In Proceedings of the 8th International Conference on Software Technology -ICST'97- Curitiba, Brazil, June 1997.

Identifying and Measuring Coupling on Modular Systems

Hakim Lounis and Walcelio Melo
CRIM
1801, McGill College Avenue, suite 800
Montréal, Canada H3A2N4
hlounis, wmelo@crim.ca

Abstract: Low module coupling is considered to be a desirable quality for modular programs to have. This paper proposes a comprehensive suite of measures to quantify the level of coupling in a modular system. This suite takes into account the different coupling mechanisms provided by the C language but it can be tailored to other languages. The different measures in our suite thus reflect different hypotheses about the different mechanisms of coupling in modular systems, and we have precisely defined the measures of coupling so that they can be determined algorithmically.

Keywords: Coupling, Modular systems, C language.

1. Introduction

Modularity has been considered an important software product quality criterion from an engineering point of view. For instance, in [SO88], modularity has been cited as a criterion which can impact several software quality factors, e.g. efficiency, flexibility, interoperability, maintainability, reusability, and verifiability. A software product is considered modular if its components exhibit a high cohesion and its components are weakly coupled [CY79]. A module has high cohesion if all of its elements are related strongly. Such elements, like statements, procedures or declarations cooperate to achieve a common goal which is the function of the module. On the other hand, coupling characterises a module's relationship to other modules of the system. It measures the interdependence of two modules (e.g. module M calls a procedure provided by module N or accesses a variable declared by module N). Figure 1 provides an illustration of two software systems with different module's coupling levels.

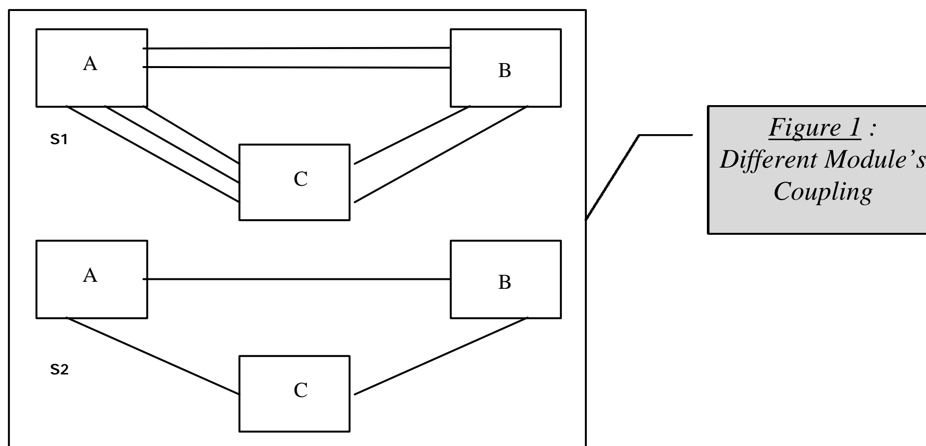


Figure 1 : Different Module's Coupling

It seems obvious that we would like that a module exhibits low module's coupling. This believe stems from the fact that we suppose that a module will be easier to understand, modify, test or reuse if the module is weakly coupled with other modules. In addition, we believe that an error in a module will propagate less into other modules of a system if the module is low coupled with the others modules of the system. Moreover, a low coupling module has a strong change to be less error-prone that a higher coupled module.

Given the importance of coupling on software quality, it would seem reasonable to measure the cohesion and coupling of a software system. By doing that, we may able to understand better the relationship between modularity (a software engineering design and implementation criterion) and software quality factors. Once we can measure the level of coupling and cohesion of a software system, we will be able to better characterise its quality, assess it with regard to other systems, and predict its product quality, e.g. maintenance effort's costs and error-proneness.

The goal of this work is two folds. Firstly, we are concerned on identifying the different forms coupling can appear in a modular software system. As point out in [PJ80], there are many different kinds of coupling. Each kind of coupling may have different impacts on software quality. Secondly, we are engaged in measuring the different kinds of coupling and evaluating its impact on error-proneness (a software quality attribute).

This paper is organised as follows. The next section describes related works. In section 3 we introduce some basic definitions, then define the context in which our study takes place, and finally identify the different levels of interconnection between modules. For each level of interconnection, we list and define a set of interesting modules' interconnection cases, and we propose simple and automated measures to quantify such cases for module's units, modules, and software systems. Finally, conclusions and directions for future research are outlined.

2. Related Works

Chidamber and Kemerer [CK94] have proposed a suite of OO design metrics, called MOOSE metrics, which have been validated in [BBM96]. They provide a very simple coupling measure, called CBO. A class is coupled to another one if it uses its member functions and/or instance variables. CBO provides the number of classes to which a given class is coupled. Similarly to MOOSE, MOOD [AC94] includes a coupling measure, called Coupling Factor. In MOOD, a class, A, is coupled with another one, B, if A sends a message to B. Both MOOSE and MOOD coupling measures are very simple and only take into account message exchange among classes.
Recently, [BDM97] have defined a suite of coupling metrics for the design of OO systems. In this work a suite of 24 kinds of OO design coupling measures have been defined. These coupling measures take into account different kinds of coupling which can exist in an OO oriented design.
Regarding code coupling, in [PJ80] it has been proposed eight different levels of coupling. For each coupling level, the shared data (parameters, global variables, etc.) are classified by the way they are used. In a more recent work, [OHK93] extended the eight levels of coupling to twelve, offering a more detailed measure of coupling. The coupling levels are defined between pairs of units, say P and Q. For each coupling level, the call/return parameters are classified by the way

they are used. These uses are classified into computation uses (C-uses), predicate uses (P-uses), and indirect uses (I-uses). We will detail these three kind of uses in the next section of the paper.
Our work is expired from the work presented in [OHK93]. In addition, we have used the same measurement framework proposed in [BDM97]. In fact, our work is complementary to the one described in [BDM97] in which OO design coupling measure have been defined. Here, we are mainly concerned with coding coupling measure.

3. Modules' Coupling

Before using notions of software system, module, and modular system , let us introduce them. We adopt in the following, basic definitions proposed by [BMB96].

3.1. Basic definitions

- *System :* a system S is represented as a pair <E, R>, where E represents the set of elements of S and R is a binary relation on E ($R \subseteq ExE$) representing the relationships between S's elements.

Ex : E represents the set of code statements and declarations and R the set of control flows from one statement to another.

- *Module :* a module M of S is the pair $<E_M, R_M>$, where E_M is a sub-set of E and R_M is a sub-set of E_M x E_M and of R.

Ex : a module M could represent a code segment, a procedure, or a set of such procedures packaged in a same file.

M's elements are connected to others system's elements by incoming InputR(M) and outcoming OutputR(M) relations:

$InputR(M)=\{<e_1, e_2> \in R \mid e_2 \in E_M \text{ and } e_1 \in E-E_M\}$= the set of relationships from elements outside M to those inside M.
$OutputR(M)=\{ <e_1, e_2> \in R \mid e_1 \in E_M \text{ and } e_2 \in E-E_M \}$ = the set of relationships from elements inside M to those outside M.

- *Modular system : a modular system is the 3-tuple* MS=<E, R, MC> if S is a system and MC a collection of S' modules.

$$\forall e \in E\ (\exists M \in MC\ (M=<E_M, R_M> \text{ et } e \in E_M)) \text{ et},$$
$$\forall M_i=<E_{Mi}, R_{Mi}> \in MC \text{ et } \forall M_j=<E_{Mj}, R_{Mj}> \in MC, E_{Mi} \cap E_{Mj}=\emptyset$$

Ex : figure 2 shows the type of modular system we will consider in the paper.

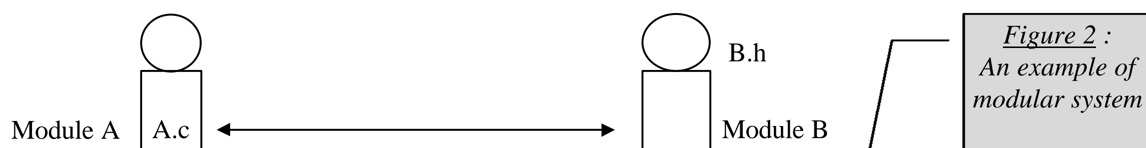


Figure 2 : An example of modular system

In this context, coupling quantify the strength of interconnection between modules of a same modular system. During the software maintenance process, coupling predicts the difficulty of changing module's programs and what are the implications for programs in other modules. [BMB96] states that a coupling measure must have some properties, for example, to be nonnegative, and null when there are no relationships among modules. An other important expected property is that merging modules can only decrease coupling, so that it encourages us to merge in a new sole module, highly coupled modules.

Before presenting in detail the set of identified modules' interconnection levels, we precise the object study.

3.2. Problem definition

The topic of this subsection is to define precisely what we call module in our study of modules' interconnection. We consider a *module* as a collection of units, collected in a file and its associate header. A program *unit* is one or more contiguous program statements having a name by which other parts of the system can invoke it (e.g. procedure, ...). We consider that all modules are written in the C programming language. Figure 3 illustrates that:

Ex :

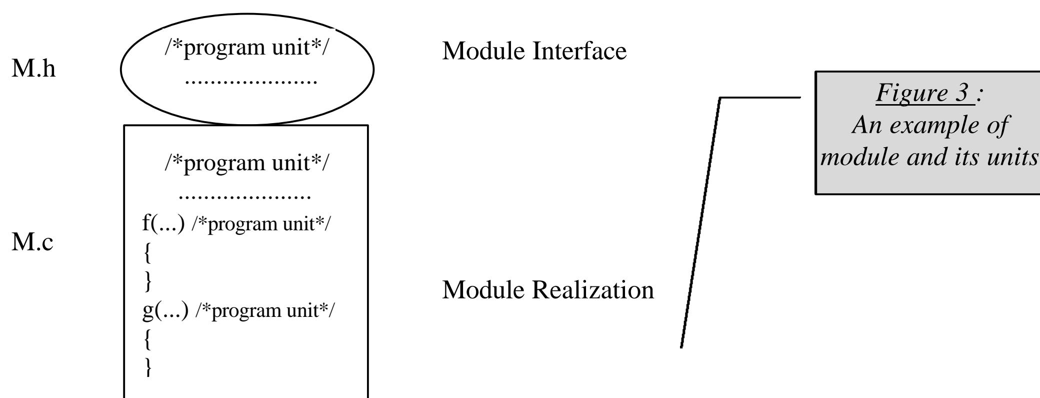


Figure 3 : An example of module and its units

A good software system should exhibit low coupling between units in different modules. Coupling increases the interconnections between the two units (so the two modules) and increases the

probability that a fault in one unit may affect others connected units. We consider also that coupling may lower the understandability and maintainability of a software system.
In our context, we are interested by identify possible interconnections between two units belonging to two different modules. We have to define different interconnection levels between two units m and n of two modules M and N. The architecture of such a system is illustrating by figure 2.

3.3. Identified levels of module's coupling

We distinguish between three kind of modules interconnections. The following figure shows that :

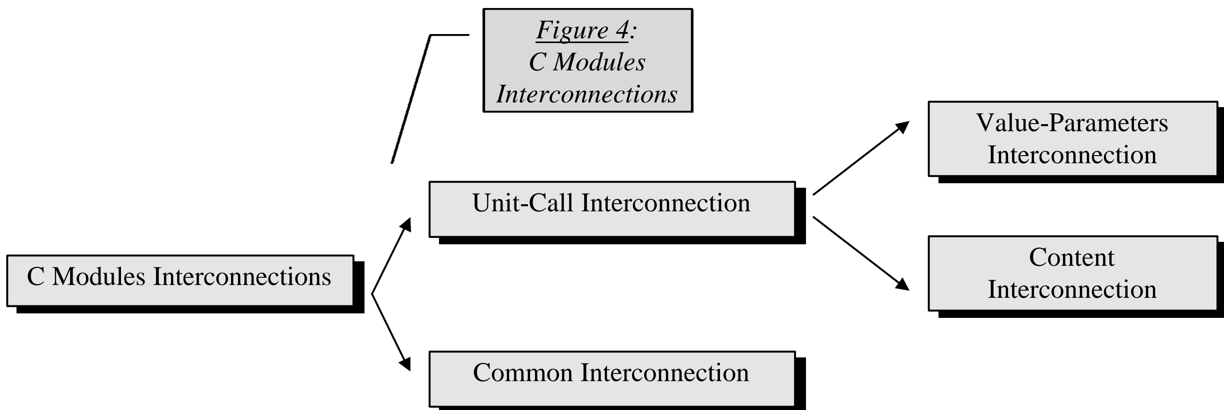


Figure 4: C Modules Interconnections

If the modules are to be used together in a useful way, there may be some external references, in which the code of one module refers to a location in another module. This reference may be to a data location defined in one module and used in another, as in *common interconnection*, or it may be to the entry point of a procedure (we said the *callee*) that appears in the code of one module and is called from another module (we said the *caller*). It is the case of *unit-call interconnection.*

The distinction between different kind of modules interconnection is done thanks to three criteria :

- The kind of information shared by interconnected modules (parameters or global areas).
- To which type belong the shared information (scalar, structure, ...)
- What use is done with this shared information.

In the context of the latter criterion, [OHK93] classified uses into computational uses *C-use*, predicate uses *P-use*, and indirect uses *I-use*. A C-use happens when a variable is used on the right side of an assignment statement or in an output statement.

C-use of x

```
{
...
c_use_of(x) ;
...
}
```

printf("%d\n", x)

Ex :

t=a*x*x-b*x ;

with c_use_of(x) → assignment_right_of(x) | output_statement_of(x)
assignment_right_of(x) → lexeme=expr_of(x)
expr_of(x) → an expression where x occurs
output_statement_of(x) → printf("...", x, ...) | ...

A P-use occurs when a variable is used in a predicate statement.

P-use of x

```
{
... ;
pred_stat_of(x) ; ... }
```

Ex :

if ((x*x-4*a*c)>0) ...

with pred_stat_of(x) → alternative_statement_of(x) | loop_statement_of(x)
alternative_statement_of(x) → if pred_of(x) ... | ...
loop_statement_of(x) → while pred(x) ... | do ... while (pred(x)) | ...
pred(x) → predicate expression where x occurs.

An I-use occurs when a variable is used in an assignment to another variable, and this latter variable is then used in a predicate statement.

I-use of x

```
{
...
assignment_right_of(x) ; ... ;
pred_stat_of(lexeme(x)) ; ... }
```

Ex :

t=a*x+b ; while (t>0) ... ;

In the following part of the paper, we list and define the identified kinds of C Modules Interconnection (CMI). All these identified CMI are disjoints, so that if $CMI_i(M)$ is defined in the following manner :

$CMI_i(M) \subseteq (InputR(M) \cup OutputR(M))$ is the set of CMI of type i in module M,

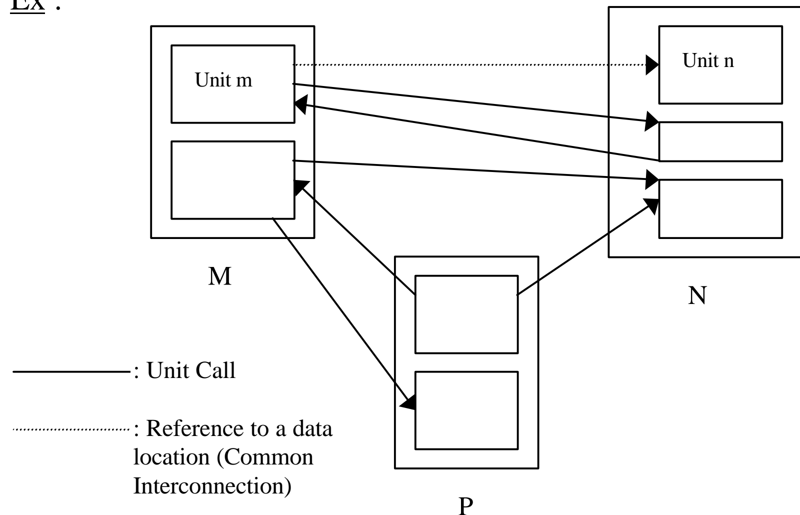
we have : $CMI_i(M) \cap CMI_j(M)= \emptyset, \forall i, j$.

NB :

We will use also two subsets of the $CMI_i(M)$ set :
$CMI_i(m) \subseteq CMI_i(M)$ is the set of CMI of type i in unit m of M
$CMI_i(m,n) \subseteq CMI_i(m) \subseteq CMI_i(M)$ is the set of CMI of type i between unit m of M and a unit n of another module of the system.

On the other hand, we will precise for each module's interconnection type, *Importing* and *Exporting* amounts relatively to the total amount of coupling. It inform us on the impact that one modules' statements has on the statements of an interconnected module.

Ex :



For units m :
$Coupling_{exp}(m)=1$; $Coupling_{imp}(m)=2$
$Coupling(m)=3$
$Coupling_{exp}(n)=1$; $Coupling_{imp}(n)=0$
$Coupling(n)=1$

For modules :
$Coupling_{exp}(M)=2$; $Coupling_{imp}(M)=4$

$Coupling(M)=6$

$Coupling(N)=5$

For the entire system :
$Coupling(System)=7$

We begin with the elementary case where there is no interconnection between studied modules.

■ *Independent Interconnection :*
It corresponds to the case where $m \in M$ does not call $n \in N$ and n does not call m. There are no common variable references or common references to external media between M and N.

3.3.1. Unit-Call Interconnection :

It corresponds to the case where m calls n or n calls m, with or without passing parameters. In the case where m calls n, m is said the *caller* and n the *callee*. We begin with the case where no parameters are transmitted from m to n.

■ *No Parameters Interconnection :*
m calls n or n calls m. No passing parameters, common variables references, or common references to external media. The number of occurrences of such a kind of interconnection called NPI, is computed for each unit module, for each module and then for the entire system. NPI is done by the following equations :

$$NPI(m) = NPI_{imp}(m) + NPI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{NPI}(m,n)\right| + \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{NPI}(n,m)\right|$$

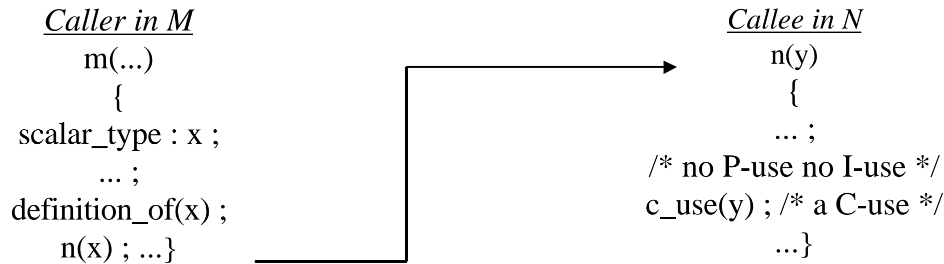
$$NPI(M) = NPI_{imp}(M) + NPI_{\exp}(M) = \sum_{m \text{ unit of } M} NPI_{imp}(m) + \sum_{m \text{ unit of } M} NPI_{\exp}(m)$$

$$NPI(S) = \sum_{M \in MC} NPI_{imp}(M) = \sum_{M \in MC} NPI_{\exp}(M), \quad \text{Since the modules are disjoint.}$$

3.3.1.1. Value-Parameters Interconnection :

The modules M and N are connected through their respective units m and n. The caller m transmit parameters to the callee n which uses them without modifying their values. The distinction between following interconnection scenarios is done thanks to two criteria : to which type belongs the transmitted information and what use is done with this information. On the other hand, we wish to compute the number of occurrences of each scenario, both for each module unit, for each module and for the whole system.

■ *Scalar-Data Interconnection :*
Some scalar variable in m is passed as an actual parameter to n and it has a C-use but no P-use or I-use.



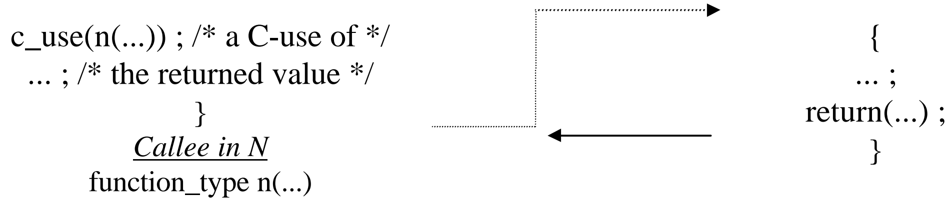
definition_of(x) → x=expression

$$ScDI(m) = ScDI_{imp}(m) + ScDI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{ScDI}(m,n)\right| + \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{ScDI}(n,m)\right|$$

$$ScDI(M) = ScDI_{imp}(M) + ScDI_{\exp}(M) = \sum_{m \text{ unit of } M} ScDI_{imp}(m) + \sum_{m \text{ unit of } M} ScDI_{\exp}(m)$$

$$ScDI(S) = \sum_{M \in MC} ScDI_{imp}(M) = \sum_{M \in MC} ScDI_{\exp}(M)$$

■ *Return-Data Interconnection :*
m and n are connected by a « return » statement. The returned value has a C-use.

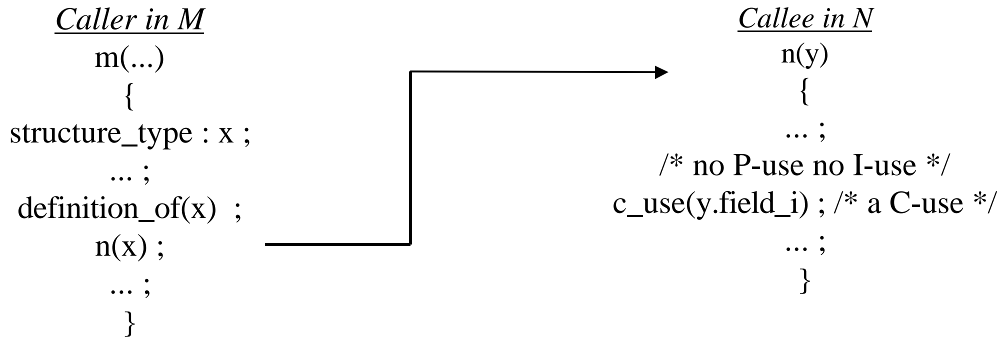


$$RDI(m) = RDI_{imp}(m) + RDI_{\exp}(m) = \sum_{n\ unit\ of\ N \in (MC-M)} \left| CMI_{RDI}(m,n) \right| + \sum_{n\ unit\ of\ N \in (MC-M)} \left| CMI_{RDI}(n,m) \right|$$

$$RDI(M) = RDI_{imp}(M) + RDI_{\exp}(M) = \sum_{m\ unit\ of\ M} RDI_{imp}(m) + \sum_{m\ unit\ of\ M} RDI_{\exp}(m)$$

$$RDI(S) = \sum_{M \in MC} RDI_{imp}(M) = \sum_{M \in MC} RDI_{\exp}(M)$$

■ *Stamp-Data Interconnection :*
A structure in m is passed as an actual parameter to n and it has a C-use but no P-use or I-use. %% write about objects %%

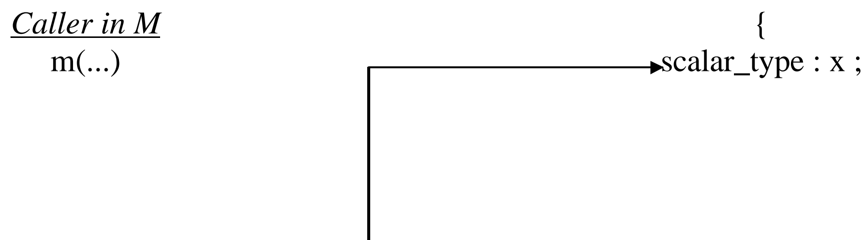


$$StDI(m) = StDI_{imp}(m) + StDI_{\exp}(m) = \sum_{n\ unit\ of\ N \in (MC-M)} \left| CMI_{StDI}(m,n) \right| + \sum_{n\ unit\ of\ N \in (MC-M)} \left| CMI_{StDI}(n,m) \right|$$

$$StDI(M) = StDI_{imp}(M) + StDI_{\exp}(M) = \sum_{m\ unit\ of\ M} StDI_{imp}(m) + \sum_{m\ unit\ of\ M} StDI_{\exp}(m)$$

$$StDI(S) = \sum_{M \in MC} StDI_{imp}(M) = \sum_{M \in MC} StDI_{\exp}(M)$$

■ *Scalar-Control Interconnection :*
Some scalar variable in m is passed as an actual parameter to n and it has a P-use.



```
... ;
definition_of(x) ;
n(x) ;
...
}
```

Callee in N

```
n(y)
{
... ;
/* no C-use no I-use */
pred_stat(y) ; /* a P-use */
...
}
```

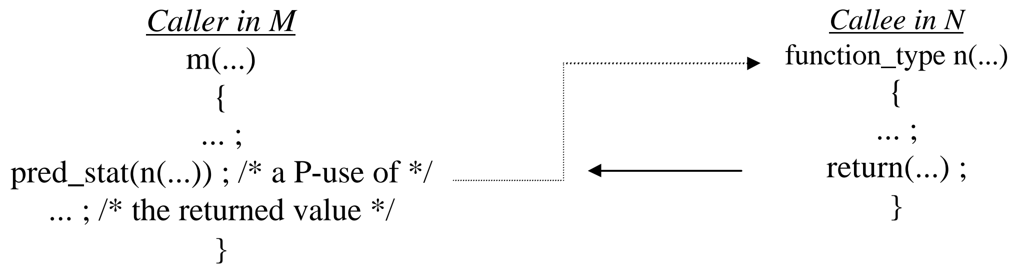
$$ScCI(m) = ScCI_{imp}(m) + ScCI_{\exp}(m) = \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{ScCI}(m,n)\right| + \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{ScCI}(n,m)\right|$$

$$ScCI(M) = ScCI_{imp}(M) + ScCI_{\exp}(M) = \sum_{m\ unit\ of\ M} ScCI_{imp}(m) + \sum_{m\ unit\ of\ M} ScCI_{\exp}(m)$$

$$ScCI(S) = \sum_{M \in MC} ScCI_{imp}(M) = \sum_{M \in MC} ScCI_{\exp}(M)$$

■ *Return-Control Interconnection :*

m and n are connected by a « return » statement. The returned value has a P-use.



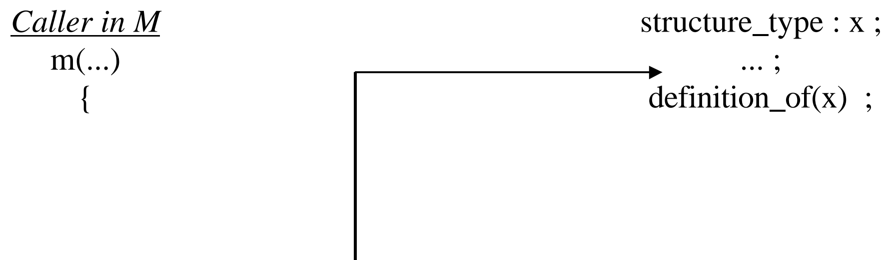
$$RCI(m) = RCI_{imp}(m) + RCI_{\exp}(m) = \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{RCI}(m,n)\right| + \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{RCI}(n,m)\right|$$

$$RCI(M) = RCI_{imp}(M) + RCI_{\exp}(M) = \sum_{m\ unit\ of\ M} RCI_{imp}(m) + \sum_{m\ unit\ of\ M} RCI_{\exp}(m)$$

$$RCI(S) = \sum_{M \in MC} RCI_{imp}(M) = \sum_{M \in MC} RCI_{\exp}(M)$$

■ *Stamp-Control Interconnection :*

A structure in m is passed as an actual parameter to n where it has a P-use.



```
n(x) ;
... ;
}
```

Callee in N

```
n(y)
{
... ;
/* no C-use no I-use */
pred_stat(y.field_i) ; /* a P-use */
... ;
}
```

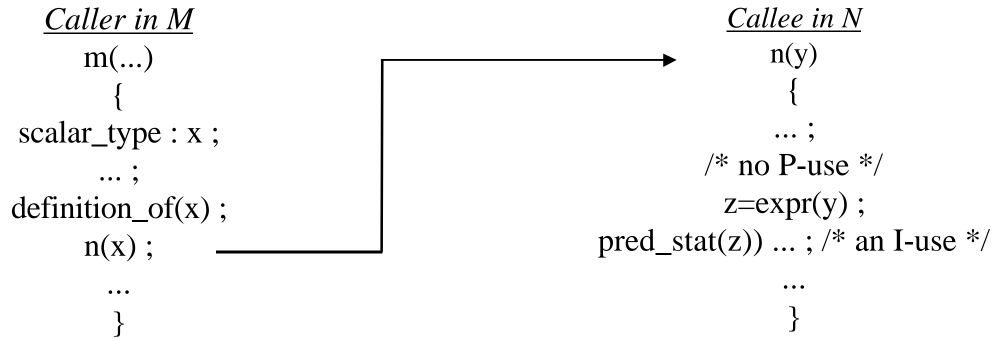
$$StCI(m) = StCI_{imp}(m) + StCI_{\exp}(m) = \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{StCI}(m,n)\right| + \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{StCI}(n,m)\right|$$

$$StCI(M) = StCI_{imp}(M) + StCI_{\exp}(M) = \sum_{m\ unit\ of\ M} StCI_{imp}(m) + \sum_{m\ unit\ of\ M} StCI_{\exp}(m)$$

$$StCI(S) = \sum_{M \in MC} StCI_{imp}(M) = \sum_{M \in MC} StCI_{\exp}(M)$$

■ *Scalar-Data/Control Interconnection :*

Some scalar variable in m is passed as an actual parameter to n where it has a I-use but no P-use.



$$ScDCI(m) = ScDCI_{imp}(m) + ScDCI_{\exp}(m) = \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{ScDCI}(m,n)\right| + \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{ScDCI}(n,m)\right|$$

$$ScDCI(M) = ScDCI_{imp}(M) + ScDCI_{\exp}(M) = \sum_{m\ unit\ of\ M} ScDCI_{imp}(m) + \sum_{m\ unit\ of\ M} ScDCI_{\exp}(m)$$

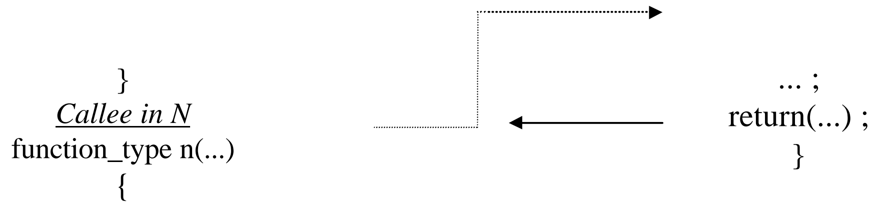
$$ScDCI(S) = \sum_{M \in MC} ScDCI_{imp}(M) = \sum_{M \in MC} ScDCI_{\exp}(M)$$

■ *Return-Data/Control Interconnection :*

m and n are connected by a « return » statement. The returned value has a I-use.

Caller in M

```
m(...)
{
... ;
z=expr(n(...)) ; /* an I-use of */
pred_stat(z) ; /* the returned value */
```

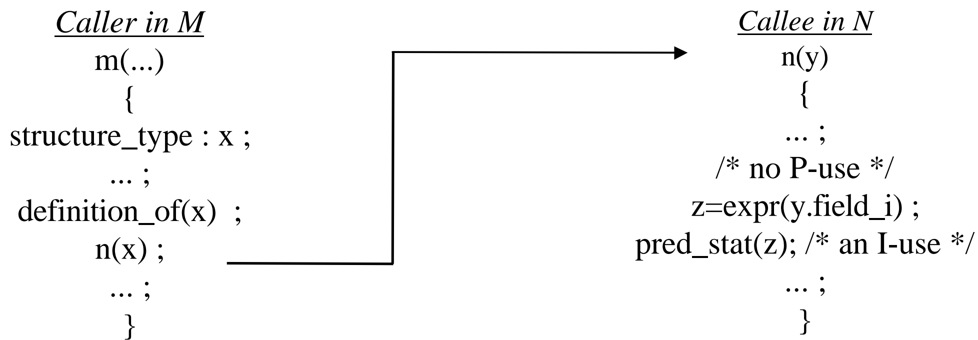


$$RDCI(m) = RDCI_{imp}(m) + RDCI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{RDCI}(m,n)\right| + \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{RDCI}(n,m)\right|$$

$$RDCI(M) = RDCI_{imp}(M) + RDCI_{\exp}(M) = \sum_{m \text{ unit of } M} RDCI_{imp}(m) + \sum_{m \text{ unit of } M} RDCI_{\exp}(m)$$

$$RDCI(S) = \sum_{M \in MC} RDCI_{imp}(M) = \sum_{M \in MC} RDCI_{\exp}(M)$$

■ *Stamp-Data/Control Interconnection :*
A structure in m is passed as an actual parameter to n where it has a I-use but no P-use.

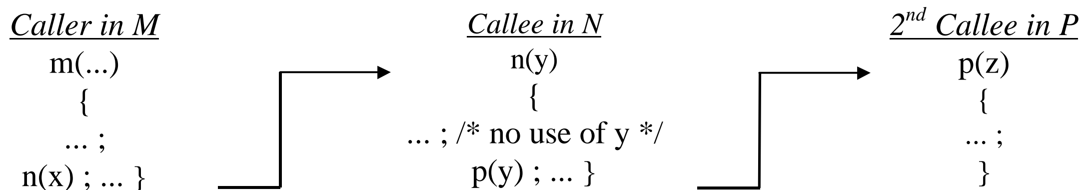


$$StDCI(m) = StDCI_{imp}(m) + StDCI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{StDCI}(m,n)\right| + \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{StDCI}(n,m)\right|$$

$$StDCI(M) = StDCI_{imp}(M) + StDCI_{\exp}(M) = \sum_{m \text{ unit of } M} StDCI_{imp}(m) + \sum_{m \text{ unit of } M} StDCI_{\exp}(m)$$

$$StDCI(S) = \sum_{M \in MC} StDCI_{imp}(M) = \sum_{M \in MC} StDCI_{\exp}(M)$$

■ *Tramp Interconnection :*
A formal parameter in m is passed to n ; n passes this latter to another unit p ∈ P without having accessed or changed the variable.



$$TrI(m) = TrI_{imp}(m) + TrI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{TrI}(m,n)\right| + \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{TrI}(n,m)\right|$$

$$TrI(M) = TrI_{imp}(M) + TrI_{\exp}(M) = \sum_{m \text{ unit of } M} TrI_{imp}(m) + \sum_{m \text{ unit of } M} TrI_{\exp}(m)$$

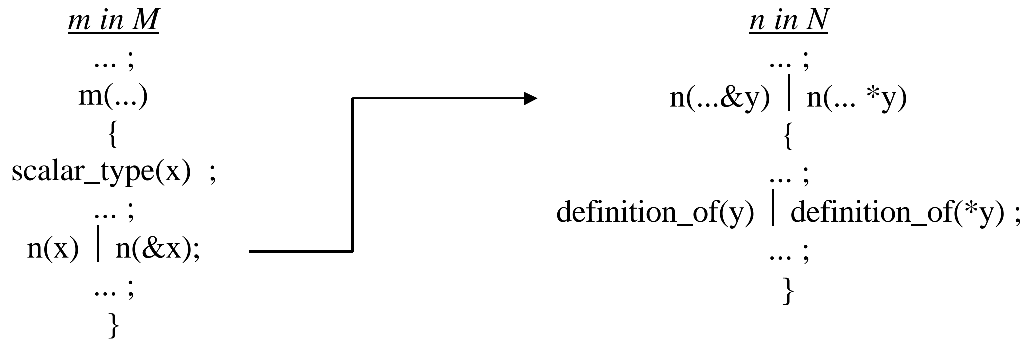
$$TrI(S) = \sum_{M \in MC} TrI_{imp}(M) = \sum_{M \in MC} TrI_{\exp}(M)$$

3.3.1.2. Content Interconnection :

This case occurs when the callee unit n of module N refers and changes parameters passed by the caller unit m of module M. These parameters are passed by address (or reference). We identify two kind of such interconnections.

■ *Scalar-Reference Interconnection :*

The address of a scalar variable in m is passed as an actual parameter to n where it is modified.



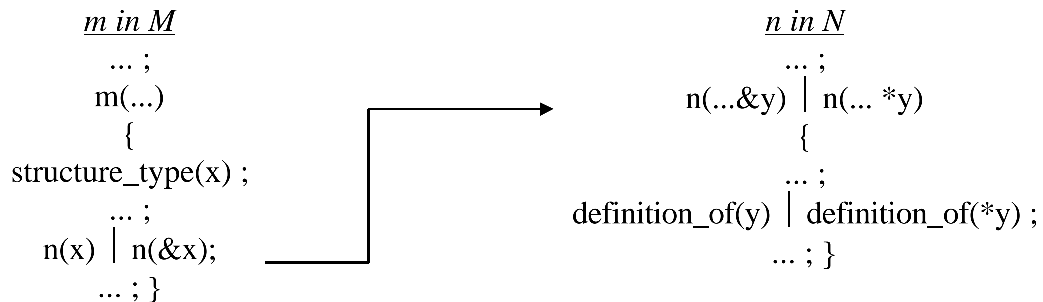
$$ScRI(m) = ScRI_{imp}(m) + ScRI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{ScRI}(m,n)\right| + \sum_{n \text{ unit of } N \in (MC-M)} \left|CMI_{ScRI}(n,m)\right|$$

$$ScRI(M) = ScRI_{imp}(M) + ScRI_{\exp}(M) = \sum_{m \text{ unit of } M} ScRI_{imp}(m) + \sum_{m \text{ unit of } M} ScRI_{\exp}(m)$$

$$ScRI(S) = \sum_{M \in MC} ScRI_{imp}(M) = \sum_{M \in MC} ScRI_{\exp}(M)$$

■ *Stamp-Reference Interconnection :*

The address of a structure variable in m is passed as an actual parameter to n where it is modified.



$$StRI(m) = StRI_{imp}(m) + StRI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left| CMI_{StRI}(m,n) \right| + \sum_{n \text{ unit of } N \in (MC-M)} \left| CMI_{StRI}(n,m) \right|$$

$$StRI(M) = StRI_{imp}(M) + StRI_{\exp}(M) = \sum_{m \text{ unit of } M} StRI_{imp}(m) + \sum_{m \text{ unit of } M} StRI_{\exp}(m)$$

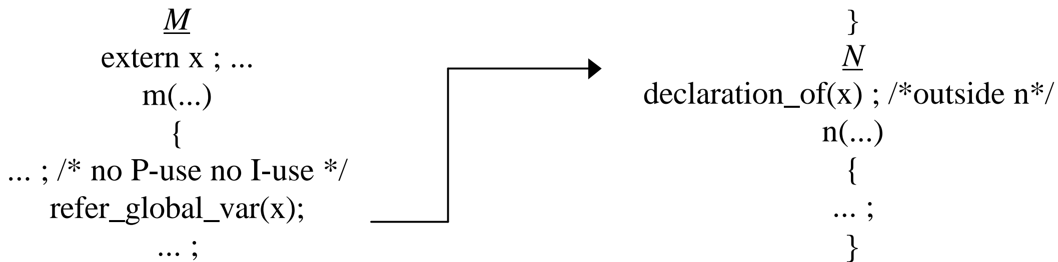
$$StRI(S) = \sum_{M \in MC} StRI_{imp}(M) = \sum_{M \in MC} StRI_{\exp}(M)$$

3.3.2. Common Interconnection :

It corresponds to the case where two modules share same « global spaces ». Instead of communicating with one another by passing parameters, two modules access and eventually change information in a global area. We distinguish five interesting kinds of interconnection :

■ Global-Data Interconnection :

M and N share references to the same global variable. This latter is defined and used in N and C-used in M. It would be possible that this variable is not visible to the entire system.



declaration_of(x) → type x

type → int | float | char | structure ... | ...

refer_global_var(x) → c_use(x)

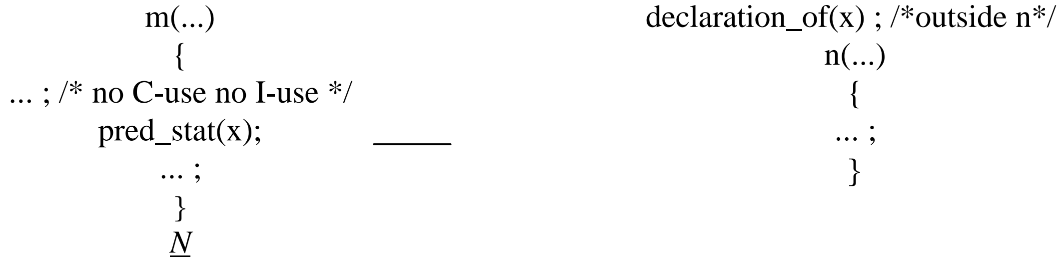
$$GDI(m) = GDI_{imp}(m) + GDI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left| CMI_{GDI}(m,n) \right| + \sum_{n \text{ unit of } N \in (MC-M)} \left| CMI_{GDI}(n,m) \right|$$

$$GDI(M) = GDI_{imp}(M) + GDI_{\exp}(M) = \sum_{m \text{ unit of } M} GDI_{imp}(m) + \sum_{m \text{ unit of } M} GDI_{\exp}(m)$$

$$GDI(S) = \sum_{M \in MC} GDI_{imp}(M) = \sum_{M \in MC} GDI_{\exp}(M)$$

■ *Global-Control Interconnection :*

M and N share references to the same global variable. This latter is defined and used in N and P-used in M.

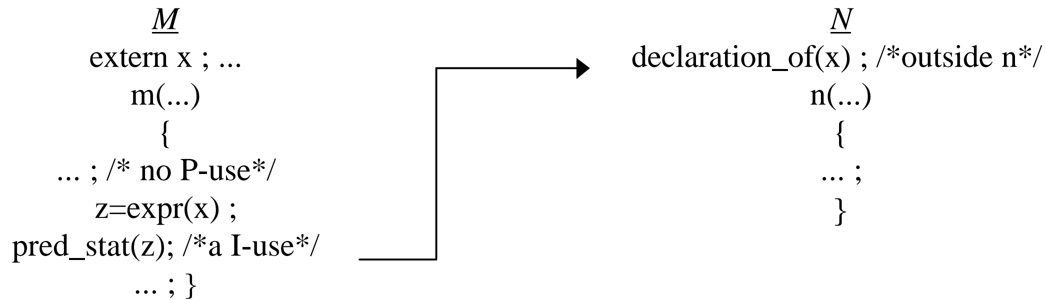


$$GCI(m) = GCI_{imp}(m) + GCI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left| CMI_{GCI}(m,n) \right| + \sum_{n \text{ unit of } N \in (MC-M)} \left| CMI_{GCI}(n,m) \right|$$

$$GCI(M) = GCI_{imp}(M) + GCI_{\exp}(M) = \sum_{m \text{ unit of } M} GCI_{imp}(m) + \sum_{m \text{ unit of } M} GCI_{\exp}(m)$$

$$GCI(S) = \sum_{M \in MC} GCI_{imp}(M) = \sum_{M \in MC} GCI_{\exp}(M)$$

■ *Global-Data/Control Interconnection :*
M and N share references to the same global variable. This latter is defined and used in N and I-used in M but no P-used.

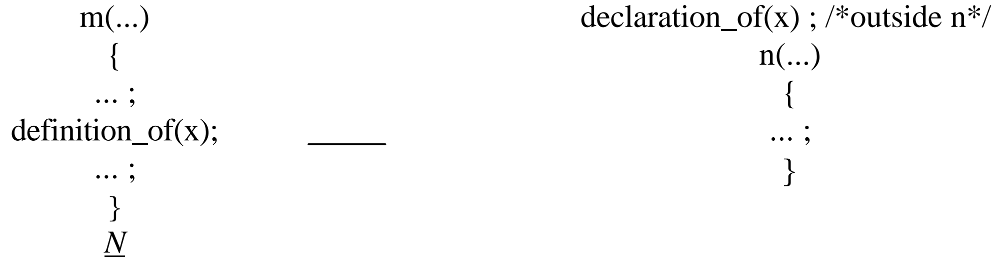


$$GDCI(m) = GDCI_{imp}(m) + GDCI_{\exp}(m) = \sum_{n \text{ unit of } N \in (MC-M)} \left| CMI_{GDCI}(m,n) \right| + \sum_{n \text{ unit of } N \in (MC-M)} \left| CMI_{GDCI}(n,m) \right|$$

$$GDCI(M) = GDCI_{imp}(M) + GDCI_{\exp}(M) = \sum_{m \text{ unit of } M} GDCI_{imp}(m) + \sum_{m \text{ unit of } M} GDCI_{\exp}(m)$$

$$GDCI(S) = \sum_{M \in MC} GDCI_{imp}(M) = \sum_{M \in MC} GDCI_{\exp}(M)$$

■ *Global-Change Interconnection :*
M and N share references to the same global variable. This latter is defined and used in N and accessed and modified in M.

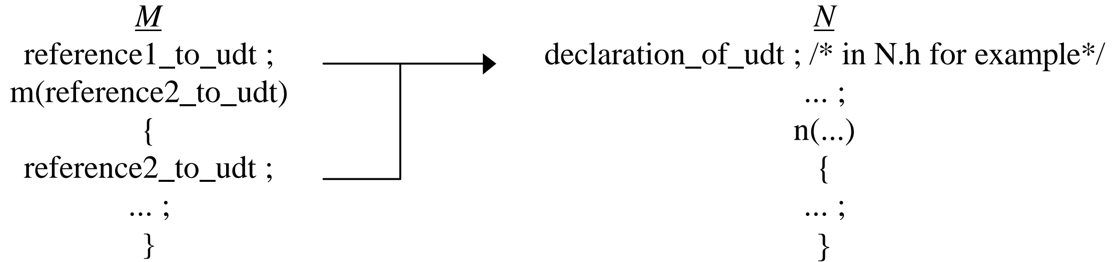


$$GChI(m) = GChI_{imp}(m) + GChI_{\exp}(m) = \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{GChI}(m,n)\right| + \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{GChI}(n,m)\right|$$

$$GChI(M) = GChI_{imp}(M) + GChI_{\exp}(M) = \sum_{m\ unit\ of\ M} GChI_{imp}(m) + \sum_{m\ unit\ of\ M} GChI_{\exp}(m)$$

$$GChI(S) = \sum_{M \in MC} GChI_{imp}(M) = \sum_{M \in MC} GChI_{\exp}(M)$$

■ *Type Interconnection :*
M and N share references to the same User Date Type (UDT). This UDT is defined and used in N and used in M. This kind of interconnection includes what previous works called external-medium coupling (communication through a file, ...).



declaration_of_udt → typedef .... | structure ... | class ...
reference1_to_udt → #include <N.h>
reference2_to_udt → occurrence of the udt defined before

$$TI(m) = TI_{imp}(m) + TI_{\exp}(m) = \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{TI}(m,n)\right| + \sum_{n\ unit\ of\ N \in (MC-M)} \left|CMI_{TI}(n,m)\right|$$

$$TI(M) = TI_{imp}(M) + TI_{\exp}(M) = \sum_{m\ unit\ of\ M} TI_{imp}(m) + \sum_{m\ unit\ of\ M} TI_{\exp}(m)$$

$$TI(S) = \sum_{M \in MC} TI_{imp}(M) = \sum_{M \in MC} TI_{\exp}(M)$$

4. Conclusion

The goal of this work is two folds. Firstly, we are concerned on identifying the different forms coupling can appear in a modular software system, because each kind of coupling may have different impacts on software quality. Secondly, we are engaged in measuring the different kinds of coupling and evaluating its impact on error-proneness (a software quality attribute).
The next steps of our work is first to extract the suite of metrics directly from the source code, then empirically validate this suite of metrics regarding its capabilities to predict software quality. The main technical risk here would come from insufficient validation; so a sufficient, minimal level of validation will be guaranteed.
The suite of metrics must be useful in predicting software quality attributes when used stand-alone or combined with existing metrics. It may happen that the suite will work better when combined with other metrics. In this case, it will be necessary to establish what kind of software quality attributes can be better predicted using different combinations of coupling measures. To do so, it will be necessary to extract several metrics from the same data set we will use. This will imply the use of existing tools, or construction of new tools which will allow me to extract metrics defined by other researchers from our data set.
The suite of metrics will be also used to generate prescriptive coding guidelines to improve coding practices so that the cost and uncertainty can be reduced. To do so, we will use techniques already used in software engineering, e.g., classification trees [B+97], OSR, logistic regression. The rules generated by these techniques will be validated with software designers.

References

[AC94] : F. B. Abreu and R. Carapuça. "Candidate metrics for object-oriented software within a taxonomy framework". *Journal of System and Software*, 26(1):87–96, 1994.

{B+97] : V. R. Basili, S. E. Condon, K. El Emam, R. Hendrick, and W. L. Melo. "Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components", in Proc. of the 19th Int'l Conf. on Software Engineering, Boston, MA, 1997. IEEE Press.

[BBM96] : V. Basili, L. Briand, and W. Melo. "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE TSE, vol 22, no. 10, October, 1996.

[BDM97] : L. Briand, P. Devanbu, and W. L. Melo. "An Investigation into Coupling Measures for C++". In Proc. of the 19th Int'l Conf. on Software Engineering, Boston, MA, 1997. IEEE Press.

[BMB96] : L. C. Briand, S. Morasca, and V. R. Basili. "Property-Based Software Engineering Measurement", IEEE TSE, 22(1), 68-85, 1996.

[CK94] : S. R. Chidamber and C. F. Kemerer. “A metrics suite for object-oriented design.”, *IEEE TSE*, 20(6), 476–493, 1994.

[CY79] : Constantine and Yourdon. "Structured Design", Prentice-Hall, Englewood Cliffs, NJ, 1979.

[OHK93] : A. J. Offutt, M. J. Harrold, and P. Kolte. "A Software Metric System for Module Coupling", Journal on Software and System, 1993.

[PJ80] : M. Page-Jones. "The Practical Guide to Structured Systems Design", Yourdon Press, New York, NY, 1980.

[SO92] : I. Sommerville. "Software Engineering", Addison-Wesley, fourth edition, 1992.