

[1 Introduction](#)

[2 Overview](#)

[3 Installation and Start of Program](#)

[4 User Interface and Functionality](#)

[5 Files](#)

[6 Tutorials](#)

[7 Appendix](#)

1 Introduction

SOTA is a tool for static program analysis and structure-oriented program testing (**Structure-Oriented Testing and Analysis**). In the course of a structure-oriented program test, the tool determines the source code coverage during the test, calculates the corresponding coverage metrics and provides a visual representation of the results. This allows the user to evaluate the program test with regards to source code coverage. Source code sections that have not been covered or conditions that have not been tested adequately can be identified easily. SOTA is not directly responsible for testing the program but serves as a tool for evaluating test cases and developing additional tests.

SOTA determines the coverage by source code instrumentation, ie the program to be tested must be available as compilable source code.

SOTA 1.0 only works on Java programs, but was developed to be able to support all major imperative and object-oriented programming languages. To use SOTA for other programming languages the user has to provide a parser as well as various classes for mapping the structure of the programming language to a more abstract structure. This is specified in detail in the developer documentation.

The program was developed as a standalone Eclipse-RCP-application and runs under Windows 2000 and upwards. For use in automatic testing systems, the non-GUI functionality of SOTA is provided by the library SOTA-ATM.jar which also serves as an API for integrating SOTA into other programs.

Chapter 2 covers the main features of the program and ways to use it. The operation of the program is then described in detail in Chapters 4-6, adopting, In Chapters 4 and 5, a systematic as well as, in Chapter 6, a process-oriented approach. The latter includes typical application scenarios in the form of tutorials. Lastly, all dimensions and other terms used in this manual are specified to provide a better understanding.

The program was developed for exclusive usage in teaching!

- [1 Introduction](#)
- [2 Overview](#)
- [3 Installation and Start of Program](#)
- [4 User Interface and Functionality](#)
- [5 Files](#)
- [6 Tutorials](#)
- [7 Appendix](#)

2 Overview

SOTA supports static program analysis and dynamic program testing.

During the static program analysis the source code of the program is analyzed and as a result SOTA determines ten software metrics like the Cyclomatic complexity or the number of modified boundary-interior paths. Before the analysis SOTA needs to parse the source code.

The dynamic program test (from now on referred to as program test) is a structure-oriented and control flow-related program test forming the main part of SOTA. Depending on the specific test case SOTA determines nine code coverage metrics during program execution, such as branch coverage or multiple-condition coverage and provides a graphical representation of the results. Before the analysis, it is necessary to instrumentate the program that will be tested.

There are three basic ways to employ SOTA in a program test: the *manual program test*, in combination with an *external testing system* or *integrated into an automatic testing system*.

During the *manual program test*, the program is tested manually, i.e. started by hand, functions are executed, etc. This can be done by using a development environment like Eclipse in addition to SOTA, or with SOTA itself by integrating an Ant build file and a startup script.

The only difference when working with an *external testing system* is the method of testing. As with the manual test, SOTA controls pre- and postprocessing. However, for the program test itself a separate testing system is used, such as ATOSj. This results in a program sequence SOTA, ATOSj, SOTA - without any internal linking.

When working with an *automatic test system* SOTA can be embedded as a library (SOTA-ATM) allowing the system to use the non-GUI functionality of SOTA. Core functions of SOTA can either be called through command line parameters or a SotaATM instance based on a class included in the library.

The work with SOTA is divided into the preparatory, testing and evaluation phases.

Basically, the *preparatory phase* for the program test consists of reading the source code, determining the type of instrumentation and instrumentating the source files. For a manual program test or the use of an external testing system this is done via the graphical user interface. When using SOTA-ATM with an automatic testing system the desired behaviour is evoked by calling the appropriate library routines or starting SOTA-ATM via command line parameters.

During the *testing phase* the compilation, start of program and program test are executed. Compiling the instrumented source files is outside the remit of SOTA. However, the compilation can be initiated by integrating a corresponding Ant build script. If a fitting batch file is available, it is even possible to start the program with SOTA. Given that these two files are included in SOTA the manual program test can be carried out without using an external development environment. During the program test, the instrumentations that were added to the source code produce a log file containing the data necessary for a complete reconstruction of the program.

Finally, during the *evaluation phase* the original source code is restored, the log files are read and the coverage metrics calculated. SOTA enables the user to evaluate the results visually and export them as an HTML report.

Program Test with SOTA

phases	tasks	exploitation method		
		manual program test	with external testing system (ATOSj)	integrated into automatic testing system

preparatory phase	reading and parsing source code	SOTA	SOTA	SOTA-ATM
	configuration of instrumentation	SOTA	SOTA	SOTA-ATM / configuration file
	instrumentation of source code	SOTA	SOTA	SOTA-ATM
testing phase	compilation of source code	external / with SOTA using a script	external	external
	start of program	external / with SOTA using script	external	external
	program test	manual	external	external
evaluation phase	restoration of original source code	SOTA	SOTA	SOTA-ATM
	reading of log files and calculation of coverage metrics	SOTA	SOTA	SOTA-ATM
	visualization of results	SOTA	SOTA	none / SOTA (after the tests)
	export of report	SOTA	SOTA	SOTA-ATM

[1 Introduction](#)

[2 Overview](#)

[3 Installation and Start of Program](#)

[4 User Interface and Functionality](#)

[5 Files](#)

[6 Tutorials](#)

[7 Appendix](#)

3 Installation und Start of Program

SOTA was developed as a standalone Eclipse-RCP-application. The only requirement a system must meet to run SOTA is an installation of Java 6.0 or higher. A simple calling of SOTA.exe starts the program. Starting SOTA for the first time configures the Eclipse-Rich-Client-Platform.

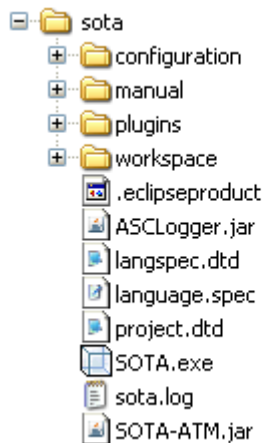


fig.: SOTA directory structure

Depending on the exploitation method it is necessary to install other systems, e.g. Eclipse or ATOSj.

[1 Introduction](#)

[2 Overview](#)

[3 Installation and Start of Program](#)

[4 User Interface and Functionality](#)

[4.1 Menus und Toolbar](#)

[4.1.1 Menu *Project*](#)

[4.1.2 Menu *Tasks*](#)

[4.1.3 Menu *Configuration*](#)

[4.1.4 Menu *Help*](#)

[4.2 Views](#)

[4.2.1 View *Project*](#)

[4.2.2 View *Testlogs*](#)

[4.2.3 View *IScheme*](#)

[4.2.4 View *Source*](#)

[4.2.5 View *CFG*](#)

[4.2.6 View *Coverage*](#)

[4.2.7 View *Metrics*](#)

[4.3 Preferences](#)

[4.3.1 Preferences View *CFG*](#)

[4.3.2 Preferences View *Coverage*](#)

[4.3.1 Preferences *General*](#)

[4.3.1 Preferences *Report*](#)

[4.3.1 Preferences View *Source*](#)

[4.4 Delete Project](#)

[5 Files](#)

[6 Tutorials](#)

[7 Appendix](#)

4 User Interface and Functionality

Next to the compulsory menu bar and the toolbar offering quick access to the most common actions the user interface consists of different views available in three areas by clicking the corresponding tabs. The upper left area offers the view with the project outline. Arranged below the project outline are two views listing the test cases and the instrumentation schemes of the project respectively. The right area takes up the largest part of the window, consisting of different views supplying detailed information about the source code, control flow graphs as well as coverage and metrics reports.

The views are linked so that e.g. selecting a file in the ProjectView causes the corresponding source code to appear in the SourceView. Selecting a test in the TestView updates the coverage metrics for the entire project and the representation of the coverage for the selected file. Wizards and dialogs are at the user's disposal in order to guide larger actions such as creating a new project.

Lastly, a status bar can be found at the bottom of the window, providing information about the status of the original source files as well as the parsed source code SOTA is displaying. The original state is designated as CLEAN whereas the instrumented state is marked as DIRTY.

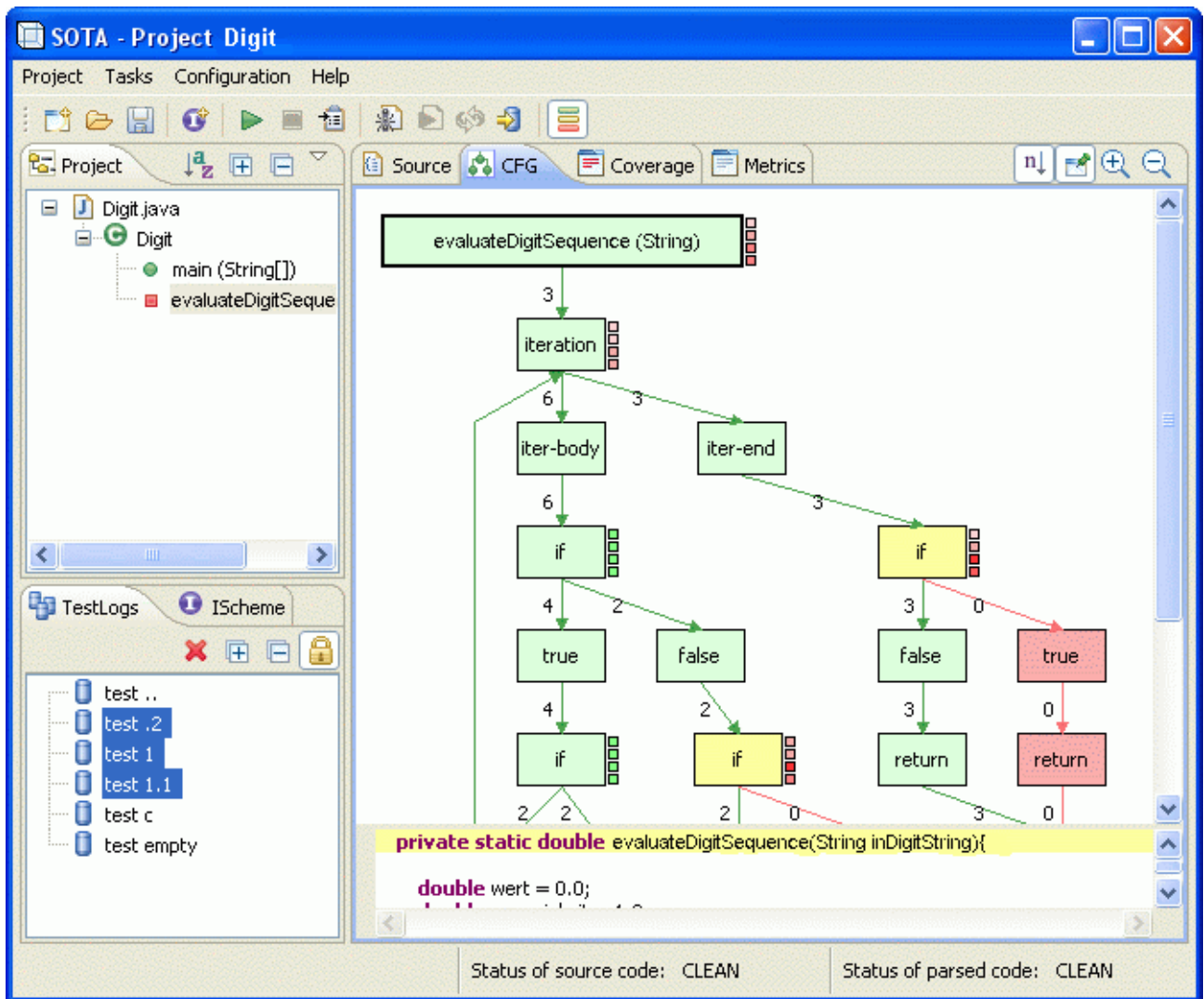


Fig.: main window of SOTA

4.1 Menus und Toolbar

4.1.1 Menu *Project*

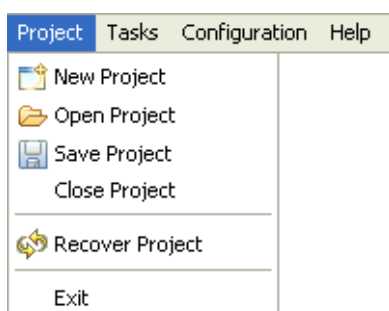


Fig.: menu *Project*

New Project



Clicking the menu item *New Project* opens the two-page wizard for creating a project. On the first page the user has to indicate a name for the project, as well as the project directory and run directory. From the project name SOTA generates the file <project name>.project placing it in the SOTA directory.

The project directory is the root directory of the test program (corresponds to ".. / workspace / project-name" in Eclipse projects). This directory contains the imported source code as well as the

reports created by SOTA at the end of a test. In the run directory the test program is started. In most cases, the run directory corresponds to the home directory of the test program, except for Eclipse-RCP-projects where it is the Eclipse base directory ".. \ eclipse \". In the run directory the ASCLogger.ini is created which is read by the ASCLogger during the test. After the test the testlogs are read from the run directory as well

Choosing a programming language in the wizard is necessary, even though the current version of SOTA only supports JAVA. Finally, the user can include an Ant build file for compiling and a batch file for

creating the test project.

On the second page the user has to include the source files to be considered for the test. They can either select a directory thus including all source files in all sub directories, or simply individual files.

After finishing the wizard, SOTA creates the project and loads all the selected source files.

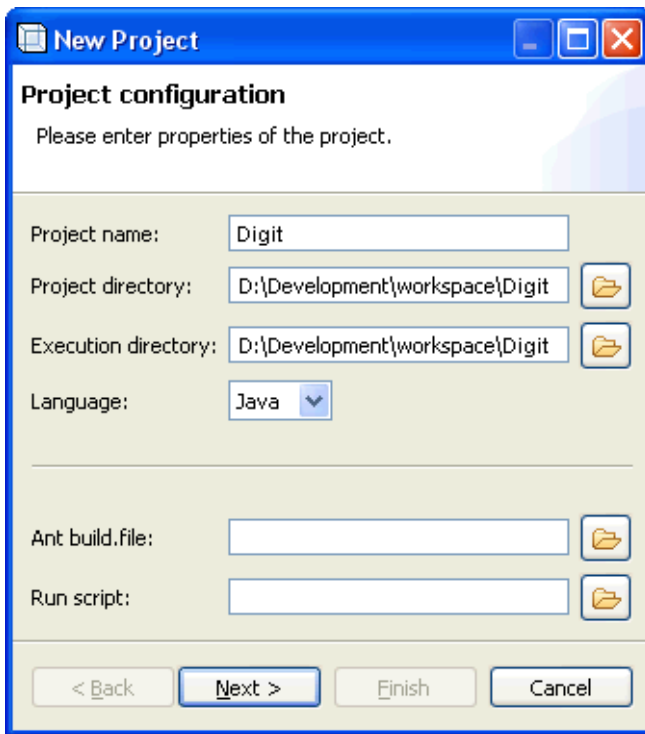


Fig.: first page of wizard

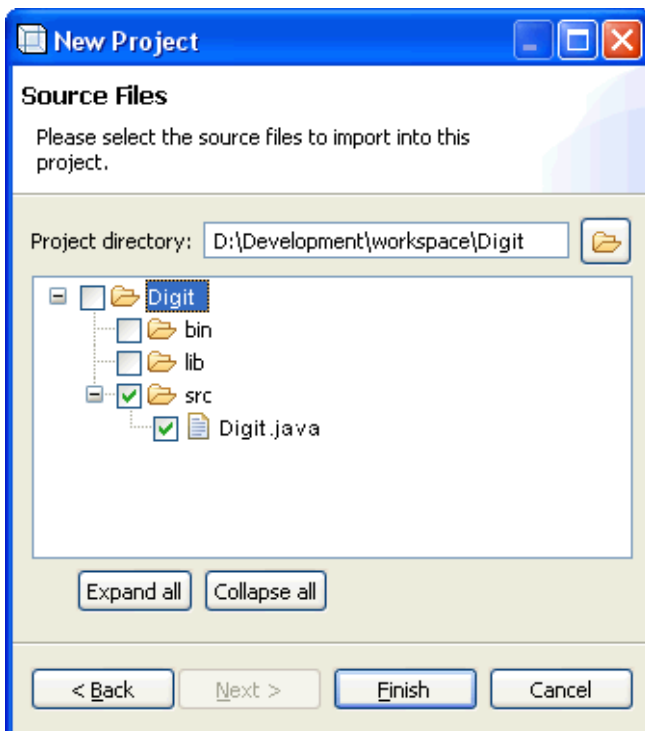


Fig.: second page of wizard

Open Project



Clicking the *Open Project* menu item opens a project created previously. A standard dialog for loading a file from the root directory of SOTA appears, where the desired project can be selected. After confirming the selection by clicking *OK*, SOTA opens the project and loads all the associated source files.

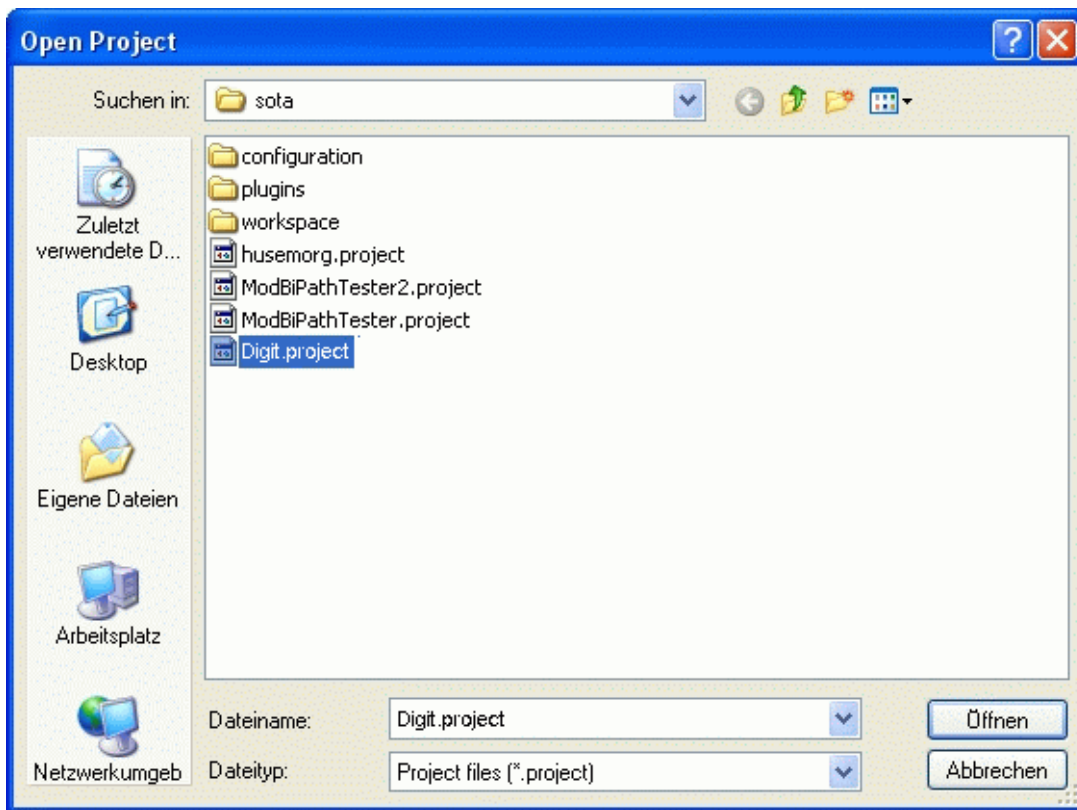


Fig.: dialog *Open Project*

Save Project



Saving the project by clicking *Save Project* creates, if done for the first time, a project file in the root directory of SOTA. The name of the file is identical with the project and ends on 'project'. This file contains the project data and all instrumentation schemes. If the file already exists, it will be overwritten with the current data.

Close Project

Clicking the menu item *Close Project* closes the current project. Consequently, SOTA is in its starting state again.

Recover Project

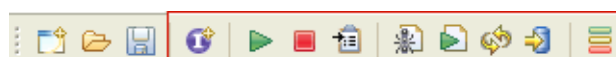


The menu item *Recover Project's* task is to restore corrupted projects. Analogous to the menu item *Open Project*, a dialog for selecting a project appears. As a result, all source files of the chosen project are restored and then opened.

Exit

Clicking the menu item *Exit* closes SOTA.

4.1.2 Menu Tasks



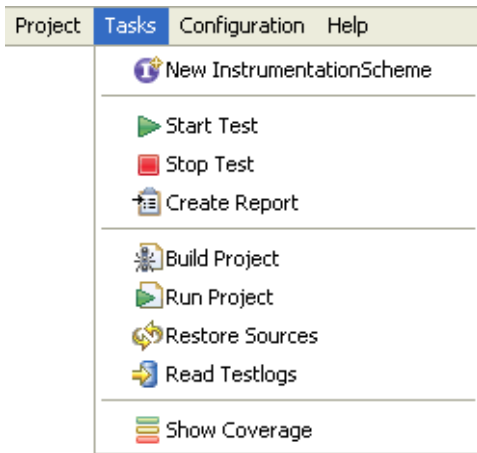


Fig.: menu *Tasks*





New InstrumentationScheme



Upon clicking the menu item *New Instrumentation Scheme* a dialog allowing the creation of a new instrumentation scheme (short: IScheme) for the current project appears. An IScheme represents an instrumentation pattern for all source files of the project. It can adopt different degrees. Depending on the test case, the memory requirements of the log files can be very high. The purpose of this configuration option is to allow the user to set a reasonable limit for the size of those files.

For creating an IScheme a name for management of the scheme inside the project, must be specified in the dialog. Entering a description is optional. The field below the

description field represents the hierarchical structure of the current project in form of a tree. It shows all files, their classes and the individual functions. A level of instrumentation from 0 to 3 can be assigned to each of these structures by clicking the matching button for the level after choosing a certain structure. Consequently, each structure is colored according to their respective levels. Furthermore, the substructures associated with them receive the same level of instrumentation. The individual levels have the following effects:

-  Level 0 - no instrumentation of the source code,
-  Level 1 - only instruments blocks necessary to determining the control-flow relevant coverage metrics (FEEC, C0, C1, MBI, BI),
-  Level 2 - additionally instruments all blocks enabling condition coverage analysis (C2, MMCC, MCDC, C3),
-  Level 3 - instruments the complete source code, i.e. evaluates all instructions and conditions.

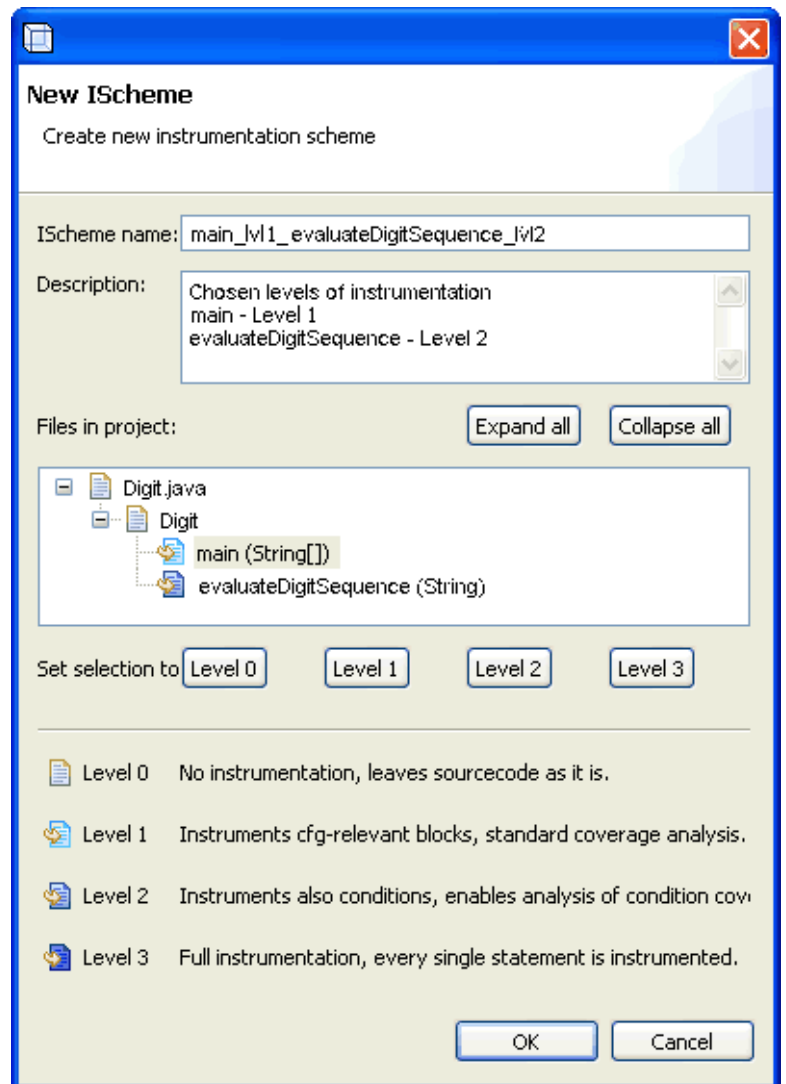


Fig.: dialog *New IScheme*

For the program test, instrumentation level 2 is recommended as it allows the identification of all coverage metrics. If necessary, a lower level can be used in order to save capacity. Using level 3 is suggested when programs or functions are not terminating properly and the user wishes to identify the exact point of abortion or where the exception is called.

After confirming the dialog, the instrumentation scheme as well as all selected settings are stored in the

project-specific .ischeme-file which is from then on available for the program test.

Start Test

▶ / Restart Test

By clicking the menu item *Start Test* the program test is initiated in SOTA. A dialog box for configuring the test appears. The user has to enter a name for the test which also serves as a name for the test log SOTA creates at the end of a test, i.e. limitations for file names set by the respective operating system must be considered. Adding a description of the test is also possible but not mandatory.

The instrumentation of the project has to be configured next. In addition to the user-generated ISchemes, the 3 basic ISchemes which instrument to whole project according to Level 1, Level 2 and Level 3 are also available. The hierarchical project list shows the specific instrumentation of the individual project structures for a selected IScheme. The list can be fully expanded or collapsed by clicking the buttons *Expand All* or *Collapse All* respectively.

Once a name for the test and an IScheme have been selected, the program test can be started by confirming the dialog. SOTA stores the information relevant for the logging

component in the file 'ASCLogger.ini' by adjusting the execution path of the project. Subsequently, a backup of all original source files is produced by changing the file extensions to '.backup' and the source code is instrumented. Then the project is parsed again. Finally, the user can view the instrumented source code in the SourceView. Now the instrumented version of the source code can be compiled and e.g. tested systematically with an external testing system.

Three more options can be selected by ticking the check boxes at the bottom of the dialog. *Rerun configuration* merely leads to a change of the name of the test and its description in the ASCLogger.ini leaving the source code unchanged. This enables re-testing the code with the same configurations but without having to instrument and compile it again. *Build Project* causes SOTA to compile the source code after the instrumentation. For that purpose it is necessary to embed an Ant version by providing the path to the file 'ant.bat' under the point *General* of the preferences. Furthermore, the user has to include an appropriate xml-file enabling SOTA to compile using Ant. This option is only available if a script exists and the option *Rerun configuration* has not been selected. Finally, the option *Run project* actually causes

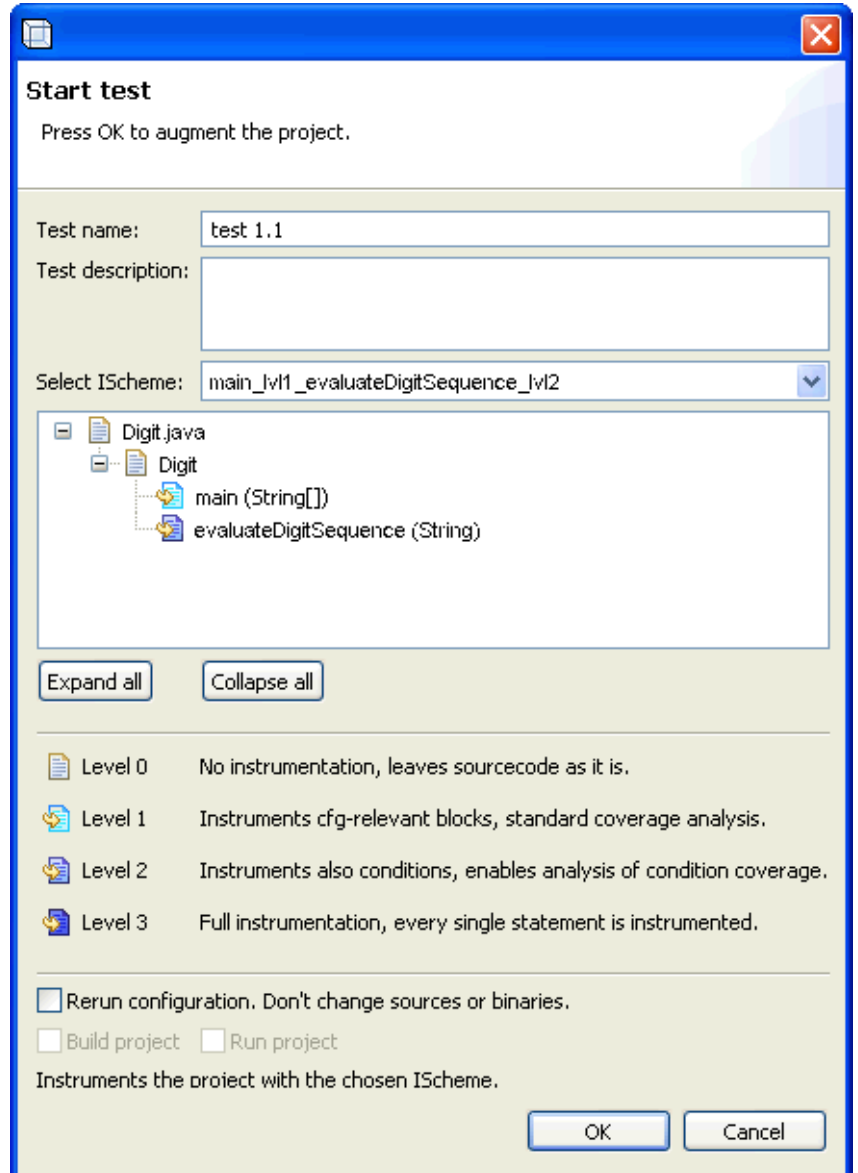


Fig.: dialog *Start Test*

the project to start after compilation as long as a startup script has been specified. This will be possible, only if one of the other two options are selected.

If a test is running already or SOTA finds the source code instrumented during parsing, the option



Start Test will be exchanged with



Restart Test. It is not possible to re-instrumentate the sources without having stopped the current test and restored the original files. However, The instrumented test program may be started again with a different name for the test. This only updates the initialization file of the logging component with the new name for the test, but no source files or binaries are changed. Restarting the test is equivalent to the *rerun* option of the normal *Start-Test-dialog*.

Stop Test



The *Stop Test* menu item terminates the current test run. Then the original source files are restored and re-read. If a corresponding Ant buildfile is available, the original sources will also be compiled again.

Finally, a dialog appears allowing the user to select log files from the run directory which SOTA will then read, analyze and list in the view *TestLogs*. Subsequently, the coverage metrics for individual test logs or combinations thereof can be computed and displayed.

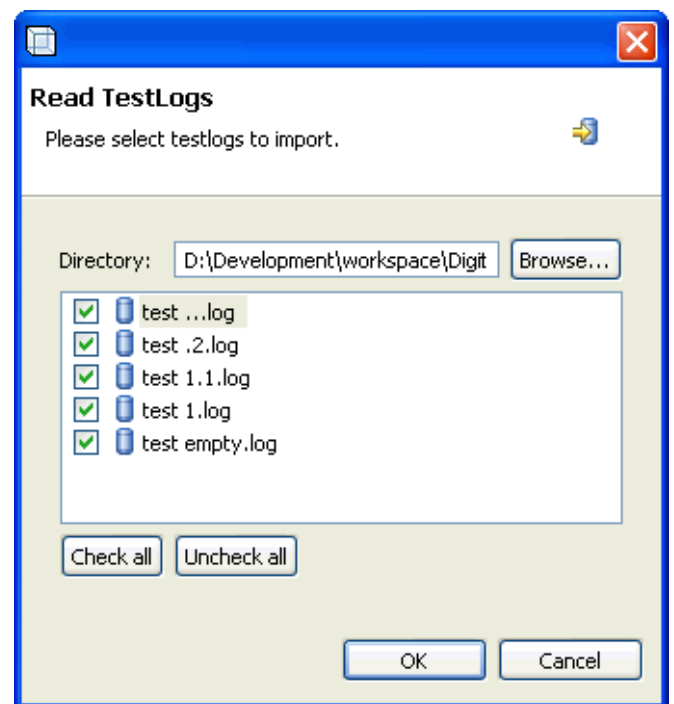


Fig.: dialog *Read Logs*

Create Report



By clicking the menu item *Create Report* it is possible to create a coverage report based on the selected test logs for the current project. The user can adjust the

preferences so that for each report a prompt appears for setting the file name and path. Otherwise, the report is stored in the root

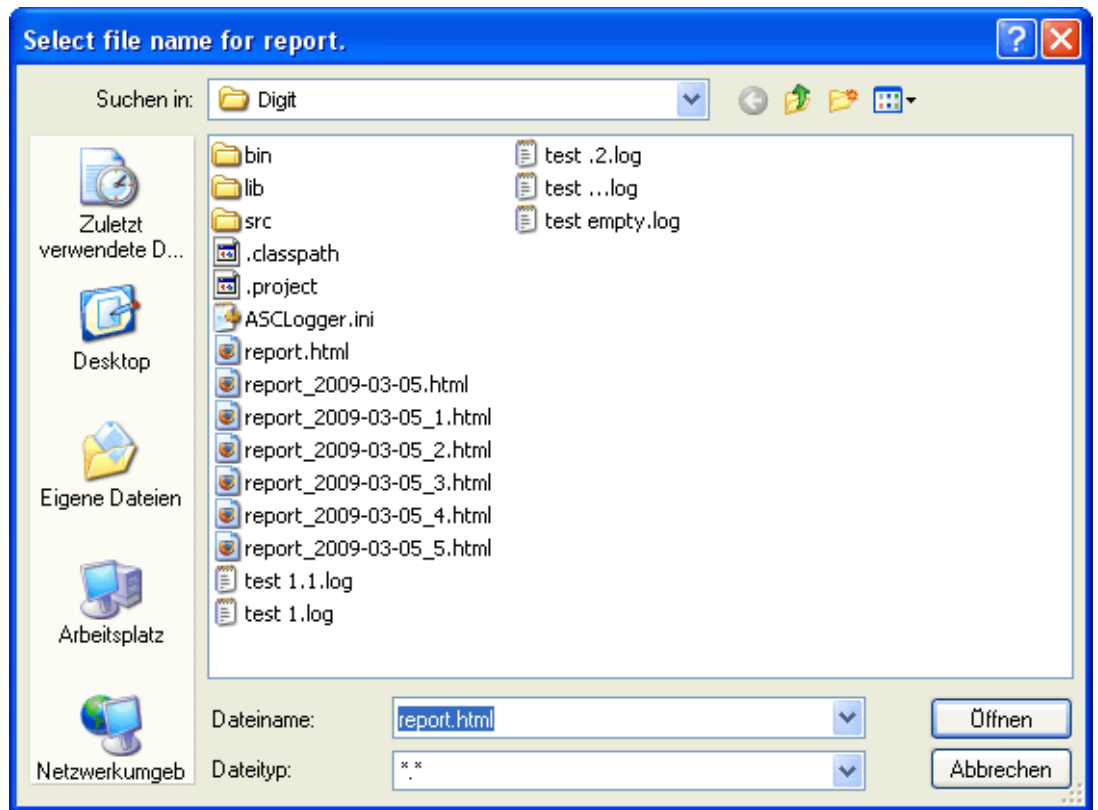


Fig.: dialog *Create Report*

directory of the project under the name 'report.html'. In the preferences there is also a setting for preventing SOTA from overwriting report files. In that case a new file is generated for each report according to the format 'report_<date>_<index>.html'. These and other settings for creating reports can be made in the preferences menu under the item *Report*

Build Project



The menu item *Build Project* enables compiling the project in SOTA on the basis of the current source files regardless of the test status. This menu item will only be activated, if an xml buildscript is specified for the current project and an Ant version is included in the preferences.

Run Project



The menu item *Run Project* will be available, if a startup script is specified for the current project. After clicking this item, the project will be executed.

Restore Sources



In order to translate the project into its original state, it is also possible to select the menu item *Restore Sources*. Unlike the *Stop Test* menu item the source files are only restored and read. They are not be re-compiled (assuming a buildfile exists), nor does the dialog for reading the test logs appear.

Read Logs



With the menu item *Read Logs* the log files can be read manually regardless of the test status of the project. For that purpose, the same dialog as for the *Stop Test* menu item appears allowing the import of log files. After selecting the log files from the run directory of the project, SOTA reads the files and processes the test results.

Show Coverage



Activating the menu item *Show Coverage* leads to a colored display of the coverage information in the source code and coverage views.

4.1.3 Menu Configuration

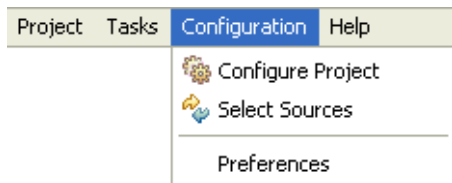


Fig.: menu *Configuration*

project. Confirming these values by selecting *Ok* overwrites the settings made when the project was created.

Select Sources



In order to modify the source files included in the project, the user has to select the menu item *Update Sources*.

Subsequently, a dialog identical to the second page of the wizard for creating a project appears. The source files can be selected analogically. After confirming the selection, the current source files are replaced by the new ones.

Configure Project



Project-specific settings can be made via the menu item *Configure Project*. In detail, these are the two paths of the test project relevant to SOTA, namely the root directory and the execution directory, and, underneath them, the optional scripts imported for the purpose of compilation and start of the

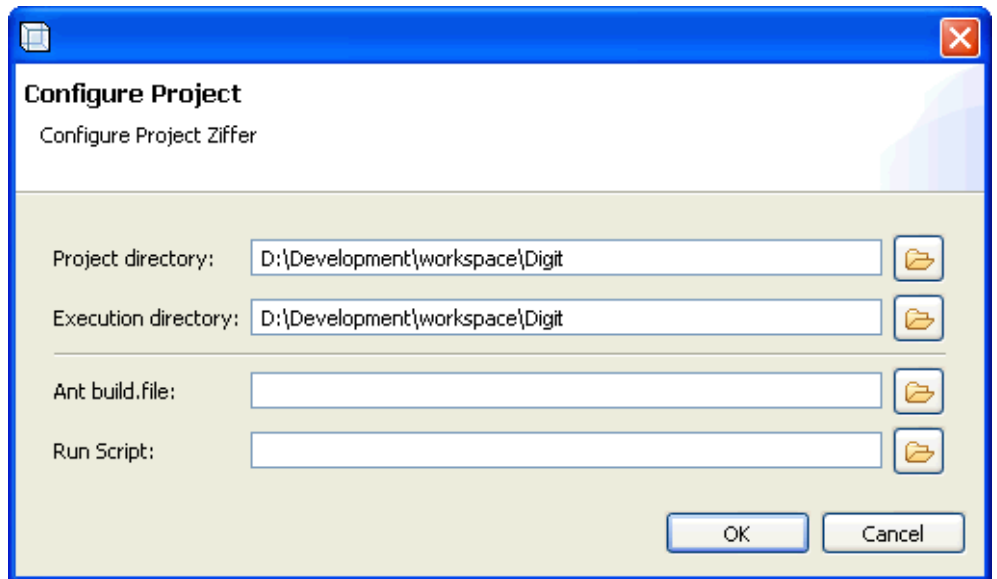


Fig.: dialog *Configure Project*

Preferences

Under the menu item *Preferences* the user can make changes to the general settings of SOTA that apply to all projects and will also be saved when exiting the program. A comprehensive explanation of the various configuration options can be found under [4.3 Preferences](#).

4.1.4 Menu Help

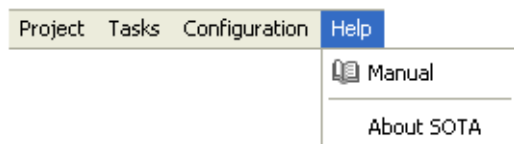


Fig.: menu *Help*

Manual



Selecting the menu item *Manual* opens the user documentation on hand in the system's standard browser

About

The menu item *About* opens a dialog containing information about the current SOTA version as well as the plug-in status of the application.

4.2 Views

4.2.1 View Project

In the view *Project* all source code files of the test program and their subordinate structures, as well as classes and methods are listed in a tree structure. The top nodes represent the source code files imported on creating the project. The top-level classes and methods and their respective internal classes and methods are represented as children in hierarchical order up to an arbitrary nesting level. Using the two buttons



and



the user can fully expand or collapse the tree. The button



alternately causes the tree to be displayed in ascending and descending order. On selecting an element of the tree the views *Source* and *CFG* are updated automatically.

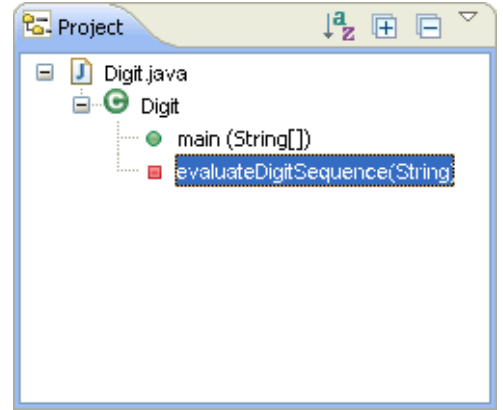








Fig.: view *Project*
(hierarchical presentation)

The individual structures listed in the tree are:

-  files
-  classes
-  internal classes
-  public functions
-  protected functions
-  private functions

If instructions for instrumentation are detected on parsing the file, e.g. during a test run on instrumented sources, the Icon will be displayed with a red exclamation mark (e.g.:



).

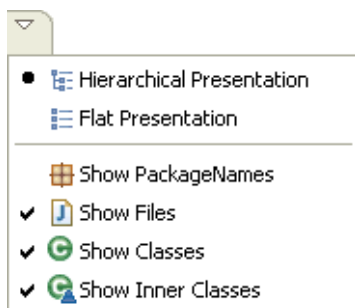


Fig.: view menu

Via the menu selectable on the top right corner of the view, the presentation of the project structures can be configured. The two presentation options



Flat

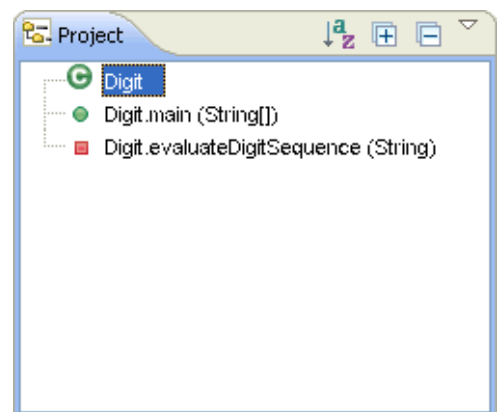






Fig.: view *Project*
(flat presentation without files)

Presentation and



Hierarchical Presentation are available. The hierarchical presentation matches the listing in a tree structure as described above which is the default setting. In the flat presentation all structure units are at root level, i.e. file, classes and functions are listed coequally.

The additional options allow the following adjustments of the presentation:

-  *Show PackageNames* - adds the packet name to the name of the structure (default: off)
-  *Show Files* - additionally lists files in the view (default: on)
-  *Show Classes* - lists classes in the view. This only affects the flat presentation, whereas the hierarchical presentation always lists classes. (default: on)
-  *Show Inner Classes* - lists inner classes in the view (default: on).




4.2.2 View TestLogs

If log files are imported into the project, they will be listed in the view *TestLogs*. In case a log file already exists, the new log data are appended to its end, so that a test log can contain a variety of test case data. As a default, the individual test cases are hidden from the user and only the test logs are listed. In order to examine the content of the test logs in detail, the user has to unlock the button



Lock TestLogs on the toolbar of the view, which lists all test cases as well as ISchemes of every test log right beneath each one.

Since not every log file may have been created by SOTA and not all the test data belongs to the current project, the view TESTLOG uses the following icons to represent compatibility:

-  - fully compatible test log and test case data
-  - partially compatible test log, also contains invalid test case data
-  - invalid log file or incompatible test case data

If valid test logs and test cases are selected, the coverage information in the views *Source*, *CFG* and *Coverage* will be updated automatically. Clicking the buttons



and



changes the entire selection. To select several test logs at once, the keys <Shift> or <Ctrl> respectively have to be held down while selecting the desired test logs.

Selected test logs can be deleted with the



Delete TestLogs button. That not only removes them from the project but from the system. This option is not available for test case data, i.e. individual parts of a test log.

On double-clicking onto one

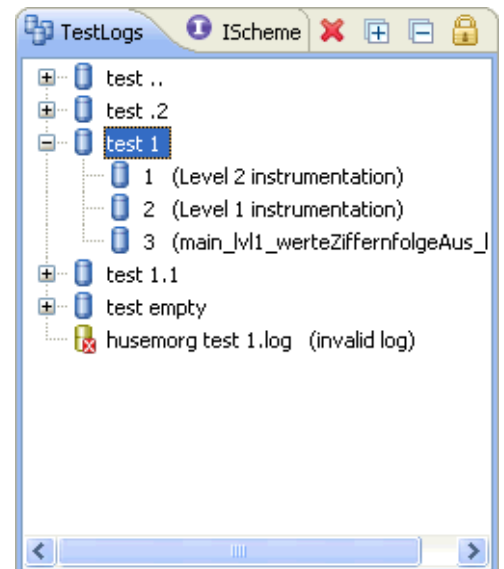


Abb.: View *TestLogs*
(unlocked)

of the listed test logs or test case data a dialog appears containing detailed information on the test data. Listed first are the name and description of the test as well as the name of the instrumentation scheme. Next, the number of paths (i.e. function cycles) contained in total in the test case and a list of all functions the test hit are indicated. For each function the number of associated paths is also listed. By clicking the column headers the presentation of the list can be sorted in ascending or descending alphabetical order by function name and number of paths.

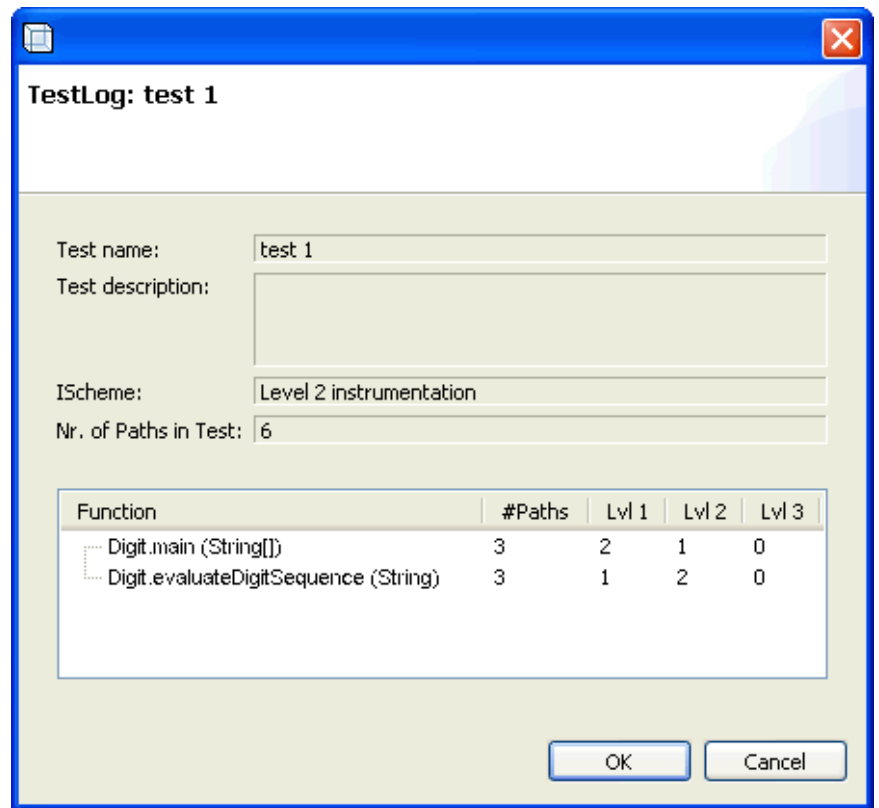


Fig.: dialog *TestLogs*

4.2.3 View *IScheme*

The view *ISchemes* contains only the instrumentation schemes (short: *ISchemes*) related to the project. For every project three *ISchemes* are created automatically, instrumenting the entire project into the levels 1, 2 and 3 respectively. The user can create additional *ISchemes* which will then appear in this view via the menu item *New IScheme*.

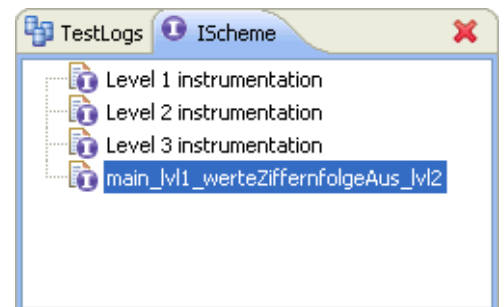


Fig.: view *ISchemes*

Double-clicking on an *IScheme* opens a dialog listing, analogous to the menu item *New IScheme*, stored instrumentation settings of the *IScheme* and enabling the user to change all information. The button



Delete IScheme removes an *IScheme* from the project.

The standard *ISchemes* across levels 1,2 and 3 are exempt from these changes and cannot be deleted.

4.2.4 View *Source*

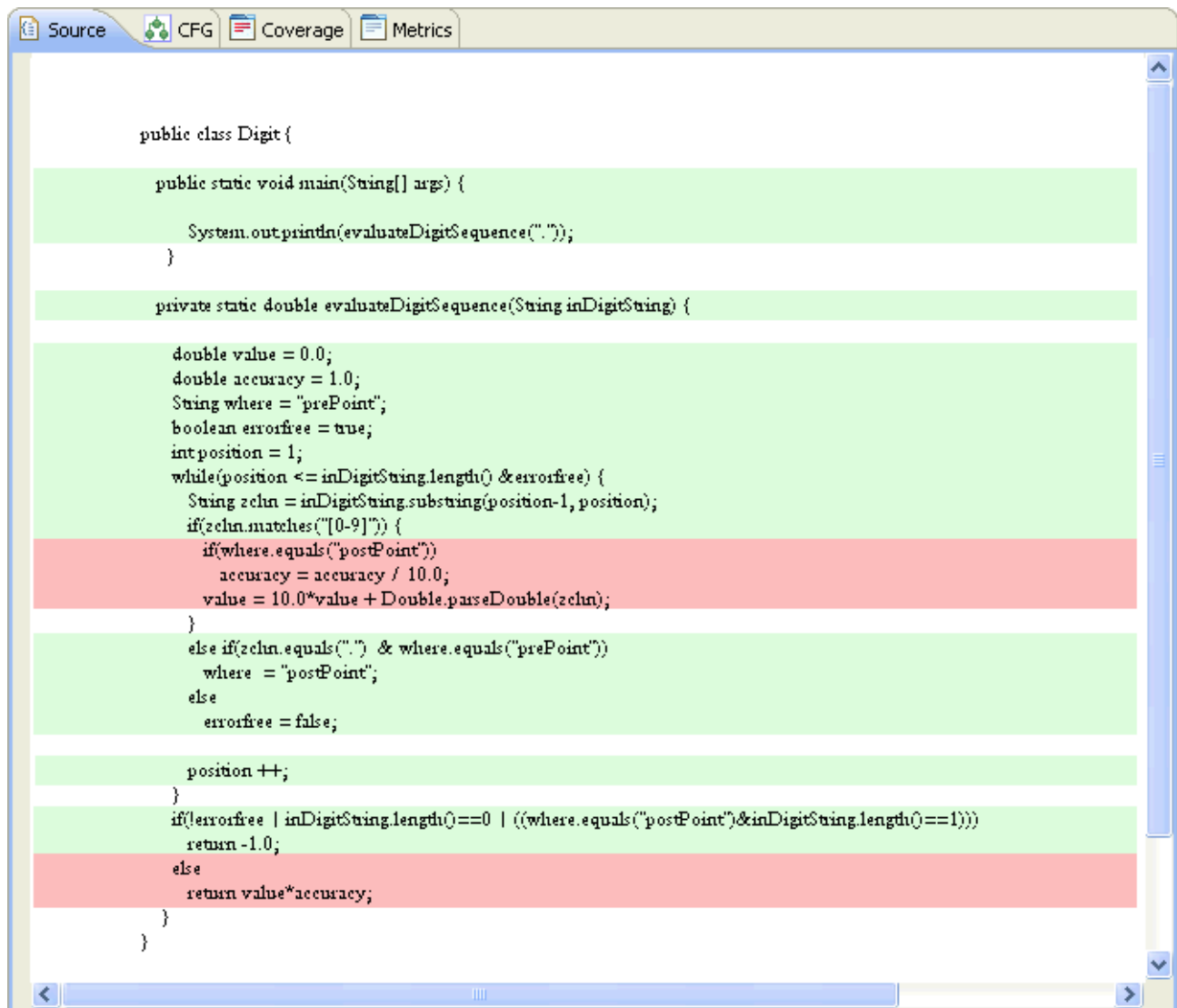
In the view *Source* the source code of the file selected in the view *Project* is shown. If individual classes or methods are selected, only the corresponding source code lines will appear.

In case the coverage indicator



***Show Coverage* in the toolbar is enabled, the coverage of the source code by the test logs selected in the view *TestLogs* is indicated with colors. Green lines in the source code were covered by the tests, whereas red ones were not traversed. A line containing several statements is marked green as soon as one of the instructions was**

covered. The corresponding colors for the line coverage and the syntax highlighting can be set in the menu item *Preferences*.



```
public class Digit {  
  
    public static void main(String[] args) {  
  
        System.out.println(evaluateDigitSequence("."));  
    }  
  
    private static double evaluateDigitSequence(String inDigitString) {  
  
        double value = 0.0;  
        double accuracy = 1.0;  
        String where = "prePoint";  
        boolean errorfree = true;  
        int position = 1;  
        while(position <= inDigitString.length() & errorfree) {  
            String zcln = inDigitString.substring(position-1, position);  
            if(zcln.matches("[0-9]")) {  
                if(where.equals("postPoint"))  
                    accuracy = accuracy / 10.0;  
                value = 10.0*value + Double.parseDouble(zcln);  
            }  
            else if(zcln.equals(".") & where.equals("prePoint"))  
                where = "postPoint";  
            else  
                errorfree = false;  
  
            position ++;  
        }  
        if(!errorfree | inDigitString.length()==0 | ((where.equals("postPoint")&inDigitString.length()==1)))  
            return -1.0;  
        else  
            return value*accuracy;  
    }  
}
```

Fig.: view *Source*

4.2.5 View *CFG*

While the view *Source* only shows line coverage and is primarily intended to give an overview of the source code coverage, the view *CFG* (Control Flow Graph) provides detailed information on the coverage of source code structures. On selecting a function in the view *Project*, the upper part of the view *CFG* shows the corresponding control flow graph. For every function appears at least one node for entering and one for exiting the function with the latter being the joint of all edges leaving the function. Each branching structure is represented by a node labeled as follows:

- conditional statement (short variant): "if", "true", "if-end"
- conditional statement (long variant): "if", "true", "false", "if-end"
- pre-test iteration/while and for loop: "iteration", "iter-body", "iter-end"
- post-test iteration/do while loop: "do", "iter-body", "iteration", "iter-end"
- selection control/switch statement: "switch", "case", "default", "switch-end"
- exceptions/try-catch block: "try", "try-block", "catch", "finally", "try-end"
- jump instruction: "break", "continue", "return", "throw"

For assigning nodes to their corresponding parts of the source code, it is sufficient to click on a node. Then the bottom of the view *CFG* focuses on the appropriate section of the source code and the line is highlighted in yellow. If the option



Pin SourceView is selected in the toolbar of the view, only on the first selection of a node the corresponding source code line will be displayed, afterwards the source code remains 'pinned' and does

not scroll automatically anymore. Selecting the option



Show Number of Paths displays the number of edge traversals during the current test on the left of every edge of the control flow graph.

In order to be able to also view large graphs clearly, the user can scale the illustration of the control flow graph via the buttons



Zoom Out and



Zoom In. SOTA offers seven zoom levels. The first three zoom levels still display the condition coverage and the fourth level still shows the caption of the nodes. The three smallest zoom levels reduce the graph to blank squares. More detailed information is available via the tooltip or the node information dialog (see below).

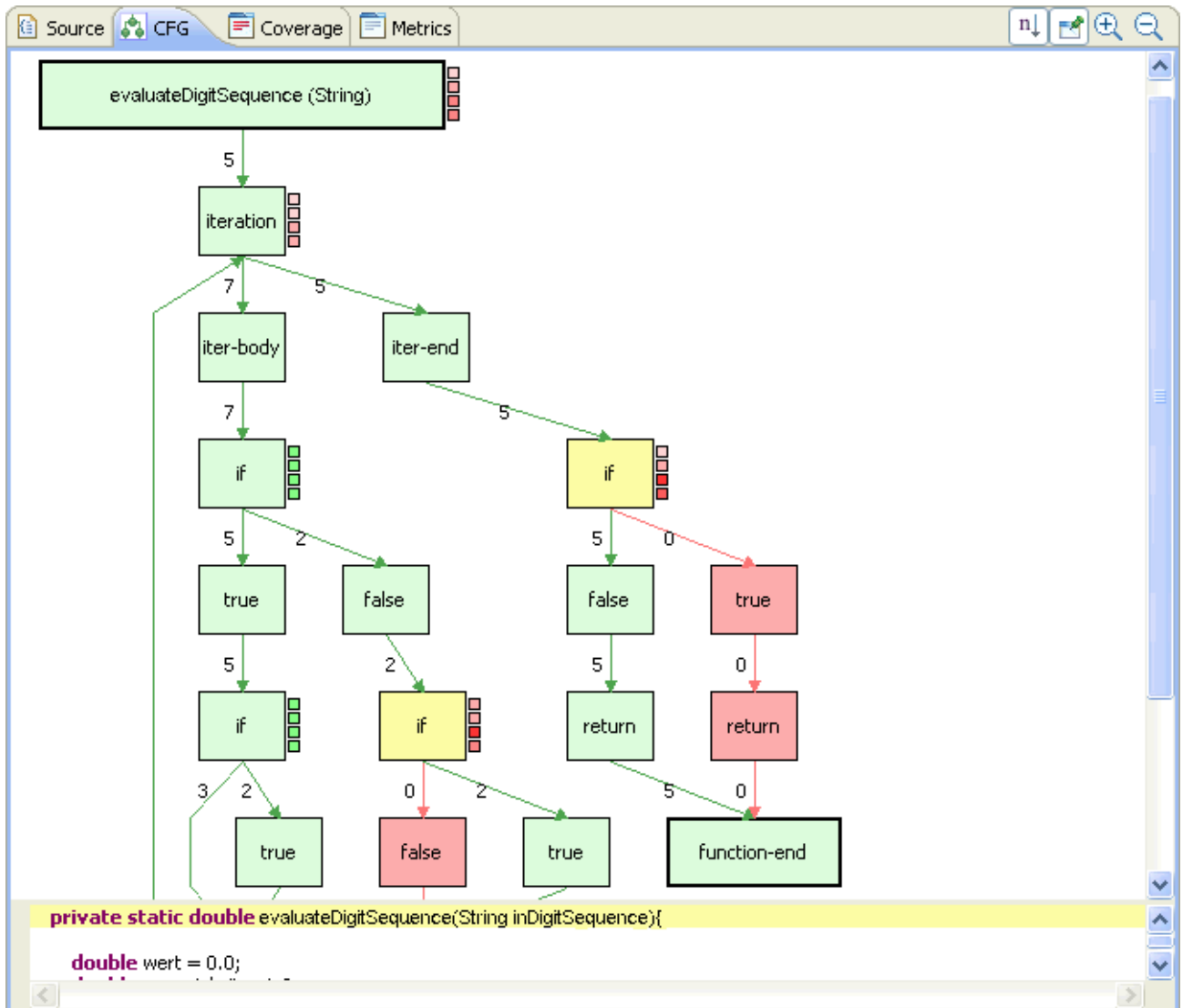


Fig.: view CFG

If test logs or test case data are selected in the view *TestLogs*, the color of the nodes and edges will change according to the coverage of the selected test case data. Also, the coverage data is updated automatically. Covered nodes and edges are colored green, whereas not covered ones are colored red. To detect covered nodes with multiple outputs of which not all are covered- which might be the cause for not covered sections of the source code- these are colored yellow. The user can configure this just like all other colors and also line style and thickness in the preferences.

The user can obtain additional information by navigating the cursor over a node. The tooltip that opens indicates the type and number of hits ('nrHits') by test case data of each node, as well as its internal project ID and the line number where it can be found. Branching nodes also contain

```

iteration
nrHits: 5
nrSkips: 0
nrSingleLoops: 3
nrMultipleLoops: 2
nrLoops: 7
ID: 22 Line: 15

```

Fig.: tooltip
(iteration-node)

information about the number of times each branch was taken. Therefore, the tooltip for if-nodes lists a value for each the true and the false branch, whereas switch-nodes contain an overview of the chosen selections and a list of the number of times each case was called. The case-node also contains this value as 'nrSelects' which can differ from the number of hits. The tooltip for iteration-nodes lists, next to the number of hits, details about how often the loop body was skipped ('nrSkips'), the number of times it was executed exactly one time ('nrSingleLoops') and two or more times ('nrMultipleLoops') as well as the overall number of executions ('nrLoops'). For the try-nodes which initiate the exception handling, it is mentioned how often the try block could be completed without an exception.

Information on condition coverage is viewable in two ways. On the one hand, located on the right side of each node containing a non-trivial condition are four small boxes representing the different degrees of condition coverage. From top to bottom, these are the single, minimal multiple, MC/DC and multiple condition coverage. The color indicates the degree of coverage from green meaning 100% to dark red meaning 0%. The threshold values and colors can be set in the preferences, where their presentation can also be turned off, and the user can select two additional sets of coverage related data to be displayed.

For more details about the coverage of individual nodes with or without a condition a dialog box which can be opened via double clicking the node is available. Firstly, all information from the tooltip and all relevant coverage data of this node are listed both by percentage and numerically. If the node contains a non-trivial condition, the logical structure of the condition as well as all assignments to their elements are also shown in a table. The first column represents the truth vector for the entire condition as read from the test log file. The second column contains the evaluation of the entire condition and the subsequent columns show the evaluation for each element. For each elementary condition the appropriate MCDC pair, if present, will be highlighted by clicking the corresponding table head.

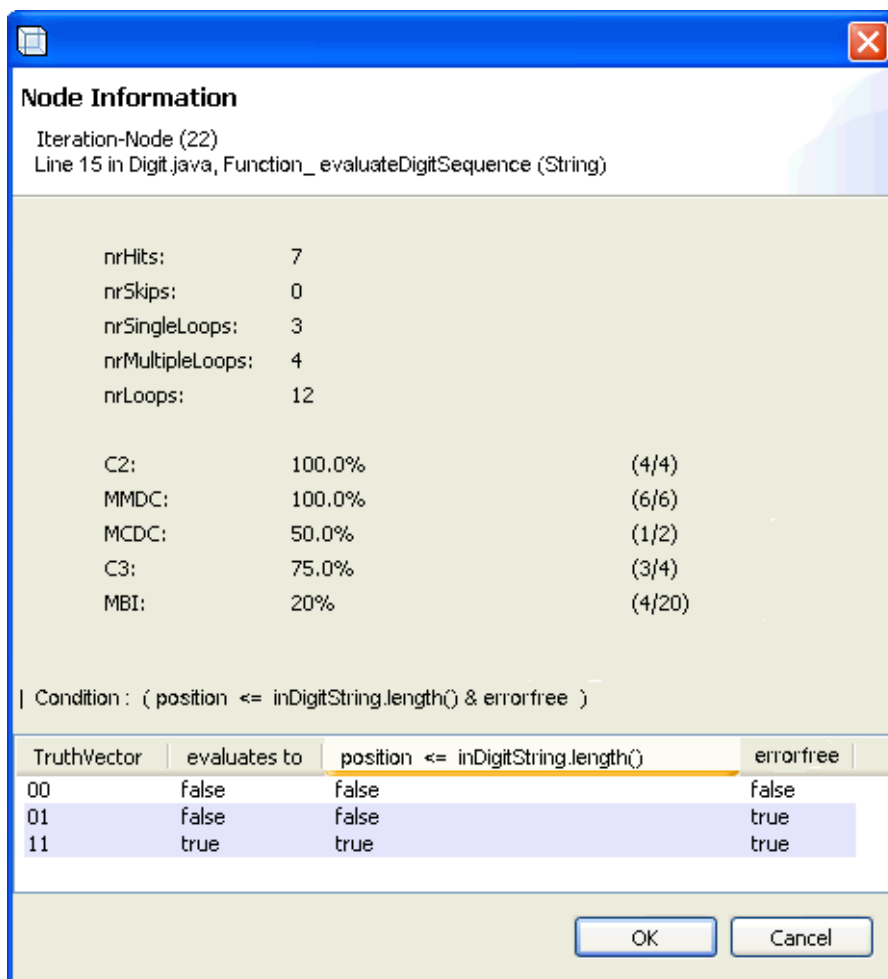


Fig.: node information with MCDC pair

4.2.6 View Coverage

The view *Coverage* offers a means for essentially analyzing the coverage data of the project. Analogous to the view *Project* all structures of the project are listed hierarchically in a tree, which can be configured

through the menu of the view to adopt a flat representation. For each project structure the percentage of the degree of coverage by various coverage metrics is specified in the corresponding columns. If the option



is activated in the toolbar, the values will be highlighted in favour of clarity. The individual colors and thresholds can be defined in the preferences.

The listed coverage metrics are: Function Entry Exit Coverage (FEEC), instruction coverage (C0), branch coverage (C1), single condition coverage (C2), minimal multiple condition coverage (MMDC), modified condition/decision coverage (MCDC), multiple condition coverage (C3), modified boundary-interior path coverage (MBI) and boundary-interior path coverage (BI). The definitions of the individual coverage metrics are listed in the Annex.

If a line instead of a value is displayed in the view, the corresponding metric is not applicable to the structure, because the class does not include e.g. conditions or instructions. The user can change the presentation of the value itself, via the option



Change Info, from percentage to ratio representation and back again. With the two buttons



Expand All and



Collapse All the hierarchical presentation of the project structure can be fully collapsed or expanded.

The user can sort the entire table by each column, i.e. by name and coverage ratio. In order to do that, it is necessary to click the corresponding column head. The sorting is based solely on the root elements of the tree, i.e. in hierarchical representation on the values of the files and classes respectively. However, in flat representation the structures can be sorted by functions.

Name	FEEC	C0	C1	C2	MMDC	MCDC	C3	MBI	BI
Project Digit	80,00%	89,47%	80,00%	75,00%	68,75%	30,00%	32,14%	13,04%	6,98%
Digit.java	80,00%	89,47%	80,00%	75,00%	68,75%	30,00%	32,14%	13,04%	6,98%
Digit	80,00%	89,47%	80,00%	75,00%	68,75%	30,00%	32,14%	13,04%	6,98%
main (String[])	100,00%	100,00%	---	---	---	---	---	100,00%	100,00%
evaluateDigitSequ	66,67%	88,89%	80,00%	75,00%	68,75%	30,00%	32,14%	9,09%	4,76%

Fig.: view *Coverage*

4.2.7 View *Metrics*

SOTA calculates a number of static metrics during the syntactic analysis of the source code while parsing. These can be evaluated immediately after the project was read in the view *Metrics*. Similar to the view *Coverage* this view consists of a project tree and a mapping of the following metrics to each project unit: cyclomatic complexity, essential complexity, lines of code, number of instructions, number of branches, number of modified boundary-interior path segments, number of boundary-interior paths, number of instructions evaluating conditions, number of elements in all conditions and number of conditions. Explanations as well as [definitions](#) of individual metrics are listed in the Annex.

As in the view *Coverage*, the user can change the presentation of the project structure in the menu of the view and expand and collapse the presentation by clicking the buttons



Expand All and



Collapse All. Sorting the table also works through clicking the column heads.

Name	Cycl.C...	Ess.C...	LOC	#Sta...	#Bran...	#ModBIP	#BIP	#Cond...	#
Project Digit	< 7	< 3	35	19	10	23	43	5	10
Digit.java	< 7	< 3	35	19	10	23	43	5	10
Digit	< 7	< 3	33	19	10	23	43	5	10
main (String[])	1	1	2	1	---	1	1	---	---
evaluateDigitSequence (Cycl.C... 6	6	2	25	18	10	22	42	5	10

Abb.: View Metrics

4.3 Preferences

4.3.1 Preferences View CFG

In the first options block of the preferences page of the view CFG, the user can determine which coverage metrics will be displayed as small square labels next to each node of the control flow graph. The default setting shows only four labels for the condition coverage metrics (C2, MMDC, MCDC, C3) next to all nodes containing non-trivial conditions, and at the function node. If the display of path coverage metrics are enabled (second option), three more labels will appear next to

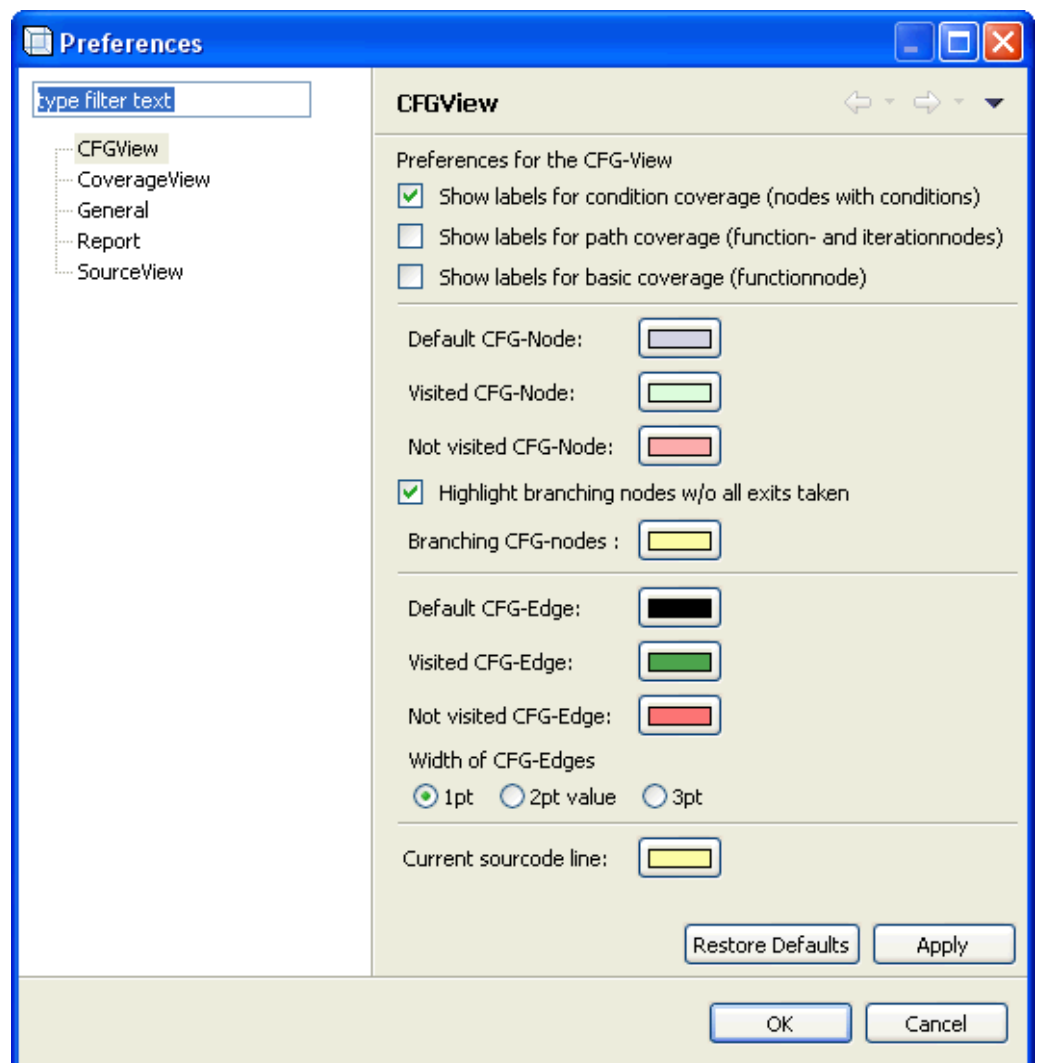


Fig.: Preferences View CFG

the function node

indicating the overall modified boundary-interior path coverage, the coverage of modified boundary-interior paths by the function body and the boundary-interior path coverage. Similar to the second label each iteration node also receives a label for the coverage of modified boundary-interior paths corresponding to the iteration. The last option causes the remaining three coverage metrics (FEEC, C0, C1) to appear at the function node.

In the second block, the user can configure the presentation of the nodes of the control flow graph. Colors for normal nodes without coverage information, covered and not covered nodes can be selected. For a more differentiated presentation branching nodes which are covered, but whose outputs are not covered completely, are highlighted. The color of the highlights can be set under *Branching CFG-nodes*. Removing the tick on the option above switches the differentiated presentation off.

Additionally, in the third block the color of the edges of the control flow graph for the presentation of edges without coverage, as well as for covered and not covered edges can be configured. The line thickness of the edges can also be set to a value from one to three.

Finally, the user can also change the color for highlighting lines in the source code corresponding to the selected node of the control flow graph.

4.3.2 Preferences View Coverage

Here, the colors for highlighting the percentage of the coverage in the view

Coverage can be defined as well as the percentage bounds.

Two elemental bounds are complete (100%) and no (0%) coverage. The user can add percentage values for five additional bounds, with the result that each cell is assigned a color according to the respective values. The default limits are: 25%, 50%, 75%, 90% and 99%. If the user specifies a bound with a value larger than one of the bounds above it, this bound will be ignored.

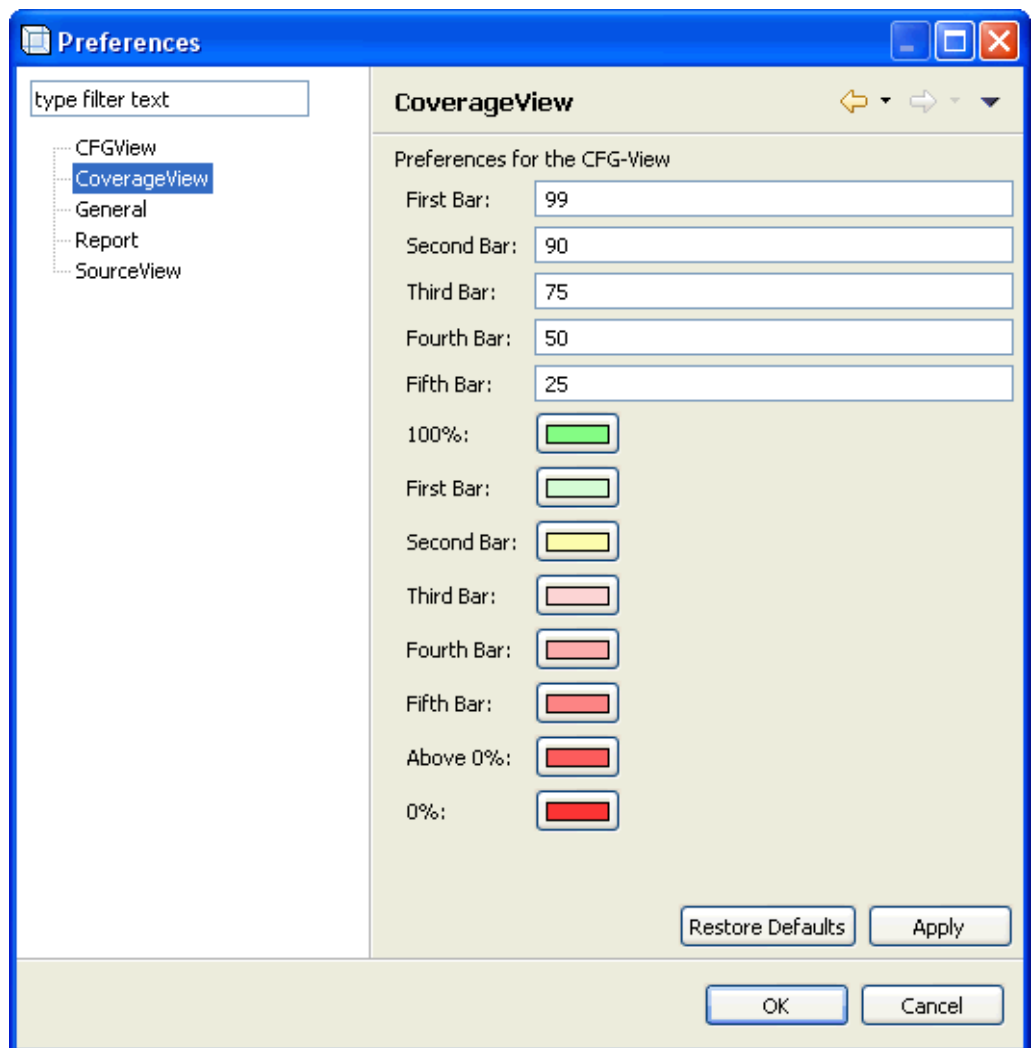


Fig.: Preferences view *Coverage*

Subsequently, the user can assign a color to each bound. In the table of the view *Coverage*, all cells where

the percentage of the coverage reaches the specified bounds will be colored as defined.

4.3.3 Preferences *General*

The user can change the general settings of SOTA under the preferences item *General*. In order to use an Ant buildfile to compile a project automatically, the corresponding Ant file (`bin\ant.bat`) of an installed version of Ant has to be embedded here. Then, it is possible to select the option *Build Project* in the dialog *Start Test*



, and the menu item *Build Project*



, as well as its equivalent in the toolbar are enabled. In Eclipse an Ant buildfile for compiling the project can be generated via *File -> Export -> Ant Buildfile*.

If *Create log file* is activated, system messages of SOTA will be saved as log files of the format `sota_<YY-MM-DD>_<index>.log` on each program boot. Also enabling the option *Overwrite existing log file* causes SOTA to create only one log file `sota.log` and overwrite it every time the program is booted.

The last two options define more general aspects of the behavior of SOTA. If *Parse instrumented source code* is activated, the project will be parsed again after launching the test, the view of coverage metrics will be disabled and the presentation of the project in all views will be based on the instrumented sources. In this case, the source code displayed in the view *Source* is always identical with the current sources, i.e. the two status indicators on the status bar always show the same value. Should the option be disabled, the backup will be parsed and displayed instead of the instrumented file. This allows an evaluation of the test logs independent of the status of the source files. The last option determines whether the test project will be compiled automatically, provided that a corresponding Ant script was included, in addition to restoring the original source files on stopping the test. Otherwise, the binaries of the test program would remain instrumented and continue creating logs.

4.3.4 Preferences *Report*

Under the preferences item *Report* the output of the report in a html-file can be configured.

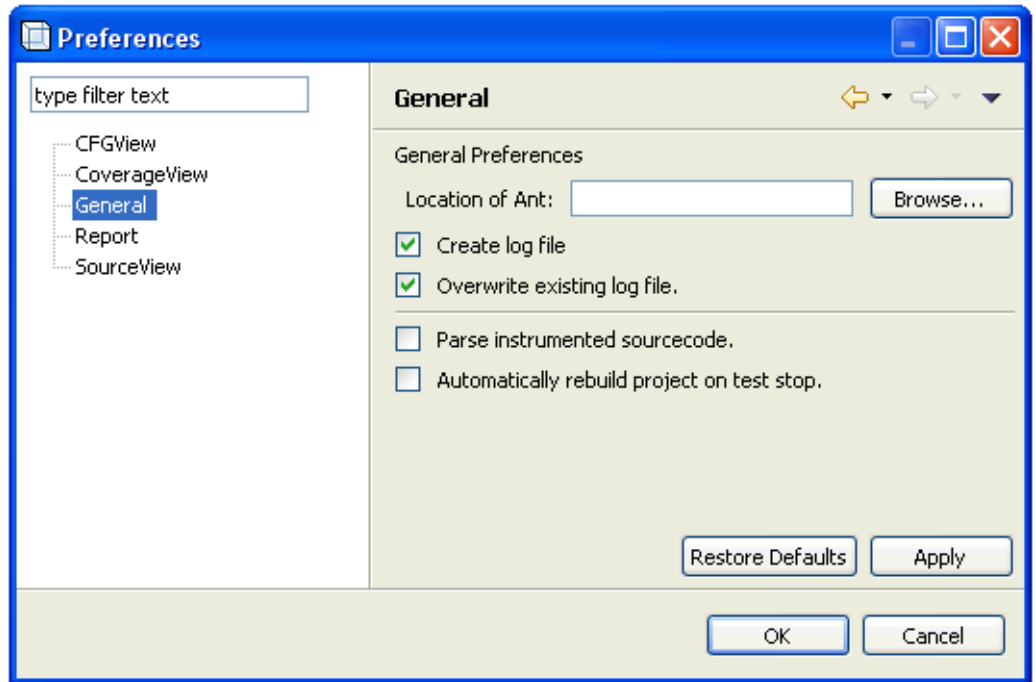


Fig.: Preferences *General*

Activating the option *Prompt for file name* in the first block, causes a file selection dialog to appear on selecting



Create Report in the menu. The dialog asks for the name of the report file which will be created. Otherwise, SOTA creates the file *report.html* in the root directory of the test program overwriting it every time a new report is generated, or SOTA saves each report according to the name scheme

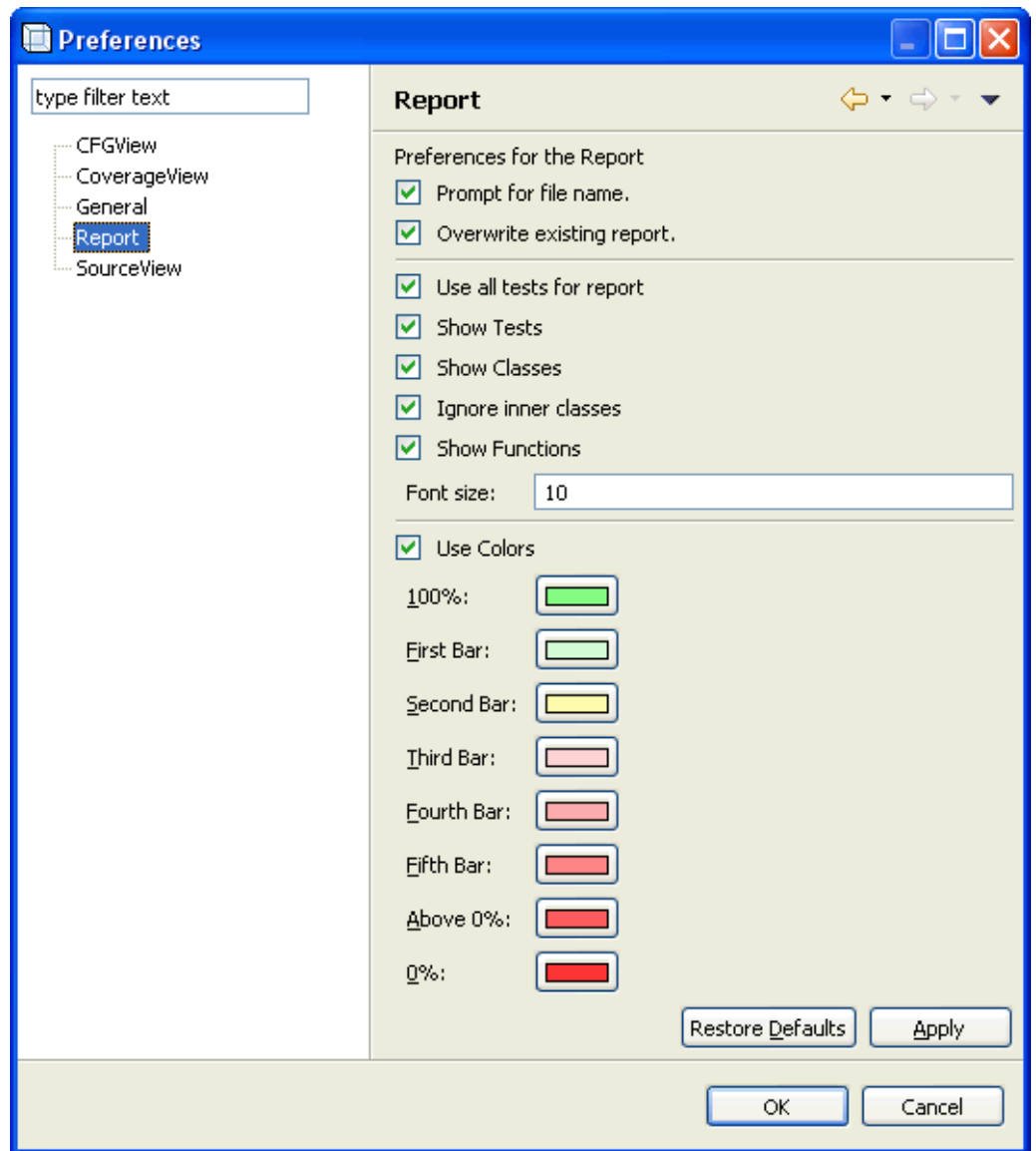


Fig.: Preferences *Report*

report_<date>_<index>.html. This behavior is determined by the option *Overwrite existing report*.

In the second block the user can define the content and the presentation of the report. It is possible to choose between using all test logs for the report or only the currently selected ones. Also, the user can select elements from the following list to be included in the report: the applied tests with ISchemes and descriptions, an overview of the coverage of all classes (including or excluding inner classes) and/or an overview of all functions, sorted according to their classes. Finally, the font size for the report file can be specified.

If *Use Colors* is selected, the coverage metrics in the report file will be highlighted in color according to the degree of coverage similar to the presentation in the view *Coverage*. The values for the bounds are adopted from the preferences item view *Coverage*, but here the colors of each bound can be defined separately for the report file.

4.3.5 Preferences View *Source*

The highlighting of syntax in the view *Source* can be adjusted in the correspondent preferences item. The colors for the

keywords of the language, comments, strings, and those comments added by instrumentation through SOTA can be chosen arbitrarily, and subsequently the font size for the presentation of the source code can be defined.

To view the coverage of the source code in the view *source*,

it is possible to select the background color for covered and not covered source code lines here. The colored highlighting will be applied as soon as test logs are read and if the option



Show Coverage is activated.

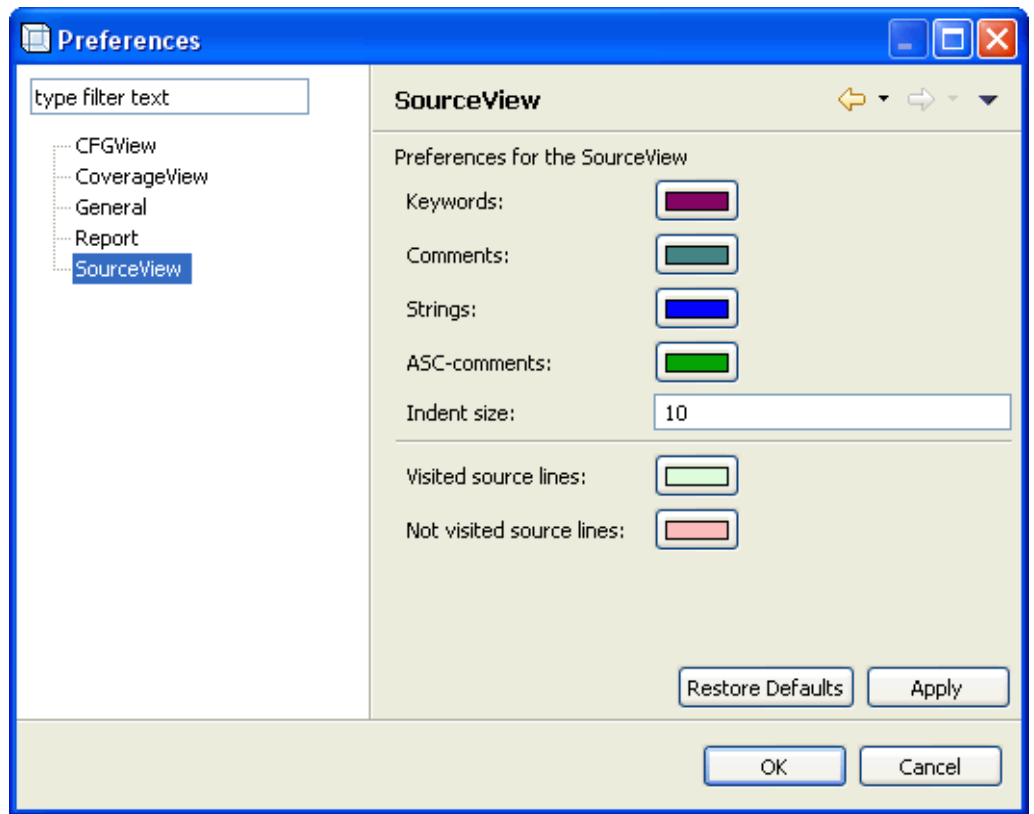


Fig.: Preferences view *Source*

4.4 Delete Project

A function 'Delete Project' was not implemented. This means that SOTA files corresponding to a project with the exception of the log files (see [View Testlogs](#)) have to be deleted by hand.

All SOTA files are described in 5.1. The files created by SOTA that have to be deleted in order to clear the system are in detail:

- SOTA root directory:
 - <projectname>.project
- root directory of test program (*Project directory*)
 - <xyz>.java.backup
 - report.html
 - report_<date>_<index>.html
 - \lib\ASCLogger.jar
- execution directory of test program (*Execution directory*)
 - <testname>.log
 - ASCLogger.ini

It is absolutely necessary to restore any instrumented source files to their original state (by choosing the menu item *Restore Sources*), before deleting the backup files as this is hardly possible without them!

[1 Introduction](#)

[2 Overview](#)

[3 Installation and Start of Program](#)

[4 User Interface and Functionality](#)

[5 Files](#)

[6.1 Overview](#)

[6.2 Project File](#)

[6.3 Report Files](#)

[6 Tutorials](#)

[7 Appendix](#)

5 Files

5.1 Overview

SOTA root directory

The SOTA root directory contains the SOTA system, as e.g. the boot file SOTA.exe and the Eclipse-Rich-Client- Platform installed with SOTA, as well as further special SOTA files and the project files.

sota.log, sota_<date>_<index>.log	Here SOTA logs all its activities.
language.spec	The specification file for all supported languages.
ASCLogger.jar	The logging component for Java test programs must be included in the test programm and demands an initialization file named <i>ASCLogger.ini</i> (see below) during the test.
<projectname>.project	For each project the general project data are noted here.
SOTA-ATM.jar	The automatic test module of SOTA enabling the testing functionality to be applicable without GUI. <i>SOTA-ATM.jar</i> is an executable jar file which can also be imported as a program library. This allows access via command line (scripts) or software. See the tutorial for more information.

Root directory of the test program (*Project directory*)

The root directory of the project contains (possibly in a subdirectory) the sources of the test program and at the same location their backups that are generated by SOTA on creating the project. SOTA puts all automatically generated report files at this location unless the user activated an inquiry via data selection dialog in the preferences.

(\src\)<xyz>.java	The source files of the test projects are (partially) instrumented after starting the test.
(\src\)<xyz>.java.backup	The backup of the original source files is created before starting the test for all source files that are not instrumented.
(\bin\)<xyz>.class	The compiled class files which may be instrumented

report.html,
report_<datum>_<index>.html
<antbuildfilename>.xml

depending on the state of the source files.

The report files generated by SOTA.

An possibly existant Ant buildfile allowing the compilation of the test program in SOTA. For Eclipse projects it can be exported via *File -> Export -> Ant Buildfile*.

<runscriptname>.bat

A batch file for booting the test program which allows, in case it is included in SOTA, the manual program test in SOTA in correspondance with the Ant build script.

Execution directory of the test program (*Execution directory*)

The test program is booted in the execution directory. In most cases this directory corresponds to the root directory of the program. An exception may e.g. be the testing of an Eclipse-RCP application in Eclipse, since then the execution direction is the root directory of the Rich-Client-Platform, i.e. in general: ..\eclipse\ .

ASCLogger.ini

On starting the test this initialization file for the logging component is copied into this directory where it is accessed by the class ASCLogger.jar in order to create the log file.

<testname>.log

The log files created by ASCLogger.ini.

5.2 Project File

To use SOTA it is not necessary to adjust the project files. However, should the user wish to adopt SOTA-ATM as an automatic test module, it may be beneficial to create or change the project files manually or with a script in order to gain comprehensive control over the test.

The project files used by SOTA are simple XML files which contain the project specific information as values of individual entities. Their format is specified by scheme definition [project.dtd](#).

The project file defines a project which is at least defined by the following values:

Name	The name of the project which must be identical with the filename without ending.
Language	The language of the project which must be listed in the language specification.
Prefix	The prefix which enables marking variables introduced by SOTA during the instrumentation process, this avoids name collisions.
BackupExtension	The ending used for backup files generated by SOTA.
ProjectDir	The root directory of the project.
ExecDir	The execution directory of the project.
SourceFiles	A list of source files (as SourceFile) belonging to the project.

The following values may be used optionally to enable special features of the program:

AntLocation	The path to the Apache-Ant installation. This is necessary to compile the project automatically.
AntBuildFile	The Ant buildfile, which enables the compilation of the buildfile.
RunScript	The script used for booting the project.
ISchemes	A list of instrumentation schemes (as IScheme) which enable the

variable instrumentation of the project. An IScheme consists of a name and a mapping of each structure of the project (data, class, function) to an instrumentation level, and, optionally, a description of the IScheme.

The following code is an exemplary project file for the project Digit. It contains the definition for an IScheme which instruments the only source code file according to level 1 and the method 'evaluateDigitSequence' according to level 2.

```
<Project>
  <Name>Digit</Name>
  <Language>Java</Language>
  <Prefix>asc</Prefix>
  <BackupExtension>backup</BackupExtension>
  <ProjectDir>D:\Development\workspace\Digit</ProjectDir>
  <ExecDir>D:\Development\workspace\Digit</ExecDir>
  <SourceFiles>
    <File>D:\Development\workspace\Ziffer\src\Digit.java</File>
  </SourceFiles>
  <ISchemes>
    <IScheme>
      <Name>Scheme F</Name>
      <Description>Digit.java Lvl1, evaluateDigitSequence Lvl2</Description>
      <Level1>
        <Item>Digit.java</Item>
      </Level1>
      <Level2>
        <Item>Digit.java:Digit::evaluateDigitSequence(String)</Item>
      </Level2>
    </IScheme>
  </ISchemes>
</Project>
```

5.3 Report File

The following report is a sample report for the project Digit for the input values '..', '.2', '1', '1.1' and without input value. (for more information on the program see [6.1.1.](#))

Each report file begins with the name of the project and the date of creation. Following that is a list of all coverage metrics including the values for this project obtained during the test, as well as individual static metrics. The other tables of the report file will only be created, if the user selected the corresponding options in the preferences. The standard settings cause the output of all tables.

If *Show Testlogs* is selected, a list with all test files used for this report follows. The corresponding ISchemes, as well as their descriptions, are also included. Activating *Use all tests for report* causes all imported test logs to be used for the report and to be listed here.

The option *Show Classes* generated a table which lists all classes of the project in addition to the overall project and their individual coverage metrics which may be highlighted in color according to their values (cf. Preferences).

Show Functions causes a table for each class to appear after the item *Detailed Coverage*. The tables contain the coverage metrics of the classes themselves and all their functions. A link from each class in the table *Coverage of Classes* refers to the corresponding list of their functions in the section *Detailed Coverage*.

A [comprehensive sample report](#) for the project HUSemOrg is included in the user documentation.

SOTA Coverage Report

Project: Digit

created: 2009-03-23 13:54:35

--	--	--	--

Class Digit	100,00	100,00	100,00	95,00	93,75	50,00	46,43	26,09	13,95
- main (String[])	100,00	100,00	---	---	---	---	---	100,00	100,00
- evaluateDigitSequence (String)	100,00	100,00	100,00	95,00	93,75	50,00	46,43	22,73	11,90

[1 Introduction](#)

[2 Overview](#)

[3 Installation and Start of Program](#)

[4 User Interface and Functionality](#)

[5 Files](#)

[6 Tutorials](#)

[6.1 Manual Program Test - Program 'Digit'](#)

[6.1.1 Program 'Digit'](#)

[6.1.2 SOTA and Eclipse](#)

[6.1.3 SOTA with Ant Buildfile and Start Script](#)

[6.2 Test using and External Testing System \(ATOSj\) - HU-Seminar-Organisation](#)

[6.2.1 ATOSj and HUSemOrg](#)

[6.2.2 SOTA and ATOSj](#)

[6.3 Automatic Testing System - SOTA-ATM](#)

[6.3.1 SOTA-ATM via command line call](#)

[6.2.2 SOTA-ATM API](#)

[7 Appendix](#)

6 Tutorials

The following three tutorials explain the behaviour of the three ways to use SOTA in manual program testing, program testing with an external test program and the test as a library in an automatic testing system.

6.1 Manual Program Test - Program 'Digit'

6.1.1 Program 'Digit'

The basis for the tutorial of the manual program test is a simple Java program which tries to read a positive rational number from a string. The program consists of the class *Digit* with a main function and the function *evaluateDigitSequence* which evaluates the string.

The string can either be passed as a parameter of the program or provided in the source code. The 'hardwired' string in the source code will be evaluated once the program is called without parameters. The output of the program is either the number in case the program was able to evaluate the string, or '-1' in case of an error, i.e. the string did not contain such a number.

```
public class Digit {
    public static void main(String[] args) {
        if(args.length==0)
            System.out.println(evaluateDigitSequence("."));
        else
            System.out.println(evaluateDigitSequence(args[0]));
    }
    private static double evaluateDigitSequence(String inDigitString) {
        double value = 0.0;
        double accuracy = 1.0;
        String where = "prePoint";
        boolean errorfree = true;
        int position = 1;

        while(position <= inDigitString.length() & errorfree) {
            String chr = inDigitString.substring(position-1, position);
            if(chr.matches("[0-9]")) {
```

```

        if(where.equals("postPoint"))
            accuracy = accuracy / 10.0;
            value = 10.0*value + Double.parseDouble(chr);
        }
        else if(chr.equals(".") & where.equals("prePoint"))
            where = "postPoint";
        else
            errorfree = false;
        position ++;
    }
    if(!errorfree | inDigitString.length()==0 |
((where.equals("postPoint")&inDigitString.length()==1)))
        return -1.0;
    else
        return value*accuracy;
}
}

```

6.1.2 SOTA and Eclipse

General procedure

The program 'Digit' is supposed to be written in Eclipse and then tested using a structure oriented program test. This procedure is split into 4 phases:

1. Eclipse: Create Program
2. SOTA: Preparatory Phase
3. Eclipse: Testing Phase
4. SOTA: Evaluation Phase.

The following data flow diagram reflects all important actions as well as inputs and outputs. The four phases result from switching between Eclipse and SOTA and are depicted in differing colors. The interface between Eclipse and SOTA is realised only with the specified files.

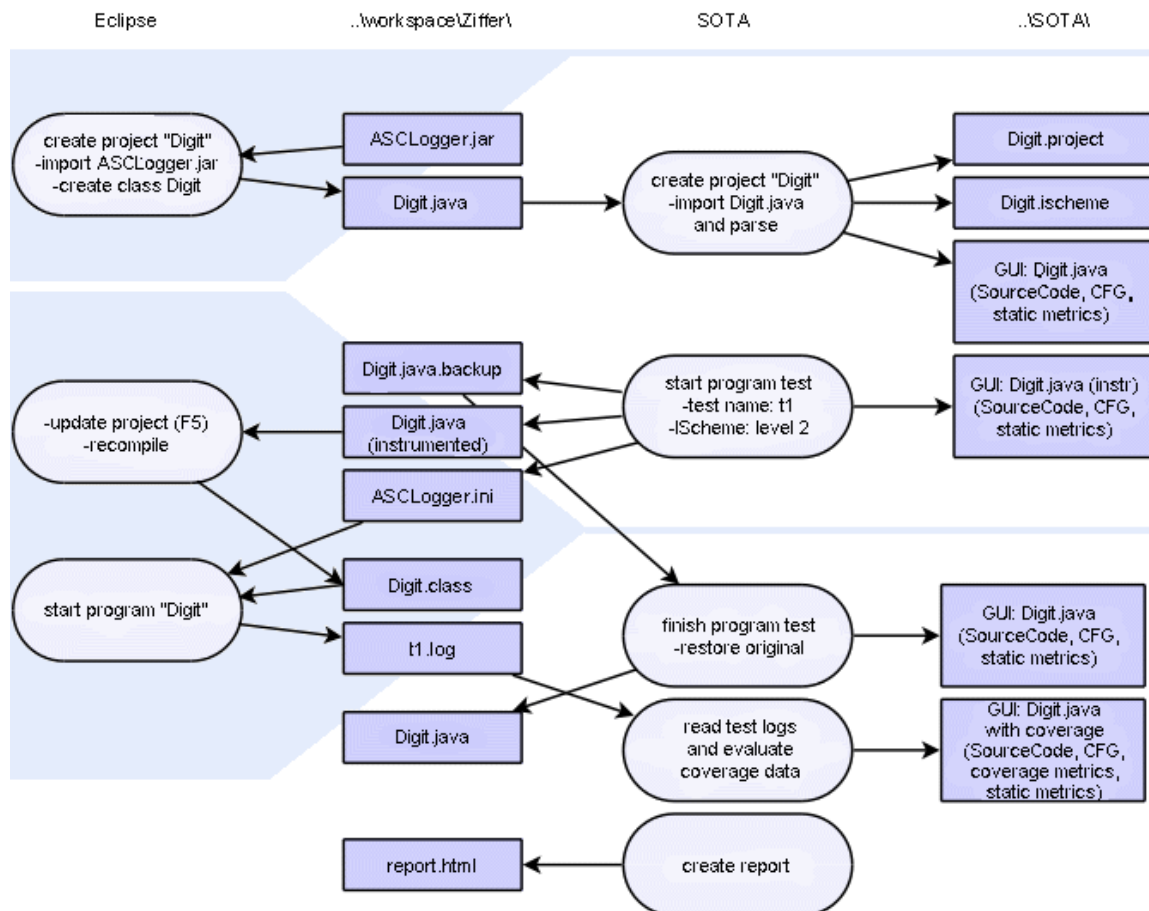


Fig.: DF-diagram manual program test with Eclipse

Detailed procedure

1. Eclipse: Create Program

The test program should be written in Eclipse. In case Eclipse is not installed yet, the user can do this following these instructions: <http://wiki.eclipse.org/Eclipse/Installation>.

Firstly, the user has to create a new project for the test program. This is done via the menu *File -> New ->*



Java Project. A dialog opens where the name for the project 'Digit' has to be entered, for all other options the default values can be used. Therefore the user can close the wizard on the first page clicking *Finish*.

To create a class *Digit* the user has to select *File -> New ->*



Class which calls the appropriate wizard. Here the name of the class - *Digit* - has to be entered before closing the dialog. Then a Java file is opened into which the source code above has to be copied.

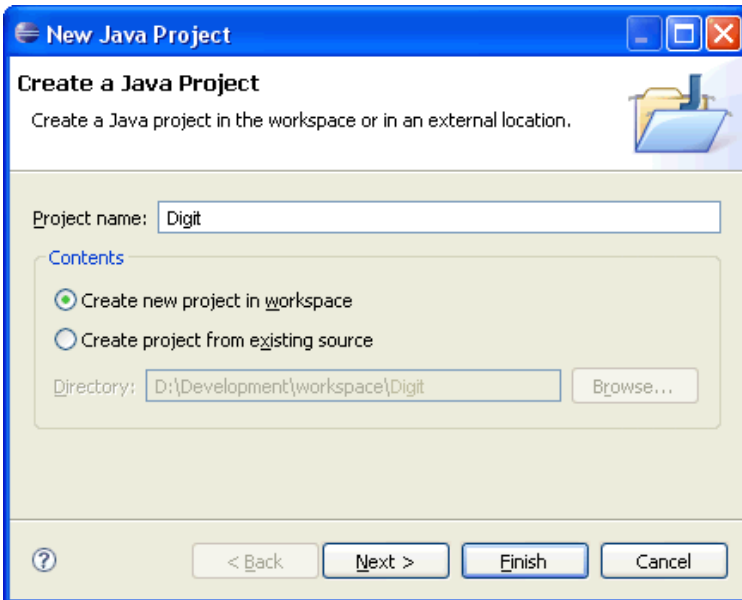


Fig.: Eclipse - *New Java Project*

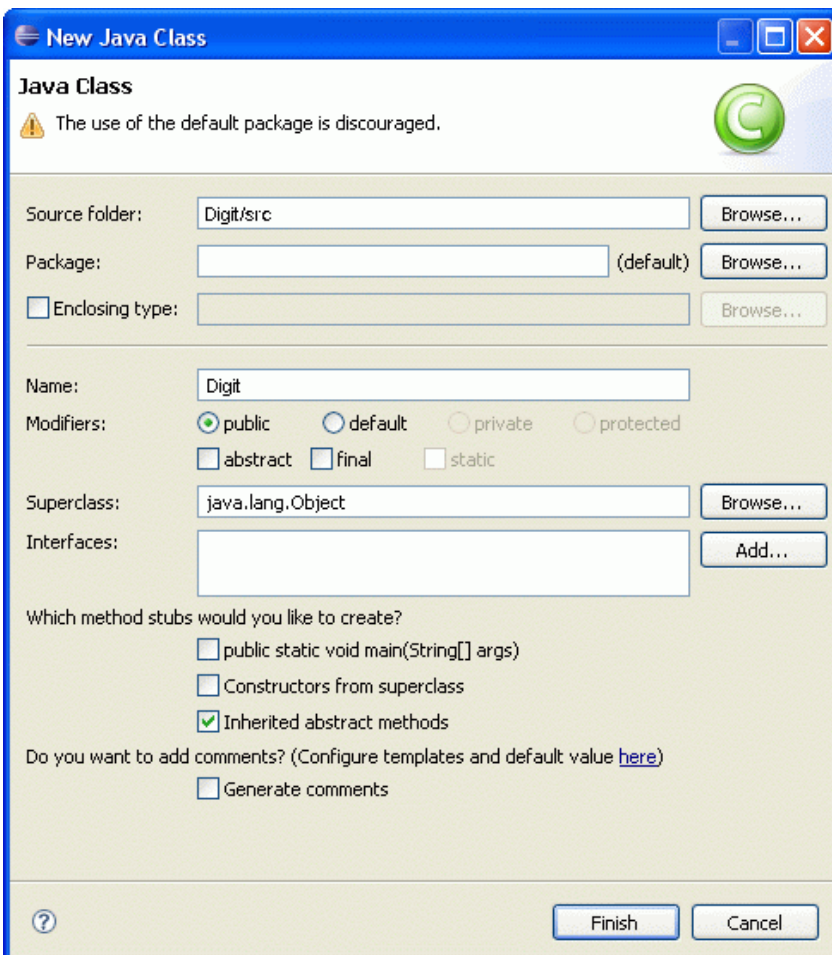


Fig.: Eclipse - New Java Class

Next it is necessary to include the logging component as a library in the project. In order to do this, the user has to copy the file *ASCLogger.jar* from the SOTA directory into a folder *lib* in the root directory of the project (*..\workspace\Digit*). On refreshing the project overview in Eclipse (F5) this file as well as the folder *lib* appear (see figure). Now the file has to be added to the build path of the project by right-clicking *ASCLogger.jar* in the context menu and selecting *Build Path ->*



Add to Build Path. The library is then included in the *Referenced Libraries*. With this the first phase - writing the program and preparing it for instrumentation by SOTA- is completed.

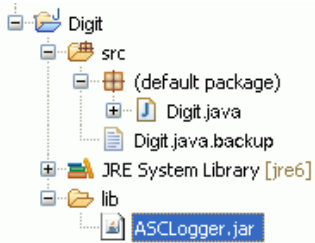


Fig.: Project 'Digit' with ASCLogger.jar ...

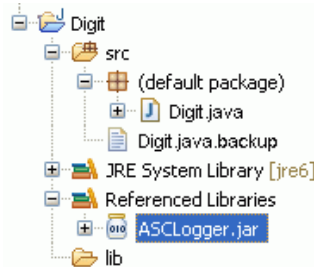


Fig.: ... and with ASCLogger.jar added to build path

2. SOTA: Preparatory Phase - Create Project

After starting SOTA via *SOTA.exe* the user has to create a project in SOTA for the test program. Selecting the menu item



New Project opens a two-page wizard guiding the creation process.

On the first page the name of the project - *Digit* - as well as the root directory (*project directory*) of the project created in point 1 have to be specified. The *execution directory* of the project is automatically linked to the same directory by SOTA needing no user input since both directories are identical in this case. The first page of the wizard is now completed and clicking the *Next* button calls the second page.

On the second page all sources of the project need to be imported. The project 'Digit' only consists of one file, so marking the root directory is sufficient. The *Finish* button closes the wizard and then SOTA reads and parses the source file.

Saving the project successfully completes the creation process. When SOTA is started again, the project can be loaded via the menu item



Open Project. Immediately after creating a project or loading an existing project respectively, the source code can be viewed in the view *Source* and the control flow graph of each function is visible in the view *CFG* after selecting one in the view *Project*. The static metrics that were computed while parsing the sources are now listed in the view *Metrics*.

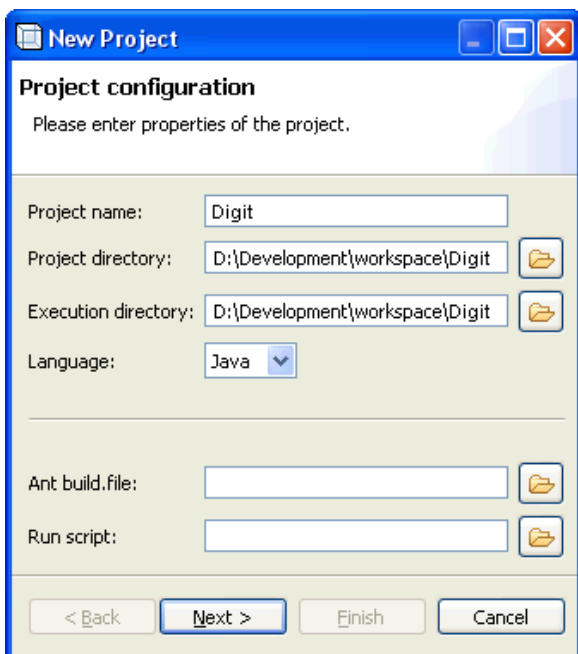


Fig.: first page of wizard

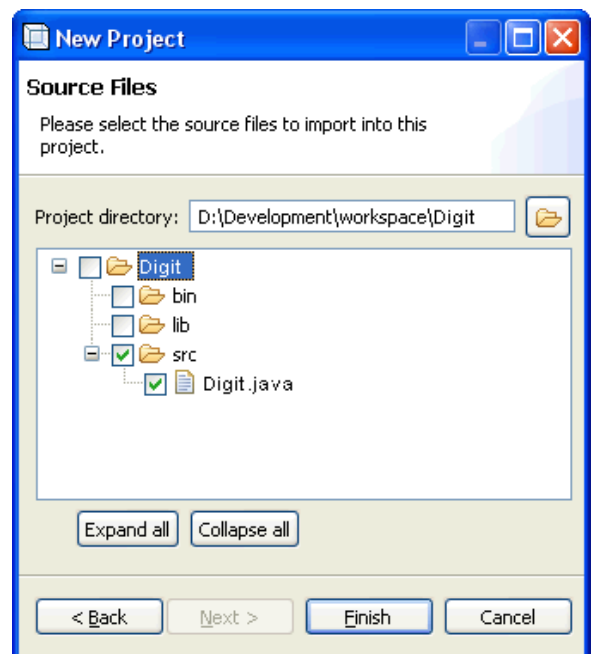


Fig.: second page of wizard

3. SOTA: Preparatory Phase - Instrumentation

The next step is to instrument the project for the next test run, i.e. adding instructions that effect in writing data into the log file during program execution, allowing its complete reconstruction. For this reason, the user has to select the menu item



Start Test. A dialog opens and a test name as well as an instrumentation scheme have to be selected. The test name also determines the name of the log file where all the log data is saved. The instrumentation scheme specifies the manner of instrumentation for all structures of the project. Usually, the user should select the IScheme *Level 2 instrumentation* which leads to a minimal instrumentation but computes all coverage metrics for all files.

After confirming the dialog the file *Digit.java* is saved as *Digit.java.backup* and then the instrumentation directions are added to the original file. With this the preparation of the instrumentation in SOTA is completed and the testing phase may begin.

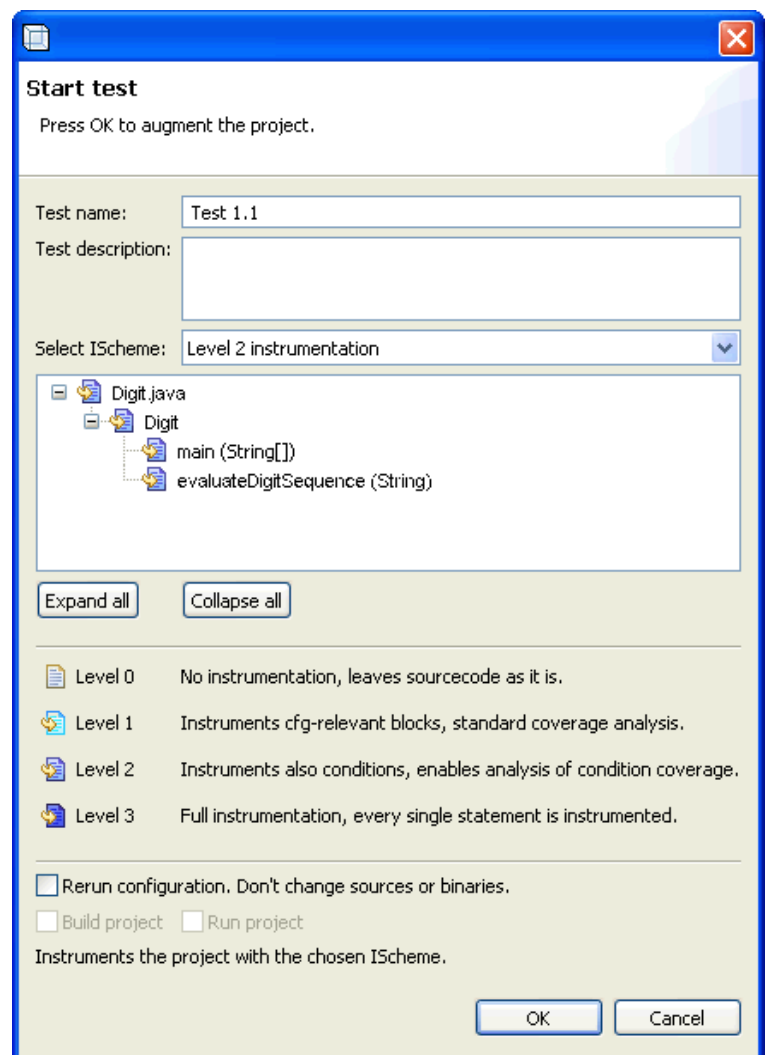


Fig.: dialog *Start Test*


4. Eclipse: Testing Phase - Compilation


At first the modified source code of the entire project has to be loaded in Eclipse. To do this, the project *Digit* has to be selected in the project overview and then refreshed by clicking 'F5' or via the context menu -> *Refresh*. Subsequently, Eclipse automatically compiles the new source files.

The instrumented sources require the library *ASCLogger.jar* which was included in step 2. Without the correct integration of the library the compilation process will evoke error messages.

5. Eclipse: Testing Phase - Program Test

After successfully compiling the program it is ready for the test. In Eclipse it is started via the button

 *Run As...* . On the first start Eclipse prompts whether the program is supposed to be started as an application or applet. Here, the user should choose *Application*. The following dialog asks for the application where the correct choice is *Digit*. Then the program starts. For the following program starts Eclipse should always choose the selected start configuration, so it is sufficient to click the button

 now labeled with *Run Digit*.

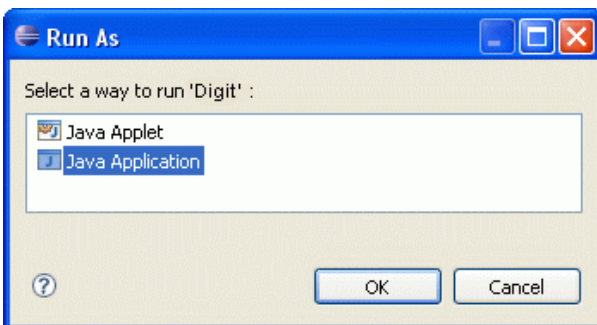


Fig.: dialog *Start Test*

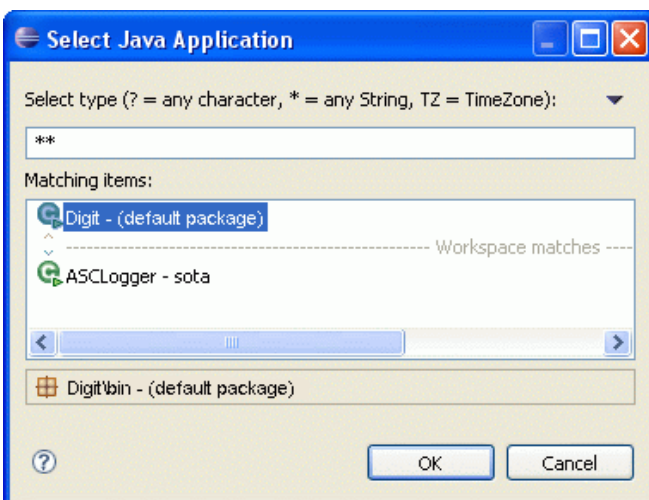


Fig.: dialog *Start Test*

The parameters for the program start in Eclipse could be written into the start configurations of the project, but since this is quite intricate for the simple test, it is recommended to change the string in the source code line

```
System.out.println(evaluateDigitSequence(""));
```

and then start the program without parameters. The console in Eclipse should now show the successful initialisation output of the *ASCLogger* as well as the result of the evaluation of the string.

Additionally, the corresponding log file with the name of the test should appear in the project overview of Eclipse after the first program test. Repeating the test adds the new log data to this file.

Remark

It is also possible, and often more convenient, to execute the test cases in an automated way, so that you don't have to enter the test data manually over and over again. Eclipse supports the creation, execution and evaluation of JUnit test-

cases. The procedure is the following:

- Add test file to Eclipse project, in a way that Eclipse will consider it a test file, i.e. use option New → JUnit Test Case
- Define a name for a file as `DigitTest` (If required, allow Eclipse to add build path for JUnit)
- In a created Java file, copy source code of a test file.

After that, the final project content should look like the following:

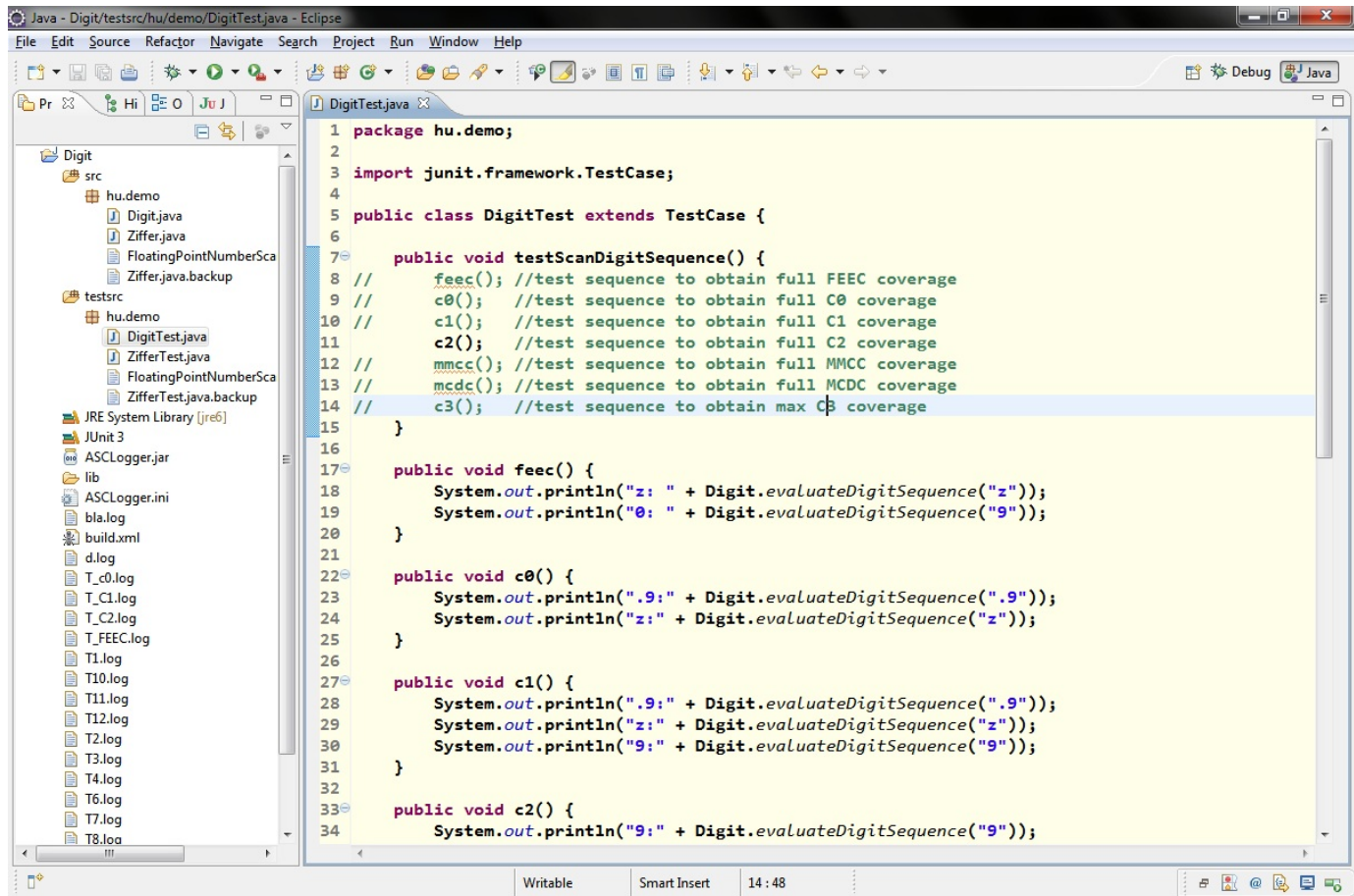


Fig.: An example for JUnit test case within Eclipse.

As can be noticed, in this test file, all of the test-cases are grouped according to the cover measure they should cover, which is stressed by the title of a method. Additionally, some formatting was added, so that test-cases are more readable. For each test-case, the form of a source code line is:

```
System.out.println("z: " + Digit.evaluateDigitSequence("."));
```

so that the resulting line in a log file would be "test-case" : "test-result".

After this, user should perform the actual test. This is performed by starting the test run with Run As → JUnit Test. The test will run as long as necessary and the results of that run can be observed in a Console window of Eclipse. With the above mentioned set of test-cases, the results are:

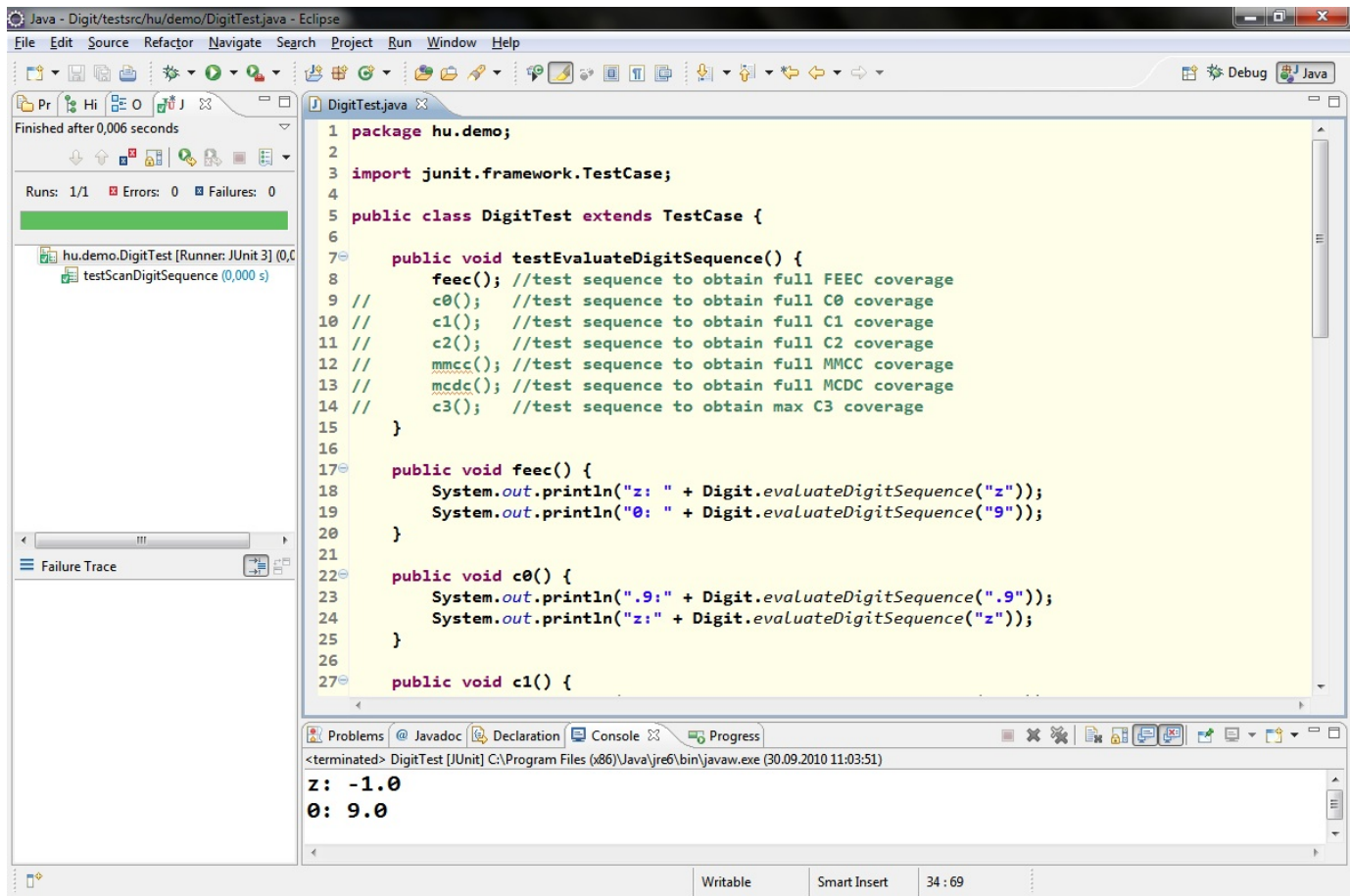



Fig.:JUnit test case execution within Eclipse.

6. SOTA: Evaluation Phase - Reconstruction

When the tests in Eclipse are completed, the user has to inform SOTA of this by selecting the menu item

 *Stop Test*. The sources are then reconstructed into their original state. Alternatively, the sources can also be rebuilt via the menu item

 *Restore Sources*.

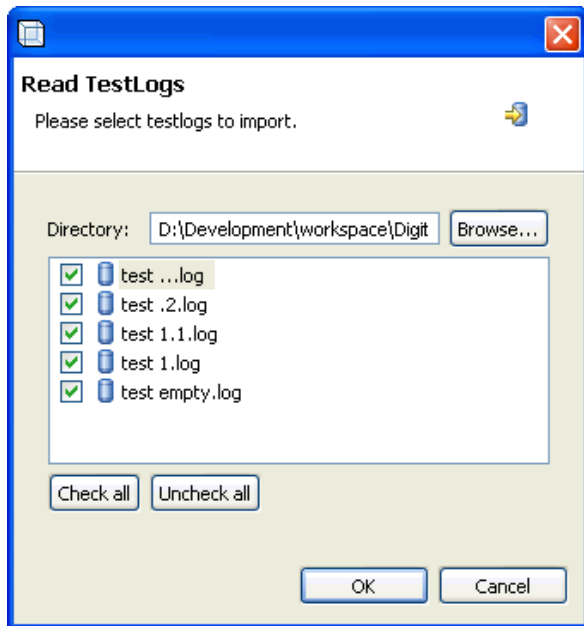


Fig.: dialog *Read Logs*

7. SOTA: Evaluation Phase - Evaluating the Test

In order to read the log files, a dialog opens immediately after selecting the menu item



Stop Test. Here, all files in the execution directory ending on log are listed for the import. The log files selected there are read, analysed for further evaluation and finally appear in the view *TestLogs*. The TestLogs marked there will now be used for computing the coverage metrics and determine the presentation of the coverage in the views *Source*, *CFG* and *Coverage*.

If the user wishes to import the log files when SOTA is not in testing mode, the menu item



Read Logs evokes the same behaviour.

8. SOTA: Evaluation Phase - Create Report

The program test is completed by creating a report. This is done via the menu item



Create Report which, depending on the settings in the preferences, either opens a dialog for entering a file name, or automatically generates a name and creates a report file in the root directory of the test program.

6.1.3 SOTA with Ant build file and start script

General Procedure

SOTA offers the possibility to execute the manual program test in SOTA by using two scripts. The procedure matches the procedure described in point 6.1.2 in principle, but the third phase, the testing phase, is not done in Eclipse which is only needed for creating the project. The following data flow diagram represents an overview of the steps taken in SOTA and the points at which the scripts complement the procedure.

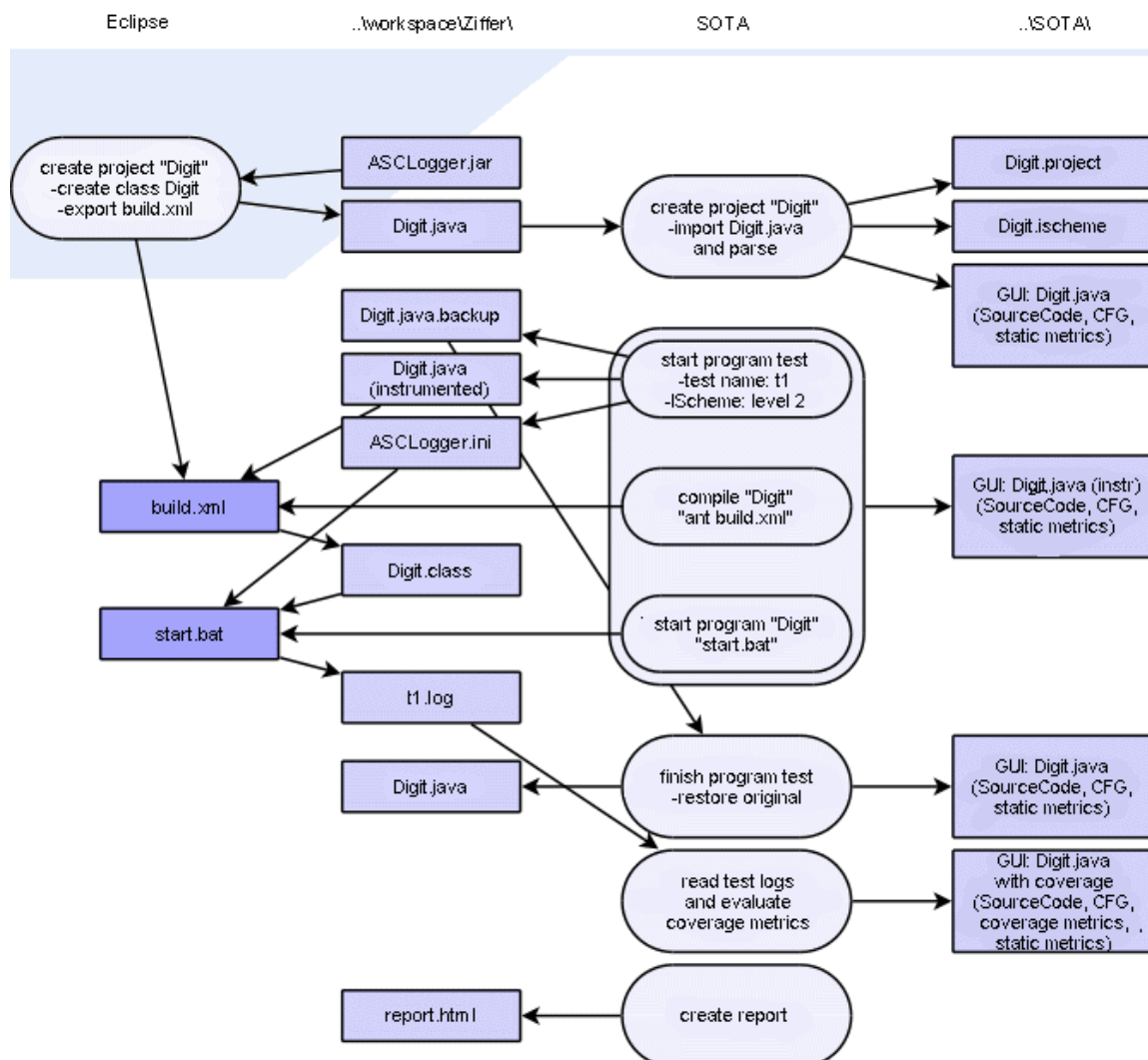



Abb.: DF diagramm *manual test with scripts*

Detailed Procedure

The detailed procedure matches the procedure described in point 6.1.2 and only differing steps are explained.

1. Create Project

Additionally to creating the project as in 6.1.2, an xml build file is exported from Eclipse. This can be done by selecting  *Export ...* in the context menu. A dialog opens where the user should select *General -> Ant Buildfiles*.

In the second dialog window only the corresponding project (here: Digit) has to be chosen and after finishing an xml file named 'build.xml' is created in the root directory of the project which enables compiling the complete project.

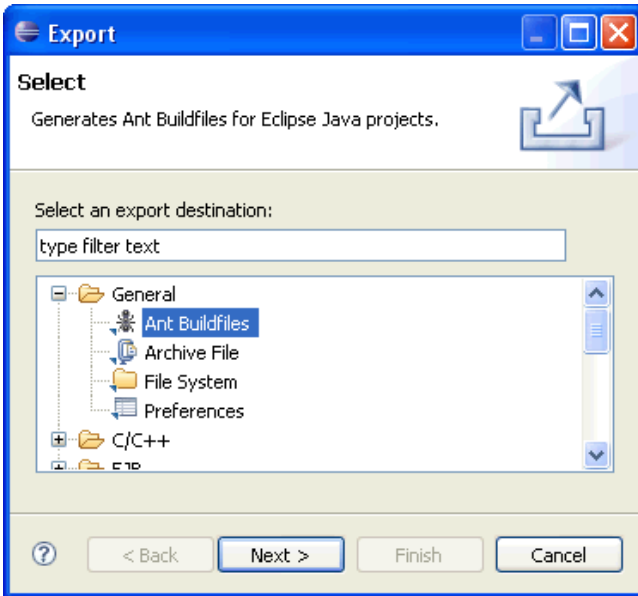


Fig.: Eclipse export dialog

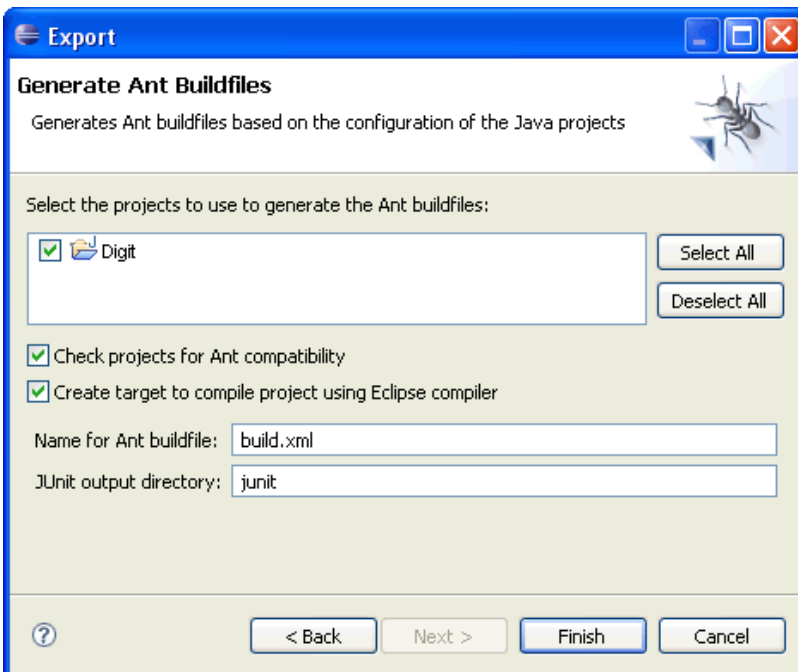


Fig.: Eclipse Ant build file dialog

Finally, a batch file 'Digit.bat' has to be created in the root directory of the project 'Digit'. This is necessary to start the program. The file has to contain the following Java command including the class path:

```
java -cp bin;lib/ASCLogger.jar; Ziffer
```

2. Preparatory Phase: Create Project

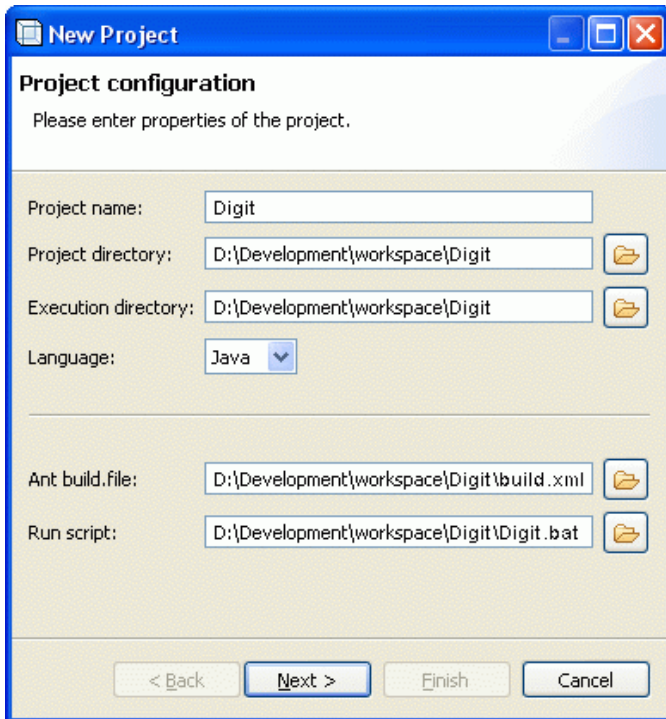



Fig.: create project with scripts

Creating the project in SOTA works as in point 6.1.2 with the only difference that the files 'build.xml' and 'Digit.bat' created in the first step, have to be imported on the first wizard page. However, it is also possible to add these files at a later point via the menu item

 *Configure Project.*

In order to successfully compile the project using the Ant build files, an Ant file 'ant.bat' that SOTA can execute also has to be included in the preferences under *Preferences -> General -> Location of Ant.* Since the settings in the preferences are effective for all projects, it is sufficient to make this entry once.

3. Preparatory Phase: Instrumentation / 4. Testing Phase: Compilation / 5. Testing Phase: Program Test

By including the two files the manual program test is available in SOTA. The dialog *Start Test* now offers the option *Build Project*, and as soon as this has been marked also the option *Run Project*. If the first option is active, the first Ant build file will be executed thus compiling the instrumented sources. The second option also runs the start script so that the program will be started. However, the start option of the current version of SOTA does not offer parameter passing to the test program which strongly limits testing options for our program 'Digit'. In order to test several strings, it is necessary to readjust the start script.

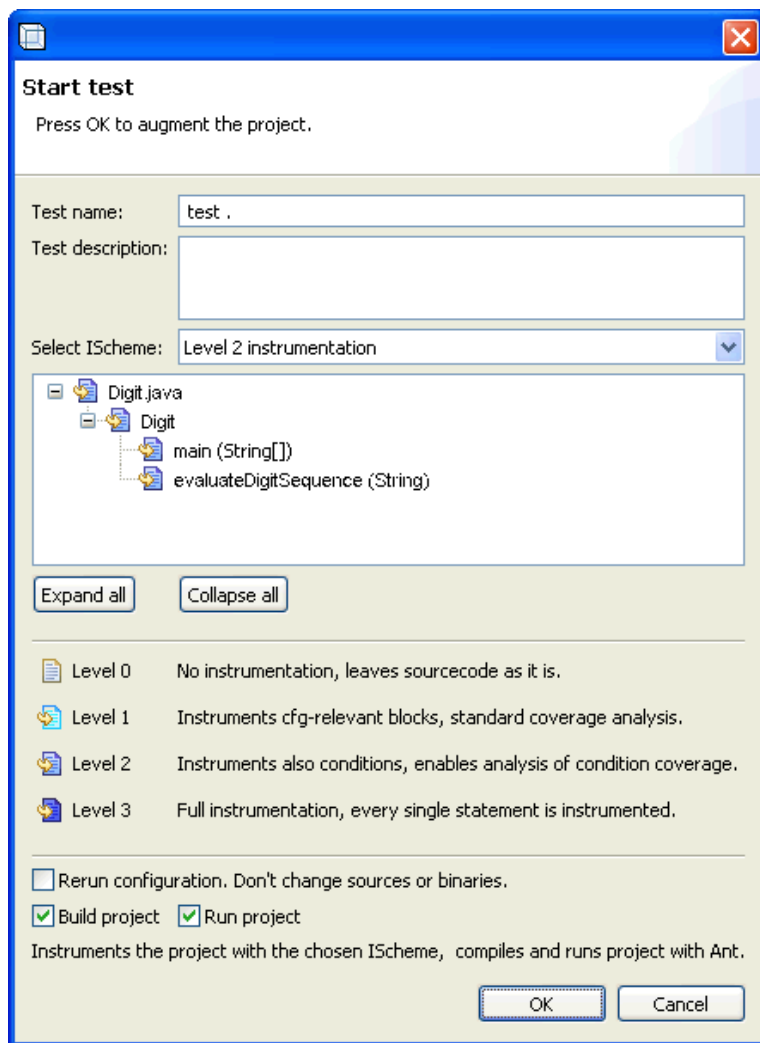


Fig.: dialog *Start Test* with compilation and start options

6. Evaluation Phase: Reconstruction / 7. Evaluation Phase: Evaluating the Test / 8. Evaluation Phase: Create Report

The remaining steps of the manual program test are consistent with the description in 6.1.2.

6.2 Test using an External Testing System (ATOSj) - HU-Seminar-Organisation

6.2.1 ATOSj and HUSemOrg

To use ATOSj as an external testing system and of HUSemOrg as a test program they need to be installed.

Instructions can be found here:

- Installation reference for the program for organising seminars HUSemOrg:
- Installation reference for ATOSj:
- ATOSj: project setup for seminar organisation:

6.2.2 SOTA and ATOSj

General Procedure

The testing procedure is similar to the manual program test with the exception that the actual testing is done using the external testing system. This leads to the following phases:

1. Eclipse: Create Program
2. SOTA: Preparatory Phase

3. ATOSj: Testing Phase
4. SOTA: Evaluation Phase.

As in 6.1.3 Eclipse is only used to create the program and plays no further role in the program test as an Ant buildfile is used. The following data flow diagram indicates the phases transferred to the external testing system.

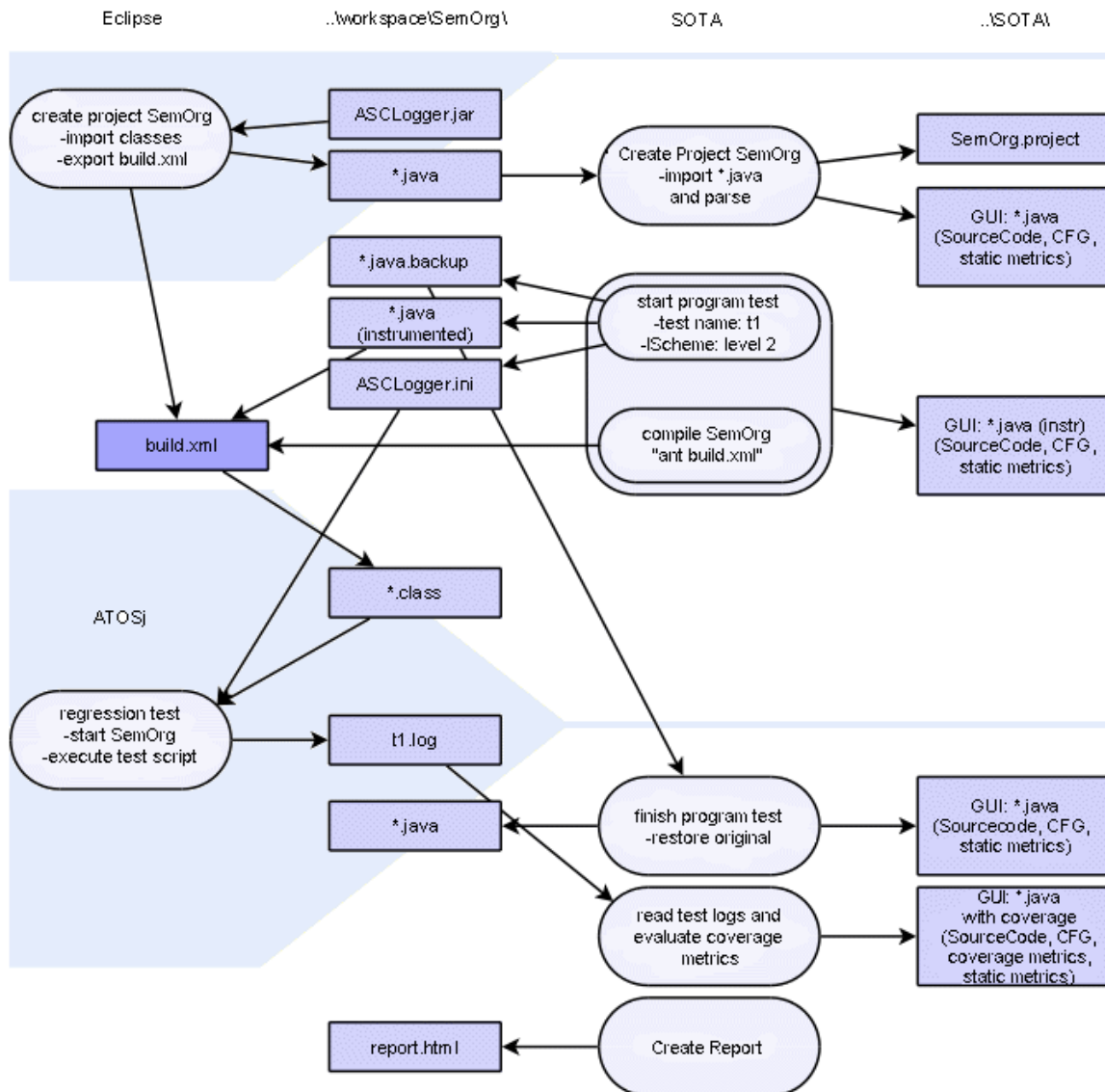


Fig.: DF diagramm *Manual Test with Scripts*

Detailed Procedure

The detailed procedure matches the procedure of the manual test with only a few exception which are described below.

1. Create Program

The program HUSemOrg should be unpacked into the Eclipse folder *workspace* and is already adapted to usage with SOTA. As in point 6.1.2 a project *husemorg* has to be created in Eclipse and the library *ASCLogger.jar* added to its build path.

2. Preparatory Phase: Create Project / 3. Preparatory Phase: Instrumentation / 4. Testing Phase: Compilation

The next three steps only refer to the preparation in SOTA and are identical to the manual test.

5. Testing Phase: Program Test

After the instrumented source files have been compiled ATOSj is started for the program test itself. The regression tests are performed on the instrumented classes. SOTA may be closed for this. The usage of ATOSj is in no way affected by the preparation phase with SOTA, there is no interaction between the two programs.

6. Evaluation Phase: Reconstruction / 7. Evaluation Phase: Evaluating the Test / 8. Evaluation Phase: Create

Report

The remaining steps match the description in 6.1.2.

6.3 Automatic Testing System - SOTA-ATM

SOTA-ATM (Automatic Test Module) is a library containing the test functionality of SOTA and needing no GUI at all. This permits controlling the instrumentation of projects as well as the evaluation of log files with other programs.

There are two control options offered for the module. On the one hand, it is possible to start SOTA-ATM as an executable Jar which can be controlled and started via command line parameters enabling its usage in simple scripts. On the other hand, SOTA-ATM is a library which can be integrated making it an interface between functions for testing the project and other programs.

6.3.1 SOTA-ATM via command line call

Parameters - Overview

SOTA-ATM is an executable Jar and can be called via command line with several parameters. The necessary values about the project can either be passed to SOTA-ATM by importing a project file (*-p*) or by passing them on calling SOTA-ATM (*-n*). A project file can be generated via the graphical user interface of SOTA, by using the option *-n* of SOTA-ATM or manually.

The other options call the different components of the module. The main aspects are the instrumentation of source files (*-i*), the reconstruction of the original sources (*-z*) and the evaluation of tests by reading the appropriate log files (*-t*) as well as creating a report thereafter (*-r*).

Additional functions can be combined with the ones above. Those are the compilation (*-c*) and start of the test program (*-s*), provided the necessary files are available. The order of options is irrelevant.

```
Usage: java -jar SOTA-ATM.jar [-options]

options include:
-c [ <ant-buildfile> ]
    compile sourcefiles; only if ant-buildfile is provided
-i ( Level1 | Level2 | Level3 | <ischeme-name> )
    instrument sourcefiles according to chosen level or IScheme
-n <name> <lang> <project-dir> [ <exec-dir> <src dir> ]
    create new project file
-p <name>.project
    open the project file
-r [ <report-file> ]
    create report-file; if no name is provided it will be stored in "report.html"
-s [ <runscript> ]
    start project; only if runscript is provided
-t <testname>.log [ <testname>.log ... ]
    name of testlog to create or to import
-z
    restore original sources
```

Parameters - detailed

-c [<ant-buildfile>]

The parameter *c* evokes the compilation of the project after the file operations have been completed. This option requires the project file to contain a reference to Apache Ant and to an Ant build file. The build file may also be passed via command line right after *-c*.

-i (Level1 | Level2 | Level3 | <ischeme-name>)

With the parameter *i* all sources of the project are saved as a backup and then instrumented. It has to be followed by either a name of an IScheme specified in the project file or one of the values 'Level1', 'Level2', 'Level3' which stand for a complete instrumentation of the entire project according to the respective levels. The project will only be instrumented, if none of the source files is instrumented already.

The parameter *i* requires the declaration of a test name via the parameter *t* and subsequently the usage of the parameters *r* to create a report and *z* to rebuild the sources.

-n <name> <lang> <project-dir> [<exec-dir> <src dir>]

The parameter *n* creates a new project on the basis of the passed values. A project name, the employed programming

language and the reference to the project directory have to be indicated at the least. The execution directory as well as the source directory are optional values and are set to the project directory if not specified otherwise.

The project file will be saved as <name>.project. Using the parameter *n* prohibits the usage of parameter *p*.

-p <name>.project

With this parameter the user can pass a project file which contains the data to characterize the project. Such a project file can be generated by using the graphical user interface, by calling SOTA-ATM with the parameter *n* (see above) or manually. Using the parameter *p* prohibits the usage of parameter *n*.

-r [<report-file>]

This parameter evokes the creation of a report containing the computed coverage metrics, on the basis of the test's log files that have to be included by calling the parameter *t*. If a reference to an html file follows the parameter, the report will be written into it. Otherwise a file 'report.html' will be created or overwritten.

The parameter *r* requires at least one log file passed via *t* and finally the parameter *i* evoking the instrumentation of the source files. If the source files are already instrumented, they will be restored to their original state before the test is read.

-s [<runscript>]

The parameter *s* starts the test program. It is necessary to reference a run script either as a parameter or in the project file.

-t <log-file>.log [<log-file>.log ...]

This parameter is used in two contexts. When an instrumentation via *i* is called, it defines the name of the test which is also the name of the log file that will be created. For creating a report with the parameter *r* all log files to be imported are listed after the parameter *t*.

-z

The parameter *z* restores all source files of the project from the backup files. It cannot be used together with the parameter *i* for the instrumentation.

Sample Usage

```
java -jar SOTA-ATM.jar -n Digit Java /workspace/Digit
```

A project 'Digit' of the programming language Java is created. Its project directory is '/workspace/Digit'. From there all java files are imported into the project. After successful completion all project information is saved in the file 'Digit.project'. Various settings, e.g. build files that should be used, can be changed in that file.

```
java -jar SOTA-ATM.jar -p Ziffer.project -i Level2 -t test1 -c -s
```

This call of the module has the effect that the project 'Digit' is parsed, fully instrumented according to level 2 and then compiled and started. Additionally, the value 'test1' is passed as a test name. The order of the options is irrelevant.

```
java -jar SOTA-ATM.jar -p Ziffer.project -z -c
```

With this call the original state of the sources is restored.

```
java -jar SOTA-ATM.jar -p Ziffer.project -t test1 -r
```

This call of the module creates a report for the project 'Digit'. The test 'test1' is read and the coverage metrics computed there are written into the standard report file 'report.html'. If the source files are instrumented, they will be restored to their original state first.

6.2.2 SOTA-ATM API

Overview

SOTA-ATM can be imported as a library into a program thus enabling the usage of the non-gui-functionality of SOTA for the static analysis and the coverage test. The javadoc documentation is available [here](#).

The central class for the program test is [SotaATM](#). Each of its objects represent a test instance of SOTA. A test instance can be configured by either loading a project file or passing an instance of [ProjectConfiguration](#) which contains all relevant data for the program test similar to a project file. For this instance the full functionality of SOTA, from instrumentation to report creation, is available (see javadoc).

Right after opening the project a [Metrics](#) object containing all metrics of the static analysis can be returned from the test instance. The coverage metrics are available in the very same object after testing the program and evaluating the generated log files.

To configure the instrumentation for a test, it is possible to define variable [ISchemes](#) which can assign a different instrumentation level to each structure (file, class, function) of the test project . The overall instrumentation of the project according to one instrumentation level is available via a [GlobalIScheme](#).

Sample Implementation of the Manual Test of HUSemOrg

The following java program code reads the given project file, starts a test with instrumentation level 2, compiles and starts the test program. After completing the test all source files are restored to their original state, the project is recompiled and an html report is generated from the imported test.

```
String projectFile = "D:/Development/eclipse/husemorg.project";
String reportFile = "report_test_1.html";
String testName = "husemorg_test_1";
String testDesc = "Test 08/15";

TreeSet<String> testSet = new TreeSet<String>();
testSet.add(testName);

SotaATM atm = new SotaATM(fileName);
atm.startTest(testName, testDesc, new GlobalIScheme("Level2", 2) , true);
atm.stopTest(testSet, true);
atm.createReport(reportFile);
```

[1 Introduction](#)

[2 Overview](#)

[3 Installation and Start of Program](#)

[4 User Interface and Functionality](#)

[5 Files](#)

[6 Tutorials](#)

[7 Appendix](#)

[7.1 Static Metrics](#)

[7.2 Coverage Metrics](#)

[7.3 Level of Instrumentation](#)

[7.4 More Terms and Definitions \(Glossary\)](#)

7 Appendix

7.1 Static Metrics

In SOTA static metrics sums up all metrics of the project that are obtained by static analysis of the source code. They are determined while parsing the source code and it is not necessary to execute the program contrary to the coverage metrics. The metrics provide on the one hand a means of estimating the complexity of the source code in terms of different criteria thus giving the user an indicator for enhancing the structure of the source code. On the other hand they enable the user to assess the costs of testing and the number of different tests for individual criteria respectively.

The static metrics are visible in the [viewMetrics](#) for all structures of the projects right after loading it. The values of the cyclomatic and essential complexity for classes, files and the project are the maximum of the values of their subordinate functions, for all other metrics these values are summed up.

Notes on the ModBI and BI values: The entire scope of exception handling eliminate the possibility to identify paths precisely. Therefore the computed value is always a lower bound, i.e. the minimal number of paths and sub-paths respectively that will be reached during the ModBI and BI test.

7.1.1 Cyclomatic Complexity

Cyclomatic complexity is computed using the control flow graph which represents all paths that might be traversed during program execution and their branching habits(cf. [view CFG](#)). The cyclomatic complexity $z(G)$ is defined as: $z(G) = e - n + 2$

where e is the number of edges and n is the number of nodes of the control flow G .

Therefore a function without branches in the program flow always has a cyclomatic complexity of 1, and each branch, e.g. an if-statement, increases the cyclomatic complexity by 1.

7.1.2 Essential Complexity

The definition of the of the essential complexity is closely related to the cyclomatic complexity. After recursively deleting all primitive control structures from a given control flow graph G , the cyclomatic complexity of the resulting graph G' is defined as the essential complexity $e(G)$ of the graph G : $e(G) = z(G')$.

All simple structures which contain no jumps, with the exception of break instructions in switch statements, are considered primitive structures. The existence of jumps out of control structures makes these structures and all structures including them irreducible thus increasing the value of the essential complexity.

7.1.3 Lines of code (LOC)

The number of the lines of code is listed here as one of the most primitive metrics of the source code, encompassing the appropriate structure. In contrast to all other metrics SOTA computes, LOC strongly depends on the structure of the source code and also the commentary. Therefore it should be regarded with care.

7.1.4 Number of Statements (#Statements)

Unlike the Lines-of-code-metric the number of statements offers an objective, format-independent metric for the extent of the project. In order to compute this metric all executable statements are summed up for all structures. The test of the coverage of statements consists of comparing the number of executed statements with the number of all statements.

7.1.5 Number of Branches (#Branches)

The number of branches is defined functionally in SOTA as a way of computing the branch coverage. While the number of branches in a function equates to the cyclomatic complexity - 1, in this case the number of branches is defined as the sum of the outputs of all branching nodes. So, for a function without branches the number of branches is zero, for each added if-statement the number increases by two.

7.1.6 Number of Modified Boundary-Interior Paths (#ModBI)

The number of modified boundary-interior paths corresponds to the number of subpaths through the control flow graph which have to be tested to fully execute the modified boundary-interior paths coverage test. The different kinds of subpaths are defined according to Liggesmeyer (*Software-Qualität*, 2002) as follows:

- all executable paths through a function which do not enter pre-test loops nor repeat post-test loops,
- all executable subpaths of each loop, which execute the body of the loop exactly once, disregarding the behavior for closed loops,
- all executable subpaths of each loop, which execute the body of the loop exactly twice, disregarding the behaviour for enclosed loops and possible following cycles of the loop body.

In the [View CFG](#) the user can find the number of subpaths which have to be tested according to the above definition for each loop in the node info (double-click on the appropriate node). Here the value is listed under 'ModBI'. In the node info of the function node the value for the entire function is listed as well as the values for subpaths of loops and subpaths through the entire function.

7.1.7 Number of Boundary-Interior Paths (#BI)

Analogous to the metric above, here the number of boundary-interior paths is specified for each function and accordingly for classes, files and the project the sum of all values contained in them. The corresponding paths are defined as all executable paths through the function in which the limit of the number of paths applies, so that on occurrence of loops only those paths need to be tested where for each loop

- the loop is skipped, i.e. the loop body is not executed (impossible for do-while loops),
- the loop body is traversed exactly once,
- the loop body is traversed at least twice, only regarding the first two iterations.

7.1.8 Number of Statements with Logical Conditions (#ConditionStmts.)

To compute the number of statements with logical conditions all occurrences of statements with evaluable logical conditions in the source code are summed up. Infinite loops ('while(true)') and loops iterating over a set ('for(Item item : set)') are not counted explicitly.

7.1.9 Number of Logical Atoms (#Atoms)

This metric corresponds to the sum of evaluable atomic conditions from all logical conditions. The logical atoms *true* and *false* are not counted since they are not evaluable with regard to the coverage test for conditions and have no influence on the control flow.

7.1.10 Number of Logical Conditions (#Conditions)

The number of logical conditions contains the sum of all atomic and compound conditions. This value is important for computing the minimum multiple condition coverage. Die Anzahl der logischen Bedingungen enthält die Summe aller atomaren und zusammengesetzten Bedingungen. Dieser Wert ist für die Berechnung der minimal Mehrfach-Bedingungsüberdeckung wichtig.

7.2 Coverage Metrics

The actual aim of SOTA is to evaluate program tests by computing coverage metrics. By including instrumentations a log file is created with the necessary data allowing SOTA to reconstruct the program flow and the evaluation of the conditions in retrospect. From these data the most common coverage metrics are determined for the individual tests. These are then listed in the [View Coverage](#).

7.2.1 Function-Entry-Exit-Coverage (FEEC)

The test for Function-Entry-Exit-Coverage requires all inputs and outputs of each function to be regarded for full coverage. It is computed as follows:

- $FEEC = (\#visited\ function\ inputs + \#visited\ function\ outputs) / (\#function\ inputs + \#function\ outputs)$

In Java there exists only one input for each function. Counted as possible outputs are the normal function ending, in case it was reached, as well as all return-statements and all throw-statements outside of try-structures.

7.2.2 Statement Coverage (C0)

For the statement coverage it is necessary that every statement in the source code was executed. Since each statement is only listed in the log file after its execution when the source code has been instrumented according to instrumentation level 3, the statement coverage is usually determined from the logged key data of the control flow after the program test.

- $C0 = \#covered\ statements / \#statements$

Note: In the view *CFG* not all nodes correspond to statements and not every statement corresponds to a node. Therefore the C0-coverage cannot be computed from the covered nodes of the control flow graph, it is rather based on the value *#Statements* from the view *Metrics*.

7.2.3 Branch Coverage (C1)

The full branch coverage was reached, if all branches of the control flow graph are covered. Computing the percentage coverage is done differently in practice, to simplify matters SOTA computes this on the basis of the branches (cf. 7.1.5) as follows:

- $C1 = \#covered\ branches / \#branches$

7.2.4 Simple Condition Coverage (C2)

The simple condition coverage exclusively tests whether all logical atoms of the conditions were evaluated true as well as false. However, this does not mean that branch coverage was reached as a minimum goal, therefore it is hardly possible to draw any conclusions from the simple condition coverage. For computing the percentage coverage SOTA counts all evaluations of each atom and compares them with the target value.

- $C2 = (\#true\text{-evaluations of all atoms} + \#false\text{-evaluations of all atoms}) / 2 * \#atoms$

7.2.5 Minimal Multiple Condition Coverage (MMCC)

The minimal multiple condition coverage has established itself as a practicable condition coverage which also includes the branch coverage. Analogous to C2 all evaluation of the logical atoms are regarded here as well as all compound, complex conditions. These have to be evaluated as true as well as false during the tests. The number of logical structures which have to be analysed corresponds to the number of logical conditions listed under 6.1.10.

- $MMCC = (\#true\text{-evaluations of all conditions} + \#false\text{-evaluations of all conditions}) / 2 * \#conditions$

7.2.6 Modified Condition/Decision Coverage (MCDC)

An even more exact test criterion is the modified condition/decision coverage test. To fulfill this coverage is not only necessary that all logical atoms of every condition adopt the values true and false. Additionally, it should apply for each atom that configurations of these conditions exist, which only differ in this atom and lead to an alternative evaluation of the complete condition. This ensures that the test checked whether changing the logical value of each atom would have an influence on the total condition. The two truth vectors of a condition fulfilling these requirements for an atom are called MCDC-couple. The coverage metric is then calculated using the MCDC-couples as follows:

- $MCDC - \text{Modified Condition/Decision Coverage} = \#MCDC\text{-couples} / \#atoms$

7.2.7 Multiple Condition Coverage (C3)

The multiple condition coverage test requires the most comprehensive condition test, since all truth vectors of every condition need to be tested. This means the costs for the test grow exponentially with the number of conditions. Additionally, in most cases it is not possible to apply all combinations of truth values. However, these impossible combinations usually cannot be recognized easily. The costs for testing $2^{(\#atoms)}$ is merely reduced by using short-circuit-operators which stop evaluating the condition as soon as the result of the complete condition was determined irrevocably.

- $C3 = \#evaluated\ truth\ vectors / \#possible\ truth\ vectors$

7.2.8 Modified Boundary-Interior Path Coverage (ModBI)

The modified boundary-interior path coverage test is a test proposed by Liggesmeyer which reduces the test cases compared to the boundary-interior path test (see definition in 7.2.9). In order to compute the coverage metric, it is necessary to compute the MBI-paths covering the paths for every one of them using a function during program testing. Then the sum of these covered subpaths is compared with the number of possible MBI-paths as defined in 7.2.9.

Since the number of ModBI-paths is only a minimum of possible subpaths according to this criterion, in practice more MBI-paths may be traversed (e.g. due to exceptions) than defined by this minimal bound. In this case the coverage value is naturally limited to 1.

- $ModBI = \#traversed\ MBI\text{-paths} / \#possible\ MBI\text{-paths}$

7.2.9 Boundary-Interior Path Coverage (BI)

The boundary-interior path coverage is computed like the modified boundary-interior path coverage. However, the BI-paths are computed only for the paths going through a function and then this value is compared to the number of possible BI-paths.

- $BI = \#traversed\ BI\text{-paths} / \#possible\ BI\text{-paths}$

7.3 Instrumentation Level

In order to allow the user to limit the memory requirements of the log files sensibly and variably, the source code can be instrumented in different levels. A configuration of the instrumentation is combined in an instrumentation scheme, short IScheme, and saved for the corresponding project. For all projects three basic ISchemes which correspond to instrumenting the code according to the respective levels, are provided by SOTA.

Level 0

Assigning level 0 as an instrumentation level for a structure causes this structure to be excluded from the instrumentation. This is sensible for functions which create a lot of log information (due to frequent execution or complex function flows), but have been tested adequately and can be excluded from further testing.

Level 1

The basic instrumentation is offered via level 1. Here all function entires, exits and all branching structures are instrumented, so that the control flow through the functions is can be reconstructed from these data. With these data it is possible to compute all coverage metrics except for the condition coverage.

Level 2

Additionally to level 1, the instrumentation according to level 2 also saves the configuration for each atom, provided it would also be evaluated in the program, in the log file. These data allow SOTA to compute the metrics as in level 2 as well as the condition coverage metrics for the program test.

Level 3

Finally, SOTA offers a full instrumentation of the source code with the instrumentation according to level 3. Next to the evaluated atoms, the log file will also include entries about the execution of all individual statements. Therefore the log file is considerably larger compared with the other instrumentation levels. This option of instrumentation is not only offered for the sake of completeness but also permits a detailed analysis of the control flow for programs terminating in an unusual way and exception handling.

7.4 More Terms and Definitions (Glossary)

Ant/Ant Buildfile

Apache Ant is a common tool, comparable to *make*, in Java development for automatically compiling source projects. Destinations and commands for the compilation are stored in an XML file, the Ant buildfile, which Ant can read and then execute the compilation.

When using Eclipse it is possible to easily export an Ant buildfile via *File -> Export -> Ant Buildfile* zu exportieren.

ASC-Logger.ini /
ASCLogger.jar

The testing of Java programs requires a logging component named ASCLogger.jar which has to be included into the project, then it administers saving the coverage data. The inclusion in Eclipse is done via *Project -> Properties -> Java Build Path -> Add JARs* or *Add External JARs*, depending on whether the user included the ASCLogger library into the project or is loading it from the SOTA directory. Information about the individual test cases, i.e. project name, test name, description and used IScheme are provided via the initialization file named

ASCLogger.ini which is created on starting the test, written into the execution directory of the test program and then read by ASCLogger.

Execution Directory of the Test Program

The execution directory of the test program is the directory from where the program is started, i.e. the directory where `java -cp .. classname` is executed or, when using a start script, the directory containing this batch file. In RCP-development with Eclipse the RCP-program is started from the base directory of the platform, i.e. Eclipse. In this case the execution directory is `..\eclipse\`.

The ASCLogger.ini is put into the execution directory. This file contains information about the test for the logging component. The log files are also written into this directory.

Base Directory of the Test Program

The base directory of the test program is its root directory where all source files and binaries (possibly in subdirectories) are located. From here are the sources and the project imported and the coverage report is put into this directory.

Base Directory of SOTA

The base directory of SOTA is `..\SOTA\`. Here are the executable SOTA.exe and the library ASCLogger.jar stored. Additionally, the project file `<projectname>.project` as well as the log file of SOTA with all program outputs are created in this directory.

Dynamic Program Test

Every test of a program that requires the program to be executed is a dynamic program test. Amongst those are functional (Black-Box-) and structure-oriented (White- or Glass-Box-)Tests. As a tool for structure-oriented testing SOTA calculates the [nine different coverage metrics](#) for each test.

Instrumentation Scheme / IScheme

SOTA offers several [levels of instrumentation](#) in order to enable limiting the overhead evoked by the instrumentation. An instrumentation scheme (short: IScheme) contains information about a specific way of instrumenting the project, i.e. it provides a mapping of all functions of the project to an instrumentation level.

SOTA always includes the three basic ISchemes allowing instrumentation according to levels 1, 2 and 3. If a new IScheme is created, these data will be saved in the project file `<projectname>.project` in the base directory of SOTA and will be available for usage in this project.

Start Script / Batch File

The start script (a batch file under Windows) is a file causing the start of the test program when executed. Therefore it merely has to contain a typical Java call `"java -cp .. classname"` in way specified for the project. If the start script is included in SOTA, the test program will be executable in manual program testing with SOTA.

Static Program Analysis

Contrary to the dynamic program test, the static program analysis is done without executing the project. The

information needed for the static analysis is determined only by parsing the program. This way SOTA identifies [ten different static metrics](#) which provide information about the structure and the complexity of the program and their components respectively.

<!ELEMENT Project (Name, Language, Prefix, BackupExtension, ProjectDir, ExecDir, SourceFiles, ISchemes?, AntLocation?, AntBuildFile?, RunScript?)>
<!ELEMENT SourceFiles (File*)>
<!ELEMENT ISchemes (IScheme*)>
<!ELEMENT IScheme (Name, Description?, Level0?, Level1?, Level2?, Level3?)>
<!ELEMENT Level0 (File*, Class*, Function*)>
<!ELEMENT Level1 (File*, Class*, Function*)>
<!ELEMENT Level2 (File*, Class*, Function*)>
<!ELEMENT Level3 (File*, Class*, Function*)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT File (#PCDATA)>
<!ELEMENT Class (#PCDATA)>
<!ELEMENT Function (#PCDATA)>
<!ELEMENT Prefix (#PCDATA)>
<!ELEMENT BackupExtension (#PCDATA)>
<!ELEMENT ProjectDir (#PCDATA)>
<!ELEMENT ExecDir (#PCDATA)>
<!ELEMENT AntLocation (#PCDATA)>
<!ELEMENT AntBuildFile (#PCDATA)>
<!ELEMENT RunScript (#PCDATA)>

SOTA Coverage Report

Project: husemorg

created: 2009-03-23 14:14:49

Function Entry-Exit Coverage (FEEC)	35,46%
Statement Coverage (C0)	37,99%
Decision Coverage (C1)	26,29%
Condition Coverage (C2)	20,42%
Minimal Multiple Decision Coverage (MMDC)	20,12%
Modified Condition Decision Coverage (MCDC)	8,67%
Multiple Condition Coverage (C3)	20,21%
Modified Boundary-Interior Path Coverage (ModBI)	0,00%
Boundary-Interior Path Coverage (BI)	0,00%

# Files	77
# Classes (TopLevel- + inner Classes)	420 (77 + 343)
# Functions	1379
# Lines	31915
# Statements	10747
# Conditions	1178

Tests

1	
Level 1 instrumentation	
2	
Level 2 instrumentation	
husemorg test 1	
Level 2 instrumentation	

Coverage of Classes

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Project husemorg	35,46	37,99	26,29	20,42	20,12	8,67	20,21	0,00	0,00
Class semorg.sql.tables.AbstractTable	58,06	37,93	35,29	39,47	39,47	31,58	39,47	2,13	0,02
Class semorg.sql.tables.Associate	56,25	67,71	37,50	20,00	20,00	20,00	20,00	40,00	40,00
Class semorg.gui.list.AssociateListWindow	38,46	37,14	33,33	50,00	50,00	0,00	50,00	29,41	29,41
Class semorg.gui.provider.AssociateTableProvider	86,05	90,00	89,19	75,00	75,00	50,00	75,00	86,49	86,49
Class semorg.gui.AssociateWindow	96,00	90,84	70,24	0,00	0,00	0,00	0,00	0,00	0,00
Class semorg.gui.util.AssociationTabControl	41,51	23,62	11,07	11,00	9,38	4,00	11,22	2,73	1,84
Class semorg.sql.tables.Booking	52,38	58,95	30,77	26,92	26,92	0,00	26,92	9,40	9,40
Class semorg.gui.util.CalendarControl	37,84	50,00	35,00	22,22	20,00	0,00	22,22	30,30	30,30
Class semorg.sql.tables.Client	48,78	52,67	25,00	29,17	29,17	16,67	29,17	29,27	26,83
Class semorg.sql.tables.ClientBooking	17,02	18,18	6,67	4,55	4,55	0,00	4,55	10,00	10,00
Class semorg.gui.list.ClientBookingListWindow	38,46	37,14	33,33	50,00	50,00	0,00	50,00	29,41	29,41
Class semorg.gui.provider.ClientBookingTableProvider	19,35	0,00	0,00	0,00	0,00	0,00	0,00	12,00	12,00
Class semorg.gui.ClientBookingWindow	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Class semorg.gui.list.ClientListWindow	46,15	40,00	33,33	50,00	50,00	0,00	50,00	35,29	35,29
Class semorg.gui.provider.ClientTableProvider	87,50	91,94	91,18	80,00	80,00	60,00	80,00	88,24	88,24
Class semorg.gui.ClientWindow	62,96	84,50	51,25	55,88	57,69	17,65	55,22	0,00	0,00
Class semorg.sql.tables.Company	91,22	84,64	43,33	46,30	46,30	7,41	46,30	6,17	6,17
Class semorg.sql.tables.CompanyBooking	47,06	47,42	18,75	20,00	20,00	20,00	20,00	38,46	38,46
Class semorg.gui.list.CompanyBookingListWindow	38,46	37,14	33,33	50,00	50,00	0,00	50,00	29,41	29,41
Class semorg.gui.provider.CompanyBookingTableProvider	65,52	75,00	68,00	50,00	50,00	0,00	50,00	60,87	60,87
Class semorg.gui.CompanyBookingWindow	73,91	86,23	55,56	66,67	68,42	40,00	65,52	13,16	13,16
Class semorg.sql.tables.CompanyInternalPresentation	40,00	37,82	18,75	19,23	19,23	7,69	19,23	6,80	6,80

Class semorg.sql.tables.Supervisor	0,00	0,00	---	---	---	---	---	0,00	0,00
Class semorg.gui.util.TableColumnProperty	0,00	0,00	---	---	---	---	---	0,00	0,00
Class semorg.gui.util.TimeControl	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
Class semorg.gui.util.UtilityGUI	100,00	92,00	83,33	63,04	66,13	43,48	65,79	21,43	21,43
Class semorg.sql.util.UtilitySQL	38,46	27,03	16,67	25,00	25,00	0,00	18,75	26,32	26,32

Detailed Coverage

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.AbstractTable	58,06	37,93	35,29	39,47	39,47	31,58	39,47	2,13	0,02
- AbstractTable (Timestamp, Timestamp)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getCreationDate ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getModificationDate ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setModificationDate (Timestamp)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createExtendedQueryString (String, Vector, String)	100,00	40,68	36,36	42,31	42,31	30,77	42,31	0,44	0,00
- getColumns (String)	100,00	100,00	---	---	---	---	---	100,00	100,00
- tableOK (String)	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- getNext (String, DBColumn, int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getPrevious (String, DBColumn, int)	40,00	29,41	33,33	0,00	0,00	0,00	0,00	25,00	25,00
- fireTableChangedEvent (int)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	71,43	100,00
- addDBTableChangedListener (DBTableChangedListener)	100,00	100,00	---	---	---	---	---	100,00	100,00
- removeDBTableChangedListener (DBTableChangedListener)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.Associate	56,25	67,71	37,50	20,00	20,00	20,00	20,00	40,00	40,00
- Associate (int, String, String, String, String, String, String, String, String, String, String, String, String, String, Date, Date, int, String, String, String, String, Timestamp, Timestamp)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getVector (ResultSet)	100,00	93,33	75,00	100,00	100,00	100,00	100,00	50,00	50,00
- getVectorFromDB (Vector, String)	100,00	100,00	---	---	---	---	---	100,00	100,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createAssociateTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getEntitlement ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getOccupation ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getPassword ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setEntitlement (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- setOccupation (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- setPassword (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- updateDB ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getNext (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getPrevious (int)	100,00	83,33	50,00	0,00	0,00	0,00	0,00	25,00	25,00
- getColumns ()	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.list.AssociateListWindow	38,46	37,14	33,33	50,00	50,00	0,00	50,00	29,41	29,41
- AssociateListWindow ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- AssociateListWindow (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- AssociateListWindow (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00

- createSpecializedMainTableListener ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createSpecializedMainTableListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedToolBarListeners ()	50,00	100,00	---	---	---	---	---	100,00	100,00
- createChooseButtonListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createChooseButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- openList ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getTableInputFromDB ()	100,00	36,36	33,33	50,00	50,00	0,00	50,00	16,67	16,67
- createWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.provider.AssociateTableProvider	86,05	90,00	89,19	75,00	75,00	50,00	75,00	86,49	86,49
- dispose ()	100,00	---	---	---	---	---	---	100,00	100,00
- getColumnImage (Object, int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getColumnText (Object, int)	87,88	91,18	89,19	75,00	75,00	50,00	75,00	87,50	87,50
- addListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00
- isLabelProperty (Object, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.AssociateWindow	96,00	90,84	70,24	0,00	0,00	0,00	0,00	0,00	0,00
- AssociateWindow (Shell)	100,00	100,00	100,00	0,00	0,00	0,00	0,00	66,67	33,33
- AssociateWindow (Shell, Associate)	100,00	100,00	---	---	---	---	---	100,00	100,00
- setInput (Associate)	100,00	84,26	63,46	0,00	0,00	0,00	0,00	0,00	0,00
- setInputComponentsEnabled (boolean)	100,00	100,00	100,00	0,00	0,00	0,00	0,00	66,67	33,33
- getEntitlementValue ()	100,00	100,00	100,00	0,00	0,00	0,00	0,00	28,57	28,57
- checkInput ()	100,00	100,00	100,00	0,00	0,00	0,00	0,00	18,75	18,75
- createButtonListener ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- commitInputToDB ()	100,00	23,53	33,33	0,00	0,00	0,00	0,00	25,00	25,00
- confirmClose ()	80,00	76,92	66,67	0,00	0,00	0,00	0,00	37,50	37,50
- onSave ()	100,00	100,00	100,00	0,00	0,00	0,00	0,00	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.util.AssociationTabControl	41,51	23,62	11,07	11,00	9,38	4,00	11,22	2,73	1,84
- AssociationTabControl (Composite, int, Shell)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	66,67	33,33
- hasLockedId (SimpleIDKey)	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- onDelete ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- determineDeleteMessage (DistinctVector)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- openEditItemWindow ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- openNewItemWindow ()	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- openListWindow ()	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- onDisconnect ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- configureClassSpecificViewerParts ()	100,00	45,16	33,33	---	---	---	---	23,08	23,08
- setInput (int)	66,67	38,00	38,89	75,00	75,00	50,00	75,00	14,29	14,29
- getObjects ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- setEnabled (boolean)	100,00	100,00	---	---	---	---	---	100,00	100,00
- addSelectedIds (SimpleIDKey[], boolean)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- commitIntoDB (int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- disconnectInDB ()	100,00	25,32	18,52	7,50	5,00	0,00	7,50	4,05	3,49

- connectInDB ()	100,00	2,82	3,57	25,00	16,67	0,00	25,00	0,24	0,24
- deleteFromDB ()	100,00	3,85	7,14	50,00	50,00	0,00	50,00	7,14	7,14
- setData (String, String)	100,00	100,00	50,00	50,00	50,00	0,00	50,00	50,00	50,00
- addItemListener (ItemListener)	100,00	100,00	---	---	---	---	---	100,00	100,00
- removeItemListener (ItemListener)	0,00	0,00	---	---	---	---	---	0,00	0,00
- fireItemChanged ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- addUpdateListener ()	100,00	46,15	33,33	---	---	---	---	23,08	23,08
- update ()	0,00	0,00	0,00	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.Booking	52,38	58,95	30,77	26,92	26,92	0,00	26,92	9,40	9,40
- Booking (int, Date, Date, Date, Date, Date, Timestamp, Timestamp)	100,00	100,00	---	---	---	---	---	100,00	100,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createBookingTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	66,67	78,57	50,00	50,00	50,00	0,00	50,00	1,56	1,56
- updateDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- removeFromDB (Iterable)	100,00	84,62	50,00	0,00	0,00	0,00	0,00	33,33	33,33
- getColumns ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getBilled ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setBilled (Date)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getConfirmed ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setConfirmed (Date)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getEnrolled ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setEnrolled (Date)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getMessaged ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setMessaged (Date)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getSignedOff ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setSignedOff (Date)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getId ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- hashCode ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- equals (Object)	66,67	66,67	50,00	50,00	50,00	0,00	50,00	50,00	50,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.util.CalendarControl	37,84	50,00	35,00	22,22	20,00	0,00	22,22	30,30	30,30
- CalendarControl (Composite, int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- onOpen ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- fireSWTCalendarPopupClosed ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- fireSWTCalendarPopupOpened ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setActivated (boolean)	100,00	100,00	100,00	50,00	50,00	0,00	50,00	100,00	100,00
- getDate ()	66,67	66,67	50,00	50,00	50,00	0,00	50,00	50,00	50,00
- setFont (Font)	0,00	0,00	---	---	---	---	---	0,00	0,00
- setDate (Date)	100,00	100,00	100,00	50,00	50,00	0,00	50,00	75,00	75,00
- addSWTCalendarPopuplistener (SWTCalendarPopupListener)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeSWTCalendarPopuplistener (SWTCalendarPopupListener)	0,00	0,00	---	---	---	---	---	0,00	0,00
- addSWTCalendarlistener (SWTCalendarListener)	100,00	100,00	---	---	---	---	---	100,00	100,00
- removeSWTCalendarlistener (SWTCalendarListener)	0,00	0,00	---	---	---	---	---	0,00	0,00
- fireSWTCalendarDatechanged ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setEnabled (boolean)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getOpenPopupButton ()	0,00	0,00	---	---	---	---	---	0,00	0,00

- isPopupOpen ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- closePopup ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setToolTipText (String)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.Client	48,78	52,67	25,00	29,17	29,17	16,67	29,17	29,27	26,83
- Client (int, String, String, String, String, String, String, String, String, String, String, String, String, String, Date, Date, String, Float, int, String, String, Timestamp, Timestamp)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getEmployerDescription ()	40,00	15,38	16,67	25,00	25,00	0,00	25,00	25,00	25,00
- getVector (ResultSet)	100,00	93,75	83,33	100,00	100,00	100,00	100,00	37,50	25,00
- getVectorFromDB (Vector, String)	100,00	100,00	---	---	---	---	---	100,00	100,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createClientTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	100,00	85,71	50,00	50,00	50,00	0,00	50,00	25,00	25,00
- updateDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getNext (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getPrevious (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getColumns ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getEmployerId ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setEmployerId (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getTask ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setTask (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getTurnover ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setTurnover (Float)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getClient (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getEmployees (int)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.ClientBooking	17,02	18,18	6,67	4,55	4,55	0,00	4,55	10,00	10,00
- ClientBooking (int, Date, Date, Date, Date, Date, int, int, int, int, Timestamp, Timestamp)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getSubstituteDescription ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getVector (ResultSet)	100,00	21,05	50,00	50,00	50,00	0,00	50,00	25,00	25,00
- getVectorFromDB (Vector, String)	100,00	100,00	---	---	---	---	---	100,00	100,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createClientBookingTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- updateDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getColumns ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getClientBooking (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getNext (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getPrevious (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getClientId ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- setClientId (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getDebtorId ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- setDebtorId (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getPresentationId ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- setPresentationId (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getBookingsOfClient (int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getBookingsForPresentation (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getSubstituteId ()	0,00	0,00	---	---	---	---	---	0,00	0,00

- setSubstituteId (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
-------------------------	------	------	-----	-----	-----	-----	-----	------	------

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.list.ClientBookingListWindow	38,46	37,14	33,33	50,00	50,00	0,00	50,00	29,41	29,41
- ClientBookingListWindow ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- ClientBookingListWindow (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- ClientBookingListWindow (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createSpecializedMainTableListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedToolBarListeners ()	50,00	100,00	---	---	---	---	---	100,00	100,00
- createChooseButtonListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createChooseButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- openList ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getTableInputFromDB ()	100,00	36,36	33,33	50,00	50,00	0,00	50,00	16,67	16,67
- createWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.provider.ClientBookingTableProvider	19,35	0,00	0,00	0,00	0,00	0,00	0,00	12,00	12,00
- dispose ()	100,00	---	---	---	---	---	---	100,00	100,00
- getColumnImage (Object, int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getColumnText (Object, int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- addListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00
- isLabelProperty (Object, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.ClientBookingWindow	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- ClientBookingWindow (Shell, AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- ClientBookingWindow (Shell, ClientBooking, AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- setInput (ClientBooking)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setInputComponentsEnabled (boolean)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createButtonListener (Composite, AssociationTabControl)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- checkInput ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- commitInputToDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- confirmClose ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- onSave ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.list.ClientListWindow	46,15	40,00	33,33	50,00	50,00	0,00	50,00	35,29	35,29
- ClientListWindow ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- ClientListWindow (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- ClientListWindow (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createSpecializedMainTableListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00

- createSpecializedMainTableListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedToolBarListeners ()	50,00	100,00	---	---	---	---	---	100,00	100,00
- createChooseButtonListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createChooseButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- openList ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getTableInputFromDB ()	100,00	36,36	33,33	50,00	50,00	0,00	50,00	16,67	16,67
- createWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.provider.ClientTableProvider	87,50	91,94	91,18	80,00	80,00	60,00	80,00	88,24	88,24
- dispose ()	100,00	---	---	---	---	---	---	100,00	100,00
- getColumnImage (Object, int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getColumnText (Object, int)	90,00	93,33	91,18	80,00	80,00	60,00	80,00	89,66	89,66
- addListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00
- isLabelProperty (Object, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.ClientWindow	62,96	84,50	51,25	55,88	57,69	17,65	55,22	0,00	0,00
- ClientWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00
- ClientWindow (Shell, Client, AssociationTabControl)	100,00	100,00	---	---	---	---	---	100,00	100,00
- ClientWindow (Shell, Client, ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- setInput (Client)	100,00	83,19	60,00	60,42	61,54	20,83	59,57	0,00	0,00
- setInputComponentsEnabled (boolean)	100,00	100,00	---	---	---	---	---	100,00	100,00
- createButtonListener (AssociationTabControl)	100,00	81,82	50,00	50,00	50,00	0,00	50,00	50,00	50,00
- createButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- checkInput ()	100,00	50,00	50,00	50,00	50,00	0,00	50,00	6,25	6,25
- commitInputToDB ()	100,00	24,32	37,50	50,00	50,00	0,00	50,00	12,50	12,50
- confirmClose ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- onSave ()	75,00	85,71	75,00	75,00	83,33	50,00	75,00	66,67	66,67

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.Company	91,22	84,64	43,33	46,30	46,30	7,41	46,30	6,17	6,17
- Company (int, String, String, String, String, String, String, String, String, String, String, String, String, String, String, String, String, Date, String, String, String, Float, Date, Timestamp, Timestamp)	100,00	100,00	---	---	---	---	---	100,00	100,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createCompanyTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	66,67	86,67	50,00	50,00	50,00	0,00	50,00	0,19	0,19
- updateDB ()	100,00	86,79	50,00	50,00	50,00	0,00	50,00	0,20	0,20
- removeFromDB (Iterable)	100,00	84,62	50,00	50,00	50,00	0,00	50,00	33,33	33,33
- getVector (ResultSet)	100,00	95,12	66,67	75,00	75,00	50,00	75,00	25,00	25,00
- getVectorFromDB (Vector, String)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getColumns ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getNext (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getPrevious (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getCompany (int)	0,00	0,00	---	---	---	---	---	0,00	0,00

(AssociationTabControl)									
- createSpecializedMainTableListener ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createSpecializedMainTableListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener (ForeignKeyConstraintSelector)	100,00	100,00	---	---	---	---	---	100,00	100,00
- createSpecializedToolBarListeners ()	50,00	100,00	---	---	---	---	---	100,00	100,00
- createChooseButtonListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createChooseButtonListener (ForeignKeyConstraintSelector)	100,00	100,00	---	---	---	---	---	100,00	100,00
- openList ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getTableInputFromDB ()	100,00	33,33	33,33	50,00	50,00	0,00	50,00	16,67	16,67
- createWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.provider.CompanyInternalPresentationTableProvider	71,43	85,71	77,78	66,67	66,67	33,33	66,67	66,67	66,67
- getColumnText (Object, int)	71,43	85,71	77,78	66,67	66,67	33,33	66,67	66,67	66,67

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.CompanyInternalPresentationWindow	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- CompanyInternalPresentationWindow (Shell)	0,00	0,00	---	---	---	---	---	0,00	0,00
- CompanyInternalPresentationWindow (Shell, CompanyInternalPresentation, AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- CompanyInternalPresentationWindow (Shell, CompanyInternalPresentation, ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- setInput (CompanyInternalPresentation)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setInputComponentsEnabled (boolean)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createButtonListener (AssociationTabControl)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- createButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- checkInput ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- commitInputToDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- confirmClose ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- onSave ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.list.CompanyListWindow	50,00	40,00	33,33	50,00	50,00	0,00	50,00	35,29	35,29
- CompanyListWindow ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- CompanyListWindow (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- CompanyListWindow (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createSpecializedMainTableListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedToolBarListeners ()	75,00	100,00	---	---	---	---	---	100,00	100,00
- createChooseButtonListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createChooseButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- openList ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getTableInputFromDB ()	100,00	36,36	33,33	50,00	50,00	0,00	50,00	16,67	16,67

- createWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00
------------------------	--------	--------	-----	-----	-----	-----	-----	--------	--------

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.provider.CompanyTableProvider	83,67	90,00	86,05	50,00	50,00	0,00	50,00	83,72	83,72
- dispose ()	100,00	---	---	---	---	---	---	100,00	100,00
- getColumnImage (Object, int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getColumnText (Object, int)	84,62	91,03	86,05	50,00	50,00	0,00	50,00	84,21	84,21
- addListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00
- isLabelProperty (Object, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.CompanyWindow	82,61	91,39	56,48	58,51	59,62	17,02	58,06	0,00	0,00
- CompanyWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00
- CompanyWindow (Shell, Company, ForeignKeyConstraintSelector)	100,00	100,00	---	---	---	---	---	100,00	100,00
- setInput (Company)	100,00	80,26	56,94	57,14	58,11	14,29	56,52	0,00	0,00
- setInputComponentsEnabled (boolean)	100,00	100,00	---	---	---	---	---	100,00	100,00
- createButtonListener (ForeignKeyConstraintSelector)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00
- checkInput ()	100,00	43,48	50,00	50,00	50,00	0,00	50,00	0,78	0,78
- commitInputToDB ()	100,00	84,00	60,00	100,00	100,00	100,00	100,00	12,50	12,50
- confirmClose ()	40,00	46,15	33,33	50,00	50,00	0,00	50,00	12,50	12,50
- onSave ()	75,00	85,71	75,00	75,00	83,33	50,00	75,00	66,67	66,67

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.Conduct	0,00	0,00	---	---	---	---	---	0,00	0,00
- Conduct (int, int, Timestamp, Timestamp)	0,00	0,00	---	---	---	---	---	0,00	0,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createConductTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- deleteFromDB ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getColumns ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getLecturerId ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getSeminarytypeId ()	0,00	0,00	---	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.util.Configuration	40,00	19,23	11,11	0,00	0,00	0,00	0,00	20,00	20,00
- Configuration (boolean)	100,00	36,36	20,00	0,00	0,00	0,00	0,00	25,00	25,00
- saveConfiguration ()	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- makeStdConfiguration (boolean)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setProperty (String, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getProperty (String)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.util.ConfigureListDlg	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- ConfigureListDlg (Shell, TableColumn[], int[])	0,00	0,00	---	---	---	---	---	0,00	0,00
- createButtonsForButtonBar (Composite)	0,00	0,00	---	---	---	---	---	0,00	0,00
- buttonPressed (int)	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- okPressed ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- cancelPressed ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createDialogArea (Composite)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getColumnProperties ()	0,00	0,00	---	---	---	---	---	0,00	0,00

- addListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00
- dispose ()	100,00	---	---	---	---	---	---	100,00	100,00
- isLabelProperty (Object, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg_gui_util.DelayedPaymentViewerControl	66,67	76,74	50,00	62,50	62,50	50,00	62,50	26,67	26,67
- DelayedPaymentViewerControl (Composite, Shell)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	33,33	33,33
- setInput (int)	66,67	60,00	50,00	50,00	50,00	33,33	50,00	22,22	22,22
- addUpdateListener (int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- update ()	0,00	0,00	0,00	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg_sql_util.DistinctVector	85,71	66,67	50,00	50,00	50,00	0,00	50,00	75,00	75,00
- DistinctVector ()	100,00	---	---	---	---	---	---	100,00	100,00
- DistinctVector (Collection)	100,00	---	---	---	---	---	---	100,00	100,00
- add (E)	66,67	66,67	50,00	50,00	50,00	0,00	50,00	50,00	50,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg_gui_util.EditorManager	0,00	0,00	---	---	---	---	---	0,00	0,00
- addEditingId (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeEditingId (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- isEditing (int)	0,00	0,00	---	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg_gui_provider.EnumDlg	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- EnumDlg (Shell, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createButtonsForButtonBar (Composite)	0,00	0,00	---	---	---	---	---	0,00	0,00
- buttonPressed (int)	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- okPressed ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- cancelPressed ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createDialogArea (Composite)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getNewValue ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- configureShell (Shell)	0,00	0,00	---	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg_sql_tables.Enumeration	54,05	33,01	23,33	33,33	33,33	33,33	33,33	18,57	7,43
- Enumeration (int, String)	100,00	100,00	---	---	---	---	---	100,00	100,00
- createEnumerationTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getVectorFromDB (int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getVector (ResultSet)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	66,67	33,33
- getValue ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getType ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createNewEnumInDB (int, String)	100,00	100,00	---	---	---	---	---	100,00	100,00
- removeFromDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setValueInDB (String)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- addEnums (Combo, int)	100,00	83,33	75,00	100,00	100,00	100,00	100,00	50,00	25,00
- addSalutationEnums (Combo)	100,00	100,00	---	---	---	---	---	100,00	100,00
- addTitleEnums (Combo)	100,00	100,00	---	---	---	---	---	100,00	100,00
- addCountryEnums (Combo)	100,00	100,00	---	---	---	---	---	100,00	100,00
- addAnnexEnums (Combo)	100,00	100,00	---	---	---	---	---	100,00	100,00

- insertSalutationStringInDB (String)	100,00	100,00	100,00	---	---	---	---	100,00	100,00
- insertTitleStringInDB (String)	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- insertcountryStringInDB (String)	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- insertAnnexStringInDB (String)	0,00	0,00	0,00	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.EnumerationWindow	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- EnumerationWindow (Shell)	0,00	0,00	---	---	---	---	---	0,00	0,00
- onDelete (Enumeration)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- onNew ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- onEdit (Enumeration)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.util.ExtensibleSearchControl	42,86	61,59	34,38	34,38	34,85	25,00	34,38	5,08	2,75
- ExtensibleSearchControl (Composite, int)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	66,67	33,33
- createFilterExtension (int)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00
- addExtensionListener (SelectionListener)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	66,67	33,33
- addModifyListener (ModifyListener)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getConstraints ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setColumns (Vector)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	55,56	4,76
- getComparatorString (String)	33,33	50,00	42,86	42,86	42,86	14,29	42,86	25,00	25,00
- getDBColumn (String)	100,00	100,00	50,00	50,00	50,00	0,00	50,00	50,00	50,00
- setData (String, String)	100,00	100,00	75,00	75,00	75,00	50,00	75,00	50,00	25,00
- getConjunction (String)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getRelation (String)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.util.ForeignKeyConstraintSelector	41,18	44,30	21,05	35,71	35,71	28,57	35,71	24,39	24,39
- ForeignKeyConstraintSelector (Composite, int)	0,00	---	---	---	---	---	---	0,00	0,00
- ForeignKeyConstraintSelector (Composite, int, boolean)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00
- updateText ()	100,00	44,74	55,56	100,00	100,00	100,00	100,00	37,50	37,50
- setSelectedId (int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- addModifyListener (ModifyListener)	100,00	100,00	---	---	---	---	---	100,00	100,00
- setEnabled (boolean)	100,00	100,00	---	---	---	---	---	100,00	100,00
- setData (String, String)	100,00	100,00	50,00	50,00	50,00	0,00	50,00	50,00	50,00
- isLockedId (int)	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- onDelete ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- determineDeleteMessage ()	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- deleteFromDB ()	0,00	0,00	0,00	---	---	---	---	0,00	0,00
- getSelectedId ()	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.Instructor	0,00	0,00	---	---	---	---	---	0,00	0,00
- Instructor (int, int, Timestamp, Timestamp)	0,00	0,00	---	---	---	---	---	0,00	0,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createInstructorTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- deleteFromDB ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getColumns ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getLecturerId ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getPresentationId ()	0,00	0,00	---	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.util.ItemListener	---	---	---	---	---	---	---	---	---

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.util.KeyPair	0,00	0,00	---	---	---	---	---	0,00	0,00
- KeyPair (int, int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- equals (KeyPair)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getKey1 ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getKey2 ()	0,00	0,00	---	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.Lecturer	36,84	37,98	25,00	27,27	27,27	27,27	27,27	17,31	13,46
- Lecturer (int, String, String, String, String, String, String, String, String, String, String, String, String, Date, Date, String, Float, String, String, Timestamp, Timestamp)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getVector (ResultSet)	100,00	94,12	87,50	100,00	100,00	100,00	100,00	13,64	4,55
- getVectorFromDB (Vector, String)	100,00	100,00	---	---	---	---	---	100,00	100,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createLecturerTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- updateDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getColumns ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getBio ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setBio (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getDailyFee ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setDailyFee (Float)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getHourlyFee ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setHourlyFee (Float)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getNext (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getPrevious (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getLectInstructing (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getLectSupervising (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getLectConducting (int)	0,00	0,00	---	---	---	---	---	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.list.LecturerListWindow	38,46	37,14	33,33	50,00	50,00	0,00	50,00	29,41	29,41
- LecturerListWindow ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- LecturerListWindow (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- LecturerListWindow (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createSpecializedMainTableListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedToolBarListeners ()	50,00	100,00	---	---	---	---	---	100,00	100,00
- createChooseButtonListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createChooseButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- openList ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getTableInputFromDB ()	100,00	36,36	33,33	50,00	50,00	0,00	50,00	16,67	16,67
- createWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.provider.LecturerTableProvider	87,80	92,19	91,67	83,33	83,33	66,67	83,33	88,57	88,57
- dispose ()	100,00	---	---	---	---	---	---	100,00	100,00
- getColumnImage (Object, int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getColumnText (Object, int)	90,32	93,55	91,67	83,33	83,33	66,67	83,33	90,00	90,00
- addListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00
- isLabelProperty (Object, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.LecturerWindow	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- LecturerWindow (Shell, AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- LecturerWindow (Shell, Lecturer, AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- setInput (Lecturer)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setInputComponentsEnabled (boolean)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createButtonListener (Composite, AssociationTabControl)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- checkInput ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- commitInputToDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- confirmClose ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- onSave ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.list.ListWindow	80,00	85,83	100,00	100,00	100,00	100,00	100,00	66,67	60,00
- ListWindow (Shell, Vector)	100,00	100,00	---	---	---	---	---	100,00	100,00
- ListWindow (Shell, Vector, AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- ListWindow (Shell, Vector, ForeignKeyConstraintSelector)	100,00	100,00	---	---	---	---	---	100,00	100,00
- createButtonArea ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setActive ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- setVisible (boolean)	100,00	100,00	---	---	---	---	---	100,00	100,00
- update ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createListToolBarBasics (boolean)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	50,00	50,00
- createTableBasics ()	100,00	100,00	100,00	100,00	100,00	100,00	100,00	66,67	33,33
- addUpdateListener (int)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.MainWindow	85,71	98,05	83,33	75,00	78,57	50,00	71,43	71,43	42,86
- MainWindow (Display)	100,00	100,00	100,00	100,00	100,00	100,00	100,00	71,43	14,29
- createListWindows ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- centerShell ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createStatusBar ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createMenu ()	100,00	99,17	50,00	50,00	50,00	0,00	33,33	50,00	50,00
- getShell ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getInstance ()	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.util.Messages	50,00	83,33	75,00	50,00	50,00	0,00	50,00	50,00	50,00
- Messages ()	0,00	---	---	---	---	---	---	0,00	0,00
- getString (String)	75,00	83,33	75,00	50,00	50,00	0,00	50,00	66,67	66,67

Class semorg.sql.tables.Presentation	36,19	25,19	6,67	3,70	3,70	0,00	3,70	1,56	1,56
- Presentation (int, int, Integer, Date, Date, Time, Time, Time, Time, String, String, String, String, String, String, boolean, Timestamp, Timestamp)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getSeminarTypeDescription ()	50,00	70,00	50,00	50,00	50,00	0,00	50,00	33,33	33,33
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createPresentationTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00
- insertIntoDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- updateDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- removeFromDB (Iterable)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getColumns ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- getVector (ResultSet)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getPresSupervisedBy (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getPresInstructedBy (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getPresForSemType (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getPresentationVectorFromDB (Vector, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getPresentation (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getColumnsWithSemtype ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- hashCode ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- equals (Object)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getShortDescription ()	100,00	87,50	50,00	50,00	50,00	0,00	50,00	25,00	25,00
- getSeminarTypeForPresId (int)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- getAnnex ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setAnnex (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getBeginningDay ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setBeginningDay (Date)	0,00	0,00	---	---	---	---	---	0,00	0,00
- isCancelled ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setCancelled (boolean)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getCity ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setCity (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getCountry ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setCountry (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getDuration ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setDuration (Integer)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getEndingDay ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setEndingDay (Date)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getEndingTime ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setEndingTime (Time)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getFirstStartingTime ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setFirstStartingTime (Time)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getLastEndingTime ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setLastEndingTime (Time)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getLocation ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setLocation (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getSeminarTypeId ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- setSeminarTypeId (int)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getStartingTime ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setStartingTime (Time)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getStreet ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- setStreet (String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- getZipCode ()	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.list.SeminarTypeListWindow	38,46	37,14	33,33	50,00	50,00	0,00	50,00	29,41	29,41
- SeminarTypeListWindow ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- SeminarTypeListWindow (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- SeminarTypeListWindow (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener ()	100,00	100,00	---	---	---	---	---	100,00	100,00
- createSpecializedMainTableListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedMainTableListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSpecializedToolBarListeners ()	50,00	100,00	---	---	---	---	---	100,00	100,00
- createChooseButtonListener (AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createChooseButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- openList ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- getTableInputFromDB ()	100,00	36,36	33,33	50,00	50,00	0,00	50,00	16,67	16,67
- createWindow (Shell)	100,00	100,00	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.provider.SeminarTypeTableProvider	73,53	82,00	75,86	50,00	50,00	0,00	50,00	71,43	71,43
- dispose ()	100,00	---	---	---	---	---	---	100,00	100,00
- getColumnImage (Object, int)	100,00	100,00	---	---	---	---	---	100,00	100,00
- getColumnText (Object, int)	70,83	83,33	75,86	50,00	50,00	0,00	50,00	69,57	69,57
- addListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00
- isLabelProperty (Object, String)	0,00	0,00	---	---	---	---	---	0,00	0,00
- removeListener (ILabelProviderListener)	100,00	---	---	---	---	---	---	100,00	100,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.gui.SeminarTypeWindow	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- SeminarTypeWindow (Shell)	0,00	0,00	---	---	---	---	---	0,00	0,00
- SeminarTypeWindow (Shell, SeminarType, AssociationTabControl)	0,00	0,00	---	---	---	---	---	0,00	0,00
- SeminarTypeWindow (Shell, SeminarType, ForeignKeyConstraintSelector)	0,00	0,00	---	---	---	---	---	0,00	0,00
- setInput (SeminarType)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- setInputComponentsEnabled (boolean)	0,00	0,00	---	---	---	---	---	0,00	0,00
- createButtonListener (AssociationTabControl)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- createButtonListener (ForeignKeyConstraintSelector)	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- checkInput ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- commitInputToDB ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- confirmClose ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
- onSave ()	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.SimpleIDKey	---	---	---	---	---	---	---	---	---

top	FEEC	C0	C1	C2	MMDC	MCDC	C3	ModBI	BI
Class semorg.sql.tables.Supervisor	0,00	0,00	---	---	---	---	---	0,00	0,00
- Supervisor (int, int, Timestamp, Timestamp)	0,00	0,00	---	---	---	---	---	0,00	0,00
- tableOK ()	0,00	0,00	---	---	---	---	---	0,00	0,00
- createSupervisorTable (Statement)	0,00	0,00	---	---	---	---	---	0,00	0,00

