CS3215 (2010) Assignment 3: Development iteration 1

Due date: Monday, March 7, by 12 noon, to your supervisor

Deliverables: Integrate descriptions produced by team members and submit one coherent document per team, organized in the format given in the Required Format for Assignments and Final Project Report. You will also present your solution during the consultation hour immediately after the assignment due.

1. Describe project plans

Describe the plan for the whole project and for this development iteration.

1.1. *Plan for the whole project*

Plan who will be doing what during the project. *Tasks* correspond to considerably large work units such as: design parser for subset of SIMPLE, design PKB API, etc. Define tasks of such size that you feel comfortable with planning.

	iteration 1			iteration 2			iteration 3		
team member	task 1	task2	task 3	task 1	task2	task 3	task 1	task2	task 3
Mary	*		*						
John		*						*	
Tom									
Suzan						*			
Jack									
Jill									

1.2. *Plan for this development iteration*

Plan who will be doing what during this development iteration. *Activity* is a smaller work unit than task, for example, you may have an activity such as test interface to AST. Tasks consist of activities. Define activities of such size that you feel comfortable with planning.

	iteration 1				
team member	activity 1	activity 2	activity 3	activity 4	
Mary	*		*		
John		*			
Suzan					
Jack					
Jill					

2. SPA implementation

Feel free to extend the suggested scope of implementation (without compromising the quality!).
If you want to change the scope of implementation (narrow in certain areas and extend in other areas), you can do so but please discuss your plan with supervisor.

The following is suggested scope of implementation for this iteration:

Implement the following subset of SIMPLE:

```
program : procedure +
procedure : ‘procedure’ proc_name ‘{‘ stmtLst ‘}’
stmtLst : (stmt )+
stmt : call  |  while | if | assign
call : ‘call’ proc_name ‘;’
while : ‘while’ var_name ‘{‘ stmtLst ‘}’
if : ‘if’ var_name ‘then’ ‘{‘ stmtLst ‘}’ ‘else’ ‘{‘ stmtLst ‘}’
assign : var_name ‘=’ var_name | const_value  ‘;’
```

Query sub-system to be implemented in this iteration

a) Queries should involve the following relationships:

Follows, Follows*, Parent, Parent*
Modifies, Uses (for all statements)

b) Implement Query Pre-processor to validate queries and build a query tree.
c) Implement **Basic Query Evaluator** (see hint below)

A check list of query formats to be implemented in this iteration:

Select result-specs **such that** Follows (s1, s2)
result-specs includes just one element (not tuple)
arguments s1 and s2 should be synonyms (stmt, assign, call, while, if), '_', or INTEGER.

Select result-specs **such that** Follows (s1, s2) **with** s2.stmt# = n
where n should be INTEGER or reference to attribute value (attrRef, Appendix A)

Queries with Follows* as above
Queries with Parent and Parent* as above except that the first argument must be while or if.

Select result-specs **such that** Modifies (s, v)
result-specs includes just one element (not a tuple)
argument s should be synonym (stmt, assign, call, while, if) or INTEGER; argument v should be synonym, "IDENT" or "_"

Queries with Uses as above.

Your SPA should allow you to enter a source program, parse it and build PKB, allow you to enter queries and value for statement number n and or variable name v, answer queries, format query result, and display query result.

Hints for Query Evaluator:

First design the best possible Basic Query Evaluator (BQE) that can correctly evaluate any query without optimizations. Query involving many relationships must be evaluated incrementally, in steps. In the design of BQE pay attention to computing and representing intermediate query results. BQE should be prepared to employ various optimization strategies such as re-arranging the order in which to evaluate relationships in a query and many others. You will have to experiment with various optimization strategies that's why it is important that different optimization strategies can be plugged-into your BQE. BQE should not depend on any specific optimization strategy, but easily work with any optimization strategy you can think of.

Hints for implementation (excerpt from Handbook Section 9):

Use abstract PKB API to guide your implementation. Maintain full traceability between abstract PKB API and relevant classes in your program.
Follow these two guidelines:

1. Do not retrofit implementation details into abstract PKB API. Abstract PKB API will fail to play its role if you did that.
2. Use the same naming conventions in abstract PKB API and corresponding implementation classes (the same names in abstract operations/arguments and member functions). Use the same symbolic names for data types in abstract PKB API and corresponding implementation classes. 'typedef' can easily assign specific C++ data type to symbolic names.

Suggested SPA implementation activities:

1. Implement parser for required subset of SIMPLE
2. Implement AST: choose the data representation for AST and implement its interface operations. Justify your choice of data representation.
3. Implement symbol tables to store variables and procedures.
4. Implement procedure call table.
5. Implement relationships Modifies and Uses for statements:
6. Choose the data representation for Modifies and Uses ADTs. Justify your choice of data representation.
7. Implement the interface operations to Modifies and Uses ADTs
8. Make your parser create variable and procedure tables, build AST, and store Modifies/Uses information (for assignment statements only).

3. Refine and complete documentation of abstract PKB API

Complete abstract PKB API specifications for all the ADTs in the PKB. Include complete and up to date documentation of PKB API in each assignment report and Final Report.

3.1. Documentation standards for abstract PKB API

State naming conventions and other standards adopted in documenting abstract PKB API.

3.2. Abstract PKB API documentation

1. VarTable API
2. AST API
3. Modifies API
4. Uses API
5. ProcTable API
6. CallsTable API
7. CFG API
8. Any extra interfaces

In the last section, specify interfaces that do not fit into any of the above ADTs, for example: Affects, Affects*. You can include any other operations that you think may be useful (e.g., mappings among ADTs).

3.3. Evaluation

Comment on your abstract PKB API in view of recommendations discussed in Handbook Section 9.

4. UML diagrams

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.

Hint: Read through relevant parts of Handbook Section 10. Refine UML sequence diagrams given in examples into more detailed sequence diagrams showing communication between SPA functional components and specific data abstractions in PKB.

5. Documentation of important detailed design decisions

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

Address detailed design decisions related to data representations for design abstractions (ADTs) in PKB, any other solutions to speed up access to information stored in PKB, algorithms to fetch data from PKB during query evaluation, and any other issues that you consider important, except query evaluation (which is addressed in separate section).

Design patterns provide standardized solutions to design problems. You studied some typical design problems for which published design patterns exist in CS2103. By applying a design pattern, you usually win more flexibility, but an overall program solution may be more complex to work with. Always evaluate carefully the trade-offs involved in terms of expected benefits and the cost of applying a design pattern. Apply a design pattern only if the benefits outweigh the cost.

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.

6. Coding standards and experiences

1) State coding standards adopted by your team.

2) Comment on how you used abstract PKB API to guide design of relevant C++ classes and how you keep abstract and concrete PKB API in sync with other.

    a) Do you use same naming conventions in abstract PKB API and your C++ program?

   b) Do you use typdef to map type names used in abstract PKB API to C++ types?

3) Comment on any experiences - problems and benefits - that you observed working from abstract PKB API towards C++ program.

7. Query processing

7.1. Validating program queries

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.

7.2. Design and implementation of query evaluator

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

Hint: Follow guidelines in Handbook Section 10.2 to properly analyze, justify and document detailed design decisions. Pay attention to clarity of the description (check hints in Section 10.2). Do not repeat what you already discussed in Section 5, but refer to relevant points.

8. Testing: Group-PKB and Group-PQL

Be sure that you understand the role of Autotester released to you, integrate it with your SPA, and use it to automate regression testing throughout the project.

8.1. Describe your test plan for this iteration

8.2. Provide examples of test cases of different categories

1. unit testing:
   a) provide FIVE samples of specific unit test cases for PKB and FIVE test cases for PQL
   b) if you used assertions, describe how and show examples
2. integration testing
   a) UML sequence diagrams show communication among SPA components. Use sequence diagrams to plan integration testing and to indicate which component integrations you have tested.
   b) provide FIVE sample integration test cases
3. validation testing
   a) provide FIVE sample test cases

Document each test case in standard way as follows:
Test Purpose: explain what you intend to test in this test case
Required Test Inputs: explain what program module (or the whole system) you test and what input must be fed to this test case
Expected Test Results: specify the results to be produced when you run this test case
Any Other Requirements: describe any other requirements for running this test case; for example, to run a test case for a program module in isolation from the rest of the system, you may need to implement a simulated environment to call the test case.

9. Discussion

Discuss problems encountered that affected project schedule.

In which areas do you plan to improve in the next iteration?

Discuss any other project experiences and issues.

--- The End ---