Kalimetrix **LOGISCOPE**

RuleChecker & QualityChecker
Getting Started

Before using this information, be sure to read the general information under the “Notices” section, on page 81.

© **Copyright Kalimetrix 2014**

About this manual

Audience

With Kalimetrix Logiscope™ *RuleChecker & QualityChecker,* you are about to discover the world of most advanced software product evaluation techniques.
The *RuleChecker & QualityChecker Getting Started* manual will lead you through a use-case situation and show you just how simple and yet complete the Logiscope toolset is. If you can spare a little of your precious time, just relax and let us take you for a guided tour of Logiscope *RuleChecker & QualityChecker.*

Overview

This manual introduces Logiscope *RuleChecker & QualityChecker* environment and will get you started. In about two hours you will become familiar with the tool main features. By the end of this phase, you will know how to use Logiscope *RuleChecker & QualityChecker* main features and become familiar with the various commands. In the process, you will see and learn:

- how to create a Logiscope project and activate Logiscope powerful source code verification modules:
  - *RuleChecker*: check that the source code complies with a set of defined coding standards and best practices leading to a satisfactory level of Maintainability, Reliability or Portability;
  - *QualityChecker*: use source code metrics to locate complex, error prone modules, analyse graphic results to assess architecture of the application and detailed design of functions.

  For more details, refer to *Kalimetrix Logiscope - Basic Concepts*;
- how to display some first results using Logiscope **Studio**:
  - Rule violations in source code ,
  - Rating of functions, classes, modules, etc. and values of complexity metrics,
- how to display graphic results using Logiscope **Viewer**:
  - Control graphs,
  - Call graphs,
  - Use graphs.
  - Inheritance graphs,

Related Documents

Reading first the following manual is highly recommended:

- *Kalimetrix Logiscope - Basic Concepts*.

Additional information can be found in:

- *Kalimetrix Logiscope - QualityChecker & RuleChecker - Ada Reference Manual*.
- *Kalimetrix Logiscope - QualityChecker & RuleChecker - C Reference Manual*.
- *Kalimetrix Logiscope - QualityChecker & RuleChecker - C++ Reference Manual*.
- *Kalimetrix Logiscope - QualityChecker & RuleChecker - Java Reference Manual*.

Before You Start

In this session, you will use examples of source code files provided in the **samples** folder of the Logiscope installation directory.

As a precaution to keep original files safe, it is recommended to copy the samples subdirectory into a working directory of your own.

In addition, you will create Logiscope projects and associated repositories: i.e. sets of files containing internal data used by Logiscope. It is recommended to a create a dedicated directory to store these data: e.g. a folder named **LogiscopeProjects**.

Conventions

The following writing conventions are used in this manual:

- **bold**: names of commands (e.g. **vcs**), files and folders (e.g. **LogiscopeProjects**), and file extensions (**.res**)
- *italic*: names of user-defined textual elements (*version_1, component_2*), notes,
- `typewriter`: screen messages (`Reference filename`) requiring user action,
- keycaps (<Enter>).

<InstallationDir> will now refer as the Logiscope installation directory.

<Version> will now refer as the Logiscope current version: e.g. 6.6 or upper.

<Language> will now refer as Ada, C, C++ or Java.

Note: Screen displays in this manual can be slightly different from those you get when running the Getting Started.

Table of Contents

Chapter 1

A Studio Guided Tour

To get access to the Logiscope key features and results, it is first necessary to specify the scope and type of the verification to be performed. This is done using a Logiscope Project.

A Logiscope project mainly consists in:

- the list of source files to be analyzed,
- applicable source code parsing options according to the compilation environment,
- the verification modules to be activated on the source code files and the associated controls and settings (e.g. metrics to be computed, rules to be checked).

A Logiscope project can be created using:

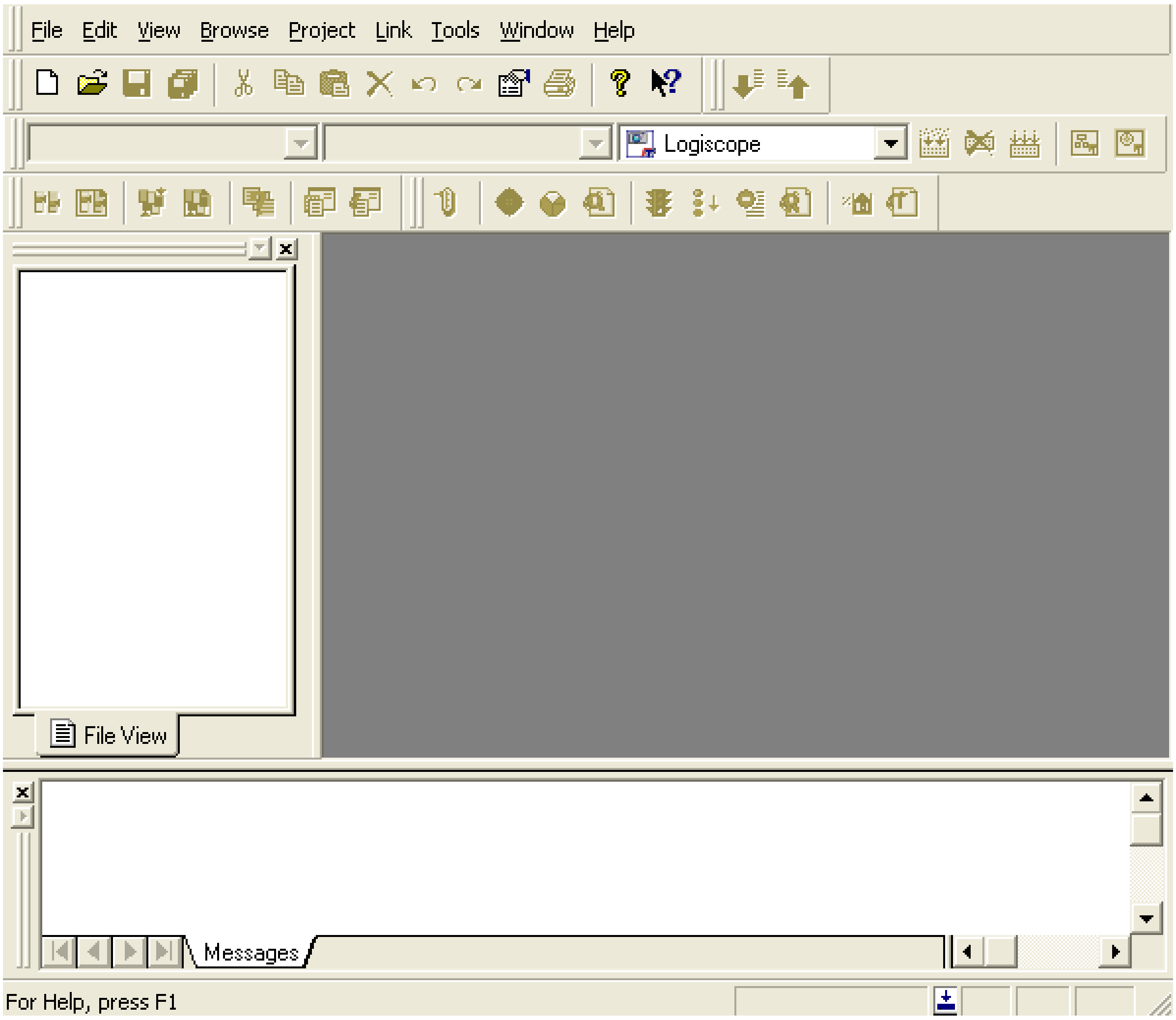
- **Logiscope Studio:** a graphical interface requiring a user interaction, as described in the following sub-sections introducing the Logiscope project settings,
- **Logiscope Command Line mode:** a tool to be used from a standalone command line or within makefiles, please refer to Chapter *Command Line Mode* in the appropriate *Kalimetrix Logiscope - QualityChecker & RuleChecker - <Language> Reference Manual*

1.1 Starting a Logiscope Studio Session

To begin a Logiscope **Studio** session:

- On UNIX (i.e. Solaris or Linux):
  - launch the **vcs** binary
- On Windows:
  - click the **Start** button and select the **Kalimetrix Logiscope 2014** item in the **Kalimetrix** Programs Group.

The *Rational Logiscope* splash screen is first displayed and then the Logiscope **Studio** main window appears as follows.

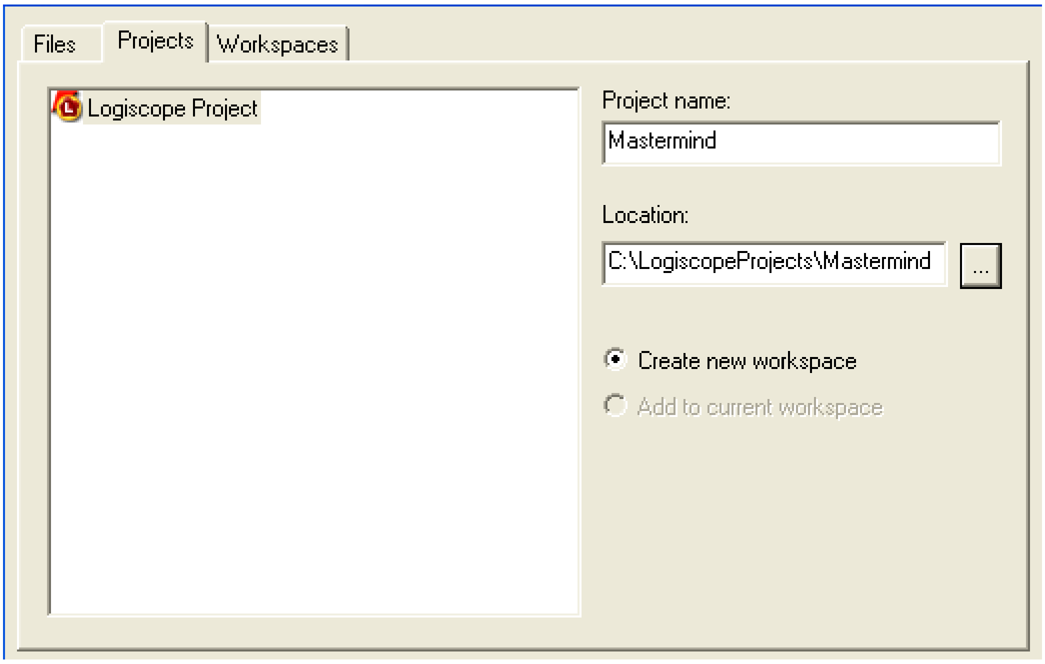


The various components and areas of the *Logiscope* **Studio** main window are detailed in section 1.3.

1.2 Creating a Logiscope Project

First, you shall define the Logiscope project specifying the scope and type of verification to be performed by Logiscope:

1. In the **File** menu, select the **New**... command or click on the 🗋 icon, you get the New Projects dialog box.
2. In the **Project name:** pane, enter the name for the new Logiscope project to be created.
   In the context of the guided tour, type *Mastermind*.
3. Then select its **Location:** i.e. the directory where the Logiscope project (i.e. a "**.ttp**" file) and the associated Logiscope repository will be created; the Logiscope repository is a folder in which Logiscope internal analysis result files are generated.
   You can either enter a directory, keep the proposed default location or specify the path to the **LogiscopeProjects** folder if previously created as recommended.



Project name: Mastermind

Location: C:\LogiscopeProjects\Mastermind

● Create new workspace

○ Add to current workspace

Note: By default, the project name is automatically added to the specified location. This implies that a subdirectory named <ProjectName> is automatically created.

4. and click the **OK** button.

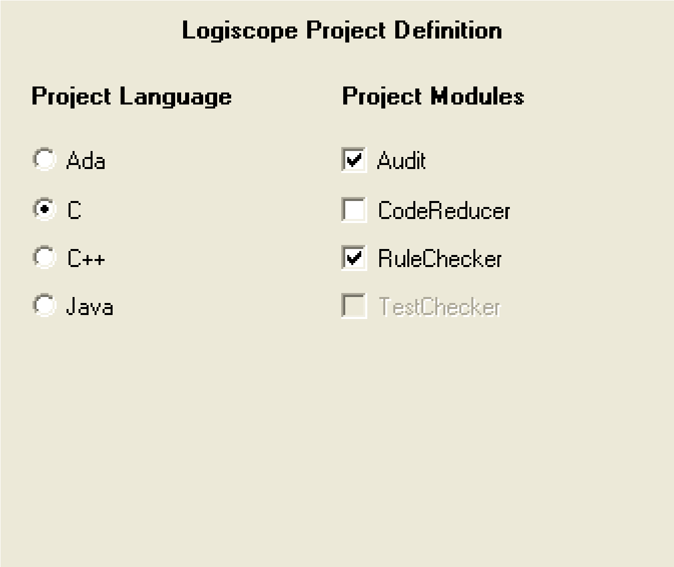
Defining the type of the Logiscope project

The **Logiscope Project Definition** dialog box appears.

5. Select the **Project Language:** i.e. the programming language in which are written the source code files to be analyzed.
For the Mastermind project, select **C**.

Note: Only one language can be selected. If your application contains source code files written in several languages, you should create several distinct Logiscope projects: one for each language.

6. Select the **Project Modules:** i.e. the verification modules to be activated on the source files of the project.
For this guided tour, select both **QualityChecker** and **RuleChecker**.



Logiscope Project Definition

Project Language

○ Ada
● C
○ C++
○ Java

Project Modules

☑ Audit
☐ CodeReducer
☑ RuleChecker
☐ TestChecker

Notes: At least one module should be selected. The *TestChecker* module cannot be selected with another module.

For more details on the *CodeReducer* module, please refer to *Kalimetrix Logiscope - CodeReducer - Identifying Code Similarities.*
For more details on *the TestChecker* module, please refer to *Kalimetrix Logiscope - TestChecker Getting Started.*

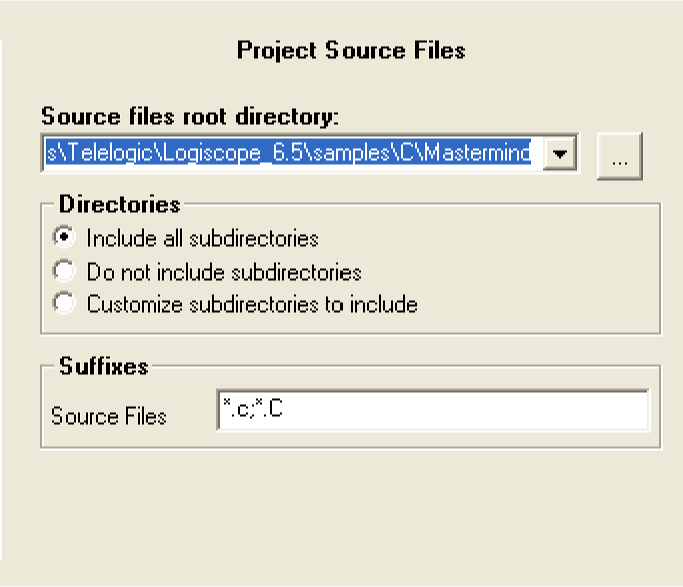
7. Click the **Next** button to continue the creation.

Specifying the source files to be analyzed

The **Project Source Files** dialog box allows to specify what source files are to be analyzed and where they are located.

The **Source files root directory** contains the location of the directory of the source files to be analyzed.

8. Browse to select the directory where the Mastermind sample source files are: i.e. in the **samples/C/Mastermind** folder of the Logiscope installation directory.



Project Source Files

Source files root directory: s\Telelogic\Logiscope_6.5\samples\C\Mastermind

Directories

● Include all subdirectories

○ Do not include subdirectories

○ Customize subdirectories to include

Suffixes

Source Files: *.c;*.C

The **Directories** choice allows to select the list of repertories covering the application source files.

- **Include all subdirectories** means that selected files will be searched for in every subdirectory of the source file root directory.
- **Do not include subdirectories** means that only files included in the application directory will be selected.
- **Customize subdirectories to include** allows the user to select the list of directories that include application files through a new page.

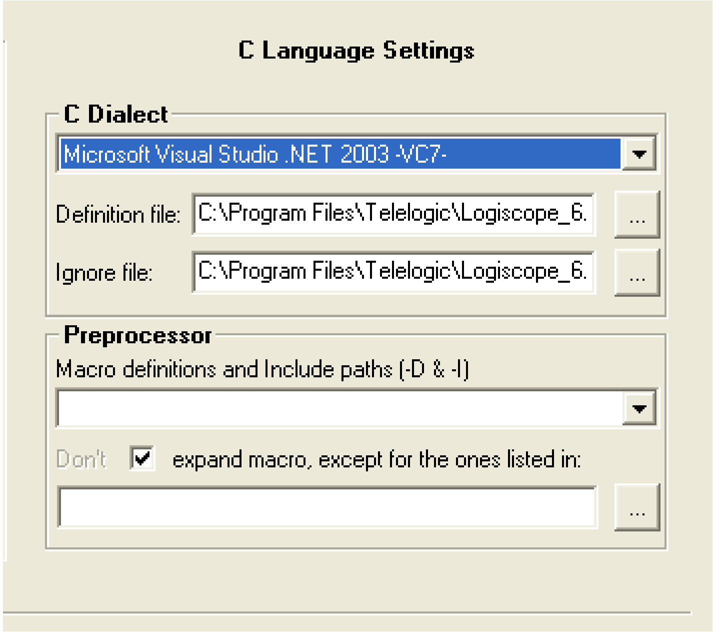
The **Suffixes** choice allows to specify applicable source, header and inline file extensions needed in the above selected directories. Extensions shall be separated with a semi-colon.

9. Click the **Next** button.

Setting Parsing Options

The next dialog box depends upon the selected language for the Logiscope project. This dialog box allows to set up some source code parsing options.

Regarding a Logiscope C project, the options are the following:



C Language Settings

C Dialect

Microsoft Visual Studio .NET 2003 -VC7-

Definition file: C:\Program Files\Telelogic\Logiscope_6.

Ignore file: C:\Program Files\Telelogic\Logiscope_6.

Preprocessor

Macro definitions and Include paths (-D & -I)

Don't ☑ expand macro, except for the ones listed in:

C Dialect: A dialect is used to take into account some special features of the development environment (e.g. compilers, IDE) to use for the project under analysis:

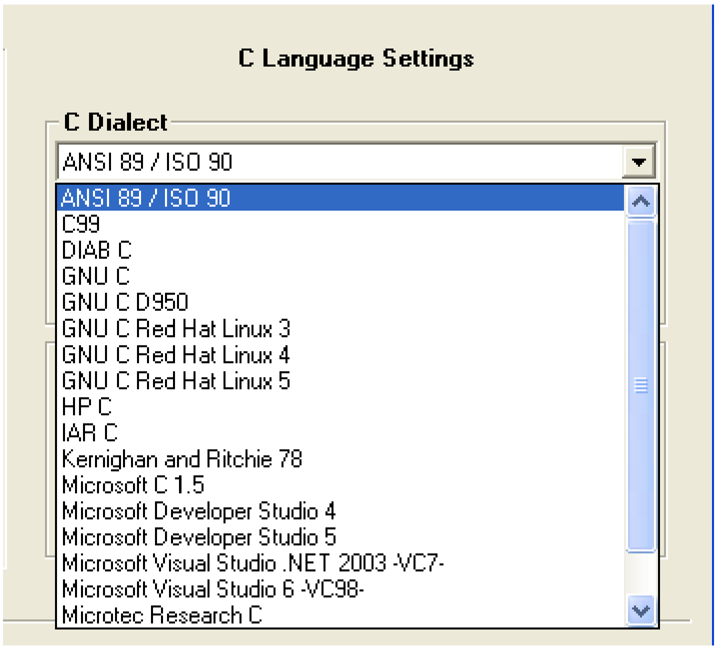
- access paths to standard inclusion directories,
- predefined macro definitions or special keywords.
- inclusion directories where rule violations shall not be reported.

Notes: There is no Language Settings dialog box for Logiscope Ada projects and Logiscope Java projects.

For more details on available C and C++ dialects, please refer to the chapter *Parsing Options* in the respective manuals:

- *Kalimetrix Logiscope RuleChecker & QualityChecker - C Reference Manual.*,
- *Kalimetrix Logiscope RuleChecker & QualityChecker - C++ Reference Manual.*

10.For the Mastermind project: select the standard **ANSI 89 / ISO 90**.



C Language Settings

C Dialect: ANSI 89 / ISO 90

Preprocessor: The source code files to be analyzed may contain some preprocessing directives (e.g. `#ifdef`). In some cases, these directives can lead to parsing errors and warnings by breaking up the code structure or missing information.

In addition to the predefined preprocessing information associated to the selected dialect, the **Preprocessor** pane can be used to provide complementary preprocessing and compilation options:

- access paths to project specific inclusion directories,
- project macro definitions.

The syntax is as for a compiler:

[-I*directory***]***

[-D*name_of_macro1_with_no_argument***[=***definition***]]* [-**

U*name_of_macro2_with_no_argument***[=***definition***]]***

A "**-I**" option defines *directory* as an access path to inclusion directories (applicable to C projects only).

A "**-D**" option defines *name_of_macro1_with_no_argument* as if it were in a `#define` directive.

A "**-U**" option considers *name_of_macro2_with_no_argument* as undefined as if it were part of an `#undef` directive.

The number of occurrences of options is unlimited.

For more details on parsing options, please refer to the corresponding chapter in the respective *Kalimetrix Logiscope - RuleChecker & QualityChecker - <Language> Reference Manual.*

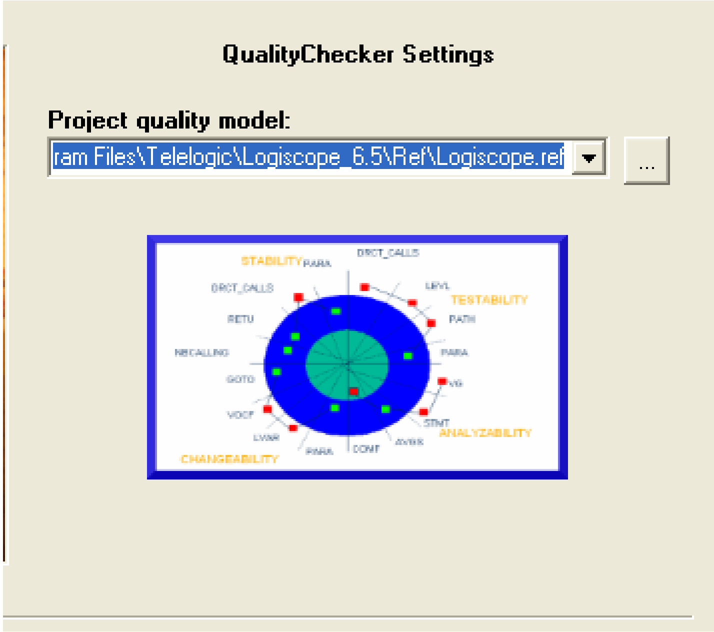
11. Click the **Next** button.

Setting QualityChecker Parameters

The next dialog box allows to specify the applicable **Project quality model:** how the *QualityChecker* module evaluates software quality characteristics (e.g. Maintainability) based on a standard factors / criteria / metrics approach.

Note: Quality models are textual files (suffixed by "**.ref**"). Default quality models are provided with the standard Logiscope installation. They should be customized to take into account the verification objectives and contexts applicable to the project.

For more information, refer to the *Kalimetrix Logiscope Basic Concepts* manual.



QualityChecker Settings

Project quality model: ram Files\Telelogic\Logiscope_6.5\Ref\Logiscope.ref

12. For the purpose of this presentation, keep the default Logiscope quality model provided with the standard Logiscope installation.

Note: For your project verification, you should define and select your own applicable quality model.

13. Click the **Next** button.

Setting RuleChecker Parameters

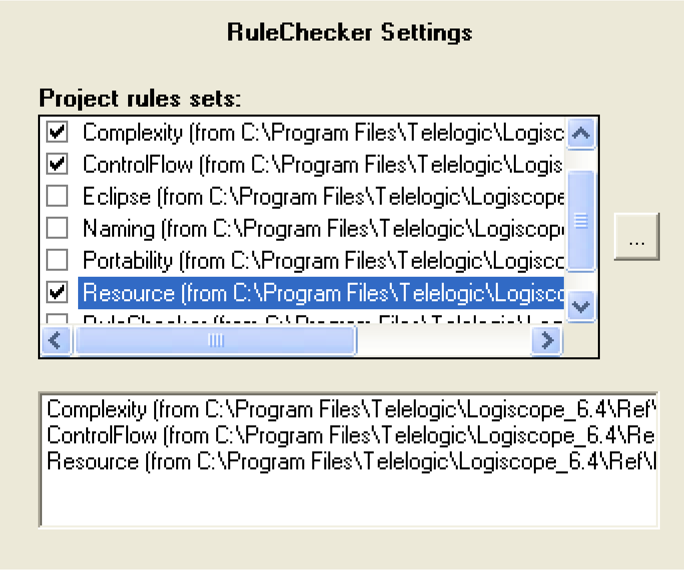
The **RuleChecker Settings** dialog box allows to specify the applicable **Project rule sets:** i.e. the rules / coding standards the *Logiscope RuleChecker* module shall verify on the project source files.

For more details on available rules and rule sets, please refer to the chapter *Standard Programming Rules* in the respective *Kalimetrix Logiscope RuleChecker & QualityChecker <Language> Reference Manual.*

14. Tick the box associated to the following **Project rule sets**:

- Complexity,
- ControlFlow, and
- Resource.

The files now appear in the bottom pane where all currently selected rule sets are listed.



RuleChecker Settings

Project rules sets:

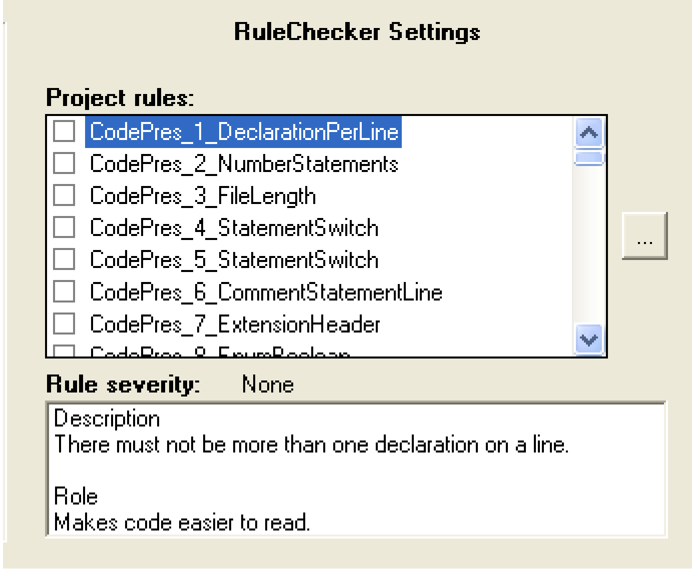
☑ Complexity (from C:\Program Files\Telelogic\Logisc
☑ ControlFlow (from C:\Program Files\Telelogic\Logis
□ Eclipse (from C:\Program Files\Telelogic\Logiscope
□ Naming (from C:\Program Files\Telelogic\Logiscop
□ Portability (from C:\Program Files\Telelogic\Logisco
☑ Resource (from C:\Program Files\Telelogic\Logisco

Complexity (from C:\Program Files\Telelogic\Logiscope_6.4\Ref\
ControlFlow (from C:\Program Files\Telelogic\Logiscope_6.4\Re
Resource (from C:\Program Files\Telelogic\Logiscope_6.4\Ref\l

15. Click the **Next** button.

The next **RuleChecker Settings** dialog box allows to fine tune the list of **Project rules.** It is possible to select or unselect some of the rules available.

The rules that are selected are those listed in the Project rule sets selected in the previous **RuleChecker Settings** dialog box.



RuleChecker Settings

Project rules:

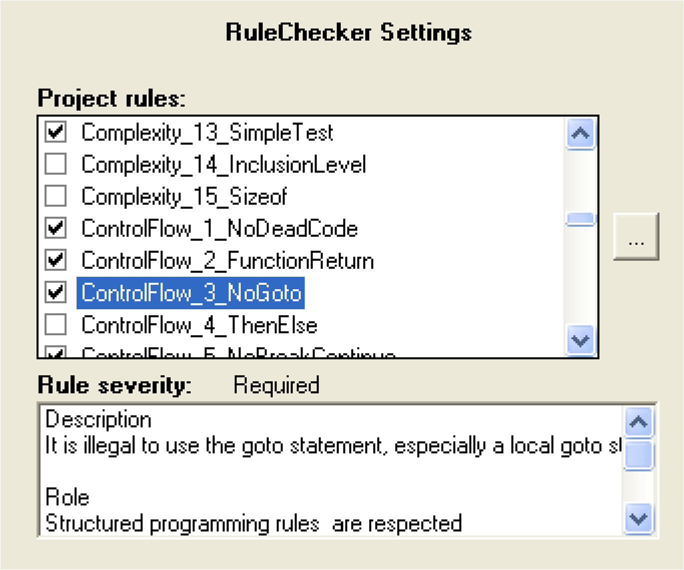
- □ CodePres_1_DeclarationPerLine
- □ CodePres_2_NumberStatements
- □ CodePres_3_FileLength
- □ CodePres_4_StatementSwitch
- □ CodePres_5_StatementSwitch
- □ CodePres_6_CommentStatementLine
- □ CodePres_7_ExtensionHeader
- □ CodePres_8_EnumBoolean

Rule severity: None

Description
There must not be more than one declaration on a line.

Role
Makes code easier to read.

16.For the purpose of the example, just uncheck the rules: **Complexity_14_InclusionLevel and ControlFlow_4_ThenElse.**



RuleChecker Settings

Project rules:

- ☑ Complexity_13_SimpleTest
- □ Complexity_14_InclusionLevel
- □ Complexity_15_Sizeof
- ☑ ControlFlow_1_NoDeadCode
- ☑ ControlFlow_2_FunctionReturn
- ☑ ControlFlow_3_NoGoto
- □ ControlFlow_4_ThenElse
- ☑ ControlFlow_5_NoBreakContinue

Rule severity: Required

Description
It is illegal to use the goto statement, especially a local goto st

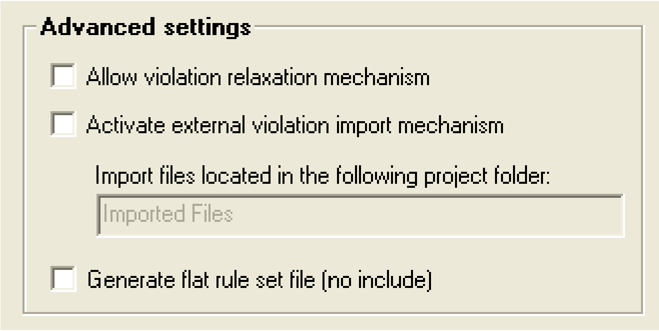
Role
Structured programming rules are respected

The description of the selected rule and the rule severity are displayed in the bottom pane.

17.And click the **Next** button.

The last **RuleChecker Settings** dialog box allows to use some advanced features of the *Logiscope RuleChecker* module.

Advanced Settings:



Advanced settings

□ Allow violation relaxation mechanism

□ Activate external violation import mechanism

Import files located in the following project folder: Imported Files

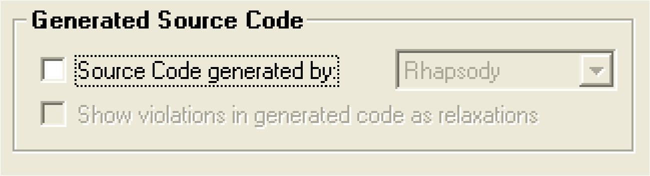
□ Generate flat rule set file (no include)

Allow violation relaxation mechanism: when the box is checked, rule violations can be relaxed using special comments in the code. For more details, please refer to section 1.5.6.

Activate external violation import mechanism: when the box is checked, the files in the specified project folder can be used to import violations generated by an external tool. For more details, please refer to section 1.5.7.

Generate flat rule set file (no include): when the box is checked, the project rule set file (i.e. with a ".rst") extension) that is generated for the project doesn't contain any includes of other rule set files. It will contain an expanded copy of the contents of any rule sets that were used for the project.

Generated Source Code:



Generated Source Code

□ Source Code generated by: Rhapsody

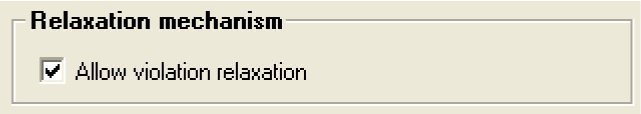
□ Show violations in generated code as relaxations

Source Code generated by: when the box is checked, allows to specify the tool (e.g. *Kalimetrix Rhapsody*) used to generate all or part of the source code under analysis. Thus, *Logiscope RuleChecker* will not considered the violations found in the generated code. For more details, please refer to section 1.5.8.

Show violations in generated code as relaxations: when the box is checked, the violations found in generated code are reported as "relaxations".

For more details on all these options, please refer to the *Kalimetrix Logiscope - Basic Concepts* document.

18. Check the box “Allow violation relaxation mechanism”.



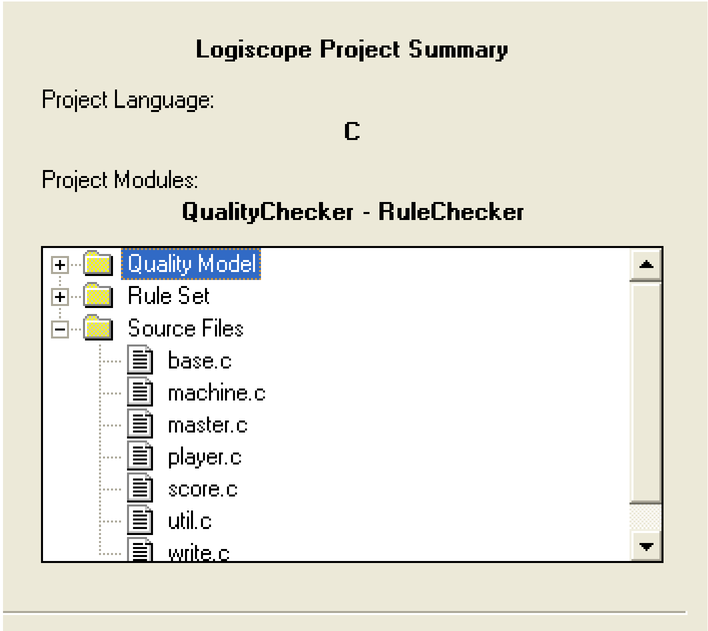
Relaxation mechanism

☑ Allow violation relaxation

19. And then, click the **Next** button.

The creation of the Logiscope project ends by a window summarizing all the project settings.

20. Check if all files are correct by expanding the Source Files folder.
    If the list is not the same as below, this may be because the Source files root directory of the project has not been correctly set. You can use the “Previous” buttons to get back to the corresponding dialog box and change the values.



Logiscope Project Summary

Project Language:

C

Project Modules:

QualityChecker - RuleChecker

- Quality Model
- Rule Set
- Source Files
  - base.c
  - machine.c
  - master.c
  - player.c
  - score.c
  - util.c
  - write.c

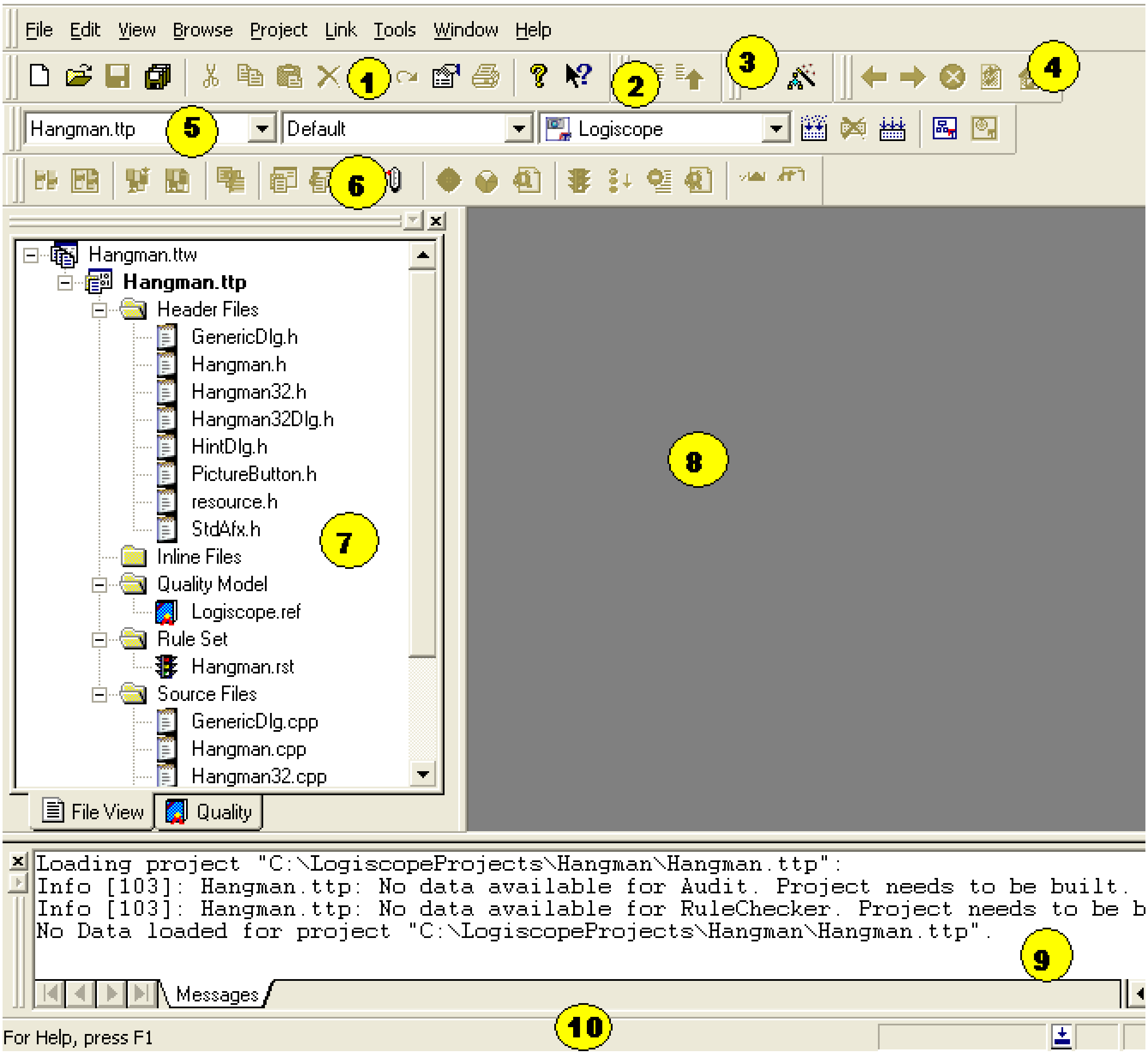
21. Click the **Finish** button.

You can still change the project settings by using the **Settings...** command in the **Project** menu.

You are now ready to “build” the project: i.e. parse the source files of the project to extract all necessary information for code verification. This process is described in section 1.4.

1.3 Introducing the Studio Main Window

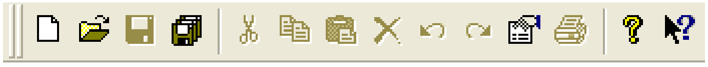
After creating the project, *Logiscope* **Studio** main window looks as follows:



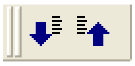
```
Loading project "C:\LogiscopeProjects\Hangman\Hangman.ttp":
Info [103]: Hangman.ttp: No data available for Audit. Project needs to be built.
Info [103]: Hangman.ttp: No data available for RuleChecker. Project needs to be b
No Data loaded for project "C:\LogiscopeProjects\Hangman\Hangman.ttp".
```

According to the default configuration, the *Logiscope* **Studio** main window contains the following components:

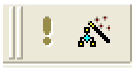
1. **Tool Bar:** provides shortcuts for most commonly used commands of **File** and **Edit** menus.



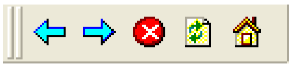
2. **Browse Bar:** provides shortcuts for Browse menu commands.



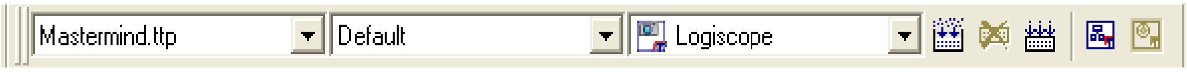
3. **TCL Script Bar:** activates the script wizard: i.e. Logiscope internal data navigator.



4. **Web Browser Bar:** allows navigation in HTML documents and internal data.



5. **Project Bar:** builds the project and starts *Logiscope Viewer* or *TestChecker.*

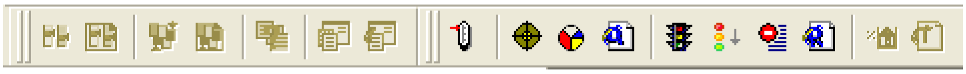


Mastermind.ttp

Default

Logiscope

6. **Logiscope Module Bar:** provides shortcuts for selected results associated, from left to right to *CodeReducer, QualityChecker, RuleChecker* and *TestChecker* modules.



Note that until the project has been "built", all the icons are greyed as no results are yet available.

The *QualityChecker* and *RuleChecker* related icons allow to (from left to right):

- Display the *QualityChecker* **Metrics Dictionary,**
- Display the *QualityChecker* **Criteria** tree,
- Display the *QualityChecker* **Factors** tree,
- Generate the *QualityChecker* HTML **Quality Report,**
- Display the *RuleChecker* **Rule violations,**
- Display the *RuleChecker* **Rule violations by severity,**
- Display the *RuleChecker* **Rule violations list** in the output window,
- Generate the *RuleChecker* HTML **rule violations report.**

7. **Workspace View:** displays a specific view related to the project: header files, the quality model file and source files.
   Double-clicking on any file will display the file contents in the Result Pane.

8. **Result Pane:** is used to display information and result windows.

9. **Output Window:** displays project messages as the first tab is created; also shows "build" messages, and results.

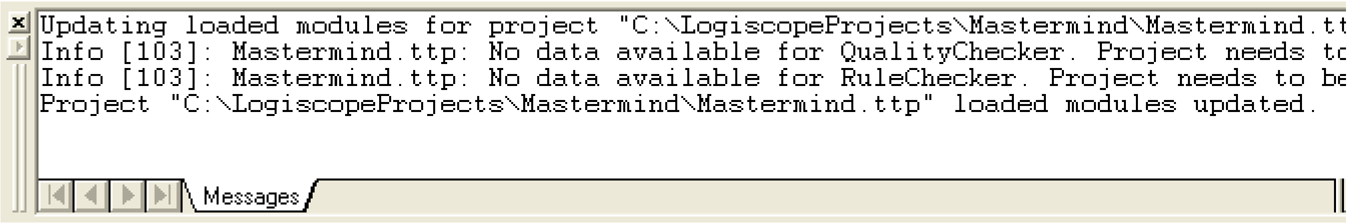
10. **Status Bar**: contains indicators when building and idle. The status bar also shows short definitions corresponding to the toolbars described above.

To add or remove one of these items, just right-click in the corresponding area and check or uncheck the relevant bar.

You can customize the toolbars using the **Tools-Customize...** command.

1.4 Building the Logiscope Project

As stated in the **Messages** tab in the **Output Window**, once the Logiscope project is created, no data is available. Indeed, the source files have not yet been parsed / analyzed by the selected Logiscope verification modules.

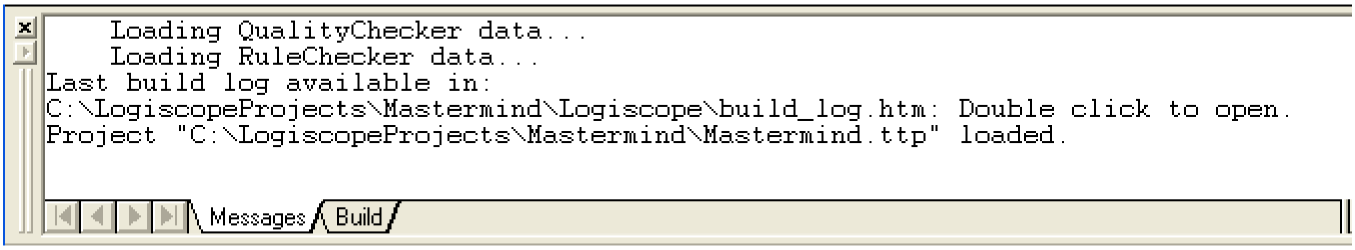


```
Updating loaded modules for project "C:\LogiscopeProjects\Mastermind\Mastermind.tt
Info [103]: Mastermind.ttp: No data available for QualityChecker. Project needs tc
Info [103]: Mastermind.ttp: No data available for RuleChecker. Project needs to be
Project "C:\LogiscopeProjects\Mastermind\Mastermind.ttp" loaded modules updated.
```

1. To get the verification results, select the **Project-Build** command or click on the icon.

A new **Build** tab is added in the **Output Window** next to **Messages**. Several messages are displayed while parsing the source files and then loading the data showing that the build process is in progress.

As soon as the `Project [...] loaded.` message is displayed in the **Messages** tab, the project is built i.e. all the source files have been analyzed and associated results generated and loaded.



```
    Loading QualityChecker data...
    Loading RuleChecker data...
Last build log available in:
C:\LogiscopeProjects\Mastermind\Logiscope\build_log.htm: Double click to open.
Project "C:\LogiscopeProjects\Mastermind\Mastermind.ttp" loaded.
```

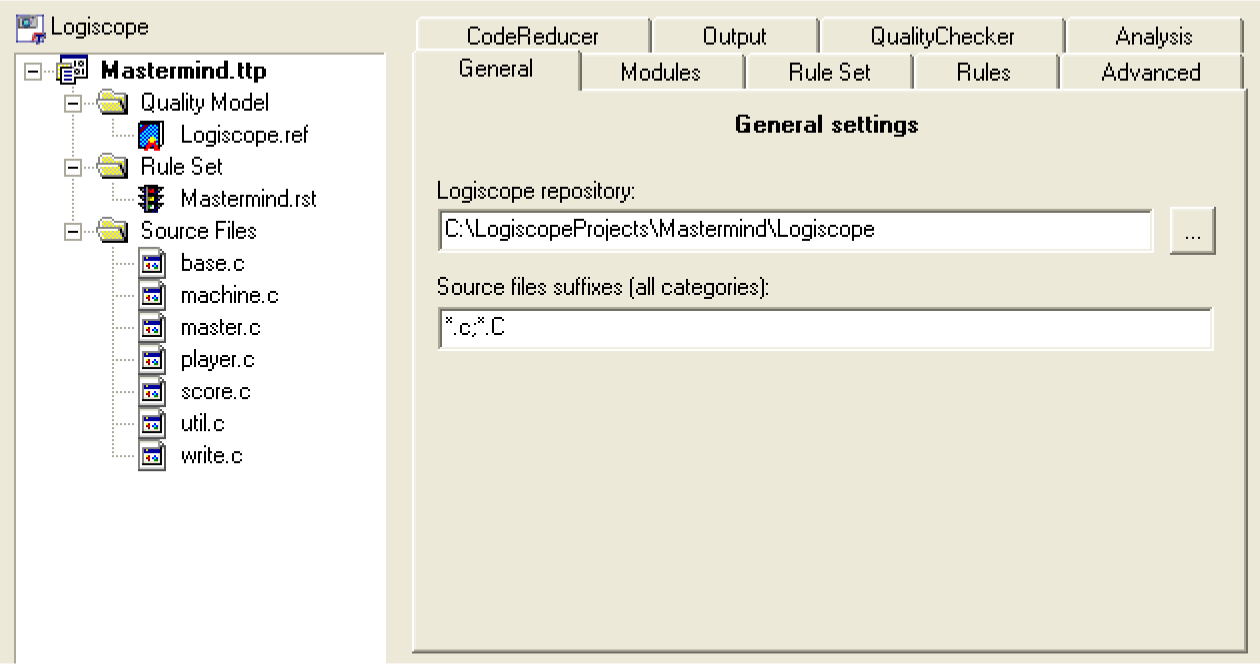
The build process creates or updates some Logiscope internal files in the Logiscope project repository.

If warning or error messages appear in the **Build** tab, double-click on the corresponding line and see the syntax that led to the error or warning. Usually, many errors or warnings are due to missing, incorrect or inconsistent parsing options: i.e. the selected dialect and/or preprocessor directives.

You can easily change them using the **Project Settings** window (see hereafter).

Project Settings Window

2. Activate the **Project** > **Settings...** command.



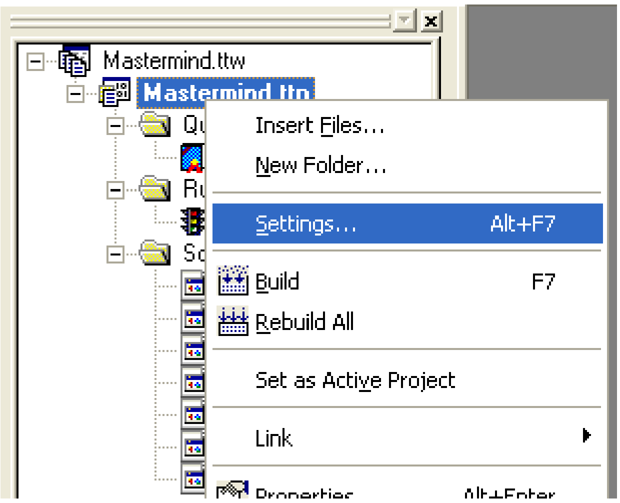
General settings

Logiscope repository: C:\LogiscopeProjects\Mastermind\Logiscope

Source files suffixes (all categories): *.c;*.C

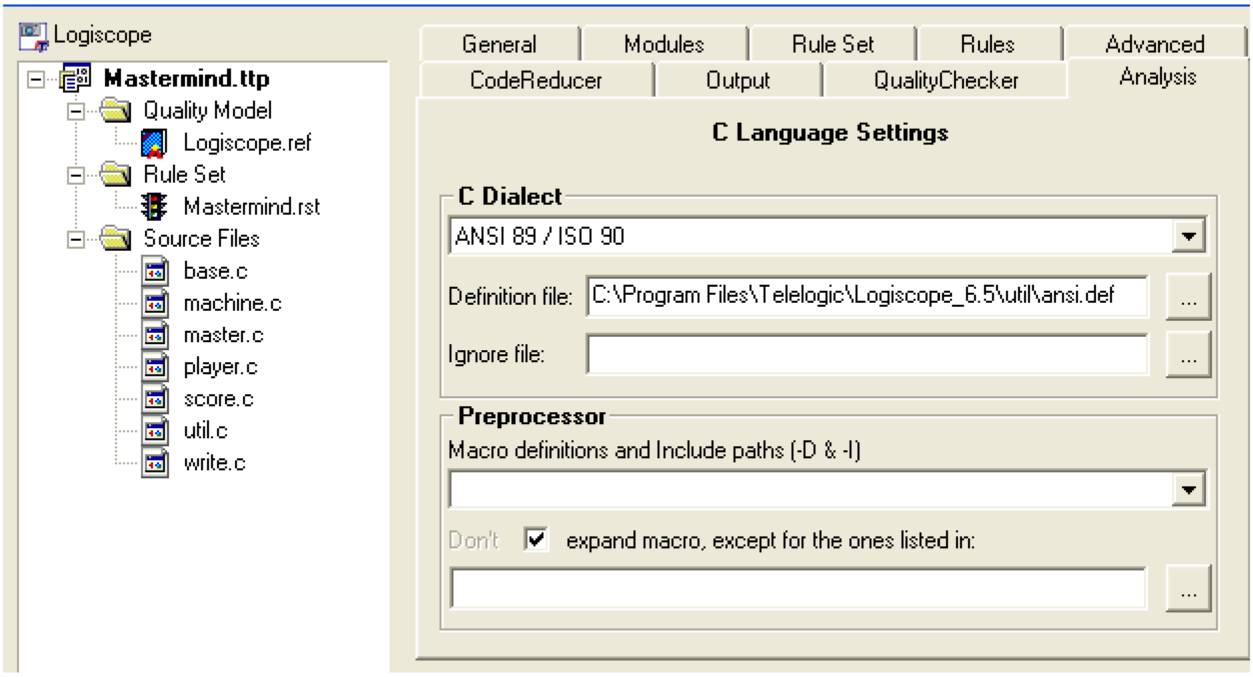
The **Project Settings** window enables to modify all Logiscope project settings including the verification modules to be activated using the various corresponding tabs.
Only the Project source file language cannot be changed.

You can also open the window using the **Alt+F7** key or a shortcut: right click on the Project filename as follows:



3. Click on the **Analysis** tab.

.



C Language Settings

C Dialect

ANSI 89 / ISO 90

Definition file: C:\Program Files\Telelogic\Logiscope_6.5\util\ansi.def

Ignore file:

Preprocessor

Macro definitions and Include paths (-D & -I)

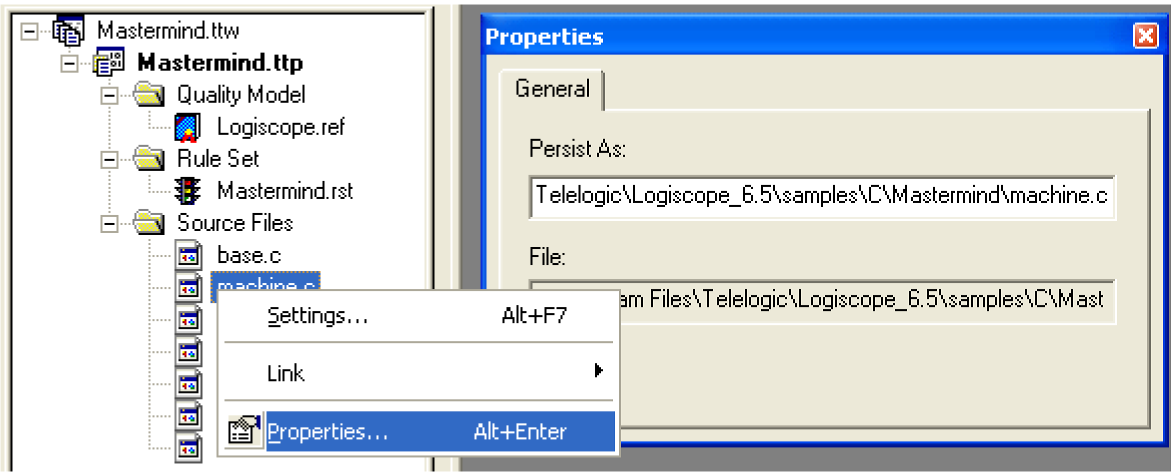
Don't ☑ expand macro, except for the ones listed in:

You can change the **Language Settings** (i.e. the source code parsing options). If you do so, you should click on the **OK** button to take into account the new project settings and then build the Logiscope project again considering the new settings.

4. Click on the **Cancel** button.

Properties Box

5. Select a source file and activate the **View-Properties...** command or the icon, it displays general properties of the selected file, the name and location:



Persist As: Telelogic\Logiscope_6.5\samples\C\Mastermind\machine.c

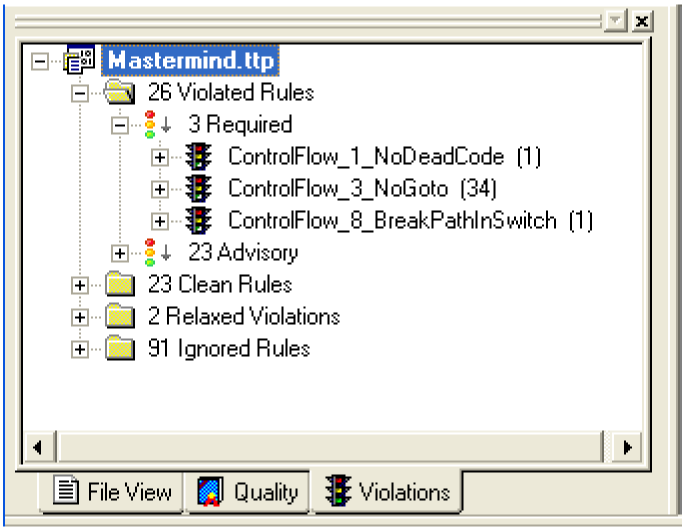
File: am Files\Telelogic\Logiscope_6.5\samples\C\Mast

You can also use a shortcut: as for **Settings...** with a right click on the file.

1.5 Viewing RuleChecker Results

1.5.1 Rule Violations

1 Select **Browse-Rule-Rule Violations** or click on the 🚦 icon.



```
Mastermind.ttp
  26 Violated Rules
    3 Required
      ControlFlow_1_NoDeadCode (1)
      ControlFlow_3_NoGoto (34)
      ControlFlow_8_BreakPathInSwitch (1)
    23 Advisory
  23 Clean Rules
  2 Relaxed Violations
  91 Ignored Rules
```

A new tab **Violations** is created in the **Workspace View** with the following folders:

- **Violated Rules:** this folder lists all the rules where at least one violation has been found in the source code files analyzed,
- **Clean Rules**: this folder lists the rules with no violation found,
- **Relaxed Violations**: this folder lists the rules where some violations have been relaxed using special comments in the source code (see section Relaxing Violations),
- **Ignored Rules**: the rules listed in this folder have not been checked by Logiscope RuleChecker (i.e. the unchecked ones when you chose to customize).

In each folder, the rules are ordered according to the associated severity of the rules: one sub folder per severity level: e.g. “Required”, “Advisory”.

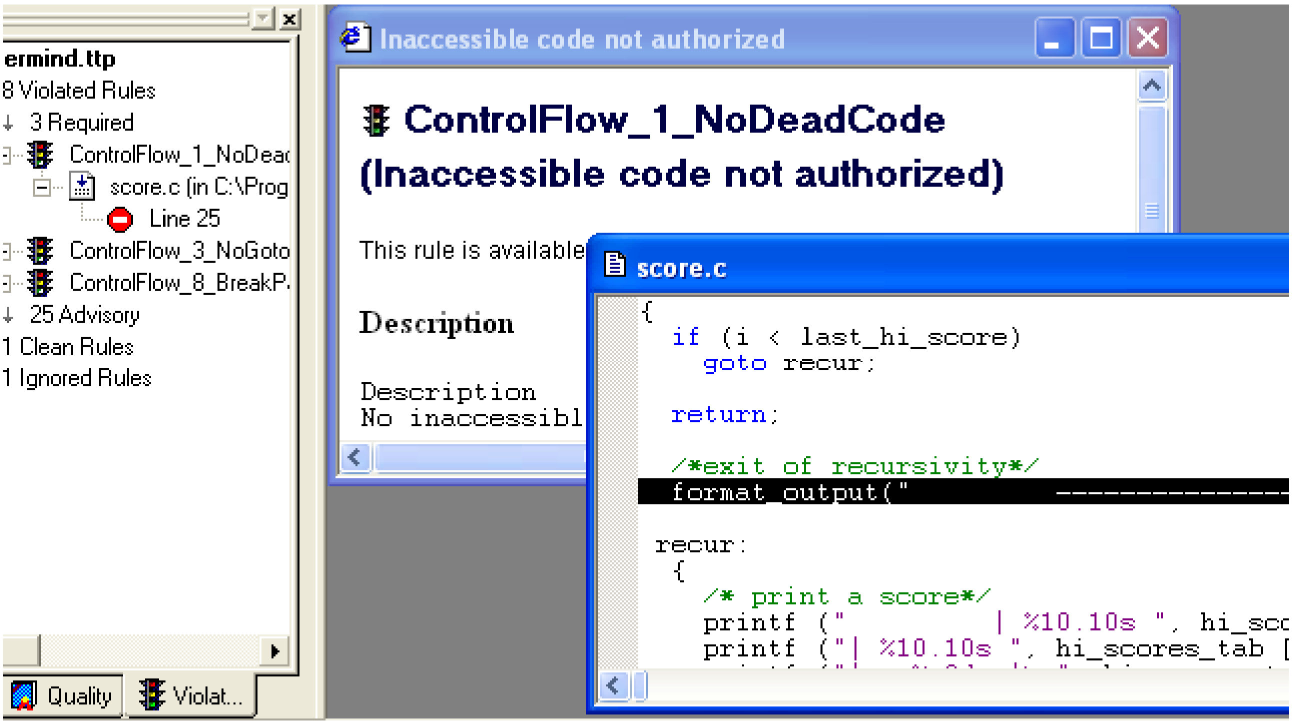
In front of each folder name is specified the number of related items: rules or violations according to the context. For instance, there are 28 violated rules, 3 “*Required*” rules and 25 “Advisory”. There are 34 violations of the *ControlFlow_3_NoGoto* rule.

Note: The default Logiscope rule sets come with predefined severities for each standard rule. You can change the severity by editing the rule set file. For more details, please refer to the document *Kalimetrix Logiscope - RuleChecker & QualityChecker - <Language> Reference Manual.*

2 Double click on the *ControlFlow_1_NoDeadCode* folder: the description of the rule is now displayed in a window in the **Result Pane**.

3 Expand the *ControlFlow_1_NoDeadCode* folder and then the folder corresponding to the file where the violation has been found.
You get the exact location of where the rule has been violated: i.e. line 25 of the score.c file in the example.

4 Double click on the line number.
The source code will be displayed with the cursor highlighting the violation.
The screen looks like this:



ControlFlow_1_NoDeadCode (Inaccessible code not authorized)

This rule is available

Description

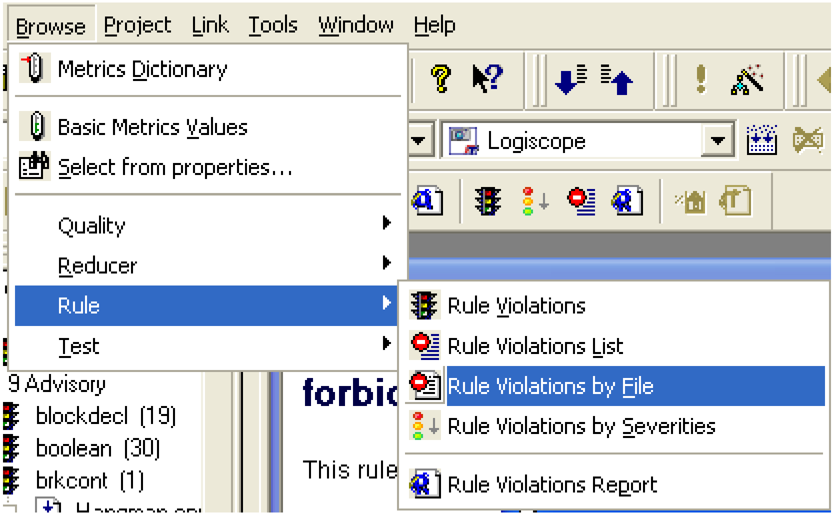
```
Description
No inaccessibl
```

```
{
  if (i < last_hi_score)
    goto recur;

  return;

  /*exit of recursivity*/
  format_output("                ----------------

 recur:
  {
    /* print a score*/
    printf ("          | %10.10s ", hi_sco
    printf ("| %10.10s ", hi_scores_tab [
```

Indeed this is dead code: i.e. some executable statements that can not be executed. The *Rational Logiscope RuleChecker* module definitively improves efficiency of the code reviewing process.

1.5.2 Rule Violations by File

5 Select **Browse-Rule-Rule Violations by File** menu:

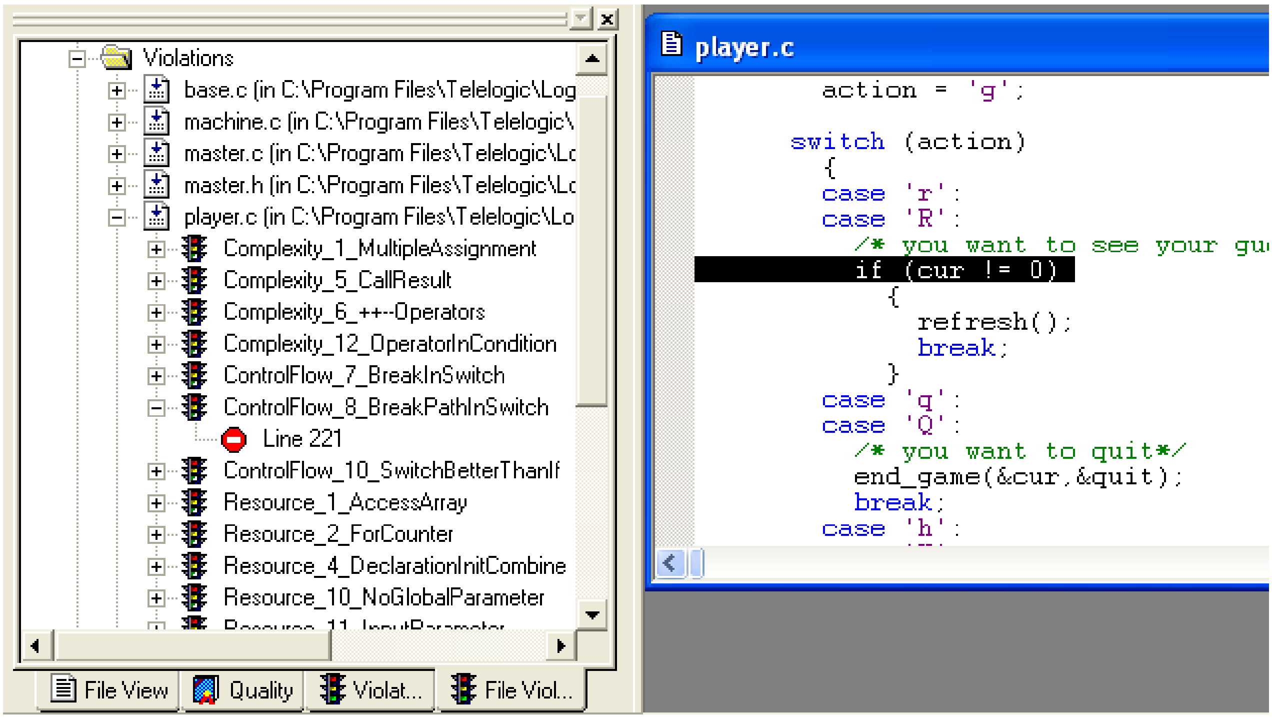


A new tab “File Violations” is added to the **Workspace View** displaying all the files where rule violations have been found.

6 Expand the folder corresponding to one source file: e.g. player.c.
The list of violated rules is now displayed: 1 sub folder per violated rule.

7 Expand the *ControlFlow_8_BreakPathInSwitch* folder/rule. The precise localization: i.e. line number of the violation found in the file is displayed.

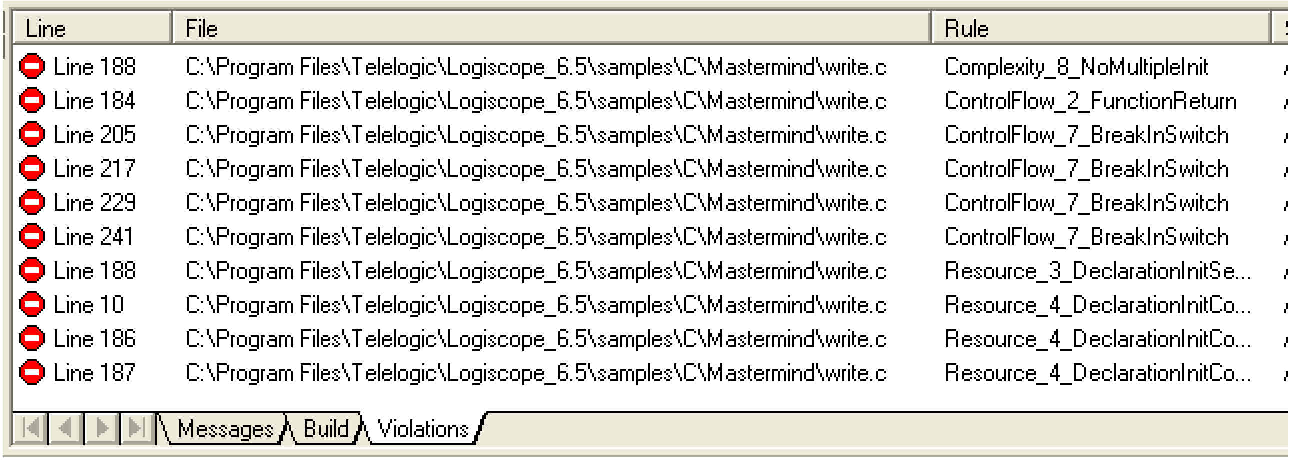
8 As before, just click on the line number to display the corresponding source code. This may be a bug!


Violations
base.c (in C:\Program Files\Telelogic\Log
machine.c (in C:\Program Files\Telelogic\
master.c (in C:\Program Files\Telelogic\Lc
master.h (in C:\Program Files\Telelogic\Lc
player.c (in C:\Program Files\Telelogic\Lo
Complexity_1_MultipleAssignment
Complexity_5_CallResult
Complexity_6_++--Operators
Complexity_12_OperatorInCondition
ControlFlow_7_BreakInSwitch
ControlFlow_8_BreakPathInSwitch
Line 221
ControlFlow_10_SwitchBetterThanIf
Resource_1_AccessArray
Resource_2_ForCounter
Resource_4_DeclarationInitCombine
Resource_10_NoGlobalParameter
File View
Quality
Violat...
File Viol...
player.c

```
    action = 'g';

  switch (action)
    {
    case 'r':
    case 'R':
      /* you want to see your gu
      if (cur != 0)
        {
          refresh();
          break;
        }
    case 'q':
    case 'Q':
      /* you want to quit*/
      end_game(&cur,&quit);
      break;
    case 'h':
```

1.5.3 Rule Violations List

9 Select the **Browse-Rule-Rule Violations List** menu or click on the icon.



Line	File	Rule
Line 188	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	Complexity_8_NoMultipleInit
Line 184	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	ControlFlow_2_FunctionReturn
Line 205	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	ControlFlow_7_BreakInSwitch
Line 217	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	ControlFlow_7_BreakInSwitch
Line 229	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	ControlFlow_7_BreakInSwitch
Line 241	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	ControlFlow_7_BreakInSwitch
Line 188	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	Resource_3_DeclarationInitSe...
Line 10	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	Resource_4_DeclarationInitCo...
Line 186	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	Resource_4_DeclarationInitCo...
Line 187	C:\Program Files\Telelogic\Logiscope_6.5\samples\C\Mastermind\write.c	Resource_4_DeclarationInitCo...

The list of all Violated Rule lines is displayed in the Output window in a dedicated tab.

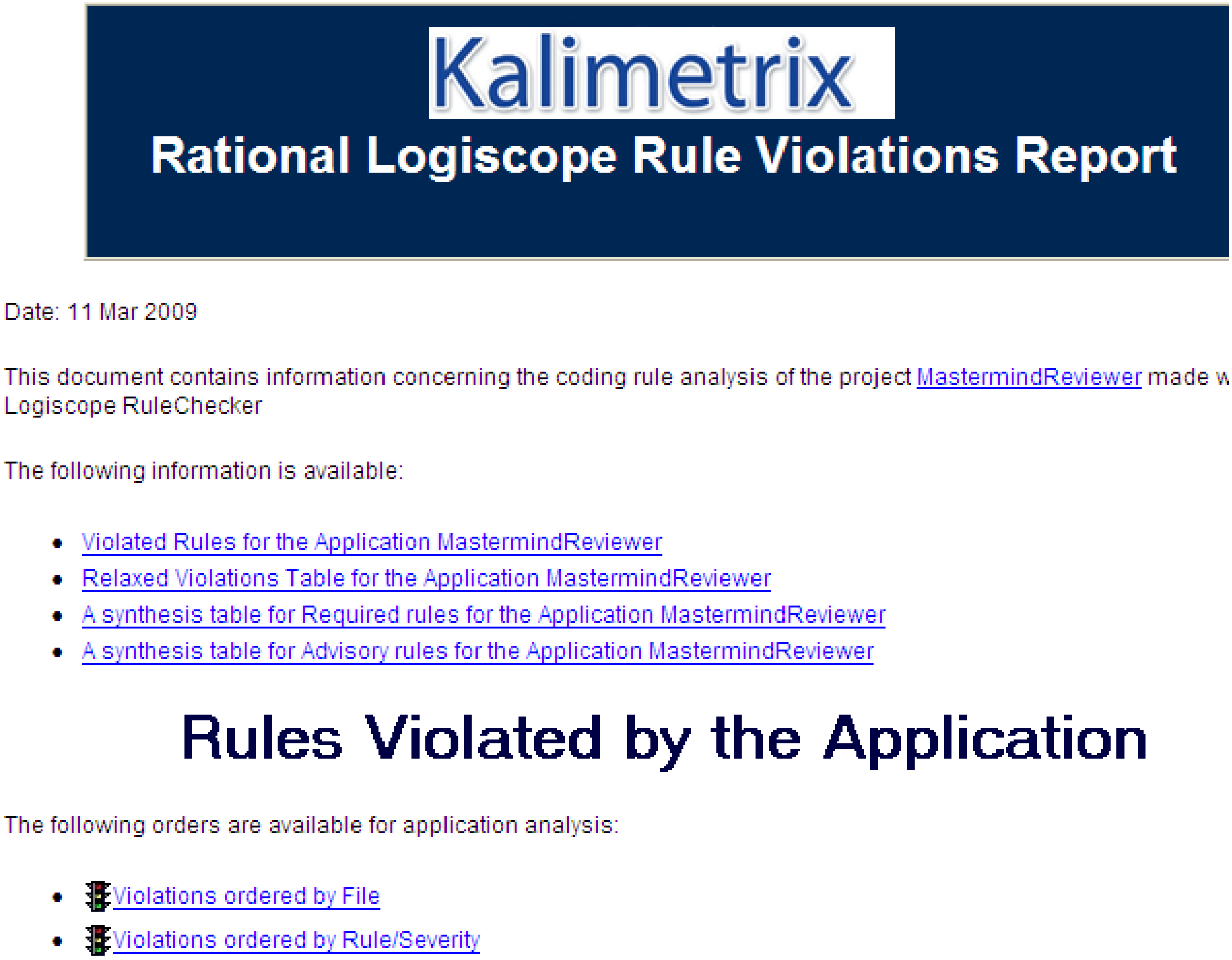
10 Double-click on a line, its corresponding source file appears with the violation highlighted.

You can go through this list using the **<F4>** and **<Shift + F4>** commands.

1.5.4 Rule Violations Report

11 Select **Browse-Rule-Rule Violations Report** or click on the icon.

Kalimetrix Logiscope Rule Violations Report gives you an HTML synthesis of the coding rule checking performed by *Logiscope RuleChecker.*



Kalimetrix

Rational Logiscope Rule Violations Report

Date: 11 Mar 2009

This document contains information concerning the coding rule analysis of the project MastermindReviewer made w Logiscope RuleChecker

The following information is available:

- Violated Rules for the Application MastermindReviewer
- Relaxed Violations Table for the Application MastermindReviewer
- A synthesis table for Required rules for the Application MastermindReviewer
- A synthesis table for Advisory rules for the Application MastermindReviewer

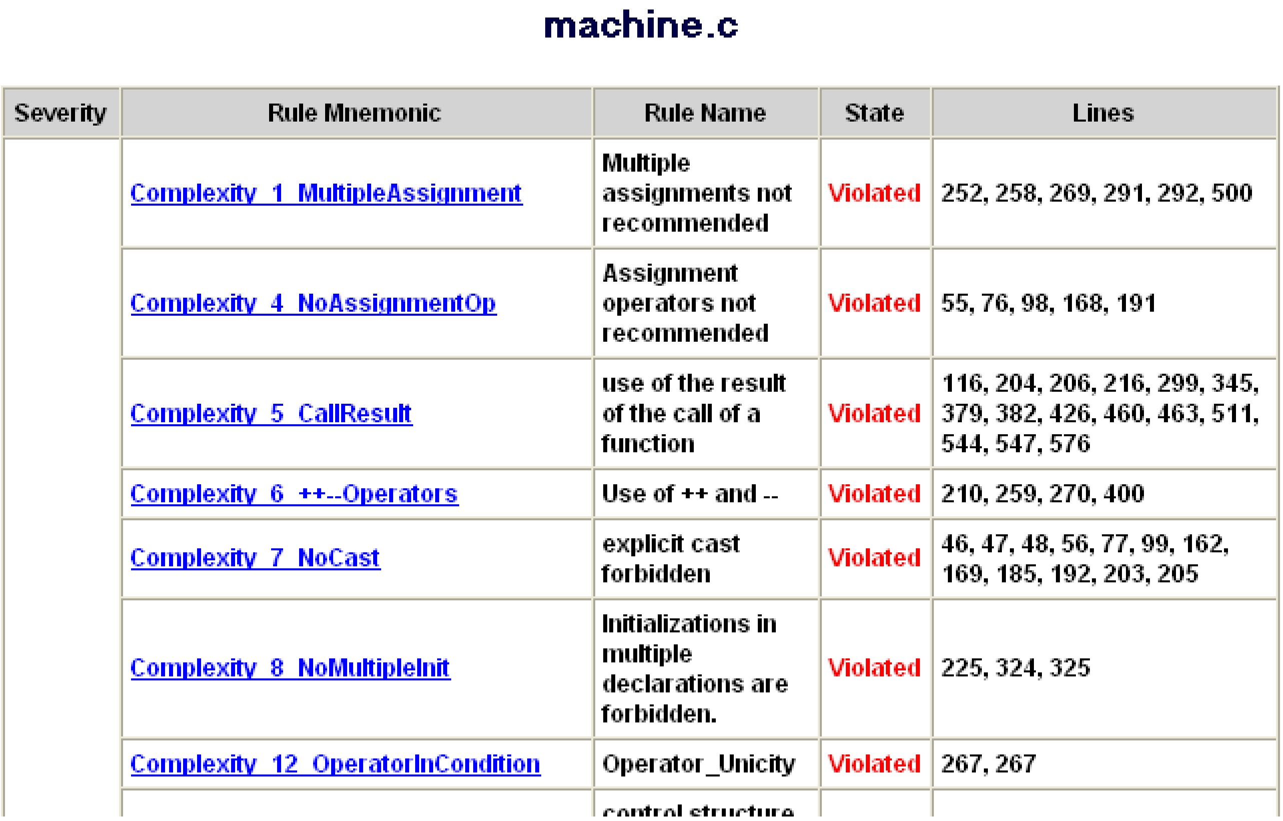
Rules Violated by the Application

The following orders are available for application analysis:

- Violations ordered by File
- Violations ordered by Rule/Severity

6. Select the Violated Rules for the Application Mastermind link.

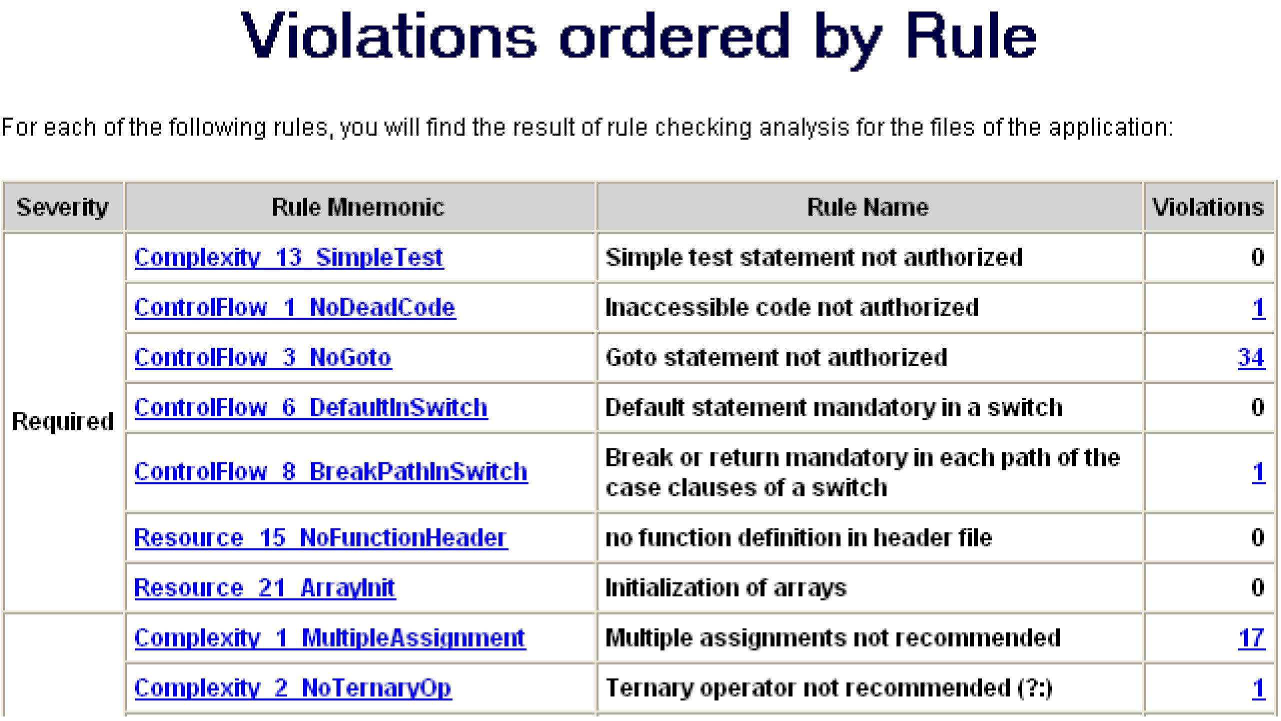
7. Click Violations ordered by File and then on the machine.c file, you get the following view:



machine.c

Severity	Rule Mnemonic	Rule Name	State	Lines
	Complexity_1_MultipleAssignment	Multiple assignments not recommended	Violated	252, 258, 269, 291, 292, 500
	Complexity_4_NoAssignmentOp	Assignment operators not recommended	Violated	55, 76, 98, 168, 191
	Complexity_5_CallResult	use of the result of the call of a function	Violated	116, 204, 206, 216, 299, 345, 379, 382, 426, 460, 463, 511, 544, 547, 576
	Complexity_6_++--Operators	Use of ++ and --	Violated	210, 259, 270, 400
	Complexity_7_NoCast	explicit cast forbidden	Violated	46, 47, 48, 56, 77, 99, 162, 169, 185, 192, 203, 205
	Complexity_8_NoMultipleInit	Initializations in multiple declarations are forbidden.	Violated	225, 324, 325
	Complexity_12_OperatorInCondition	Operator_Unicity	Violated	267, 267
		control structure		

8. Now click the Back arrow ⇦ twice in the **HTML Browser Toolbar**.
9. Click the Violations ordered by Rule/Severity link.

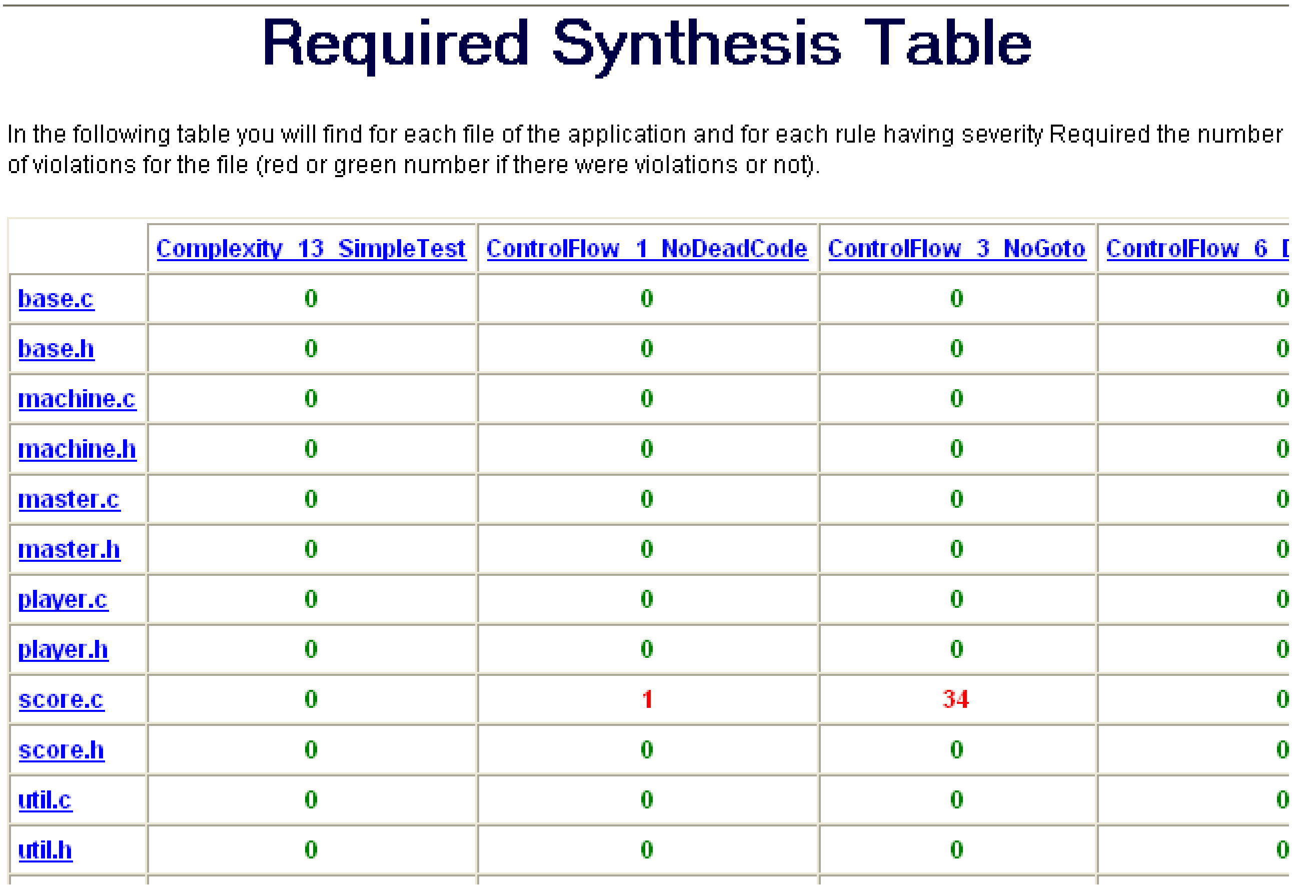


Violations ordered by Rule

For each of the following rules, you will find the result of rule checking analysis for the files of the application:

Severity	Rule Mnemonic	Rule Name	Violations
Required	Complexity_13_SimpleTest	Simple test statement not authorized	0
	ControlFlow_1_NoDeadCode	Inaccessible code not authorized	1
	ControlFlow_3_NoGoto	Goto statement not authorized	34
	ControlFlow_6_DefaultInSwitch	Default statement mandatory in a switch	0
	ControlFlow_8_BreakPathInSwitch	Break or return mandatory in each path of the case clauses of a switch	1
	Resource_15_NoFunctionHeader	no function definition in header file	0
	Resource_21_ArrayInit	Initialization of arrays	0
	Complexity_1_MultipleAssignment	Multiple assignments not recommended	17
	Complexity_2_NoTernaryOp	Ternary operator not recommended (?:)	1

10. Now, get back to the beginning of the **HTML Report** and select the A synthesis table for Required rule for the application Mastermind.



Required Synthesis Table

In the following table you will find for each file of the application and for each rule having severity Required the number of violations for the file (red or green number if there were violations or not).

	Complexity_13_SimpleTest	ControlFlow_1_NoDeadCode	ControlFlow_3_NoGoto	ControlFlow_6_I
base.c	0	0	0	0
base.h	0	0	0	0
machine.c	0	0	0	0
machine.h	0	0	0	0
master.c	0	0	0	0
master.h	0	0	0	0
player.c	0	0	0	0
player.h	0	0	0	0
score.c	0	1	34	0
score.h	0	0	0	0
util.c	0	0	0	0
util.h	0	0	0	0

This synthesis table gives you a general view of violations of the Required rule. Above, the rule ControlFlow_3_NoGoto has been violated 34 times in the score.c file.

Each time you click on a rule, you get its definition and each time you click on a file you get the corresponding table (Violated Rules ordered by file).

The color code is defined as:

red: number of Rule Violations.

green: Rule Not Violated

yellow: Ignored Rules.

Next click on the ControlFlow_3_NoGoto column, the definition appears.

1.5.5 Customizing Rules and Rule Sets

Logiscope *RuleChecker* is highly customizable. It allows you to adapt the rule checking to your specific context taking into account the applicable coding standard.

- The project settings help you to choose what rules shall be checked on the code. This may depend on the type of code under verification. One given rule set can be defined to check newly developed sub-contracted applications when an other can be defined for old legacy internal source code. Definitely, the applicable standards are not the same.
- In Ada, C++ or Java, some rules have parameters that allow to customize the verification. For instance, you can easily define your own list of forbidden functions checked by the rule *funcres* or specify the description of the expected function header comments checked by the rule *headercom*.
- The default name of a standard rule can be changed to fit to the name and/or identifier specified in the company coding standard.
  You can even have the same standard rule used twice with different names and different parameters.
- You can change the severity level of a rule.
- You can define your own severity levels with a specific ordering: e.g. “Mandatory”, “Highly recommended”, “Recommended”.
- You can change the standard description of a rule and put the one written in your coding standard.

To modify the parameters of a rule, double click on the **.rst** file and edit the corresponding part of the rule specification you want to customize then save the file.

If you did not choose to generate a flat rule set when you created the project you may have to edit the rule set file that is included in your project’s “**.rst**” file.

For more details, please refer to the *Kalimetrix Logiscope - RuleChecker & QualityChecker - <Language> Reference Manual*.

1.5.6 Relaxing Violations Using Special Comments

When this feature is activated, rule violations that have been checked and that you have decided are acceptable exceptions to the rule, can be relaxed for future builds: **they will no longer appear in the list of rule violations.** This can be very useful when checking violations in a context where multiple reviews are performed.

The violations that have been relaxed will remain accessible for future reference.

The relaxation mechanism is based on comments inserted into the code where the tolerated violations are. There are two ways to do this, depending on whether there is a single rule violation to relax on the line, or multiple ones to relax on the given line.

Relaxing a single rule violation

If there is a single violation to relax, it can be done as a comment on the same line as the code, using the following syntax (for C code):

```
some code /* %RELAX<rule_mnemonic> justification */
```

where:

- `rule_mnemonic`: is the mnemonic of the rule that you want to ignore violations of on the current line.

- `justification`: is free text, allowing to justify the relaxation of the rule violation.

Note that the combination of characters introducing / closing a comment shall be adapted according to the syntax applicable to the language of the project, e.g.:

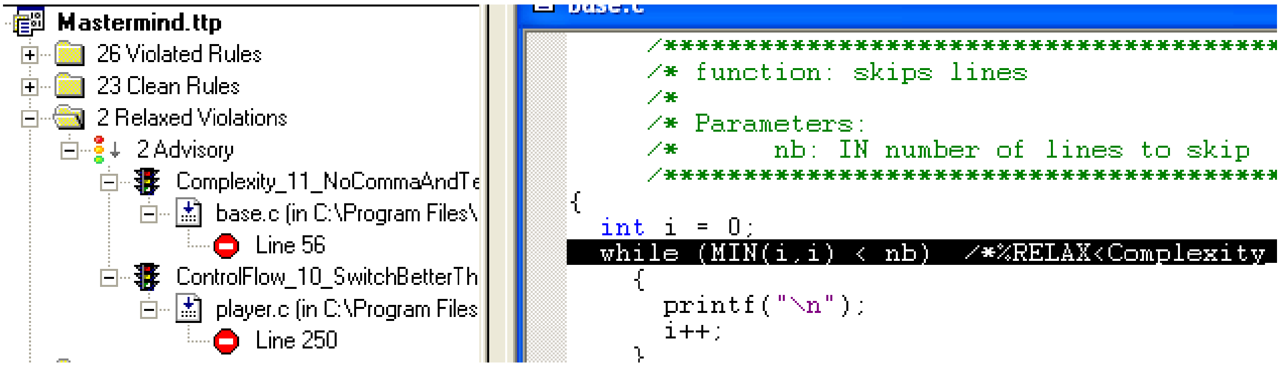
- `--`: for Ada.

- `// ...` : for C++ and Java.

The Logiscope project in use already contains 2 Relaxed Violations as shown in the tree displayed in the Violations tab.

11. Fully expand the Relaxed Violations folder in the **Violations** tab.

12.Double-click on the node “Line 56” to display the corresponding source code.


Mastermind.ttp
26 Violated Rules
23 Clean Rules
2 Relaxed Violations
2 Advisory
Complexity_11_NoCommaAndTe
base.c (in C:\Program Files\
Line 56
ControlFlow_10_SwitchBetterTh
player.c (in C:\Program Files
Line 250
base.c

```
      /*************************************
      /* function: skips lines
      /*
      /* Parameters:
      /*        nb: IN number of lines to skip
      /*************************************
{
  int i = 0;
  while (MIN(i,i) < nb)   /*%RELAX<Complexity
    {
      printf("\n");
      i++;
    }
```

Indeed, the rule is violated because of the use of the ternary operator in the definition of the MIN macro. This may justify someone relaxing the violation.

Note that in such a case, the correct parsing approach would have been to declare the MIN macro as NOT to be expanded when setting the project parsing options. Please refer to the *Kalimetrix Logiscope RuleChecker & QualityChecker C Reference Manual* for more details on this feature.

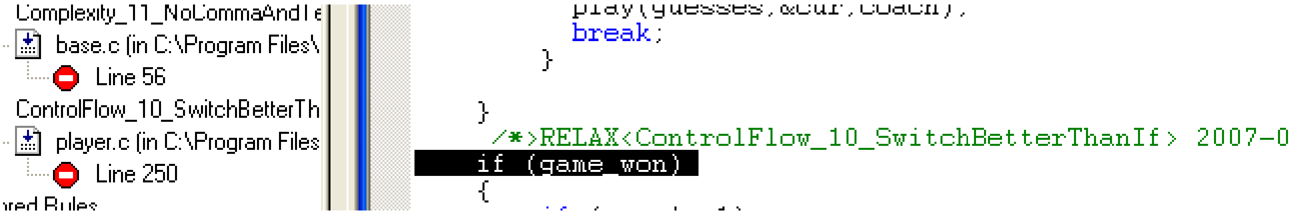
Relaxing several violations and/or adding a longer justification

If there are several violations to relax for a same line (several violations occurring in different places in the code at the same time cannot be relaxed), or if the justification of the violation should be written over several lines, the following syntax should be used.

```
/* >RELAX<rule_mnemonic> justification */
```

followed by any number of empty lines, comment lines, or relaxations of other rules relating to the same code line, then by the code line of the violation.

13.In the **Violations** tab, double-click on the node “Line 250” to display the relaxed violation found in the file player.c:


Complexity_11_NoCommaAndTe
base.c (in C:\Program Files\
Line 56
ControlFlow_10_SwitchBetterTh
player.c (in C:\Program Files
Line 250
red Rules

```
          play(guesses,&cur,coach);
          break;
        }

    }
     /*>RELAX<ControlFlow_10_SwitchBetterThanIf> 2007-0
    if (game_won)
    {
```

In this context, the violation has been relaxed during a formal Code Review as mentioned in the justification that could have been written on several lines.

Relaxing all violations in pieces of code

If all the violations of one or more rules are to be relaxed in a given piece of code (e.g. reused code included in a newly developed file), the piece of code should be surrounded by:

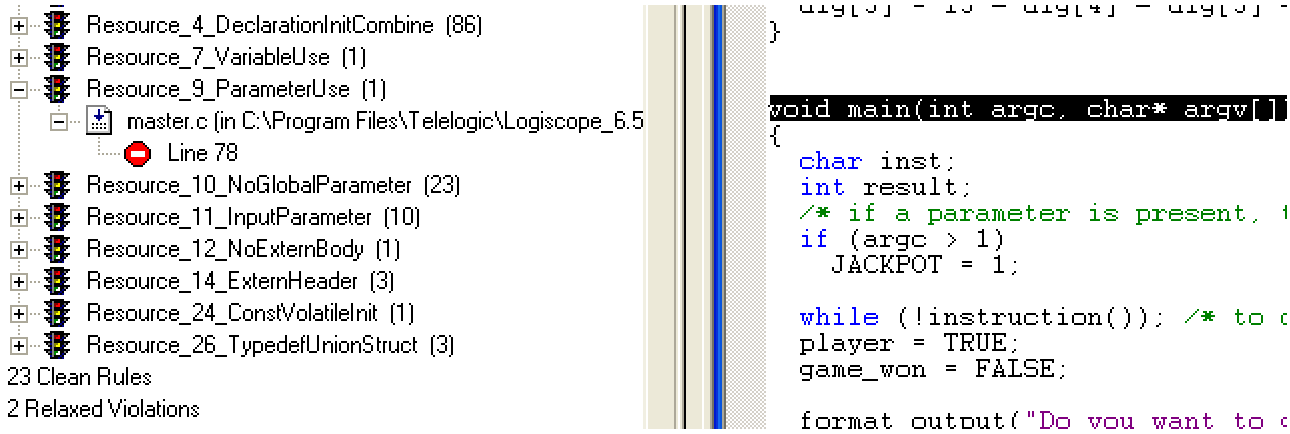
```
 //  {{RELAX<list_of_rule_mnemonics> justification
the piece of code
 //  }}RELAX<list of rule mnemonics>
```

where:

- `list_of_rule_mnemonics`: is the list of all mnemonics of the rules that you want to ignore violations of on the piece of code.
The rule mnemonics shall be separated by a comma.

Relaxing a violation in the Logiscope project.

In the *Logiscope* project created in the previous section, you want to relax the violation of the *Resource_9_ParameterUse* rule using one of the various syntaxes of comments introduced in the previous sections.
Indeed, the violation relates to the *main* function and the program specification states that the behavior must changed if there is one or more arguments whatever the arguments, so the argv parameter is not used intentionally.



```
⊞ Resource_4_DeclarationInitCombine (86)
⊞ Resource_7_VariableUse (1)
⊟ Resource_9_ParameterUse (1)
    ⊟ master.c (in C:\Program Files\Telelogic\Logiscope_6.5
        Line 78
⊞ Resource_10_NoGlobalParameter (23)
⊞ Resource_11_InputParameter (10)
⊞ Resource_12_NoExternBody (1)
⊞ Resource_14_ExternHeader (3)
⊞ Resource_24_ConstVolatileInit (1)
⊞ Resource_26_TypedefUnionStruct (3)
23 Clean Rules
2 Relaxed Violations
```

```
}

void main(int argc, char* argv[])
{
  char inst;
  int result;
  /* if a parameter is present, t
  if (argc > 1)
    JACKPOT = 1;

  while (!instruction()); /* to c
  player = TRUE;
  game_won = FALSE;

  format_output("Do you want to c
```

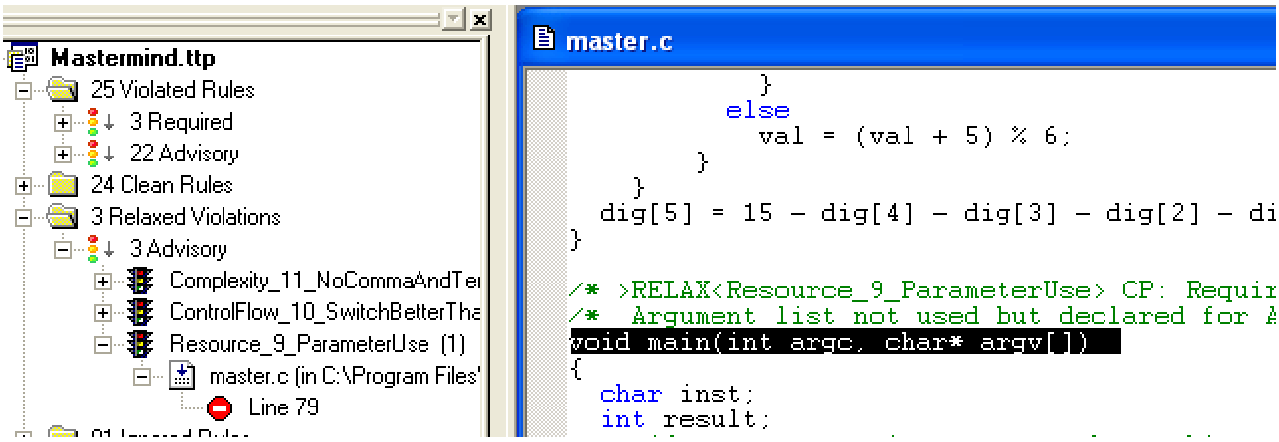
To relax the violation which is highlighted in the code:

1. Add the following comment before the line 78 in the file master.c:

```
/*>RELAX<Resource_9_ParameterUse> CP: Requirement Id. FUN_625: */
/* Argument list not used but declared for ANSI compliance     */
```

2. Save the file using `<CTRL S>`
3. Rebuild the projet using the command **Projet-Build**.
4. Select again the **Browse-Rule-Rule Violations by File** menu to refresh the results.

The violation is removed from the list of violations and now appears in the Relaxed Violations:

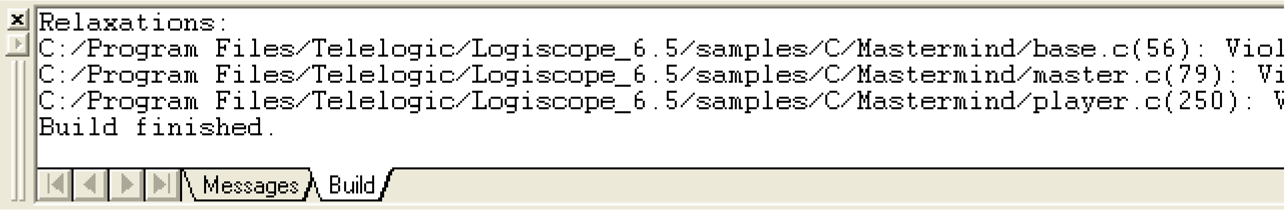


```
Mastermind.ttp
⊟ 25 Violated Rules
    ⊞ 3 Required
    ⊞ 22 Advisory
⊞ 24 Clean Rules
⊟ 3 Relaxed Violations
    ⊟ 3 Advisory
        ⊞ Complexity_11_NoCommaAndTe
        ⊞ ControlFlow_10_SwitchBetterTha
        ⊟ Resource_9_ParameterUse (1)
            ⊟ master.c (in C:\Program Files'
                Line 79
```

master.c

```
            }
          else
            val = (val + 5) % 6;
        }
    }
  dig[5] = 15 - dig[4] - dig[3] - dig[2] - di
}

/* >RELAX<Resource_9_ParameterUse> CP: Requir
/*  Argument list not used but declared for A
void main(int argc, char* argv[])
{
  char inst;
  int result;
```

Note that the relaxations found in the code are notified in the **Build Output** tab:



```
Relaxations:
C:/Program Files/Telelogic/Logiscope_6.5/samples/C/Mastermind/base.c(56): Viol
C:/Program Files/Telelogic/Logiscope_6.5/samples/C/Mastermind/master.c(79): Vi
C:/Program Files/Telelogic/Logiscope_6.5/samples/C/Mastermind/player.c(250): V
Build finished.
```

1.5.7 Importing External Violations

By checking the **Activate external violations import** box, you can specify a Logiscope Project folder in which you will put files that contain violation information from external tools, for example compilation results, or the results of a program that you have written. These files will then be taken into account and their violations added to the violation information available in the project.

These data files containing the external violation information shall comply with the following format.

Import data file format

The data file to import should contain lines respecting one of the following formats:

```
"pathname" line_number "rule_mnemonic" ["message"]
```

or

```
"pathname" start_char end_char line_number "rule_mnemonic" ["message"]
```

where:

- **pathname**: is the pathname (either relative to the project, or the full path) of the source file where a violation has been found,
- **line_number**: is the line number of the violation,
- **rule_mnemonic**: is the mnemonic of the violated rule,
- **start_char**: is the position of the first character of the text to select for the violation (counting from the beginning of the file),
- **end_char**: is the position of the last character of text to select for the violation (counting from the beginning of the file),
- **message**: is an optional free text comment associated to either the violation or the rule.

When start_char and end_char are omitted, the whole line is selected when locating the violation in the file.

Each line in the file will be transformed into a violation.

Example:

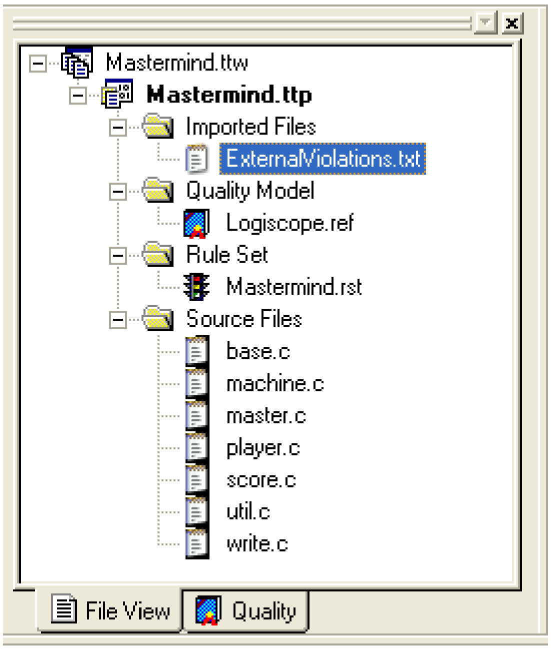
```
"C:\Mastermind\machine.c" 14 "indentation"
"C:\Mastermind\machine.c" 21 "indentation"
"C:\Mastermind\machine.c" 153 160 7 "reserved_classes"
```

Files to import can have any suffix as long as the data they contain respect the specified format.

Adding a violation data file into a Logiscope Project

The files to be imported have to be added to the Logiscope Project as described below:

- Create a new folder in the Project named "Imported Files".
- Add all files to be imported in the newly created folder.



Add rule files to your project that correspond to the rules that are going to be found in the violation file. The rule files should be in the following format:

- the file for the rule rule_mnemonic should be called rule_mnemonic.std
- the contents of the file should follow this syntax:

  .NAME long_name

  .DESCRIPTION user_description

  .COMMAND external

where:

- **long_name** is free text, that can include spaces. It’s a more detailed title of the rule. It will appear as an explanation of the rule name in Logiscope.
- **user_description** is the description of the rule, that will be available in Logiscope.
- **external** is the type of command used for this rule, and should not be changed.

Example: The rule file is called indentation.std.

It contains these lines:

```
.NAME Use indentation
.DESCRIPTION
Use correct indentation to clarify code.
.COMMAND external
```

The external rules should be added to the project. You can use the project settings.

You will need to rebuild the project to see the imported violations in the Logiscope project.

For more details, please refer to the *Kalimetrix Logiscope Basic Concepts* manual.

1.5.8 Managing Generated Source Code

It is possible to specify to *Logiscope RuleChecker* that all or part of the source code has been generated by a user-specified tool (e.g. *Rational Rhapsody).* Therefore, rule violations found in the generated code are hidden. The violations found in the generated source code can also be seen as relaxed violations by setting the “Show violations in generated code as relaxations» option.

A source code file is is considered as tool-generated if it contains one of the following:

- a comment starting by #[,
- a comment starting by ##,
- a comment containing `//! Generated Date:`

Otherwise, the source code file is considered as “hand written”.

In generated file, user-written code is detected using the following rules:

- a line following a line starting by a comment starting with ## is a user code line,
- a line ending with comment starting with ## is a user code line,
- a line starting with a comment starting with #[open a block of user code ending on a line starting with a comment starting with #].

Exception: if a comment ## is followed by “`auto_generated`” it does not introduce user code.

1.6 Viewing QualityChecker Results

Viewing results through the **Browse** menu and the **Logiscope** toolbar gives a full description of the analysis performed and of the language used.

The analysis of the project performed with Logiscope *QualityChecker* gives out information on: Application , Modules, Classes , Methods & Functions scopes.

For each of these scopes, information is available at the:

- Factors level ,
- Criteria level and,
- Metrics level

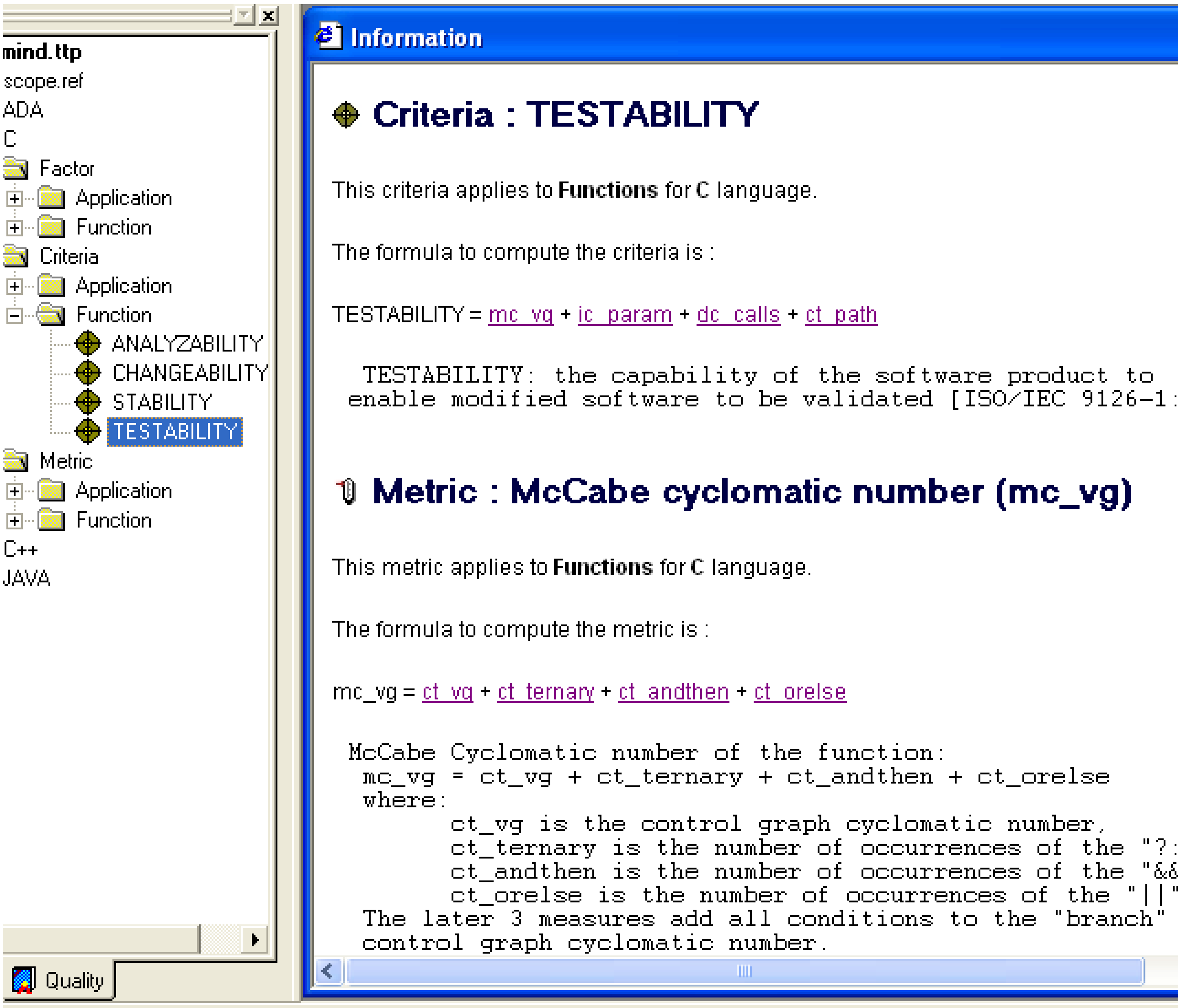
1.6.1 Quality Model

1. Select the **Quality** tab in the **Workspace View** and expand the folder corresponding to the Logiscope project programming language: e.g.: **C**.
   The **Quality** tab allows to see the specification of the Quality Model: the breakdown in Factors- Criteria - Metrics levels at each scope (e.g. Application, Packages, Modules, Classes or Functions) depending on the programming language.

2. Expand the **Criteria** folder and then the **Function** sub-folder
   The Criteria included in the chosen Quality Model file at Function scope are now listed. In this context, the four criteria are the sub-characteristics associated to the Maintainability characteristic as specified in the ISO 9126-1: 2001 international standard. The default Logiscope Quality Model complies with the recommendations of the ISO standard.

3. Double-click on the different items: e.g. **TESTABILITY**, an Information window is displayed with their definitions.
   For instance, at function scope, the level of TESTABILITY is evaluated by combining 4 metrics.

4. Click on one of the metric identifiers to get its description: e.g. mc_vg.



Criteria : TESTABILITY

This criteria applies to **Functions** for **C** language.

The formula to compute the criteria is :

TESTABILITY = mc_vg + ic_param + dc_calls + ct_path

```
 TESTABILITY: the capability of the software product to
enable modified software to be validated [ISO/IEC 9126-1:
```

Metric : McCabe cyclomatic number (mc_vg)

This metric applies to **Functions** for **C** language.

The formula to compute the metric is :

mc_vg = ct_vg + ct_ternary + ct_andthen + ct_orelse

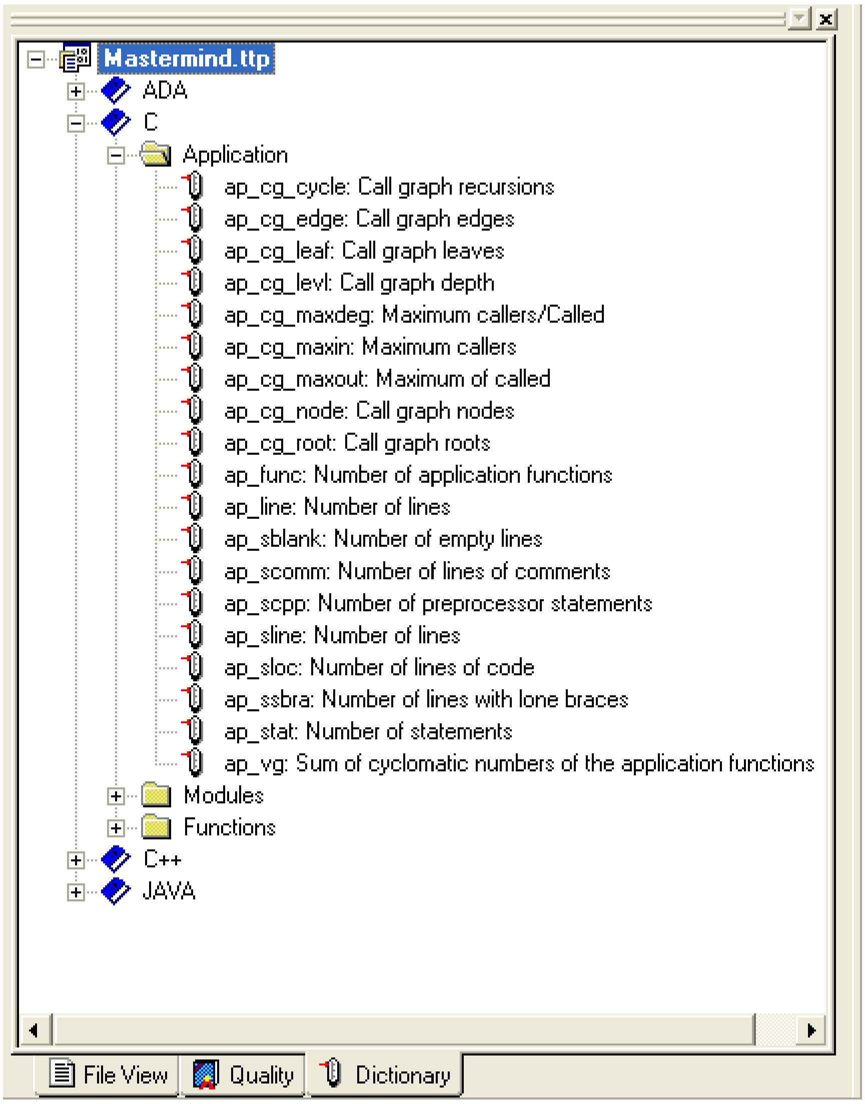
```
McCabe Cyclomatic number of the function:
 mc_vg = ct_vg + ct_ternary + ct_andthen + ct_orelse
 where:
       ct_vg is the control graph cyclomatic number,
       ct_ternary is the number of occurrences of the "?:
       ct_andthen is the number of occurrences of the "&&
       ct_orelse is the number of occurrences of the "||"
 The later 3 measures add all conditions to the "branch"
 control graph cyclomatic number.
```

In this case, the metric mc_vg is not a basic metric but the sum of some other basic metrics provided by Logiscope *QualityChecker*. This definition is written in the Quality Model file. Of course, such a definition can be changed to be tuned to match the quality requirements of the project under review.

The Metrics Dictionary provides an exhaustive list of the metrics.

1.6.2 Metrics Dictionary

5. Select the **Browse-Metrics Dictionary** menu or click on the icon.
6. Expand the folder corresponding to the Logiscope project programming language: e.g.: **C**.
   The various scopes of metrics available for this programming language are displayed as folders: e.g.: Application, Modules, Functions
7. Expand the **Application** folder.
   All the basic metrics available in Logiscope *QualityChecker* for assessing the level of quality of the Application appear.



Mastermind.ttp
ADA
C
Application
ap_cg_cycle: Call graph recursions
ap_cg_edge: Call graph edges
ap_cg_leaf: Call graph leaves
ap_cg_levl: Call graph depth
ap_cg_maxdeg: Maximum callers/Called
ap_cg_maxin: Maximum callers
ap_cg_maxout: Maximum of called
ap_cg_node: Call graph nodes
ap_cg_root: Call graph roots
ap_func: Number of application functions
ap_line: Number of lines
ap_sblank: Number of empty lines
ap_scomm: Number of lines of comments
ap_scpp: Number of preprocessor statements
ap_sline: Number of lines
ap_sloc: Number of lines of code
ap_ssbra: Number of lines with lone braces
ap_stat: Number of statements
ap_vg: Sum of cyclomatic numbers of the application functions
Modules
Functions
C++
JAVA

With a double-click on a metric its full description is displayed.

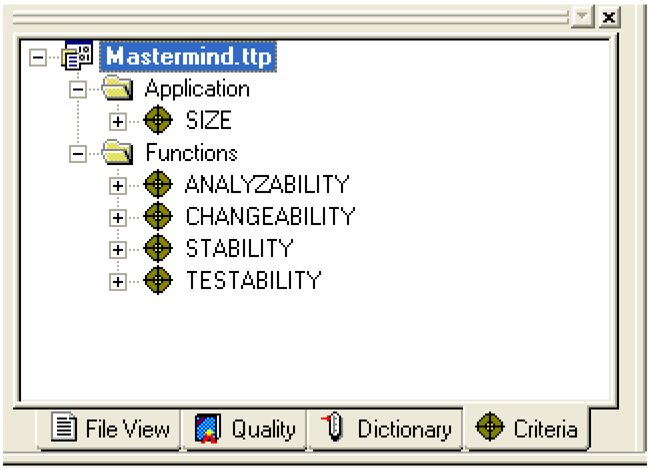
All the metrics can be used to determine calculated metrics or evaluate criteria in the Quality Model. For more details, please refer to the Chapter *Standard Metrics* in the corresponding *Kalimetrix Logiscope RuleChecker & QualityChecker - <Language> Reference Manual*.

1.6.3 Criteria Level

8. Select the **Browse-Quality-Criteria Level** menu or click on the icon:

A new tab **Criteria** appears in the **Workspace View.** It lists all the types of software components analyzed (e.g.: Function, Application according to the programming language) and for each, the list of Quality criteria defined in the Quality Model: e.g. ANALYZABILITY, CHANGEABILITY, STABILITY, TESTABILITY as specified in the ISO/IEC 9126-1 Quality Model internal standard.

Note: you can define your own Quality criteria by editing the Quality Model.

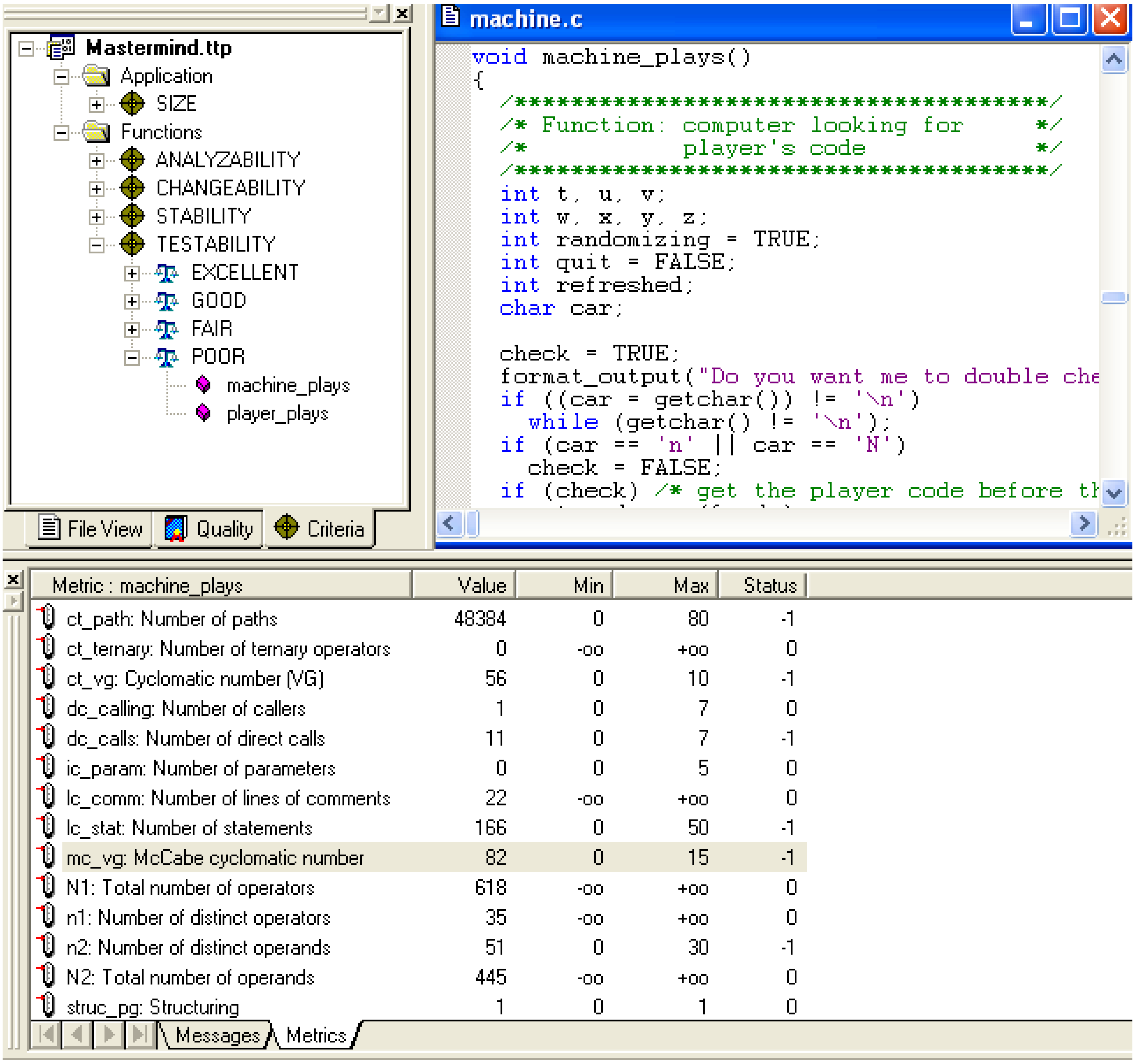


Different actions are now possible from this tab:

9. Expand any criterion to see associated rating levels or categories specified in the Quality Model in use: for instance, TESTABILITY
   : EXCELLENT, GOOD, FAIR, POOR.

10.Expand each rating level to discover the components which have been rated according to the Quality Model.
For the TESTABILITY criterion, 2 functions are rated “POOR”

11. ouble-click on one of the listed items: function, class or file to make the corresponding source code appear. In the output window, the list of metrics used for the analysis with each corresponding value per metric; minimum and maximum thresholds is also displayed. A **status** column indicates if the metric value is correct or out of range (-1).
A click on the column title allows to sort the metric list according to name, value or status.
Double-click on a metric in the **Output window** and its corresponding definition is displayed in the result pane.



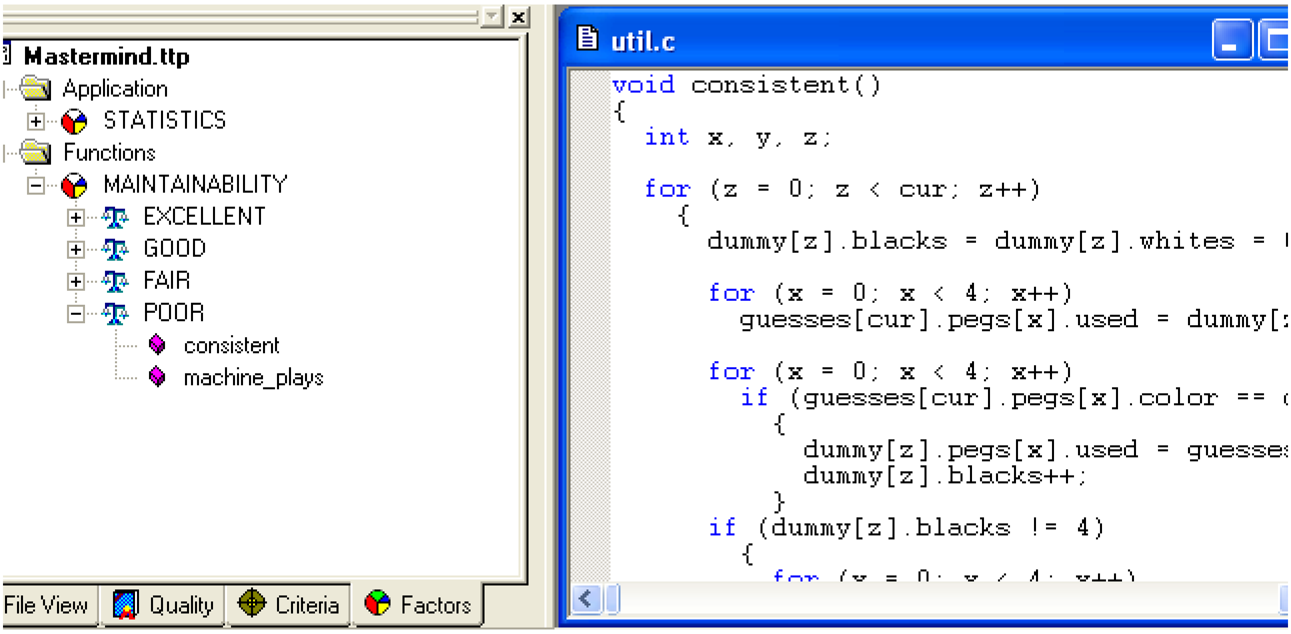
```
void machine_plays()
{
  /****************************************/
  /* Function: computer looking for      */
  /*           player's code             */
  /****************************************/
  int t, u, v;
  int w, x, y, z;
  int randomizing = TRUE;
  int quit = FALSE;
  int refreshed;
  char car;

  check = TRUE;
  format_output("Do you want me to double che
  if ((car = getchar()) != '\n')
    while (getchar() != '\n');
  if (car == 'n' || car == 'N')
    check = FALSE;
  if (check) /* get the player code before th
```

Metric : machine_plays	Value	Min	Max	Status
ct_path: Number of paths	48384	0	80	-1
ct_ternary: Number of ternary operators	0	-oo	+oo	0
ct_vg: Cyclomatic number (VG)	56	0	10	-1
dc_calling: Number of callers	1	0	7	0
dc_calls: Number of direct calls	11	0	7	-1
ic_param: Number of parameters	0	0	5	0
lc_comm: Number of lines of comments	22	-oo	+oo	0
lc_stat: Number of statements	166	0	50	-1
mc_vg: McCabe cyclomatic number	82	0	15	-1
N1: Total number of operators	618	-oo	+oo	0
n1: Number of distinct operators	35	-oo	+oo	0
n2: Number of distinct operands	51	0	30	-1
N2: Total number of operands	445	-oo	+oo	0
struc_pg: Structuring	1	0	1	0

12. Close the Source file window in the **Result** window.

1.6.4 Factor Level

13. Select **Browse-Quality-Factor Level** or click on the Factor icon of the Logiscope Toolbar.
    A new tab titled **Factors** in the **Workspace** view appears.
14. Then browse through as for the Criteria level to discover the list of functions that have been rated “Poor” for MAINTAINABILITY.



```
void consistent()
{
  int x, y, z;

  for (z = 0; z < cur; z++)
    {
      dummy[z].blacks = dummy[z].whites =

      for (x = 0; x < 4; x++)
        guesses[cur].pegs[x].used = dummy[

      for (x = 0; x < 4; x++)
        if (guesses[cur].pegs[x].color ==
          {
            dummy[z].pegs[x].used = guesse
            dummy[z].blacks++;
          }
      if (dummy[z].blacks != 4)
        {
          for (y = 0; y < 4; y++)
```

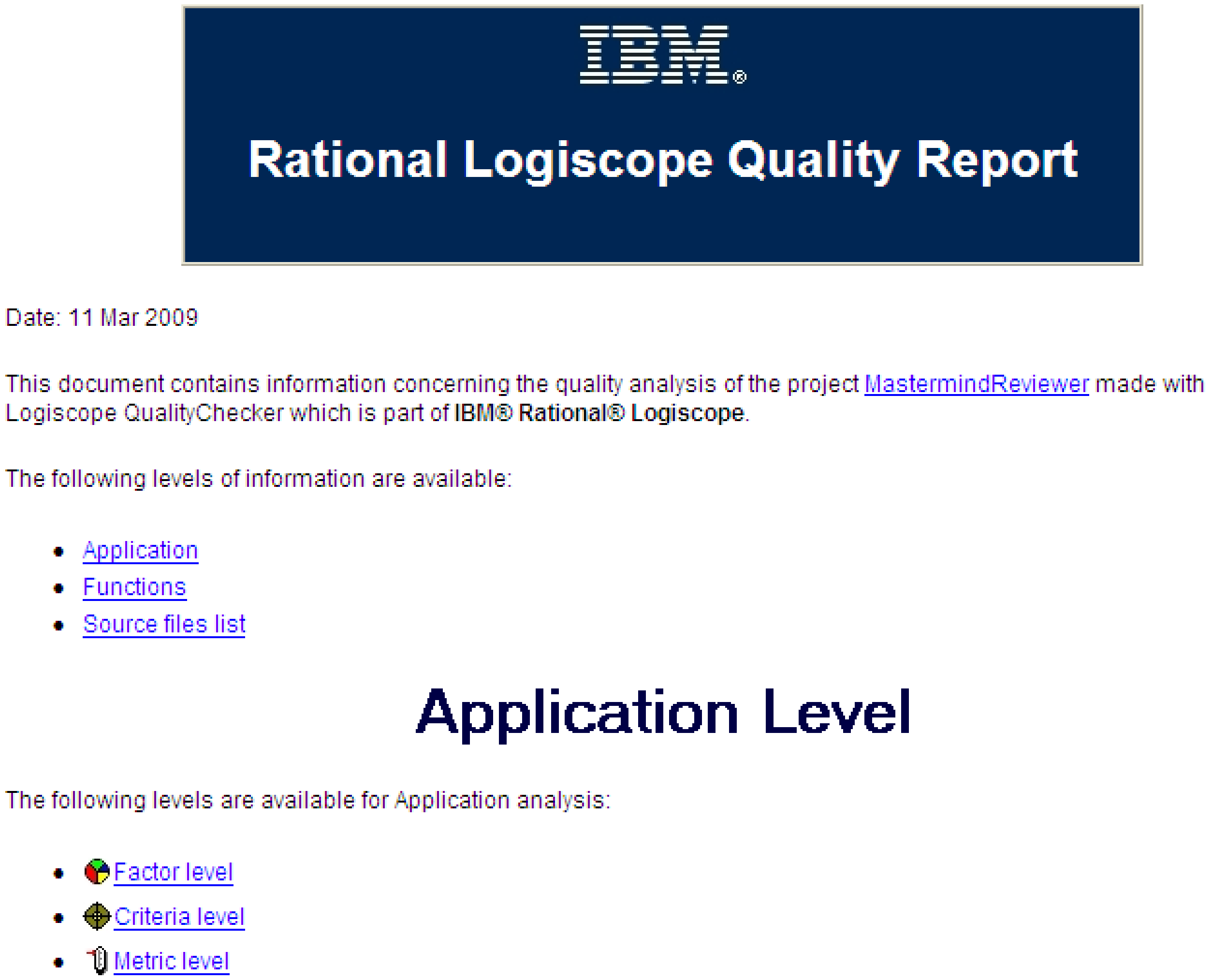
The 2 “POOR” rated functions for MAINTAINABILITY should be subject to a detailed code review to understand the reasons of this status and decide of the appropriate actions to help the code to reach a higher level of Maintainability.

The results provided by Logiscope Viewer can help in such a code review (see the next chapter).

1.6.5 Quality Report

15.Select the **Browse-Quality-Quality Report** or click on the icon.

The Logiscope Quality Report gives you an HTML synthesis at each level of the Application analysis performed with Logiscope *QualityChecker.*



IBM®

Rational Logiscope Quality Report

Date: 11 Mar 2009

This document contains information concerning the quality analysis of the project MastermindReviewer made with Logiscope QualityChecker which is part of **IBM® Rational® Logiscope**.

The following levels of information are available:

- Application
- Functions
- Source files list

Application Level

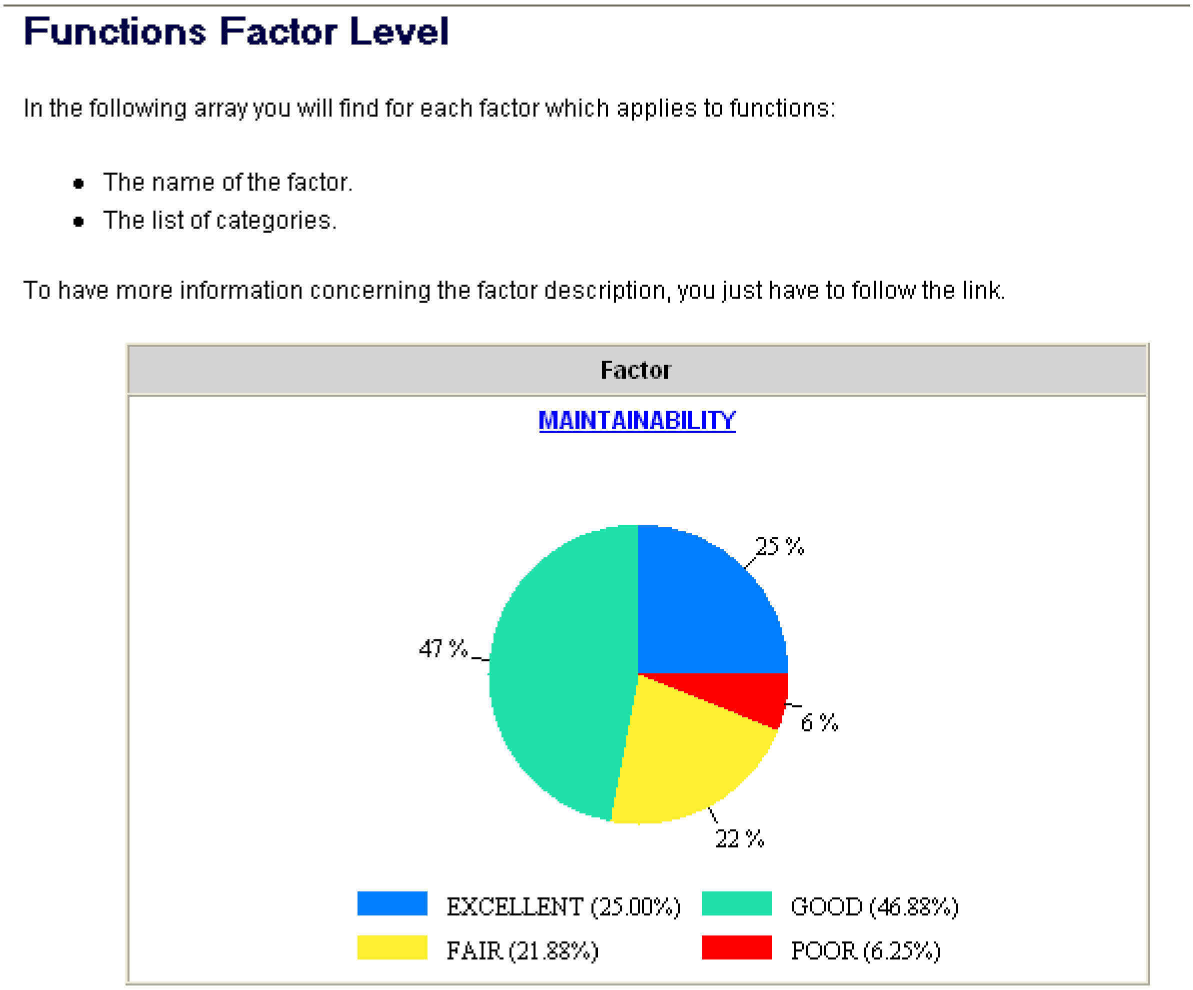
The following levels are available for Application analysis:

- Factor level
- Criteria level
- Metric level

Use the **HTML Browser Toolbar** to navigate back and forth in the report. It is possible to save the Quality Report by using **File-Save As**... Therefore it can be edited on an intranet in order to be accessible by everyone in the company.

16.Click on the Functions link: the Quality Report is updated.

17.Click on the Factor level link, you get the following graph:



Functions Factor Level

In the following array you will find for each factor which applies to functions:

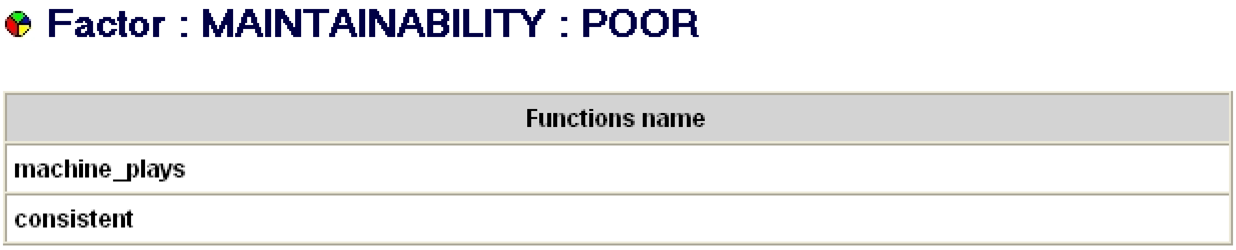
- The name of the factor.
- The list of categories.

To have more information concerning the factor description, you just have to follow the link.

Factor
MAINTAINABILITY

Note: The pies presented in the following chapters are not available on UNIX. They are replaced by tables. To generate a report with pies, change HTML reports option in Tools-Options... command (you will see the pies in your favorite Internet navigator).

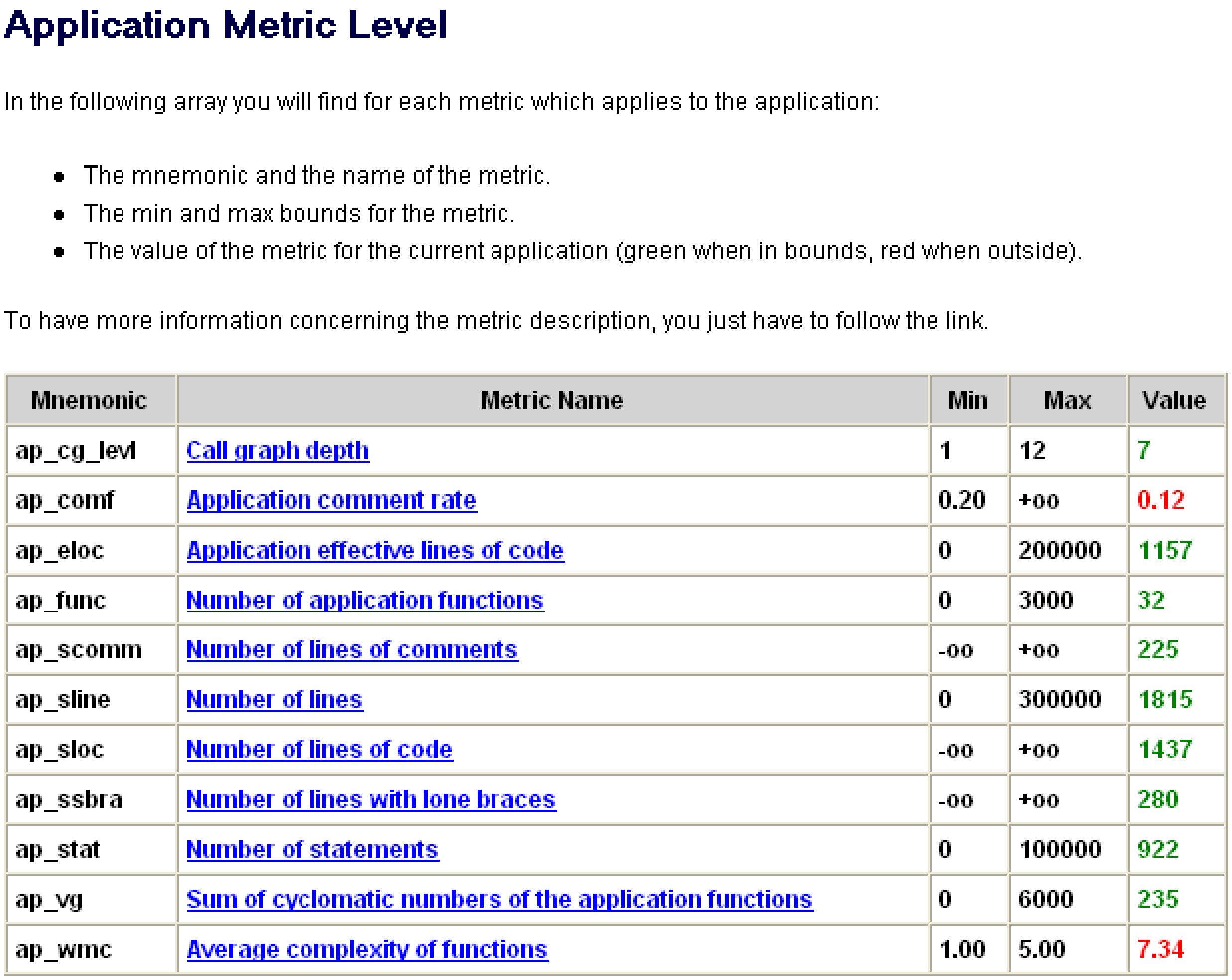
18.Click on the “POOR” part (i.e.: the red slice) in the MAINTAINABILITY pie and the Quality Report looks will list the “POOR” rated functions:



Factor : MAINTAINABILITY : POOR

Functions name
machine_plays
consistent

19. Click on the **Application Level** in the left frame of the Quality Report and select the **Application Metric Level.**



Application Metric Level

In the following array you will find for each metric which applies to the application:

- The mnemonic and the name of the metric.
- The min and max bounds for the metric.
- The value of the metric for the current application (green when in bounds, red when outside).

To have more information concerning the metric description, you just have to follow the link.

Mnemonic	Metric Name	Min	Max	Value
ap_cg_levl	Call graph depth	1	12	7
ap_comf	Application comment rate	0.20	+oo	0.12
ap_eloc	Application effective lines of code	0	200000	1157
ap_func	Number of application functions	0	3000	32
ap_scomm	Number of lines of comments	-oo	+oo	225
ap_sline	Number of lines	0	300000	1815
ap_sloc	Number of lines of code	-oo	+oo	1437
ap_ssbra	Number of lines with lone braces	-oo	+oo	280
ap_stat	Number of statements	0	100000	922
ap_vg	Sum of cyclomatic numbers of the application functions	0	6000	235
ap_wmc	Average complexity of functions	1.00	5.00	7.34

The values of the metrics selected at Application scope in the Quality Model are available in the Quality Report:

- Call graph depth,
- Effective lines of code
- Number of statements
- Cyclomatic number: total and average,
- etc.

Clicking on a Metric Name would provide the definition of the metric.

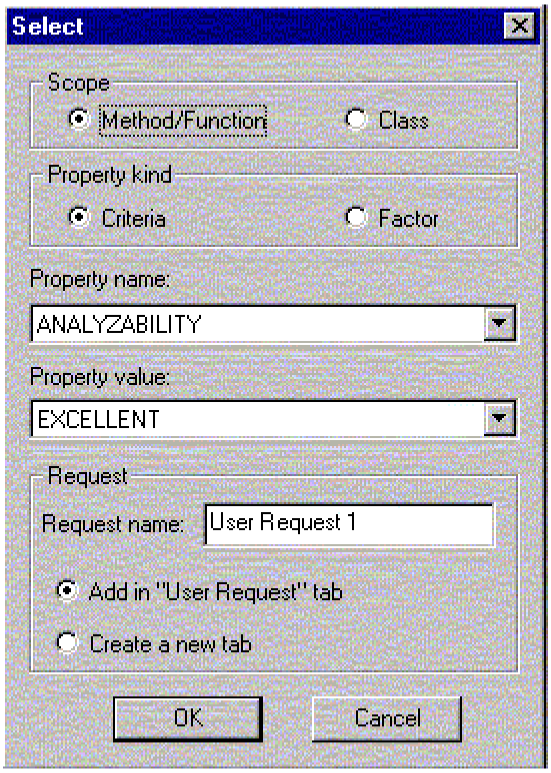
20.Close the Quality Report.

1.6.6 Select from Properties

Browse-Select from properties

The Select window appears and is used to build a request according to:

- a Scope: Method/Function or Class
- a Property kind: Criteria or Factor
- a Property name of criterion or factor
- a Property value: EXCELLENT, GOOD, FAIR or POOR
- a Request: rename the request User Request1 to "MyRequest", press OK and



Scope: ● Method/Function ○ Class

Property kind: ● Criteria ○ Factor

Property name: ANALYZABILITY

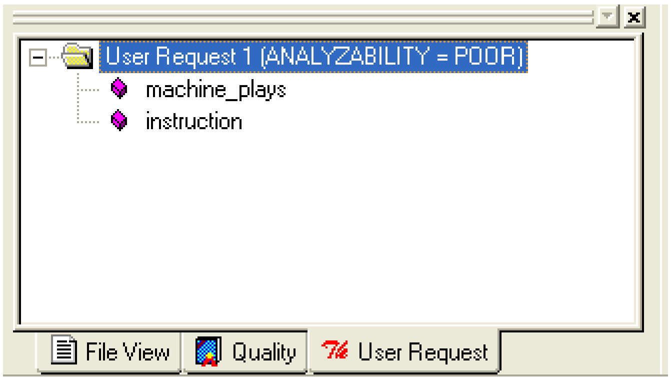
Property value: EXCELLENT

Request name: User Request 1

● Add in "User Request" tab

○ Create a new tab

21. Change the Property value: to "POOR" and click on OK. A new tab appears in the Workspace view with the list of functions satisfying the request.



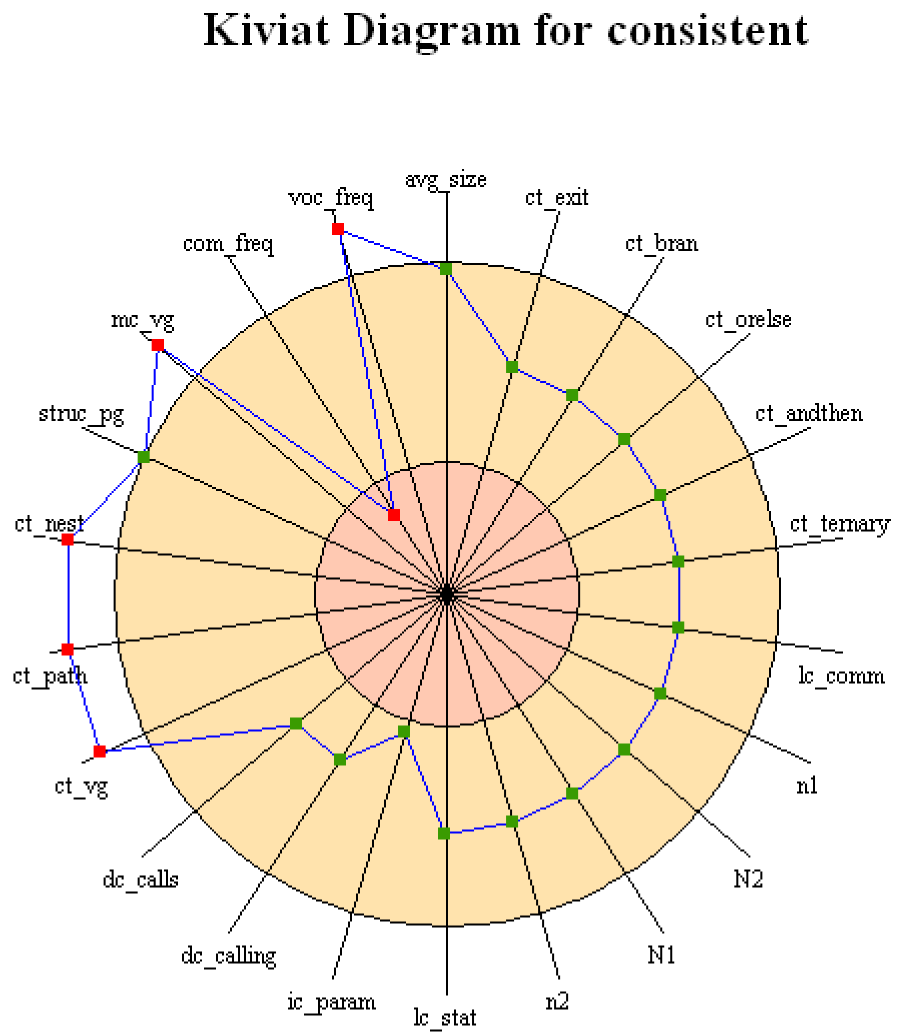
User Request 1 (ANALYZABILITY = POOR)
machine_plays
instruction

A double-click on any function will display the source file window and shows a metric list in the Output Window.

You can make as many requests as you like either by adding them in the **User Request** tab or by creating a new tab.

1.6.7 Kiviat Diagram (only available on Windows)

22. Click on the **Factor** tab of the **Workspace** view. Expand successively the **Functions, MAINTAINABILITY, POOR tree** nodes and select the function **consistent.**
23. Select the **View-Kiviat graph** menu:



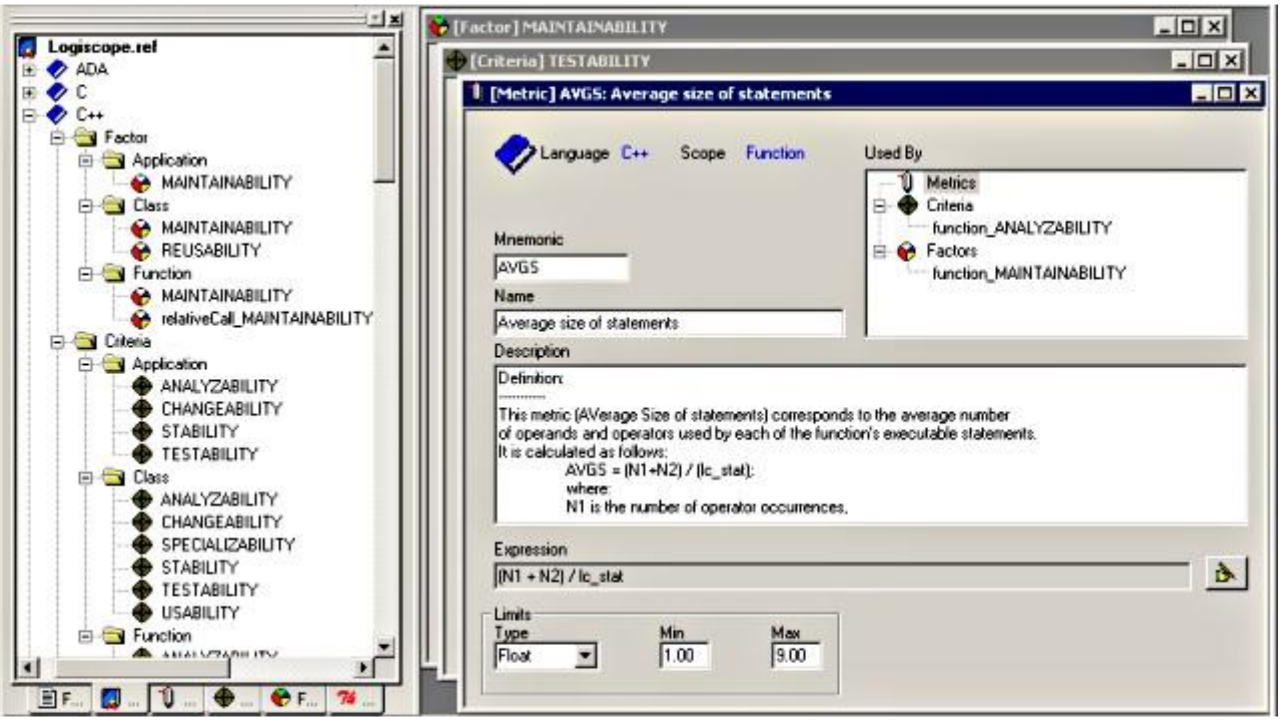
This graph represents all metric values. The outer circle symbolizes upper bounds and the inner circle symbolizes lower bounds.

The **Kiviat** view is also available for the application and the functions as well as from the **Factors** tab of the **Workspace View**.

1.6.8 Quality Model Editor

24.Click on the **File View** tab in the **Workspace View** and then double-click on the quality model file: i.e. the **Logiscope.ref** file associated to the icon .

The Quality Model editor is displayed together with the metrics, criteria and factors used in the Quality Model for the selected language.



[Factor] MAINTAINABILITY

[Criteria] TESTABILITY

[Metric] AVGS: Average size of statements

Language C++ Scope Function

Mnemonic: AVGS

Name: Average size of statements

Description:

```
Definition:
-----------
This metric (AVerage Size of statements) corresponds to the average number
of operands and operators used by each of the function's executable statements.
It is calculated as follows:
            AVGS = (N1+N2) / (lc_stat);
            where:
            N1 is the number of operator occurrences.
```

Expression: (N1 + N2) / lc_stat

Limits: Type Float; Min 1.00; Max 9.00

This editor is used to modify or create new metrics, criteria and factors which are then taken into account during the analysis of a *QualityChecker* project.

Before starting to modify a Quality Model, it is highly recommended to make a copy of the corresponding file. Some default Quality Models can be found in the directory **Ref** in the Logiscope installation directory.

In this view, it is possible to add a new factor, criteria or metric. Select the chosen language, right-click and select the new functionality required among:

New Application Factor
New Module Factor
New Class Factor
New Function Factor
New Application Criteria
New Module Criteria
New Class Criteria
New Function Criteria

New Application Metric
New Module Metric
New Class Metric
New Function Metric

A dialog box is then opened allowing to supply name, description, expression, categories and other items depending on the context.

The result pane may have to be enlarged to display all information because there is no scroll bar.

To add a new category in a factor or a criterion, select the top category in the Categories field and press the Insert key on the keyboard: a new category is inserted at the top, its name and limits can then be changed. Watch the information that may be displayed in the Messages View.

To suppress a category, select it and press the Delete/Suppr key on the keyboard.

1.7 Conclusion

You are now able to build a Logiscope project, you can check programming rules and source code metric analysis results of the project using the Logiscope **Studio**.

You can either quit Logiscope **Studio**:

25.Select **File-Exit;**

or learn how to view some more *QualityChecker* results using Logiscope **Viewer** in the next chapter.

Chapter 2

A Viewer Guided Tour

Now that you know how to create and build Logiscope *QualityChecker* projects, you can start the second phase: result exploration with Logiscope *Viewer.* You will see and learn how to display results with Logiscope *Viewer* such as:

- the Control Graph of a selected function,
- the Metrics Kiviat Graph of a selected function,
- the Criteria Kiviat Graph of a selected function,
- the Quality Report.

2.1 Starting a Logiscope *Viewer* Session

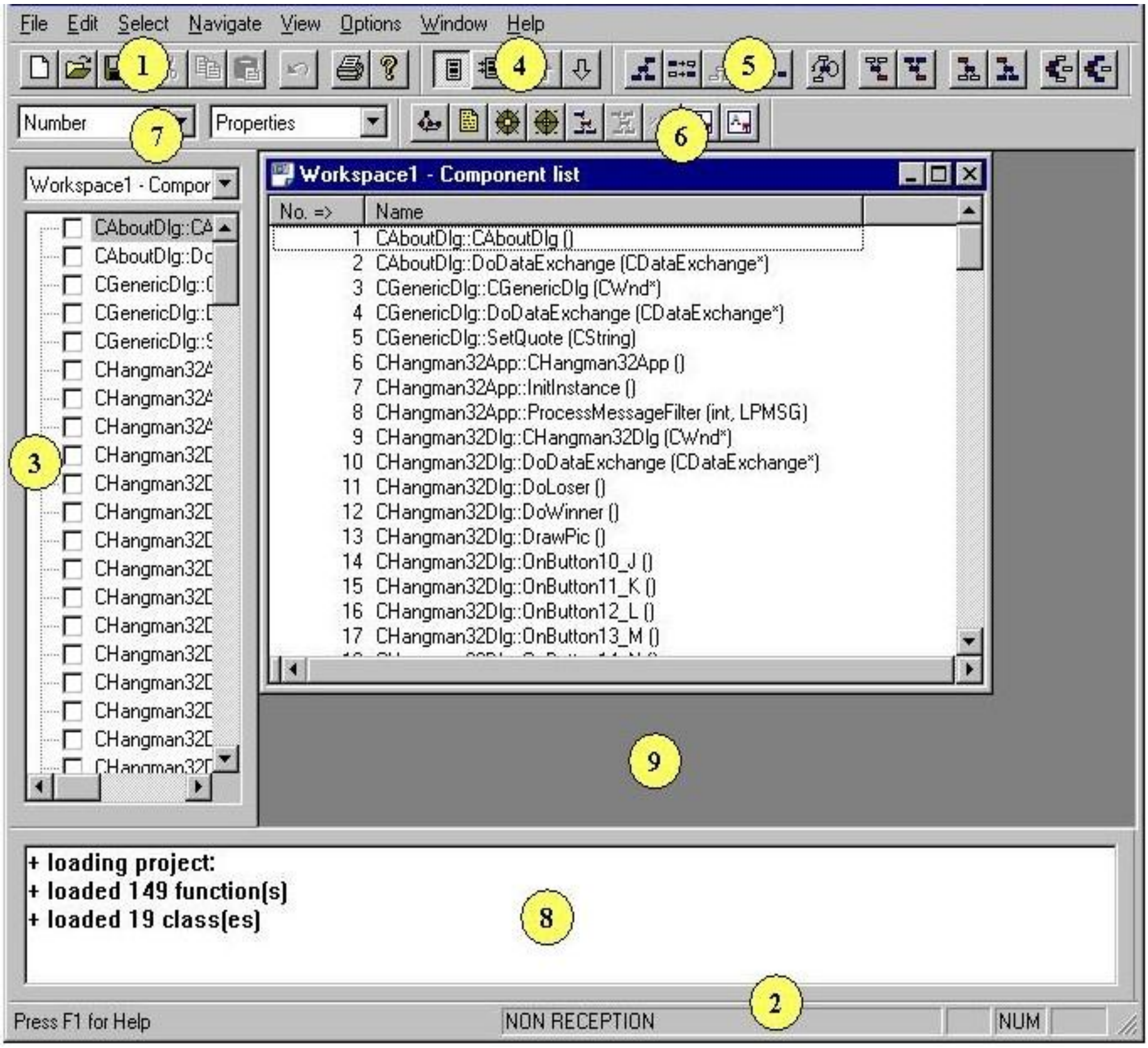
To begin a Logiscope **Viewer** session:

- from the **Studio** with an active project: **Project-Start Viewer** command.
- On UNIX (i.e. Solaris or Linux):
  - launch the **lgviewer** binary
- On Windows:
  - click the **Start** button, select the **Kalimetrix** Programs Group, then **Kalimetrix Tools,** then **Kalimetrix Logiscope 2014** and at last, **Logiscope Viewer**.

The Logiscope splash screen is first displayed and then the Logiscope **Viewer** main window appears.

If you do not launch Logiscope Viewer from Studio, open the Logiscope project you built in Chapter 2 (the “**.ttp”** file) with the **File-Open** command.

The Logiscope **Viewer** main window looks as follows:


File Edit Select Navigate View Options Window Help
Number
Properties
Workspace1 - Compor
Workspace1 - Component list
Press F1 for Help
NON RECEPTION
NUM
1
2
3
4
5
6
7
8
9

No. =>	Name
1	CAboutDlg::CAboutDlg ()
2	CAboutDlg::DoDataExchange (CDataExchange*)
3	CGenericDlg::CGenericDlg (CWnd*)
4	CGenericDlg::DoDataExchange (CDataExchange*)
5	CGenericDlg::SetQuote (CString)
6	CHangman32App::CHangman32App ()
7	CHangman32App::InitInstance ()
8	CHangman32App::ProcessMessageFilter (int, LPMSG)
9	CHangman32Dlg::CHangman32Dlg (CWnd*)
10	CHangman32Dlg::DoDataExchange (CDataExchange*)
11	CHangman32Dlg::DoLoser ()
12	CHangman32Dlg::DoWinner ()
13	CHangman32Dlg::DrawPic ()
14	CHangman32Dlg::OnButton10_J ()
15	CHangman32Dlg::OnButton11_K ()
16	CHangman32Dlg::OnButton12_L ()
17	CHangman32Dlg::OnButton13_M ()

```
+ loading project:
+ loaded 149 function(s)
+ loaded 19 class(es)
```

This window contains the following elements:

1. **Toolbar:**
   Provides shortcuts for the most commonly used commands of the **File** and **Edit** menus.
2. **Status Bar**
   Mostly indicates the status (RECEPTION, NON RECEPTION) of the active window displayed in the Result Pane.
3. **Control Palette: Workspace1-Component list Window**
   Displays a view of the components after loading a Logiscope project. Select or deselect the one you want to explore.
4. **Navigation bar**
   Provides shortcuts for the commands of the **Navigate** menu.

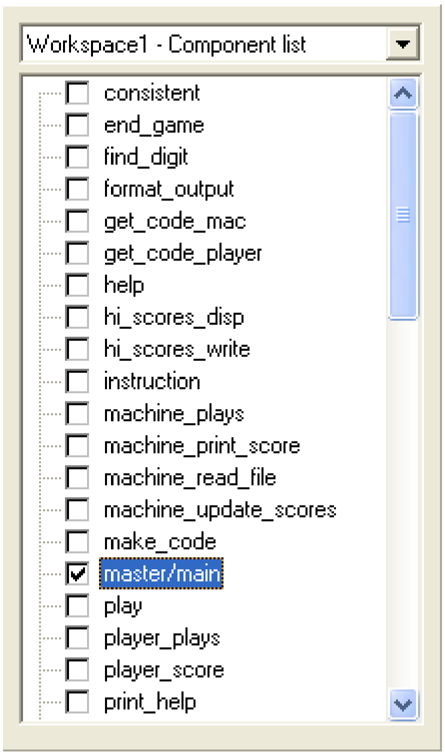
5. **Selection Bar**
   Provides shortcuts for the most commonly used commands of the **Select** menu.
6. **Component windows bar**
   Allows to display graphical results: control graph, source code, Metric and Criteria Kiviat graph, and to go to the Application window.
7. **Selector Bar**
   Use the selectors to choose and display additional information in the active Domain window. The content of the Selector bar depends on the view being displayed in the active Domain window.
8. **Messages window**
   Displays error messages or indications on loading the project.
9. **Result Pane**
   Used to display **Window** command results.

Use the **View** menu to show or hide some items described above and to customize the **Viewer** main window as you wish.

2.2 Selecting/Deselecting a function

You can indifferently use either the **Control Palette** or the **Workspace1-Component list** Domain window in the **Result Pane** to select or deselect functions.

1. In the **Control Palette,** click the function **master/main**.



Workspace1 - Component list

- ☐ consistent
- ☐ end_game
- ☐ find_digit
- ☐ format_output
- ☐ get_code_mac
- ☐ get_code_player
- ☐ help
- ☐ hi_scores_disp
- ☐ hi_scores_write
- ☐ instruction
- ☐ machine_plays
- ☐ machine_print_score
- ☐ machine_read_file
- ☐ machine_update_scores
- ☐ make_code
- ☑ master/main
- ☐ play
- ☐ player_plays
- ☐ player_score
- ☐ print_help

The *master/main* function is now also selected in the **Workspace1-Component list** Domain window.

2. Click now on the *help* component in the **Workspace1-Component list** Domain window.

The *master/main* function is now also deselected**.** Only the *help* function remains selected: i.e. ticked.

If you want to select several components, you can:

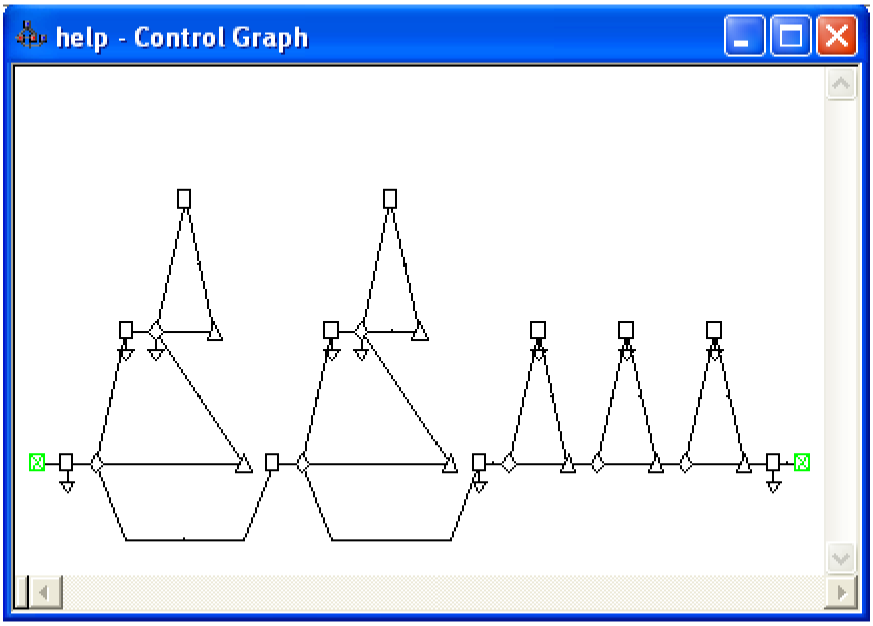
- either use the **Control Palette**,
- or use `<Ctrl>` + click mode in the **Domain** window.

2.3 Viewing the first results

2.3.1 Control Graph

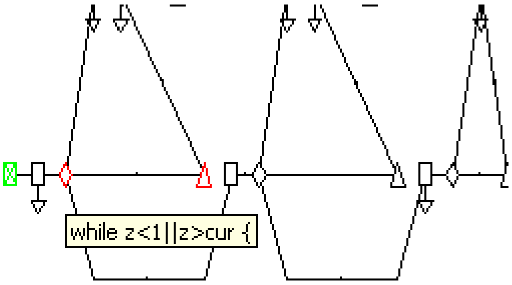
You want to explore further into the function and get an inside view of it? Well, the control graph is the map you need now. It is the graphical representation of a particular function in a program.

1. Click on the Control Graph icon or select the **Window-Control** command. A new window called **help-Control Graph** is displayed in the **Result Pane**.



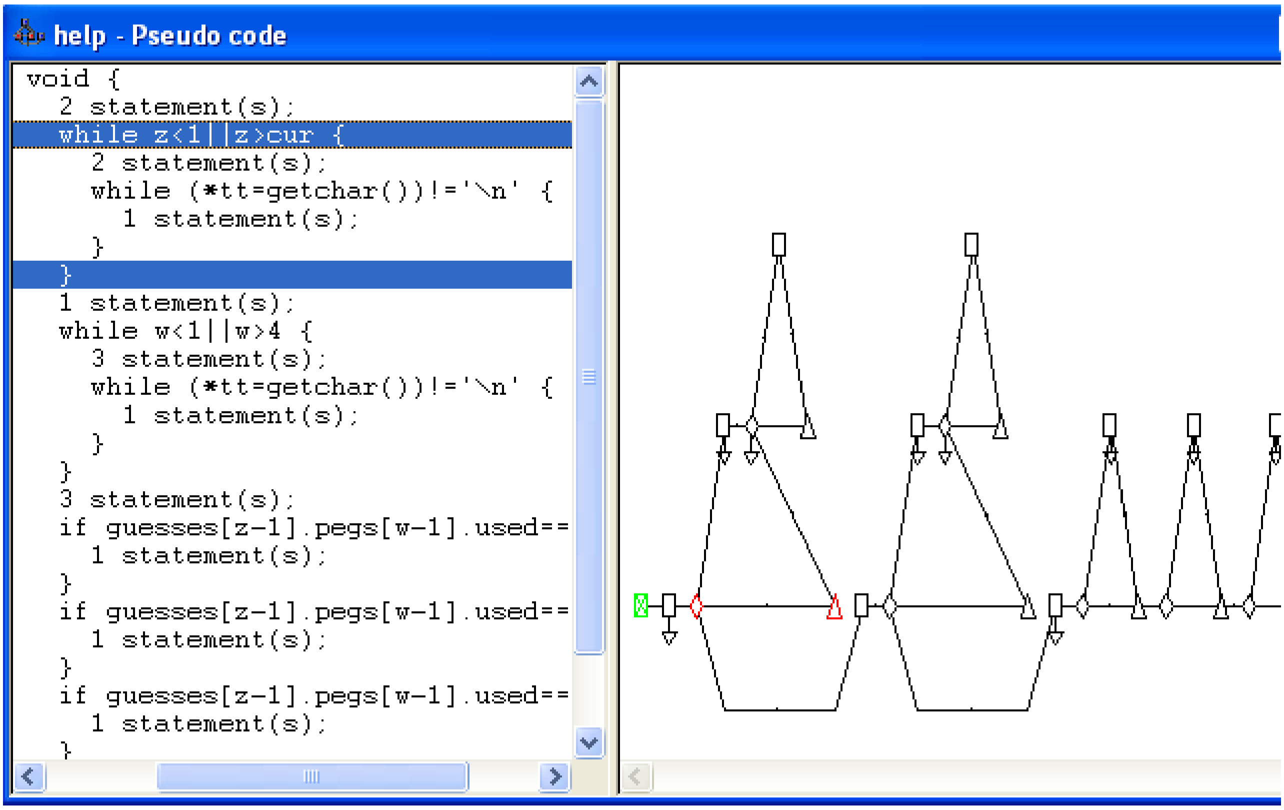
You can see the representation featuring geometric symbols (nodes) linked by arrows (edges). From experience you may have already guessed that the control graph represents the selected function's logical structure. If you are a newcomer to Logiscope *QualityChecker,* this kind of representation may mean nothing to you yet. Do not worry, you will quickly become familiar with it.

2. Place the cursor over the first diamond-shaped node in the lower left side of the graph. Some information now appears on the Control Graph, as illustrated below:



It represents the pseudocode linked with this node.

3. Now, select the **Window-Split** command and move the cursor from left to right to display the pseudocode associated with the control graph.



```
void {
  2 statement(s);
  while z<1||z>cur {
    2 statement(s);
    while (*tt=getchar())!='\n' {
      1 statement(s);
    }
  }
  1 statement(s);
  while w<1||w>4 {
    3 statement(s);
    while (*tt=getchar())!='\n' {
      1 statement(s);
    }
  }
  3 statement(s);
  if guesses[z-1].pegs[w-1].used==
    1 statement(s);
  }
  if guesses[z-1].pegs[w-1].used==
    1 statement(s);
  }
  if guesses[z-1].pegs[w-1].used==
    1 statement(s);
  }
```

Now it becomes crystal clear: each geometric symbol (node) represents one or more control statements. The diamond you have selected indicated a *for* statement.
Selection propagation works the other way too, from pseudocode to control graph.

For more details on the control graph representation, please refer to the *Kalimetrix Logiscope Basic Concepts* manual.

2.3.2 The Source Code

Logiscope **Viewer** can also display the function source code.

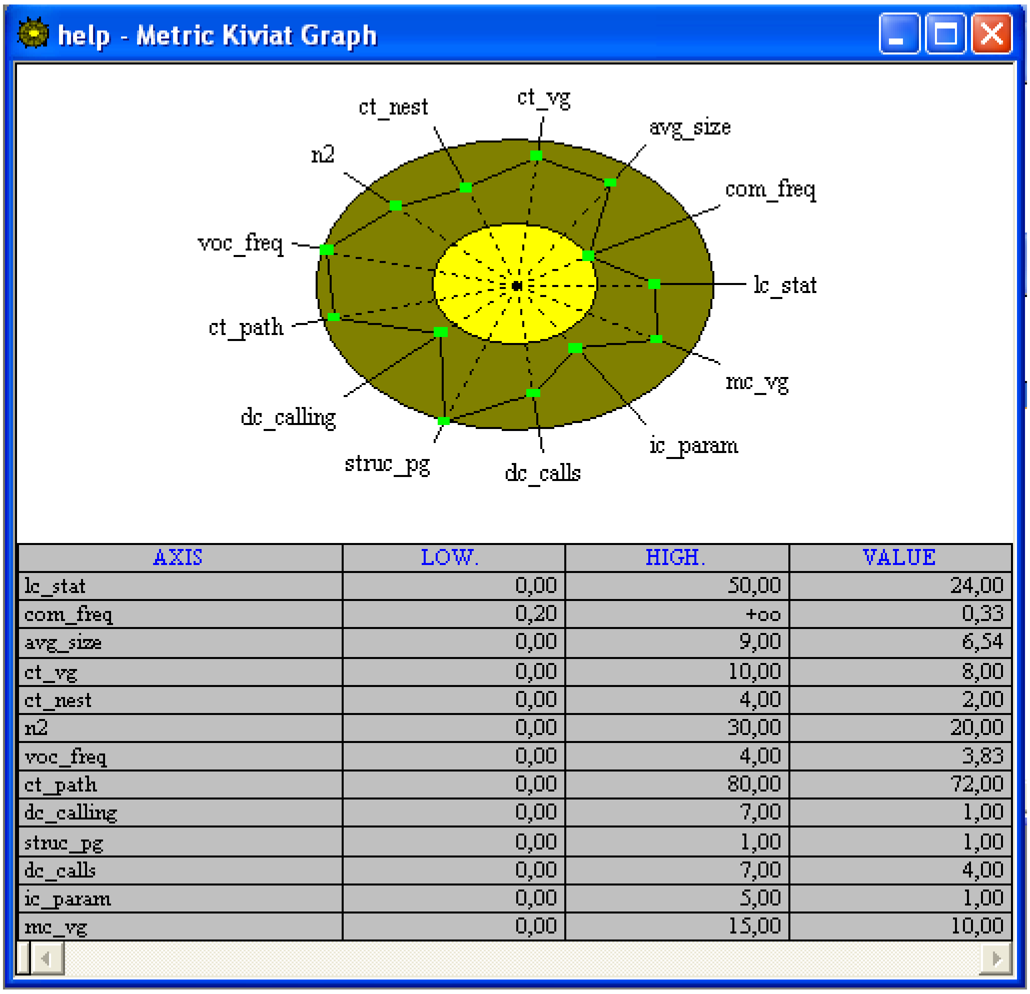
4. Click on the Source Code icon, or select the **Window - Source** command.

The Source window pops up and displays the content of the *player.c* source file in which the function *help* is defined. The first line of the function definition is shown in reverse video.
Clicking on a node in the control graph moves the selection to the corresponding statement(s) in the Source window.

2.3.3 Metric Kiviat Graph

Now, request the Metric Kiviat graph to check whether the *help* function meets C quality requirements.

5. Click on the icon or select the **Window - Metric Kiviat** command.
   A new window named **help -Metric Kiviat Graph** is displayed in the **Result Pane.**



AXIS	LOW.	HIGH.	VALUE
lc_stat	0,00	50,00	24,00
com_freq	0,20	+oo	0,33
avg_size	0,00	9,00	6,54
ct_vg	0,00	10,00	8,00
ct_nest	0,00	4,00	2,00
n2	0,00	30,00	20,00
voc_freq	0,00	4,00	3,83
ct_path	0,00	80,00	72,00
dc_calling	0,00	7,00	1,00
struc_pg	0,00	1,00	1,00
dc_calls	0,00	7,00	4,00
ic_param	0,00	5,00	1,00
mc_vg	0,00	15,00	10,00

If the information displayed in the Metric Kiviat Graph is difficult to read you can maximize the window by clicking on the button located in the upper right corner.

You can see two concentric circles and coordinate axes radiating from the graph center. Each axis represents one of the complexity metrics selected for assessing the function quality. For instance, the **lc_stat** axis corresponds to the number of executable statements contained in the function.

Circles represent lower and upper limit values for metrics represented in the graph. Boxes on the axes indicate the metric value for the selected function. If a box is out of the inner circle and inside the outer circle, then the metric value for the function being inspected is within acceptable limits (displayed in green).

From this graph, you can see that the function *help* has all metric figures within limits.

The table below the graph provides a list of metric values. For the metric **lc_stat:** Number of statements, the lower limit (LOW) is set to 1, the upper limit (HIGH) is 50 and the value measured is 24.

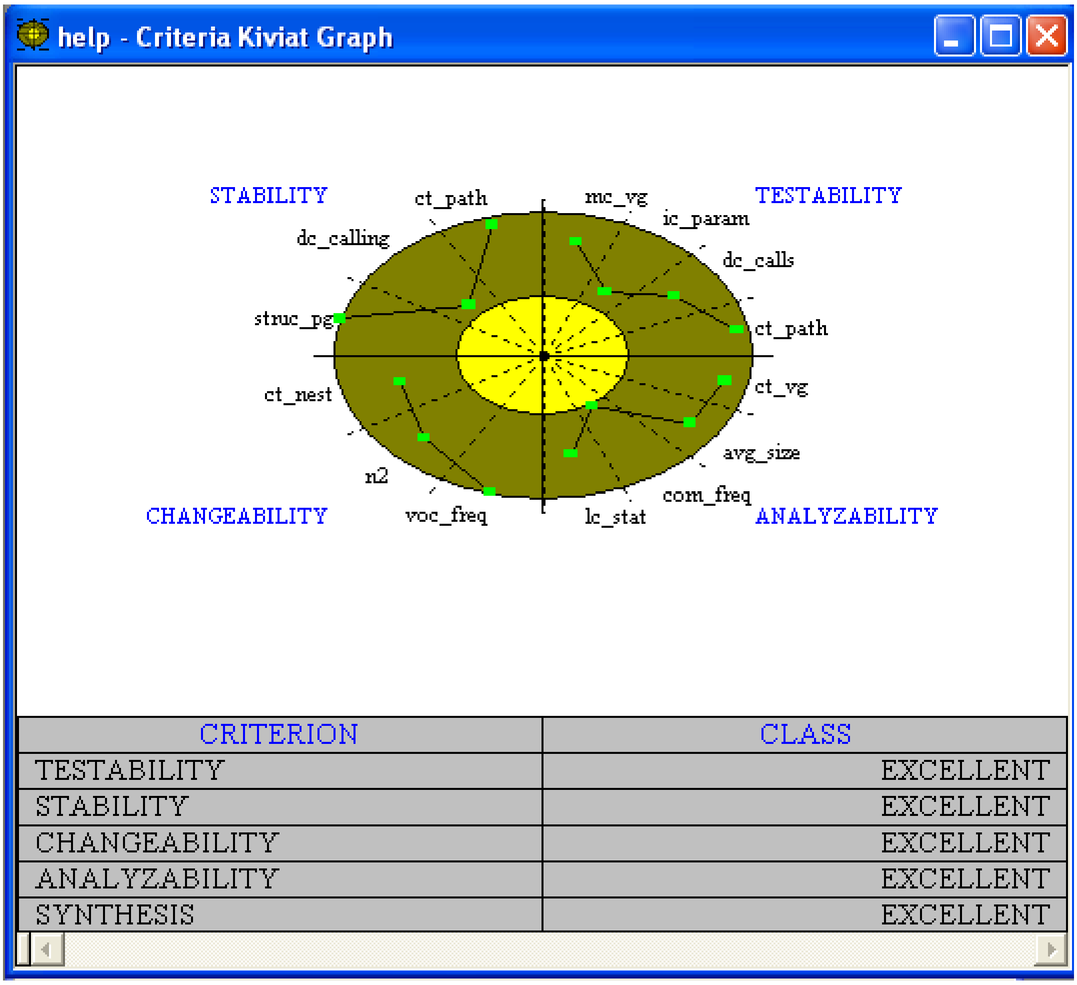
6. Select **View - Kiviat Table** command to view the selected metric's full name.

These measurements are of significant interest to developers, who use the Reference File (the first file you selected when creating your Logiscope project) to choose metrics they want to appear on the metric Kiviat graph. Lower and upper limits are also defined in this file and depend on programming standards used for the project.

2.3.4 Criteria Kiviat Graph

The **Criteria Kiviat Graph** is a graphical representation with a terminology closer to what is generally used to describe software quality.

7. Click on the ⊕ icon or select the **Window-Criteria Kiviat** command.
   A new window named **help-Criteria Kiviat Graph** is displayed in the Result Pane.



CRITERION	CLASS
TESTABILITY	EXCELLENT
STABILITY	EXCELLENT
CHANGEABILITY	EXCELLENT
ANALYZABILITY	EXCELLENT
SYNTHESIS	EXCELLENT

The Criteria Kiviat Graph is based on the same principle as the Metric Kiviat Diagram. Experience has shown that more than one metric should be used for evaluating quality criteria. For instance, to assess the extent to which the ANALYZABILITY criterion has been met, a combination of four different metrics is used:

- **ct_vg**: Cyclomatic number,
- **avg_size**: Average size of statements,
- **lc_stat**: Number of statements,
- **com_freq:** Comment frequency.

Below the graph, the table indicates the level achieved for the four evaluated criteria. This level depends on how many metrics are in the limits specified. The function *help* falls into the EXCELLENT category for all criteria: all of its metric results are within the limits specified. The bottom table row, labeled SYNTHESIS, gives a global result, a comprehensive idea of the degree to which the different quality criteria have been met for the *help* function.

For more details on the Quality Model, please refer to the *Kalimetrix Logiscope Basic Concepts* manual.

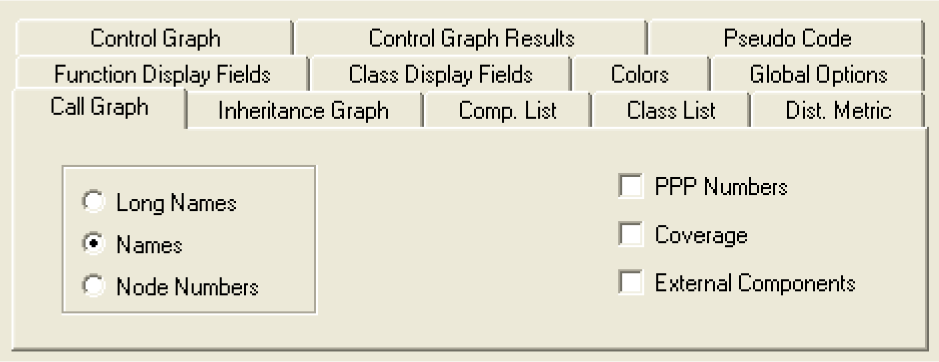
8. Close all windows related to the *help* function to clear the **Result Pane**. Be careful **not to** close the **Workspace1-Component list** window.

9. Click on the icon in the **Selection Bar** or choose the **Select-Deselect** command to deselect all functions on the Component List.

2.3.5 Preferences

You can customize the look of the **Viewer** graphs. Follow the next instructions:

1. Select the **File-Preferences** menu.

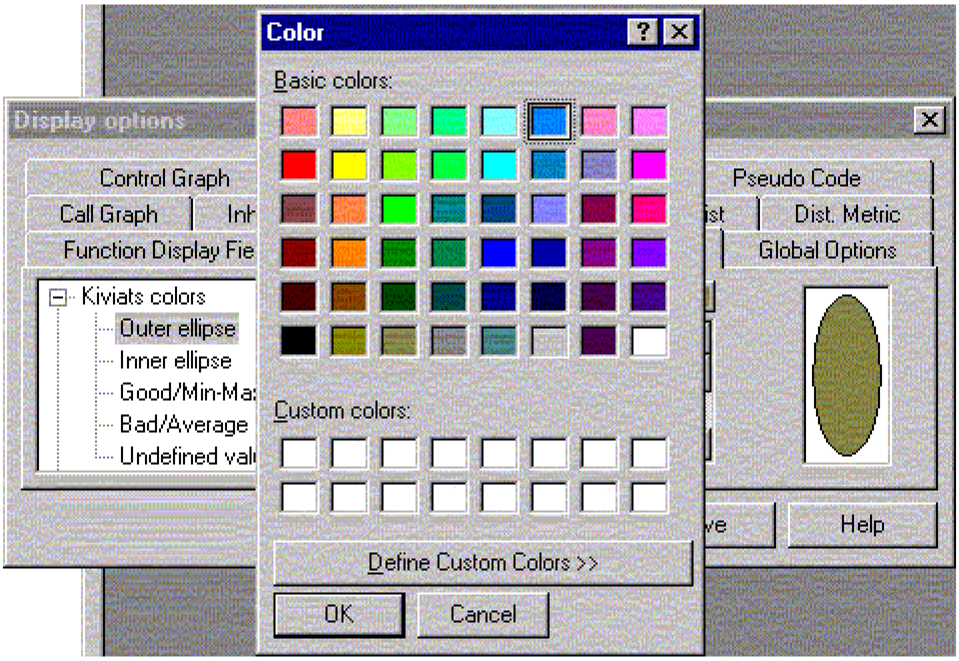


○ Long Names
● Names
○ Node Numbers

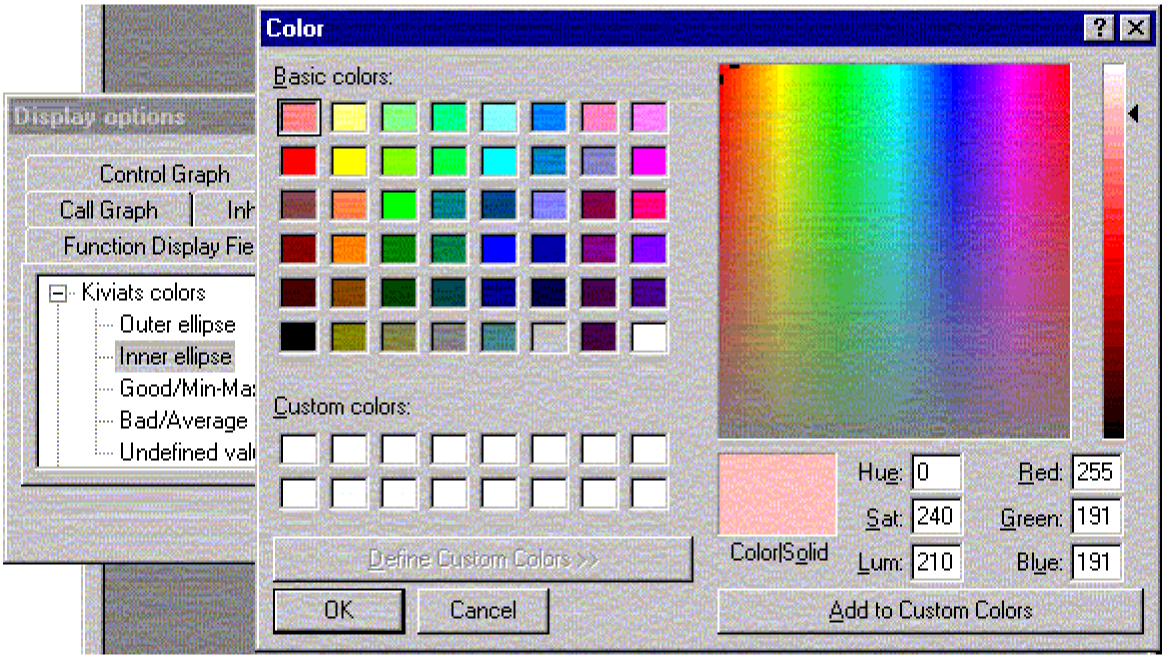
□ PPP Numbers
□ Coverage
□ External Components

2. Click the **Colors** tab.
3. Open the **Kiviats colors** option.

4. Change the color of the outer ellipse (into Blue in this example) by clicking on the **Brush** button:



5. Change the color of the inner ellipse (Pink) by clicking on the **Brush** and **Define Custom Colors.** Use the arrow on the right to define a new color, click **OK** when satisfied.



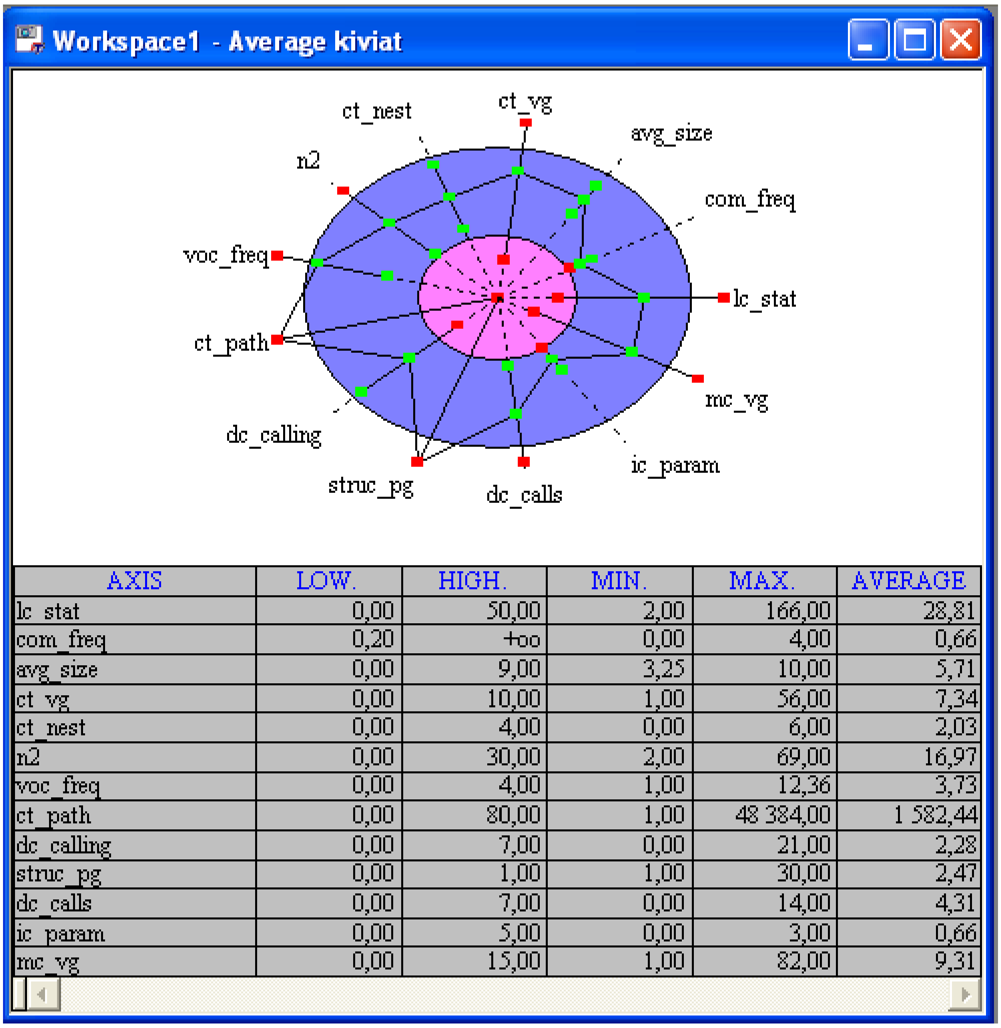
Hue: 0
Sat: 240
Lum: 210
Red: 255
Green: 191
Blue: 191

6. Click **OK** in the Preferences window to confirm and if you want to keep the new defined Kiviat Colors for good, click **Save** first.

2.3.6 Average Kiviat Graph

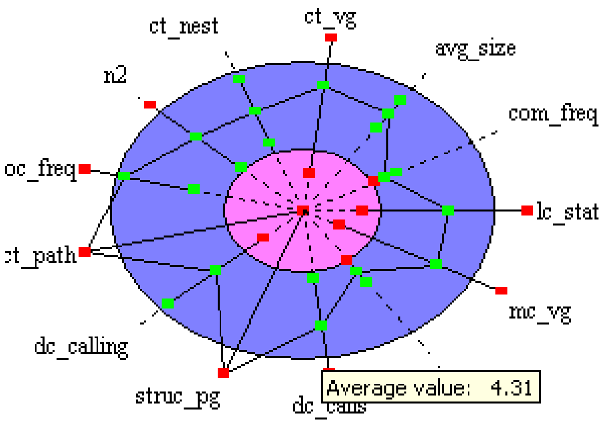
This Kiviat graph displays average and standard deviation values for all analyzed component metrics

1. Select the **View-Average Kiviat,** you get the following graph**:**

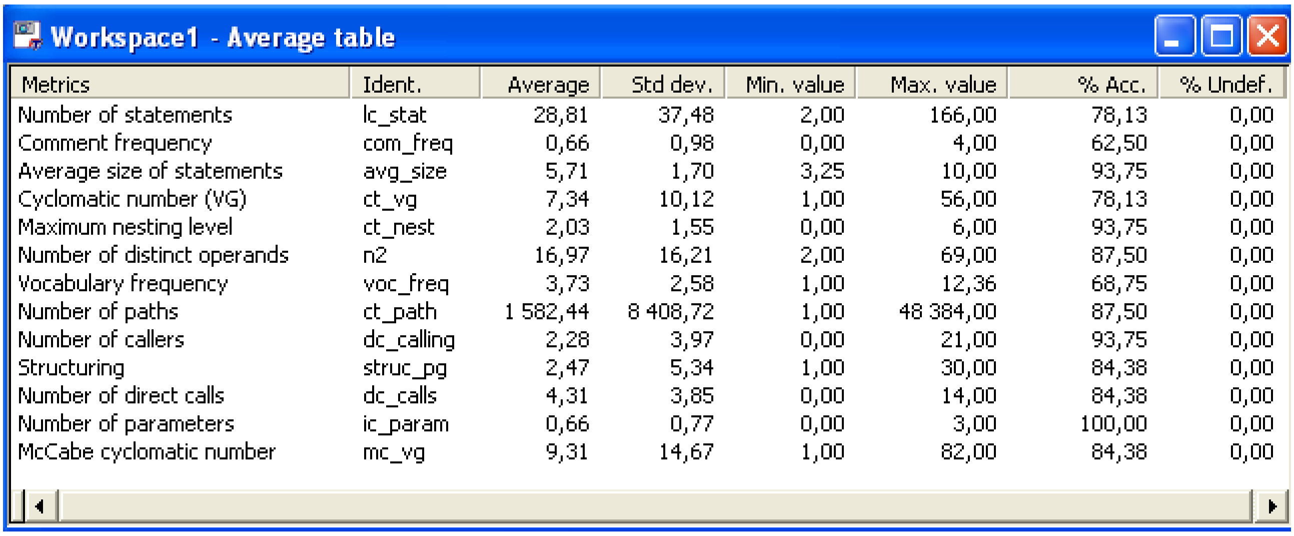


AXIS	LOW.	HIGH.	MIN.	MAX.	AVERAGE
lc stat	0,00	50,00	2,00	166,00	28,81
com_freq	0,20	+oo	0,00	4,00	0,66
avg_size	0,00	9,00	3,25	10,00	5,71
ct vg	0,00	10,00	1,00	56,00	7,34
ct_nest	0,00	4,00	0,00	6,00	2,03
n2	0,00	30,00	2,00	69,00	16,97
voc_freq	0,00	4,00	1,00	12,36	3,73
ct_path	0,00	80,00	1,00	48 384,00	1 582,44
dc_calling	0,00	7,00	0,00	21,00	2,28
struc_pg	0,00	1,00	1,00	30,00	2,47
dc_calls	0,00	7,00	0,00	14,00	4,31
ic param	0,00	5,00	0,00	3,00	0,66
mc_vg	0,00	15,00	1,00	82,00	9,31

2. Move the mouse close to the metric boxes to get the corresponding values:



3. Select the **View-Average Table** to see computed values for each metric:



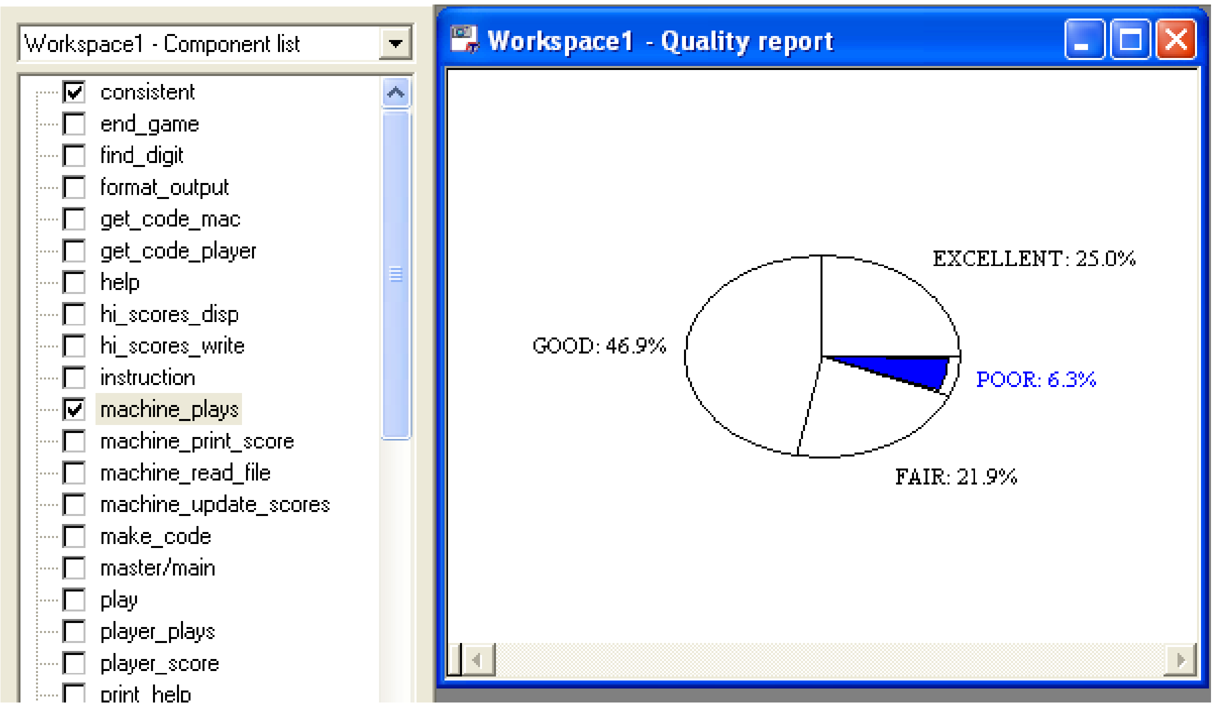
Workspace1 - Average table

Metrics	Ident.	Average	Std dev.	Min. value	Max. value	% Acc.	% Undef.
Number of statements	lc_stat	28,81	37,48	2,00	166,00	78,13	0,00
Comment frequency	com_freq	0,66	0,98	0,00	4,00	62,50	0,00
Average size of statements	avg_size	5,71	1,70	3,25	10,00	93,75	0,00
Cyclomatic number (VG)	ct_vg	7,34	10,12	1,00	56,00	78,13	0,00
Maximum nesting level	ct_nest	2,03	1,55	0,00	6,00	93,75	0,00
Number of distinct operands	n2	16,97	16,21	2,00	69,00	87,50	0,00
Vocabulary frequency	voc_freq	3,73	2,58	1,00	12,36	68,75	0,00
Number of paths	ct_path	1 582,44	8 408,72	1,00	48 384,00	87,50	0,00
Number of callers	dc_calling	2,28	3,97	0,00	21,00	93,75	0,00
Structuring	struc_pg	2,47	5,34	1,00	30,00	84,38	0,00
Number of direct calls	dc_calls	4,31	3,85	0,00	14,00	84,38	0,00
Number of parameters	ic_param	0,66	0,77	0,00	3,00	100,00	0,00
McCabe cyclomatic number	mc_vg	9,31	14,67	1,00	82,00	84,38	0,00

2.3.7 Quality Report

To get a global view of the whole program quality level: use the Quality Report.

1. Select the **View-Report** command from the menu bar. Note that a significant percentage (6.3%) of the *Mastermind* program functions belong to the "POOR" category.
2. Click on the “POOR” slice, all the components classified in this category are automatically selected in the **Control Palette**.

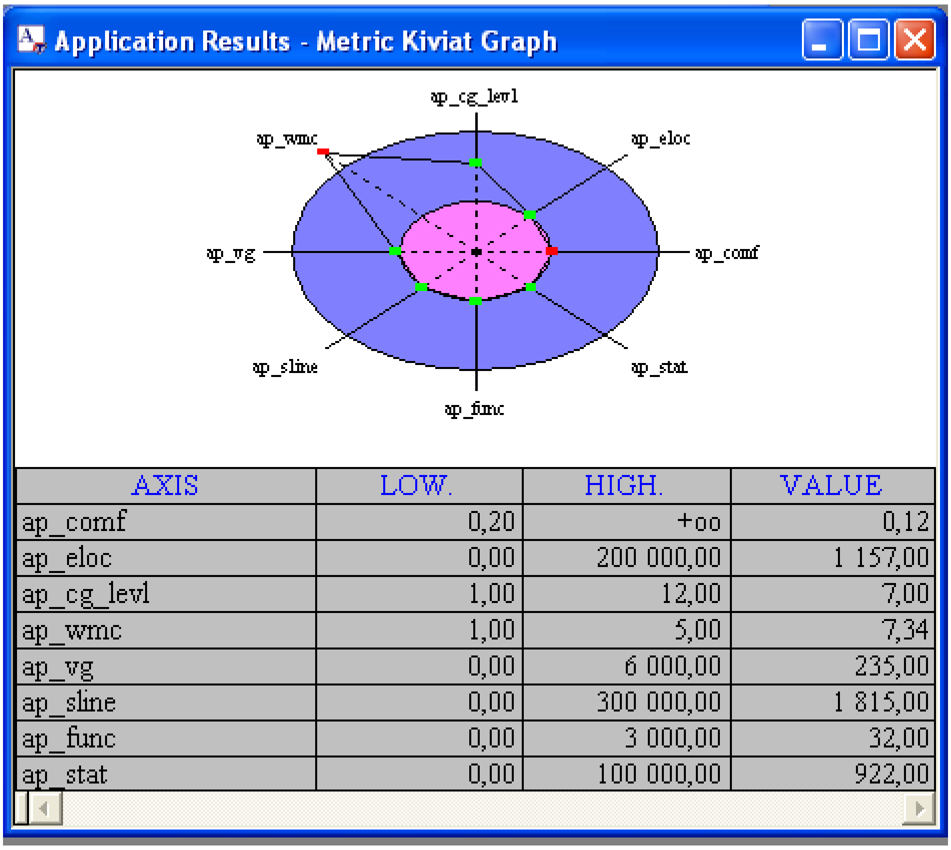


Workspace1 - Component list

- ☑ consistent
- ☐ end_game
- ☐ find_digit
- ☐ format_output
- ☐ get_code_mac
- ☐ get_code_player
- ☐ help
- ☐ hi_scores_disp
- ☐ hi_scores_write
- ☐ instruction
- ☑ machine_plays
- ☐ machine_print_score
- ☐ machine_read_file
- ☐ machine_update_scores
- ☐ make_code
- ☐ master/main
- ☐ play
- ☐ player_plays
- ☐ player_score
- ☐ print_help

2.3.8 Application Results

You can view the Application general results. Select **Window-Application** or click on the icon to get the following graph:



AXIS	LOW.	HIGH.	VALUE
ap_comf	0,20	+oo	0,12
ap_eloc	0,00	200 000,00	1 157,00
ap_cg_levl	1,00	12,00	7,00
ap_wmc	1,00	5,00	7,34
ap_vg	0,00	6 000,00	235,00
ap_sline	0,00	300 000,00	1 815,00
ap_func	0,00	3 000,00	32,00
ap_stat	0,00	100 000,00	922,00

From this graph, you can see the Application scope metric values.

Congratulations, you have just completed the first part of your Logiscope *Viewer* tour.

3. Close all windows in the **Result Pane.**
   The **Viewer** looks empty as no project workspace is activated.

Now you will learn how to get results about Classes.

2.4 Reviewing Classes

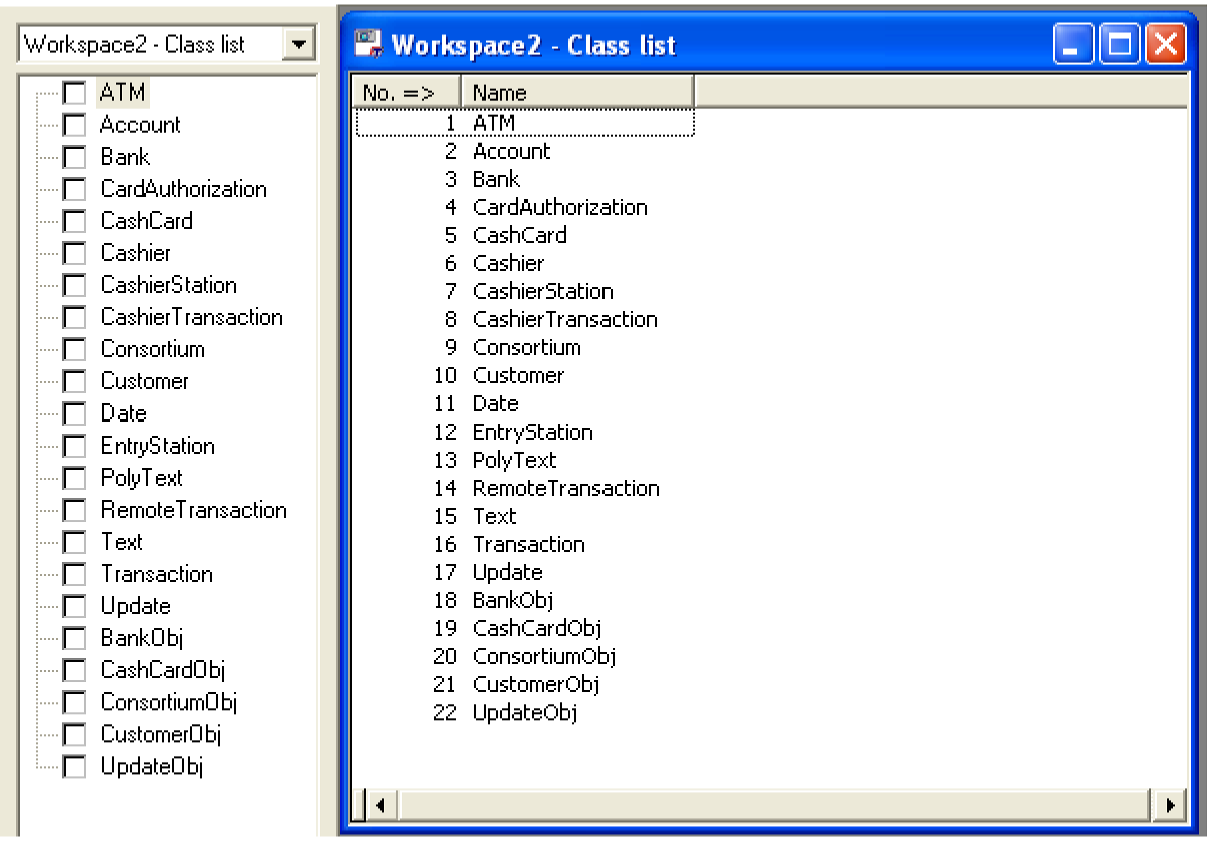
2.4.1 Opening an existing project

1. Select the **File-Open...** command.
2. In the **Open** window, browse to select the **ATMAudit.ttp** Logiscope project file in the **samples/C++/LogiscopeProjects** folder in the Logiscope installation directory.

2.4.2 Opening a Class List Window

1. First of all, click on the button to iconify the **Workspace-Component list** window, you will not need it.
2. Select the **File-New...** command.
3. In the New dialog box, select **Class Workspace** and click **OK**.

A window called **Workspace-Class list** is displayed. This window shows all classes of the *ATM* program:



- □ ATM
- □ Account
- □ Bank
- □ CardAuthorization
- □ CashCard
- □ Cashier
- □ CashierStation
- □ CashierTransaction
- □ Consortium
- □ Customer
- □ Date
- □ EntryStation
- □ PolyText
- □ RemoteTransaction
- □ Text
- □ Transaction
- □ Update
- □ BankObj
- □ CashCardObj
- □ ConsortiumObj
- □ CustomerObj
- □ UpdateObj

No. =>	Name
1	ATM
2	Account
3	Bank
4	CardAuthorization
5	CashCard
6	Cashier
7	CashierStation
8	CashierTransaction
9	Consortium
10	Customer
11	Date
12	EntryStation
13	PolyText
14	RemoteTransaction
15	Text
16	Transaction
17	Update
18	BankObj
19	CashCardObj
20	ConsortiumObj
21	CustomerObj
22	UpdateObj

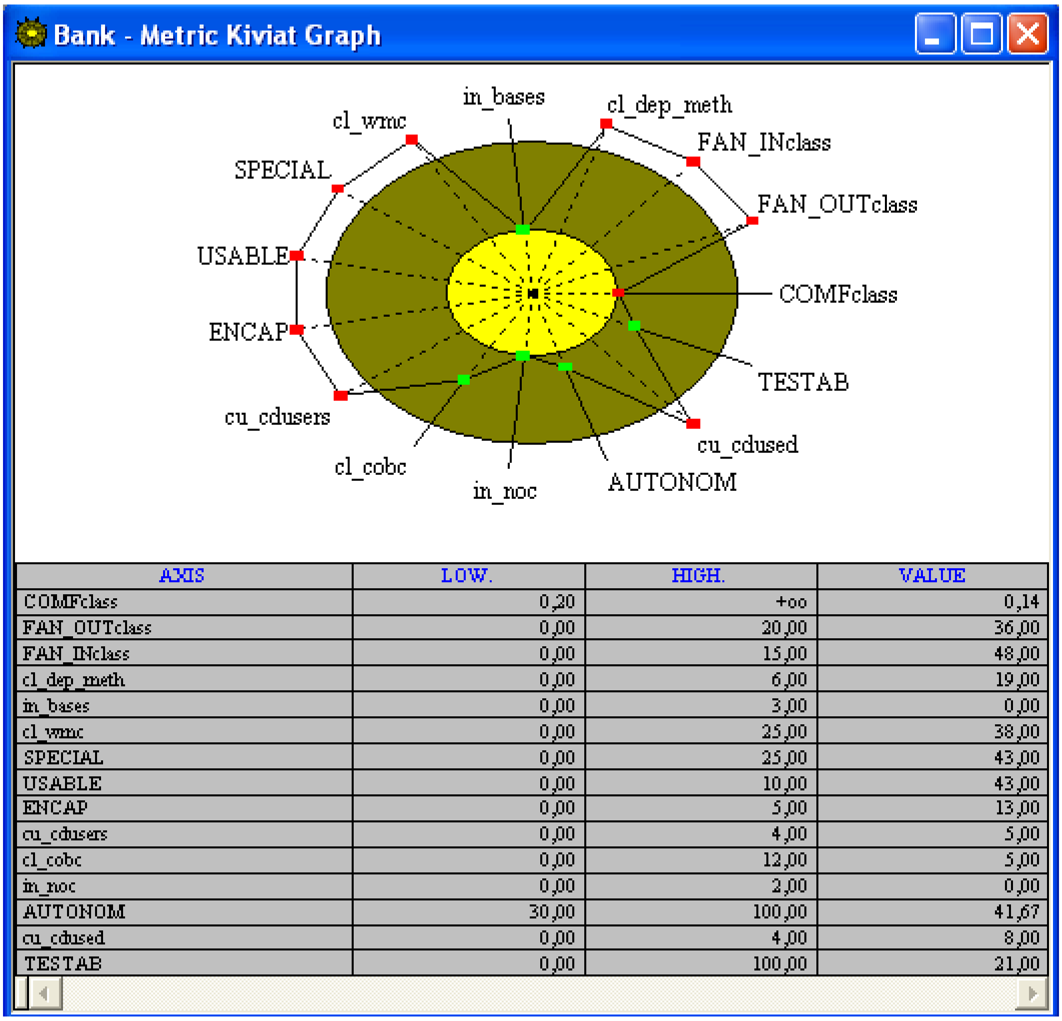
From this point on, and as long as you work on a class workspace, function-related features such as the control graph and coverage charts will remain unavailable.

2.4.3 Viewing Class Results

To find out about analysis results of a given class, proceed as follows:

1. From the **Control Palette**, click on the class named *Bank*. This class becomes also active in the **Workspace-Class list** window.

2. Click on the icon and the *Bank* Metric Kiviat graph is displayed.



AXIS	LOW.	HIGH.	VALUE
COMFclass	0,20	+oo	0,14
FAN_OUTclass	0,00	20,00	36,00
FAN_INclass	0,00	15,00	48,00
cl_dep_meth	0,00	6,00	19,00
in_bases	0,00	3,00	0,00
cl_wmc	0,00	25,00	38,00
SPECIAL	0,00	25,00	43,00
USABLE	0,00	10,00	43,00
ENCAP	0,00	5,00	13,00
cu_cdusers	0,00	4,00	5,00
cl_cobc	0,00	12,00	5,00
in_noc	0,00	2,00	0,00
AUTONOM	30,00	100,00	41,67
cu_cdused	0,00	4,00	8,00
TESTAB	0,00	100,00	21,00

This graph shows analysis results for object-oriented metrics defined in the Quality Model in use. These are different from component and application metrics explored before.

For more on object-oriented metrics, please refer to the *Kalimetrix Logiscope Basic Concepts* manual.

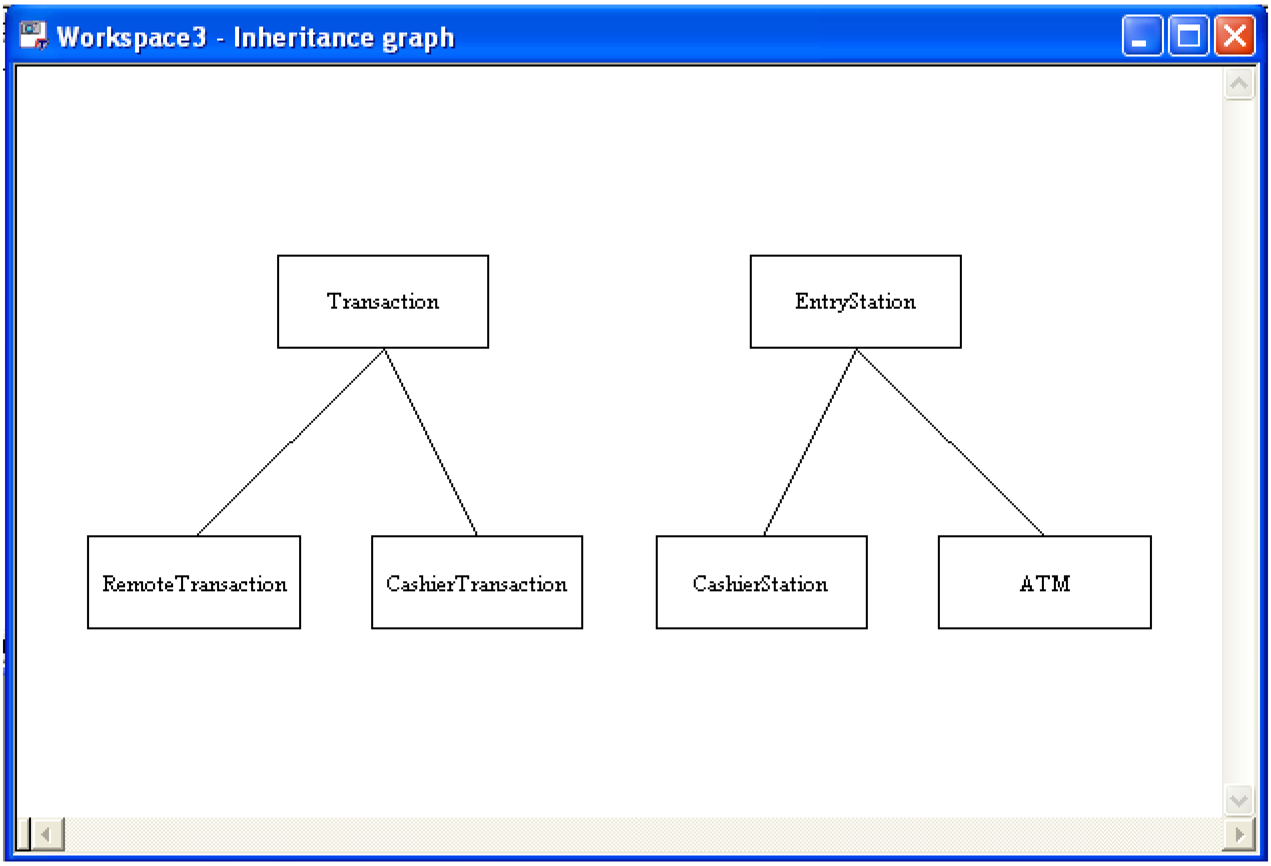
3. To get a source free space, close the Metric Kiviat graph window by clicking on the icon.

2.4.4 The Inheritance Graph

The inheritance graph is a powerful feature for analyzing inheritance relationships between classes of an application.

4. Make sure the Workspace - Class List window is currently selected.
5. Select **View - Inheritance Graph.**

The inheritance graph appears in the left-hand pane, as shown below.



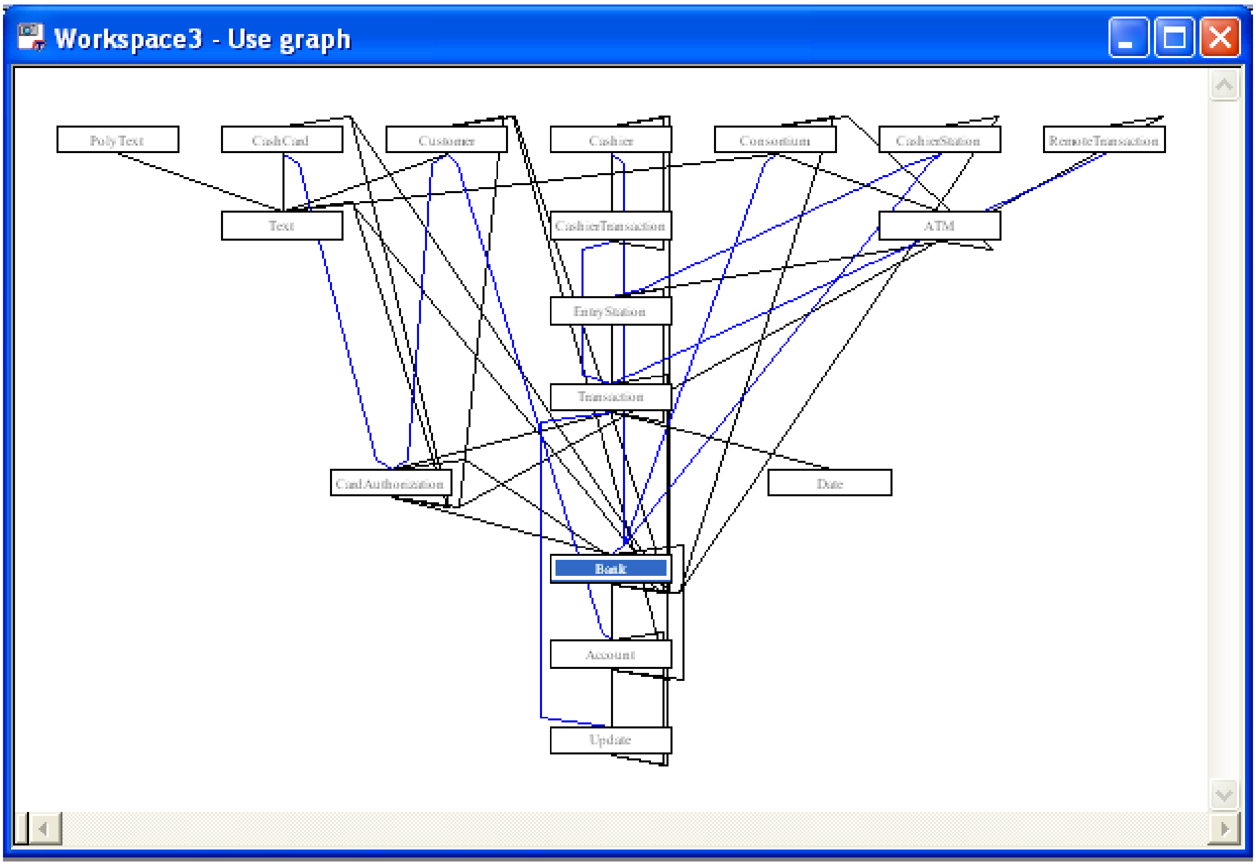
A simple inheritance graph just as an example ...

2.4.5 The Use Graph

The Use Graph gives the use classes relationship.

6. Select **View - Use Graph.**

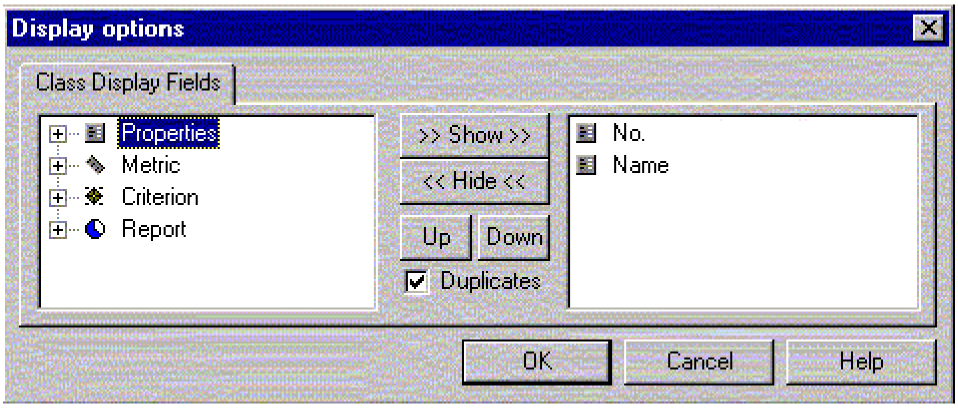
The use graph of the classes defined in the program appears in the left-hand pane, as shown below. A bit more complex than the inheritance graph ...



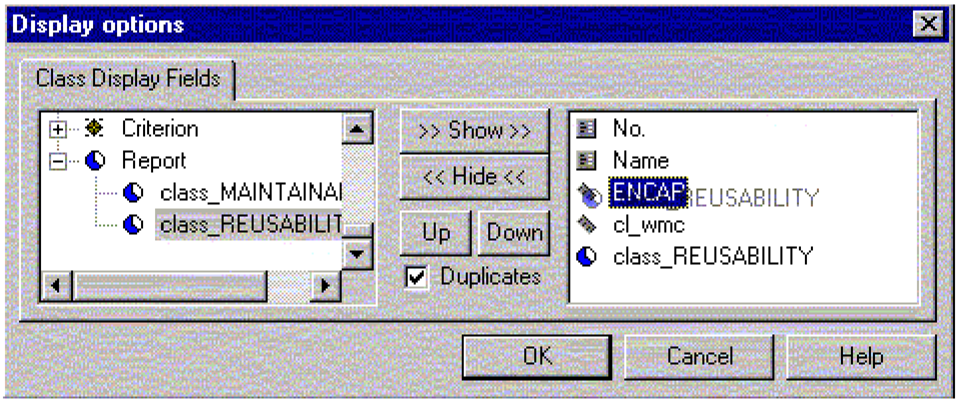
2.4.6 Getting More Detailed Class Results

More analysis results about internal classes can be obtained, provided you specify the exact information you want the **Viewer** to display.

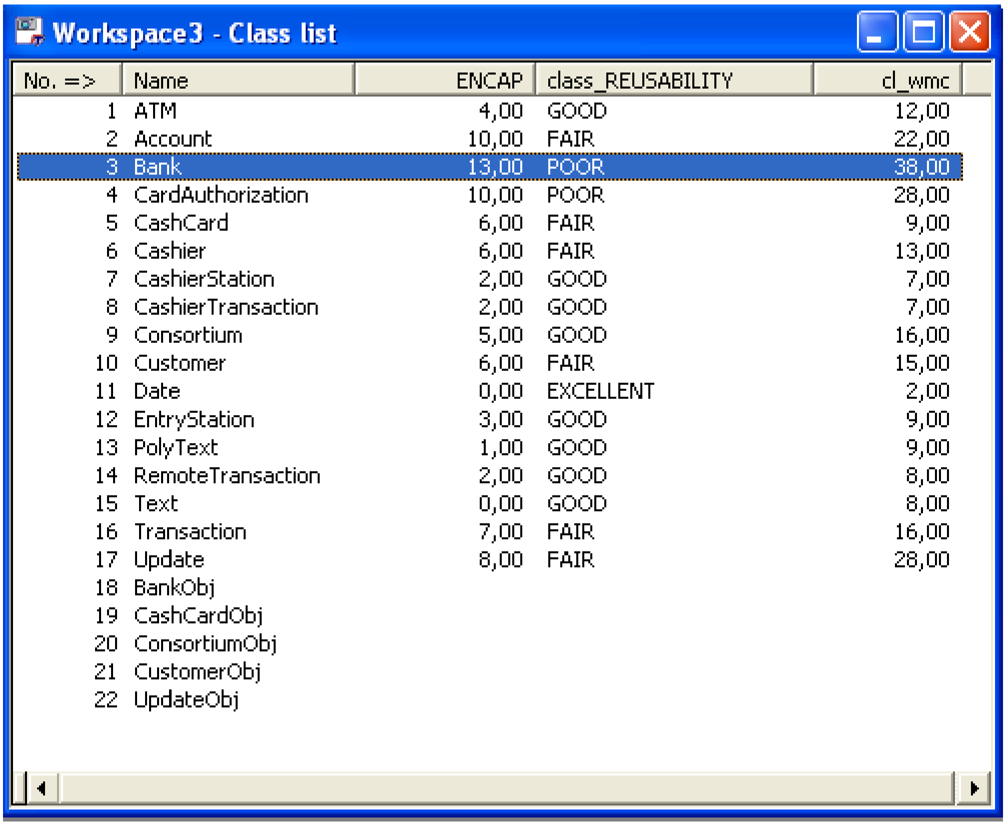
7. Select View - Class list to get back to the list of the classes.
8. Click **Options-Display Fields** command.



9. The **Display options** dialog box appears. This dialog box has two list boxes and a certain number of buttons. The list box on the left contains information fields available in the current project and the list box on the right shows information currently displayed.
10. From the left list box, click on **Metric** to see items it contains.
11. Double-click **ENCAP**
12. Select **cl_wmc** and press the >>**Show**>> button
    These metrics names are copied to the list box on the right.
13. Scroll down to **Report** and click on it to see items it contains.
14. Double-click **class_REUSABILITY**.
    The criterion is copied to the list box on the right.
15. Click on this item, now displayed in the list box on the right, drag it up, move it until **ENCAP** is highlighted and drop it.



16. Click on **OK** to take into account these changes.
As you can see, the **Workspace-Class list** window now shows values about metrics **cl_wmc** and **ENCAP** and class factor **REUSABILITY** for all classes of the application.



No. =>	Name	ENCAP	class_REUSABILITY	cl_wmc
1	ATM	4,00	GOOD	12,00
2	Account	10,00	FAIR	22,00
3	Bank	13,00	POOR	38,00
4	CardAuthorization	10,00	POOR	28,00
5	CashCard	6,00	FAIR	9,00
6	Cashier	6,00	FAIR	13,00
7	CashierStation	2,00	GOOD	7,00
8	CashierTransaction	2,00	GOOD	7,00
9	Consortium	5,00	GOOD	16,00
10	Customer	6,00	FAIR	15,00
11	Date	0,00	EXCELLENT	2,00
12	EntryStation	3,00	GOOD	9,00
13	PolyText	1,00	GOOD	9,00
14	RemoteTransaction	2,00	GOOD	8,00
15	Text	0,00	GOOD	8,00
16	Transaction	7,00	FAIR	16,00
17	Update	8,00	FAIR	28,00
18	BankObj			
19	CashCardObj			
20	ConsortiumObj			
21	CustomerObj			
22	UpdateObj			

Results windows can be customized freely in order to see the results you are interested in. Now you can decide to print the contents of this window for the next code review or move on to another result window.

Chapter 3

What a Logiscopian Quality Engineer Does

You heard through the grapevine that a Logiscopian quality manager will shortly be organizing a source code review of the *Mastermind* application. Rumor has it that the agreement reached between Logiscopians and their clients specifies that Logiscopians should meet the MAINTAINABILITY requirement as defined in ISO/IEC 9126.

In order to list the functions to be checked according to quality standards, you will:

- select the functions to be checked,
- create a workspace step by step,
- use the control graph to make a quality diagnosis,
- print quality results.

3.1 Loading an Existing Logiscope Project

1. First, start a new Logiscope *Viewer* session in the manner described in the previous chapter.
2. Select the **File-Open...** command.

In the **Open** window, browse to select the **MastermindAudit.ttp** Logiscope project file in the **samples/C/LogiscopeProjects** folder in the Logiscope installation directory.

The **Workspace1 -Component list** window is displayed in the Result Pane.

3.2 Creating a Workspace of Functions to be Inspected

You are going to fill an empty workspace with an initial set of critical functions and then refine this workspace on the basis of quality and complexity measurements.

1. Select the **File-New...** command.
2. In the **New** dialog box, select **Function Workspace** and click the **OK** button.
   A new **Workspace2-Component list** window is displayed in the Result Pane.
3. From the title bar, activate the **Edit-Component Workspace** command.
4. In the Component workspace window, click on the **Remove all** button.
5. Click on the **OK** button to confirm and close the Component workspace window.

The **Workspace2-Component list** window is now empty.

6. Activate the **Window-Vertical Tile** command to arrange the Workspace1 and Workspace2 windows side by side.

This way, you will see better step by step how the initial list of functions to be inspected is created.

3.2.1 From the Quality Report

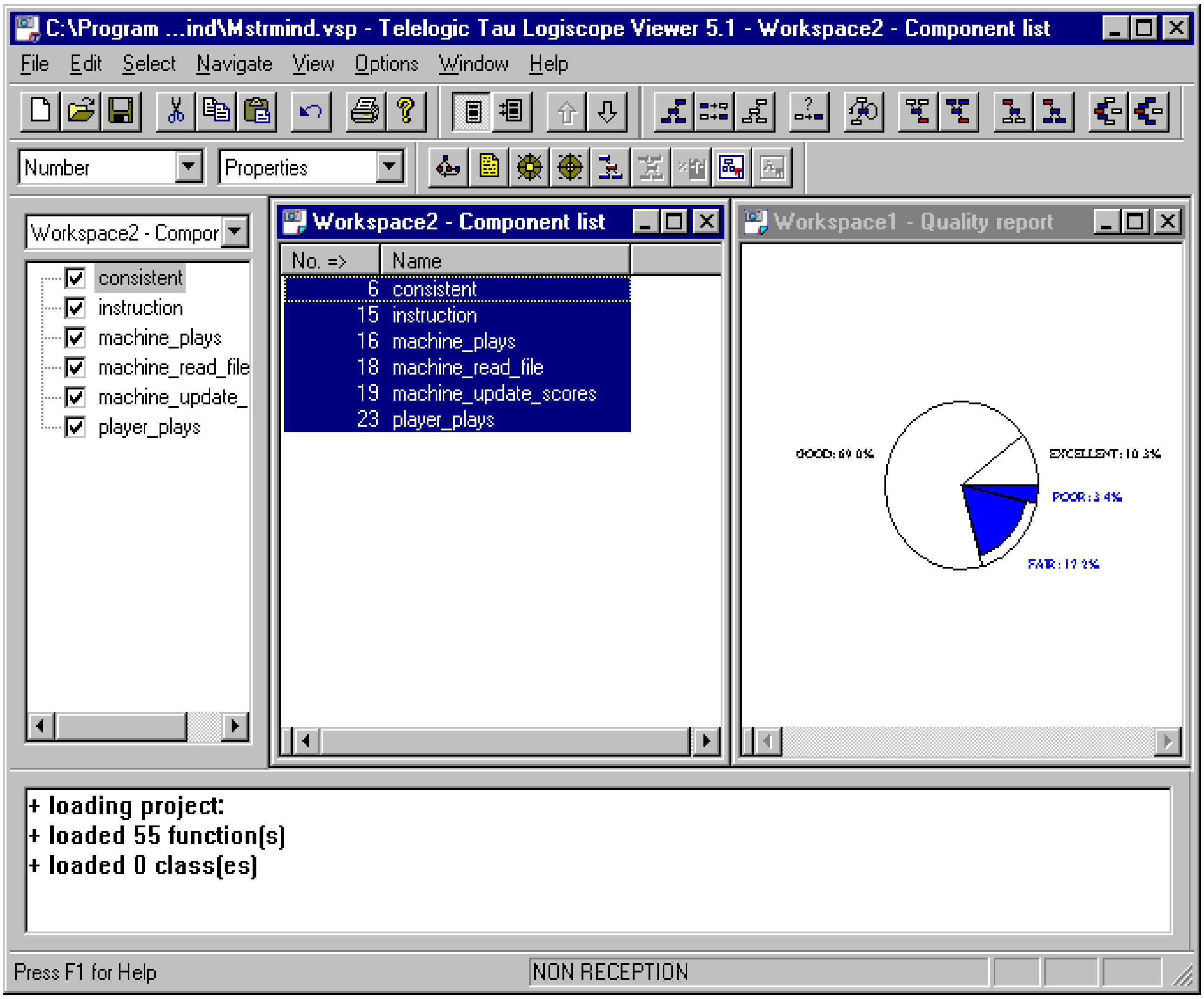
You are about to explore the quality report for the *Mastermind* program as the first viewpoint for selecting critical functions.

1. Click **Workspace1-Component list** to make this window active.
2. Select the **View-Report** command from the menu bar.
3. The quality report is displayed in the Workspace1 window.
4. Click on the *POOR* and *FAIR* slices of the quality report pie chart.

Functions which belong to the *POOR* and *FAIR* categories are selected, as shown in the Control Palette. These functions may not satisfy the client and a code review could be needed here.

5. Click on the ⧉ icon or select the **Edit-Copy** command.
6. Click in the **Workspace2-Component list** window to make it the active one.
7. Click on the 📋 icon or select the **Edit-Paste** command.

The list of functions which rank *POOR* and *FAIR* in the quality report are now listed in the **Workspace2-Component list** window. You will soon see how in the Logiscopian quality control department these functions are being used.



No. =>	Name
6	consistent
15	instruction
16	machine_plays
18	machine_read_file
19	machine_update_scores
23	player_plays

```
+ loading project:
+ loaded 55 function(s)
+ loaded 0 class(es)
```

3.2.2 From the Call Graph

Logiscopian quality managers are extremely professional and want to examine most strategically significant call graph functions. Get ready to inspect the following three types of functions:

- functions which are part of a recursive path,
- functions calling a significant number of functions,
- the main function.

First, let's start with recursive functions.

1. Click on the **Workspace1-Quality report** window. It becomes the active window.
2. From the menu bar, select the **View-Call Graph** command.

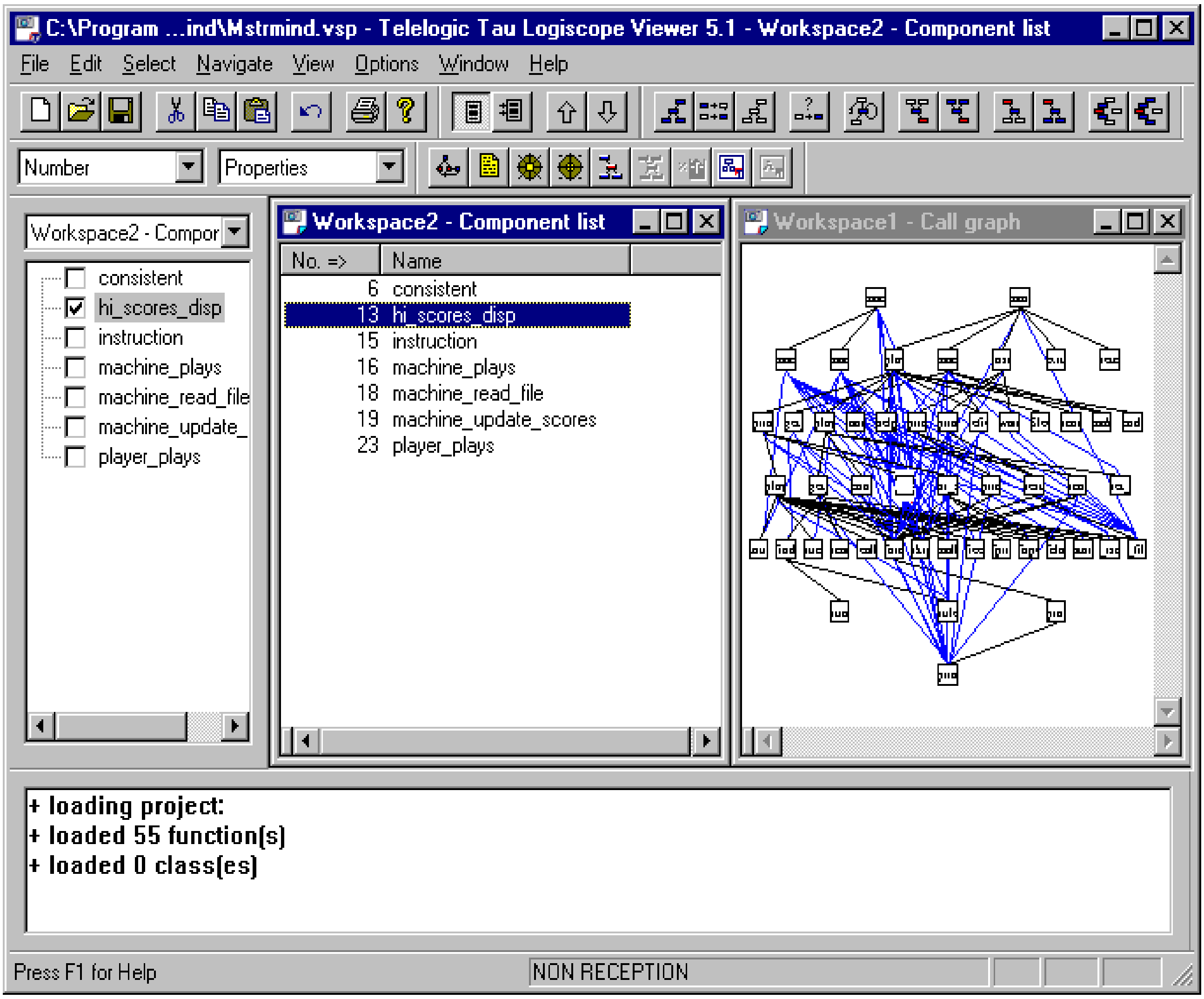
The call graph appears in the Workspace1 window. It is so complex that even with your well-trained eyes you cannot tell which functions follow a recursive path. Logiscopians know how to select it directly.

3. Click on the icon or choose the **Select - Recursive** command.

Function *hi_score_disp* is now selected and it is the only recursive-path function.

You are going to copy this function into the Workspace2 window as described below:

4. Click on the icon or select the **Edit - Copy** command.
5. Click in the **Workspace2 - Component list** window.
6. Click on the icon or select the **Edit - Paste** command.



C:\Program ...ind\Mstrmind.vsp - Telelogic Tau Logiscope Viewer 5.1 - Workspace2 - Component list
File Edit Select Navigate View Options Window Help
Number
Properties
Workspace2 - Compor
Workspace2 - Component list
Workspace1 - Call graph

□ consistent
☑ hi_scores_disp
□ instruction
□ machine_plays
□ machine_read_file
□ machine_update_
□ player_plays

No. =>	Name
6	consistent
13	hi_scores_disp
15	instruction
16	machine_plays
18	machine_read_file
19	machine_update_scores
23	player_plays

```
+ loading project:
+ loaded 55 function(s)
+ loaded 0 class(es)
```

Press F1 for Help   NON RECEPTION

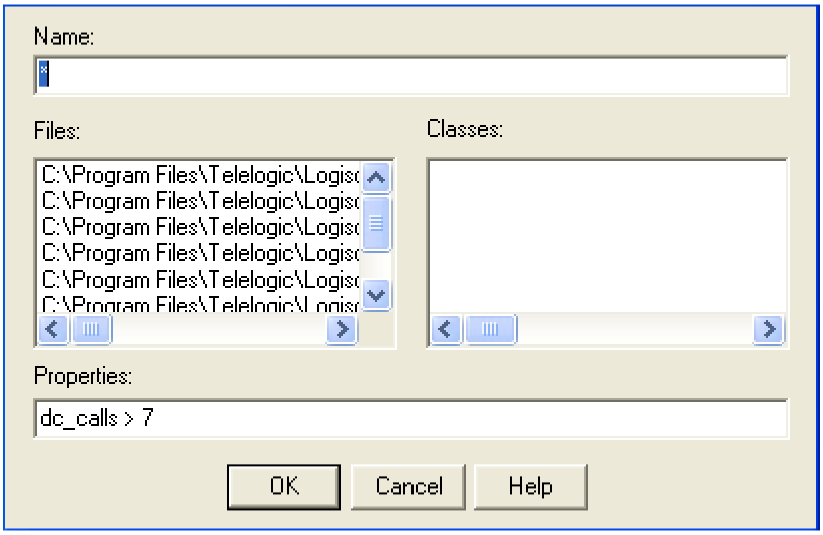
hi_score_disp is now displayed in the list of functions in the Workspace2 window.

Functions calling a significant number of functions are just as difficult to spot. Use the Logiscopian selection box to sort this out:

7. Click in the **Workspace1-Call graph** window.
8. From the menu bar, select the **Select - From Properties...** command.

Logiscopians use the **Query on components** window to select functions with specific characteristics. The **dc_calls** metric measures the number of distinct calls stemming from a function. Logiscopians use this metric to spot the functions which have a large number of call relations.

9. In the Properties: pane, enter the following expression: **dc_calls > 7**



Name:

Files:

C:\Program Files\Telelogic\Logisc
C:\Program Files\Telelogic\Logisc
C:\Program Files\Telelogic\Logisc
C:\Program Files\Telelogic\Logisc
C:\Program Files\Telelogic\Logisc
C:\Program Files\Telelogic\Logisc

Classes:

Properties: dc_calls > 7

10. Click on the **OK** button.

The five functions which call more than eight other functions are now selected on the call graph.

11. Add the function *main* to the selection by clicking on the appropriate node (top right root) on the call graph.

You have selected two other types of strategic functions in the call graph. Add them to the initial list of functions to be inspected in the **Workspace2** window.

12. Click on the icon or select the **Edit - Copy** command.
13. Click on the Workspace2 - Component list window to make it become the active window.
14. Click on the icon or select the **Edit - Paste** command.

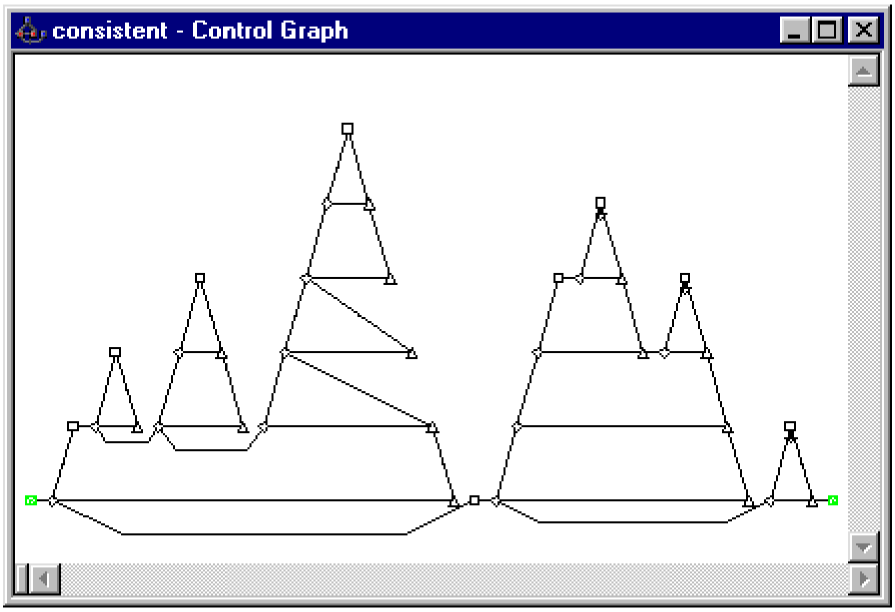
3.3 Inspecting Selected Function Control Graphs

A quick glance at functions selected in the Workspace2 window sends shivers down your spine! If you spend time exploring all functions, Logiscopians will spot you, and they might not like their source code being viewed by foreign explorers. Besides, your team is expecting you back with excellent information within a couple of hours. Since you are a sensible, cool-headed explorer, you will disregard functions which have a simple control structure. You will however need to view the control graph for each selected function.

1. Deselect all functions in the **Workspace2-Component list** window and click on the first function (i.e. *consistent*) of the list.

2. Click on the icon or select the **Window - Control** command

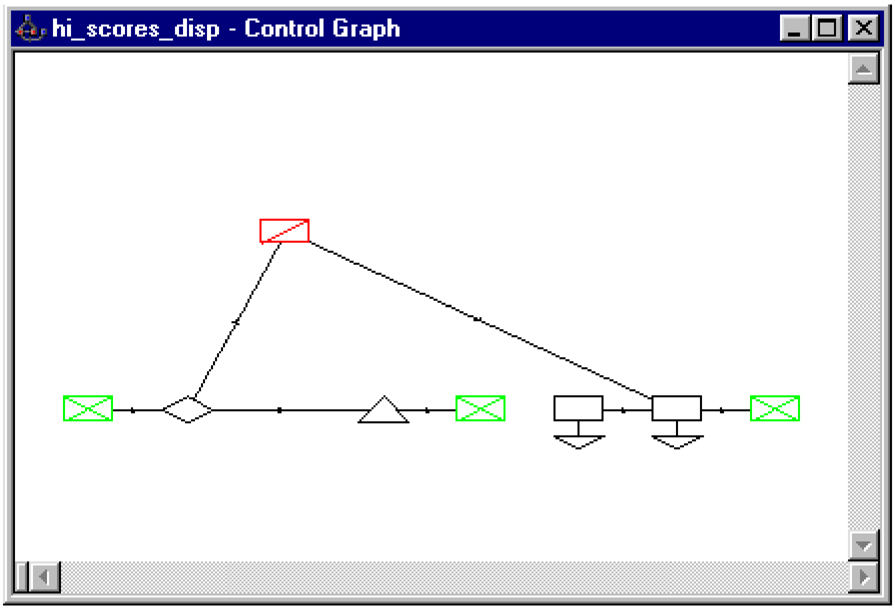
The control graph of the *consistent* function is displayed.



The control flow is clear but the function looks like it is overloaded with processes. Actually, it looks as if there was a sequence of two functionalities. If this is the case, why not make them two different C functions? A more clear-cut function would certainly improve maintenance performance, or possible reuse of this portion of code. Logiscopians will certainly explore further in their code review.

3. Click on the ⇩ icon to examine the next function.

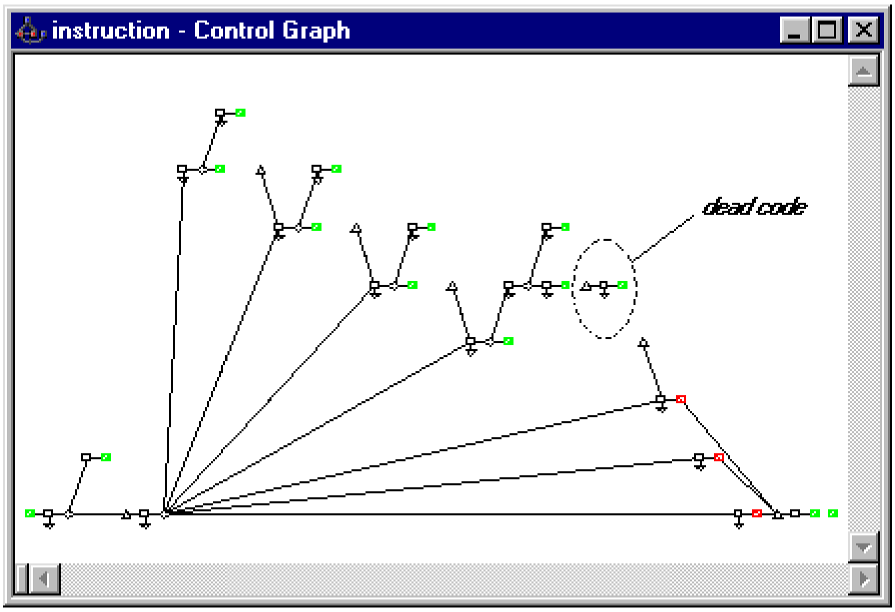
The control graph of *hi_score_disp* function is displayed.



What a wonderful *goto* instruction! What else, a portion of the code seems out of reach. You may wonder whether or not this is normal. It should be easy enough to make this function easier to maintain than that! You will definitely discuss the matter in the code review.

4. Click on the ⇩ icon to examine the next function.

The control graph of the *instruction* function is displayed.

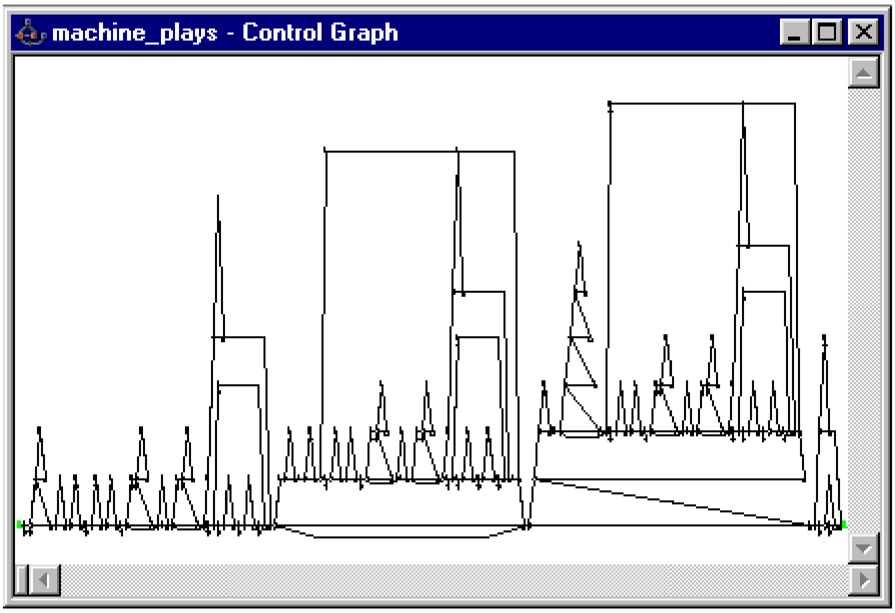


What a mess! (your telescopic eye just spotted dead code). Could a bug be crawling about? Has a Logiscopian developer missed this one?

Looking further into it, you can see that *switch* structures are odd. The first five cases leave the function with a *return* instruction whereas the last two cases are processed differently. This lack of homogeneity will make readability difficult and maintenance will be risky, but let's carry on.

5. Click on the ⇩ icon to examine the next function.

The control graph of the *machine_plays* function is displayed.



Control Graph (machine_plays)

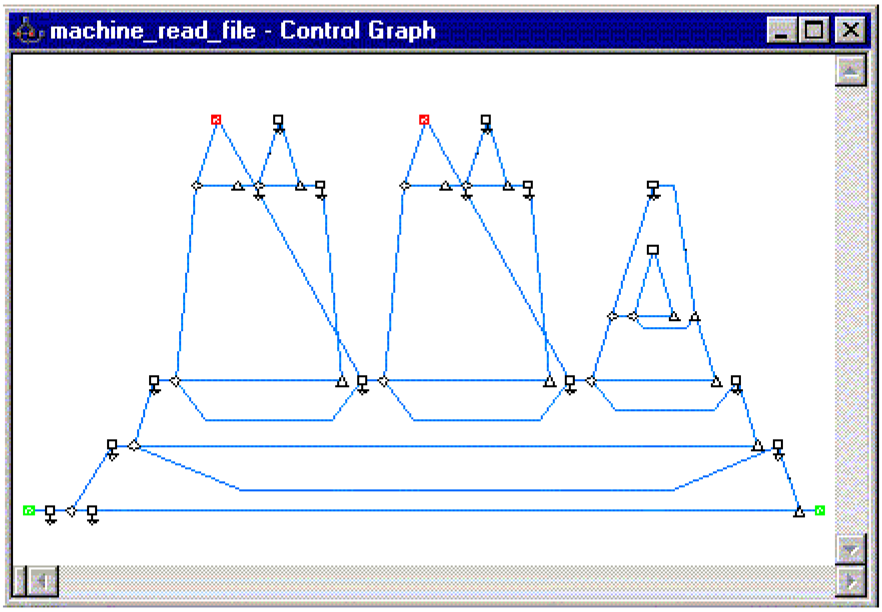
Now you thought you had seen a mess before! Your quality manager will not mince his words about the Logiscopian developer when he sees this! Why are there repeated groups of structures? Let's take a closer look at all this.

Everything is packed so tight on the screen that you can't see the wood for the trees. To get a clearer view of the control graph:

6. Press and hold the <ALT+SHIFT> keys.
7. Click the mouse and pull the selection box around the area you want to enlarge.
8. Release the mouse button.
9. Now it all looks a little clearer. There seems to be no code factorization at all. Remember, when you explored the component control graph in phase 2, your guides advised you to keep it in mind for the report. Logiscopians will soon be told to re-read this, and correct it fast.

10. Click on the ⇩ icon to examine the next function.

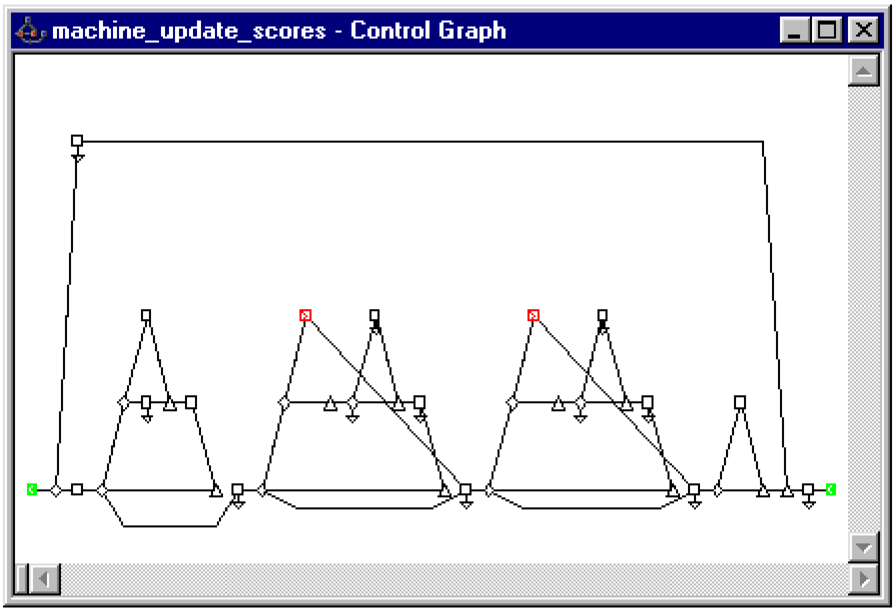
The control graph for function *machine_read_file* is displayed.



Here you see an apparently clear sequence of nested control structures. However, there seems to be a duplication of a portion of the code including a break statement. You will keep this in mind for the review.

11. Click on the ⇩ icon and proceed to the next function.

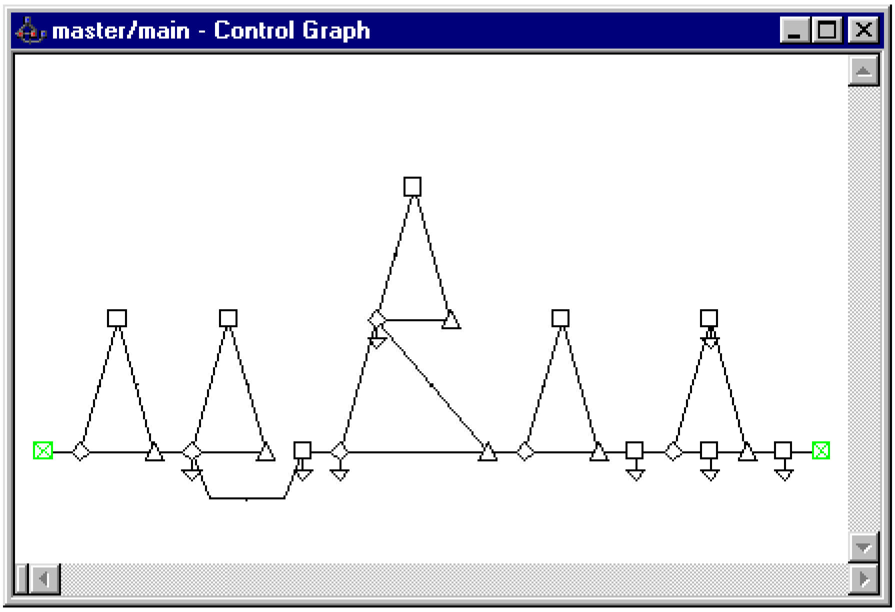
The control graph of function *machine_update_scores* is displayed



The same portion of code with the break statement as in the previous function seems to be repeated twice. Lack of code factorization between these two functions? You will discuss the matter, no doubt.

12. Click on the ⇩ icon to examine the next function.

The control graph of the *master/main* function is displayed.



Its structure looks so simple that you need not inspect it, better take it off the list of functions to be inspected now.

13. Click in the **Workspace2 - Component list** window.

14. Deselect all items in the graph to avoid taking them off the list.

15. Click on the *master/main* function to select it.

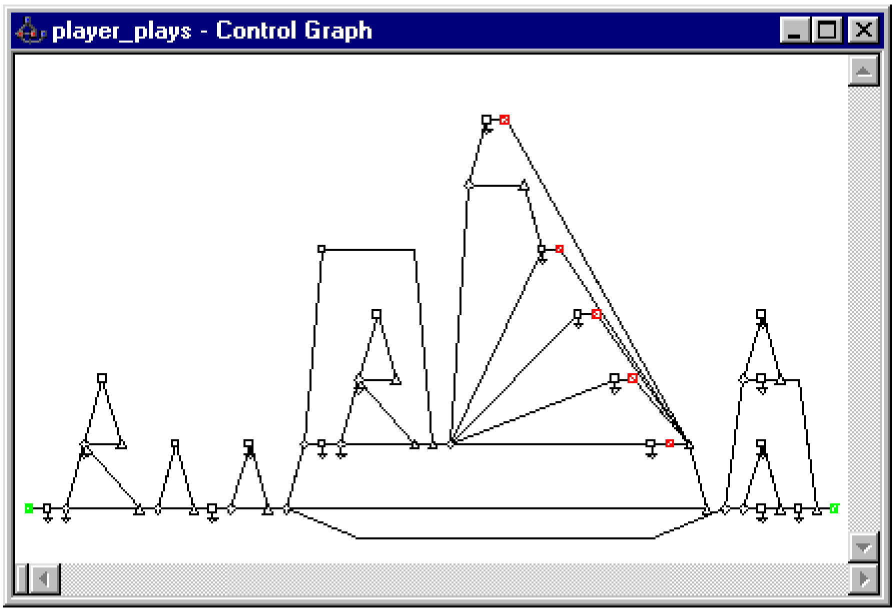
16. Click on the ✂ icon or select the **Edit - Cut** command.

The *master/main* function has been removed from the list of functions to be inspected. Logiscope *Viewer* automatically updates the Control window with the next function in the list.

17. Click in the **Control window**.

The control graph of *player_plays* function is displayed.

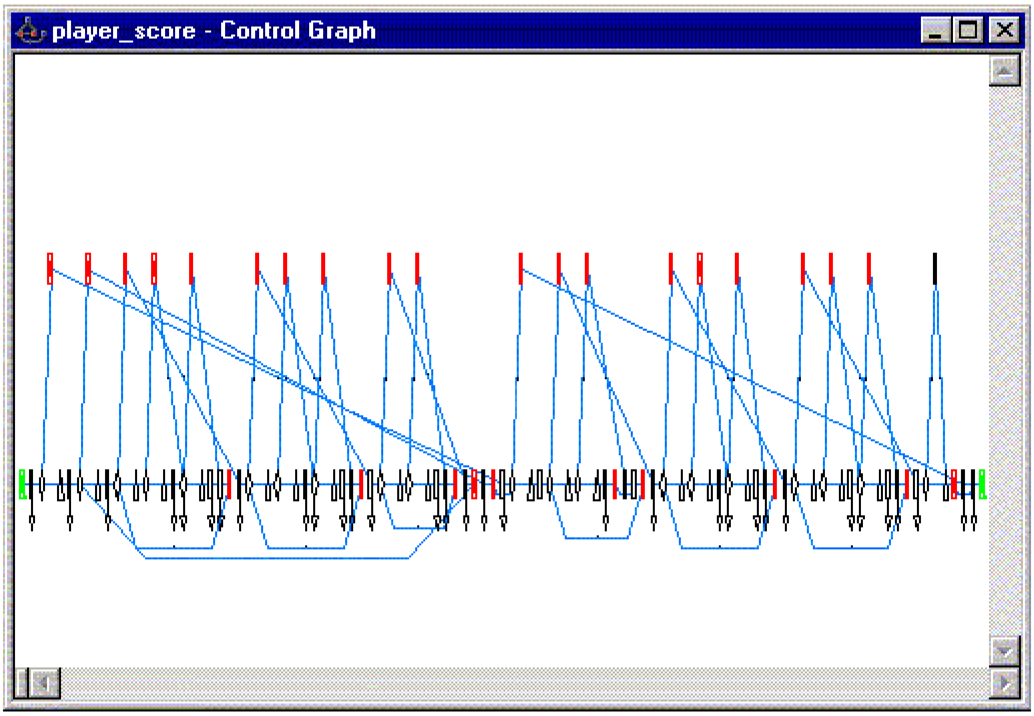
⇩



At a first glance, the graph seems to be clear but your natural insight tells you to explore further into the *switch* structure. You were right, a *switch* path jumps from the first to the second case. You have no idea whether this is correct or whether the Logiscopian developer omitted a *break* statement.

18. Click on the ⇩ icon to examine the next function.

The control graph of the *player_score* function is displayed.



Your expert saw that: this function has many *goto* instructions. A little bit dangerous isn't it?

To show all this to your team back home and illustrate your report, you will print the control graph as explained below.

3.4 Printing a Control Graph

1. In order to get a preview of the Control Graph, you can select **File-Print Preview...** option.
2. Click on the 🖨 icon and then click on the **OK** button

The current control graph is to be printed out.

3.5 Saving a Workspace

Okay, finished with control graph exploration. The workspace has been narrowed down to the eight functions which remain listed in the Control Palette. You've got your function list ready for the code review.

Save this list as a Logiscope workspace for a further round of inspection, which will be carried out in the next phase.

1. Click in the **Workspace2-Component list** window.
2. Select the **File - Save** command.
3. A **Save As** window is now displayed.
4. Use the **Look In** box to select the **C:\Mastermind** directory.
5. In the **Name** pane, enter the name of the Logiscope workspace to be saved: *Review.*
   A**.ws** extension (**L**ogiscope **W**ork**S**pace) is automatically added to the file name.

You can now end the *Viewer* session (not necessarily for another coffee break but mainly for the needs of this story).

Select the **File - Exit** command.

⇩

Chapter 4

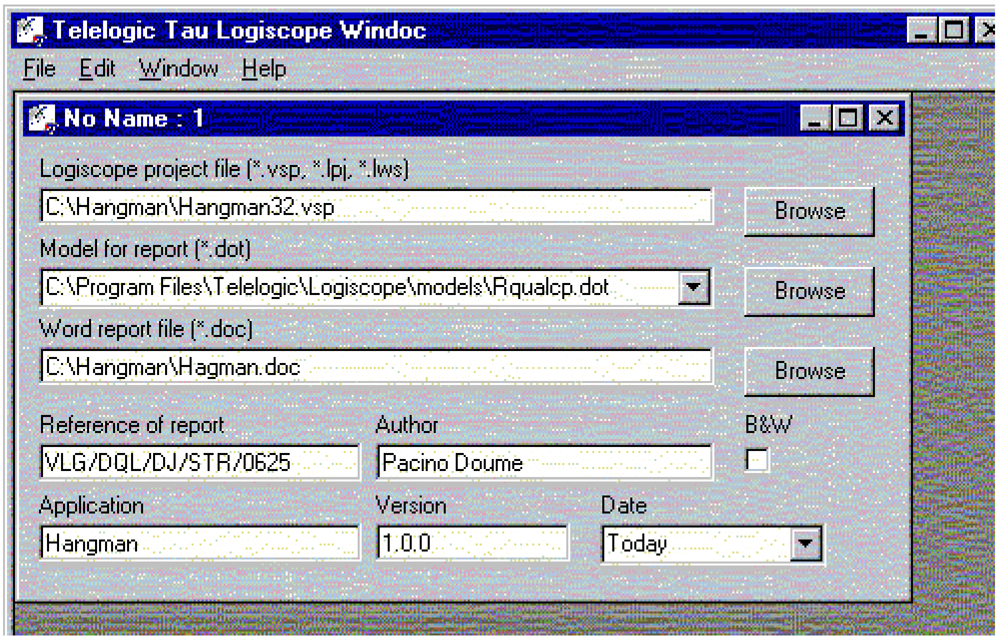
Generating a Quality Report Automatically

Note: This section is applicable on Windows platforms only.

Automation is one of the computer industry hottest buzzwords of the day when it comes to improving productivity, especially when talking of documentation production. With this in mind, you will certainly appreciate how Logiscope users generate automatically advanced reports using Logiscope **WinDoc**.

1. Click the **Start** button, and then select **Logiscope WinDoc** in the **Kalimetrix** > **Kalimetrix Tools** > **Kalimetrix Logiscope 2014** program group.

The Logiscope **WinDoc** main window appears :



Logiscope project file (*.vsp, *.lpj, *.lws): C:\Hangman\Hangman32.vsp

Model for report (*.dot): C:\Program Files\Telelogic\Logiscope\models\Rqualcp.dot

Word report file (*.doc): C:\Hangman\Hagman.doc

Reference of report: VLG/DQL/DJ/STR/0625

Author: Pacino Doume

B&W: □

Application: Hangman

Version: 1.0.0

Date: Today

The **No Name: 1** window contains several fields for specifying different parameters of the documentation to be generated.

Fill in all of these fields as described hereafter.

2. In the **Logiscope project file** pane, enter the name of the Logiscope project: **e.g. C:\LogiscopeProjects\Mastermind.ttp**, or use the **Browse...** button and navigate to this file.

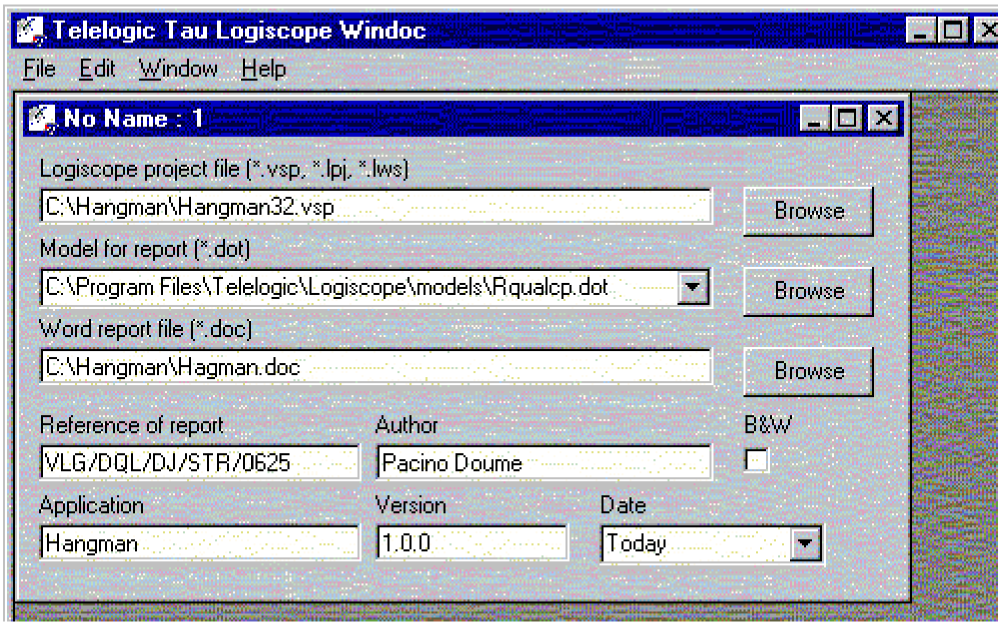
The **Model for report** pane allows you to select the type of report to be generated. Logiscope *RuleChecker* and *QualityChecker* offers various types of reports:

•Software Quality Evaluation Report,

•Software Maintenance Report, ...

3. Click on the **Browse...** button located next to the **Model for report** pane and select the **models** folder in the Logiscope installation directory.
4. Select **Rqual.dot** file in the **Open** dialog box.
5. In the **Word report file** pane, directly type the name of the Word document to be generated: e.g. **C:\LogiscopeProjects\Mastermind_QualityReport.doc**, or use the **Browse...** button.
6. Fill the 5 other panes with the appropriate information:

   Reference of report: gives the identification information you want to the generated document,
   Author: enter your name,
   Application: type the name of the project: e.g. *Mastermind*,
   Version: type the version number of the report : e.g. V1.0,
   Date: leave the *Today* default value for this field.
   B&W: uncheck if you want the colored report.



Logiscope project file (*.vsp, *.lpj, *.lws): C:\Hangman\Hangman32.vsp

Model for report (*.dot): C:\Program Files\Telelogic\Logiscope\models\Rqualcp.dot

Word report file (*.doc): C:\Hangman\Hagman.doc

Reference of report: VLG/DQL/DJ/STR/0625

Author: Pacino Doume

B&W: □

Application: Hangman

Version: 1.0.0

Date: Today

7. When you have entered the requested information, select the **File-Build report** command.

A message window is displayed.

8. When report generation is complete, simply open the generated Word report: e.g. : C:\LogiscopeProject\Mastermind_QualityReport.doc and appreciate how efficient Logiscope **WinDoc** automatic documentation generation is.
9. Save the current Quality Report description, so you can apply it to another document later.
10. Select the **File - Exit** command to end the Logiscope **WinDoc** session.

Notices

© Copyright 2014

The licensed program described in this document and all licensed material available for it are provided by Kalimetrix under terms of the Kalimetrix Customer Agreement, Kalimetrix International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-Kalimetrix products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Kalimetrix has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Kalimetrix products. Questions on the capabilities of non-Kalimetrix products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Copyright license

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to Kalimetrix, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. Kalimetrix, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
© (your company name) (year). Portions of this code are derived from Kalimetrix Corp. Sample Programs. © Copyright Kalimetrix Corp. _enter the year or years_.

Trademarks

Kalimetrix, the Kalimetrix logo, Kalimetrix.com are trademarks or registered trademarks of Kalimetrix, registered in many jurisdictions worldwide. Other product and services names might be trademarks of Kalimetrix or other companies.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, FrameMaker, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

AIX and Informix are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

HP and HP-UX are registered trademarks of Hewlett-Packard Corporation.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Macrovision and FLEXnet are registered trademarks or trademarks of Macrovision Corporation.

Microsoft, Windows, Windows 2003, Windows XP, Windows Vista and/or other Microsoft products referenced herein are either trademarks or registered trademarks of Microsoft Corporation.

Netscape and Netscape Enterprise Server are registered trademarks of Netscape Communications Corporation in the United States and other countries.

Sun, Sun Microsystems, Solaris, and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Pentium is a trademark of Intel Corporation.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.