Figure 1

Figure 1 depicts the Impacted Classes technique. First, the tool identifies modified assets using syntactic analysis. For each modified asset, the tool computes its dependences using **data-flow analysis**, that is, the set of other assets needed to compile the modified asset. We call this set of modified assets with their dependences as sub products. The approach compiles the source and target versions of each sub product. It then checks, for each sub product, whether they have compatible observable behavior, generating test only for modified classes.

What’s the goal?

Use figure 1 to explain how Impacted Class approach check SPL evolution.

What this figure stands for?

This figure illustrates an SPL refactoring, where we have an evolution in the class *TestSuite.* During the class maintenance, the developer alters the *isEquals* method behavior.

This figure shows a refactoring in a method responsible for comparing two objects.As you can see, before the evolution we have an if-else branch that firstly performs an *exclusive* *or*, which is evaluated to true, if one, and only one, of the input objects is equals to null. Following the method logic, the else branch means that we have two possibilities: either both objects are null or both objects are not null. Therefore,

in the else branch, it is only needed to check if the first object is different to null, which if evaluated as true, implies that the second object is also not null and the comparison can be made. Following the same reasoning, if the first object is null, even the second object being concrete the comparison cannot be executed since it would throw a NullPointerException.

On the evolved method, the developer removes the *xor* verification and only checks if the second parameter is equals to null, since the first object is already checked on the upper method *compare.*

if we perform a narrow minded analysis, too focused on the impacted method, we will see that this evolution do not preserved the method behavior, since we can create a test which passes a first object being null to the *isEquals* method and the second one a concrete method. This way , apparently, the comparison can be made, however when this is executed the method throws a NullPointerException because the first method is null.

That’s exactly, what this technique does, it ignore the whole context where the change has been applied and hastily consider the SPL evolution as a Non-Refinement.

Note that the Impacted Class approach approximates the evolution checking, since it only analyses the impacted class during the SPL development.

In this case it only checks the TestSuite class.