**Software Construction**

**Pseudocode Programming Process**

**for Control Flow Graph Generator**

**06/27/2016**

**Team 7**

**Class: GUI**

**returnCFG(Graph obj)**

Receives request for creating CFG

If XML file exists and contains tags that identify the components of a CFG diagram

Return CFG

**generateCFGdiagram(File XML)**

Creates CFG iterating to each parent tag and its child tags

By each parent tag, generate GUI element to be used as graphic visualization.

The parent tag generates a node graphic.

Use entry/exit tag points for each parent and child tag to draw a direct line using the GUI library that generates the line.

**Class: XML**

**convertToXML()**

Reads parsed code

Identify every node containing a node ID, statement type, entry point, exit point.

Every node from the CFG class should not be empty

If a node is empty or contains invalid arguments (such as entry/exit point, nodeID, statement type)

Handle the error describing that current node is empty

Handle the error describing that an argument has invalid data type

Terminate execution time of CFG program.

Else traverse every node and call the functions that allows to create XML tag for nodes and edges

Generates XML code

Based on each node defined by CFG class, iterate number of existing nodes

If XML file contain the XML declaration at the beginning of the file

Write XML code in the XML file

If no such XML declaration included

Display error message saying that the XML file will not be able to be read by the GUI

**createXMLnode(Node obj)**

Define a unique XML tag for nodes called parent XML tag

Parent XML tag will be considered has parent tag because it can contain multiple nodes that reflects nested statements such iterative/sequential/conditional.

Multiple nodes contained by the parent tag are defined as child XML tags.

A parent tag might contain zero to many child tags

A child tag can also become a parent tag if the child tag contains multiple nodes

Identify statment type of each node and define it as statment tag.

**createXMLedge(Node obj)**

Identify entry/exit point to be defined as point tags.

Define unique XML tag for entry points called exit point identifier XML tag

Define unique XML tag for exit points called exit point identifier XML tag

**Class: Control Flow Graph**

**getCFG()**

Reads XML file

If node is found

Gets graphical representations of nodes

If edge is found

Gets graphical representation of edges

Returns CFG

**Class: Edge**

**setEdge()**

Create unique ID for the edge

**getEdge()**

Return edge’s UID

**Class: Node**

**setNode()**

Create unique ID for the node

**getNode()**

Return Node’s UID

**Class: Parser**

**parseSourceCode()**

Reads Source code given by the grammar class

Checks grammar for the code

Parse the code according to the statement type

Returns parsed code

**Class: Grammar**

**applyGrammar()**

Read rules from file

Return rules

**Class: Statement**

**createStatement(Rules, Syntax)**

Use syntax rules that identify syntax.

Identify terminals, non-terminals, basic block using syntax

Determine the end of a statement using terminal rule .

Determine literals using Non-terminal rule .

Determine special keywords(such as: if, switch, for, while, break, do-while) that is considered a node using Basic block rule

Identify basic blocks enclosed with open/close curly bracket.

If terminal rule with special keywords such as if, switch are identified

Set the statement as a Conditional statement type

If basic block starts with the following keywords:for, while, break, do-while

Set the statement as a Iterative statement type

Else, Identify terminal rule with variable names with a semicolon

Set the statement as a sequential statement type