Team Teamwork – CSStructural

Milestone 1 Report

Team Member	**Role**	**Tasks Assigned**	**Completed**
Joshua Feltman	Contact, Developer	Use Case Diagram, Software Documentation, Milestone Report Code: Halstead checks, number of operators and operands	Yes
Matthew Johnson	Developer	Component and Architecture Diagram Code: Number of comments, lines of comments, variable declarations, and external/local method references.	Yes
Andrew Fallin	Developer	Code: Number of looping statements and expressions	Yes
Benjamin Hamlin	Developer	UML Class Diagram, Code: Number of casts and maintainability index.	Yes

The software process that we have chosen is Agile Development. We first chose to use the Waterfall process since we thought it fit in with how the deliverables were set up for the class since we had the design and code in deliverable 1 and the testing in deliverable 2. After designing the program then starting the actual code portion, we realized our design was pretty far off from how we thought the plugin extension would work, so we had to redesign our class and component diagram. Because of this, we are more accurately following an Agile process since our design is evolving with our actual code.

The two antipatterns we used for deliverable 1 are:

1. Blob class – We recognized this antipattern in our code with the SuperCheck class since the class is computing 8 of the structural metrics when it could have been separated into multiple classes.
2. Spaghetti Code – This pattern was realized in our code with the visitToken function inside the SuperCheck class. This function is doing all the logic for 8 different structural metric checks, which leaves the function a jumbled mess. This function could easily be better broken up into separate helper functions to do each check on its own.

Major Project Activities:

- Meetings:
  - September 3, 2018
    - First team meeting
    - Decided to use Waterfall software process

- September 10, 2018
    - Discussed requirements and design specifications
    - Discussed which Antipatterns we want to use
- September 17, 2018
    - Divided up design/requirement diagrams amongst team members
    - Most team member still have not had the development environment set up, waiting for help from professor or class mates.
- September 24, 2018
    - All diagrams have been uploaded to GitHub
    - All team members have gotten the project set up
    - Divided up all the different metrics amongst team members
- October 1, 2018
    - Most of the code has been committed to GitHub
    - Discussed redesigning class and component diagrams due to the way the CheckStyle plugin is extended
    - Switching from Waterfall to Agile software process

Software Documentation

Installation:

To install our software, clone the extended github repository with the command:

git clone https://github.com/jfeltman/Team-Teamwork-CSStructural.git

Once the repository has been cloned and assuming CheckStyle is already installed, follow these instructions to import the project into eclipse:

1. Go to 'File"
2. Click on Import -> General -> Existing Project into Workspace
3. Click in 'Browse' and go to the directory where you cloned Team-Teamwork-CSStructural
4. Select the net.sf.eclipse.sample directory of the repository
5. Select all projects
6. Click on "Copy into Workspace"
7. Click on Finish

Now you should have the sample project imported into Eclipse, and it can be run by:

1. Right clicking net.sf.eclipse.sample (root directory)
2. Then select "Run as Eclipse Application"

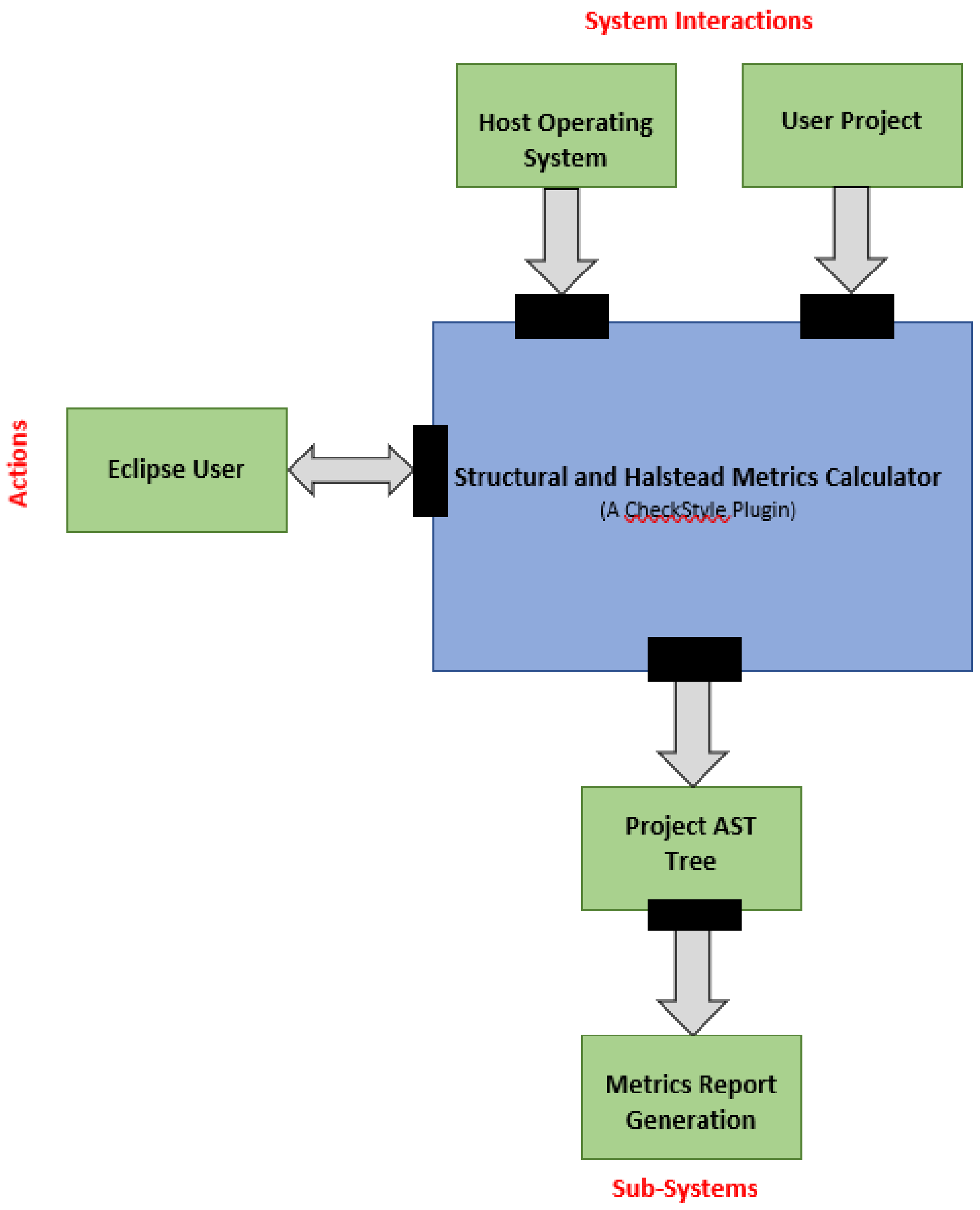
Use:

To use our project, please follow the instructions above for installing it, then once you have run the Eclipse Application follow these instructions:

1. Go to "Eclipse-> Preferences->CheckStyle"
2. Create new configuration
    a. Click on New and give it a name then ok
    b. Double click to configure
    c. Select "Super Check", "Halstead Check", and "Maintainability Index" from My Custom Checks and click on Add then Ok.
    d. Set the new config as default then click on "Apply and close"
3. Too see the checks
    a. Import a new project (In the running Eclipse Application)
    b. Right click on project -> Properties. Make sure the right config is selected.
    c. Right click on the source code file and go to Checkstyle->Check code with CheckStyle
    d. Open CheckStyle viewer to see all the Structural Metrics

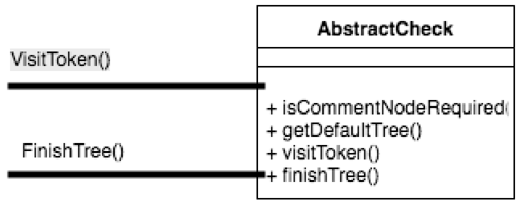
Design Specification

Architectural Context Diagram

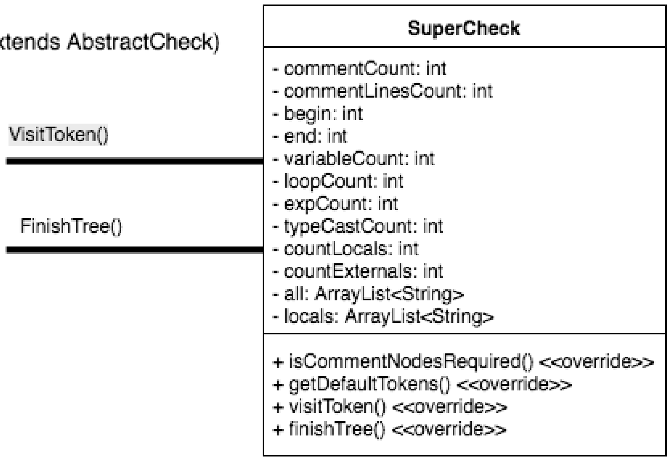


Elaborated Component Diagram

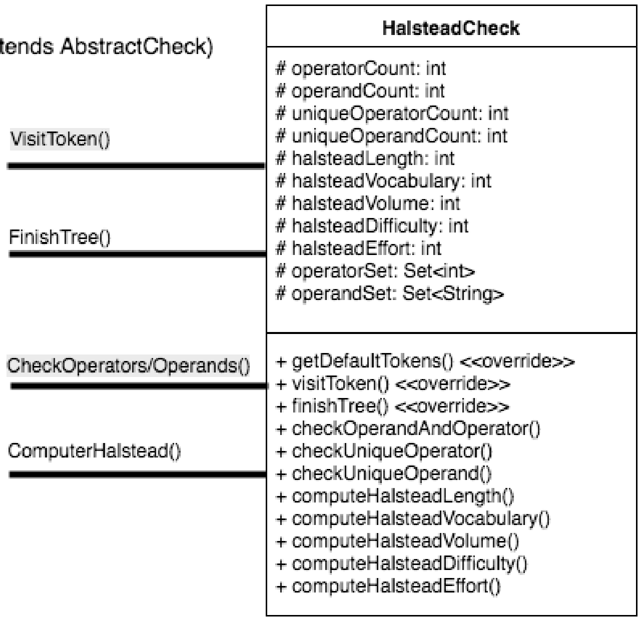
1.1 AbstractCheck



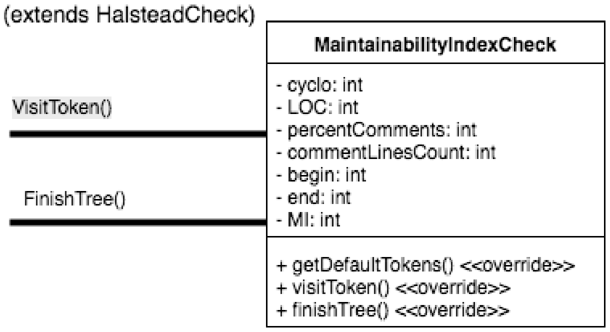
1.2 SuperCheck (extends AbstractCheck)



1.3 HalsteadCheck (extends AbstractCheck)

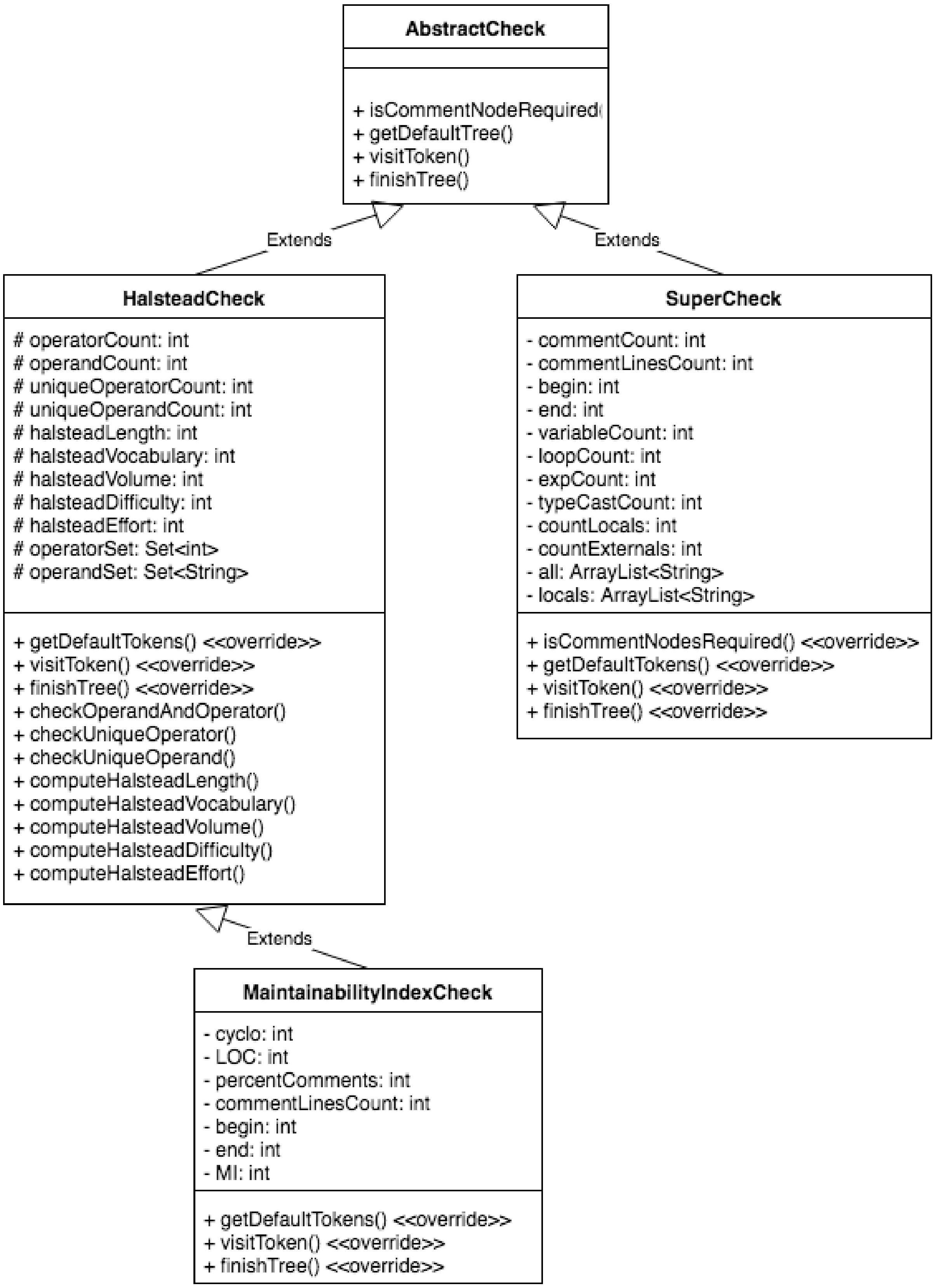

HalsteadCheck
VisitToken()
FinishTree()
CheckOperators/Operands()
ComputerHalstead()
# operatorCount: int
# operandCount: int
# uniqueOperatorCount: int
# uniqueOperandCount: int
# halsteadLength: int
# halsteadVocabulary: int
# halsteadVolume: int
# halsteadDifficulty: int
# halsteadEffort: int
# operatorSet: Set<int>
# operandSet: Set<String>
+ getDefaultTokens() <<override>>
+ visitToken() <<override>>
+ finishTree() <<override>>
+ checkOperandAndOperator()
+ checkUniqueOperator()
+ checkUniqueOperand()
+ computeHalsteadLength()
+ computeHalsteadVocabulary()
+ computeHalsteadVolume()
+ computeHalsteadDifficulty()
+ computeHalsteadEffort()

1.3 MaintainabilityIndexCheck (extends HalsteadCheck)

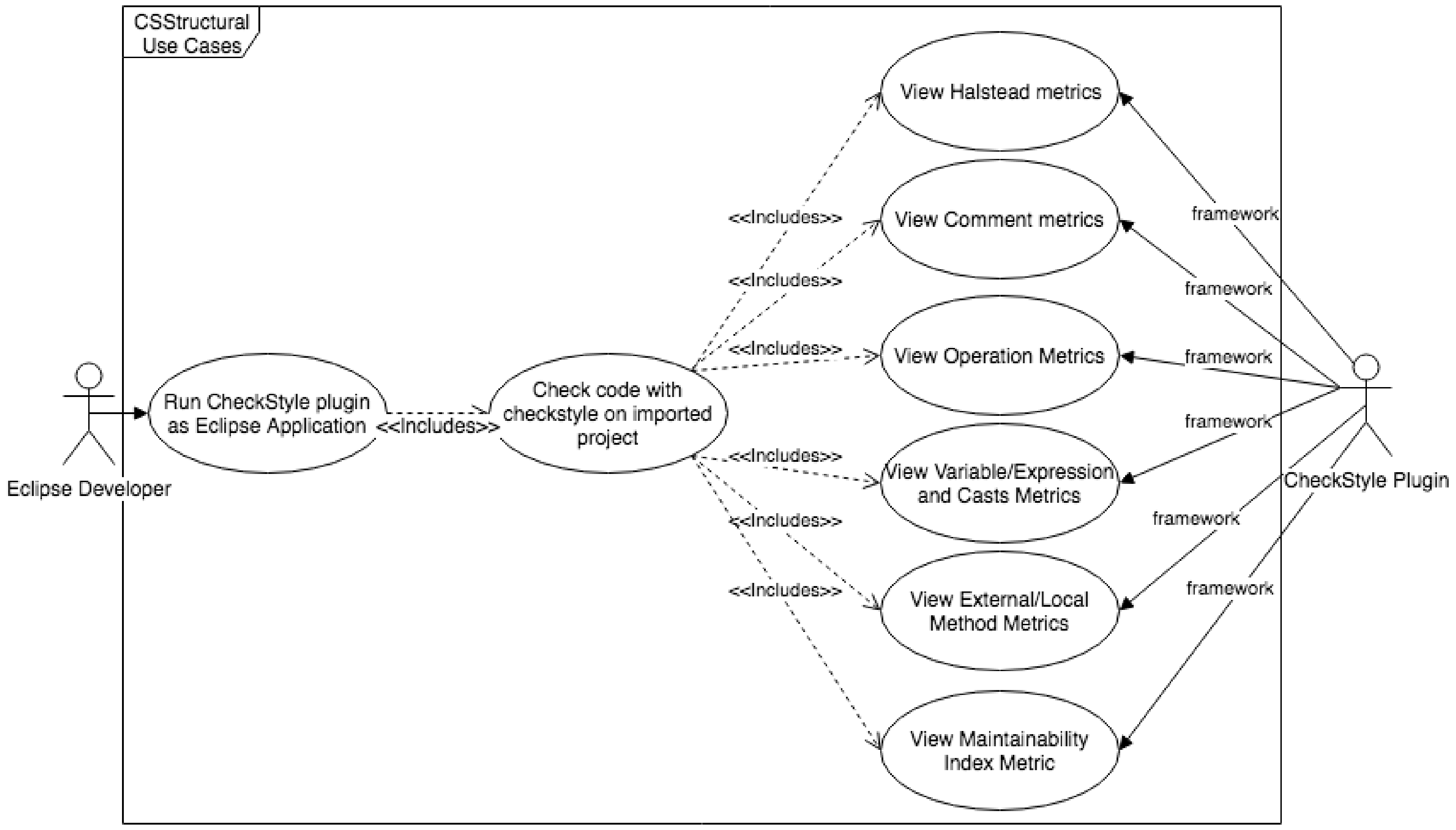


Requirements Specification

Class Diagram


AbstractCheck
+ isCommentNodeRequired(
+ getDefaultTree()
+ visitToken()
+ finishTree()
Extends
Extends
HalsteadCheck
# operatorCount: int
# operandCount: int
# uniqueOperatorCount: int
# uniqueOperandCount: int
# halsteadLength: int
# halsteadVocabulary: int
# halsteadVolume: int
# halsteadDifficulty: int
# halsteadEffort: int
# operatorSet: Set<int>
# operandSet: Set<String>
+ getDefaultTokens() <<override>>
+ visitToken() <<override>>
+ finishTree() <<override>>
+ checkOperandAndOperator()
+ checkUniqueOperator()
+ checkUniqueOperand()
+ computeHalsteadLength()
+ computeHalsteadVocabulary()
+ computeHalsteadVolume()
+ computeHalsteadDifficulty()
+ computeHalsteadEffort()
SuperCheck
- commentCount: int
- commentLinesCount: int
- begin: int
- end: int
- variableCount: int
- loopCount: int
- expCount: int
- typeCastCount: int
- countLocals: int
- countExternals: int
- all: ArrayList<String>
- locals: ArrayList<String>
+ isCommentNodesRequired() <<override>>
+ getDefaultTokens() <<override>>
+ visitToken() <<override>>
+ finishTree() <<override>>
Extends
MaintainabilityIndexCheck
- cyclo: int
- LOC: int
- percentComments: int
- commentLinesCount: int
- begin: int
- end: int
- MI: int
+ getDefaultTokens() <<override>>
+ visitToken() <<override>>
+ finishTree() <<override>>

Use Case Diagram



Run CheckStyle Plugin as Eclipse Application

Actors:	• Eclipse Developer
Goal:	• Run the CheckStyle plugin on java file
Preconditions:	• Java file must exist
Scenarios:	• User wants to check structural metrics
Exceptions:	• None

Check Code with Checkstyle on Imported Project

Actors:	• Eclipse Developer
Goal:	• Use the CheckStyle plugin to find all the structural metrics
Preconditions:	• Java file must exist • CheckStyle plugin installed
Scenarios:	• User wants to check structural metrics
Exceptions:	• None

View Halstead Metrics

Actors:	• Eclipse Developer, CheckStyle Plugin
Goal:	• Output Halstead Length • Output Halstead Vocabulary • Output Halstead Volume • Output Halstead Difficulty • Output Halstead Effort
Preconditions:	• Java file must exist • User must have run CheckStyle plugin
Scenarios:	• User wants to check structural metrics
Exceptions:	• None

View Comment Metrics

Actors:	• Eclipse Developer, CheckStyle Plugin
Goal:	• Output number of comments • Output number of lines of comments
Preconditions:	• Java file must exist • User must have run CheckStyle plugin
Scenarios:	• User wants to check structural metrics
Exceptions:	• None

View Operation Metrics

Actors:	• Eclipse Developer, CheckStyle Plugin
Goal:	• Output number of operands • Output number of operators
Preconditions:	• Java file must exist • User must have run CheckStyle plugin
Scenarios:	• User wants to check structural metrics
Exceptions:	• None

View Variable/Expression and Casts Metrics

Actors:	• Eclipse Developer, CheckStyle Plugin
Goal:	• Output number of expressions • Output number of variable declarations • Output number of casts
Preconditions:	• Java file must exist • User must have run CheckStyle Plugin
Scenarios:	• User wants to check structural metrics
Exceptions:	• None

View External/Local Method Metrics

Actors:	• Eclipse Developer, CheckStyle Plugin
Goal:	• Output number of methods called from external class • Output number of methods called from same class
Preconditions:	• Java file must exist • User must have run CheckStyle Plugin
Scenarios:	• User wants to check structural metrics
Exceptions:	• None

View Maintainability Index Metrics

Actors:	• Eclipse Developer, CheckStyle Plugin
Goal:	• Output Maintainability Index
Preconditions:	• Java file must exist • User must have run CheckStyle Plugin
Scenarios:	• User wants to check structural metrics
Exceptions:	• None