

**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](#_Toc402389959)

[1.1. Architecture 3](#_Toc402389960)

[1.2. Interaction 4](#_Toc402389961)

[2. Summary of Achievements 6](#_Toc402389962)

[2.1. Basic SPA 6](#_Toc402389963)

[2.2. Bonus Features 7](#_Toc402389964)

[2.2.1. Flexible CodeParser 7](#_Toc402389965)

[2.2.2. Flexible QueryParser 8](#_Toc402389966)

[2.2.3. Highly Organised Repository 9](#_Toc402389967)

[3. Project Plan 13](#_Toc402389968)

[3.1. Schedule For Whole Project 13](#_Toc402389969)

[3.2. Comments & Problems 13](#_Toc402389970)

[4. Components 14](#_Toc402389971)

[4.1. Code Parser 14](#_Toc402389972)

[4.2. PKB 15](#_Toc402389973)

[4.3. Design Extractor 16](#_Toc402389974)

[4.4. Query Processor 17](#_Toc402389975)

[4.4.1 Query Processor 17](#_Toc402389976)

[4.4.2 Query Parser 17](#_Toc402389977)

[4.4.3 Query Evaluator 18](#_Toc402389978)

[5. Testing 20](#_Toc402389979)

[6. Coding Standards 21](#_Toc402389980)

[7. Project Evaluation 22](#_Toc402389981)

[8. API 23](#_Toc402389982)

# 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 SPA during CS3201 and CS3202.

## Architecture

The architecture for the prototype consists of 3 main components: the Code Parser, the PKB and the Query Processor. Both the Code Parser and the Query Processor are dependent on PKB but not dependent on each other. 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.



Figure 1

## Interaction

/\* **DINDA**

PLEASE REMOVE THIS COMMENT

Draw UML diagrams that you found useful. For each diagram that you draw, explain how you used it (e.g., in project planning, communication, test planning or in other situations), and comment on the value a diagram added to your project.

\*/

CodeParser works by evaluating each line of the given source code. It creates AST Node, set the pointers accordingly; set the tables and the appropriate databases in PKB.

The attributes in PKB (the tables) will then be used by Query evaluator to answer queries. Testing for CodeParser is done by checking the content of each table, whether it has set the values properly, and check the content of each node in the AST, whether it matches the correct AST.



Figure 2

Figure 3 shows the sequence diagram of query evaluation process. This diagram was useful in demarcating the responsibilities of each PQL group member. For example, QueryEvaluator directly assumes that the Query it receives is valid and syntactically correct. Therefore it is the responsibility of QueryParser to validate each query before passing it to the evaluator.



Figure 3

This diagram also helps to keep track of the dependencies between components. This is especially useful during debugging process of integration testing. When QueryProcessor fails to return the correct result, the team knows that the errors could come from at least three places, i.e. QueryParser, QueryEvaluator, and PKB.

# Summary of Achievements

## Basic SPA

/\***IPSITA**\*/

## Bonus Features

/\***KESTER**\*/

* + 1. Flexible CodeParser

Our Static Program Analyser (SPA) has been extended beyond its call of duty. The extended (extra) features will be explained in more details below:

The source code tested for the SPA, is defined to be in standardized format and neatly arranged. They have regular and consistent spacing, indentation, tabs, and endline characters. Please refer to the example below:

procedure Orchid {

while idx {

y = z\*3 + 2\*x;

call Tulip;

idx = idx - 1; }

z = z + y + idx; }

procedure Lily {

while v {

y = x + y; }

x = y + x; }

However, our SPA source code parser is very flexible. It is able to detect erroneous and inconsistent spaces, tabs, endline characters and erroneous close curly brackets. A flexible CodeParser is one of our extended features, as it is not required to build one for this project. To demonstrate the ability of our CodeParser, please refer to the example below (which will have the same information/result as the example above):

procedure Orchid

{

while idx {

y = z \*3 + 2\* x ;

call Tulip ;

idx = idx -1;

}

z=z+y+idx;

}

procedure Lily{

while v{

y =x + y ;}

x = y + x;}

### Flexible QueryParser

The queries being tested with SPA in this module are defined to be in standard format. They have regular and consistent lowercase/uppercase command, spacing, and characters. Please refer to the sample standard query below:

if ifstat; Select ifstat such that Follows\* (ifstat, 17)

assign a; Select a such that Modifies (a, "idx") and Uses (a, "idx")

assign a; while w; Select a such that Modifies (a, "idx") and Uses (a, "idx") and Follows (15, a) and Parent\* (w, a)

However, our SPA QueryParser is very flexible and able to detect queries with inconsistent/extra spaces and lowercase/uppercase clauses. It is not required to build a flexible QueryParser for this project. To demonstrate the ability of our QueryParser, please refer to the example below (which will have the same information/result as the example above):

If ifstat ; Select ifstat such that Follows\* ( ifstat,17)

assign a;Select a such that Modifies(a,"idx") and Uses(a,"idx")

assign a ; while w ; Select a such that Modifies (a, "idx" ) and Uses ( a , "idx") and Follows (15, a) and Parent\* ( w , a )

### Highly Organised Repository

Our codes and files in the repository are very organized. We follow standard team naming conventions and structure the directory in the repository to make sure every team member understands and has easy access.



Appropriate milestones and issues are also tracked and reported regularly to achieve any goals or/and objectives.





Every commit related to certain issues or milestones will be tagged and linked to the issue for easy tracking and reference in the future. It is also used to monitor the progress of any issues and the SPA system as a whole. Github features such as milestones, issue tracker with assignees, milestones, and issue category (bugs, testing, documentation, enhancement, etc) are highly utilized by our team.





Our API and code is also well-documented. Every detail of the code is described clearly, showing and explaining all methods, attributes, and inheritance diagram whenever applicable.



In conclusion, our progress in building this project is thoroughly tracked and documented so that anyone in the future will be able to understand project with ease.

Our project is hosted on GitHub, and the website links are as follows:

* Repository : <https://github.com/yulonglong/Static-Code-Analyzer>
* Issue tracker : <https://github.com/yulonglong/Static-Code-Analyzer/issues>
* Milestones : <https://github.com/yulonglong/Static-Code-Analyzer/milestones>
* Commits : <https://github.com/yulonglong/Static-Code-Analyzer/commits/master>

# Project Plan

/\* **DINDA**

PLEASE REMOVE THIS COMMENT

Describe how you organized project work, the actual schedule, etc. Organize your description into the following sub-sections:

**The actual schedule for the project, milestones**

Discuss problems encountered that affected project schedule.

**Any comments on division of work and project discussion meetings**

**\*/**

## 3.1. Schedule For Whole Project

## 3.2. Comments & Problems

# 4. Components

## 4.1. Code Parser

/\***KESTER**

PLEASE REMOVE THIS COMMENT

Follow guidelines in Handbook Section 10.2 to analyze, justify and document detailed design decisions. Pay attention to clarity of the description (check hints in Section 10.2).

If you applied design patterns, document them in this section:

a) Explain the design problem and pattern you applied

b) Document expected benefits and costs of applying a design pattern

c) Document the actual benefits and costs of a design pattern that you experienced in the project after applying it.

\*/

## 4.2. PKB

/\***YOLIM**

PLEASE REMOVE THIS COMMENT

Follow guidelines in Handbook Section 10.2 to analyze, justify and document detailed design decisions. Pay attention to clarity of the description (check hints in Section 10.2).

If you applied design patterns, document them in this section:

a) Explain the design problem and pattern you applied

b) Document expected benefits and costs of applying a design pattern

c) Document the actual benefits and costs of a design pattern that you experienced in the project after applying it.

\*/

## 4.3. Design Extractor

/\***IPSITA**

PLEASE REMOVE THIS COMMENT

Follow guidelines in Handbook Section 10.2 to analyze, justify and document detailed design decisions. Pay attention to clarity of the description (check hints in Section 10.2).

If you applied design patterns, document them in this section:

a) Explain the design problem and pattern you applied

b) Document expected benefits and costs of applying a design pattern

c) Document the actual benefits and costs of a design pattern that you experienced in the project after applying it.

\*/

## 4.4. Query Processor

Query processor consists of three parts: query processor (controller), query parser, and query evaluator.

### 4.4.1 Query Processor

/\***HISYAM**

\*/

Query Processor is a façade class for the whole component. The following shows the steps it takes:

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.

### 4.4.2 Query Parser

/\***KESTER**

PLEASE REMOVE THIS COMMENT

Describe query validation rules, only in case there is some difference as compared to what you described in your previous assignment. An example of query validation rule is: “checking if all relationships have correct number and types of arguments, as defined in PQL definition in Handbook”. DO NOT provide procedural description (pseudocode) of how Query Pre-processor checks the rules.

If you use table-driven approach to query validation – show the structure of your tables.

Follow guidelines in Handbook Section 10.2 to analyze, justify and document detailed design decisions. Pay attention to clarity of the description (check hints in Section 10.2).

If you applied design patterns, document them in this section:

a) Explain the design problem and pattern you applied

b) Document expected benefits and costs of applying a design pattern

c) Document the actual benefits and costs of a design pattern that you experienced in the project after applying it.

\*/

### 4.4.3 Query Evaluator

/\***LACIE**

PLEASE REMOVE THIS COMMENT

1. Describe data representation for program queries

2. Describe your strategy for Basic Query Evaluation (BQE)

3. Describe optimizations

4. Discuss detailed design decisions regarding BQE and optimizations

\*/

/\***HISYAM**

PLEASE REMOVE THIS COMMENT

1. Describe data representation for program queries

2. Describe your strategy for Basic Query Evaluation (BQE)

3. Describe optimizations

4. Discuss detailed design decisions regarding BQE and optimizations

\*/

# 5. Testing

/\* **DINDA**

PLEASE REMOVE THIS COMMENT

**Testing: Group-PKB and Group-PQL**

Describe your testing experience (not ex ceeding TWO pages).

\*/

# 6. Coding Standards

/\***IPSITA**

Include experiences

\*/

Our team members adopted similar coding standards which are adjusted appropriately and respectively according to the design specifications of various components. Some of the coding standards that the components possess are listed below:

1. Indentation and whitespace
   1. a. Indication of code segments
2. Comments to enhance understanding and communication
3. Descriptive variable declarations
   1. Always start with lower case
   2. Use CamelCase
   3. Use only letters and numbers
4. Informative function naming conventions
   1. All getters start with “get”
   2. All setters start with “set”
   3. All functions that start with “is” returns a Boolean value
5. Keep it simple and effective
   1. Avoid complex code fragments
6. Refactoring

**Standards between abstract APIs and concrete APIs**

The correspondence between the abstract and concrete APIs was enhanced by doing the following:

1. The abstract APIs provides the interface for the concrete APIs
2. Making abstract APIs as comprehensive as possible a. Offering an Extensive description of the abstract APIs b. Specifying the complete parameters needed for the function

# 7. Project Evaluation

/\***YOLIM** \*/

1. How would you improve your SPA if more time was available?

2. What would you done differently if you were to start project again?

3. Comment on the experience gained in this project in respect to: a) working in the team,

b) incremental development,

c) complexity of the SPA problem and program solution,

d) what did work well?

e) what did not work well?

f) what did you learn in this project course?

4. Comment on the tools used for the project a) Were the recommended tools useful?

b) What other tools did you use (if any), and in what ways were they useful?

c) What were the problems you faced when using each tool?

d) In which areas would you like to have had more tool support?

/\***IPSITA** \*/

5. What management lessons have you learned?

6. What advice would you give to the students who will take this course in the future?

7. Suggest how we could improve this project course.

8. Discuss any other experiences.

9. Comment on Handbook

# 8. API

Please view our Doxygen at:

[www.comp.nus.edu.sg/~kester/cs3202](http://www.comp.nus.edu.sg/~kester/cs3202)