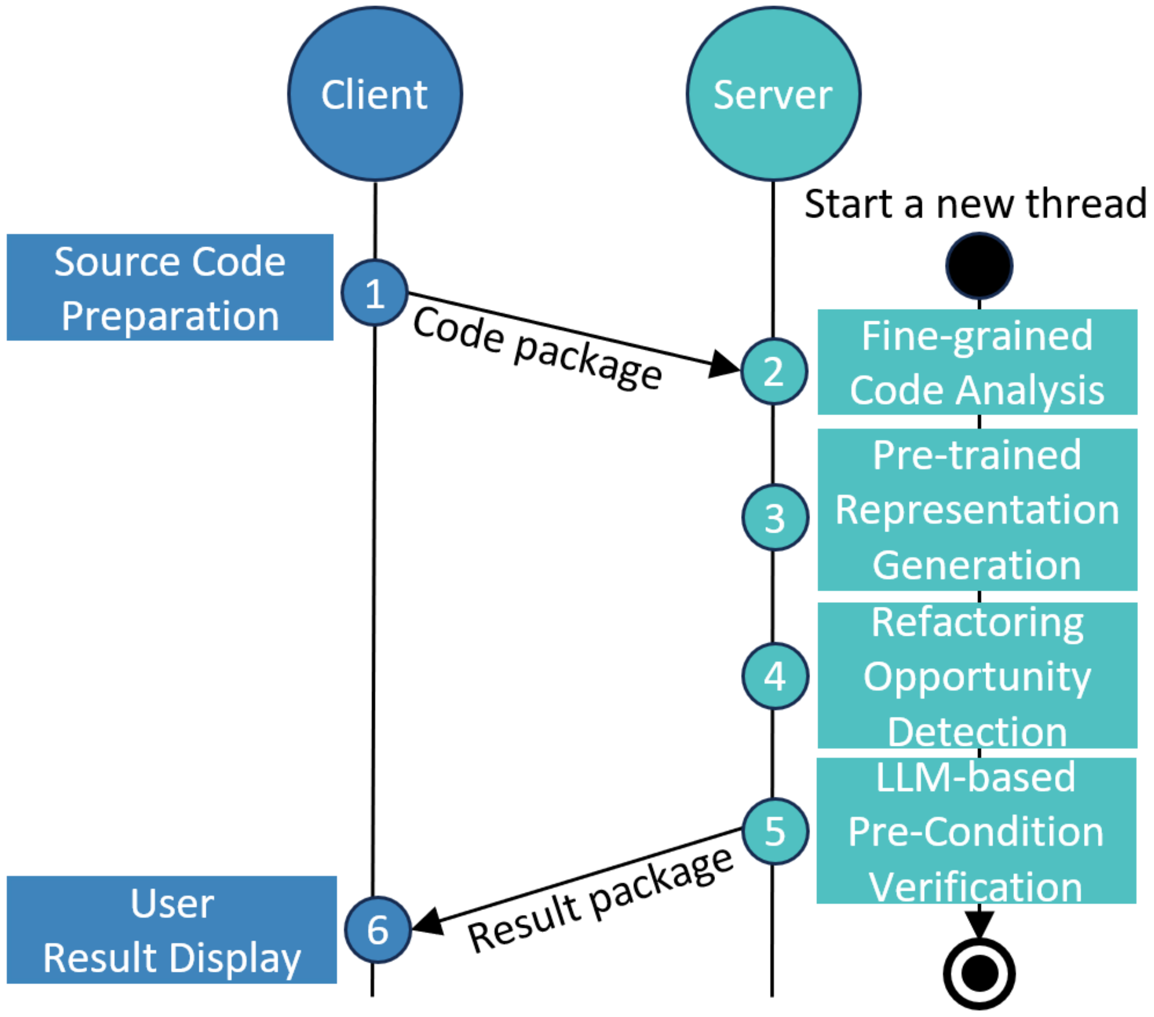
HECS: An Intelligent Refactoring Plugin

Plugin Overview



Client
Server
Start a new thread
Source Code Preparation
1
Code package
2
Fine-grained Code Analysis
3
Pre-trained Representation Generation
4
Refactoring Opportunity Detection
5
LLM-based Pre-Condition Verification
Result package
6
User Result Display

The HECS Intelligent Refactoring Plugin is an extension for performing intelligent code refactoring in Visual Studio Code. It provides a set of prediction and refactoring tools to help developers improve code quality and readability. Our prototype implementation adopts a two-layer C/S architecture: the client layer for interaction with users and the server layer for refactoring opportunity detection. It is worth noting that in the server layer, we have deployed a large model for automated verification of the legitimacy of refactoring candidates.

We regret to inform you that due to version iteration and upgrades, our plugin is currently experiencing some issues with front-end and back-end integration. As a result, it is temporarily unavailable. We are working diligently to resolve these problems and will release our VSCode plugin in the next version as soon as possible. In the meantime, you can install our plugin to experience the front-end UI part.

Installation

1. Download the .vsix file from our package.
2. Open Visual Studio Code (VS Code).
3. Go to the Extensions view (Ctrl+Shift+X).
4. Click the menu button (three dots) at the top right of the Extensions view.

5. Select "Install from VSIX."
6. Navigate to and open .vsix extension file using the file browser.

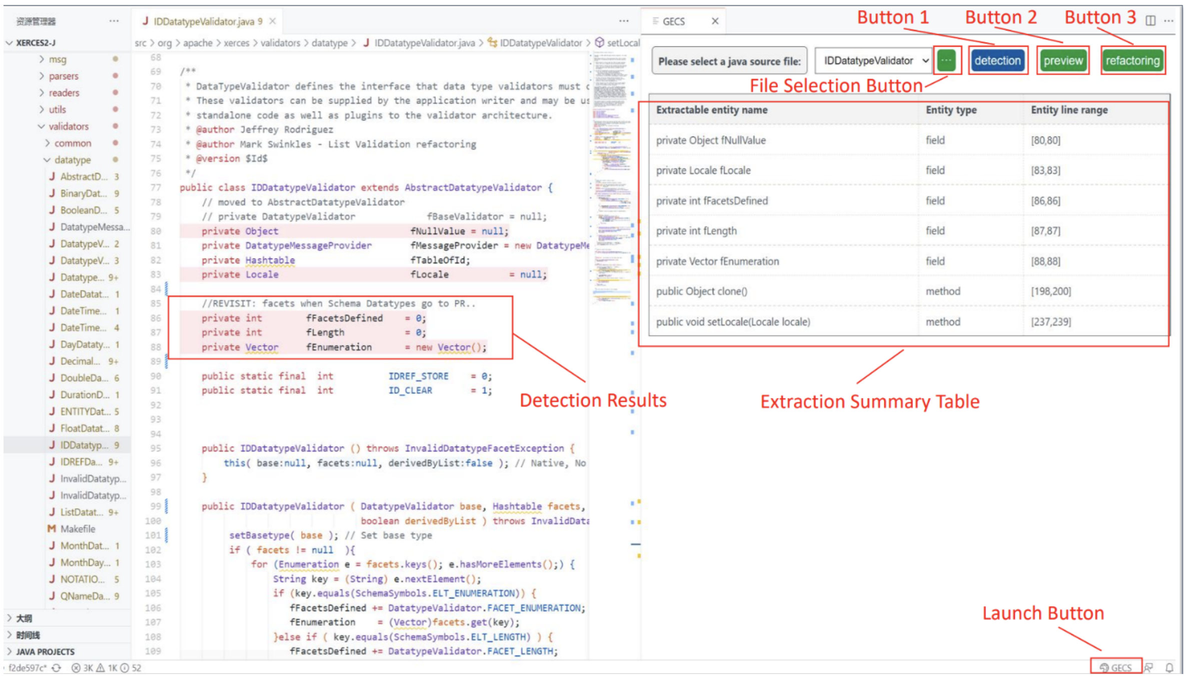
Usage

1. Open your test project.
2. Open the Java file that you want to refactor in the editor.
3. Click the HECS icon in the bottom-right corner to open the plugin interface.
4. In the plugin interface, select the file and the refactoring operation:
   - Choose the Java source file to refactor.
   - Predict: Click to predict code refactoring.
   - Preview: View a preview of the code refactoring.
   - Refactor: Execute the code refactoring.
5. Follow the prompts to complete the refactoring.

Plugin Entry Point

You can find the HECS icon in the bottom-right corner of VS Code. Click it to open the plugin interface.

The Client Layer

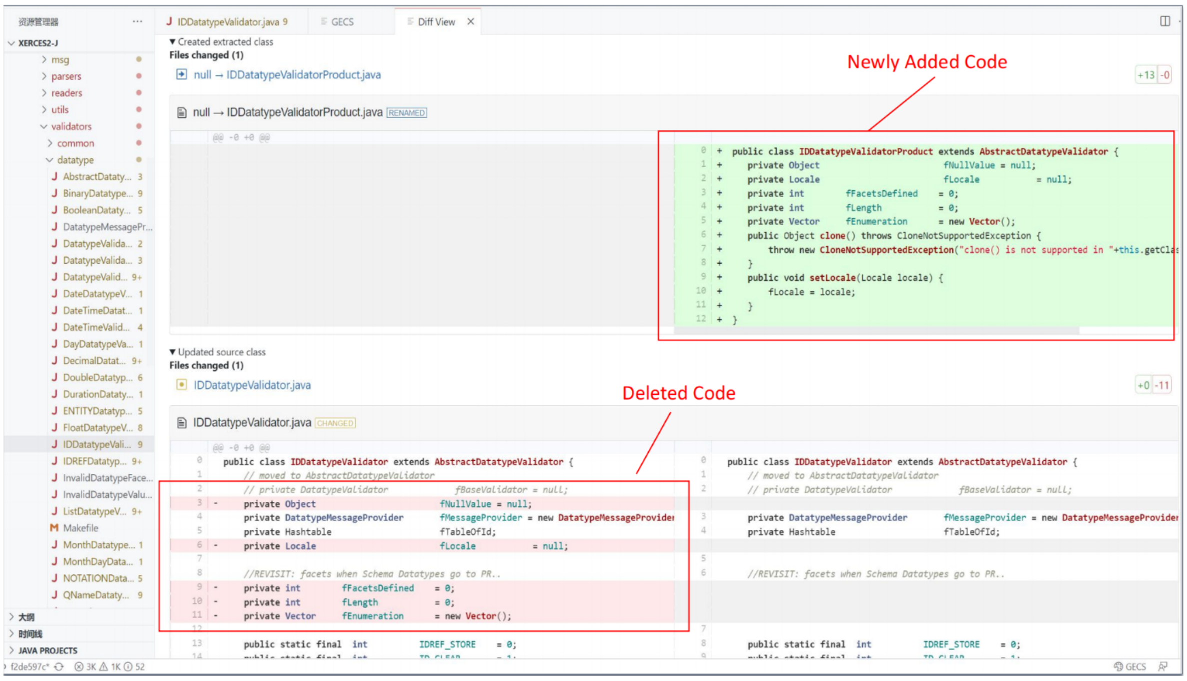

IDDatatypeValidator.java 9
src > org > apache > xerces > validators > datatype > IDDatatypeValidator.java > IDDatatypeValidator > setLocal
GECS
Button 1
Button 2
Button 3
Please select a java source file:
IDDatatypeValidator
···
detection
preview
refactoring
File Selection Button
Detection Results
Extraction Summary Table
Launch Button
GECS

```
/**
 * DataTypeValidator defines the interface that data type validators must c
 * These validators can be supplied by the application writer and may be us
 * standalone code as well as plugins to the validator architecture.
 * @author Jeffrey Rodriguez
 * @author Mark Swinkles - List Validation refactoring
 * @version $Id$
 */
public class IDDatatypeValidator extends AbstractDatatypeValidator {
    // moved to AbstractDatatypeValidator
    // private DatatypeValidator              fBaseValidator = null;
    private Object                         fNullValue = null;
    private DatatypeMessageProvider        fMessageProvider = new DatatypeMe
    private Hashtable                      fTableOfId;
    private Locale                         fLocale           = null;

    //REVISIT: facets when Schema Datatypes go to PR..
    private int          fFacetsDefined    = 0;
    private int          fLength           = 0;
    private Vector       fEnumeration      = new Vector();

    public static final  int          IDREF_STORE    = 0;
    public static final  int          ID_CLEAR       = 1;



    public IDDatatypeValidator () throws InvalidDatatypeFacetException {
        this( base:null, facets:null, derivedByList:false ); // Native, No
    }

    public IDDatatypeValidator ( DatatypeValidator base, Hashtable facets,
                                 boolean derivedByList ) throws InvalidData
        setBasetype( base ); // Set base type
        if ( facets != null  ){
            for (Enumeration e = facets.keys(); e.hasMoreElements();) {
                String key = (String) e.nextElement();
                if (key.equals(SchemaSymbols.ELT_ENUMERATION)) {
                    fFacetsDefined += DatatypeValidator.FACET_ENUMERATION;
                    fEnumeration    = (Vector)facets.get(key);
                }else if ( key.equals(SchemaSymbols.ELT_LENGTH) ) {
                    fFacetsDefined += DatatypeValidator.FACET_LENGTH;
```

Extractable entity name	Entity type	Entity line range
private Object fNullValue	field	[80,80]
private Locale fLocale	field	[83,83]
private int fFacetsDefined	field	[86,86]
private int fLength	field	[87,87]
private Vector fEnumeration	field	[88,88]
public Object clone()	method	[198,200]
public void setLocale(Locale locale)	method	[237,239]

The client layer serves as a Visual Studio Code (VSCode) extension, which accepts user inputs and communicates with the server layer. Once installing this extension, users can access the client layer with a launch button shown in Fig.1 and Fig.2. The client layer has two different views: detection view and refactoring view, which are illustrated as follows: 1) The Detection View. Fig.1 presents a snapshot of the detection view. First, users should click the file selection button to select a source file for detection. Next, when the user clicks the `detection` button (Button 1) to require detection results, the client layer will send an HTTPS request to the server layer and receive the data package simultaneously. The detection results will further be summarized with a table (Extraction Summary Table), where each row presents names, types, and code line ranges of the field or method to be extracted. The detection results will also be highlighted and displayed with the red colour in the window of the source file. The functionality of the `preview` button

(Button 2) and `refactoring` button (Button 3) will be illustrated in the refactoring view. 2) The Refactoring View.



▼ Created extracted class
Files changed (1)
null → IDDatatypeValidatorProduct.java
+13 -0

null → IDDatatypeValidatorProduct.java RENAMED

```
@@ -0 +0 @@
+ public class IDDatatypeValidatorProduct extends AbstractDatatypeValidator {
+    private Object                        fNullValue = null;
+    private Locale                        fLocale          = null;
+    private int         fFacetsDefined    = 0;
+    private int         fLength           = 0;
+    private Vector      fEnumeration      = new Vector();
+    public Object clone() throws CloneNotSupportedException {
+       throw new CloneNotSupportedException("clone() is not supported in "+this.getCla
+    }
+    public void setLocale(Locale locale) {
+       fLocale = locale;
+    }
+ }
```

▼ Updated source class
Files changed (1)
IDDatatypeValidator.java
+0 -11

IDDatatypeValidator.java CHANGED

```
@@ -0 +0 @@
  public class IDDatatypeValidator extends AbstractDatatypeValidator {
     // moved to AbstractDatatypeValidator
     // private DatatypeValidator             fBaseValidator = null;
-    private Object                        fNullValue = null;
     private DatatypeMessageProvider       fMessageProvider = new DatatypeMessageProvide
     private Hashtable                     fTableOfId;
-    private Locale                        fLocale          = null;

     //REVISIT: facets when Schema Datatypes go to PR..
-    private int         fFacetsDefined    = 0;
-    private int         fLength           = 0;
-    private Vector      fEnumeration      = new Vector();

     public static final  int          IDREF_STORE    = 0;
```

```
  public class IDDatatypeValidator extends AbstractDatatypeValidator {
     // moved to AbstractDatatypeValidator
     // private DatatypeValidator             fBaseValidator = null;

     private DatatypeMessageProvider       fMessageProvider = new DatatypeMessageProvide
     private Hashtable                     fTableOfId;


     //REVISIT: facets when Schema Datatypes go to PR..


     public static final  int          IDREF_STORE    = 0;
```

Fig.2 presents a snapshot of the detection view. When the user clicks the `preview` button (Button 2) in Fig.1, a new window will be created to demonstrate fine-grained code differences between the target class and extracted class, where differences with the green colour represent the extracted added code and differences with the red colour represent the extracted deleted code. We employ a state-of-the-art tool: diff2html, to support the visualization of code differences in an HTML format. The refactoring view assists users in examining each refactoring operation. Users can further click the `refactoring` button (Button 3) in Fig.1 to automatically and accurately execute detected extract class refactoring opportunities.