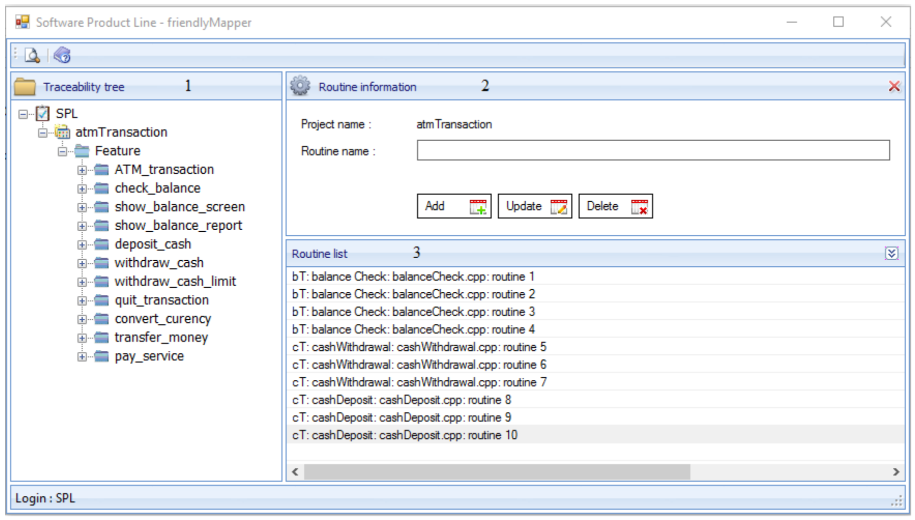
Tool Support: friendlyMapper

1 Introduction

This section presents the friendlyMapper tool, which supports the feature mapping approach. In particular, it supports the feature-code mapping and feature mapping conformance macro steps. The tool supports a set of tasks that can be used by software engineers to create, store, and maintain the relationships between features belonging to the current FM and feature-related-code fragments (i.e., routines) of the code, using the traceability tree. The tool includes modeling and visualizing the traceability tree, which consists of traceability links that relate a feature to the feature-related-code fragments (i.e., routines).

Figure 1 is a screenshot of the friendlyMapper tool. It shows an example of ATM resulting SPL (final SPL). The figure shows three windows: (1) Traceability tree, (2) Routine information, and (3) Routine list. The first window (Traceability tree) shows the features of the current FM taken directly from the XML file related to a feature tree, by selecting Import features when showing a context menu, upon right clicking ăon ăthe Feature ăitem (see Figure 1).



SPL
atmTransaction
Feature
ATM_transaction
check_balance
show_balance_screen
show_balance_report
deposit_cash
withdraw_cash
withdraw_cash_limit
quit_transaction
convert_curency
transfer_money
pay_service

Project name : atmTransaction
Routine name :

bT: balance Check: balanceCheck.cpp: routine 1
bT: balance Check: balanceCheck.cpp: routine 2
bT: balance Check: balanceCheck.cpp: routine 3
bT: balance Check: balanceCheck.cpp: routine 4
cT: cashWithdrawal: cashWithdrawal.cpp: routine 5
cT: cashWithdrawal: cashWithdrawal.cpp: routine 6
cT: cashWithdrawal: cashWithdrawal.cpp: routine 7
cT: cashDeposit: cashDeposit.cpp: routine 8
cT: cashDeposit: cashDeposit.cpp: routine 9
cT: cashDeposit: cashDeposit.cpp: routine 10

Login : SPL

Figure (1) Screenshot for the main windows of friendlyMapper.

The second window (Routine Information) presents a window that allows software engineers to enter the routine information of the resulting SPL (e.g., code-fragments of the ATM resulting SPL) by selecting the Routine information item when showing the context menu, upon right-clicking on the resulting SPL

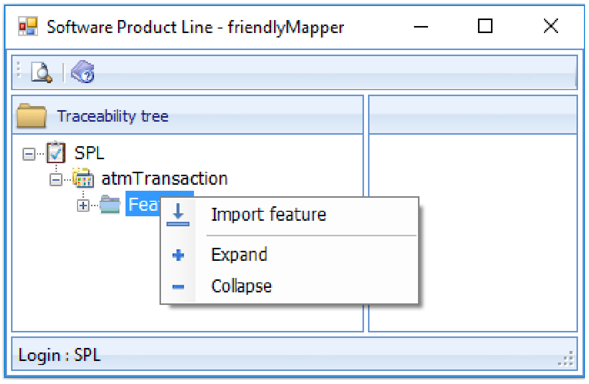


Figure (2) Import features of the current FM from XML file in friendlyMapper tool.

package (e.g., atmTransaction), as shown in Figure 3. The Routine information window allows software engineers to add, update, and delete routines (see Window 2 of Figure 1).

To add a routine, software engineers can enter the routine information (i.e., sub-package: module: source file.cpp: routine name) in the text field labeled with Routine name text, and then they can press the Add button associated with window 2 (Routine information) of the tool. To update a routine name, software engineers can select routine from the list labeled with Routine list text, edit the routine name when it appears in the text field labeled with routine name text, and press the Update button associated with Window 2 (Routine information) of the tool. To delete a routine from the list, software engineers can select a routine from the list labeled with Routine list text and then press the delete button associated with Window 2 (Routine information) of the tool. In consequence, the routine will be removed from the routine list.

The third window (Routine list) in the figure shows the list of routines in the resulting SPL (final SPL) that are responsible for a feature implementation in the code. There is another window related to this window (Routine list), called Feature routine information. Figure 1 depicts the fourth window ă(Feature routine information), which appears by selecting the Add routine item from the context menu (see Figure 3), when right-clicking on the Routine sub-branch of a feature (e.g., check_balance). This window allows software engineers to select a routine that is related to a specific feature (e.g., check_balance) from the left-hand side list and add it to the right-hand side list, using the buttons occupied in the middle position of the two lists. At the same time, the routine will be added to the Routine sub-branch of the specified feature on the feature level (e.g., check_balance), as shown in Figure 4.

Traceability tree of the friendlyMapper The tool uses a traceability tree to create and update the traceability links whenever feature changes occur.

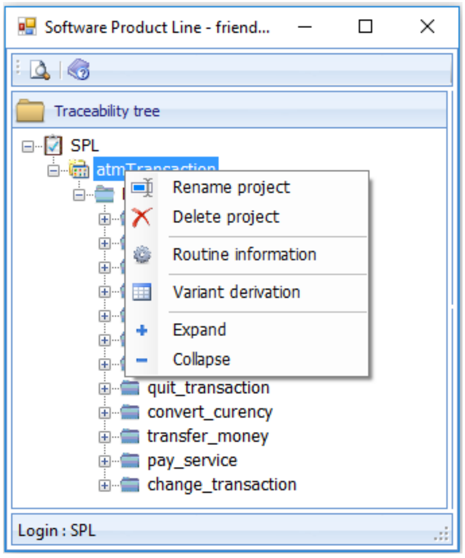
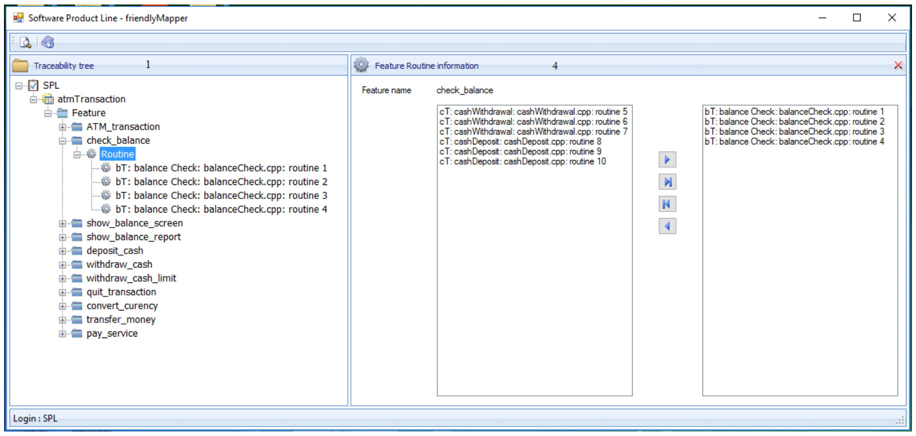


Figure (3) Add routine information of the code in the friendlyMapper tool.


Software Product Line - friendlyMapper
Traceability tree
1
Feature Routine information
4
Login : SPL

```
SPL
  atmTransaction
    Feature
      ATM_transaction
      check_balance
        Routine
          bT: balance Check: balanceCheck.cpp: routine 1
          bT: balance Check: balanceCheck.cpp: routine 2
          bT: balance Check: balanceCheck.cpp: routine 3
          bT: balance Check: balanceCheck.cpp: routine 4
      show_balance_screen
      show_balance_report
      deposit_cash
      withdraw_cash
      withdraw_cash_limit
      quit_transaction
      convert_curency
      transfer_money
      pay_service
```

Feature name check_balance

```
cT: cashWithdrawal: cashWithdrawal.cpp: routine 5
cT: cashWithdrawal: cashWithdrawal.cpp: routine 6
cT: cashWithdrawal: cashWithdrawal.cpp: routine 7
cT: cashDeposit: cashDeposit.cpp: routine 8
cT: cashDeposit: cashDeposit.cpp: routine 9
cT: cashDeposit: cashDeposit.cpp: routine 10
```

```
bT: balance Check: balanceCheck.cpp: routine 1
bT: balance Check: balanceCheck.cpp: routine 2
bT: balance Check: balanceCheck.cpp: routine 3
bT: balance Check: balanceCheck.cpp: routine 4
```

Figure (4) Feature routine information window of the friendlyMapper tool.

The traceability is defined as a one-to-many relationship (a feature is related to a routine or more). Traceability tree requires (1) features of the current FM (taken from an XML file) and (2) routine information about the code. The tool allows software engineers to use a traceability tree to store and maintain relationships from a feature to routines. The tool semi-automatically manages the relationship between each feature of the current FM and routines corresponding to the feature in the code.

As shown in Window 1 of Figure 4, a traceability tree is a kind of dropdown tree, starting its third level at the Feature item. When dropping down the Feature

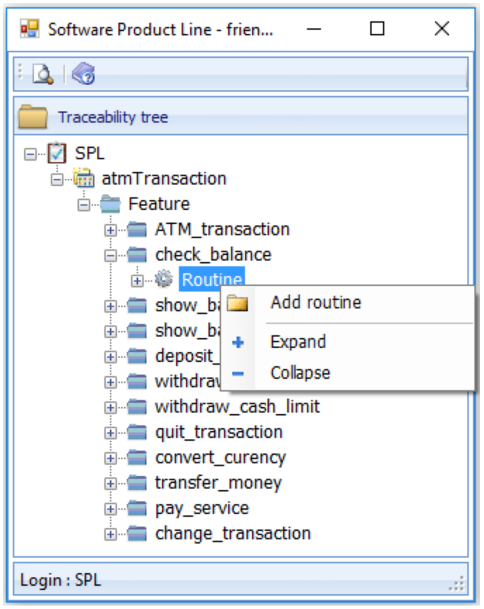


Figure (5) Add routine to a specific feature in friendlyMapper tool.

level, the list of features (which are imported from the XML file that represents the current FM) appears sequentially. Each feature has a sub-branch called Routine, which shows routines that are related to the feature. The Traceability tree maps the artifacts of the resulting SPL (the current FM and code) to each other. It allows software engineers to trace a feature of the current FM to routines just by dropping down each level, starting at the SPL project, the resulting SPL package, features, and moving toward routines of the selected feature (e.g., check_balance).

Routine information Routine is one of the code elements (software units) that may implement a feature in the feature mapping activity. In addition, the tracing rules define how to relate features belonging to the current FM to feature-related-code fragments (e.g., routines) belonging to the code. FriendlyMapper introduces feature-related-code fragments as a code portion that maintains feature implementation, which is defined in the tool as a routine.

In friendlyMapper, the traceability tree allows software engineers to trace a feature to routines with their positions in the code, just by defining the routine location in the resulting SPL main package, starting at sub-package, module, source file, and moving toward routine name (i.e., sub-package: module: souce-file.cpp: routine), but it ignores attributes, statements, and expressions (see Routine list of window 3 in Figure 1 and Figure 4).

2 Tool Capability

As mentioned earlier in this section, the friendlyMapper tool is implemented in this thesis to support the feature mapping activity and its related macro steps (i.e., feature-code mapping and feature mapping conformance macro steps). Thus, this subsection presents the main capabilities of the tool according to them, as follows:

- Feature-code mapping macro step: to support this macro step, the tool is able to perform the following:

    1. Create a new SPL project by selecting the new project item from the context menu, as depicted in the right-hand side window of Figure 6, and right-clicking on the SPL item (the first level of the traceability tree), as depicted in the left-hand side window of Figure 6.

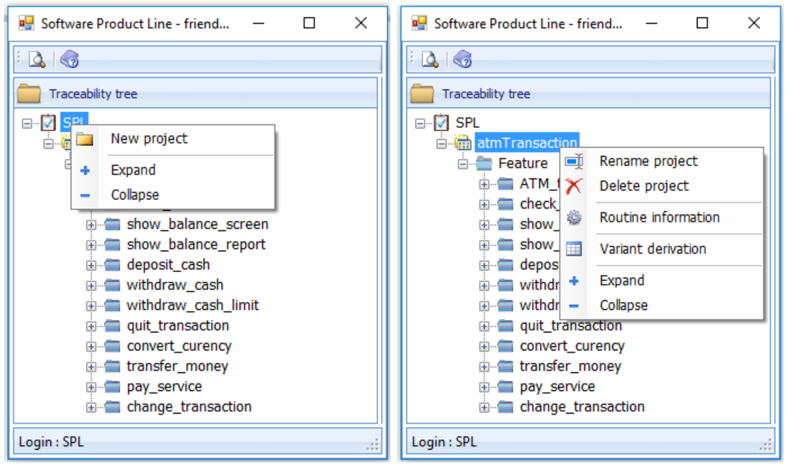


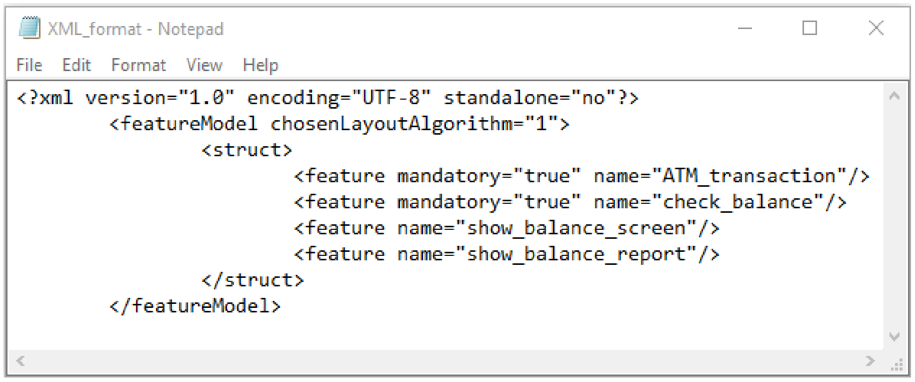
Figure (6) Sample example of creating and deleting an SPL project in friendlyMapper.

    2. Remove an SPL project by selecting the delete project item from the context menu when right-clicking on the project name (e.g., atmTransaction, the second level of the traceability tree), as depicted in the right-hand side window of Figure 6.

    3. Import features of the current FM model a given resulting SPL from the XML file written in a specific format, as shown in Figure 7.

    4. ă ăCreate a features list of the traceability tree under the Feature item (on the Feature level), as depicted in Figure 2. The result of the feature list of the traceability tree is depicted in Window 1 of Figure 1.

    5. Create a traceability link from a feature (e.g., check_balance) of the features list to one or more routines (e.g., bT: balance Check:



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
        <featureModel chosenLayoutAlgorithm="1">
                <struct>
                        <feature mandatory="true" name="ATM_transaction"/>
                        <feature mandatory="true" name="check_balance"/>
                        <feature name="show_balance_screen"/>
                        <feature name="show_balance_report"/>
                </struct>
        </featureModel>
```

Figure (7) The XML file format related to the current FM and accepted by friendlyMapper.

balanceCheck.cpp: routine 1), by selecting the Add routine item of the context menu when selecting the feature (see Figure 4).

- feature mapping conformance macro step: to support this macro step, the tool is able to perform the following:

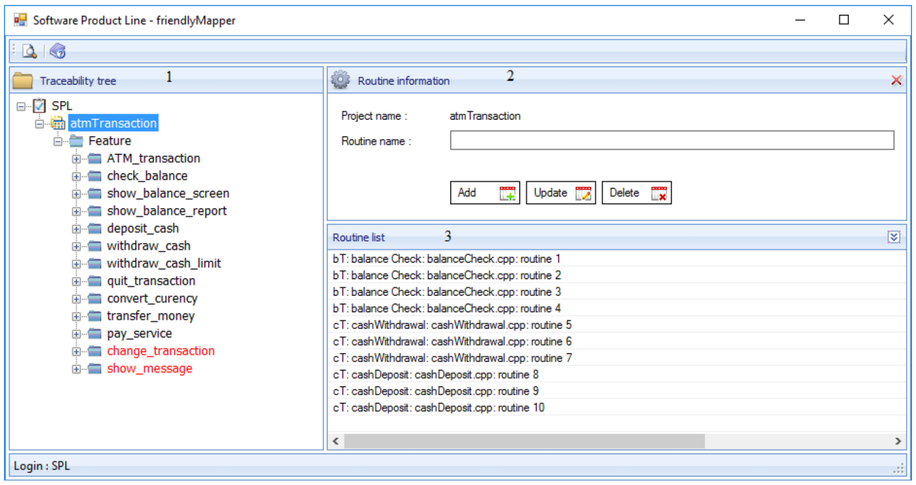
    1. Update automatically the traceability tree and routine information of the related screens (windows), whenever changes occur on all the levels (project, features, and subroutines) of its drop-down tree, which ensures that the features and code are consistent.

       For example, software engineers can relate an existing routine to a selected feature, and the traceability tree automatically shows the new traceability link in its tree. Also, software engineers can add new routines to ăRoutine lists, and the tool automatically updates Routine lists in Window 2 (Routine information) and Window 4 (Routine feature information) of Figure 1.

    2. Update the features list of the traceability tree with new features whenever a change occurs to the current FM by enabling software engineers to (i) re-import the features again by reading the XML file containing the updated features and (ii) give the new features a red color to alert software engineers about the existence of the new features (e.g., change_transaction and show_message), as shown in Figure 8.

    3. Remove a feature from the feature list of the traceability tree by selecting Delete feature from the context menu when right-clicking on the feature name. The feature will be removed from the features list upon approval of the deletion from the confirmation dialog box, as shown in Figure 9.

    4. Add a new traceability link to the traceability tree by first selecting the feature, opening its sub-branch (which is indicated with a plus


Software Product Line - friendlyMapper

Traceability tree 1

- SPL
  - atmTransaction
    - Feature
      - ATM_transaction
      - check_balance
      - show_balance_screen
      - show_balance_report
      - deposit_cash
      - withdraw_cash
      - withdraw_cash_limit
      - quit_transaction
      - convert_curency
      - transfer_money
      - pay_service
      - change_transaction
      - show_message

Routine information 2

Project name : atmTransaction

Routine name :

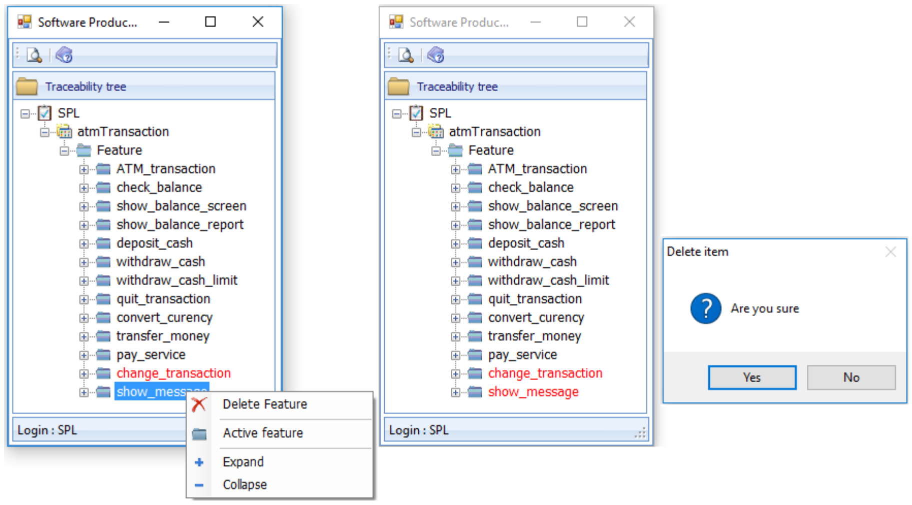
Add Update Delete

Routine list 3

```
bT: balance Check: balanceCheck.cpp: routine 1
bT: balance Check: balanceCheck.cpp: routine 2
bT: balance Check: balanceCheck.cpp: routine 3
bT: balance Check: balanceCheck.cpp: routine 4
cT: cashWithdrawal: cashWithdrawal.cpp: routine 5
cT: cashWithdrawal: cashWithdrawal.cpp: routine 6
cT: cashWithdrawal: cashWithdrawal.cpp: routine 7
cT: cashDeposit: cashDeposit.cpp: routine 8
cT: cashDeposit: cashDeposit.cpp: routine 9
cT: cashDeposit: cashDeposit.cpp: routine 10
```

Login : SPL

Figure (8) Features list of the traceability tree after re-importing features from the XML file including new ones.


Software Produc...

Traceability tree

- SPL
  - atmTransaction
    - Feature
      - ATM_transaction
      - check_balance
      - show_balance_screen
      - show_balance_report
      - deposit_cash
      - withdraw_cash
      - withdraw_cash_limit
      - quit_transaction
      - convert_curency
      - transfer_money
      - pay_service
      - change_transaction
      - show_message

Login : SPL

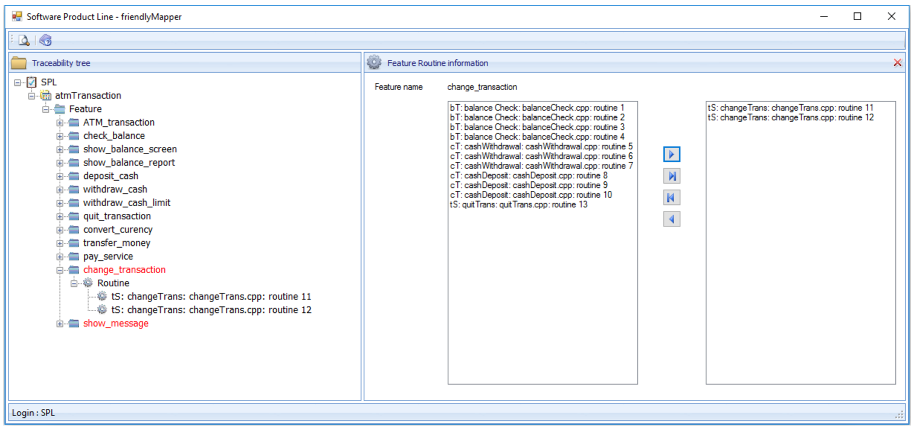
Delete item

Are you sure

Figure (9) Delete show_message feature from the traceability tree.

sign (+) in front of the feature name), and selecting the Add routine' item from the context menu upon right-clicking the Routine sub-branch of the selected feature. Once window 4 (Feature information routine) appeared, select the routine from the left-hand side text and move it to the right-hand side text, using the buttons occupied in the middle position of the sides (see Figure 5, Figure 4, and Figure 10).

5. Import new features with a red color to avoid software engineers from missing updating the code (see Figure 8).


Software Product Line - friendlyMapper
Traceability tree
Feature Routine information
Login : SPL

```
SPL
  atmTransaction
    Feature
      ATM_transaction
      check_balance
      show_balance_screen
      show_balance_report
      deposit_cash
      withdraw_cash
      withdraw_cash_limit
      quit_transaction
      convert_curency
      transfer_money
      pay_service
      change_transaction
        Routine
          tS: changeTrans: changeTrans.cpp: routine 11
          tS: changeTrans: changeTrans.cpp: routine 12
      show_message
```

Feature name change_transaction

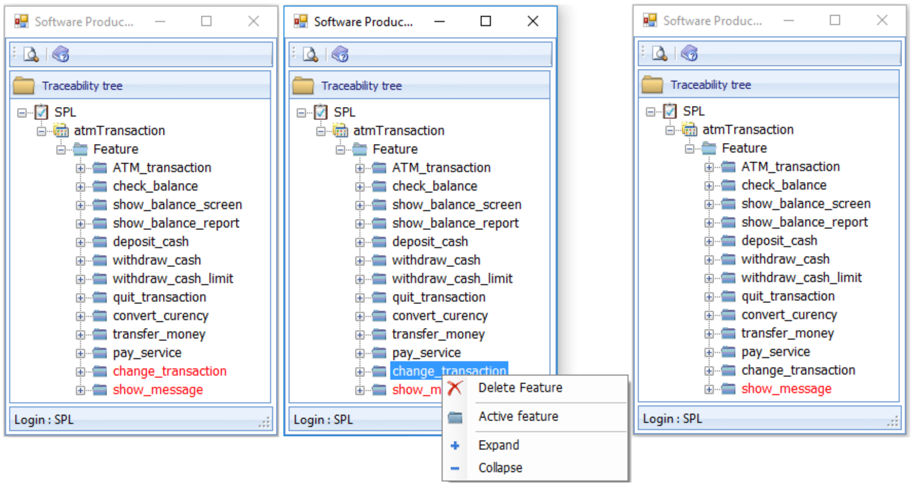
```
bT: balance Check: balanceCheck.cpp: routine 1
bT: balance Check: balanceCheck.cpp: routine 2
bT: balance Check: balanceCheck.cpp: routine 3
bT: balance Check: balanceCheck.cpp: routine 4
cT: cashWithdrawal: cashWithdrawal.cpp: routine 5
cT: cashWithdrawal: cashWithdrawal.cpp: routine 6
cT: cashWithdrawal: cashWithdrawal.cpp: routine 7
cT: cashDeposit: cashDeposit.cpp: routine 8
cT: cashDeposit: cashDeposit.cpp: routine 9
cT: cashDeposit: cashDeposit.cpp: routine 10
tS: quitTrans: quitTrans.cpp: routine 13
```

```
tS: changeTrans: changeTrans.cpp: routine 11
tS: changeTrans: changeTrans.cpp: routine 12
```

Figure (10) Traceability tree after adding a new traceability link from the change_transaction feature to its corresponding routines.

As shown in Figure 10, the new features of the traceability tree are imported in a red color. This enables developers to confirm that they update routines of the code and provide feedback on changes made, which helps to preserve consistency between the current FM and the code of the resulting SPL.

6. Change the color of the new feature from red to black (see Figure 11) when the software engineers update the feature-related-code fragments.



```
SPL
  atmTransaction
    Feature
      ATM_transaction
      check_balance
      show_balance_screen
      show_balance_report
      deposit_cash
      withdraw_cash
      withdraw_cash_limit
      quit_transaction
      convert_curency
      transfer_money
      pay_service
      change_transaction
      show_message
```

Figure (11) Change color of the change_transaction feature from red color to black color.