CodeSonar Plugin

Plugin Information
View CodeSonar on the plugin site for more information.

Developed by

Sponsored by

The CodeSonar plugin for Jenkins is developed by Praqma and sponsored by GrammaTech, Inc. It is maintained in the scope of Joint Open Source Roadmap Alliance (JOSRA).

Introduction

The CodeSonar plugin collects project analysis data from a designated CodeSonar hub.

- Historical data about CodeSonar warning counts and code size is presented in the Job Dashboard.
- The plugin can be configured to change the build result if the CodeSonar analysis results meet specified conditions.

The following documentation cites relevant sections in the CodeSonar manual. These citations take the form
MANUAL: Subject > ... > Page Title
where Subject > .... > Page Title denotes a navigation path through the CodeSonar manual table of contents.

About CodeSonar ®

CodeSonar, GrammaTech's flagship static analysis software, identifies programming bugs that can result in system crashes, memory corruption, leaks, data races, and security vulnerabilities.

By analyzing both source code and binaries, CodeSonar empowers developers to eliminate the most costly and hard-to-find defects early in the application development lifecycle.

Setting Up The Plugin

These instructions assume that you have...

- ... installed Jenkins.
- ... established a Jenkins job to build your software.
- ... installed CodeSonar.
- ... installed the CodeSonar plugin for Jenkins.

Setting up the plugin involves three steps, each detailed below.

- A. Make sure CodeSonar is ready to analyze your software
- B. Incorporate the CodeSonar build/analysis in your Jenkins job
- C. Apply the CodeSonar plugin to your Jenkins job

A. Make sure CodeSonar is ready to analyze your software

Work through the following steps to make sure that CodeSonar is in a suitable state to be invoked by your Jenkins job.

1. Make sure that the path to `/codesonar/bin` is in the `PATH` of the user who is running Jenkins.
    - Otherwise, your `codesonar analyze` command will have to specify the path to the `codesonar` executable.
2. Start the CodeSonar hub to use for recording the analysis results (if it is not already running).
   MANUAL: How CodeSonar Works > CodeSonar Structure > Hub > Starting a Hub
   The remainder of these instructions will refer to the hub location as *host:port*.
3. Establish a project directory and project name for the CodeSonar project that will be built and analyzed.
    - If you have previously analyzed your software with CodeSonar, you can use the existing project infrastructure.
    - Otherwise, choose a project directory, and create the directory if it does not already exist.
      In either case, make sure the project directory has a suitable location and read/write settings. If Jenkins is running with different OS credentials to your own, remember to take this into account.
    - Use the same project directory and project name every time you perform the CodeSonar build/analysis for a given project.
    - The project directory should not be deleted at the end of the build: the CodeSonar GUI needs to interact with its contents, and incremental builds need information that is stored there.
        - Make sure that the project directory is in a location where Jenkins will not automatically delete it after running the job. For example, it is probably a good idea to locate it outside the Jenkins workspace.
        - Similarly, take steps to ensure that your other build tools will not delete the project directory.
    - Set the project directory permissions to allow the Jenkins process to read and write to it.
      The remainder of these instructions will refer to the project directory as *projdir* and the project name as *proj-name*.
4. If the project directory does not include a general project configuration file (for example, because you just created the directory in the previous step), create one now:

   ```
   codesonar create-conf projdir/proj-name
   ```

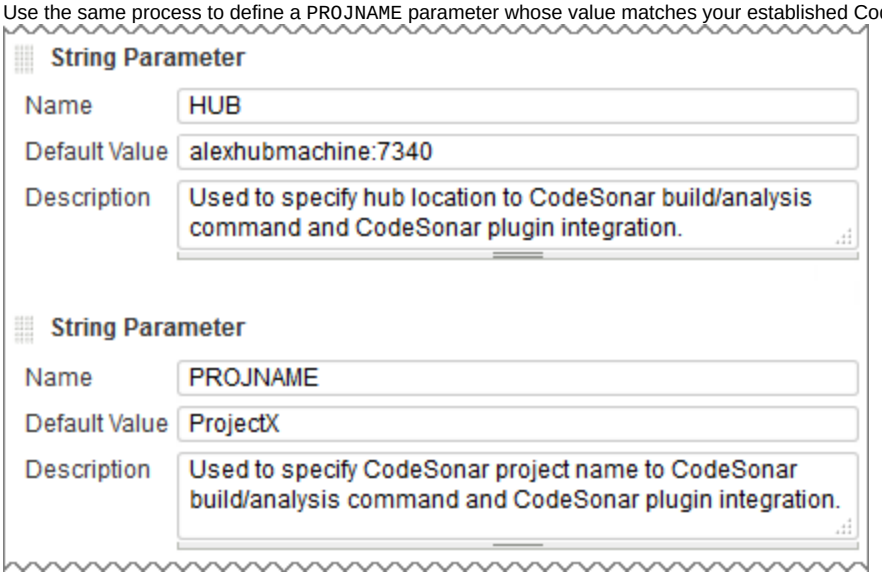
   MANUAL: Using CodeSonar > Building and Analyzing Projects > Options, Preferences, and Configuration Files > Configuration Files
5. Edit the general project configuration file (*projdir*/*proj-name*.conf) to specify your required configuration parameter settings (unless the factory settings are suitable).
   MANUAL: Using CodeSonar > Building and Analyzing Projects > Options, Preferences, and Configuration Files > Compiler-Independent Configuration File Parameters for CodeSonar
   In particular:
    - You may wish to specify one or more `CFLAGS_APPEND` rules.
    - If you are performing a clean build every time, set `INCREMENTAL_BUILD=No`.
6. Make sure there is a CodeSonar launch daemon running on the analysis machine, with the same owner as the Jenkins process.
   MANUAL: How CodeSonar Works > Build and Analysis > cslaunchd: The CodeSonar Launch Daemon
    - *If the analysis machine is running Windows*, check to see whether there is a `cslaunchd` service on the analysis machine, with the same owner as the Jenkins process. If not, set one up.
      MANUAL: Using CodeSonar > Building and Analyzing Projects > Continuous Integration > Using CodeSonar With Continuous Integration Tools
      Note that if Jenkins is running as a service, its owner will usually be SYSTEM.
    - *Otherwise*, arrange to start the launch daemon at system startup.
7. Go on to **Incorporate the CodeSonar build/analysis in your Jenkins job**.

B. Incorporate the CodeSonar build/analysis in your Jenkins job

You will incorporate the CodeSonar build/analysis in your Jenkins job by extending the current contents of the **Build** section as described in the following steps.

1. View the Job Dashboard for the Jenkins job that is building your software.
2. Click **Configure** to open the **Job Configurations** page.
3. Define a `HUB` parameter for the job so you can use it both for the CodeSonar analysis invocation and to configure the CodeSonar plugin later:
    a. Make sure **This build is parameterized** is selected.
    b. Under **This build is parameterized**, click **Add Parameter**, then select **String Parameter** from the menu that pops up.
    c. Jenkins will display a set of fields for setting up your new parameter. Fill them out as follows.
        - Name: `HUB`
        - Default Value: the location (*host:port*) of your CodeSonar hub. For example, `alexhubmachine:7340`.
        - Description: you may want to enter a short description to remind yourself why you have this variable.

4. Use the same process to define a `PROJNAME` parameter whose value matches your established CodeSonar project name (*proj-name*).



5. Edit the **Build** section to integrate the CodeSonar build/analysis.
   Remember to specify authentication options in your build/analysis commands if they will be required by your hub.
   MANUAL: How CodeSonar Works > CodeSonar Structure > Hub > Authentication and Access Control

Project Language	Editing the Build Section
C, C++	For every existing build step that involves C/C++ compilation, edit the build step to incorporate the CodeSonar build/analysis command. If the current build step or steps contain one command that involves C/C++ compilation, this will involve constructing a single `codesonar analyze` command. Otherwise there are two possible approaches: • Accumulate components into a CodeSonar project by constructing a `codesonar build` command for each software build command that involves C/C++ compilation, then add a final `codesonar analyze` command to analyze the project. *or* • Replace the text of the build step or steps with an invocation of a shell script or batch file with equivalent contents, then construct a single `codesonar analyze` command based on that invocation. The `codesonar analyze` command must include the `-foreground` option. See Example 1 and Example 2.
Java	Add a new, final build step that executes the CodeSonar Java build/analysis on the bytecode produced by the other build steps. The `codesonar analyze` command must include the `-foreground` option. See Example 3. MANUAL: Using CodeSonar > Building and Analyzing Projects > Java > Build and Analysis for Java Projects
Mixed Java and C/C++	Combine the approaches for Java-only and C/C++-only projects: a. Edit the build steps to incorporate a `codesonar build` command for each software build command that involves C/C++ compilation. b. Add a new build step that executes `codesonar build` on any Java bytecode produced by earlier build steps. c. Add a new, final build step that invokes `codesonar analyze` to analyze the project. See Example 4 and Example 5.

6. Click **Save**.
7. Check that everything is working properly:
   a. Click **Build with Parameters**, check that the parameter settings are correct, and click **Build**.
      Jenkins will execute the updated job.
   b. Check that the Jenkins job executed successfully, and check the job's **Console Output** to ensure that the build proceeded as you expected.
      - If necessary, click **Configure** and adjust your edits, and make any other changes necessary to get your job running correctly.
      - If the CodeSonar build/analysis is not running to completion, the manual section on Troubleshooting the build may be helpful.
        MANUAL: Using CodeSonar > Building and Analyzing Projects > Troubleshooting the Build
   c. Open the CodeSonar GUI in your web browser and inspect your analysis results on the Analysis page.
      MANUAL: Using CodeSonar > GUI Reference > GUI Reference
8. Go on to **Apply the CodeSonar plugin to your Jenkins job**.

C. Apply the CodeSonar plugin to your Jenkins job

Once your Jenkins job is correctly invoking the CodeSonar analysis, you can apply the CodeSonar plugin to collect analysis information from the hub.

1. Go back to **Job Configurations** page for the Jenkins job that is building your software.
2. Under **Post-build Actions**, click **Add post-build action**, and select **CodeSonar** from the menu that pops up.
    - If CodeSonar is not a menu option, the plugin may not be installed.
3. Jenkins will display fields for you to configure this application of the plugin.
    a. Select the protocol used by your hub from the **Protocol** menu: either `http` or `https`.
    b. Enter `${HUB}` in the **hub address** field.
    c. Enter `${PROJNAME}` in the **Project name** field.
    d. Click the **Add** button next to the **Credentials** field, then fill in the **Add Credentials** form that opens and click **Add**.
        i. Set **Kind** to "Username with password" or "Certificate".
        ii. Set **Scope** to Global.
        iii. Use the remaining fields to specify the hub user account credentials that the plug-in should use in obtaining analysis information from the hub. See below for information about the permissions required and additional manual references.
        These credentials will *not* be applied to the build/analysis commands you specified in the previous step. If you want to specify authentication credentials for those commands, use the appropriate command-line authentication options.
    e. [Optional] If you want to configure one or more "CodeSonar conditions", see the descriptions below.
4. Click **Save**.
    - The list of links at the left hand side of the Job Dashboard will now include a **Latest CodeSonar Analysis** link. This navigates to the CodeSonar GUI Analysis page for the most recently executed analysis of this project.
5. Check that everything is working properly:
    a. Click **Build with Parameters**, check that the parameter settings are correct, and click **Build**.
    Jenkins will execute the updated job.
    b. Check that the Jenkins job executed successfully, and check the job's **Console Output** to ensure that the build proceeded as you expected.
        - If necessary, click **Configure** and adjust your edits, and make any other changes necessary to get your job running correctly.
6. Notice that the dashboard now contains charts of "Total number of warnings" and "Lines of Code" (if it doesn't, reload the page). These charts represent CodeSonar analysis history for this project.

Build Step Examples

These examples all assume the following:

- Build parameter `${HUB}` has been established and set to the hub location.
- Build parameter `${PROJNAME}` has been established and set to the CodeSonar project name.
- The project directory is `/myfiles/csonar_projects/projX`

Example 1: C/C++ project; Jenkins build steps include one command that involves C/C++ compilation.

Suppose that the Jenkins job build step text is:

```
cd /myfiles/src/projX && make normal
```

Then replace the build step text with:

```
cd /myfiles/src/projX && codesonar analyze /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB} make
normal
```

Example 2: C/C++ project; Jenkins build steps include multiple commands that involve C/C++ compilation.

Suppose that the Jenkins job build step text is:

```
cd /myfiles/src/projX
rm -f *.o
gcc -c A.c
gcc -c B.c
gcc -c C.c
```

There are several possible approaches.

Option 1

Replace the build step text with:

```
cd /myfiles/src/projX
rm -f *.o
codesonar build /myfiles/csonar_projects/projX/${PROJNAME} -foreground  ${HUB} gcc -c A.c
codesonar build /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB} gcc -c B.c
codesonar build /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB} gcc -c C.c
codesonar analyze /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB}
```

Option 2	Collect the build step text into a single shell script `/path/to/dir/mybuildscript.sh`:
	`cd /myfiles/src/projX`
	`rm -f *.o`
	`gcc -c A.c`
	`gcc -c B.c`
	`gcc -c C.c`
	then replace the build step text with:
	`cd /path/to/dir && codesonar analyze ${PROJNAME} -foreground ${HUB} sh -xe mybuildscript.sh`

Option 3	Collect the build step text into a single batch file `path\to\dir\mybuildbat.bat`:
	`cd \myfiles\src\projX`
	`rm -f *.o`
	`gcc -c A.c`
	`gcc -c B.c`
	`gcc -c C.c`
	then replace the build step text with:
	`codesonar analyze ${PROJNAME} -foreground ${HUB} path\to\dir\mybuildbat.bat`

Example 3: Java project

Suppose that the Jenkins job writes Java build output to `/myfiles/buildoutput/classes`.

Then add a new "Execute shell" build step with the following contents.

```
codesonar analyze /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB} cs-java-scan /myfiles
/buildoutput/classes
```

Example 4: Mixed C/C++ and Java project; single build command

Suppose that the Jenkins job build step text is:

```
cd /myfiles/src/projX
make all
```

and that the Jenkins job writes Java build output to `/myfiles/buildoutput/classes`.

Then replace the build step text with:

```
cd /myfiles/src/projX
codesonar build /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB} make all
codesonar build /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB} cs-java-scan /myfiles/buildoutput
/classes
codesonar analyze /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB}
```

Example 5: Mixed C/C++ and Java project; multiple build commands

Suppose the Jenkins job build step text is:

```
cd /myfiles/src/projX
rm -f *.o
rm -f *.class
```

```
gcc -c A.c
gcc -c B.c
javac J.java
```

and that the Jenkins job writes Java build output to `/myfiles/buildoutput/classes`.

There are several possible approaches.

Option 1	Move the build text to a Makefile, shell script, batch file, or similar, then follow the approach illustrated in Example 4.

Option 2	Replace the build step text with:
	`cd /myfiles/src/projX`
	`rm  \-f  *.o`
	`rm  \-f  *.class`
	`codesonar build /myfiles/csonar_projects/projX/${PROJNAME} -foreground  ${HUB} gcc -c A.c`
	`codesonar build /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB} gcc -c B.c`
	`javac J.java`
	`codesonar build /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB} cs-java-scan /myfiles /buildoutput/classes`
	`codesonar analyze /myfiles/csonar_projects/projX/${PROJNAME} -foreground ${HUB}`

Configuration Settings

The CodeSonar plugin for Jenkins has two required configuration settings and a number of optional ones.

- Protocol, Hub address, Project name are always required.
- The Credentials setting is required if special user `Anonymous` does not have sufficient permissions to interact with the CodeSonar analysis information on the hub.
- Users can also specify zero or more CodeSonar conditions.

Required Configuration Settings

Setting Name	Description	Notes
Protocol	The protocol that should be used to communicate with the hub: either `http` or `https`. MANUAL: How CodeSonar Works > CodeSonar Structure > Hub > Hub Location	Always explicitly specify `https` if you are concerned about security, otherwise you may be sending unencrypted data to an imposter hub.
Hub address	The location of the CodeSonar hub that you are using to manage your analysis results, in format `hostname:port`. MANUAL: How CodeSonar Works > CodeSonar Structure > Hub > Hub Location	If you have set up a ${HUB} parameter for the job, you can use it here.
Project name	The CodeSonar project name. MANUAL: How CodeSonar Works > CodeSonar Structure > Project	If you have set up a ${PROJNAME} parameter for the job, you can use it here.

Optional Configuration Settings

Setting Name	Description	Notes
Credentials	The hub user account credentials that the plugin should use when obtaining analysis information from the hub. MANUAL: How CodeSonar Works > CodeSonar Structure > Hub > Hub User Accounts MANUAL: Role-Based Access Control (RBAC) > RBAC: Role-Permissions	The hub user account you specify here must have sufficient permissions to access the relevant analysis information: **Global Permissions**: `G_LIST_PROPERTIES`, `G_LIST_USERS`, `G_SIGN_IN`, `G_SIGN_IN_CERTIFICATE` or `G_SIGN_IN_PASSWORD`, depending on the **Kind** of credentials you are specifying. **Permissions on analyzed project ${PROJNAME}**: `ANALYSIS_ANNOTATE`, `ANALYSIS_EXISTS`, `ANALYSIS_READ`, `ANALYSIS_WARNING_EXISTS`, `ANALYSIS_WRITE`

	PROJECT_EXISTS PROJECT_READ
Permissions on Named Searches	NAMEDSEARCH_READ for the following built-in warning searches: active, new, active and new.

You do not need to configure credentials if special user `Anonymous` has sufficient permissions to obtain analysis information from the hub: in this case, the plugin will interact with the hub in an anonymous session (`Anonymous` does not need `G_SIGN_IN_CERTIFICATE` or `G_SIGN_IN_PASSWORD`).
Note that these credentials are not used to authenticate CodeSonar build/analysis commands issued by your Jenkins job. If you want to specify authentication credentials for those commands, use the appropriate command line authentication options.

CodeSonar Conditions

Users can specify zero or more *CodeSonar conditions*. Each condition specifies a bound on some particular property of the CodeSonar analysis results, along with the build result setting to be applied if the property's value lies outside the specified bound.

There are six different condition types.

CodeSonar Condition Type	Set the build result to the specified value if ...
Cyclomatic complexity	... one or more procedures has cyclomatic complexity (as determined by CodeSonar) that exceeds the specified limit. MANUAL: How CodeSonar Works > CodeSonar Structure > Metrics
Red alerts	... the number of red alerts from CodeSonar analysis exceeds the specified limit. MANUAL: Using CodeSonar > GUI Reference > Alerts
Warning count increase: new only	... the number of *new* warnings issued by the CodeSonar analysis exceeds the number issued for the previous analysis by more than the specified percentage. MANUAL: How CodeSonar Works > CodeSonar Structure > Warnings > Warnings: Instances and Groups
Warning count increase: overall	... the number of warnings issued by the CodeSonar analysis exceeds the number issued for the previous analysis by more than the specified percentage. MANUAL: How CodeSonar Works > CodeSonar Structure > Warnings > Warnings: Instances and Groups
Warnings count increase: specified score and higher	... the number of warnings in the specified score range issued by the CodeSonar analysis exceeds the number issued for the previous analysis by more than the specified percentage. MANUAL: How CodeSonar Works > CodeSonar Structure > Warnings > Warnings: Instances and Groups
Yellow alerts	... the number of yellow alerts from the CodeSonar analysis exceeds the specified limit. MANUAL: Using CodeSonar > GUI Reference > Alerts

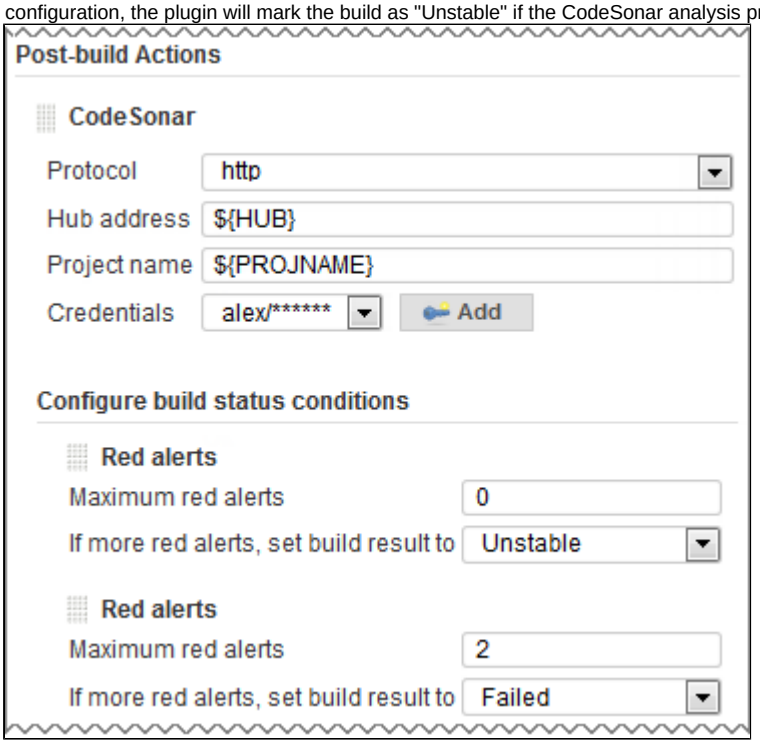
Where conditions are concerned with increase with respect to a"previous analysis", the previous analysis is considered to be the CodeSonar analysis invoked by the previous Jenkins build. Note that there may be more recent analyses on the CodeSonar hub if they were invoked from outside Jenkins. The first analysis of the project has no "previous analysis": in this case, the increase-based conditions are considered to be unsatisfied. Warning count for these conditions is determined with respect to the default warning visibility setting for the hub user account whose credentials you have supplied.

You can specify multiple conditions, including multiple conditions of a single type. When multiple conditions are specified they are all checked and the "worst" marking from the set of satisfied conditions is applied. That is:

- If any satisfied condition specifies "Failed", the plugin will mark the build as "Failed".
- If no satisfied condition specifies "Failed" but at least one satisfied condition specifies "Unstable", the plugin will mark the build as "Unstable".

Configuration Example

With this configuration, the plugin will mark the build as "Unstable" if the CodeSonar analysis produces one or two red alerts, but "Failed" if there are three or more.



Jenkins Job DSL

Available options

```
job{
    publishers{
        codeSonar(String hubAddress, String projectName){
            cyclomaticComplexity(int maxComplexity, boolean markAsFailed)
            redAlert(int maxAlerts, boolean markAsFailed)
            yellowAlert(int maxAlerts, boolean markAsFailed)
            newWarningCountIncrease(float percentage, boolean markAsFailed)
            overallWarningCountIncrease(float percentage, boolean markAsFailed)
            rankedWarningCountIncrease(int minRank, float percentage, boolean markAsFailed)
        }
    }
}
```

Example

```
job('myProject_GEN'){
    publishers{
        codeSonar('hub','proj'){
            cyclomaticComplexity(20, false)
            redAlert(3, true)
            yellowAlert(10, false)
            newWarningCountIncrease(5, true)
            overallWarningCountIncrease(5, false)
            rankedWarningCountIncrease(30, 5, true)
        }
    }
}
```

Support and Contact

Please send us an email on support@praqma.net if you have a request or question regarding the plugin.

Issues

type	key	summary

⚠ Can't show details. Ask your admin to whitelist this Jira URL.

View these issues in Jira

Changes

Version 2.0.9 (TBD)

- Java 11 compatibility
- Jenkins core requirement is upped to 2.164

Version 2.0.8 (March 1, 2019)

- Added new threshold option: https://github.com/Praqma/codesonar-plugin/issues/51

Version 2.0.6 (February 6, 2018)

- The plugin is now more memory efficient
- Less data is stored on master

Version 2.0.5 (March 22, 2017)

- Added support for pipeline job types.
- Added support for projects inside tree structures on the hub.

Version 2.0.4 (September 12, 2016)

- Fixed compatibility bugs regarding codesonar 4.2
- fixed bug where plugin used to fail with a nullpointer exception after a fresh restart

Version 2.0.2 (August 9, 2016)

- Fixed compatibility bug for Jenkins 1.609.2

Version 2.0.1 (June 16, 2016)

- Fixed a possible NPE related to alerts (#2)

Version 2.0.0 (June 13, 2016)

- Added compatability for codesonar 4.2+

Version 1.0.1 (October 15, 2015)

- Better error messages in console (JENKINS-30386)
- Added JobDSL support (JENKINS-30888)

Version 1.0

- Release of the first stable version