all use bit arrays (and the reverse mapping as well). Using bit array will not only save memory but it will also speed up the searching compared to normal vector. Searching of a specific combination will be done in O(1) while listing down of all the values for a given index will be done in O(k) where k is the size of the answer.

1. Next

For next, we will be using vector. We will also be using a separate table that stores a pair of ranges which is optimised for the Query Processor.

## 2.3. Design Extractor

The main role of the design extractor is to extract relationships about the SIMPLE program that could not be extracted in the one-time parsing done by the Code Parser. This includes:

* + Extracting information about Modifies and Uses for procedures and for program lines that are calls statements
  + Building the Control Flow Graph (CFG) from the AST and subsequently storing it in the PKB and storing the Next relationship in the PKB.

## 2.4. Query Processor

Query Processor consists of three parts: Query Processor (controller), Query Parser, and Query Evaluator.

### 2.4.1. Query Processor

Query Processor is a façade class for the whole component. Its responsibilities include:

1. Query Processor calls QueryParser to create a Query object from the given query string.
2. Query Processor then passes the Query object to the QueryEvaluator.
3. Query Evaluator will compute all necessary relations and return the results in the form of a list of integers.
4. Query Processor transforms the result into the correct display format and returns the answer to the user.

### 2.4.2. Query Parser

### 2.4.3. Query Evaluator

# 3. Testing

We did testing on 3 different levels, namely unit testing (using CPPUnit), integration testing (using CPPUnit) and system testing (using AutoTester). Unit Testing was done while coding the components, while integration testing was done between SIMPLE program parser and PKB and between PKB and Query component.

From the testing experience in this project, we realised the need for timely and consistent unit, integration and system testing. By testing individual components early, we detect bugs earlier in the project’s lifetime, thus, saving us time towards the end of the project. We also did regression testing by reusing our unit tests and system tests. This helped us to quickly identify bugs that could have been introduced while we were trying to solve other bugs.

## 

## 3.1. System Testing Plan for Iteration 3

The following tables illustrate the system test cases used during Iteration 3

|  |  |  |  |
| --- | --- | --- | --- |
| **Level** | **Function** | **Parameters** | **Select** |
| Basic | Affects | #, a2 | a2 |
|  |  | #, a2 | boolean |
|  |  | a1, # | a1 |
|  |  | a1, # | boolean |
|  |  | #, # | boolean |
|  |  | a1, a2 | a1 |
|  |  | a1, a2 | a2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Level** | **Function** | **Parameters** | **Select** |
| Basic | Affects\* | #, a2 | a2 |
|  |  | #, a2 | boolean |
|  |  | a1, # | a1 |
|  |  | a1, # | boolean |
|  |  | #, # | boolean |
|  |  | a1, a2 | a1 |
|  |  | a1, a2 | a2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Level** | **Function** | **Parameters** | **Select** |
| Intermediate | Tuple | Calls\*(p1, p2) | <p1, p2> |
|  |  | Calls(p1, p2) | <p2, p1> |
|  |  | Modifies\*(s1, s2) | <s1, s2> |
|  |  | Modifies(s1, s2) | <s2, s1> |
|  |  | Modifies\*(p1, s1) | <p1, s1> |
|  |  | Modifies(p1, s1) | <s1, p1> |
|  |  | Uses\*(s1, s2) | <s1, s2> |
|  |  | Uses(s1, s2) | <s2, s1> |
|  |  | Uses\*(p1, s1) | <p1, s1> |
|  |  | Uses(p1, s1) | <s1, p1> |
|  |  | Parent\*(s1, s2) | <s1, s2> |
|  |  | Parent(s1, s2) | <s2, s1> |
|  |  | Follows\*(s1, s2) | <s1, s2> |
|  |  | Follows(s1, s2) | <s2, s1> |
|  |  | Next\*(n1, n2) | <n1, n2> |
|  |  | Next(n1, n2) | <n2, n1> |
|  |  | Affects\*(a1, a2) | <a1, a2> |
|  |  | Affects(a1, a2) | <a2, a1> |

|  |  |  |
| --- | --- | --- |
| **Level** | **Function** | **Clause** |
| Intermediate | Multiple clause using Affects | Modifies\* |
|  |  | Uses\* |
|  |  | Parent\* |
|  |  | With |

|  |  |  |
| --- | --- | --- |
| **Level** | **Function** | **Clause** |
| Intermediate | Multiple clause using Affects\* | Follows\* |
|  |  | Affects |
|  |  | With |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Level** | **Function** | **Clause 1** | **Clause 2** | **Select** |
| Advanced | Multiple clause using tuple | Calls\*(p1, p2) | Modifies\*(p1, s1) | <p1, p2> |
|  |  | Calls\*(p1, p2) | Modifies\*(p1, s1) | <p1, s1> |
|  |  | Modifies\*(p1, s1) | Uses\*(s1, s2) | <p1, s2> |
|  |  | Uses\*(p1, s1) | Parent\*(s1, s2) | <s1, s2> |
|  |  | Next(n1, n2) | Next\*(n2, n3) | <n1, n3> |
|  |  | Follows\*(a1, a2) | Affects\*(a1, a2) | <a1, a2> |
|  |  | Calls\*(p1, p2) | Uses\*(s1, s2) | <p1, s2> |

# 4. API

Please view our Doxygen at:

[www.comp.nus.edu.sg/~kester/CS3202](http://www.comp.nus.edu.sg/~kester/CS3202)