CCIntegrator: A Tool for Software Product Line Platform Construction to Support Product Family Development

Summary

Goal: CCIntegrator (Cloned Code Integrator) automates the migration from the Clone-and-Own (CAO) approach to Software Product Line Engineering (SPLE) for product family development.

Problem Definition: While the CAO approach is popular for its low initial cost (simply cloning and modifying existing products), long-term use leads to:

- Maintenance nightmare: Independent maintenance of an ever-growing number of product variants.
- Traceability loss: Difficulty in tracking and managing modified clones across products.
- Management overhead: Increased complexity in propagating bug fixes and new features.

Approach: CCIntegrator integrates cloned code from CAO-developed products to construct a reusable product line platform and derive configurable product variants.

- Platform construction: Automatically identifies and integrates cloned source code from existing products into a unified product line platform.
- Product configuration: Facilitates the derivation of specific products based on the newly constructed platform, ensuring consistency and reuse.

Contents

1. Introduction

In the process of rebuilding a product line platform, CCIntegrator performs the series of tasks as shown in Figure 1-1. Taking a product family developed via the CAO approach as input, CCIntegrator generates a product line platform that integrates cloned code to facilitate the derivation of products.

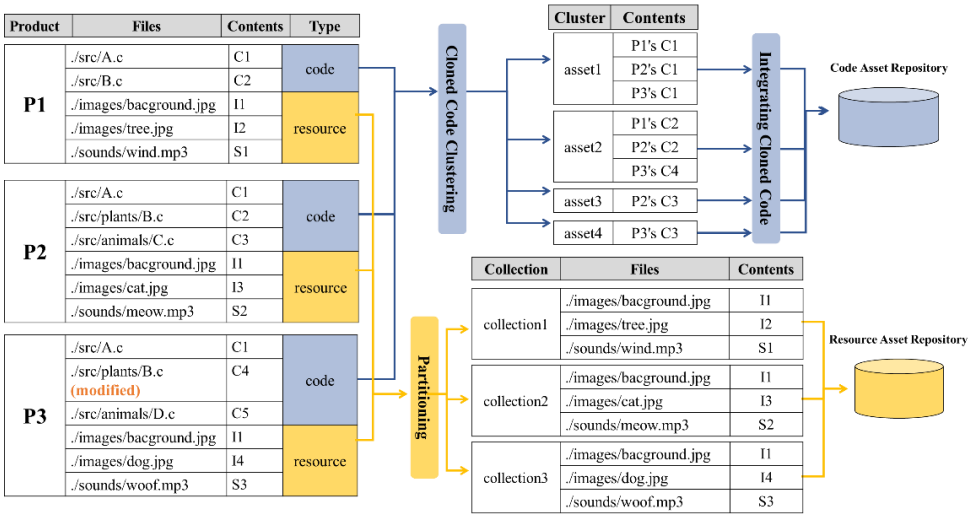

Fig. 1-1. Constructing code assets and resource assets for the platform

Figure 1-2 shows the steps of CCIntegrator for platform construction. CCIntegrator first determines a product family by removing dead code and measuring inter-product code similarity based on a predefined threshold. It then clusters cloned code, identifies common and variable code within the resulting clusters, and integrates them to construct a software product line platform. These phases are executed sequentially, with CCIntegrator automating the core activities required for CAO-to-SPLE migration.

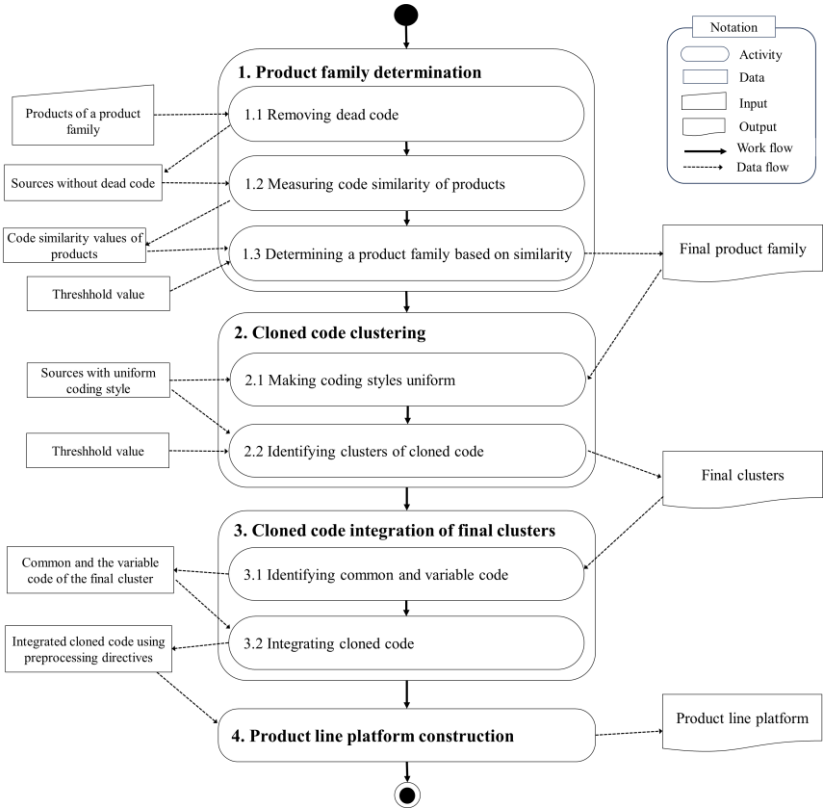

Fig. 1-2. Steps of CCIntegrator for product line platform construction

1) **Product family determination**
In this phase, functionally similar products are identified from a set of simiar products, and a target product family suitable for platform construction is determined based on code similarity. The detailed tasks are as follows:

A. Identify and remove dead code in each product.

B. Measure product family level-code similarity based on the proportion of methods shared across multiple products .

C. Determine the final product family based on a user-defined code similarity threshold.

2) **Cloned code clustering**
In this phase, cloned code is identified and grouped from the source code of products of the selected product family, resulting in a set of cloned code clusters. Each final cluster is treated as a candidate asset for platform construction. The detailed tasks are as follows:

A. Uniform coding styles of all source files in the product family. A single coding style guarantees consistent token representation, enabling efficient and accurate code similarity analysis through line-by-line comparison.

B. Determine the final cloned code clusters based on a code similarity threshold applied to the code fragments composing each cluster.

3) **Cloned code integration of final clusters**

In this phase, common and variable code are extracted from each clusters and integrated to construct reusable platform assets. Common code corresponds to lines shared by all products, while variable code corresponds to lines present only in some products. Variable code is encapsulated using preprocessing directives to enable configuration. The detailed tasks are as follows:

A. Classify common and variable code lines within each cloned code cluster.

B. Group consecutive variable code lines in each source into code blocks, annotate them with preprocessing directives using product IDs, and integrate them with common code to form reusable assets.

4) **Product line platform construction**

In this phase, all reusable assets derived from the product family are registered in an asset repository to generate products again from the platform. Figure 1-3 depicts how assets are organized and managed within the repositories. The detailed tasks are as follows:

A. Register each asset in the repository according to the defined asset DTD. The 'model.xml' file captures all information necessary to reuse and configure assets within a product line platform.

	P1	P2	P3
asset1	./src/A.c	./src/A.c	./src/A.c
asset2	./src/B.c	./src/plants/B.c	./src/plants/B.c
asset3		./src/animals/C.c	
asset4			./src/animals/D.c
collection1	X		
collection2		X	
collection3			X

Fig. 1-3. An example of an asset information table describing reusable assets in the product line platform repository

2. Installation

This section describeses how to import CCIntegrator into the Eclipse IDE.

1) Copy the repository URL from GitHub.

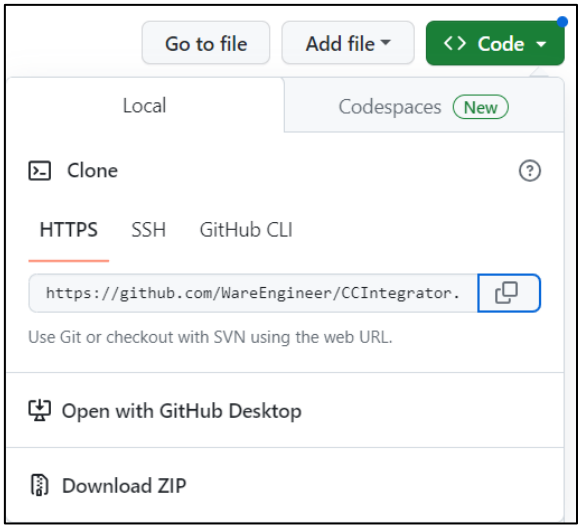

Fig. 2-1. Installation of CCIntegrator using GitHub

2) Launch Eclipse.

3) From the top menu, select File→Import...

4) Type “git” in the search box and select “Projects from Git”.

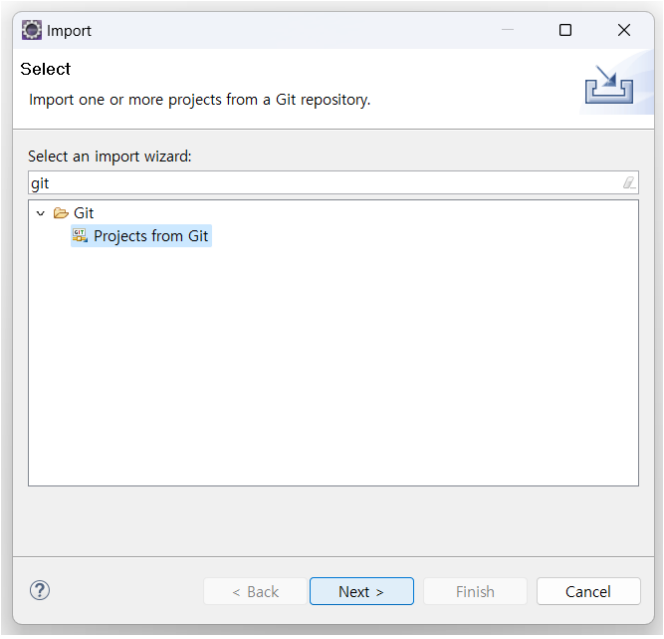

Fig. 2-2. Selecting “Projects from Git” in the Eclipse Import wizard

5) Choose “Clone URI” and click “Next”.

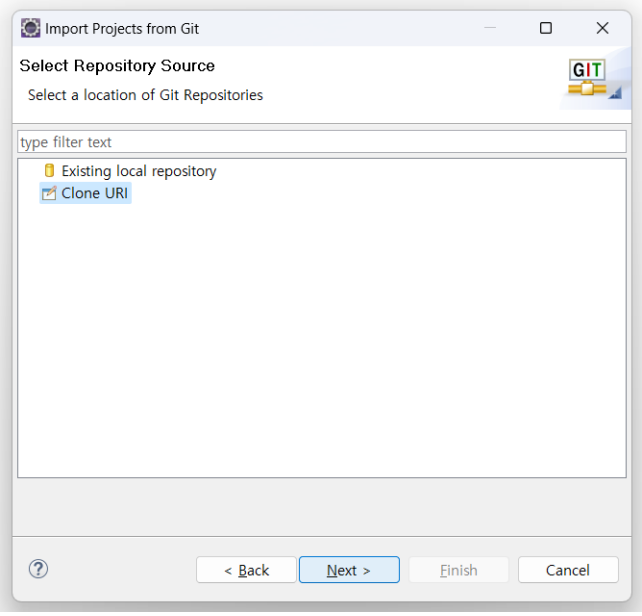

Fig. 2-3. Selecting “Clone URI” as the repository source in Eclipse

6) Paste the repository URI copied from GitHub.

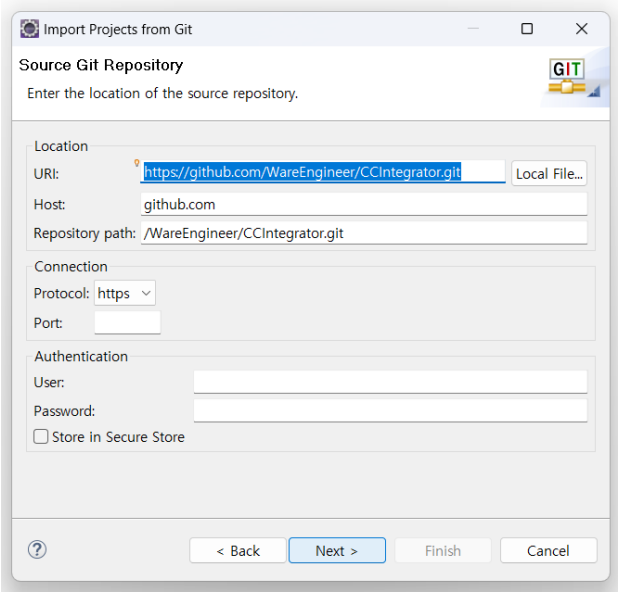

Fig. 2-4. Entering the GitHub repository URI in the Eclipse Import wizard

7) Click “Next” to proceed through the remaining dialogs and clone the CCIntegrator repository.

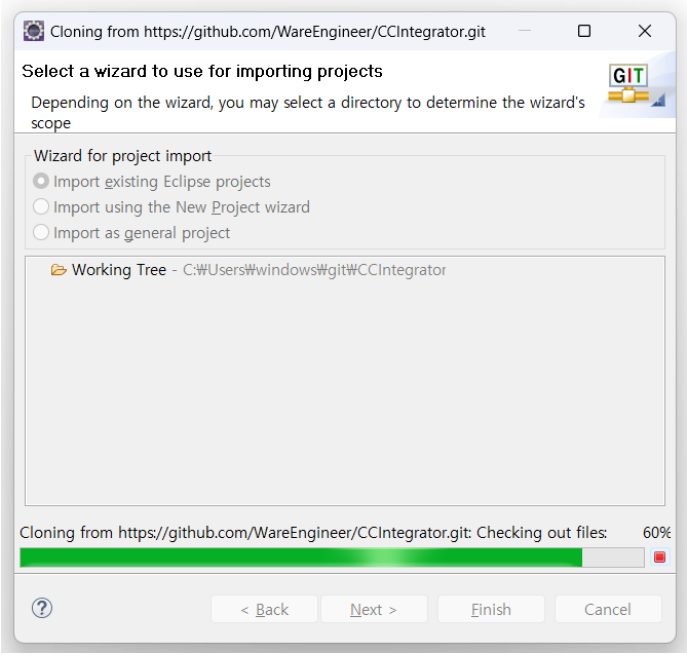

Fig. 2-5. Cloning progress during the CCIntegrator import process

8) Click “Finish” to complete the import. CCIntegrator is then successfully imported into the Eclipse IDE.

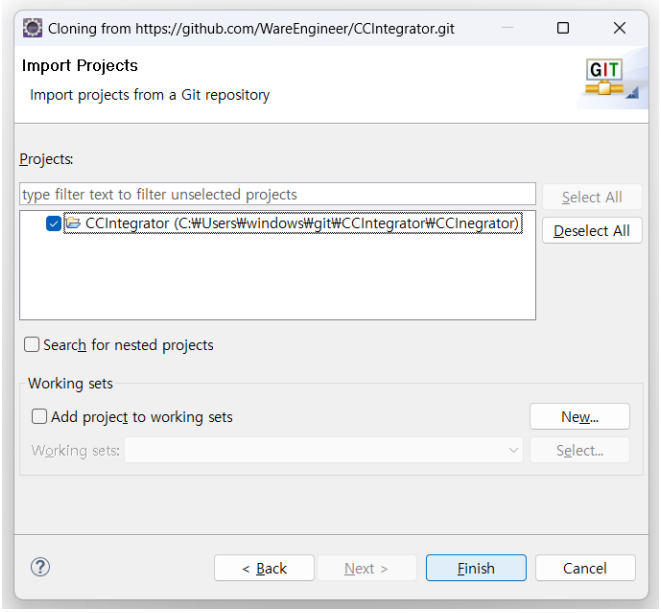

Fig. 2-6. Completing the import of CCIntegrator into the Eclipse IDE workspace

3. Directory structure of CCIntegrator

The CCIntegrator project is organized into 5 main directories: src, examples, temp, repository, and products (see Figure 3-1).

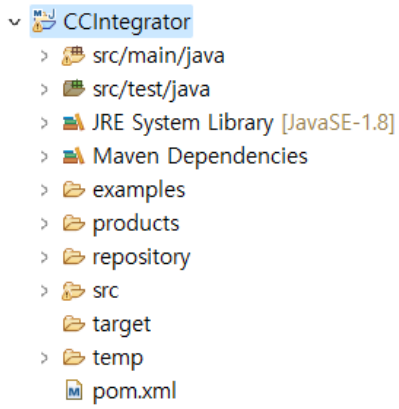

Fig. 3-1. Directory structure of CCIntegrator

1) **src**: Contains the Java source files that implement the core functionality of CCIntegrator.
2) **examples**: Provides example product families that allow users to easily explore CCIntegrator's features and observe the resulting platform construction and product derivation.
3) **temp**: Stores a working copy of the original product family. All platform construction operations are performed on this copy to preserve the the original products.
4) **repository**: Stores the constructed product line platform.
5) **products**: Stores configured product variants based on the constructed platform.

4. Guide 1: Building a Platform

This guide describes how to build a product line platform using CCIntegrator. When CCIntegrator is executed, the menu shown in Figure 4-1 is displayed in the console window. To build a product line platform for a product family, users must first prepare a text file listing the products to be migrated along with their relative paths, and then select option "1" from the menu.

```
= MENU ================================================
1. Migrating from CAO based products to SPL repository.
2. Generating products from migrated SPL repository.
3. EXIT
========================================================
$ 1
```

Fig. 4-1. Initial menu with option "1" selected for migration

As shown Figure 4-2, when no product family is prepared in advance, users may use example product families provided with CCIntegrator. The CCIntegrator/examples directory contains four java-based product families: ArgoUML, ApoGames, Elevator, and HelloWorld.

- ArgoUML: https://github.com/but4reuse/argouml-spl-benchmark
- ApoGames: https://variability-challenges.github.io/2018/ApoGames/index.html
- Elevator and HelloWorld: https://www.featureide.de/index_tool.php

```
To apply CCIntegrator to your own product variants developed with Clone-and-Own approach,
   you create a text file listing root-paths of product variants
   and enter the path of the text file as above.

We offer four text files for simple examples.
   1) .\examples\elevator.txt
   2) .\examples\helloWorld.txt
   3) .\examples\apoGames.txt
   4) .\examples\argoUML.txt

Enter a text file's path.
$
```

Fig. 4-2. Product family selection for migration

For each example, a corresponding text files (argoUML.txt, apoGames.txt, elevator.txt, and helloWorld.txt) is provided as input to CCIntegrator. Each text file lists the paths of all product variants that consist the product family. Figure 4-3 shows an example of such a text file. To migrate a product family, users can create a similar text file following this example.

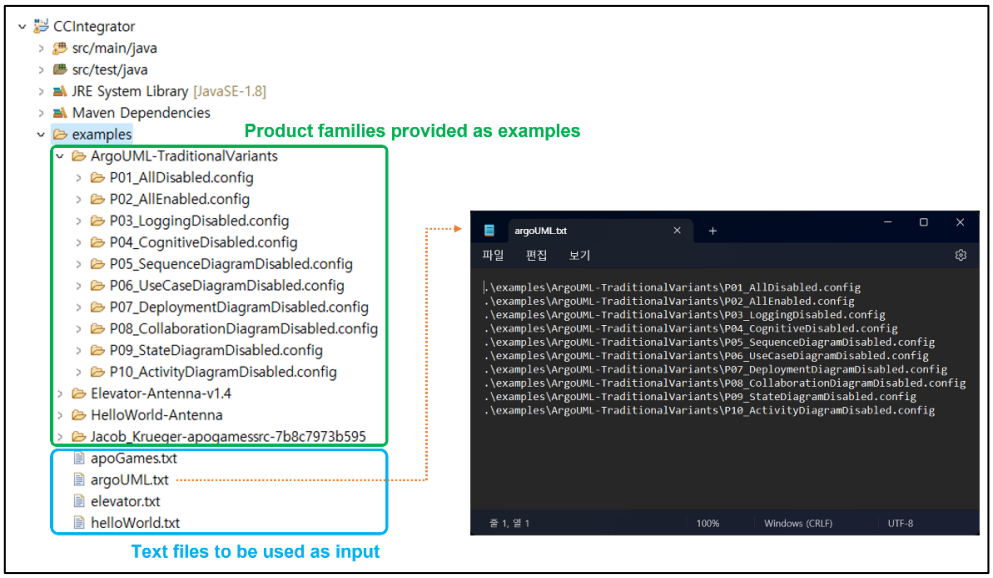

Fig. 4-3. Example text file of a product family

This guide explains the subsequent process with ApoGames example. With the given text file, CCIntegrator automatically performs the following operations:

- Sequentially reads the root directory of product variants listed in the text file and copies them to a temporary directory “CCIntegrator/temp” as shown in Figure 4-4.

- Scans all files of the product variants and removes bytecode files with the “.class” extension.

- Split a source file containing multiple top-level classes so that the source file contains exactly one top-level class named after the class, within the same package.

- Parse Java source files for syntax analysis using JavaParser (https://javaparser.org/).

```
Enter a text file's path.
$ .\examples\apoGames.txt

Loading products' files ...
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoBot
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoCheating
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoCommando
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoDefence
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoIcarus
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoIcejump
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoImp
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoMarc
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoMario
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoNotSoSimple
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoPongBeat
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoRelax
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoSimple
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoSimpleSudoku
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoSkunkman
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoSlitherLink
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoSnake
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoSoccer
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\ApoStarz
   C:\Users\windows\git\CCIntegrator\CCInegrator\.\examples\Jacob_Krueger-apogamessrc-7b8c7973b595\Java\TutorVolley
   Total Products: 20
Copying all products to a temporary directory ...
Deleting files with the class extension(.class) ...
Decomposing java files that have multiple top-level classes in one file ...
Parsing Java Source Codes ...
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoBot
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoCheating
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoCommando
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoDefence
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoIcarus
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoIcejump
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoImp
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoMarc
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoMario
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoNotSoSimple
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoPongBeat
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoRelax
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoSimple
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoSimpleSudoku
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoSkunkman
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoSlitherLink
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoSnake
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoSoccer
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\ApoStarz
   TARGET PRODUCT: C:\Users\windows\git\CCIntegrator\CCInegrator\.\temp\TutorVolley
   Total Files: 1318
```

Fig. 4-4. Copying a product family to the temporary directory

After these preprocessing steps are completed for each product variant, CCIntegrator identifies dead code within each variant as shown in Figure 4-5. Dead code refers to code that is not reachable from any program entry point and therefore does not affect program execution even if removed. In Java, the “main()” method serves as the entry point, and multiple main() methods may exist within a project. In such cases, users must manually select a single entry point.

```
Deleting Dead Codes...
TARGET PRODUCT: C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot
There are multiple main classes.
1) C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\ApoStart.java
2) C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\apoBot\ApoBotMain.java
Please choose the number to be a start point.
$ 2
=> All Files: 48
=> Dead Files: 15
=> Active Files: 33
Do you want to delete the UNREACHABLE_CLASSES? [y/n]
$ y
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\entity\ApoDragObject.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\apoBot\ApoBotApplet.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\ApoAppletBufferedStrategy.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\ApoMain.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\help\ApoCopy.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\help\ApoFileNameFilter.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\help\ApoClassLoader.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\entity\ApoButtonText.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\ApoStart.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\ApoApplet.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\ApoNewThread.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\ApoComponent.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\ApoTimer.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\help\ApoInterface.java"
   Delete "C:\Users\windows\eclipse-workspace\CCIntegrator\.\temp\ApoBot\org\apogames\sound\ApoSounds.java"

   ... (ellipsis)

=> Out of a total of 1318 files that make up the product family,
   360 files were deleted because they were dead files.
=> Total active files: 958
```

Fig. 4-5. Selecting Y/N in the console window for removing dead code

Following dead code elimination, CCIntegrator selects functionally similar product variants from the initial product family. Building a platform from functionally similar variants improves the effectiveness of product family management and derivation. To this end, users specify a similarity threshold, and only products whose similarity exceeds the threshold are selected as the final product family for migration. As shown in Figure 4-6, when the similarity threshold is set to 0.5, 11 products are selected.

```
Deciding a scope of the product family.
Please enter a threshold for excluding product variants with low functional-similarity from the product family.
$ 0.5
@ VMS (Variable Method Signature)
@ CMS (Common Method Signature)
@ EPR (Expected Platform Reusability)
=> Step1. (VMS:3291, CMS:  28, EPR:0.15) Exclude a product with Min_similarity. -TutorVolley
=> Step2. (VMS:3258, CMS:  28, EPR:0.14) Exclude a product with Min_similarity. -ApoCheating
=> Step3. (VMS:3117, CMS:  35, EPR:0.17) Exclude a product with Min_similarity. -ApoDefence
=> Step4. (VMS:2776, CMS:  74, EPR:0.31) Exclude a product with Min_similarity. -ApoStarz
=> Step5. (VMS:2658, CMS:  79, EPR:0.32) Exclude a product with Min_similarity. -ApoSoccer
=> Step6. (VMS:2209, CMS:  81, EPR:0.35) Exclude a product with Min_similarity. -ApoBot
=> Step7. (VMS:2097, CMS:  93, EPR:0.38) Exclude a product with Min_similarity. -ApoSlitherLink
=> Step8. (VMS:1958, CMS: 104, EPR:0.41) Exclude a product with Min_similarity. -ApoIcejump
=> Step9. (VMS:1810, CMS: 111, EPR:0.42) Exclude a product with Min_similarity. -ApoMario
=> Step10. (VMS:1462, CMS: 162, EPR:0.55) EPR Become above the required threshold.
=> List of products to be migrated
   1. ApoCommando
   2. ApoIcarus
   3. ApoImp
   4. ApoMarc
   5. ApoNotSoSimple
   6. ApoPongBeat
   7. ApoRelax
   8. ApoSimple
   9. ApoSimpleSudoku
   10. ApoSkunkman
   11. ApoSnake
```

Fig. 4-6. Entering a threshold value for determining the final product family

Next, as shown in Figure 4-7, CCIntegrator uniformly applies a single coding style to all Java source files in the previously determined product family.

```
Applying single code style to every java file in the product family.
=> BEFORE: The original codes
   LOC of all products: 74674
   Distinct Lines: 29278 (Common: 403, Variable: 28875)
=> Apply single coding style to products.
   TARGET PRODUCT: ApoCommando
   TARGET PRODUCT: ApoIcarus
   TARGET PRODUCT: ApoImp
   TARGET PRODUCT: ApoMarc
   TARGET PRODUCT: ApoNotSoSimple
   TARGET PRODUCT: ApoPongBeat
   TARGET PRODUCT: ApoRelax
   TARGET PRODUCT: ApoSimple
   TARGET PRODUCT: ApoSimpleSudoku
   TARGET PRODUCT: ApoSkunkman
   TARGET PRODUCT: ApoSnake
=> AFTER: The formatted codes
   LOC of all products: 71422
   Distinct Lines: 23665 (Common: 849, Variable: 22816)
```

Fig. 4-7. Making the coding style uniform

CCIntegrator then identifies cloned code fragments among the source files and groups them into candidate clusters. Each cluster consists of an original source and its clones. The similarity of sources within each cluster is computed, and clusters whose similarity exceeds a specified threshold are determined as final clusters. Fig. 4-8 illustrates part of the clustering progress using a threshold value of 0.5. As shown in Figure 4-8, when the threshold value is set to 0.5, the clustering results include 13 clusters common to all products, one cluster shared by eight products, one cluster shared by seven products, and additional clusters shared by fewer products.

```
Clustering cloned-codes generated during the creation of product variants.
Please enter a threshold for clustering cloned-codes between product variants.
$ 0.5
@ NOC (the Number Of Clusters)
=> The number of products: 11
=> Total Clusters: 343
   NOC consisting of 1 cloned files: 308
   NOC consisting of 2 cloned files: 11
   NOC consisting of 3 cloned files: 5
   NOC consisting of 4 cloned files: 3
   NOC consisting of 5 cloned files: 1
   NOC consisting of 6 cloned files: 0
   NOC consisting of 7 cloned files: 1
   NOC consisting of 8 cloned files: 1
   NOC consisting of 9 cloned files: 0
   NOC consisting of 10 cloned files: 0
   NOC consisting of 11 cloned files: 13
```

Fig. 4-8. Entering a threshold value of 0.5 for determining final clusters

Once the final clone clusters are determined, CCIntegrator builds a platform by integrating the source files within each cluster. Common lines shared across all products are extracted first, after which variable code lines are identified and annotated using preprocessing directives (See Figure 4-9). Variable code lines are annotated with unique identifiers. This integration enables the management of the product family using a single, configurable code base.

```
Merging cloned-codes into one by using annotations at the variation points.
@ VP (Variation Points)
@ CV (Code Variants of variation points)
@ CL (Common Lines)
@ SL (Shared Lines that is in V related with multi-products)
@ UL (Unique Lines that is in V related with only one product)
=> Migrated Products:11
   Assets:343 (Common:13 , Shared:22 , Unique:308)
   ReusedAssets:35 (Deterministic:21 , Non-deterministic:14)
   PlatformAssets:10
   VP:795, CV:1245, CL:1357, SL:2761, UL:47824
```

Fig. 4-9. Progress of integrating cloned code using preprocessing directives

To complete platform construction, CCIntegrator registers all assets of the product family in the asset repository. CCIntegrator uses preprocessor directives to specify variability. Source code assets are stored in the "repository/sourcecode" directory.

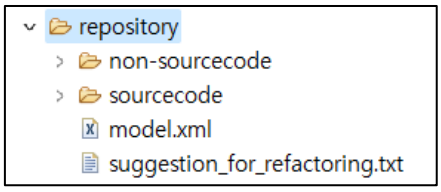

Fig. 4-10. Contents of repository directory

In addition to source code, products may include non-source artifacts such as images and sound files. These assets are stored in the "repository/non-sourcecode" directory.

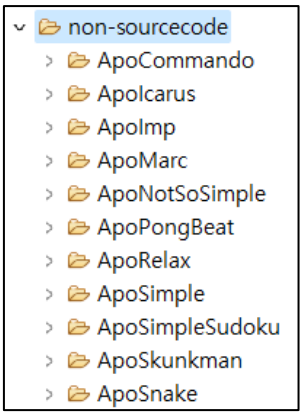

Fig. 4-11. Product-specific non-source code assets

As shown in Figure 4-12, each source code asset is stored as a separate text file, with one file generated for each clone cluster.

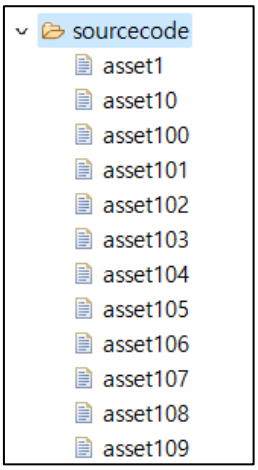

Fig. 4-12. Assets that are part of the platform

The "Model.xml" file specifies the metadata required to reuse and configure assets within the product line platform. Figure 4-13 shows the schema and the XML file of this example

Fig. 4-13. Schema and example XML document for assets supporting platform reuse

Finally, the "suggestion_for_refactoring.txt" file records cases where reusable assets appear under different file names across products. Applying the suggested refactoring helps ensure structural consistency of reusable assets, thereby strengthening the resulting product line platform. Figure 4-14 shows the refactoring suggestions for this example.

```
<CLUSTER with different relative paths among the cloned files>
=> \org\apogames\images\ApoImage.java : ApoSkunkman
=> \org\apogames\image\ApoImage.java : ApoImp
=> \org\apogames\image\ApoImage.java : ApoMarc
=> \org\apogames\image\ApoImage.java : ApoCommando
=> \org\apogames\image\ApoImage.java : ApoNotSoSimple
=> \org\apogames\image\ApoImage.java : ApoSimple
=> \org\apogames\images\ApoImage.java : ApoSimpleSudoku
=> \org\apogames\image\ApoImage.java : ApoRelax
=> \org\apogames\image\ApoImage.java : ApoSnake
=> \org\apogames\image\ApoImage.java : ApoIcarus
=> \org\apogames\image\ApoImage.java : ApoPongBeat
</CLUSTER>

<CLUSTER with different relative paths among the cloned files>
=> \org\apogames\image\ApoImageFromValue.java : ApoRelax
=> \org\apogames\image\ApoImageFromValue.java : ApoNotSoSimple
=> \org\apogames\image\ApoImageFromValue.java : ApoCommando
=> \org\apogames\image\ApoImageFromValue.java : ApoImp
=> \org\apogames\image\ApoImageFromValue.java : ApoPongBeat
=> \org\apogames\image\ApoImageFromValue.java : ApoIcarus
=> \org\apogames\image\ApoImageFromValue.java : ApoSimple
=> \org\apogames\image\ApoImageFromValue.java : ApoMarc
=> \org\apogames\images\ApoImageFromValue.java : ApoSimpleSudoku
=> \org\apogames\images\ApoImageFromValue.java : ApoSkunkman
=> \org\apogames\image\ApoImageFromValue.java : ApoSnake
</CLUSTER>
```

Fig. 4-14. Contents of the suggestion for refactoring file

5. Guide 2: Configuring product variants based on the platform

This guide describes how CCIntegrator derives product variants from a constructed product line platform. CCIntegrator supports the automated configuration and generation of product variants based on the platform.

After the platform has been constructed following Guide 1, product variants that can be derived from the platform. To configure a product variant, enter “2” in the console input.

```
= MENU ================================================
1. Migrating from CAO based products to SPL repository.
2. Generating products from migrated SPL repository.
3. EXIT
========================================================
$ 2
```

Fig. 5-1. Initial menu with option “2” selected for product configuration

When CCIntegrator displays the list of product variants that can be generated from the platform, users select the product that you want to configure. If you want to end product generation and return to the main menu, press “0”. Figure 5-2 shows the selection of product “8”, corresponding to *ApoSimple*.

```
= List of product variants you can generate ============
0. RETURN MENU
1. ApoCommando
2. ApoIcarus
3. ApoImp
4. ApoMarc
5. ApoNotSoSimple
6. ApoPongBeat
7. ApoRelax
8. ApoSimple
9. ApoSimpleSudoku
10. ApoSkunkman
11. ApoSnake
========================================================
$ 8
```

Fig. 5-2. Selecting a product variant to be configured

Figure 5-3 illustrates the derivation process of deriving *ApoSimple* from the platform.

```
Generating a product variant that you choose.
     1: asset1 => (.\products\ApoSimple\org\apogames\entity\ApoButton.java)
     2: asset2 => (.\products\ApoSimple\org\apogames\help\ApoHelp.java)
     3: asset3 => (.\products\ApoSimple\apoSimple\ApoSimpleComponent.java)
     4: asset4 => (.\products\ApoSimple\org\apogames\help\ApoFloatPoint.java)
     5: asset5 => (.\products\ApoSimple\org\apogames\ApoLibraryGame.java)
     6: asset6 => (.\products\ApoSimple\org\apogames\image\ApoImage.java)
     7: asset7 => (.\products\ApoSimple\org\apogames\ApoDisplayConfiguration.java)
     8: asset8 => (.\products\ApoSimple\org\apogames\ApoSubGame.java)
     9: asset9 => (.\products\ApoSimple\org\apogames\entity\ApoEntity.java)
    10: asset10 => (.\products\ApoSimple\org\apogames\input\ApoKeyboard.java)
    11: asset11 => (.\products\ApoSimple\org\apogames\image\ApoImageFromValue.java)
    12: asset12 => (.\products\ApoSimple\org\apogames\ApoScreen.java)

       ... (ellipsis)

    93: asset324 => (.\products\ApoSimple\apoSimple\game\ApoSimpleHighscoreHelp.java)
    94: asset329 => (.\products\ApoSimple\apoSimple\entity\ApoSimpleEntityMenu.java)
    95: asset335 => (.\products\ApoSimple\apoSimple\entity\ApoSimpleString.java)
    96: asset338 => (.\products\ApoSimple\apoSimple\puzzle\ApoSimplePuzzleEntity.java)
It is completed to generate a product variant (ApoSimple) in the path (.\products\ApoSimple).
```

Fig. 5-3. Configuration progress of *ApoSimple*

CCIntegrator refers to model.xml to derive the selected product variant. During this process, platform annotations are preprocessed according to the selected product identifier, and the resulting code is mapped to product-specific

files. As shown in Figure 5-4, the configured product variant is stored in the “CCIntegrator/products” folder.

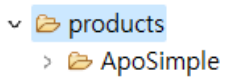

Fig. 5-4. Configuration result of *ApoSimple* in the CCIntegrator/products folder

Once the configured product variant is imported from the “CCIntegrator/products” folder into Eclipse, it can be executed without errors.

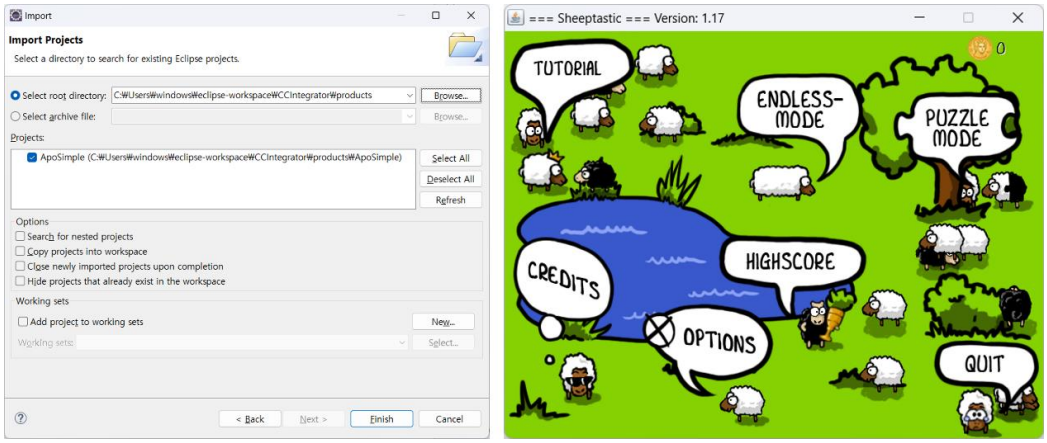

Fig. 5-5. Execution result of *ApoSimple* generated from the platform

References

Lee J, Kim T, Kang S. Recovering software product line architecture of product variants developed with the clone-and-own approach.In: Proceedings of COMPSAC 20,Madrid,Spain,13–17 July 2020. Piscataway, NJ: IEEE, 2020, 985–90.

Kim T, Lee J, Kim E. A code clustering technique for unifying method full path of reusable cloned code sets of a product family, KIPS Trans. Softw. and Data Eng. Vol.12, No.1 2023. (in Korean)

Kim T, Lee J, Kang S. Extracting common and variable code using the LCS algorithm for migration to SPLE. In: Proceedings of COMPSAC 23, Torino, Italy, 26–30 June. Piscataway, NJ: IEEE, 2023, 1004–5.

Kim T, Lee J, Kang S. Cloned code clustering for the software product line engineering approach to developing a family of products. In: Proceedings of COMPSAC 24, Osaka, Japan, 2–4 July. Piscataway, NJ: IEEE, 2024, 1356–61.

Kim T, Lee J, Kang S. Software product line platform construction for migration from the clone-and-own approach to developing software product families, Computer J., 2025, 1-17, https://doi.org/10.1093/comjnl/bxaf134.