

CS3213

Architecture Overview

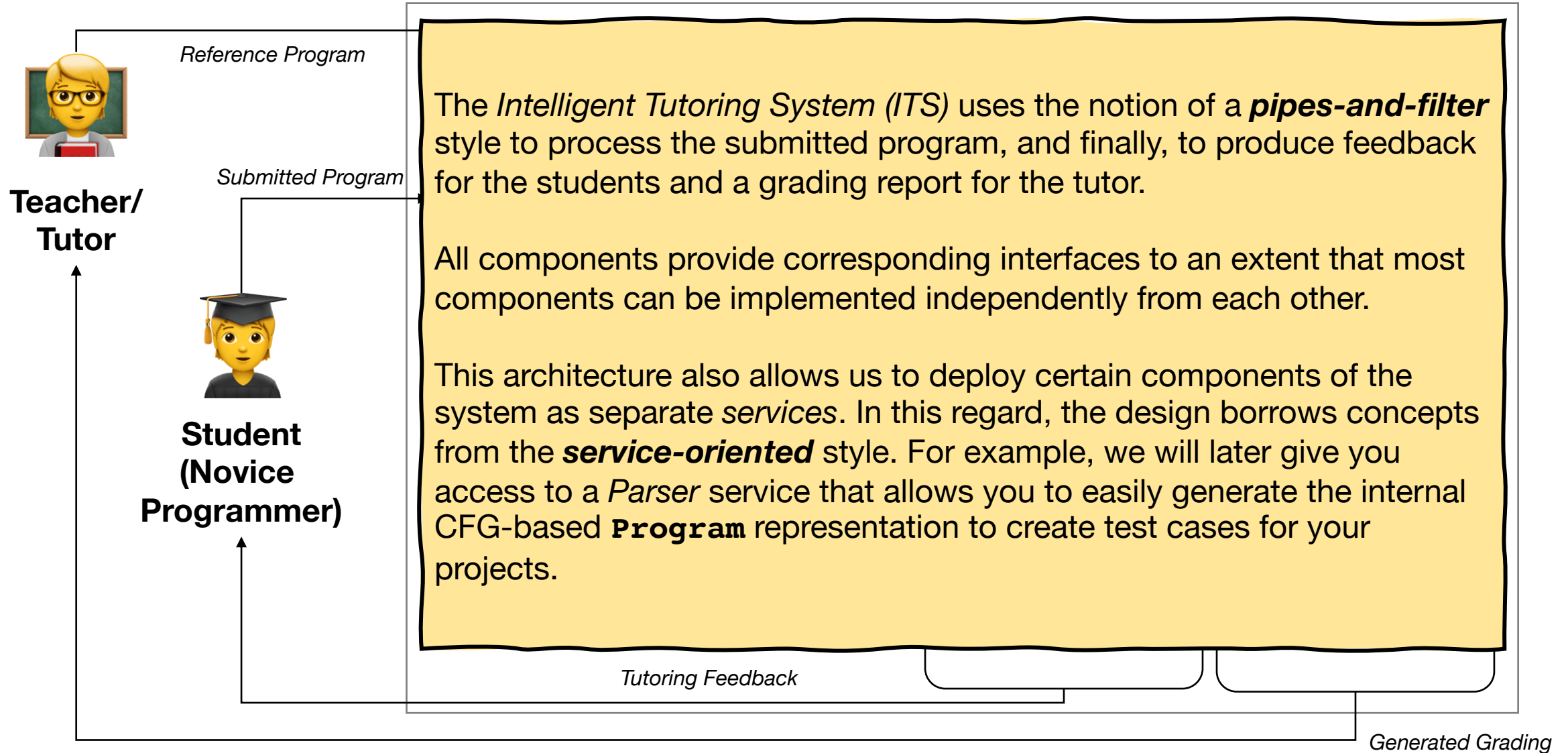
- ❑ Sample Workflow
- ❑ Components in the its-core

Sample Workflow

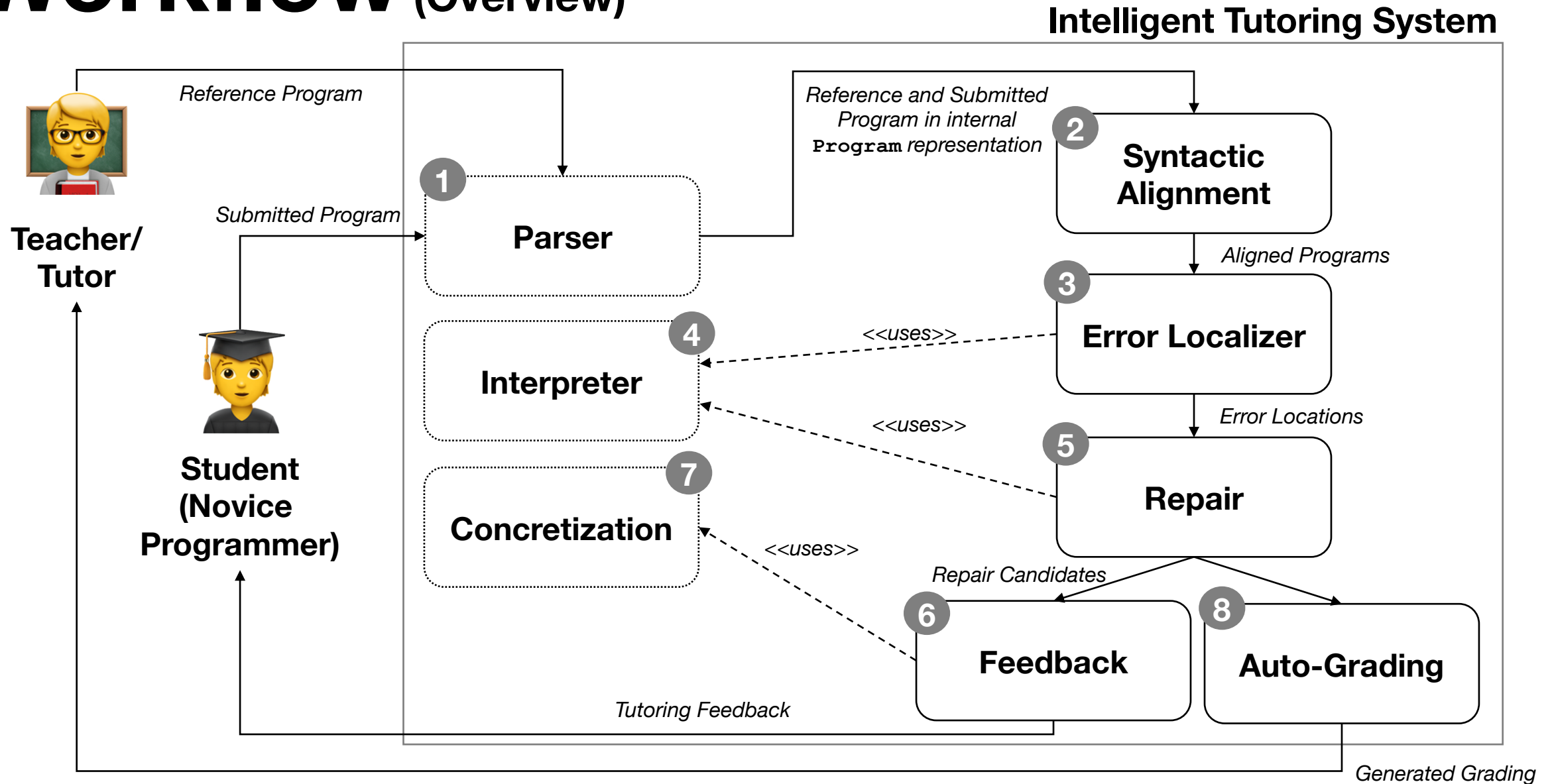
The next slides show the intended workflow through the *Intelligent Tutoring System (ITS)*. Note that there are points of variation (static and dynamic) that depend, e.g., on the programming language of the programming assignments and the intended repair strategies. Many of the current components can be implemented in many different ways.

Workflow (Overview)

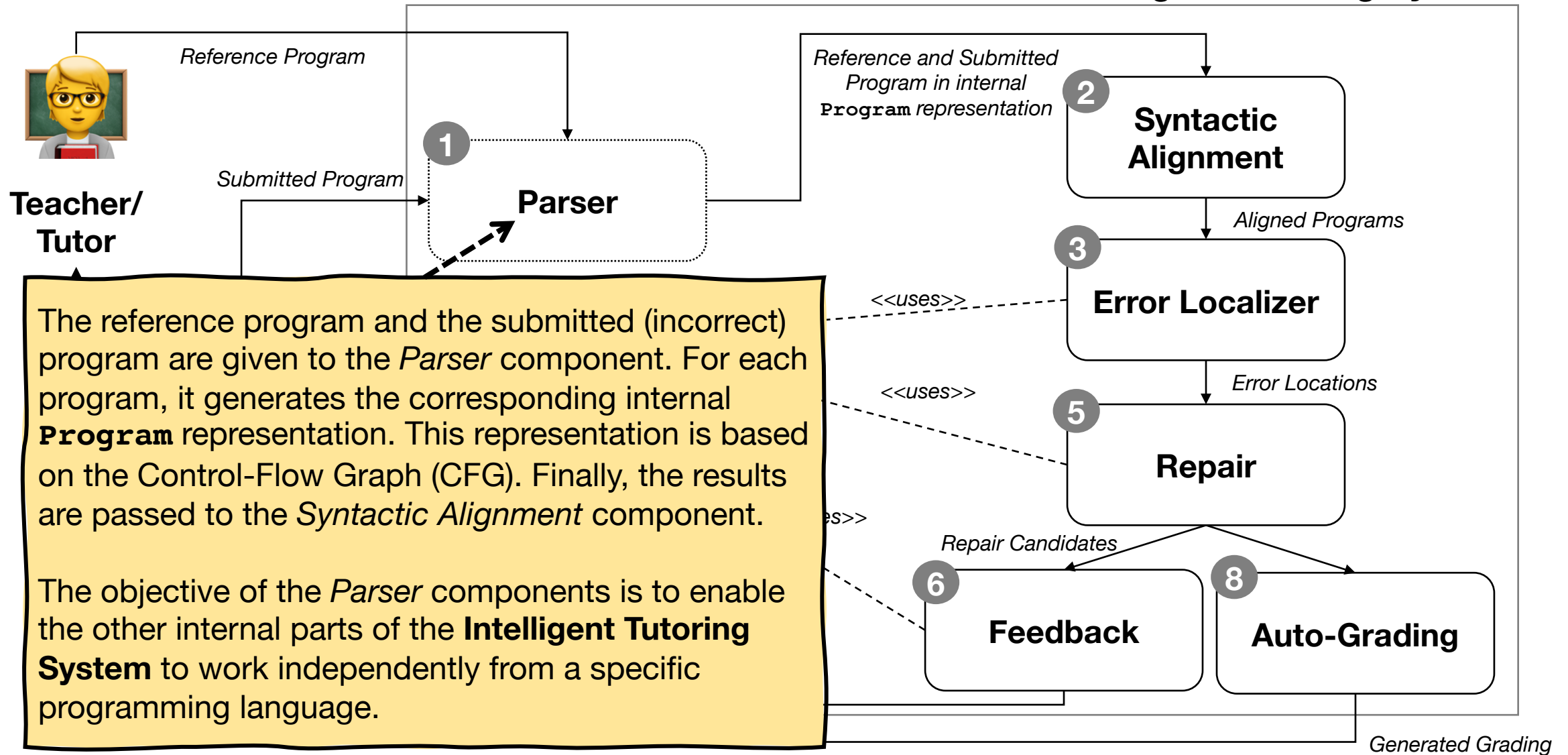
Intelligent Tutoring System



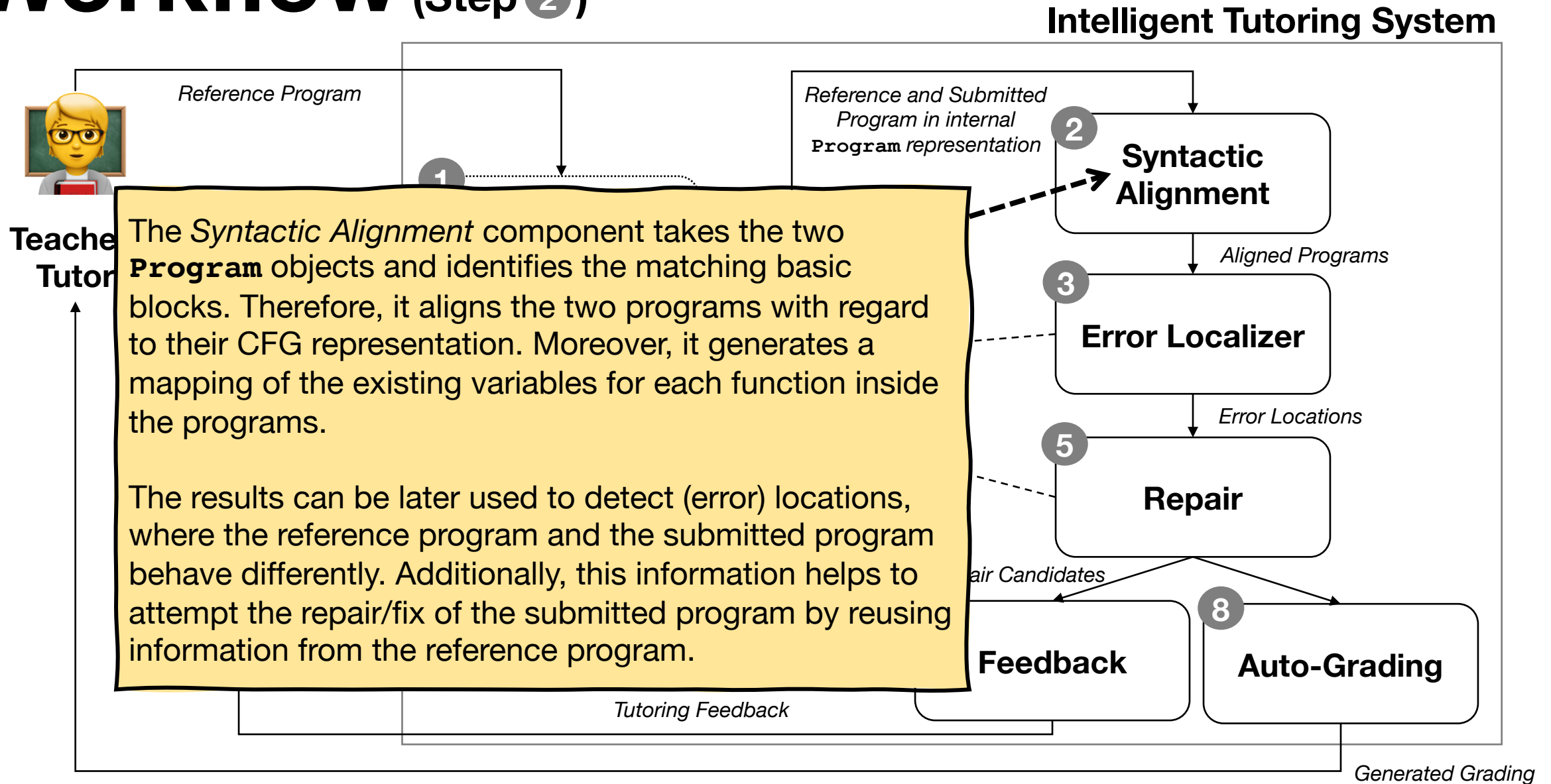
Workflow (Overview)



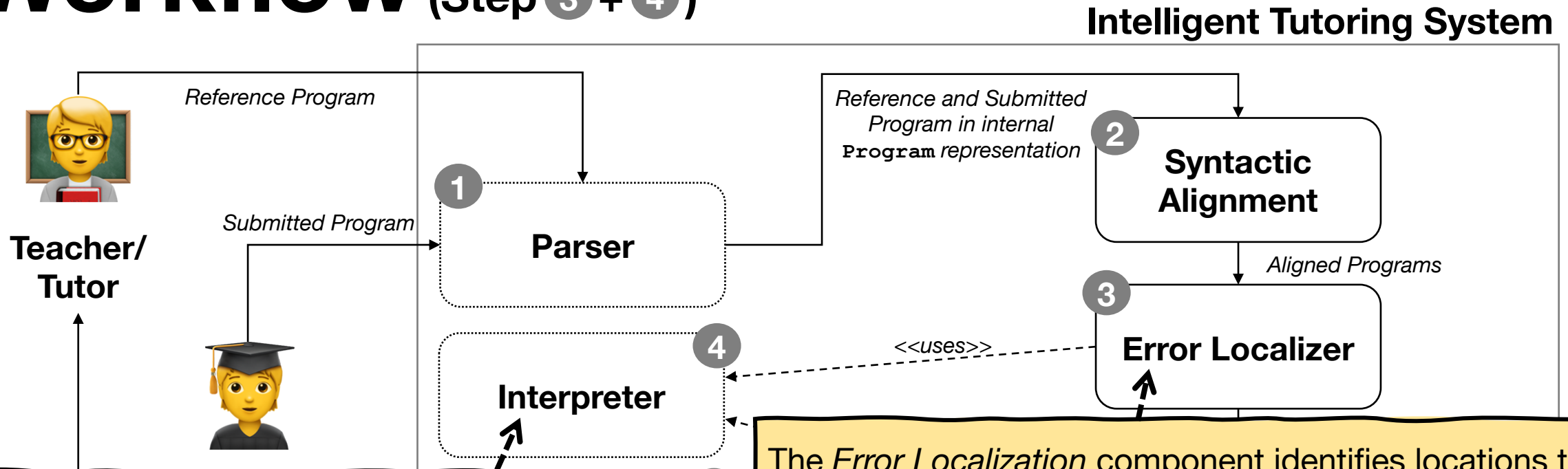
Workflow (Step 1)



Workflow (Step 2)



Workflow (Step 3 + 4)



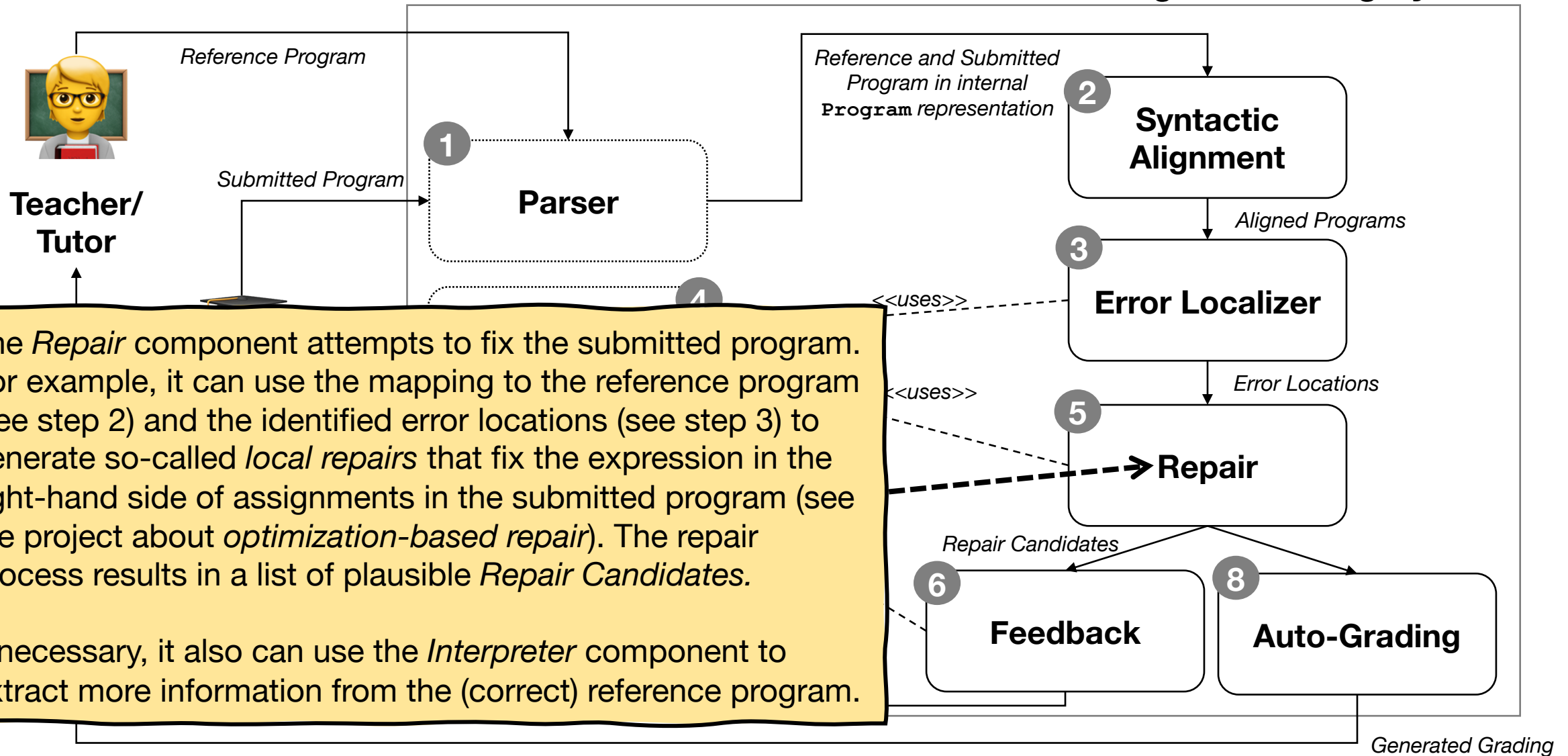
The *Interpreter* component allows the execution of a program in its CFG-based representation without any compilation or execution on the actual system. It generates an execution *trace* that includes the sequence of executed basic blocks and a *memory* object, which holds the variable values at specific locations.

The *Error Localization* component identifies locations that show erroneous behavior in the submitted program. These locations are also called *error locations*. This information enables the upcoming components in the workflow to formulate a repair/fix.

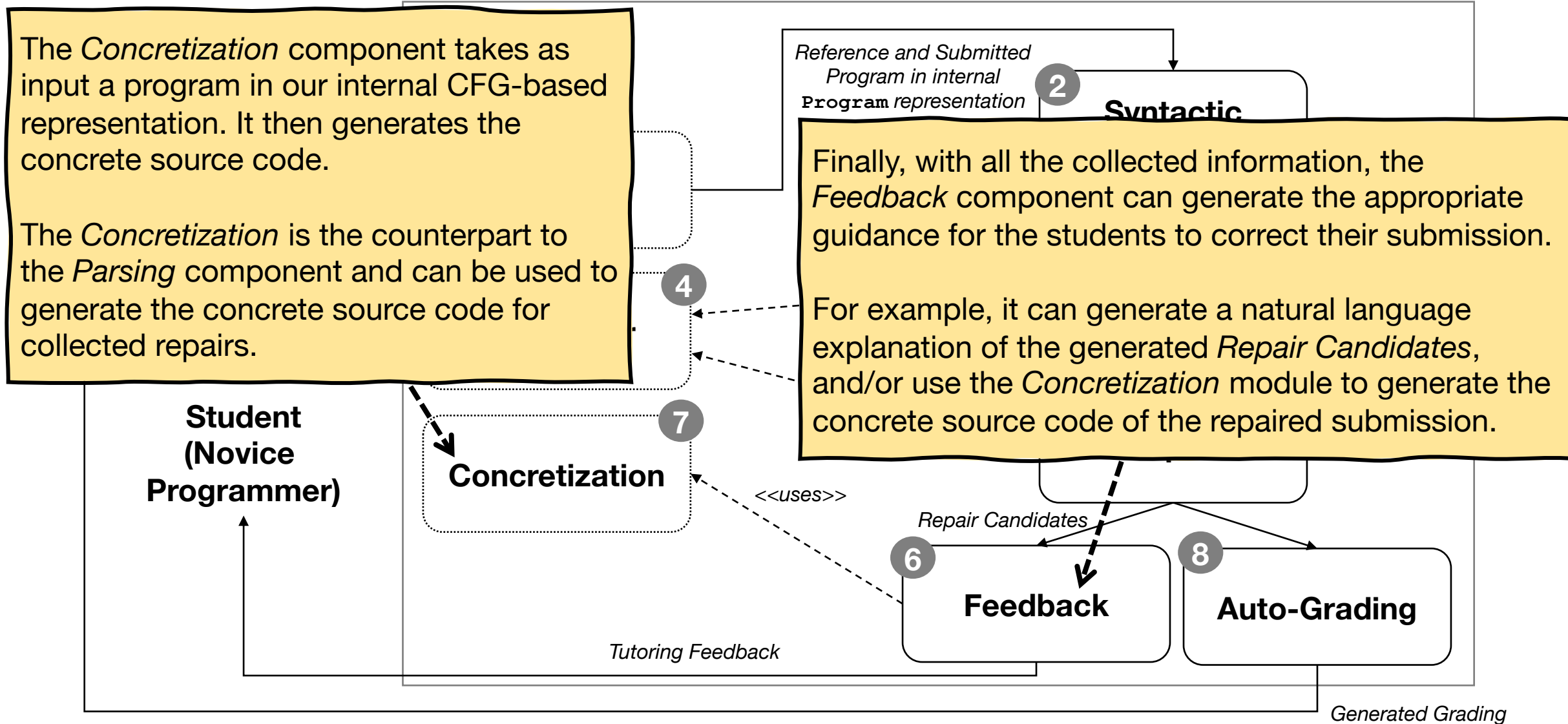
The *Error Localization* component has access to the *Interpreter* component to execute test cases while observing the values of variables at specific locations. In particular, it can use the *Interpreter* to detect semantic differences between the reference program and the submitted program.

Workflow (Step 5)

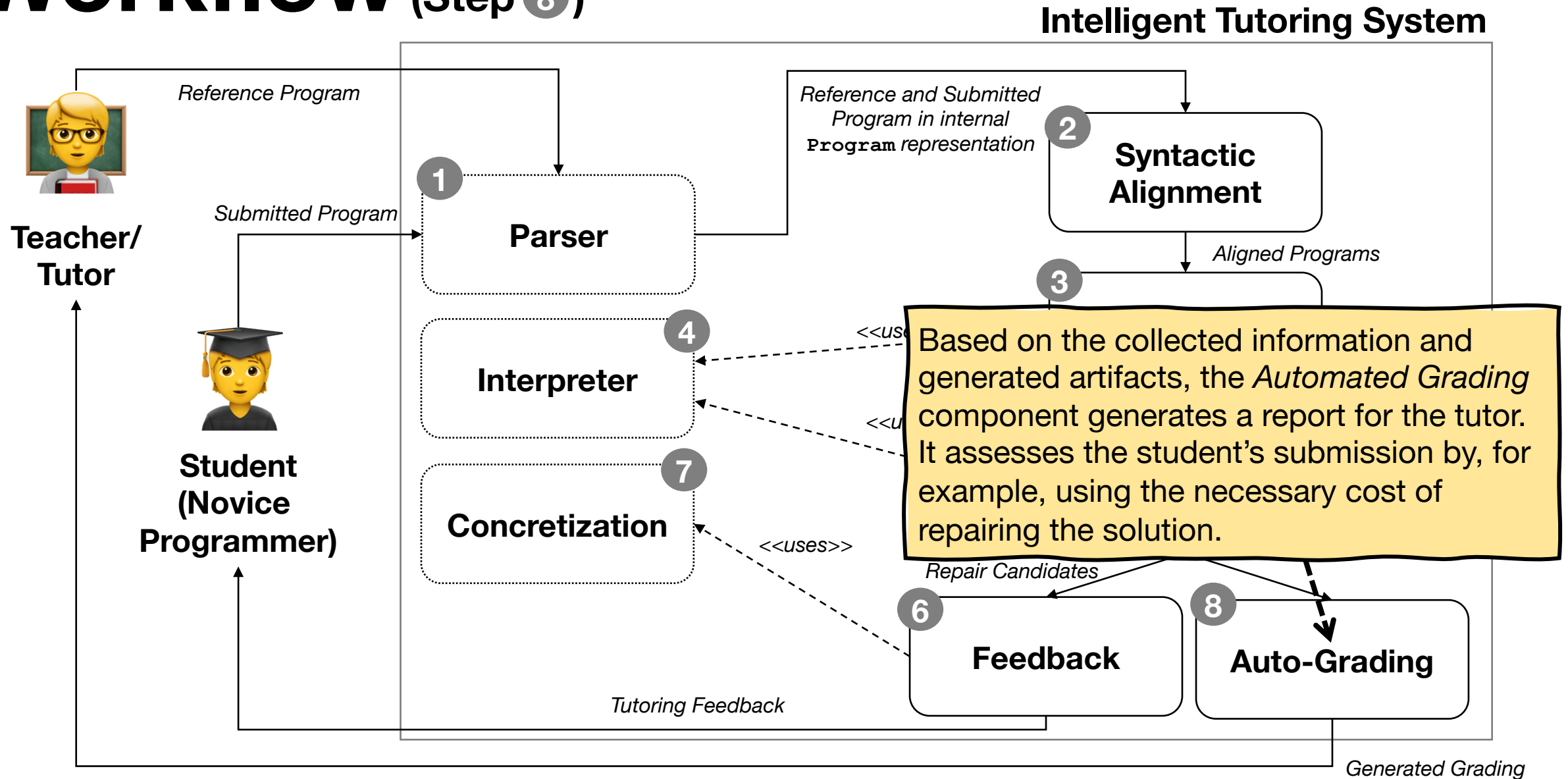
Intelligent Tutoring System



Workflow (Step 6 + 7)



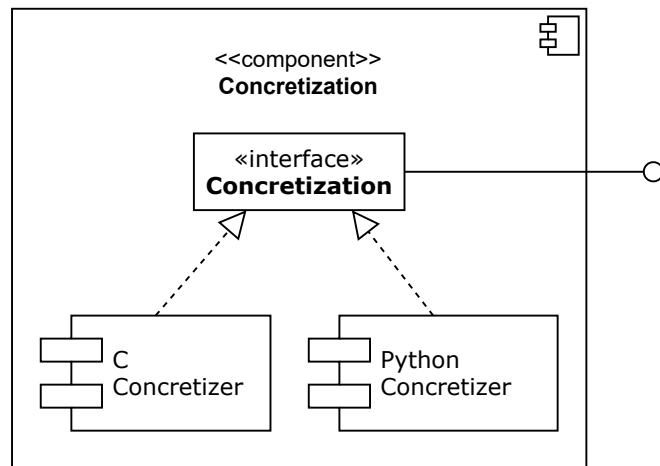
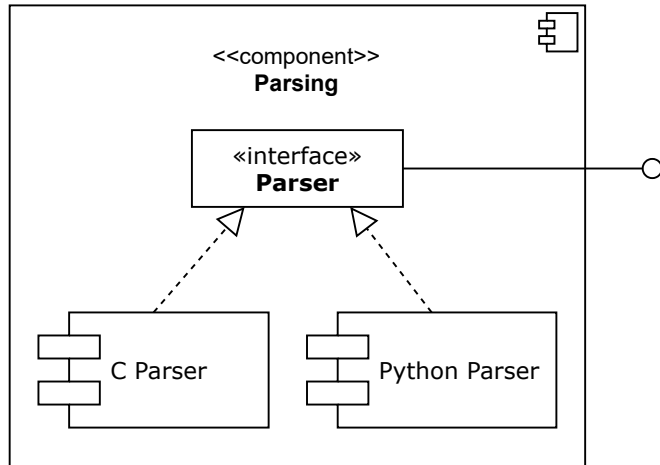
Workflow (Step 8)



Components in the context of the **its-core** baseline

The next slides show more details about the components and their interfaces. For your projects, you will get access to the **its-core**, which includes all interfaces, common data structures, and some utility functions.

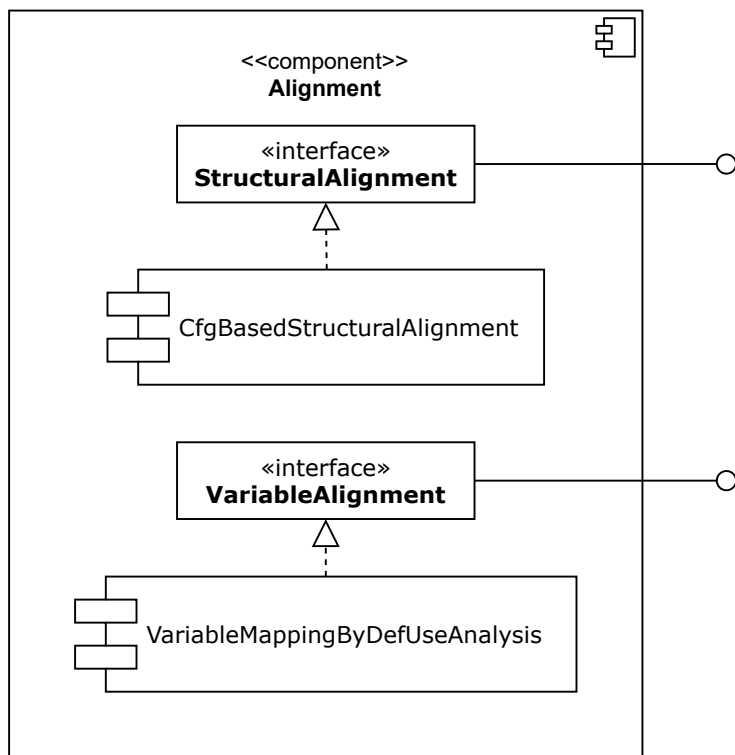
Components (1/4)



```
its-core > src/main/java > sg.edu.nus.se.its.parser > Parser  
1 package sg.edu.nus.se.its.parser;  
2  
3 import java.io.File;  
6  
7 /**  
8  * Interface for parsing a source code file into the internal data structure representation.  
9  */  
10 public interface Parser {  
11  
12     /**  
13      * Parses the program source code and returns the program in form of the internal object  
14      * representation.  
15      *  
16      * @param filePath - the path to the program text file.  
17      * @return the internal representation of the program source code.  
18      * @throws IOException if the file does not exist or could not be parsed.  
19      */  
20     public Program parse(File filePath) throws IOException;  
21 }  
23
```

```
its-core > src/main/java > sg.edu.nus.se.its.concretization > Concretization  
1 package sg.edu.nus.se.its.concretization;  
2  
3 import java.io.File;  
5  
6 /**  
7  * Interface for the concretization of Program objects.  
8  */  
9 public interface Concretization {  
10  
11     /**  
12      * Returns the file object of the repaired program in the internal representation.  
13      *  
14      * @param repairedProgram - the repaired program in its internal representation.  
15      * @return the source code representation of the given Program object  
16      */  
17     public File concretize(Program repairedProgram, String fileName) throws Exception;  
18 }  
20
```

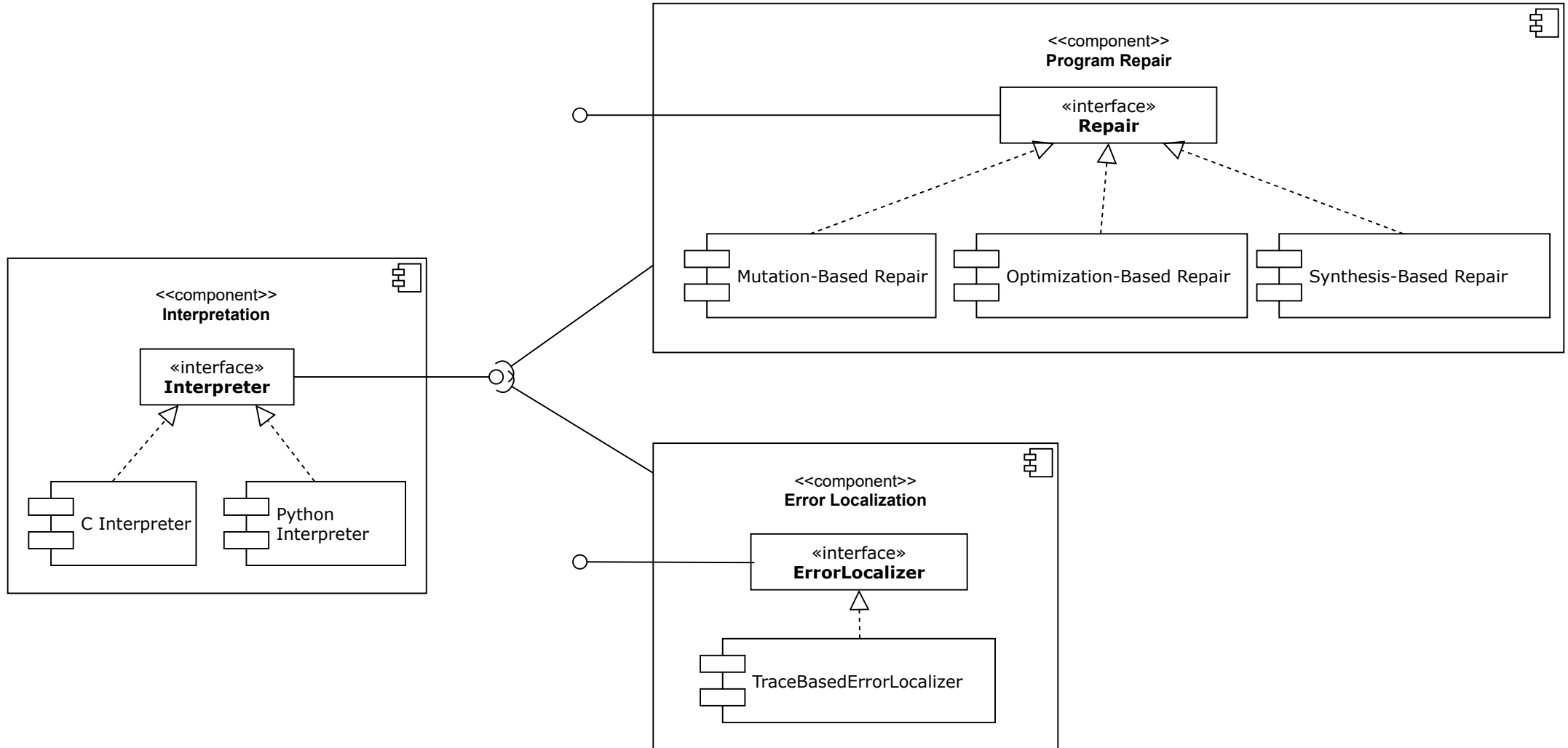
Components (2/4)



```
its-core > src/main/java > sg.edu.nus.se.its.alignment > StructuralAlignment
1 package sg.edu.nus.se.its.alignment;
2
3 import sg.edu.nus.se.its.exception.AlignmentException;
4
5
6 /**
7  * Interface for the syntactical alignment of the reference program and the submitted program.
8  */
9 public interface StructuralAlignment {
10
11     /**
12      * Generates the structural alignment for the given two programs.
13      *
14      * @param reference - the reference program
15      * @param submission - the submitted/incorrect program
16      * @return the structural mapping
17      * @throws AlignmentException - a custom alignment exception
18      */
19     StructuralMapping generateStructuralAlignment(Program reference, Program submission)
20         throws AlignmentException;
21 }
22
```

```
its-core > src/main/java > sg.edu.nus.se.its.alignment > VariableAlignment
1 package sg.edu.nus.se.its.alignment;
2
3 import sg.edu.nus.se.its.exception.AlignmentException;
4
5
6 /**
7  * The interface for any variable alignment between two structural aligned program.
8  */
9 public interface VariableAlignment {
10
11     /**
12      * Generates variable alignment.
13      *
14      * @param reference - the reference program
15      * @param submission - the submitted/incorrect program
16      * @param strucAlignment - the structural alignment
17      * @return the resulting variable alignment
18      * @throws AlignmentException - a custom alignment exception
19      */
20     VariableMapping generateVariableAlignment(Program reference, Program submission,
21         StructuralMapping strucAlignment) throws AlignmentException;
22 }
23
```

Components (3/4)



Components (3/4)

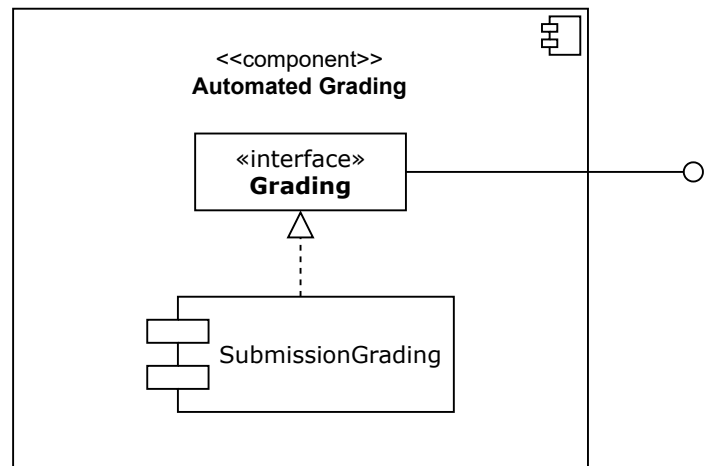
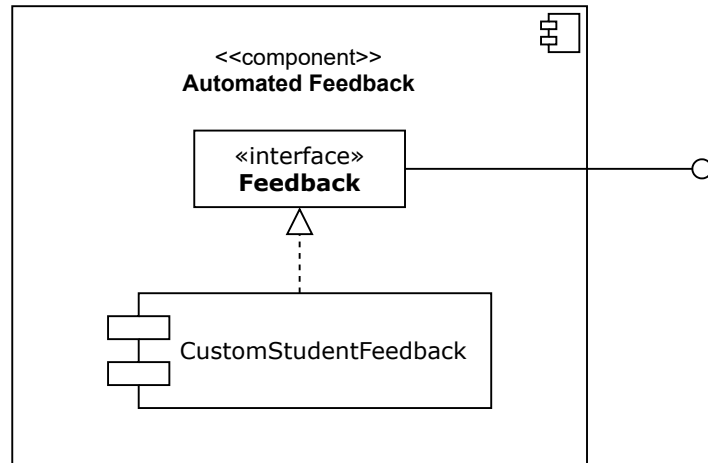
```
its-core ▶ src/main/java ▶ sg.edu.nus.se.its.interpreter ▶ Interpreter ▶
1 package sg.edu.nus.se.its.interpreter;
2
3 import sg.edu.nus.se.its.model.Constant;
4
10
11 /**
12  * Interfaces for the interpretation of the program execution.
13  */
14 public interface Interpreter {
15
16     public Trace executeProgram(Program program);
17
18     public Trace executeProgram(Program program, Input input);
19
20     public Object execute(Executable executable, Memory memory);
21
22     public Object executeFunction(Function function, Memory memory);
23
24     public Object executeConstant(Constant constant, Memory memory);
25
26     public Object executeOperation(Operation operation, Memory memory);
27
28     public Object executeVariable(Variable variable, Memory memory);
29
30     public void setTimeout(int timeout);
31
32 }
33
```

```
its-core ▶ src/main/java ▶ sg.edu.nus.se.its.repair ▶ Repair ▶
1 package sg.edu.nus.se.its.repair;
2
3 import java.util.List;
4
10 /**
11  * Interface for the repair module.
12  */
13 public interface Repair {
14
15     /**
16      * Repairs the student's incorrect program, given the reference solution and a mapping of
17      * erroneous blocks. The repaired program is in the internal representation format.
18      *
19      * @param referenceProgram -- the parsed internal representation of the reference program.
20      * @param submittedProgram -- the parsed internal representation of the student's incorrect
21      *      program.
22      * @param errorLocations -- a list of bijective set of erroneous locations
23      * @param variableMapping -- the variable mapping
24      * @param inputs -- set of inputs used for the program
25      * @return a list of possible repairs for the student's incorrect program
26      * @see Program
27      */
28     public List<RepairCandidate> repair(Program referenceProgram, Program submittedProgram,
29         ErrorLocalisation errorLocations, VariableMapping variableMapping, List<Input> inputs,
30         Interpreter interpreter) throws Exception;
31 }

```

```
its-core ▶ src/main/java ▶ sg.edu.nus.se.its.errorlocalizer ▶ ErrorLocalizer ▶
1 package sg.edu.nus.se.its.errorlocalizer;
2
3 import java.util.List;
4
10 /**
11  * Interface for the error localization.
12  */
13 public interface ErrorLocalizer {
14
15     /**
16      * Returns a list of erroneous blocks of the submitted program, given the reference program, the
17      * student program and the mapping of aligned blocks.
18      *
19      * @param submittedProgram -- the parsed internal representation of the student's program.
20      * @param referenceProgram -- the parsed internal representation of the reference program.
21      * @param inputs -- list of String value for the inputs to test for (can be null)
22      * @param functionName -- function to analyze
23      * @param structuralMapping -- the alignment of the reference program with the student's program.
24      * @param variableMapping -- the mapping of the variables in both programs
25      * @param interpreter -- interpreter object for the execution of the traces
26      * @return list of error location pairs
27      */
28     public ErrorLocalisation localizeErrors(Program submittedProgram, Program referenceProgram,
29         List<Input> inputs, String functionName, StructuralMapping structuralMapping,
30         VariableMapping variableMapping, Interpreter interpreter);
31
32 }
33
```

Components (4/4)



```
its-core > src/main/java > sg.edu.nus.se.its.feedback > Feedback >  
1 package sg.edu.nus.se.its.feedback;  
2  
3 import sg.edu.nus.se.its.model.Program;  
4  
5  
6 /**  
7  * Interface for the feedback module.  
8  */  
9 public interface Feedback {  
10  
11 /**  
12  * Generates feedback for the student based on the identified list of repairs.  
13  *  
14  * @param repairCandidate -- one set of consistent local repairs  
15  * @param submittedProgram -- the submitted program  
16  * @return feedback in form of a String object  
17  */  
18 public String provideFeedback(RepairCandidate repairCandidate, Program submittedProgram);  
19 }  
20  
21
```

Note that for both, *Feedback* and *Grading*, the interfaces are not 100% fixed. Depending on "what" information your *Feedback* and *Grading* strategy requires, we will adapt the interfaces accordingly. We will also discuss with you how feasible your plans are and how they can be adjusted to fit the system.