ASE TEAM 5

User Manual

HUSACCT – DEFINE

June 22, 2013

HOGESCHOOL UTRECHT
SOFTWARE ARCHITECTURE
COMPLIANCE CHECKING
TOOL

ROB UITHOL – SETH SNEL- LEANDER STOLK
BAYRAM KORKMAZ- REUBEN KROZENDIJK

CONTENTS

INTRODUCTION

The define component of the HUSACCT application is responsible for defining the logical architecture, modules and rules, and mapping the logical architecture to a physical one. In this manual we will describe how to use the define part of the application with features that the users can use.

The precondition before this part of the application can be used is:

- A work space must be created.

1 BASICS

This chapter describes the basics of the HUSACCT.

1.1 KNOWLEDGE

WHAT IS A LOGICAL ARCHITECTURE?

The logical architecture is a detailed structure which defines what has to be done to support the user services. It defines the processes that perform functions and the information or data flows that are shared between these processes. Logical architectures do not include physical server names or addresses. They do include any business services, application names and details, and other relevant information for development purposes.

WHAT ARE LAYERS?

Layering in architectures is a common concept. Logical **layers** are merely a way of organizing your application. Typical layers include Presentation, Business and Data – the same as the traditional 3-tier model. All we are doing is discussing a way of organizing a code into a set of layers defined by specific functions.

MODULE TYPES

Within the HUSACCT you can define different kind of modules. These modules will have a different effect from others and will open new options. The four different module types are: “Subsystem”, “Layer”, “Component” and “External Library”.

- **Subsystem**
  A Subsystem is just a normal module with no strings attached.
- **Layer**
  A Layer is a module which can be restricted to perform Skip Calls and Back Calls. These rules are only available on Layer modules.
- **Component**
  A component is a module that can only be approached through the façade. The façade is automatically created with the component.
- **External Library**
  An external library is a library that is used from a third-party.

1.2 RULE TYPES

The following rules are supported in the HUSACCT application and can be defined and validated. We will now give a brief explanation per rule type.

INTERFACE CONVENTION

Each class in the specified module must implement the specified interface. An exception on this rule will exclude certain classes or packages from this rule.

SUPERCLASS INHERITANCE CONVENTION

Each class in the specified module must extend the specified class. An exception on this rule will exclude certain classes or packages from this rule.

NAMING CONVENTION

All the classes and/or packages in the specified module must meet to the specified Regular Expression (see table 1). The regular expressions are case-sensitive.
The possible exceptions are:

- *A possibility to exclude classes/packages from the regex*
- *Add another naming convention for classes/packages to overrule the main rule for certain classes/packages.*

EXPRESSION	VALIDATES	RESULT
Friends	domain.locationbased.foursquare.History	False
	domain.locationbased.latitude.Friends	True
	infrastructure.socialmedia.locationbased.foursquare.FriendsDAO	True
	infrastructure.socialmedia.locationbased.foursquare.MyFriendsDAO	True
***Account**	domain.locationbased.foursquare.MyAccount	True
	domain.locationbased.latitude.Map	False
	infrastructure.socialmedia.locationbased.foursquare.AccountDAO	False
DAO *	infrastructure.socialmedia.locationbased.foursquare.DAOFourSquare	True
	infrastructure.socialmedia.locationbased.foursquare.IMap	False
	domain.locationbased.foursquare.History	False

TABLE 1

The first regex of table 1 enforces that a class/packages must contain the word 'Friends'.
The second regex enforces that a class/package must end with the word 'Account').
The third regex enforces that all the classes and/or packages must start with the word 'DAO'.

VISIBILITY CONVENTION

All the classes and/or packages in the specified module must have the specified visibility or lower.
The possible exceptions are:

- *a possibility to exclude classes/packages*
- *add another visibility convention for classes/packages that were specified in the main rule*

IS ALLOWED TO USE

This rule can only be defined as exception rule. All the classes/packages between two logical modules are allowed to have a dependency. This is normally the case, that's why it can only be defined as an exception rule; otherwise it would have no use.

IS NOT ALLOWED TO USE

All the classes/packages between two logical modules are not allowed to have a dependency with each other.
An exception on this rule will exclude certain classes or packages from this rule and allow them to have dependencies with each other.

IS ONLY ALLOWED TO USE

One defined logical module is only allowed to have dependencies with another defined logical module.
An exception on this rule will exclude certain classes or packages from this rule and allow them to have dependencies with other modules.

IS ONLY MODULE ALLOWED TO USE

The selected module is the only module in the architecture that is allowed to use the specified module.
This rule will restrict all other modules from having dependencies with the specified module.
There are no possible exception rules.

MUST USE

At least one class/package from the selected module needs to have a dependency with the other.
The possible exception is to define a 'is allowed to use' rule. An exception on this rule will exclude certain classes or packages from this rule and allow them to don't have any dependencies with each other.

IS NOT ALLOWED TO MAKE BACK-CALL

This rule can only be applied on modules of type layer. A layer may not have dependency with layers that are defined with a higher hierarchal level then the layer on which the rule is applied to.
An exception on this rule will exclude certain classes or packages from this rule and allow them to have dependencies with each other.

IS NOT ALLOWED TO MAKE SKIP-CALL

This rule can only be applied on modules of type layer (see for more information the manual of the define component). A layer may not have dependencies with layers that are defined with a lower hierarchal level other than the layer directly under the layer on which the rule is applied to. An exception on this rule will exclude certain classes or packages from this rule and allow them to have dependencies with each other

1.2 LAYOUT

The layout of the define component is split into 5 areas (*figure 1)*. Each area is described below.

#1 MODULE HIERARCHY

This area shows the logical defined architecture. Here you can see the defined modules. It also allows you to select modules for the rest of the application. You can select a module by simply clicking on it.

#2 SELECTED MODULE PROPERTIES

This area shows the properties of the currently selected module. You can change the module name and/or description.

#3 SELECTED MODULE SOFTWARE UNITS

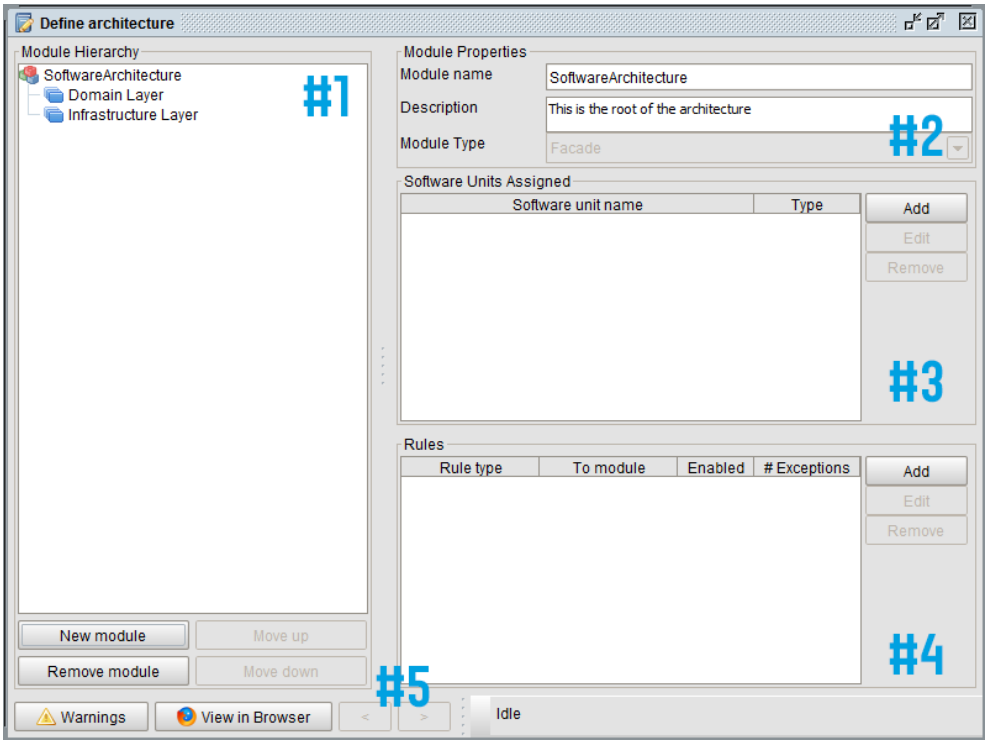
This area shows the software units that are currently mapped to the selected module. The buttons are only enabled if a module is selected.

#4 SELECTED MODULE RULES

This area shows the rules that are currently defined for the selected module. The buttons are only enabled if a module is selected.

#5 TOOLBAR

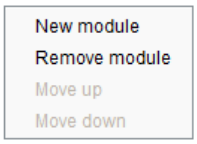
The toolbar has some nifty features that are not necessary to define the architecture, but can make it a lot easier or better. The first button is the warnings button, when clicking it, it will show a dialog with all the warnings that your architecture has. When clicking on the second button, the 'View in Browser'-button, you get a HTML-report/overview with all modules, applied rules and software units that you defined. Next to these buttons are the undo and redo buttons. Made a mistake? Undo it! Undid something you needed? Redo!



FIGUR

2 COMMON TASKS

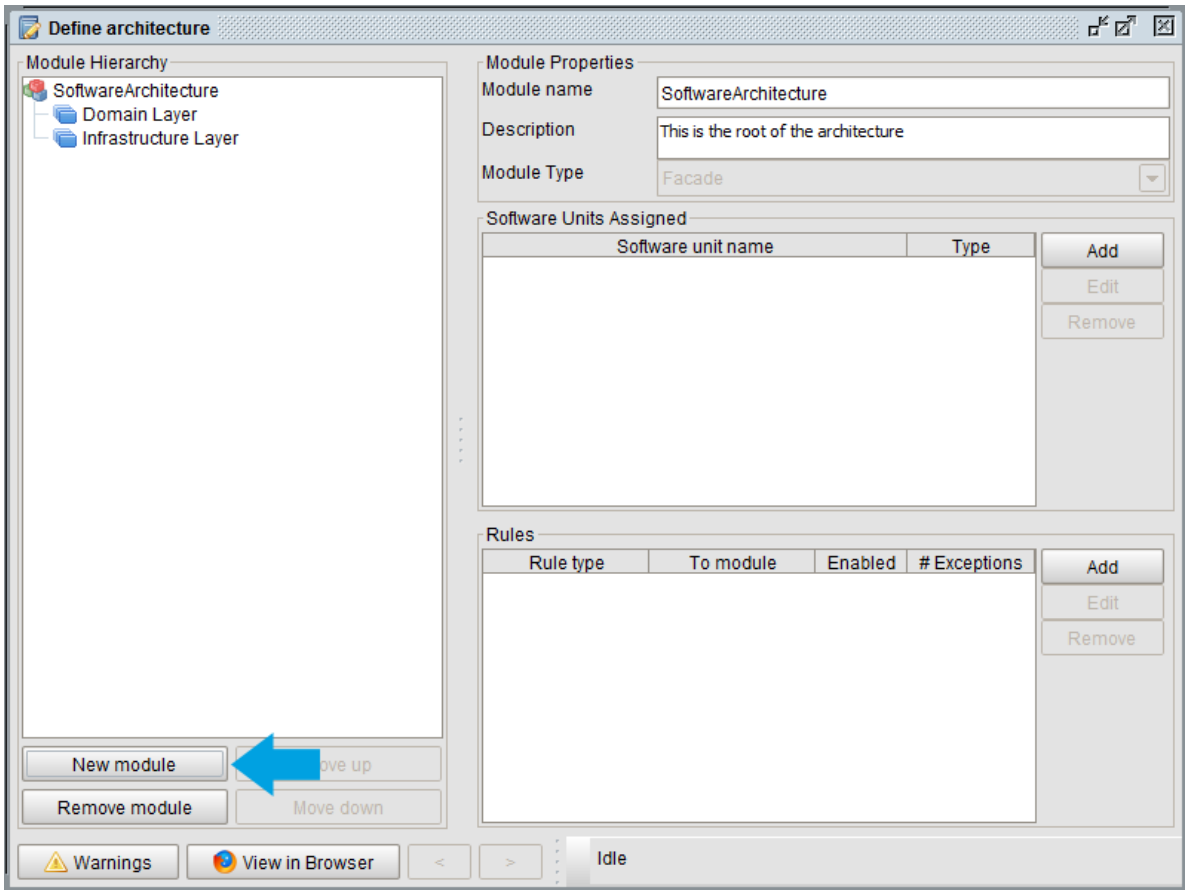
This chapter will elaborate more on the common tasks that you will need to perform to define an architecture. Keep in mind that most of these functionalities can also be accessed by using the right-click mouse button.



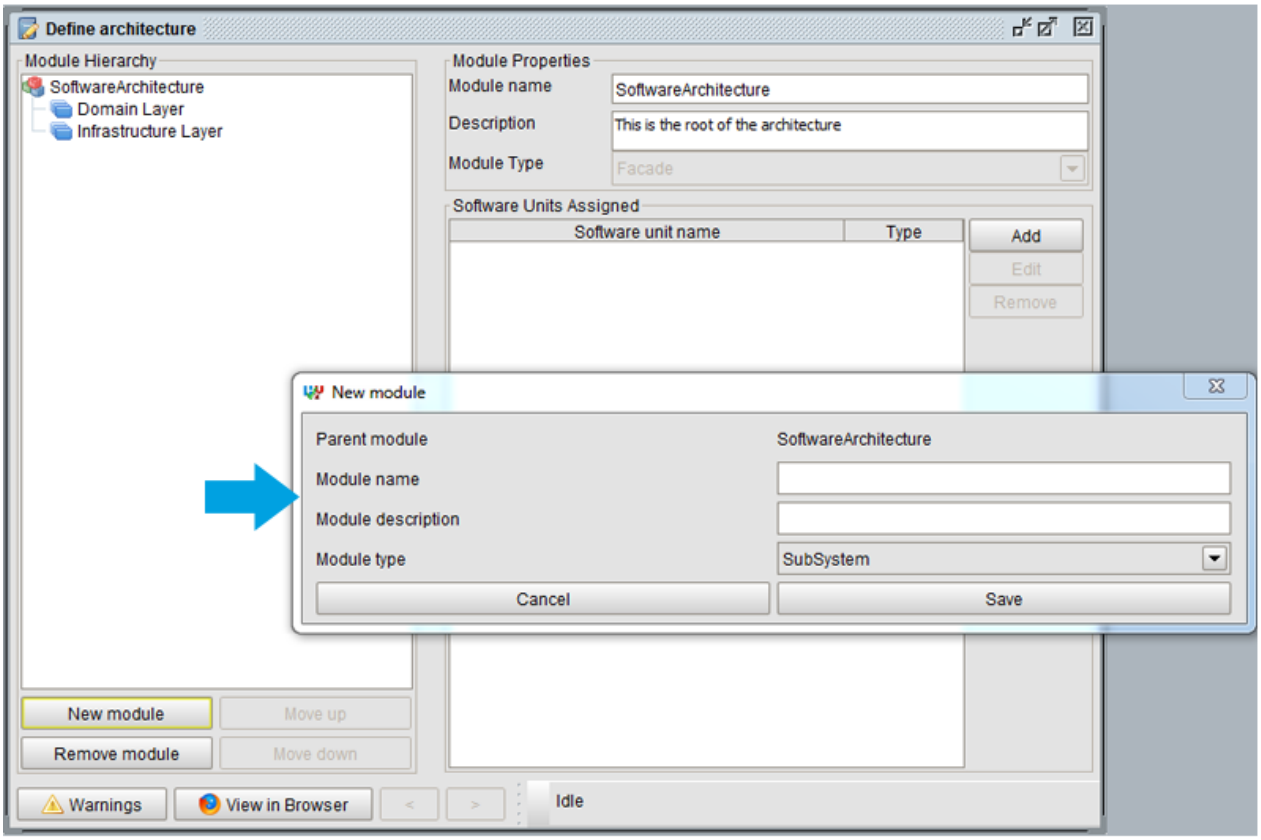
2.1 ADD MODULE

To add a module to the logical architecture you will need to do the following steps:

1. Select nothing or select the root node if you want to create a module to the root of the architecture. Otherwise select the module you wish to create a child module for.
2. Click on the "New Module" button.(*Figure 2*)
3. Enter a module name.(*Figure 3*)
4. Optional: Enter a description.



FIGUR



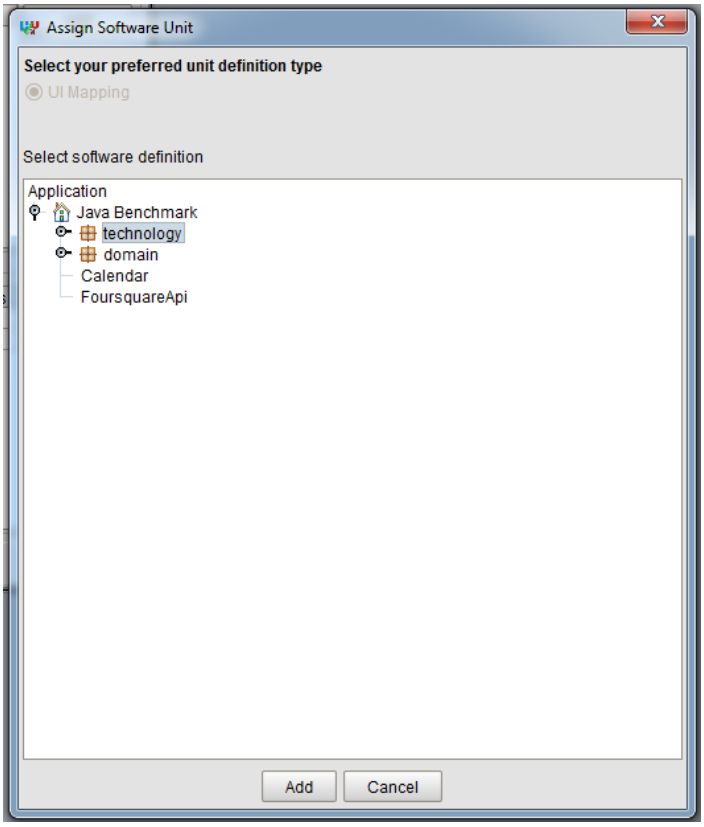
FIGUR

5. Select a module type.
6. Click on the “Save” button.
7. You can now see that a new module has been added in module hierarchy.

2.2 MAP SOFTWAREUNITS

To map a software unit to a module you will need to do the following steps:

1. Select the module you wish to map.(*Figure 4*)
2. Click on the “Add” button in the *Software Units Assigned* section.
3. Select the software unit you wish to map to the selected module.
   You can hold the “Ctrl” button to select multiple software units or drag the selected units to their layer.
4. Click on the “Add” button.
5. You can now see that the selected software units are now mapped to the selected module.

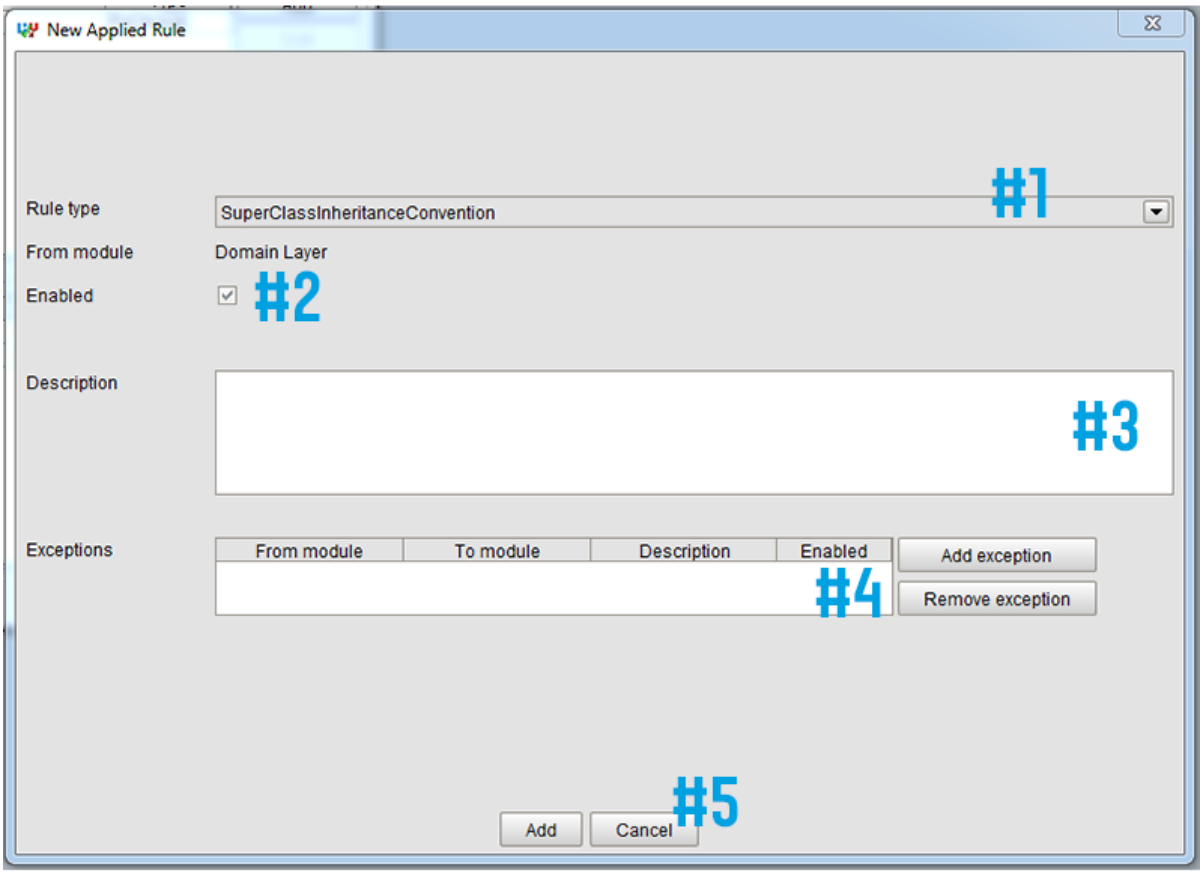


FIGUR

2.3 DEFINE RULES

To add a rule to a module you will need to do the following steps:

1. Select the module you wish to create a rule for.
2. Click on the "Add" button in the *Rules* section.
3. Select the rule type you wish to create.(*Figure 5, #1*)
4. Optional: Select to which this rule will apply.
5. Enable or Disable this rule when it's created.(*Figure 5, #2*)
6. Optional: Enter a description for this rule. (*Figure 5, #3*)
7. Optional: Add exception rules. See chapter: Advanced Tasks, Adding Exceptions(*Figure 5*, #4)
8. Fill in any details required by the rule type.
    a. Naming convention rule: enter a regex; use "Configure filter" to specify class or package level.
    b. Visibility convention rule: use "Configure filter" to specify the visibility level.
9. Click on the "Add" button to add this rule with all its exception rules. (*Figure 5,* #5)


New Applied Rule
Rule type
SuperClassInheritanceConvention
#1
From module
Domain Layer
Enabled
#2
Description
#3
Exceptions
From module
To module
Description
Enabled
Add exception
Remove exception
#4
#5
Add
Cancel

FIGUR

3 ADVANCED TASKS

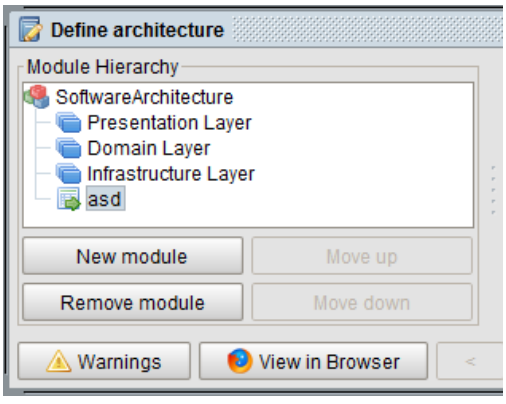
3.1 ADDING EXCEPTIONS

To add an exception rule to a rule please follow the following steps:

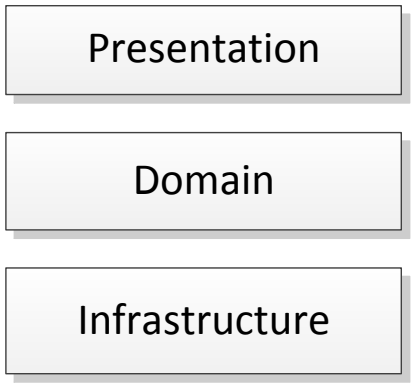
1. Open the rule details panel. You can do this by adding a new rule, or by selecting an existing rule and pressing the “Edit” button in the *Rules* section.
2. Press the “Add exception” button.
3. Select the rule type you wish to create. Although most rules have only one exception rule available.
4. Select from where this rule will apply.
5. Select to where this rule will apply.
6. Enable or Disable this exception rule when it’s created.
7. Optional: Enter a description for this rule.
8. Optional: Configure the Violation Types.
9. Fill in any details required by the rule type. For example, the naming convention rule requires you to enter a regex.
10. Click on the “Add” button to add this exception rule to the main rules.

3.2 MOVING LAYERS

The order of the layers in the Module Hierarchy represents a layer model. The higher the layer is in the Module Hierarchy, the higher it is in the layer model.
Please note that this only matters with layers, the order of the rest of the module types do not matter.
The module hierarchy in Figure 6 this is equivalent to the layer module in Figure 7.



FIGUR



FIGUR

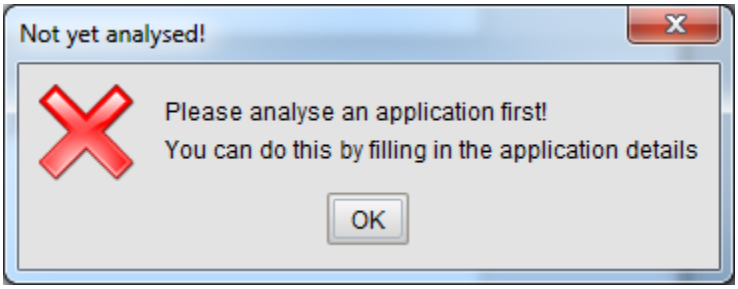
To move a layer up or down you will need to:

1. Select a layer.
2. Click on the “Move up” or “Move down” button in the *Module Hierarchy* section.

4 TROUBLESHOOTING

4.1 CANNOT MAP SOFTWARE UNITS TO A MODULE

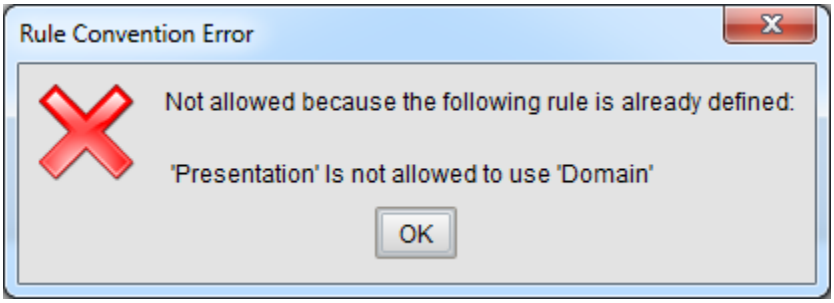
When you have yet to analyze the application or when you're just making a logical architecture it is not possible to map a software unit to a module. You will need to analyze an application first, if you haven't done so far, you will see the following error message when you click on the "Add" button in the "Software Units Assigned" section.



FIGUR

4.2 CONFLICTING RULES

It is not possible to define conflicting rules. For instance, if you were to create a rule that would state that the "Presentation" module is not allowed to use the "Domain" module. It would be impossible to define a rule that would state that the "Presentation" module must use the "Domain" module. If you would try to, this will result in the following error message.



FIGUR

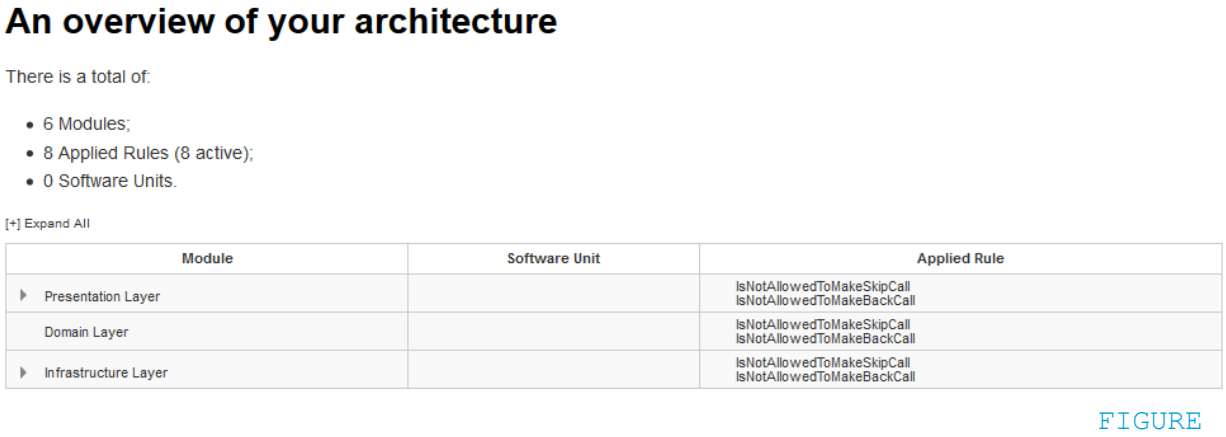
Here is a list of all rules that cannot be defined if a rule of a certain type is already in place.

USE CASE	FORBIDDEN WHEN FOLLOWING RULE IS DEFINED
Naming convention	• "Naming convention" rule in the same module
Visibility convention	• "Visibility convention" rule in the same module
Subclass convention	• "Subclass convention": rule in the same module • Same checks as a "must use" rule
Interface convention	• "Interface convention" rule in the same module • Same checks as a "must use" rule
Is not allowed to use	• "Is only allowed to use", "is only module allowed to use", "Is allowed to use" or "must use" rule from the selected module to the selected "module to"
Is only allowed to use	• "Is not allowed to use" rule from this module to the selected "module to" • "Is only allowed to use", "is only module allowed to use", "is

	allowed to use" or "must use" rule from this module to other then the selected "module to" • "Is only module allowed to use" rule from other then the selected module to the selected "module to"
Is only module allowed to use	• "Is not allowed to use" rule from this module to the selected "module to" • "Is only module allowed to use", "is only module allowed to use", "is allowed to use" or "must use" rule from other then the selected module to the selected "module to"
Is allowed to use	• "Is not allowed to use" rule from this module to the selected "module to" • "Is only allowed to use" rule from this module to other then the selected "module to" • "Is only module allowed to use" rule from other then the selected module to the selected "module to"
Must use	• "Is not allowed to use" rule from this module to the selected "module to" • "Is only allowed to use" rule from this module to other then the selected "module to" • "Is only module allowed to use" rule from other then the selected module to the selected "module to"
Skip call	• Same checks as a "is not allowed to use" rule for the 2nd layer below the selected layer, and each layer below this 2nd layer. You can see this layer as the selected "module to" layer.
Back call	• Same checks as a "is not allowed to use" rule for each layer above the selected layer. You can see this layer as the selected "module to" layer.

4.3 VIEW IN BROWSER

When you click on the 'View in Browser'-button on the main screen of the define component, a report will be generated. This report consists out of an overview of modules, applied rules and software units and of a table with all the modules with their software units and applied rules. See figure 10 and 11 for an example.



An overview of your architecture

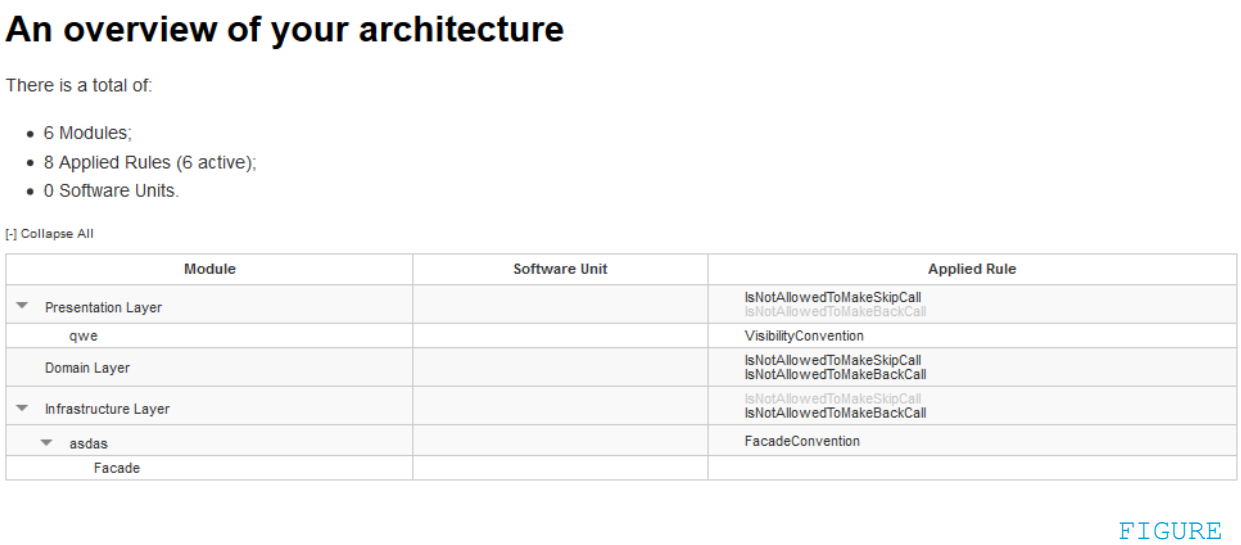
There is a total of:

- 6 Modules;
- 8 Applied Rules (8 active);
- 0 Software Units.

[+] Expand All

Module	Software Unit	Applied Rule
▸ Presentation Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
Domain Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
▸ Infrastructure Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall

FIGURE



An overview of your architecture

There is a total of:

- 6 Modules;
- 8 Applied Rules (6 active);
- 0 Software Units.

[-] Collapse All

Module	Software Unit	Applied Rule
▾ Presentation Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
qwe		VisibilityConvention
Domain Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
▾ Infrastructure Layer		IsNotAllowedToMakeSkipCall IsNotAllowedToMakeBackCall
▾ asdas		FacadeConvention
Facade		

FIGURE

5 GLOSSARY

SOFTWARE ARCHITECTURE

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both. The term also refers to documentation of a system's software architecture.

MODULE

A module is a generic term for a part of an application. Software is composed of separate, interchangeable components called modules by breaking down program functions into modules, each of which accomplishes one function and contains everything necessary to accomplish this. This can be logical or physical.

MODULE TYPE

The term module type is used to specify what kind of module is referred to.

RULE

A rule lays constraints on modules which the modules will need to follow.

RULE TYPE

A rule type is a term used to specify what kind of rule is referred to.

EXCEPTION RULE

An exception rule is an extension of a rule that allows for immunity of the defined rule. This means the rule will ignore all parts where an exception rule is defined on.

SOFTWARE UNIT

A software unit is a physical part of the application. An example of this is: a package, class, enumeration or interface.

VIOLATION TYPES

A violation type is the type of the violation that can be checked on during the validation process.