Chapter 1

FINT

This document contains the FINT user manual.

FINT [1] is a (Java) source code analysis tool for detecting smells of crosscutting implementation of concerns. The tool is available as a plug-in for the Eclipse IDE (v.3.0.x – v.3.3).[2]

1.1 Installation

The installation procedure simply requires to download and save the "jar" distribution of the tool into the "plugins" directory of Eclipse and then (re-)start the IDE.

1.2 User manual

FINT implements three code analysis techniques that we shall see next in action:

- *Fan-in analysis* looks for crosscutting concerns by investigating all the method call relations in a system, and allowing the user to select those methods that are called from many different places, possibly from similar calling contexts. For example, a tracer for all the method executions in a system might consist of calls to a tracing method attached at the beginning of each of the system's methods. The tracer method will have a large number of callers, and hence a high fan-in metric value, which makes it likely to be identified by our technique.

  A typical refactoring to aspect-oriented programming (AOP) of the concerns identified by fan-in analysis consists of replacing the scattered method calls identified by this technique with pointcut and advice constructs.

[1] `http://swerl.tudelft.nl/view/AMR/FINT`

[2] Some of the figures may be more difficult to read on paper. We refer the reader to the FINT web site for the on-line version of this manual, in which the figures are available in high resolution.

- *Grouped calls analysis* targets similar code smells and concerns as the previous technique; however, instead of single method invocations, this technique is looking for groups of (at least two) methods that are called by the same callers. Examples of crosscutting concerns following this implementation idiom include programmatic (JTA) transaction management, where the transaction demarcation is realized by calls to methods such as `begin`, `commit`, and `rollback`.

- *Redirections finder* aims at identifying wrapper classes (such as decorators) by analyzing the methods of all the classes in the system under investigation for exclusive one-to-one call relations with methods of another class.

Each technique has a dedicated view for investigating and further refining the results. A fourth view allows us to collect and save the results that participate in, and hence point us to, crosscutting concerns; these results are crosscutting concern *seeds*. The views can be opened from Eclipse's "Window/Show View/Other..." menu, under the FINT group, as shown in Figure 1.1.

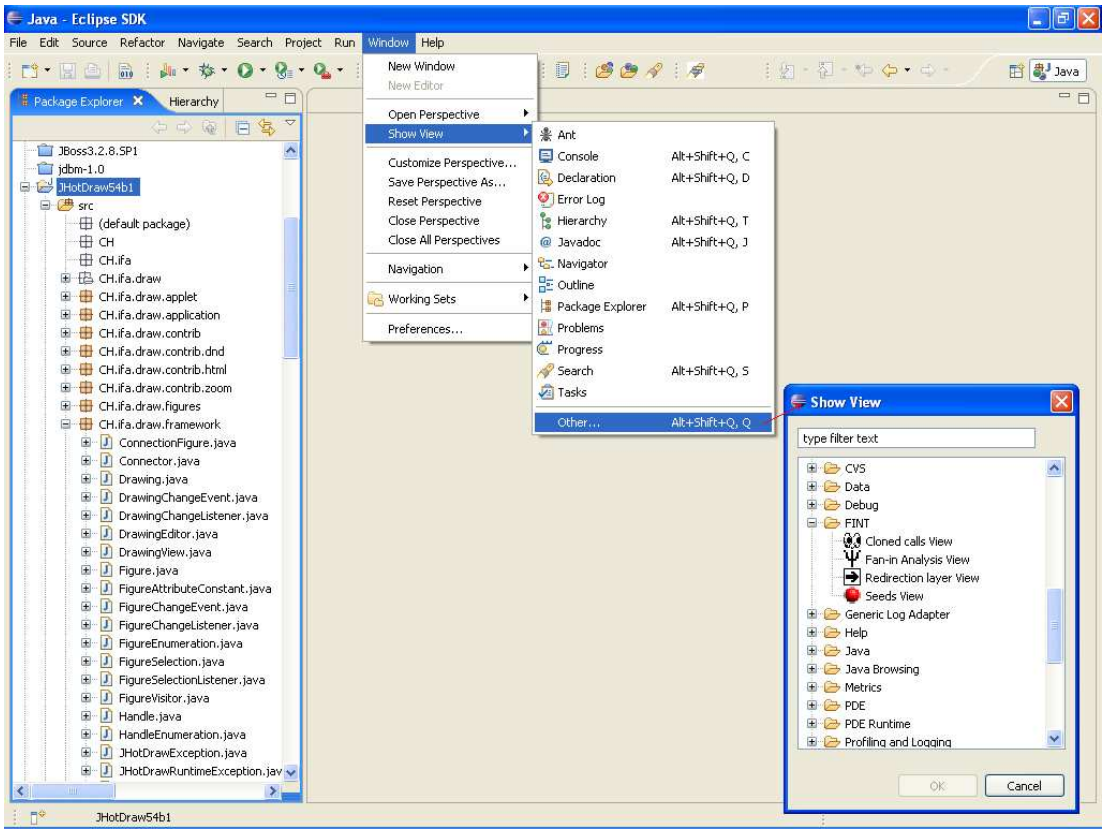

Figure 1.1: The FINT views.

1.2.1 Fan-in analysis

Fan-in is the default analysis in the tool. To run the analysis, the user chooses the program elements to be analyzed in the *Package Explorer* view of Eclipse and selects from the context menu of these elements (right click) the *Fan-in Analysis* option, as illustrated in Figure 1.2.

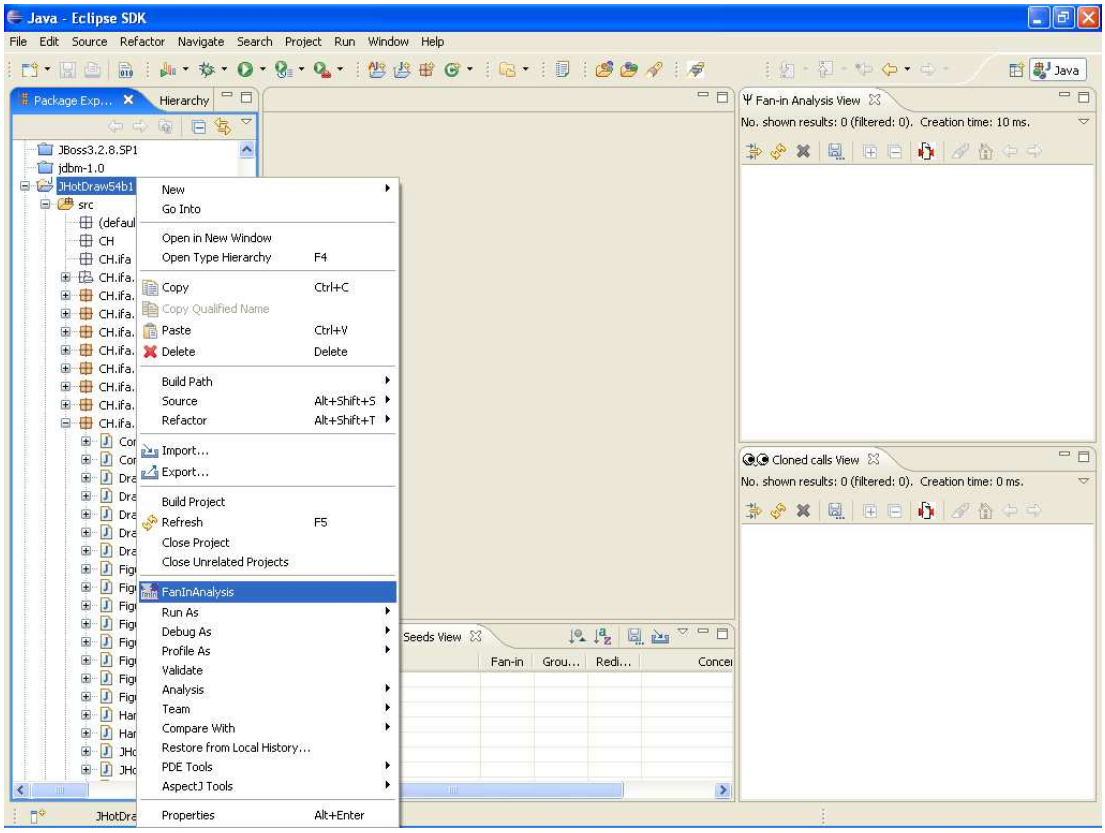

Figure 1.2: Run the analysis from the context menu.

The tool first parses the source code of the selected elements, and builds an internal, in-memory model of the source code that will be used by all the analyses available in FINT (Figure 1.3). The model building (for fan-in analysis) takes about 30 seconds for a system of 20,000 non-comment lines of code (NCLOC) and around 5 minutes for over 360,000 NCLOC, on a Pentium 4 machine (2.66 GHz).

Fan-in analysis looks at all call relations in the system under investigation and displays the results in a dedicated view, as shown in Figure 1.4. In this view, each callee method is the tree root of its callers and has attached to its name the number of callers. The fan-in value of each callee is indicated next to its name.

As shown in the detailed Figure 1.5 the view presents the user with a number of options, like:

- Sorting the results by their name or by their fan-in value;

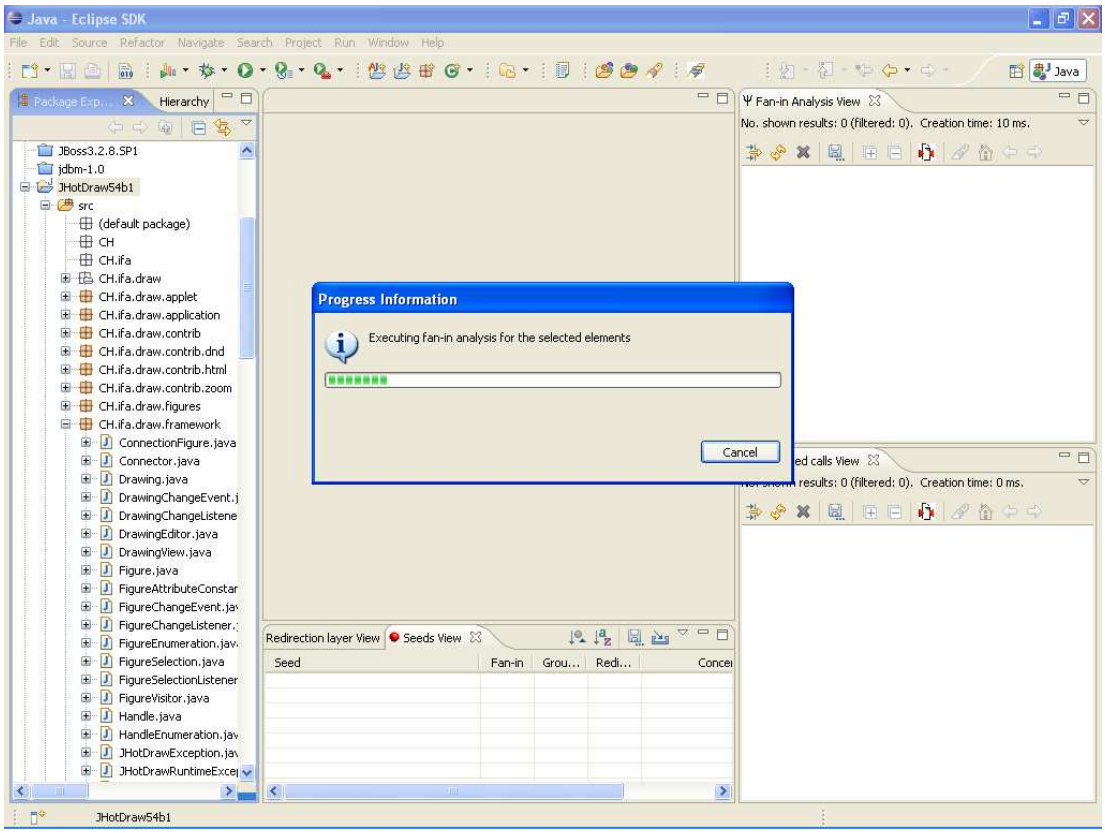

Figure 1.3: A progress bar shows the time left to complete the internal model and to execute the fan-in analysis.

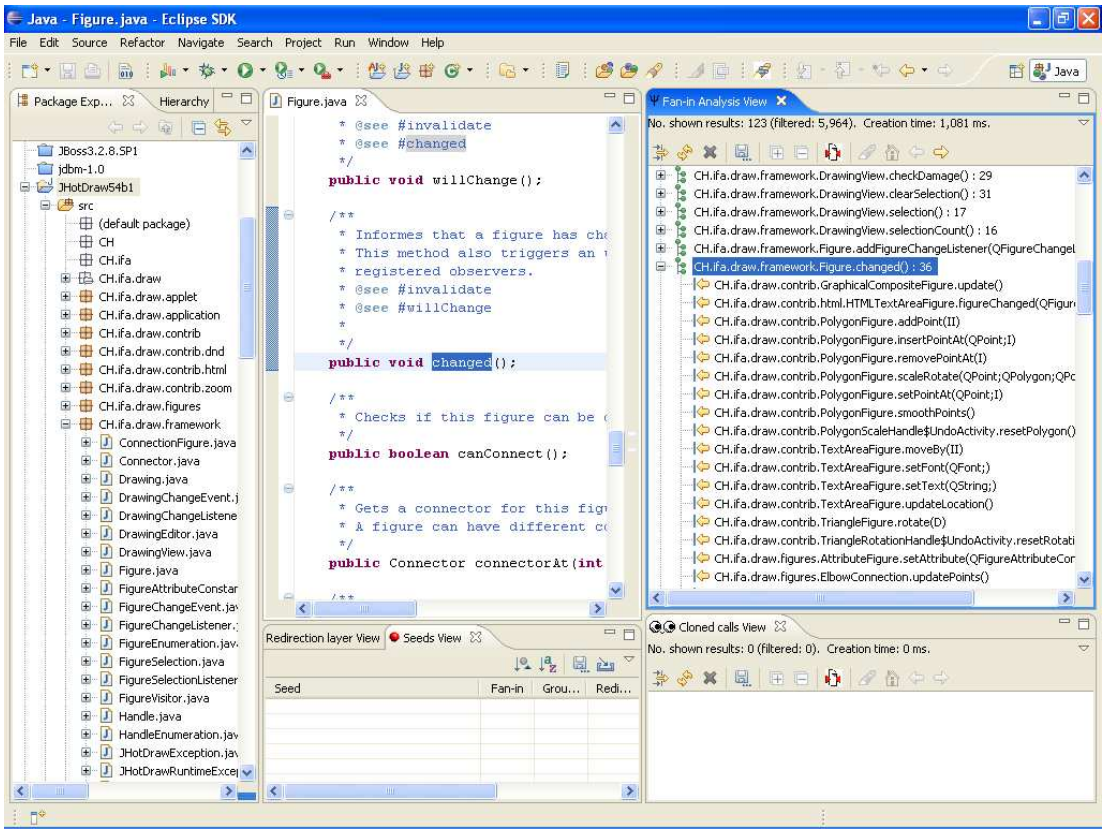

Figure 1.4: The results of fan-in analysis displayed in the dedicated view.

- Showing/Hiding the library-methods in the view (that is, methods that are called from the analyzed element but not declared in this element. Such elements include, for instance, the JDK libraries);

- Showing/Hiding the accessor-methods in the view. The tool checks accessor-methods by their name (get* and set* methods) or by implementation (methods that simply return a reference or set the value of a field).

Note that the filter for accessors also checks methods in interfaces and eliminates those methods for which all implementations are accessors.

By double-clicking any element in the view, the user can inspect the code for that element. Further on, the menus in the view allow us to:

- Change the settings of the analysis, like the fan-in threshold value;

- Refresh the model by re-analyzing (/re-building the model for) the last analyzed element;

- Clean the model built for the analyzed element and release memory;

- Save the results to file.

Besides the filters for the accessor and library methods, also shown in Figure 1.6, the set of filters include:

- Callees filters
  - Fan-in threshold: Methods with a fan-in value below the chosen threshold are not shown in the view;
  - ”Utility” methods: methods that the user chooses to ignore and that will not be shown in the view;

- Callers filters
  - Methods that should not contribute to the fan-in value of their callees.

To select ”utility” elements, the user is presented with the Java element hierarchy of the analyzed Java element. The user can check the ”utility” elements in the dialog window of the Fan-in Analysis Setup. Such utility elements could include, for instance, (JUnit) test packages, as in the example shown in Figure 1.6.

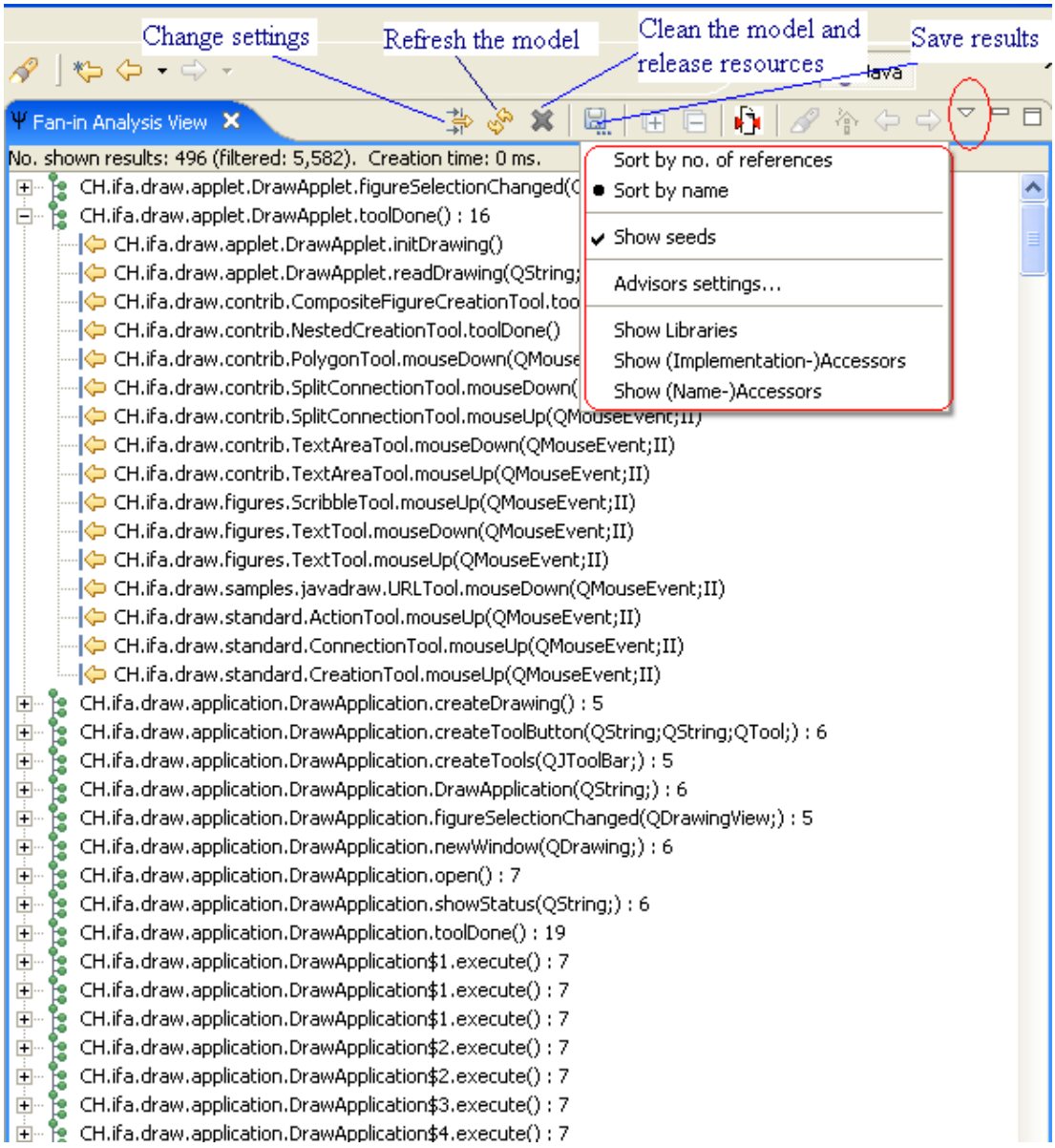

Figure 1.5: The Fan-in analysis view and the menus for various options.

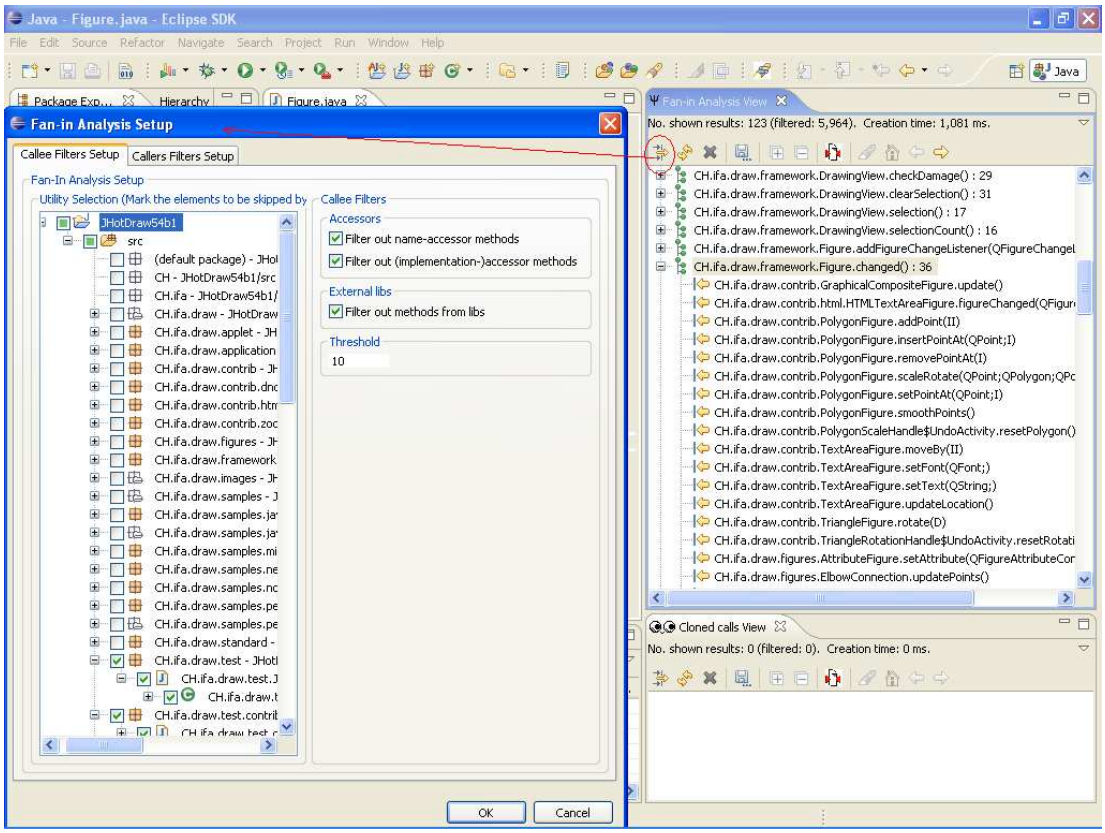

Figure 1.6: The dialog to set the filters for the fan-in analysis results.

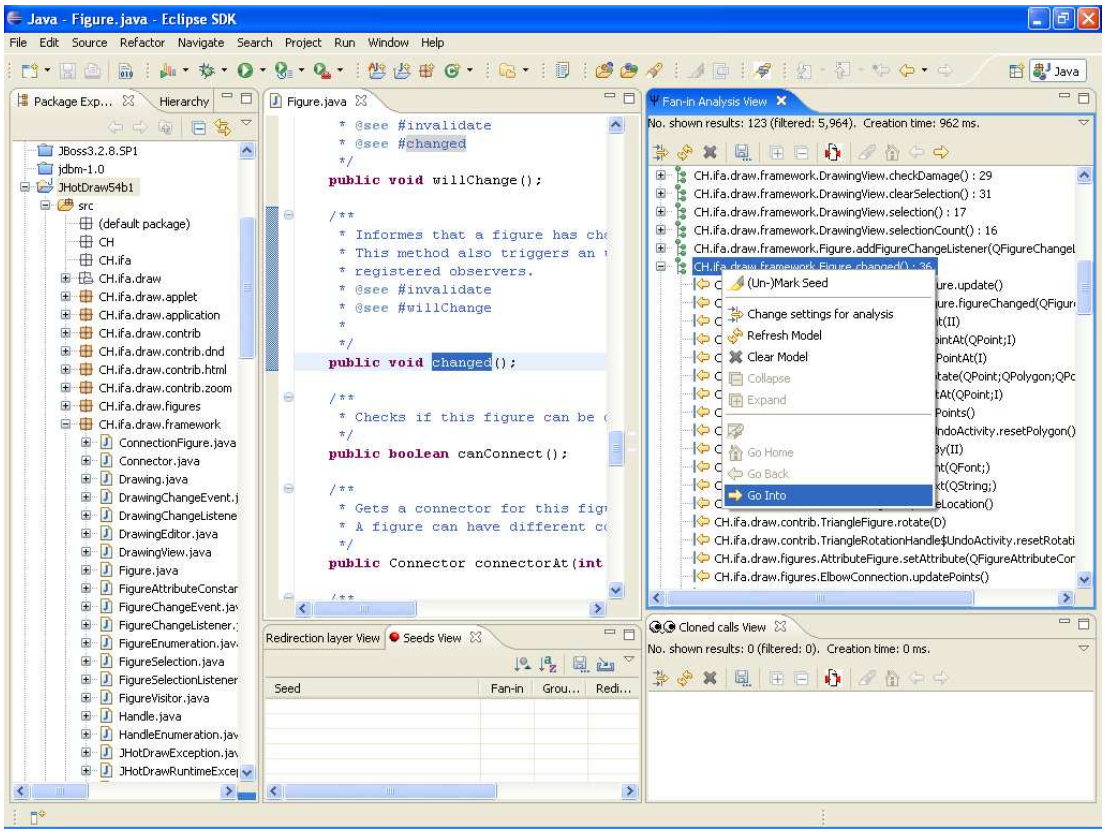

Figure 1.7: Select a method to analyze its incoming call realtions.

Reasoning about a candidate

The filtered callee-methods are our *candidates* for crosscutting concern seeds. To reason about a candidate, we select it in the view and choose the "Go Into" option from its context menu, as illustrated in Figure 1.7. This command opens the list of its callers and activates the toolbar button for launching various analyses for the callers.

FINT assists the user through several analyses to decide whether a (high fan-in) method is a concern seed. These analyses can be accessed from the menu of the Fan-in analysis view, as illustrated in Figure 1.8.

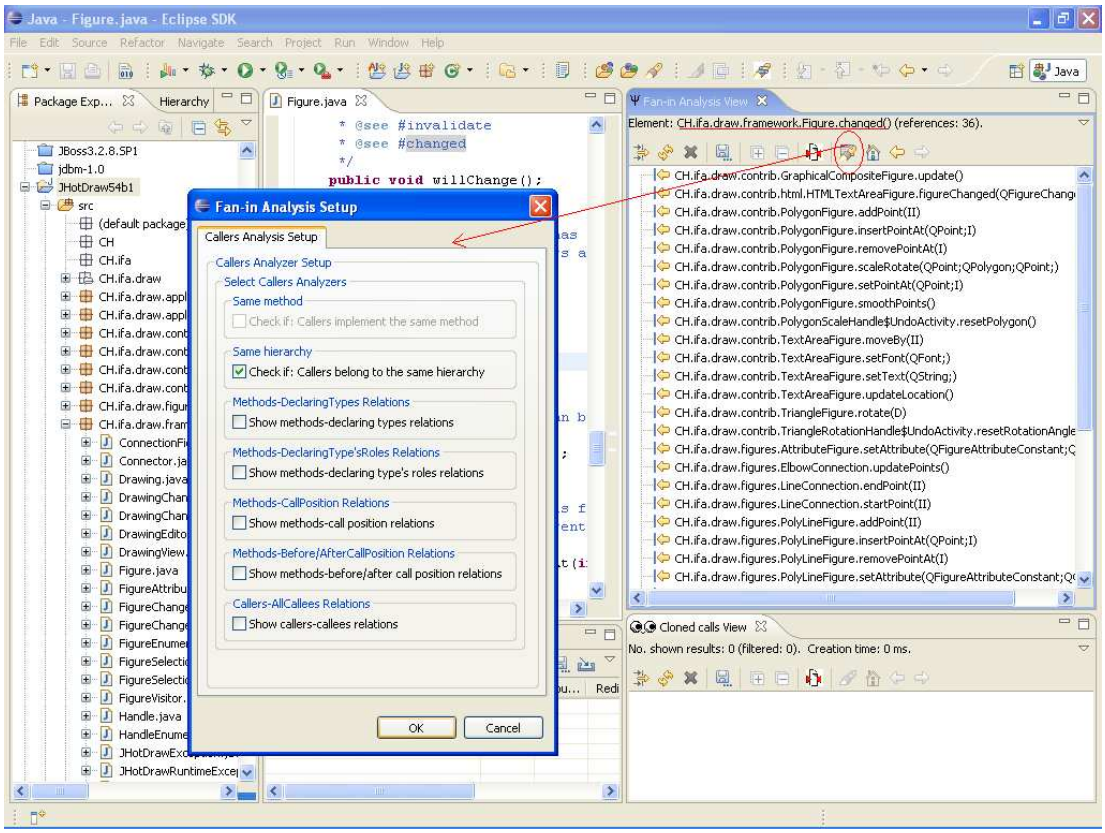

Figure 1.8: The dialog to select analyses for the callers of a method (with a high fan-in value).

The option for analyzing the hierarchies of the callers will check the top-level declaring type (i.e., interfaces/classes) of each caller, and highlight with the same color those methods that are declared by the same type (see Figure 1.9).

The option for analyzing the position of the calls to the (analyzed) method with a high fan-in value opens a window that shows all the callers of the method and the position of the calls to this method. The positions are relative to the caller's body. This analysis is illustrated in Figure 1.10.

A similar analysis is shown in Figure 1.11: in this case, we look at all the call

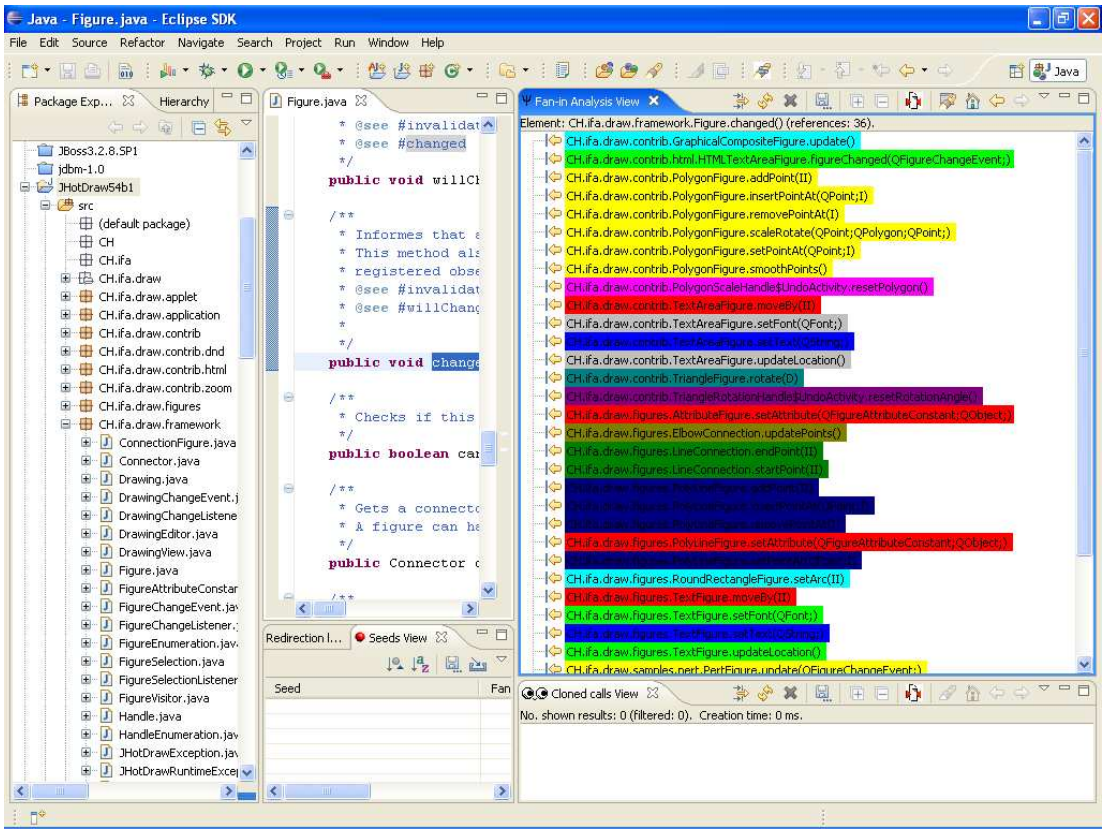

Figure 1.9: Same-hierarchy analysis.

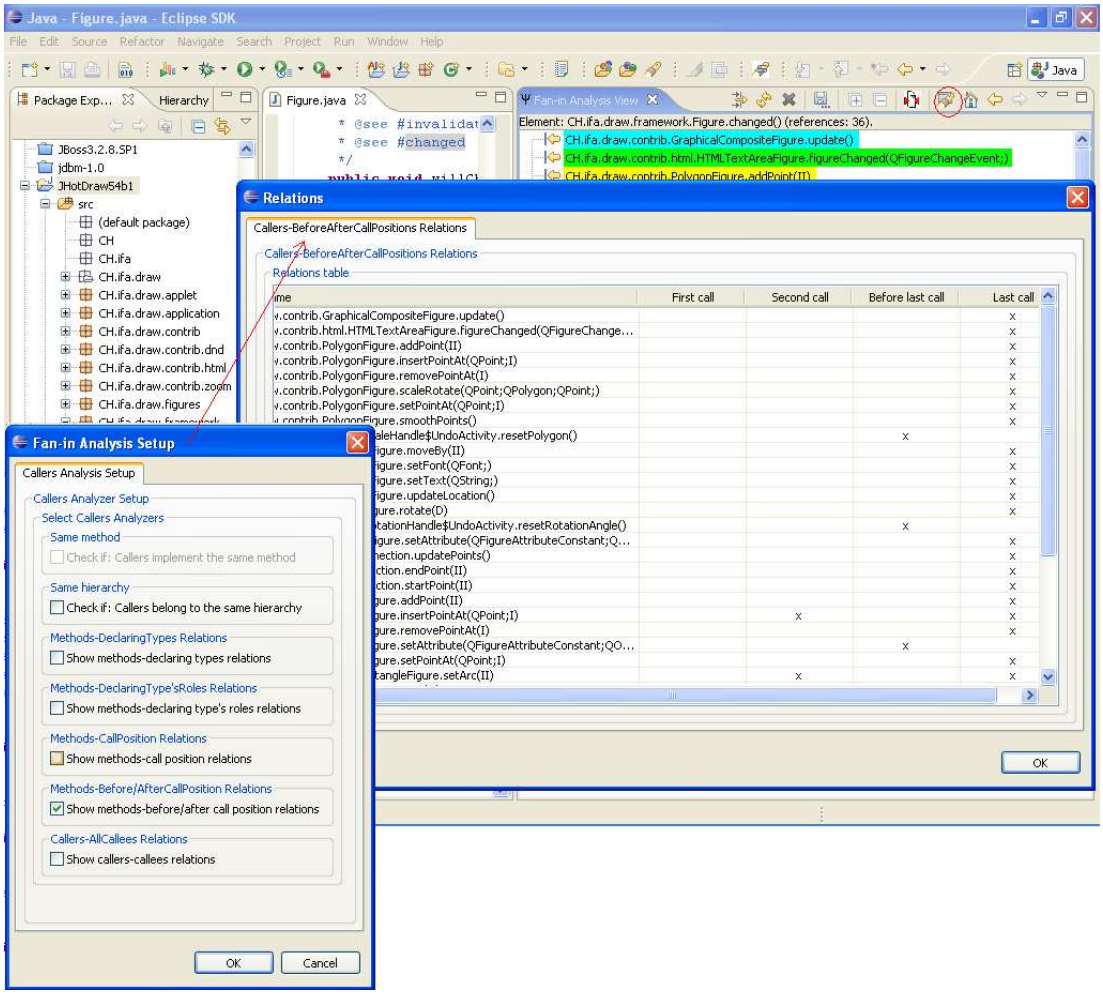

Figure 1.10: Analyzing the position of the calls in the caller-method's body.

relations for the callers of our method to see whether these callers have other callees in common besides the method we analyze.

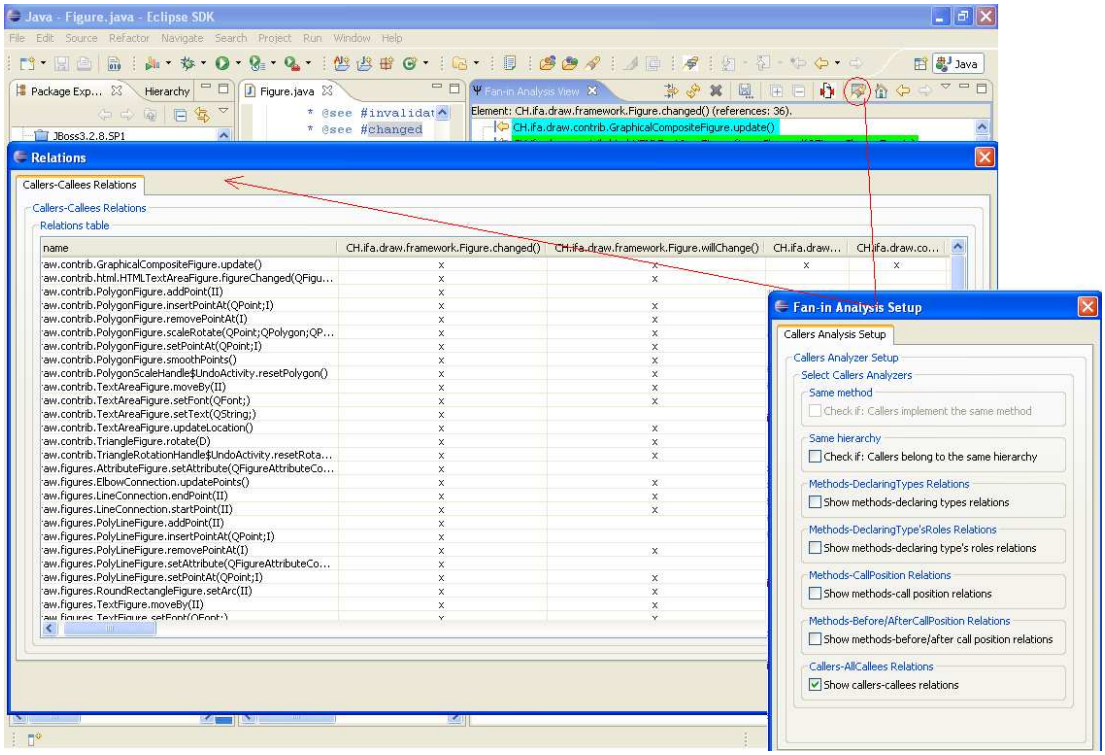

Figure 1.11: Callers-callees analysis: Display all the callees for all the callers of the method with the a high fan-in value.

If we decide that a method with a high fan-in value is part of a crosscutting concern implementation (i.e., it is a concern *seed*), we can mark it as such by selecting the Mark Seed option from the context menu of the candidate (right-click the candidate in the view). The method will be marked distinctively and displayed in the Seeds view, as in Figure 1.12.

1.2.2 Grouped calls analysis

Grouped calls analysis requires the model previosuly built by fan-in analysis. This new analysis can be run from the Grouped calls view, as shown in Figure 1.13.

The candidate-seeds consist of groups of methods that share their callers. The candidates are displyed in the view as a tree hierarchy, with each group of methods at the root of the list of their common callers, as illustrated in Figure 1.14.

The results of Grouped calls analysis can be sorted and filtered similarly to Fan-in analysis. These filters include checking for setters/getters, checking for libraries methods, as well as for “utilities”.

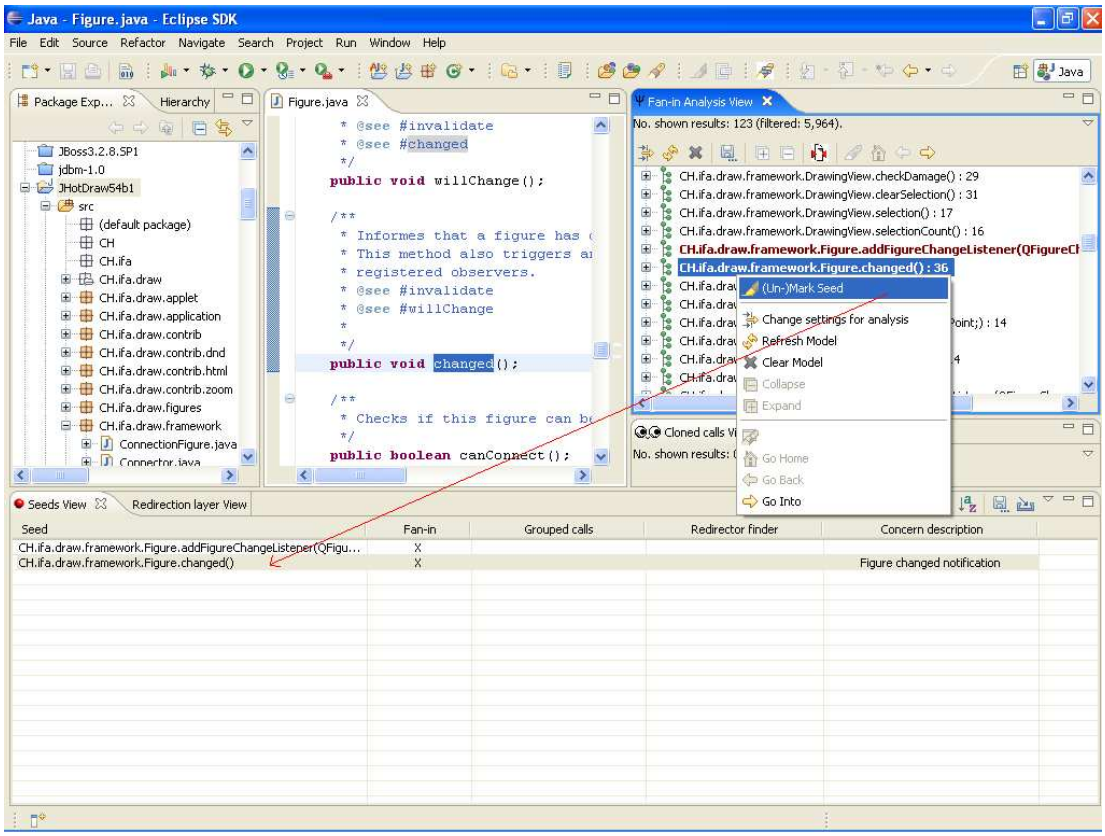

Figure 1.12: Marking selected method as a crosscutting concern seed.

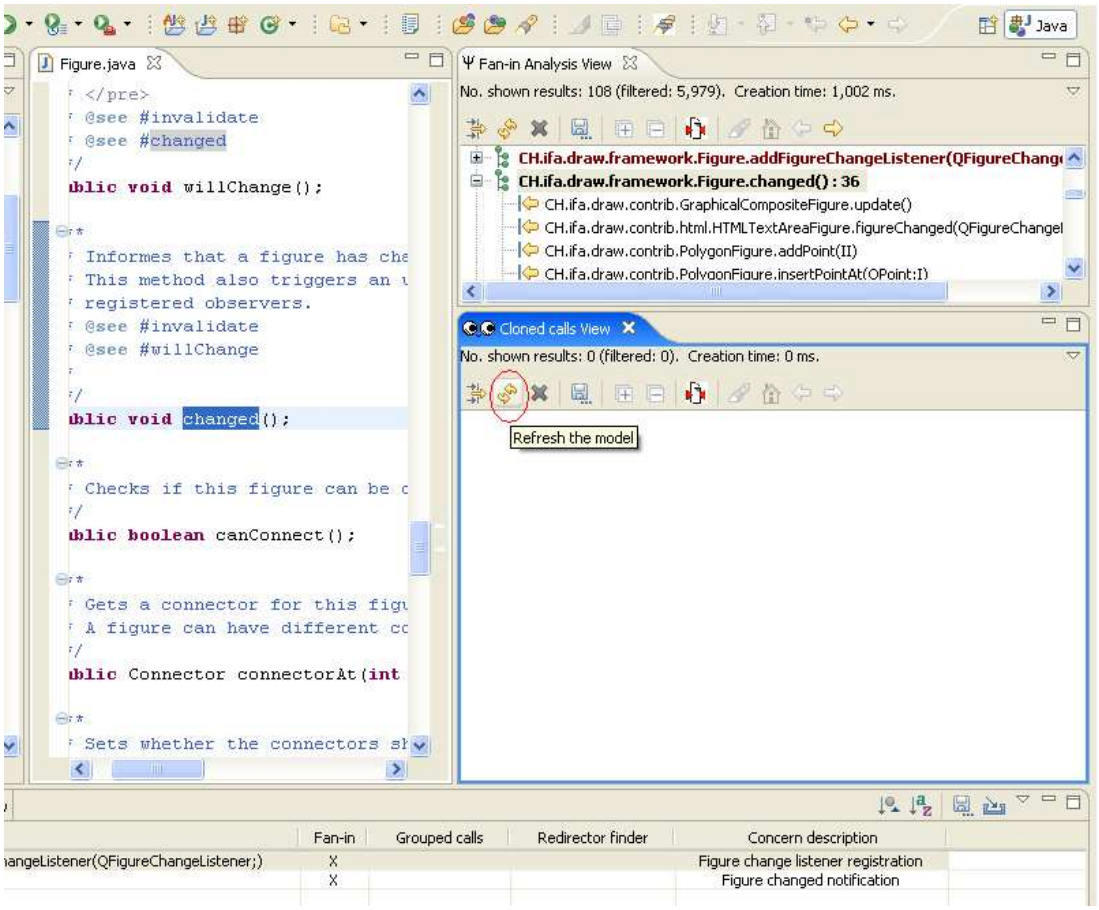

Figure 1.13: Running grouped calls analysis.

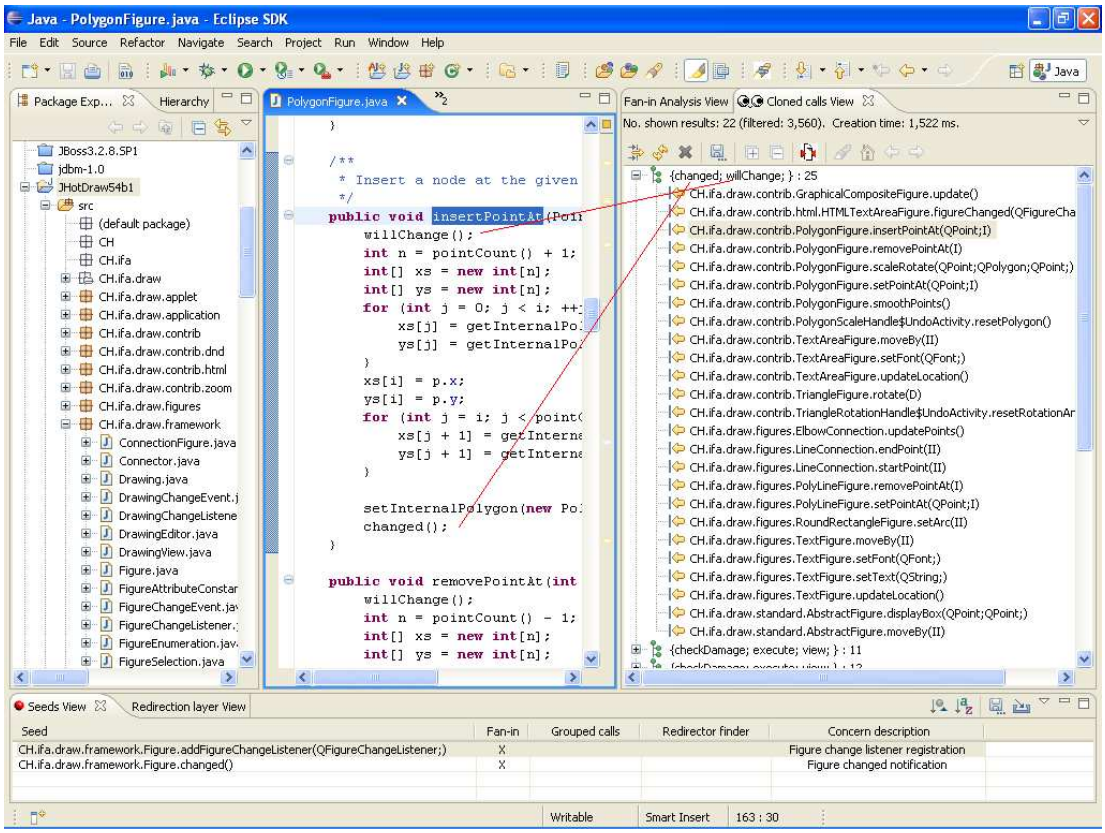

Figure 1.14: Grouped calls analysis results.

Besides the threshold for the minimum number of callers of a candidate, we can also set the minimum number of grouped methods that share their callers. All these filters are shown in Figure 1.15.

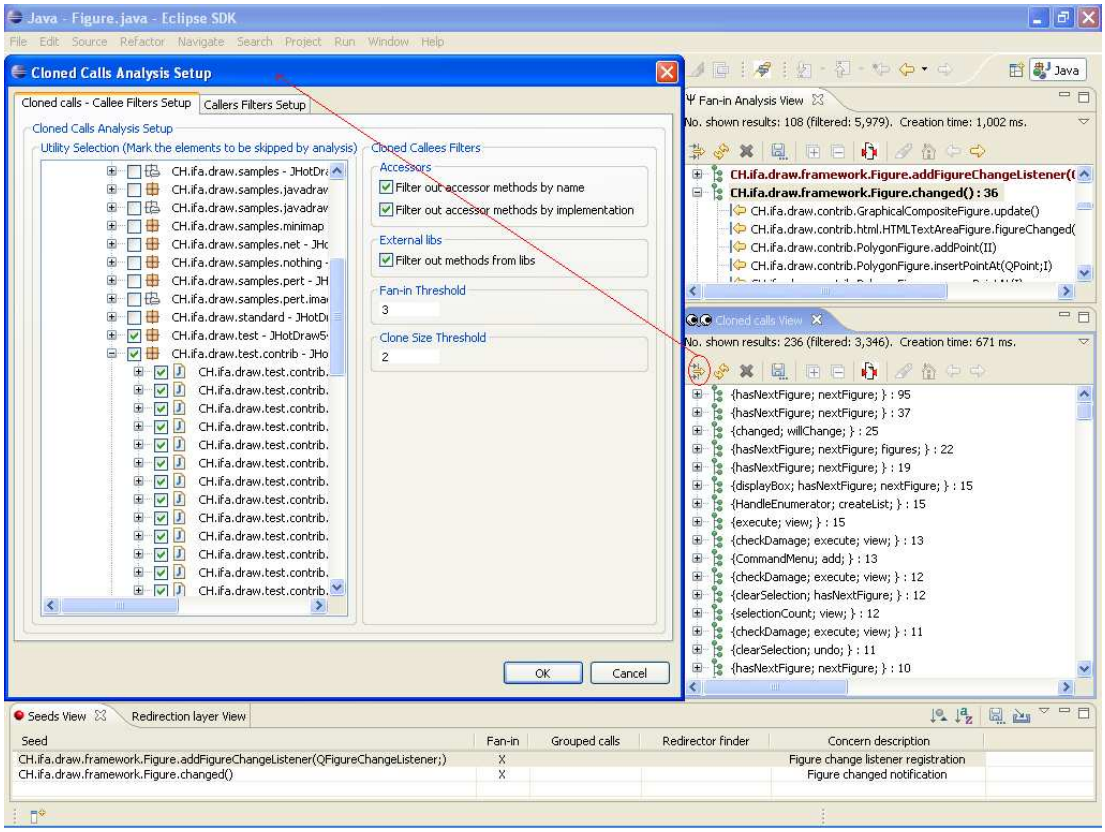

Figure 1.15: The filters for Grouped calls analysis.

The filters for the callers are again similar to Fan-in analysis and allow the user to ignore certain calls in the analysis, such as, for example, those from unit tests.

Marking a seed for this analysis proceeds again as described for the previous technique. Each of the methods grouped by this analysis is shown in the Seeds view, together with the other methods in the same group, as illustrated in Figure 1.16.

1.2.3 Redirections finder

Redirections finder requires too the model built by Fan-in analysis. To run the search for redirections in the code, the user needs to select the Refresh button in the Redirections finder view, which is marked with a circle in Figure 1.17.

The same figure shows the results of the analysis: the redirector class and the receiver of the redirection are shown as the root of the set of methods from each class related by an exclusive one-to-one relationship. Such a relantionship means that a redi-

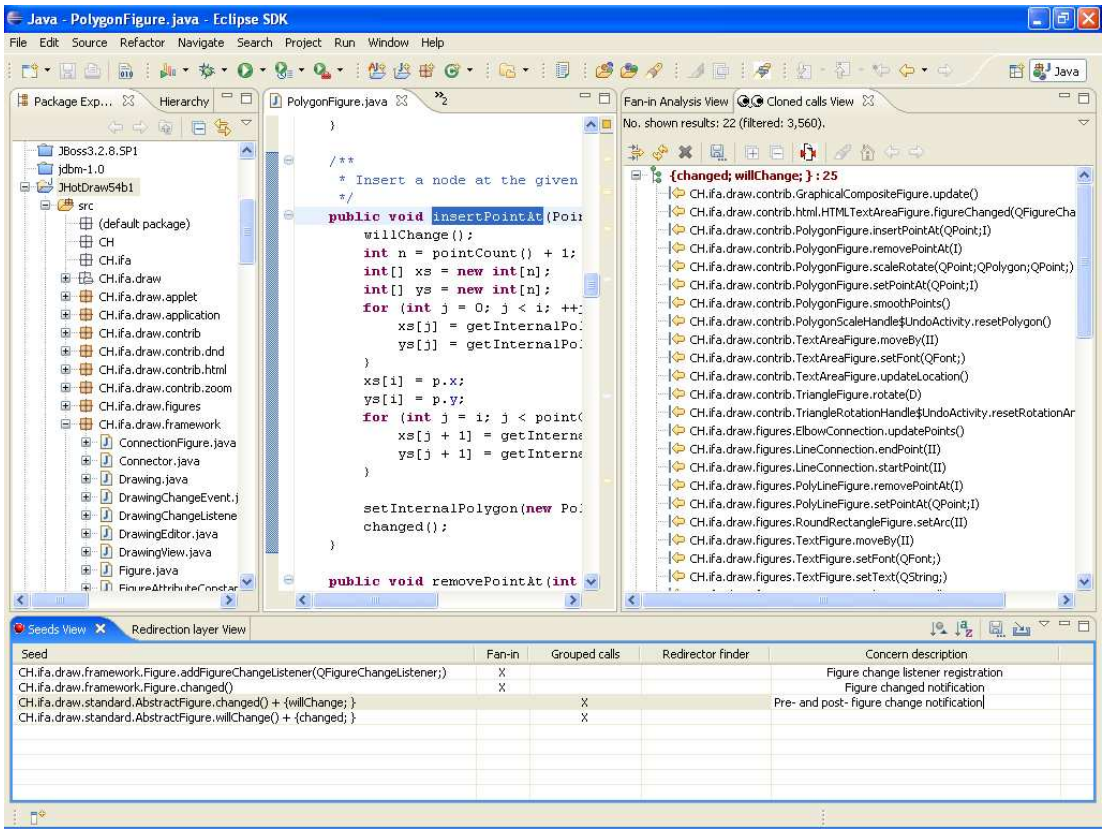

Figure 1.16: Seeds for Grouped calls analysis.

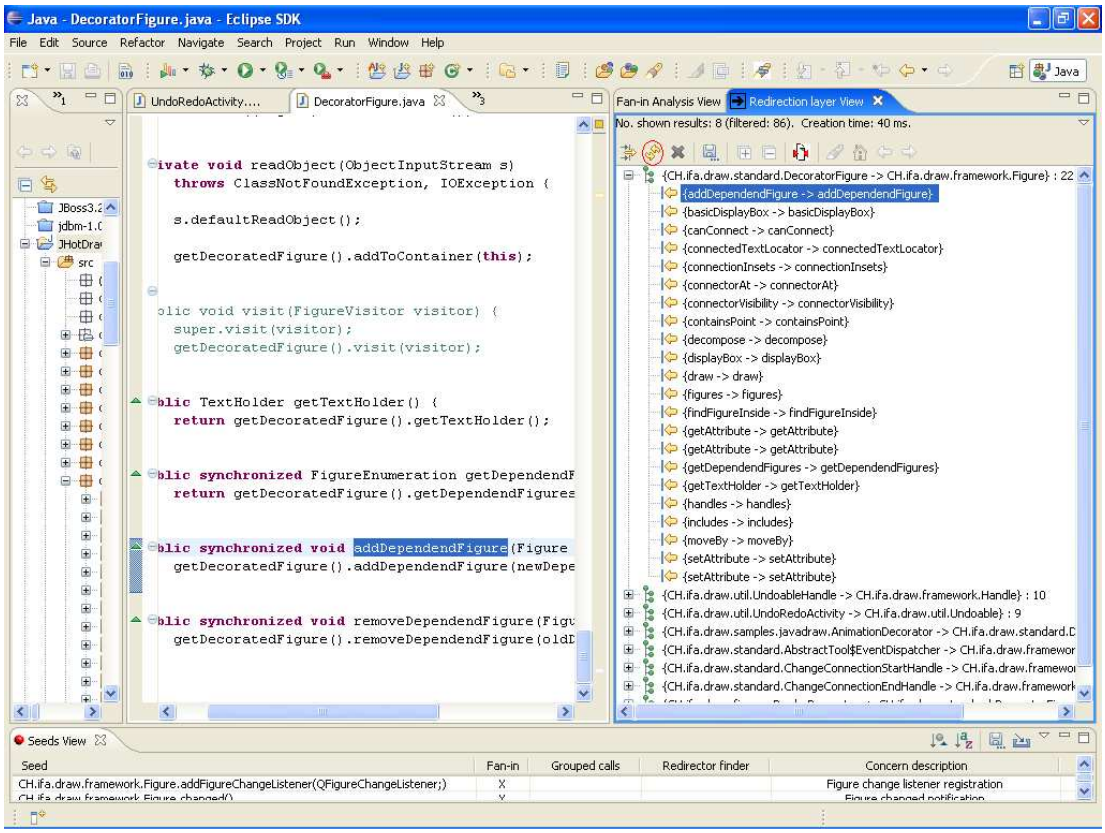

Figure 1.17: The view for the Redirections finder analysis.

rector method calls only one method in the class receiving the redirection, and that the receiver method is not called by any other method in the redirector class.

The filters that can be applied to this analysis are shown in Figure 1.18. In this dialog, we can select the minimum number of redirector methods in a class, according to the previosuly defined rule, as well as the minimum percentage of redirector-methods. In our example, a candidate-redirector class has to have at least 3 methods implementing a redirection and then these methods count for at least 50% of all the methods in that class.

The utility filter is based on the same considerations as the techniques described earlier.

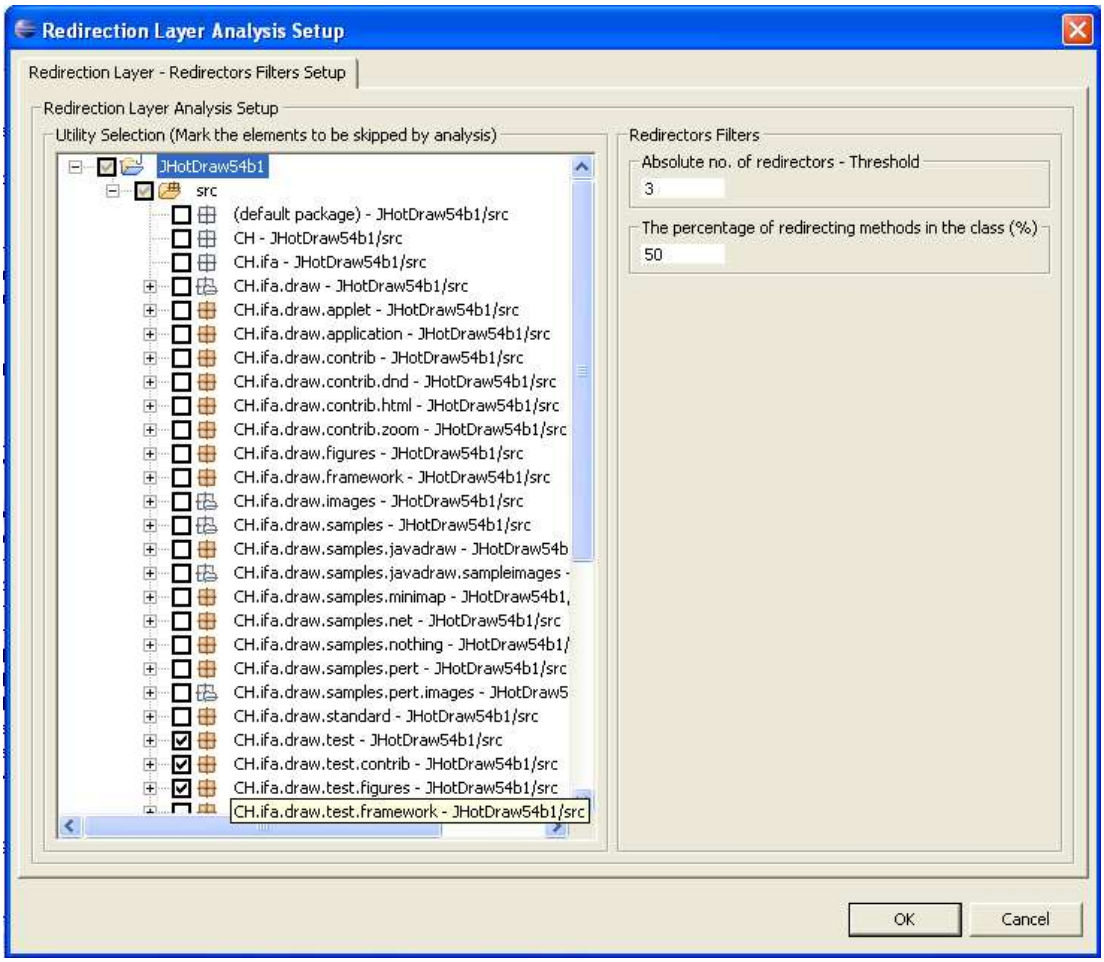

Figure 1.18: Redirections finder filters.

1.2.4 Combination of techniques

FINT also allows the user to combine techniques, namely those techniques that target concerns following similar implementation idioms. Two such techniques are Fan-in

and Grouped calls analysis, which both analyze method-call relations for crosscutting concerns. The combination consists of searching for the results of one technique among the results of the other one. As Grouped calls analysis makes a stricter selection of the methods with a large number of callers, we can select a lower fan-in threshold for this technique and then look for the grouped methods among the results of Fan-in analysis. For each of these methods, we might find a larger number of callers in Fan-in analysis and hence a better coverage of the concern of that method.

The combination can be launched as illustrated in Figure 1.19.

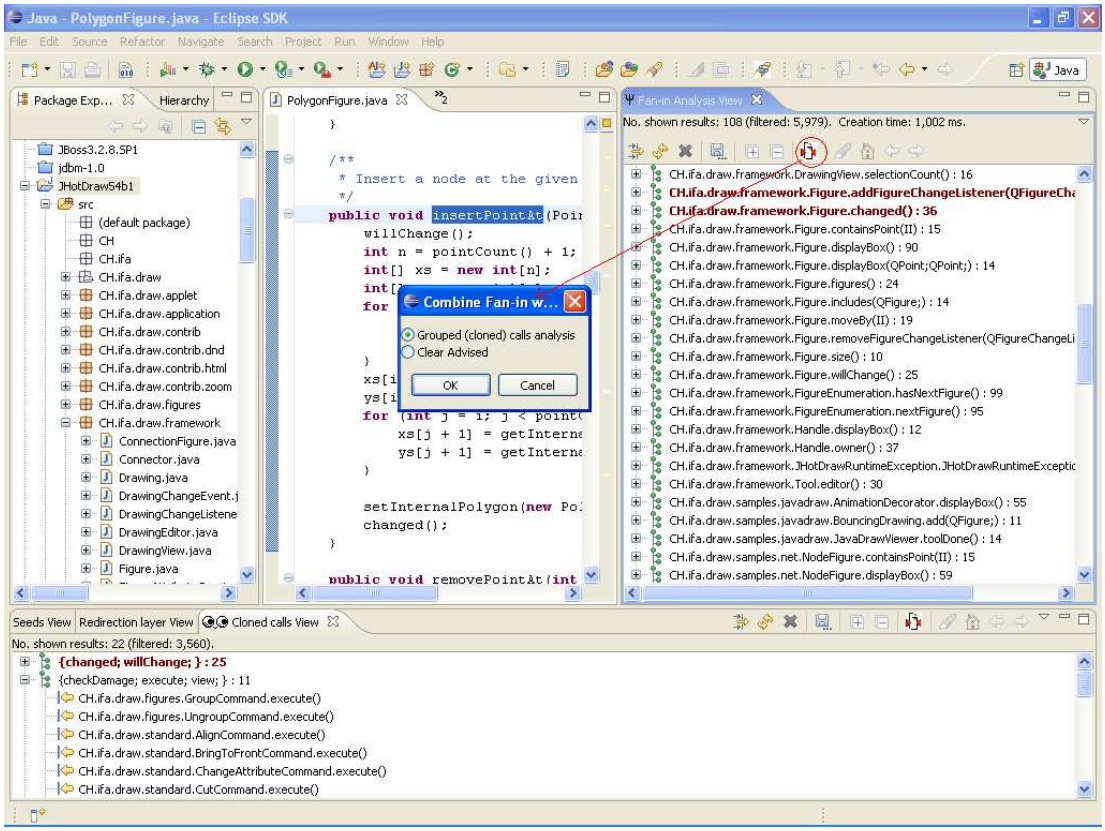

Figure 1.19: Combining Fan-in and Grouped calls analysis.

The intersection of results of the two techniques is highlighted in the view of Fan-in analysis, as shown in Figure 1.20.

1.2.5 Seeds management

All the seeds marked by the user are collected and displayed in the Seeds view next to the technique that identified them, as shown in Figure 1.21. The view also allows the user to add a short description of the concern implemented by each of the seeds, next to each seed. The list of seeds can be saved to or re-loaded from file.

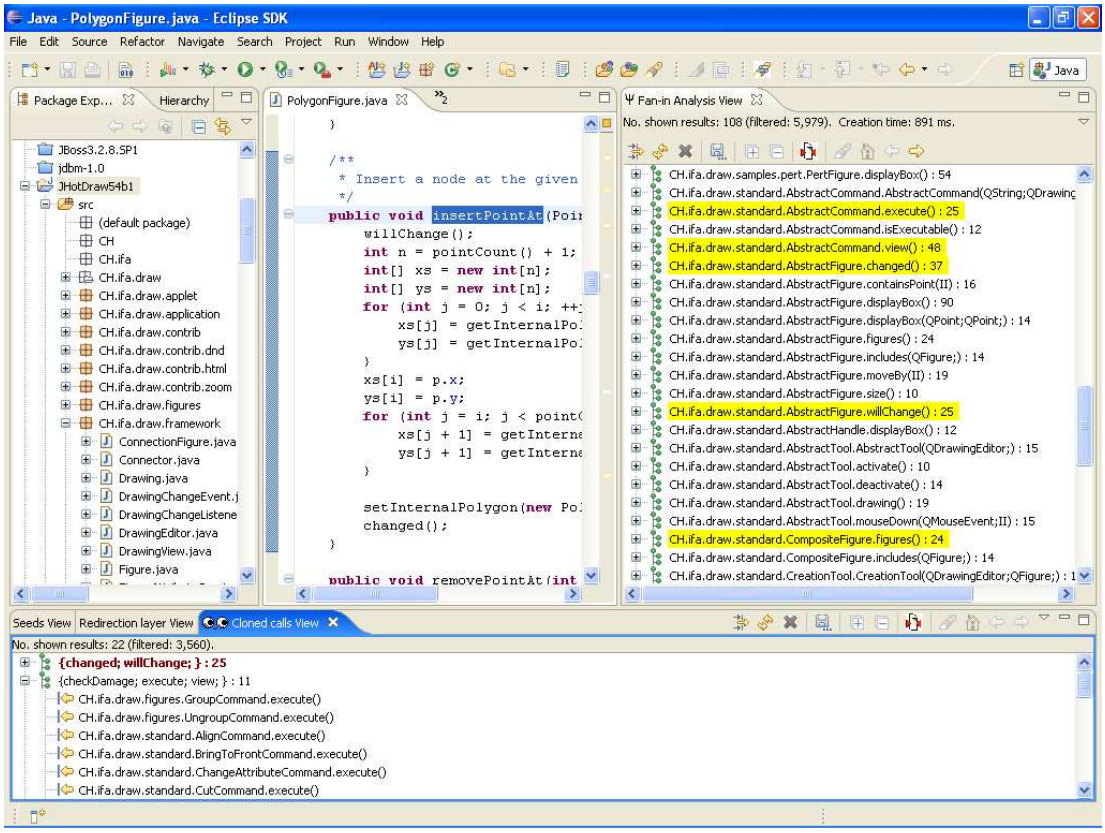

Figure 1.20: Combination results.

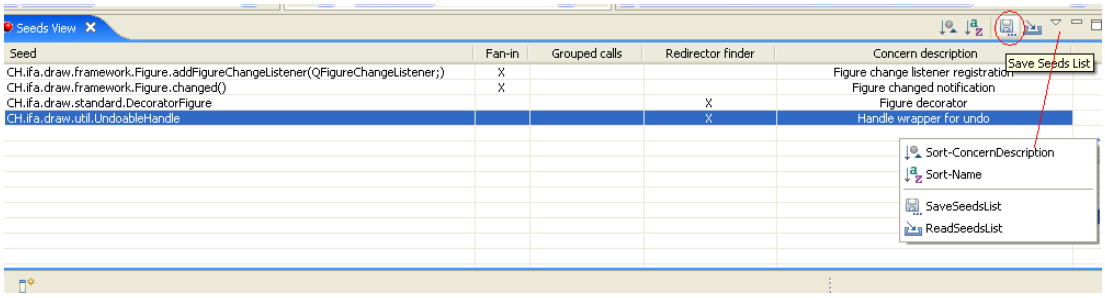

Figure 1.21: The Seeds view.

1.3 Contact

Questions, comments, bug reports, etc. are welcome at Marius Marin: a.m.marin@tudelft.nl

Bibliography