# Testing

## Testing Experience

Testing of the SPA was carried out in three stages. Firstly, for each component of the PKB and PQL, we carried out unit testing. Unit testing tests the internal functions of the component. After unit testing was completed, we carried out integration testing of the different components. This ensures that the different components are working well together, for instance, the Parser and the Query Evaluator. Once we have tested that the important components work well together, we then test the system as a whole in validation testing.

The most vigorous type of testing that was carried out was validation testing. We tested the system with hundreds of test cases and many complex source codes.

## Three stages of testing

### Unit Testing

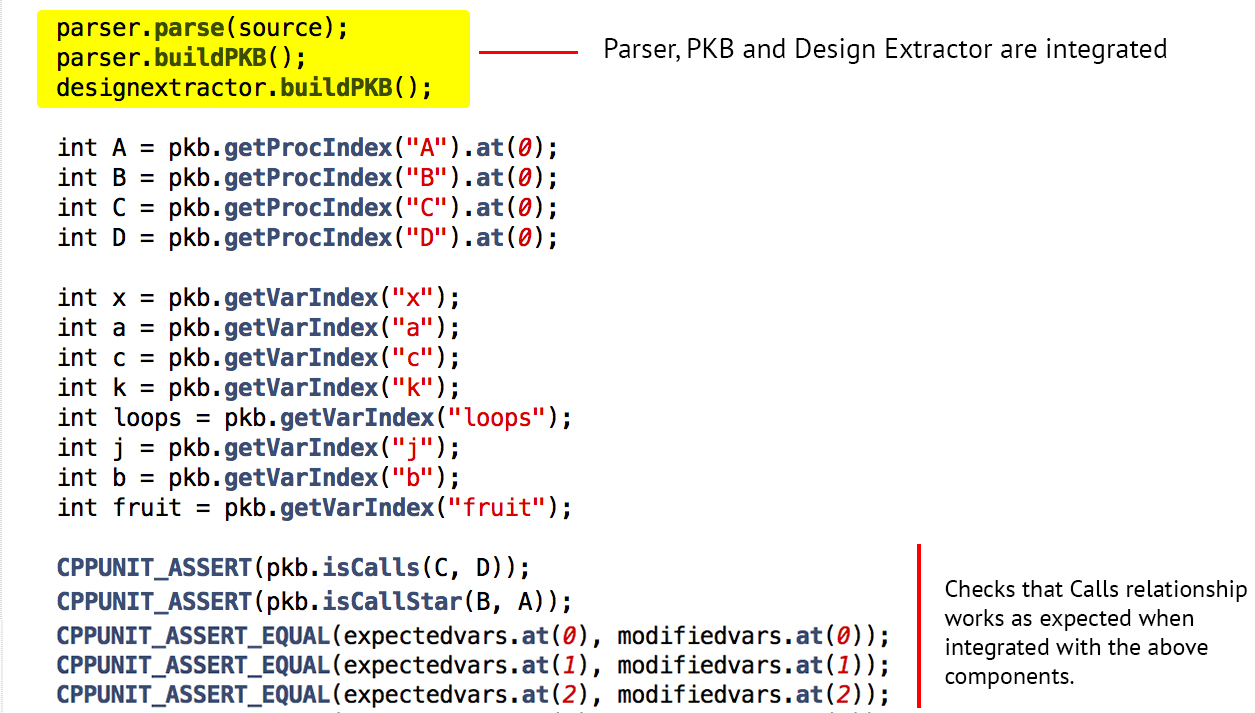
During unit testing, we discovered any logical errors present in the code, and this saved us the hassle of running into such errors during further stages of testing. Unit testing was done by manually inserting values and asserting that the function outputs were as expected.

The following example illustrates the unit testing of the ListTable component.

#### 

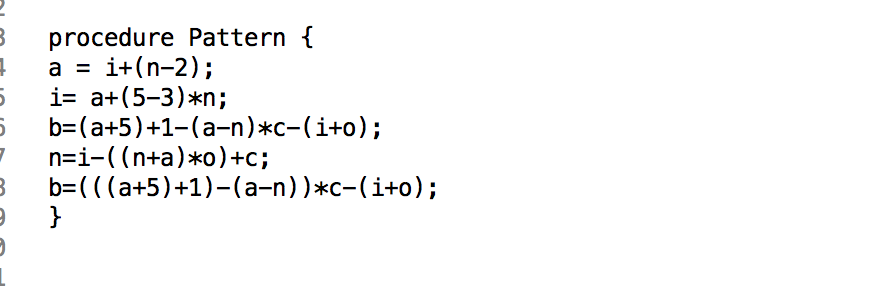
### Integration Testing

Many of the components work well when tested individually. However when they were integrated with other components, there were errors faced. This is due to the reason that the PKB, Parser and PQL were written by different members. Hence the components had slightly different methods of implementation and different expected inputs and outputs. During integration testing, these flaws between the components became apparent to us. The Parser parses the simple source code, which is provided as the input file, and the PKB is built from it. In the following example, the integration between these components, including the Design Extractor, is illustrated when testing the Calls relationship.

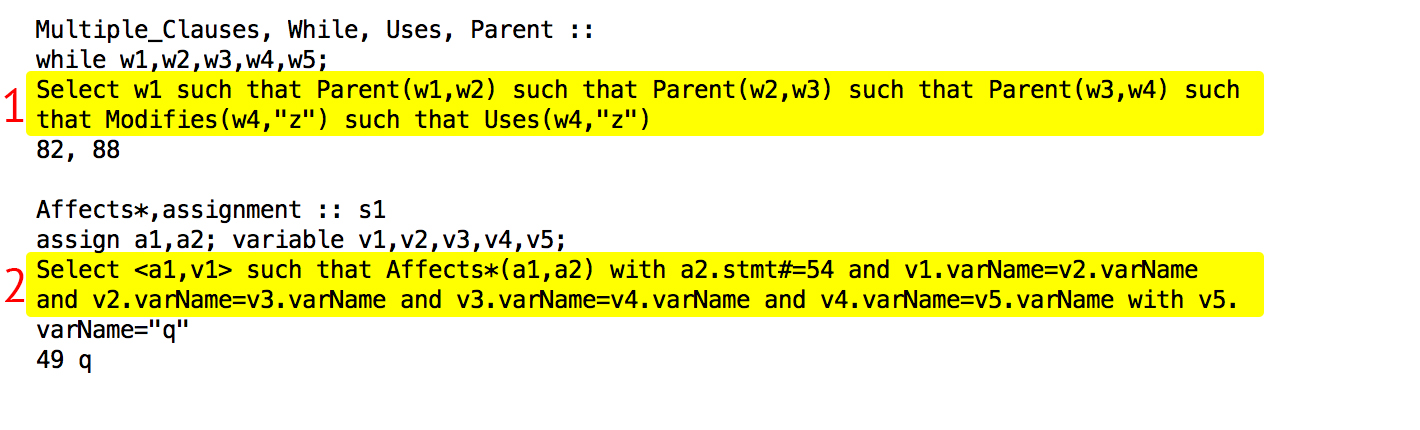


### Validation Testing

Validation testing, also known as System testing, is crucial to test the SPA system as a whole. There are many logical loopholes that we may have missed during unit and system testing that we can spot during extensive system testing. Thus, it is important to have a variety of test cases and source codes. The aim is to ensure that the system can handle queries of multiple complexities and parse hundreds of lines of source code which might be even haphazardly formatted.

For instance, the following Simple source code aims to test complex pattern queries by using multiple expressions and variables, the heavy use of brackets is also to be noted.

The figure below shows two query examples. The first query tests the combination of multiple relationship clauses. This tests that the evaluated results for the different clauses are merged correctly to output only the correct answers that satisfy the whole query. This is essential for evaluation of any query that has more than one clause. The second query focuses on selecting variables by their attributes, such as ‘stmt#’ and ‘varName’. This is a commonly used selection method in queries, hence it’s essential to ensure that all such possible attribute selections have been addressed by the system.



# Appendix B – Comments on Handbook

Overall, we found the handbook very helpful in giving us ideas for the implementation of the SPA. One thing that could have been better was if there had been more examples and perhaps some sample exercises for us to practice our concepts on. The extensions such as Contains and Siblings could also be explained briefly in the handbook instead of introducing them later on. This means we would have rough idea of how these relationships work beforehand and when it is time to consider whether or not to implement them, we can focus on finding the best way to integrate them rather than spending time understanding and grasping the new concepts of these relationships, which reduces crucial time which could be spent on the actual implementation.