CxQL - Best Coding Practices

Table Of Contents

1. Abstract

As we stand right now, Checkmarx is a market leader in our sector.
We have many customers and all of them read, use and modify our queries on a daily basis, it is part of their work.
Therefore, it is our responsibility to provide them the best experience possible when using our queries.

Readability, **usability** and above all **performance**.

Setting some base rules for how to write queries will help us achieve a quality standard that will only improve the clients experience with our offered solutions and leave us with a good lasting impression.
More than that, it will help Checkmarx developers when it comes to the **maintainability** of existing queries taking us less time to understand the query's logic and purpose.

Something that happens with all coding standards set by companies onto developers is that, not everyone will agree with all the rules.
It is normal and expected to happen because we all have our ideas and our opinions on what is more or less readable or understandable.
The key point here is compromise. We need to understand that as we grow as a company, also grows our responsibilities to the thousands of developers that use our software.
So, the coding standards of the company should be a reflection of all the developers working as Checkmarx and not the reflection of a single developer.

It is now our objective to follow this rules on all new queries and queries that will end up fixed in the future.
If you are working on an existing query that doesn't follow one or more of this rules, fixed them as part of your work.

As a final note, none of the rules mentioned in this document are final or permanent.
If you believe some rule does not represent the company in the best way, you are more then welcome to propose a different solution.
Any new proposed rule or proposed modification to an existing rule will be discussed with the developers, team leaders and managers responsible for this coding standards to validate their viability.

2. Variables

Whenever you declare a variable, you should keep in mind the following rules:

2.1 Explicit Types

Always use the **explicit type** of the variable. Using **"var"** only hides visual information, specially because most of the types you see in our queries are exclusive to Checkmarx (**CxList**, **TypeRef**, **MethodDecl**, etc).
Please try to avoid the use of **"var"** in any query.
If, for some reason you truly believe that some special case requires the use of **"var"**, you can use it upon validation from your code reviewer.

Example
```
CxList dbIn = Find_DB_In();
```

2.2 Lower Camel Case Naming

The name of the variable should always be lower camel case. Some exceptions may apply when dealing with industry know acronyms like "HTML" or "XSRF" for example.
You may use a different approach in such exceptions upon validation from your code reviewer.

Example
```
CxList setRequestHeader =
Find_Methods().FindByShortName("setRequestHeader");
```

2.3 Meaningful and Logic Naming

Name the variable in a meaningful and logical way. The name should explicitly say exactly what is stored in the variable without causing confusion or doubt.
There are no exceptions for this rule.

Example
```
CxList scriptTag = Find_MethodDecls().FindByShortName("script__*");
```

3. Operators

3.1 Operators "+" and "+="

Whenever you work with CxList, avoid using the operator "+" or "+=" to add new elements to the list.
Use the method "Add" instead because it will improve the performance of the operation.

The problem comes from the fact that a simple "a + b" will be transformed into "a = a + b" resulting in the creation a intermediate object during the operation.
This results in more memory consumption and more processing power used.

Wrong
```
result = NumberCast + numberSanitizer + booleanConditions;
```
Correct
```
result = NumberCast.Clone();
result.Add(numberSanitizer);
result.Add(booleanConditions);
```

3.2 Operators "-" and "-="

Using the operators "-" and "-=" will have the same performance issues has in the operators "+" and "+=".
However, in this case, we don't have any method that replaces this operations.

The only thing we recomend when using this operators is to mitigate the situation by reducing the amount of times you use the operator.
Try to agregate all elements that you want to subtract in a new list and perform a single subtraction in the end.

Wrong
```
result = NumberCast - numberSanitizer - booleanConditions;
```
Correct
```
CxList sanitizers = NumberCast;
sanitizers.Add(numberSanitizer);
sanitizers.Add(booleanConditions);
result -= sanitizers;
```

4. General rules

4.1 Keyword "return"

The **"return"** keyword should be used for debug purposes.
Never leave a return in a production query.

4.2 Use of "All"

The CxList **"All"** gives you access to all nodes of the current language. For most situations this list is to big to be used without afecting the overall performance of the query.
Avoid using this CxList by filtering and reducing this list to a subset of itself.
Using general queries like **Find_Methods()** or **Find_UnknownReferences()** is an easy way to avoid using **"All"**.

4.3 Use of Atomic Queries with Flow

Whenever you use Atomic Queries that rely on flows, please reduce the source and target CxList's as much as possible.
This is crucial for performance reasons, speacially when using loops.

4.4 Variable Assignments and their References

Keep in mind that, assigning a CxList to another CxList will only share their memery adress.
This means any modification you do to one of this lists, will also afect the other.
Use the method "Clone" when you assign the list to avoid this issue.

Wrong
```
CxList methods = Find_Methods();
CxList unknownReferences = Find_Unknown_References();
CxList methodsAndUnknownRefs = methods;
methodsAndUnknownRefs.Add(unknownReferences);
```
Correct
```
CxList methods = Find_Methods();
CxList unknownReferences = Find_Unknown_References();

CxList methodsAndUnknownRefs = methods.Clone();
methodsAndUnknownRefs.Add(unknownReferences);
```
Correct
```
CxList methods = Find_Methods();
CxList unknownReferences = Find_Unknown_References();

CxList methodsAndUnknownRefs = All.NewCxList();
methodsAndUnknownRefs.Add(methods);
methodsAndUnknownRefs.Add(unknownReferences);
```

4.5 Long Query Lines

Break query lines so that you can read them without horizontal scrolling.

4.6 Use of "result =" and "result.Add()"

Use "result =" or "result.Add()" as less as possible.
Also, try to do it only at the end of the query.
It will be easier for other developers to understand what is being returned as result making the query more readable and maintainable.

5. Comments

5.1 General and Information Queries

Most of our Executable queries have descriptions associated. These descriptions are created by our Application Security Team and they detail all the relevant information in order to understand the purpose of the query.
This, however, doesn't happen in General and Information queries.

Therefor, whenever you work on any of this two types of queries, keep a comment with detailed description at the top of the query.
This comment should tell, to however reads the query, what it's purpose is and what is expected to find.

The comment block should look like this:

Example

```
/*
This query checks if any of the values in an input CxList appears in
any type of loop.
We do 2 different checks - one for typeof(IterationStmt) and one for
typeof(ForEachStmt)

Parameters:
 param[0] - the input CxList, which we check if it contains members
that are in a loop
 param[1] - a relevant set of objects in which we test this query
 param[2] - a flag, ... have to check :)
*/
```

5.1 Logic and Purpose of the Algortihm

Writing a query without a single comment, unless it is a very simple query, is not a good practice. It will take more time for someone working with it to understand what you were trying to do and debug looking for issues.

The inverse is, also, not a good practice. You don't need to specify the purpose of every single line of code in a query.
Try to find a good balance with the purpose of leaving a more maintainable code.

This is a good example of the use of comments to help understand the logic of the query and it's steps:

Example

```
// Find apex code
CxList apex = Find_Apex_Files();

// Calculate all methods calls that contain a member of the
listToCheck, up to 5 level deep
CxList methods =
apex.FindAllReferences(listToCheck.GetAncOfType(typeof(MethodDecl)));
int numMeth = 0;
for(int i = 0; i < 5 && methods.Count > numMeth; i++)
{
 numMeth = methods.Count;
methods.Add(apex.FindAllReferences(methods.GetAncOfType(typeof(MethodDe
cl))));
}

// Add these methods to the list to check, because these methods might
also be called in a loop
listToCheck.Add(methods);

// Leave only results in the defined subset
listToCheck *= subset;

/// Part 1 - IterationStmt
CxList iterations = listToCheck.GetAncOfType(typeof(IterationStmt));
CxList listToCheckIter = listToCheck.GetByAncs(iterations);
CxList somethingInIteration = All.NewCxList();
foreach (CxList iteration in iterations)
{
 try
 {
  IterationStmt iter = iteration.data.GetByIndex(0) as IterationStmt;
  if (iter != null)
  {
   CxList memberInLoop = listToCheckIter.GetByAncs(iteration);

   if (memberInLoop.Count > 0)
   {

somethingInIteration.Add(iteration.Concatenate(Connect_Loop_To_DB(membe
rInLoop, bareListToCheck, checkForNull), true));
   }
  }
 }
 catch (Exception ex)
 {
  cxLog.WriteDebugMessage(ex);
 }
}
```

6. Loops

Atomic queries are complex enough without being called recursively. Try to search for different solutions that don't require the use of a loop.
There will be situations where this is unavoidable, please confirm wih your code reviewer that this is indeed the only solution and get is approvel to do so.

6.1 Use of "All" Inside Loops

The use of the CxList **"All"** should always be avoided and this is especially important when you need to use a loop.
Never use **"All"** inside a **"for"**, **"foreach"** or **"while"** loops.
Just imagine that, for every iteration of the loop, you will access ALL DOM nodes detected for the given language.

The performance of this operation is terrible and there are plenty of general queries that can provide you filtered subsets of this list.
Otherwise, just filter the list manualy to a new smaller list.

6.2 Use of Nested Loops

If loops should be avoided due to performance reasons, you can imagine the kind of degradation a nested loop can cause in the execution time of a query.
Although, the same rule applies to nested loops as in single loops. If this is the only way you can achieve the desired result, please approve the algorithm with you code reviewer.
Try to minimize the CxList's and atomic queries used inside this loops to mitigate the performance costs.

7. Atomic Queries

In this section you can find a set of good practices for some of the most used atomic queries.
Following this guidelines can not only improve performance, readability and maintainability but also prevent unexpected behaviours in some cases.

7.1 CxList.FindByName()

Whenever you are using this atomic query, keep in mind that it uses the full namespace, not the member type.
Also, for most cases, what you need is **CxList.FindByShortName()**.

7.2 CxList.FindByShortNames()

Use **CxList.FindByShortNames()** instead of multiple uses of **CxList.FindByShortName()**.

Wrong
```
CxList methods = Find_Methods;

CxList dbIn = methods.FindByShortName("executeQuery");
dbIn.Add(methods.FindByShortName("executeUpdate"));

result = dbIn;
```
Correct
```
CxList methods = Find_Methods;

CxList dbIn = methods.FindByShortNames(new List<string> {
 "executeQuery",
 "executeUpdate"
});

result = dbIn;
```
Correct
```
CxList methods = Find_Methods;

List<string> dbInMethodNames = new List<string> { "executeQuery",
"executeUpdate" };
CxList dbIn = methods.FindByShortNames(dbInMethodNames});
```

```
result = dbIn;
```

7.3 CxList.FindByTypes()

Use **CxList.FindByTypes()** instead of multiple uses of **CxList.FindByType()**.

Wrong
```
CxList numericTypes = All.FindByType("integer");
numericTypes.Add(All.FindByType("long"));
numericTypes.Add(All.FindByType("double"));
numericTypes.Add(All.FindByType("decimal"));
numericTypes.Add(All.FindByType("datetime"));
numericTypes.Add(All.FindByType("date"));
numericTypes.Add(All.FindByType("id"));

result = numericTypes;
```

Correct
```
string[] types = {"integer", "long", "double", "decimal", "datetime",
"date", "id"};
CxList numericTypes = All.FindByTypes(types);

result = numericTypes;
```

7.3 cxLog()

The method **cxLog.WriteDebugMessage()** will log any message to the log window and log file.
This method should only be used for debug purposes and inside a **"Try Catch"** statement.
Don't leave **cxLog.WriteDebugMessage()** methods in any query unless you are using a **"Try Catch"**.
Also, don't use **cxLog.WriteDebugMessage(e)**. This will throw a sandbox exception trying to print the stack trace.
Use **cxLog.WriteDebugMessage(e.Message)** instead.
If you are working on a special case and think that you should leave a debug message, please confirm with your code reviewer and get his\her approval to do so.

Wrong
```
try
{
 //Code
}
catch (Exception e)
{
 cxLog.WriteDebugMessage(e);
}
```
Correct
```
try
{
 //Code
}
catch (Exception e)
{
 cxLog.WriteDebugMessage(e.Message);
}
```

8. Removing\Deleting Queries

Removing\Deleting queries should be done only after talking to PM.

There are 2 main rules that you should follow whenever you need to remove a query:

8.1 Work with PM's on opening the necessary PBI's

You should open a new PBI ahead at least 2 versions.
The relevant PM should be involved in this process.

Besides the PBI to delete the query, there should also be a PBI for the Installer team to check the existence of client deprecated queries.

8.2 Removing the query

All code must be removed from the query. Don't leave it commented out, completely delete it.
Also, remove this query from DB and from all presets which it might be included.

8.3 Add a comment and debug message

A comment explaining the reason for the query to be deleted must be added as follows:

For deprecated queries:

Deprecated
```
//This query is deprecated.
cxLog.WriteDebugMessage(“The query NAME_OF_NEW_QUERY is deprecated”);
```

For replaced queries:

Replaced
```
//This query is deprecated. Please consider NAME_OF_NEW_QUERY instead.
cxLog.WriteDebugMessage(“The query NAME_OF_NEW_QUERY is deprecated”);
```

8.4 Add the query to the excluded queries

Go to the following path on your CxEngine folder:

"..\CxEngine\Tools\ExcludeQueries"

Now open the file "Query_Preset_exclude.xml" and add a new entry on each preset with the ID and name of the query to be excluded.
This will make sure that the "ValidateQueries" tool will not give any errors regarding the excluded query.

For more information please see: