SonargraphBuild User Manual

Version 8.9.2

SonargraphBuild User Manual: Version 8.9.2

Table of Contents

Chapter 1. Sonargraph's Next Generation - SonargraphBuild

SonargraphBuild integrates quality checks into the continuous integration build and can create XML and HTML reports via an Ant task or shell scripts. These reports contain all information about quality issues and calculated metrics. The XML report can be used for further downstream processing via transformations. The XML schema for the report can be found in <sonargraphBuild-inst>/doc.

SonargraphBuild additionally offers the possibility to mark the build as failed based on issues detected during the analysis. So, if you have written custom queries via Groovy scripts that check on the proper usage of an external library or detect a code smell, you can be sure that it is detected immediately.

Chapter 2. Licensing

When you start *Sonargraph* you will be asked for an activation code or a license file. For additional licensing and pricing information please contact <sales@hello2morrow.com>

2.1. Getting an Activation Code or a License

When you have purchased a *Sonargraph* license, an activation code or a license file will be delivered to you.

There might be a program for free *Sonargraph* licenses which are time-limited and/or size-limited. Please register on our website and check the available programs.

In order to replace a valid license by a new one, choose "Help" → "Manage License..." from the user menu in the GUI-based product. *Sonargraph* licenses are bound to a named user. The usage by a different user is a violation of the license agreement.

2.2. Activation Code Based Licensing

Activation code based licensing activates *Sonargraph* licenses via Internet. Every activation code is customer specific and represents a pool of *Sonargraph* user licenses as purchased and licensed to the specific customer. Activation code based licensing technically requires that *Sonargraph* has internet access.

Activation code based licenses support a feature that allows customer-driven transfer of a *Sonargraph* user license to another user after some amount of time. This works like this:

- When an activation code based license is requested, *Sonargraph* automatically requests a license ticket from the hello2morrow license server. This ticket expires after some time, for example after 30 days. During these 30 days, the use of the *Sonargraph* installation that requested the ticket is licensed (by the user who ran *Sonargraph* when the license ticket was requested). *Sonargraph* can be used during this period without any access to the Internet.

- After the ticket of a *Sonargraph* installation has expired (in our example scenario, this happens on the 31st day after the ticket has been requested), one of two things typically happen:

  1. The same *Sonargraph* installation is started again. *Sonargraph* then notices that the license ticket has expired and lets the user know about it by presenting a dialog to manually request a new ticket from the hello2morrow license server, for the same activation code or a different one if desired. The new ticket again is valid for the same time period. You can toggle the feature at ' Help → Renew License Ticket Automatically ' to have *Sonargraph* silently perform license ticket requests using the current activation code, without further user interaction.

  2. Alternatively, the user of the installation might not continue to work with *Sonargraph* ; then the license is now, after the expiration of the ticket in the *Sonargraph* installation, available to some other user. The hello2morrow license server will supply a license ticket to the next user that requests one for the given activation code.

Note that the number of license tickets that can be supplied by the license server for some activation code might be more than one. For example, a company might license *Sonargraph* for 20 users. The same activation code can be used by all of them, but as soon as the 21st license ticket is requested for this activation code, this request will be denied. A new request for a ticket will only be fulfilled after one of the already supplied tickets has expired, so that at any one moment, at most 20 non-expired license tickets exist for the activation code.

It is not required that the same user requests a replacement of an expired license ticket; any user that knows the activation code can request one of the free tickets. This mechanism reduces the effort needed for license management in a changing user group. However, in order to avoid any misuse we strongly encourage you to restrict the information about your activation code to those persons who are supposed to use *Sonargraph* .

If you have any suspicion about misuse please inform <support@hello2morrow.com> immediately. We can promptly deactivate an activation code so that any further misuse is stopped and provide a new activation code to you.

2.3. Proxy Settings

Activation code based licensing technically requires that *Sonargraph* has Internet access. If your network configuration does not allow direct Internet access, but provides access through an HTTP proxy instead, you can specify the host name and port of the proxy server. If the proxy server access is password protected, you can supply a user name and a password in order to authenticate.

For the GUI-based product, the activation code and proxy access data is supplied by selecting "Help" → "Manage License" → "Configure Proxy Settings" or "Preferences…" → "Proxy Settings" .

Chapter 3. Getting Started

This chapter summarizes what is needed for *SonargraphBuild* to run.

3.1. Installation Requirements

The following prerequisites must be fulfilled for *SonargraphBuild* :

1. Microsoft™ Windows™ , Mac OS-X or Linux® operating system.
2. Java Runtime Environment 1.8 or higher
3. At least 2048 MB RAM (Win32: 1400 MB)

3.2. Prerequisites

1. If you plan to run *SonargraphBuild* via ANT or the command line you need to download it from our web site: *https://www.hello2morrow.com/products/downloads* and extract the Zip file to a convenient location. If you are using Maven for your build process you only need to install *SonargraphBuild* if your build server has no Internet connectivity.
2. If the machine that executes *SonargraphBuild* has internet access, use an activation code parameter to obtain a ticket from your pool of licenses. If the machine does not have internet access, you need to obtain a license file and pass the location of this file as a parameter to your build.
3. For integration with Shell script or Ant a "software system" must have been created via Sonargraph containing a valid workspace configuration including modules and root directories.

   Integrations with Maven and Gradle allow to dynamically create a "software system" on the fly and create a report for it. Those integrations can be used to create an initial software system that is refined using Sonargraph rich-client application.

Chapter 4. Executing from the Command-line

SonargraphBuild can be executed as a standalone Java application which enables the integration in any kind of continuous integration environment. The necessary configuration is straight-forward and an example shell script is provided in the directory <inst-dir>/example/bin. The batch script starts *SonargraphBuild* and specifies an XML file for the detailed configuration.

NOTE

SonargraphBuild returns an exit code != 0 if there are any configuration errors or the analysis detected issues defined by the failset.

NOTE

The attribute "logLevel" affects the logging after the SonargraphBuild engine has been started. Setting it to "debug" and below will generate additional debug information into the report.

WARNING

Setting the attribute reportType to "standard" only generates metric values for "system" and "module" levels. "full" generates values for all element levels, but results in a significant larger report!

4.1. Report Creation

The following table lists all parameters that are available to create a report for an existing Sonargraph system:

Attribute	Mandatory	Description	Default
installationDirectory	Yes	Installation directory of SonargraphBuild	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. If no value is specified, all languages will be initialized.	Java, CSharp, CPlusPlus
logFile	No	Path of the log file to be used for SonargraphBuild.	${currentDir}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
compilerDefinitionPath	No	The path to the active compiler definition file to be used for parsing a C/C++ system. If a built-in or automatically generated definition should be used, prefix the definition with "CPlusPlus:", e.g. CPlusPlus:GnuCpp.cdef. **NOTE:** If the standalone Sonargraph application is used on the same machine with the same user	If empty, the default compiler definition for the build server's operating system is used: GnuCpp.cdef (Linux), CLang.cdef (Mac), VisualCpp*.cdef

Attribute	Mandatory	Description	Default
		and this parameter is empty, the active definition specified with the standalone application will be used.	(Windows, generated definition for the latest Visual Studio installation)
systemDirectory	Yes	Directory of the Sonargraph System (xyz.sonargraph)	No default
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
workspaceProfile	No	The profile file name (e.g. "BuildProfile.xml") for transforming the workspace paths to match the build environment.	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. All scripts, analyzer configurations and architecture files present in the system are ignored. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm"	No default
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	Current directory
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	Current directory
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
proxyHost	No	Proxy host	No default
proxyPort	No	Proxy port	No default
proxyUsername	No	Proxy user name	No default
proxyPassword	No	Proxy password	No default

Table 4.1. Configuration for Element "sonargraphBuild"

Example

This is an example configuration for creating an XML and HTML report:

```
<sonargraphBuild
    activationCode="_your activation code_"
    languages="Java"
    installationDirectory="../.."
    systemDirectory="../javaProject/AlarmClock.sonargraph"
    reportDirectory="./_temp/report"
    reportFileName=""
    reportType="full"
    reportFormat="xml,html"
    snapshotFileName="./_temp/AlarmClock.snapshot"
    proxyHost=""
    proxyPort=""
    proxyUsername=""
    proxyPassword=""
    logLevel="warn">
</sonargraphBuild>
```

4.2. Specify Conditions for Build Failure

SonargraphBuild can check for the existence of specific issues and mark the build as failed. The nested "failSet" element of the "sonargraphBuild" element can include any number of "include" and "exclude" definitions based on the issues that are either built-in (like duplicate warnings, cycle group warnings, etc.) or custom issues created via Groovy scripts.

TIP

The issue type that must be specified for include/exclude definitions can be determined via the Properties View of *Sonargraph* .

TIP

If you want the build to fail only for newly introduced issues, apply resolutions like "Ignore" or "Fix" to the already know issues in *Sonargraph* and only filter for resolution value "none".

TIP

If you want the build to fail because of certain metric values (e.g. Lines of Code per Source File), define a threshold for that metric in *Sonargraph* to create issues if files grow too large.

TIP

The include/exclude definitions are applied in the following sequence, regardless of their definition order:

1. First all include definitions are matched against the set of existing issues.
2. Then the exclude definitions are applied and the set of previously matched issues is reduced accordingly.

Element	Parameter	Mandatory	Description	Default
failSet	failOnEmptyWorkspace	No	Marks the build as failed if modules and root directories are detected but no components are found. Possible values are "true" or "false".	true
include/exclude	issueType	Yes	Name of the issue type or "any" for wildcard matching.	No default
include/exclude	severity	No	Severity of the issue. Possible values are: error, warning, info, none, any	any
include/exclude	resolution	No	The issue's resolution type. Possible values are: task, ignore, any, none	none

Table 4.2. Configuration Parameters for Build Failure

Example

This is an example failSet definition:

```
<sonargraphBuild
...
  <failSet failOnEmptyWorkspace="false">
    <include issueType="any" severity="error" resolution="none"/>
    <exclude issueType="ScriptCompilationError"/>
    <include issueType="Supertype uses subtype"/>
    <include issueType="any" severity="warning"/>
    <exclude issueType="ThresholdViolation"/>
  </failSet>
</sonargraphBuild>
```

The console output provides some basic information about the number of issues matched by either "include" and "exclude":

Failed:
Sonargraph: Start creating report...
Sonargraph: Opening system...
Sonargraph: Refreshing system...
Sonargraph: Creating report...
Sonargraph: Check if build should be marked as failed...
Include filter [issueType=any, severity=error, resolution=none] matches 0 issue(s).
Include filter [issueType=Supertype uses subtype, severity=any, resolution=none] matches 0 issue(s).
Include filter [issueType=any, severity=warning, resolution=none] matches 2 issue(s).
Exclude filter [issueType=ScriptCompilationError, severity=any, resolution=none] removes 0 previously matched issue(s).
Exclude filter [issueType=ThresholdViolation, severity=any, resolution=none] removes 0 previously matched issue(s).
Summary: Build failed as 2 issue(s) match the specified failset on virtual model 'Modifiable.vm'.
Sonargraph: Finished.

Chapter 5. Integrating with Ant

The provided SonargraphReportTask makes it easy to integrate *SonargraphBuild* into Apache Ant based builds and generate HTML or XML reports containing info about metrics and issues of a software system. Additionally, using the optional "failSet" element, the Ant build can be marked as failed if certain issues exist.

Prerequisites:

1. You need at least Ant 1.8.3 installed.

2. Set the environment variable ANT_HOME.

3. Include ANT_HOME/bin in your PATH environment variable.

The following shows the SonargraphReportTask definition:

```
<taskdef name="sonargraphBuild"
     classname="com.hello2morrow.sonargraph.build.client.ant.SonargraphReportTask">
  <classpath>
    <fileset dir="${sonargraph.build.installation}/plugins">
      <include name="org.eclipse.osgi_3.10*.jar" />
      <include name="com.hello2morrow.sonargraph.build.client*.jar"/>
    </fileset>
    <fileset dir="${sonargraph.build.installation}/client" includes="*.jar" />
  </classpath>
</taskdef>
```

An example Ant build.xml is provided in the directory <inst-dir>/example/ant. The parameters are the same as for the shell integration described in Chapter 4, *Executing from the Command-line*

Chapter 6. Integration with Maven

The *SonargraphBuild* Maven plugin makes it easy to check the quality of your projects. The plugin is able to automatically download and install *SonargraphBuild* if your build server has Internet connectivity. You can make the build fail depending on issues detected by *SonargraphBuild*.

There are two different goals available:

1. **create-report**: Creates a report for an existing system.

2. **dynamic-report**: Creates a system on-the-fly and creates a report for it. This is currently only available for Java systems.

A third command **help** shows more information and can be parameterized to show detailed parameter info.

Prerequisites:

1. You need at least Maven 3.0.5 installed.

2. The plugin requires at least a Java 8 runtime.

6.1. Maven Tips and Best Practices

TIP

Add the following repository to your Maven settings.xml, so you do not need to repeat it in your project's pom.xml:

```
<pluginRepository>
   <id>hello2morrow.maven.repository</id>
   <url>http://maven.hello2morrow.com/repository</url>
</pluginRepository>
```

TIP

The goals are **not** configured to be executed within any default Maven lifecycle phase. Typically you would run the plugin with a command-line like the following to ensure that everything is compiled from scratch before the report is created. The first command-line explicitly specifies a version, the second one uses the Maven prefix resolution (check *Maven Prefix Resolution* for details):

```
mvn clean compile com.hello2morrow:sonargraph-maven-plugin:8.9.2:create-report
```

```
mvn clean compile sonargraph:create-report
```

TIP

All parameters of the top-level goals (i.e. not the failSet) can equally be set via the command-line using system properties of the form

```
-Dsonargraph.<parameter-name>=<value>
```

TIP

To enforce certain rules, specify a failSet that lets the build fail based on detected issues. See Section 6.4, “Maven FailSet Configuration”

NOTE

The attribute "logLevel" affects the logging after the SonargraphBuild engine has been started. Setting it to "debug" and below will generate additional debug information into the XML report.

WARNING

Setting the attribute reportType to "standard" only generates metric values for "system" and "module" levels. "full" generates values for all element levels, but results in a significant larger report!

6.2. Parameters of Goal "create-report"

The following table lists all parameters that are available to create a report for an existing Sonargraph system. The build can be marked as failed based on a failSet. See Section 6.4, “Maven FailSet Configuration”.

Attribute	Mandatory	Description	Default
sonargraphBuildVersion	No	Allows you to use a specific or restricted version of SonargraphBuild. Can be used in combination with 'autoUpdate'. As an example, if you specify '8.7' the newest available version starting with '8.7' will be used. If you specify '8.7.0.361' you are locked on that specific version of SonargraphBuild. There are two special values: "same" and "newest". If "same" is defined the version of SonargraphBuild must be the same as the version of the Maven plugin. If "newest" is defined the plugin will always try to use the newest version of SonargraphBuild.	same
autoUpdate	No	If the plugin is configured to download SonargraphBuild automatically, this parameter decides if it also should be updated automatically if a new version becomes available.	false
useHttpProxyHost	No	The id of a proxy entry in the Maven settings. If defined the plugin will use this proxy for all HTTP communication.	No default
installationDirectory	No	Installation directory of SonargraphBuild. If unspecified the plugin will automatically download SonargraphBuild. The version to be downloaded can be controlled by the parameter 'sonargraphBuildVersion'.	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. Possible values: Java, CSharp, CPlusPlus.	Java
logFile	No	Path of the log file to be used for SonargraphBuild.	${baseDir}/${target}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
compilerDefinitionPath	No	The path to the active compiler definition file to be used for parsing a C/C++ system. If a built-in or automatically generated definition should be used, prefix the definition with "CPlusPlus:", e.g. CPlusPlus:GnuCpp.cdef.	If empty, the default compiler definition for the build server's operating system is used: GnuCpp.cdef

Attribute	Mandatory	Description	Default
		NOTE: If the standalone Sonargraph application is used on the same machine with the same user and this parameter is empty, the active definition specified with the standalone application will be used.	(Linux), CLang.cdef (Mac), VisualCpp*.cdef (Windows, generated definition for the latest Visual Studio installation)
systemDirectory	No	Directory of the Sonargraph System (xyz.sonargraph)	${baseDir}/${artifact.id}.sonargraph
overrideSonargraphWorkspace	No	If true the workspace will contain the output directories of the Maven build instead of the ones defined in the Sonargraph system.	true
includeTestCode	No	If true the workspace will also contain the test source and test class file directories.	false
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. All scripts, analyzer configurations and architecture files present in the system are ignored. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm"	No default
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	${basedir}/${target}
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	${basedir}/${target}/sonargraph
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
prepareForSonarQube	No	Creates an XML report and stores it at ${basedir}/${target}/sonargraph/sonargraph-sonarqube-report.xml, where the SonarQube Sonargraph plugin expects it.	false

Table 6.1. Configuration for goal "create-report"

Related topics:

- Section 6.1, “Maven Tips and Best Practices”
- Section 6.4, “Maven FailSet Configuration”
- Section 6.5, “Example POM”

6.3. Configuration for goal "dynamic-report"

The following table lists all parameters that are available to create a report for a Java project where no Sonargraph system has been defined. A Sonargraph system is created on the fly based on the workspace information contained in the Maven project setup. The build can be marked as failed based on a failSet. See Section 6.4, “Maven FailSet Configuration”.

Attribute	Mandatory	Description	Default
sonargraphBuildVersion	No	Allows you to use a specific or restricted version of SonargraphBuild. Can be used in combination with 'autoUpdate'. As an example, if you specify '8.7' the newest available version starting with '8.7' will be used. If you specify '8.7.0.361' you are locked on that specific version of SonargraphBuild. There are two special values: "same" and "newest". If "same" is defined the version of SonargraphBuild must be the same as the version of the Maven plugin. If "newest" is defined the plugin will always try to use the newest version of SonargraphBuild.	same
autoUpdate	No	If the plugin is configured to download SonargraphBuild automatically, this parameter decides if it also should be updated automatically if a new version becomes available.	false
useHttpProxyHost	No	The id of a proxy entry in the Maven settings. If defined the plugin will use this proxy for all HTTP communication.	No default
installationDirectory	No	Installation directory of SonargraphBuild. If unspecified the plugin will automatically download SonargraphBuild. The version to be downloaded can be controlled by the parameter 'sonargraphBuildVersion'.	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. Possible values: Java, CSharp, CPlusPlus.	Java
logFile	No	Path of the log file to be used for SonargraphBuild.	${baseDir}/${target}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
systemBaseDirectory	No	The directory where the Sonargraph System (${artifactId}.sonargraph) is created.	${baseDir}/${target}

Attribute	Mandatory	Description	Default
systemId	No	A system id should stay constant over the lifetime of a software system and should be also unique with respect to other systems. If you anticipate that the the group id or artifact id of the root pom might change you should assign a value to this parameter.	${groupdId}_ ${artifactId}
useGroupIdInModuleName	No	If true the module names will use group id and artifact id as their name, separated by an underscore. By default only the artifact id is used as the module name.	false
includeTestCode	No	If true the workspace will also contain the test source and test class file directories.	false
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. All scripts, analyzer configurations and architecture files present in the system are ignored. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm"	No default
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	${basedir}/${target}
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	${basedir}/${target}/ sonargraph
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
prepareForSonarQube	No	Creates an XML report and stores it at ${basedir}/ ${target}/sonargraph/sonargraph-sonarqube-report.xml, where the SonarQube Sonargraph plugin expects it.	false

Table 6.2. Configuration for goal "dynamic-report"

Related topics:

- Section 6.1, “Maven Tips and Best Practices”
- Section 6.4, “Maven FailSet Configuration”
- Section 6.5, “Example POM”

6.4. Maven FailSet Configuration

The following elements allow to mark a build as failed. An example is shown in the next section Section 6.5, “Example POM”.

TIP

The issue type that must be specified for include/exclude definitions can be determined via the Properties View of *Sonargraph* .

TIP

If you want the build to fail only for newly introduced issues, apply resolutions like "Ignore" or "Fix" to the already know issues in *Sonargraph* and only filter for resolution value "none".

TIP

If you want the build to fail because of certain metric values (e.g. Lines of Code per Source File), define a threshold for that metric in *Sonargraph* to create issues if files grow too large.

TIP

The include/exclude definitions are applied in the following sequence, regardless of their definition order:

1. First all include definitions are matched against the set of existing issues.
2. Then the exclude definitions are applied and the set of previously matched issues is reduced accordingly.

Element	Parameter	Mandatory	Description	Default
failSet	failOnEmptyWorkspace	No	Marks the build as failed if modules and root directories are detected but no components are found. Possible values are "true" or "false".	true
include/exclude	issueType	Yes	Name of the issue type or "any" for wildcard matching.	No default
include/exclude	severity	No	Severity of the issue. Possible values are: error, warning, info, none, any	any
include/exclude	resolution	No	The issue's resolution type. Possible values are: task, ignore, any, none	none

Table 6.3. Configuration Parameters for Build Failure

6.5. Example POM

The following example shows how to integrate the Sonargraph Maven plugin into your project specific pom file. For multi-module projects it is sufficient to only add the plugin to the pom of the root project. It runs as an aggregator after all modules have been compiled. The example project in the installation contains a complete pom.xml. Typically you would run the plugin with a command-line like the following to ensure that everything is compiled from scratch before the report is created. The first

command-line explicitly specifies a version, the second one uses the Maven prefix resolution (check *Maven Prefix Resolution* for details):

```
mvn clean compile com.hello2morrow:sonargraph-maven-plugin:8.9.2:create-report
```

```
mvn clean compile sonargraph:create-report
```

The following shows the relevant section of a pom.xml file that demonstrates the configuration of the Sonargraph functionality:

```
  <pluginRepositories>
    <pluginRepository>
        <id>hello2morrow.maven.repository</id>
        <url>http://maven.hello2morrow.com/repository</url>
    </pluginRepository>
  </pluginRepositories>
  <build>
    <plugins>
        <plugin>
            <groupId>com.hello2morrow</groupId>
            <artifactId>sonargraph-maven-plugin</artifactId>
            <version>8.9.2</version>
            <configuration>
                <systemDirectory>${basedir}/crm-domain-example.sonargraph</systemDirectory>
                <activationCode>...</activationCode>
                <autoUpdate>true</autoUpdate>
                <failSet>
                    <failOnEmptyWorkspace>true</failOnEmptyWorkspace>
                    <includes>
                        <include>
                            <issueType>ArchitectureViolation</issueType>
                        </include>
                        <include>
                            <issueType>any</issueType>
                            <severity>error</severity>
                        </include>
                    </includes>
                    <excludes>
                        <exclude>
                            <issueType>ScriptCompilationError</issueType>
                            <resolution>none</resolution>
                        </exclude>
                    </excludes>
                </failSet>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>create-report</goal>
                        <goal>dynamic-report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

In this example the build will fail if the project contains a package cycle or an architecture violation without a resolution. Since the parameter 'installationDirectory' is not defined, the Maven plugin will automatically download the newest release of *SonargraphBuild* and also will keep it updated automatically. Of course this requires that the build server has access to the Internet.

Chapter 7. Integration with Gradle

The *SonargraphBuild* Gradle plugin makes it easy to check the quality of your projects. The plugin is able to automatically download and install *SonargraphBuild* if your build server has Internet connectivity. You can make the build fail depending on issues detected by *SonargraphBuild*.

There are two different Gradle "tasks" available for which parameters can be defined in Gradle "extensions" with the same names:

1. **sonargraphReport**: Creates a report for an existing system.
2. **sonargraphDynamicReport**: Creates a system on-the-fly and creates a report for it. This is currently only available for Java systems.

Prerequisites:

1. You need at least Gradle 2.9 installed.
2. The plugin requires at least a Java 8 runtime.

7.1. Gradle Tips and Best Practices

TIP

To enforce certain rules, specify a failSet that lets the build fail based on detected issues. See Section 7.4, “Gradle FailSet Configuration”

NOTE

The attribute "logLevel" affects the logging after the SonargraphBuild engine has been started. Setting it to "debug" and below will generate additional debug information into the XML report.

WARNING

Setting the attribute reportType to "standard" only generates metric values for "system" and "module" levels. "full" generates values for all element levels, but results in a significant larger report!

7.2. Parameters of Task "sonargraphReport"

The following table lists all parameters that are available to create a report for an existing Sonargraph system. The build can be marked as failed based on a failSet. See Section 7.4, “Gradle FailSet Configuration”.

Attribute	Mandatory	Description	Default
sonargraphBuildVersion	No	Allows you to use a specific or restricted version of SonargraphBuild. Can be used in combination with 'autoUpdate'. As an example, if you specify '8.7' the newest available version starting with '8.7' will be used. If you specify '8.7.0.361' you are locked on that specific version of SonargraphBuild. There are two special values: "same" and "newest". If "same" is defined the version of SonargraphBuild must be the same as the version of the Maven plugin. If "newest" is defined the plugin will always try to use the newest version of SonargraphBuild.	same
autoUpdate	No	If the plugin is configured to download SonargraphBuild automatically, this parameter decides if it also should be updated automatically if a new version becomes available.	false

Attribute	Mandatory	Description	Default
useHttpProxyHost	No	If true, the proxy configuration of the Gradle settings is used. The plugin will use this proxy for all HTTP communication.	false
installationDirectory	No	Installation directory of SonargraphBuild. If unspecified the plugin will automatically download SonargraphBuild. The version to be downloaded can be controlled by the parameter 'sonargraphBuildVersion'.	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. Possible values: Java, CSharp, CPlusPlus.	Java
logFile	No	Path of the log file to be used for SonargraphBuild.	${project.buildDir}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
compilerDefinitionPath	No	The path to the active compiler definition file to be used for parsing a C/C++ system. If a built-in or automatically generated definition should be used, prefix the definition with "CPlusPlus:", e.g. CPlusPlus:GnuCpp.cdef. **NOTE:** If the standalone Sonargraph application is used on the same machine with the same user and this parameter is empty, the active definition specified with the standalone application will be used.	If empty, the default compiler definition for the build server's operating system is used: GnuCpp.cdef (Linux), CLang.cdef (Mac), VisualCpp*.cdef (Windows, generated definition for the latest Visual Studio installation)
systemDirectory	No	Directory of the Sonargraph System (xyz.sonargraph)	${project.buildDir}/ ${project.group} .sonargraph
overrideSonargraphWorkspace	No	If true the workspace will contain the output directories of the Maven build instead of the ones defined in the Sonargraph system.	true
includeTestCode	No	If true the workspace will also contain the test source and test class file directories.	false
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. All scripts, analyzer configurations and architecture files present in the system are	No default

Attribute	Mandatory	Description	Default
		ignored. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm"	
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	${project.buildDir}
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	${project.buildDir}/sonargraph
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
prepareForSonarQube	No	Creates an XML report and stores it at ${basedir}/${target}/sonargraph/sonargraph-sonarqube-report.xml, where the SonarQube Sonargraph plugin expects it.	false

Table 7.1. Configuration for Task/Extension "sonargraphReport"

Related topics:

- Section 7.1, “Gradle Tips and Best Practices”
- Section 7.4, “Gradle FailSet Configuration”
- Section 7.5, “Example Gradle Build File”

7.3. Configuration for Task "sonargraphDynamicReport"

The following table lists all parameters that are available to create a report for a Java project where no Sonargraph system has been defined. A Sonargraph system is created on the fly based on the workspace information contained in the Gradle project setup. The build can be marked as failed based on a failSet. See Section 7.4, “Gradle FailSet Configuration”.

Attribute	Mandatory	Description	Default
sonargraphBuildVersion	No	Allows you to use a specific or restricted version of SonargraphBuild. Can be used in combination with 'autoUpdate'. As an example, if you specify '8.7' the newest available version starting with '8.7' will be used. If you specify '8.7.0.361' you are locked on that specific version of SonargraphBuild. There are two special values: "same" and "newest". If "same" is defined the version of SonargraphBuild must be the same as the version of the Maven plugin. If	same

Attribute	Mandatory	Description	Default
		"newest" is defined the plugin will always try to use the newest version of SonargraphBuild.	
autoUpdate	No	If the plugin is configured to download SonargraphBuild automatically, this parameter decides if it also should be updated automatically if a new version becomes available.	false
useHttpProxyHost	No	If true, the proxy configuration of the Gradle settings is used. The plugin will use this proxy for all HTTP communication.	false
installationDirectory	No	Installation directory of SonargraphBuild. If unspecified the plugin will automatically download SonargraphBuild. The version to be downloaded can be controlled by the parameter 'sonargraphBuildVersion'.	No default
activationCode	No	Sonargraph license activation code. If this parameter is not specified, you must specify a license file parameter (see below).	No default
licenseFile	No	Sonargraph license file location. If this parameter is not specified, you must specify the activation code parameter (see above).	No default
languages	No	The languages that should be initialized, separated by ",". The fewer languages are specified the faster the startup of SonargraphBuild. Possible values: Java, CSharp, CPlusPlus.	Java
logFile	No	Path of the log file to be used for SonargraphBuild.	${project.buildDir}/ sonargraph_build.log
logLevel	No	Level of logging detail. One of: off, error, warn, info, debug, trace, all	info
systemBaseDirectory	No	The directory where the Sonargraph System (${artifactId}.sonargraph) is created.	${project.buildDir}
systemId	No	A system id should stay constant over the lifetime of a software system and should be also unique with respect to other systems. If you anticipate that the the group id or artifact id of the root pom might change you should assign a value to this parameter.	${project.group}_ ${project.name}
useGroupIdInModuleName	No	If true the module names will use group id and artifact id as their name, separated by an underscore. By default only the artifact id is used as the module name.	false
includeTestCode	No	If true the workspace will also contain the test source and test class file directories.	false
virtualModel	No	The virtual model to be used when checking for issues. This parameter overrides the default virtual model that is set when the system is opened. The default virtual model is "Modifiable.vm", if virtual models are licensed, "Parser" if not licensed. **Parameter can only be used with Sonargraph Architect license.**	No default
qualityModelFile	No	The path to the quality model file (xyz.sgqm) that should be applied for the report creation. All scripts, analyzer configurations and	No default

Attribute	Mandatory	Description	Default
		architecture files present in the system are ignored. Built-in quality models are the language-independent "Sonargraph:Default.sgqm" and language-specific "Sonargraph:Java.sgqm", "Sonargraph:CSharp.sgqm" and "Sonargraph:CPlusPlus.sgqm"	
snapshotDirectory	No	Target directory for the created snapshot. Only if either this parameter or snapshotFileName is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	${project.buildDir}
snapshotFileName	No	The target file name (without extension). Only if either this parameter or snapshotDirectory is provided, a snapshot will be generated. Parameter can only be used with Sonargraph Architect license.	<system-name>_<timestamp>
reportDirectory	No	Target directory for the created report	${project.buildDir}/sonargraph
reportFileName	No	The target file name (without extension)	<system-name>_<timestamp>
reportType	No	"standard" only creates metric information of system and module level. "full" creates metric information of all levels.	standard
reportFormat	No	"xml", "html" or "xml, html"	html
prepareForSonarQube	No	Creates an XML report and stores it at ${basedir}/${target}/sonargraph/sonargraph-sonarqube-report.xml, where the SonarQube Sonargraph plugin expects it.	false

Table 7.2. Configuration for Task/Extension "sonargraphDynamicReport"

Related topics:

- Section 7.1, “Gradle Tips and Best Practices”
- Section 7.4, “Gradle FailSet Configuration”
- Section 7.5, “Example Gradle Build File”

7.4. Gradle FailSet Configuration

The following elements allow to mark a build as failed. An example is shown in the next section Section 7.5, “Example Gradle Build File”.

TIP

The issue type that must be specified for include/exclude definitions can be determined via the Properties View of *Sonargraph* .

TIP

If you want the build to fail only for newly introduced issues, apply resolutions like "Ignore" or "Fix" to the already know issues in *Sonargraph* and only filter for resolution value "none".

TIP

If you want the build to fail because of certain metric values (e.g. Lines of Code per Source File), define a threshold for that metric in *Sonargraph* to create issues if files grow too large.

TIP

The include/exclude definitions are applied in the following sequence, regardless of their definition order:

1. First all include definitions are matched against the set of existing issues.

2. Then the exclude definitions are applied and the set of previously matched issues is reduced accordingly.

Element	Parameter	Mandatory	Description	Default
failSet	failOnEmptyWorkspace	No	Marks the build as failed if modules and root directories are detected but no components are found. Possible values are "true" or "false".	true
include/exclude	issueType	Yes	Name of the issue type or "any" for wildcard matching.	No default
include/exclude	severity	No	Severity of the issue. Possible values are: error, warning, info, none, any	any
include/exclude	resolution	No	The issue's resolution type. Possible values are: task, ignore, any, none	none

Table 7.3. Configuration Parameters for Build Failure

7.5. Example Gradle Build File

The following example shows how to integrate the SonargraphBuild Gradle plugin into your project. For multi-project builds it is sufficient to only add the plugin to the root project. It runs as an aggregator after all modules have been compiled. The example project in the installation contains a complete build.gradle file. Typically you would run the plugin with a command-line like the following to ensure that everything is compiled from scratch before the report is created:

```
gradlew clean build sonargraphReport
```

The following shows the relevant section of a build.gradle file that demonstrates the configuration of the Sonargraph functionality:

```
apply plugin: 'com.hello2morrow.sonargraph'

task wrapper(type: Wrapper)
{
    gradleVersion = '2.11'
}

buildscript
{
    repositories
    {
        mavenLocal()
        mavenCentral()
        maven
        {
            url 'http://maven.hello2morrow.com/repository'
        }
        maven
        {
            url 'http://maven.hello2morrow.com/snapshots'
        }
    }

    dependencies
    {
        classpath('com.hello2morrow:sonargraph-gradle-plugin:8.9.2')
    }
}

sonargraphReport
{
    // This is a activation code for Sonargraph-Explorer Build which you can use for testing.
    // Replace with your own if you have one.
    activationCode = "36E2-0F3E-643F-B4F2"
    failSet
    {
        failOnEmptyWorkspace = true
        include(issueType: "any", severity: "error", resolution: "none")
        include(issueType: "ArchitectureViolation")
        include(issueType: "any", severity: "warning")
        exclude(issueType: "ScriptCompilationError", resolution: "none")
        exclude(issueType: "ThresholdViolation")
    }
}

sonargraphDynamicReport
{
    activationCode = "36E2-0F3E-643F-B4F2"
    qualityModelFile = "Sonargraph:Java.sgqm" //default Java quality model
    failSet
    {
        failOnEmptyWorkspace = true
        include(issueType: "any", severity: "error", resolution: "none")
        include(issueType: "ArchitectureViolation")
        include(issueType: "any", severity: "warning")
        exclude(issueType: "ScriptCompilationError", resolution: "none")
        exclude(issueType: "ThresholdViolation")
    }
}
```

In this example the build will fail if the project contains a package cycle or an architecture violation without a resolution. Since the parameter 'installationDirectory' is not defined, the Maven plugin will automatically download the newest release of *SonargraphBuild* and also will keep it updated automatically. Of course this requires that the build server has access to the Internet.

Chapter 8. Integration with SonarQube

For Java projects the findings of Sonargraph can be stored and visualized in *SonarQube* using the *SonarQube Sonargraph Integration Plugin*. The plugin is currently only available for download on our website *https://www.hello2morrow.com/products/downloads* and on the plugin's GitHub page at *https://github.com/sonargraph/sonar-sonargraph-integration/releases*. The plugin is compatible with SonarQube versions 5.3 and higher.

NOTE

The plugin reads the information of the XML report that has been generated using *SonargraphBuild*. You need to configure your build pipeline accordingly.

NOTE

The plugin is currently only available for Java systems.

8.1. SonarQube Configuration

The first step is to activate Sonargraph rules in the quality profile of the project. If no Sonargraph rules are activated, the plugin will skip this project. You can either search for rules using the term "Sonargraph Integration", or the tag "sonargraph-integration".

The next step is to configure a dashboard to include the Sonargraph widgets.

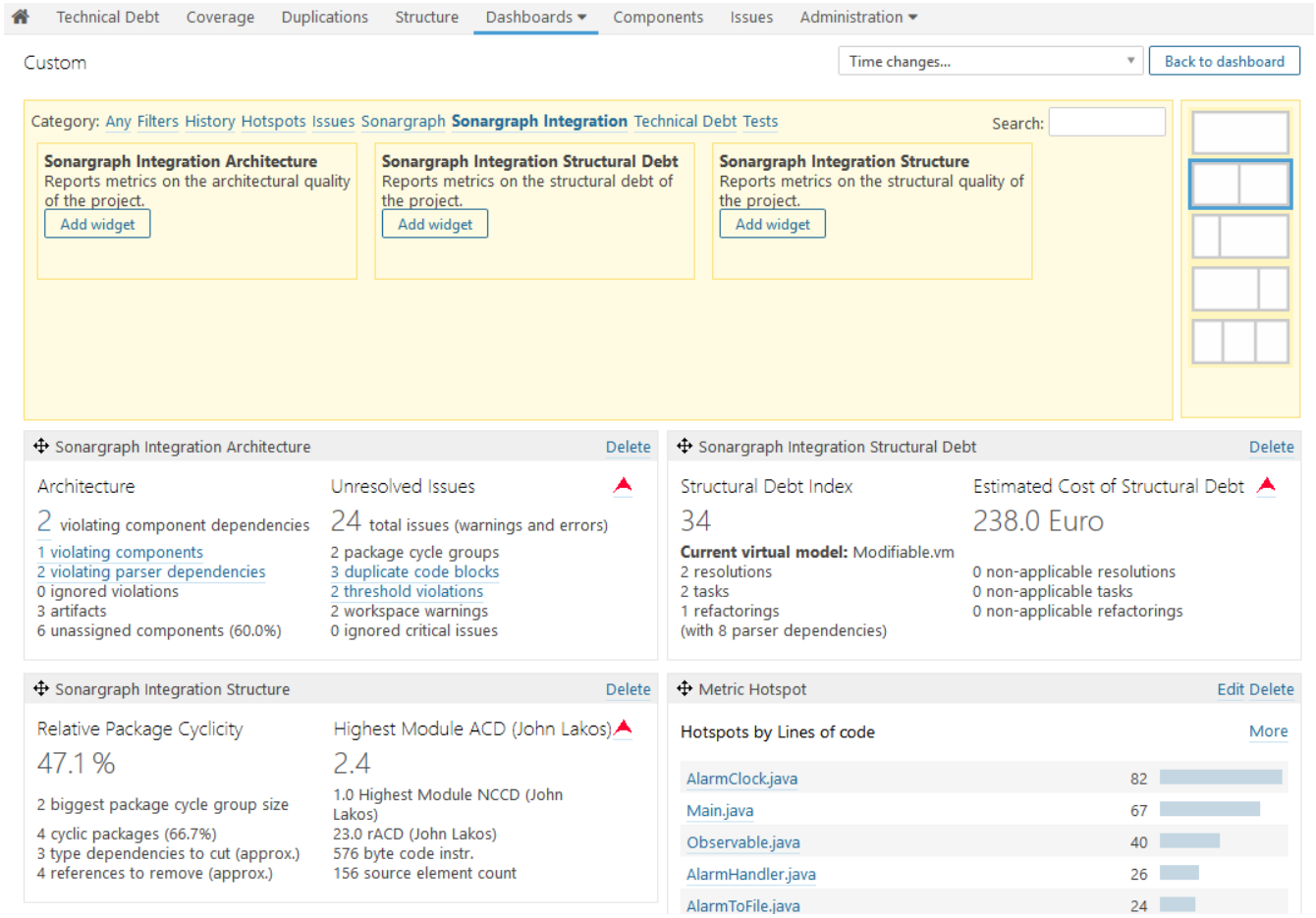

Figure 8.1. SonarQube Dashboard Configuration

Core metrics and rules of Sonargraph are pre-defined in the plugin. If you want to track custom metrics that are generated via scripts, you first need to export the report meta-data via the standalone application's menu "File" → "Export Meta-Data...". The

directory of this meta-data file needs to be specified in the plugin's configuration page. The additional metrics will be available after a restart of the SonarQube server and an additional execution of the SonarQube checks.

> **NOTE**
>
> This configuration is affecting all projects that use the Sonargraph plugin. If you have several projects with different metrics, store the separate meta-data files in the same directory. The plugin will merge the info of the different configuration files.

Related topics:

- See the section about "Workspace Profiles" in the user manual of the standalone application, if the root directories on your build server do not match the workspace definition.
- Chapter 6, *Integration with Maven*
- Chapter 7, *Integration with Gradle*
- Chapter 5, *Integrating with Ant*

8.2. SonarQube Maven Configuration

If you use the SonarQube Maven plugin, you must set the following parameter in the configuration of the SonargraphBuild Maven plugin in your project's pom.xml:

```
<configuration>
    <prepareForSonarQube>true</prepareForSonarQube>
    ...
</configuration>
```

The SonargraphBuild Maven plugin will automatically create an XML report (if not already configured) and will copy the report to ${target}/sonargraph/sonargraph-sonarqube-report.xml for the root project and all modules (excluding those with packaging "pom").

The example project contains an example pom.xml and also a batch file that demonstrates how the check can be called from the command-line.

Related topics:

- Chapter 6, *Integration with Maven*
- Section 6.5, “Example POM”

8.3. SonarQube Gradle Configuration

If you use the SonarQube Gradle plugin, you must set the following parameter in the configuration of the SonargraphBuild tasks in your project's build.gradle:

```
sonargraphReport
{
    activationCode = "36E2-0F3E-643F-B4F2"
    prepareForSonarQube = "true"
}
```

The SonargraphBuild Gradle plugin will automatically create an XML report (if not already configured) and will copy the report to ${target}/sonargraph/sonargraph-sonarqube-report.xml for the root project and all modules.

Related topics:

- Chapter 7, *Integration with Gradle*

- Section 6.5, “Example POM”

8.4. SonarQube Ant Runner Configuration

If you use the SonarQube Ant Runner the Sonargraph XML report must have been created and this report must be configured for the Sonargraph SonarQube plugin using the following parameter:

```
<property name="sonar.sonargraph_integration.report.path" value="${path.target.report}" />
```

The example project contains this configuration in the Ant build file.

Related topics:

- Chapter 5, *Integrating with Ant*

Chapter 9. Integration with Jenkins

With *Jenkins Sonargraph Integration Plugin* for *Jenkins* jobs the findings of *Sonargraph* can be used to let builds fail, or mark them unstable. Additionally *Sonargraph* metric values are stored for every build and can be visualized as charts.

9.1. Jenkins Server Configuration

The first step is to configure one or more versions of Sonargraph Build in "Manage Jenkins" → "Configure System". Click "Sonargraph Build installations..."

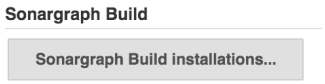

Figure 9.1. Jenkins - Sonargraph Build Configuration

and select a name, a version and an installer.

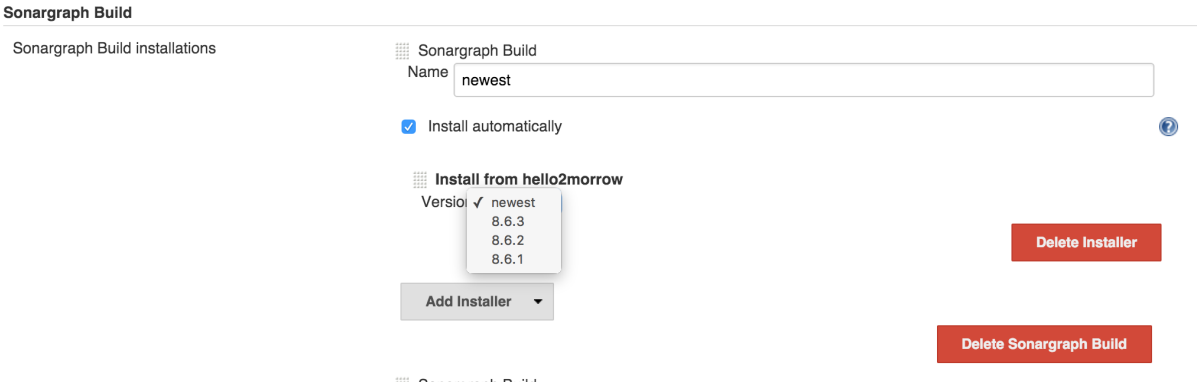

Figure 9.2. Jenkins - New Sonargraph Build

9.2. Jenkins Job Configuration

Add post build action "Sonargraph Integration Report Generation & Analysis" to your job.

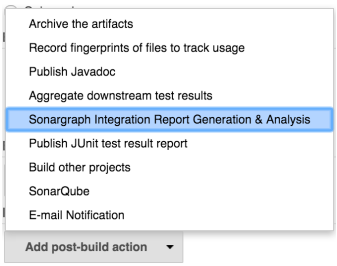

Figure 9.3. Job - Post Build Action

First decide if *Sonargraph Build* is used to create the report,

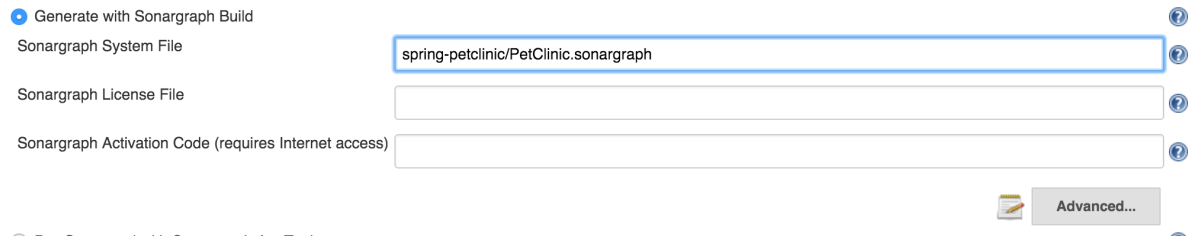

Figure 9.4. Report - Generate With Sonargraph Build

or there already exists a report generated by an upstream build action.

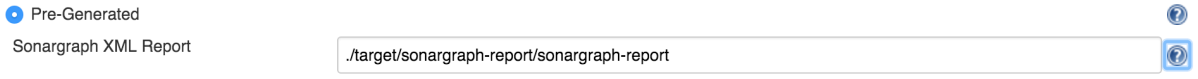

Figure 9.5. Report - Pre Generated

When *Sonargraph Build* is used to create the report fill out all required information:

Figure 9.6. Report - Standard Options

By pressing "Advanced..." some more options pop up:

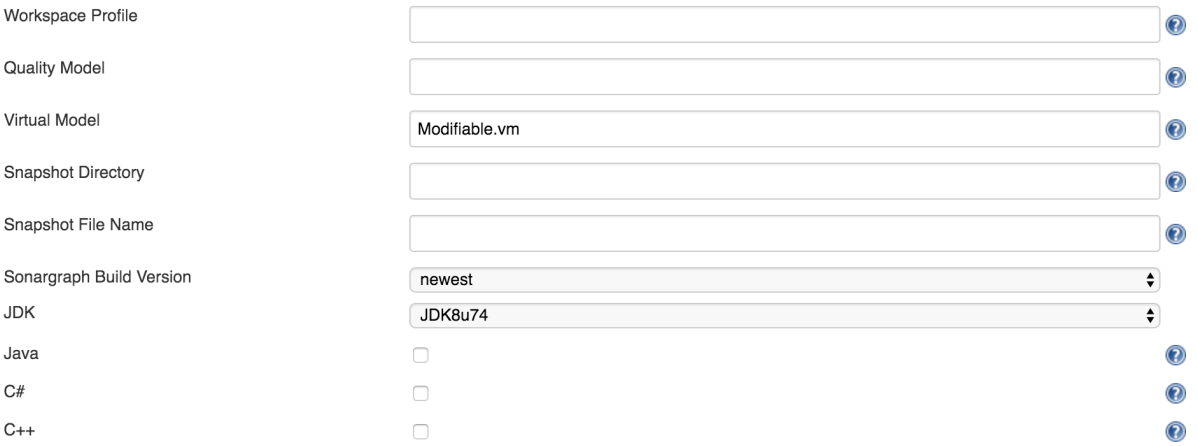

Figure 9.7. Report - Advanced Options

9.3. Charts Configuration

To see some charts, a meta-data file must be configured, and either all contained charts/metrics are shown, or a list of charts/metrics to be shown can be given. If you want to track custom metrics that are generated via scripts, you first need to export the report meta-data via the standalone application's menu "File" → "Export Meta-Data...".

Figure 9.8. Job - Chart Configuration

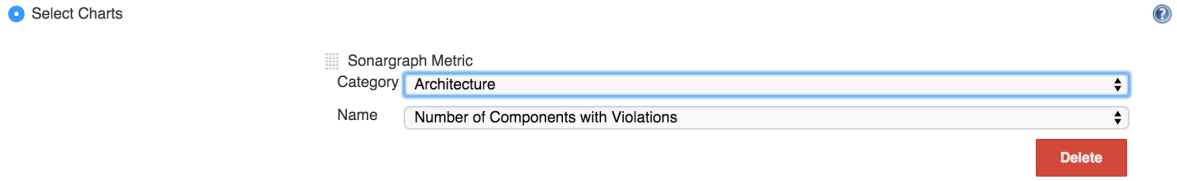

Figure 9.9. Job - Select Charts

9.4. Build Configuration

Finally the reasons for marking the build as failed or unstable can be set:

Figure 9.10. Mark build failed or instable

Related topics:

- See the section about "Workspace Profiles" in the user manual of the standalone application, if the root directories on your build server do not match the workspace definition.
- Chapter 6, *Integration with Maven*

Chapter 10. Trademark Attributions, Library License Texts, and Source Code

Eclipse is a trademark of Eclipse Foundation, Inc.

IntelliJ is a trademark of JetBrains s.r.o.

Java and all Java-based trademarks are trademarks of Oracle Corporation in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft, and Windows are trademarks of Microsoft Corporation in the United States and other countries.

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

Chapter 11. Legal Notice

Appendix A. SonargraphBuild API Documentation

SonargraphBuild API is documented via JavaDoc that is available within the installation of the product.

Link to JavaDoc of SonargraphBuild API.

Index