Checking Coding Rules in OO Languages Using CRISP

Guillem Marpons

Joint work with:
Julio Mariño  Manuel Carro  Ángel Herranz
Lars-Åke Fredlund  Juan José Moreno-Navarro

1Universidad Politécnica de Madrid
gmarpons@fi.upm.es

COST Action IC0701 Meeting, Madrid December 2008
Motivation: C++ “Strange” Behavior

class A
{
public:
    A();
    virtual void func();
};

class B : public A
{
    B() : A() {}
    virtual void func();
};

A::A() {
    func();
}

B *d = new B();

// A::func or B::func?
Motivation: C++ “Strange” Behavior

class A
{
public:
    A();
    virtual void func();
};

class B : public A
{
    B() : A() {}
    virtual void func();
};

A::A() {
    func();
}

B *d = new B();
// A::func or B::func?

Coding Rule:
“Do not invoke virtual methods of the declared class in a constructor or destructor.”
Coding Rules

Definition

Coding Rules constrain admissible constructs of a language to help produce more reliable and maintainable code.

Standard coding rule sets do exist, e.g.:

- High-Integrity C++ (HICPP): general C++ applications
- MISRA-C (C language): automotive industry / embedded systems

Many organisations need to write their own rule sets or adapt existing ones.
Other Tools

Proprietary tools:
- Compilers: IAR Systems (C)
- QA: Parasoft, Klocwork, Coverity, Semmle Code (Java)

Free software:
- Checkstyle (Java)
- Gendarme (ECMA CIL, Mono and .Net)

Drawbacks:
- Lack of appropriate extensibility mechanisms
- Ambiguity in natural language
- Interoperability is difficult
Proposed Approach

1. Formalize rules in a logic-based specification language that is executable: CRISP

2. Use GCC for gathering necessary program information
Our Rule Checking Procedure

1. Coding rule(s) written once in the logic-based formalism
2. Extract program information (+ analysis information if available) using GCC, and store it
3. Search (using a Prolog engine) for a counterexample
Our Rule Checking Procedure

1. Coding rule(s) written *once* in the logic-based formalism
2. Extract program information (+ analysis information if available) using GCC, and store it
3. Search (using a Prolog engine) for a counterexample
Our Rule Checking Procedure

1. Coding rule(s) written once in the logic-based formalism
2. Extract program information (+ analysis information if available) using GCC, and store it
3. Search (using a Prolog engine) for a counterexample
Our Rule Checking Procedure

1. Coding rule(s) written once in the logic-based formalism
2. Extract program information (+ analysis information if available) using GCC, and store it
3. Search (using a Prolog engine) for a counterexample
Variable,DataMember,LocalVariable
Function,MemberFunction,Constructor
Type,PointerType,Record
Scope,Namespace,Record,CompoundStatement
Operator
ArgumentTypeInFunctionType
ClassMember
Thing
<table>
<thead>
<tr>
<th>Class</th>
<th>Relation</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>calls</td>
<td>Function</td>
</tr>
<tr>
<td>Record</td>
<td>hasImmediateBase</td>
<td>Record</td>
</tr>
<tr>
<td>Variable</td>
<td>hasType</td>
<td>NonFunctionType</td>
</tr>
<tr>
<td>Function</td>
<td>hasType</td>
<td>FunctionType</td>
</tr>
<tr>
<td>Thing</td>
<td>isDefinedIn</td>
<td>Scope</td>
</tr>
<tr>
<td>Scope</td>
<td>isNestedIn</td>
<td>Scope</td>
</tr>
<tr>
<td>Record</td>
<td>hasMember</td>
<td>MemberFunction</td>
</tr>
<tr>
<td>Record</td>
<td>hasMember</td>
<td>DataMember</td>
</tr>
<tr>
<td>Record</td>
<td>hasBase</td>
<td>Record</td>
</tr>
<tr>
<td>Record</td>
<td>isPrivateBaseOf</td>
<td>Record</td>
</tr>
<tr>
<td>Record</td>
<td>isVirtualBaseOf</td>
<td>Record</td>
</tr>
<tr>
<td>PointerType</td>
<td>hasPointedType</td>
<td>Type</td>
</tr>
<tr>
<td>FunctionType</td>
<td>hasReturnType</td>
<td>Type</td>
</tr>
<tr>
<td>Record</td>
<td>hasFriend</td>
<td>Record</td>
</tr>
<tr>
<td>Record</td>
<td>hasFriend</td>
<td>MemberFunction</td>
</tr>
<tr>
<td>ClassMember</td>
<td>hasVisibility</td>
<td>Visibility</td>
</tr>
</tbody>
</table>

Guillem Marpons et al. (UPM)  Coding Rule Checking  Madrid December 2008  8 / 18
Rule HICPP 3.3.13:

“Do not invoke virtual methods of the declared class in a constructor or destructor.”
Example of Rule Formalization

Rule HICPP 3.3.13:

“Do not invoke virtual methods of the declared class in a constructor or destructor.”

rule HICPP 3.3.13’
violated by Caller : MemberFunction; Callee : VirtualFunction
when exists R : Record such that

( R hasMember Caller
and R hasMember Callee
and
( Caller is Constructor
or Caller is Destructor
)
and Caller calls+ Callee
)
Auxiliary Sorts and Relations

relation F : Function overloads F' : Function when exists S : Scope ; N : String such that (  
    F isDefinedIn S  
    and F' isDefinedIn S  
    and F hasUnqualifiedName N  
    and F' hasUnqualifiedName N  
    and F \neq F'  
  )  
.

sort M : ClassMember is PrivateClassMember when exists V : Visibility such that (  
    M hasVisibility V and V is ‘private’  
  )  
.

Guillem Marpons et al. (UPM) Coding Rule Checking Madrid December 2008 10 / 18
Integration into GCC

- GlobalGCC Project (Eureka/ITEA). Enhance the GNU compiler collection with:
  - Project-wide static analysis
  - Global optimization
  - Coding rule checking

- Huge potential user base

- All facts of a project in a single file

- User interface. Two steps:
  1. `g++ -fipa-codingrules -fipa-codingrules-file=FILENAME`
     Pass appropriate CXX to `./configure`
  2. `checkrules -s RULE_SET [-r RULES] FILENAME`

Code available at [http://www.ggcc.info](http://www.ggcc.info)
### Experimental Results

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>KLOC</th>
<th>LOAD TIME</th>
<th># VIOLATIONS (CHECKING TIME)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.3.1</td>
</tr>
<tr>
<td>Bacula</td>
<td>20</td>
<td>0.24</td>
<td>0 (0.0)</td>
</tr>
<tr>
<td>CLAM</td>
<td>46</td>
<td>1.62</td>
<td>1 (0.0)</td>
</tr>
<tr>
<td>Firebird</td>
<td>439</td>
<td>2.61</td>
<td>16 (0.0)</td>
</tr>
<tr>
<td>IT++</td>
<td>39</td>
<td>0.42</td>
<td>0 (0.0)</td>
</tr>
<tr>
<td>OGRE</td>
<td>209</td>
<td>3.05</td>
<td>0 (0.0)</td>
</tr>
<tr>
<td>Orca</td>
<td>89</td>
<td>1.17</td>
<td>1 (0.0)</td>
</tr>
<tr>
<td>Qt</td>
<td>595</td>
<td>10.42</td>
<td>15 (0.0)</td>
</tr>
</tbody>
</table>

All times expressed in seconds.
Work in Progress

1 Implement / Enrich the CRISP Language

2 Implement more rules with information given by other tools

3 Open our abstract representation of programs to external tools
Implement / enrich the CRISP language

- Quantification and true negation needed
  - Both performed over certain domains (sorts)
  - Infinite domains may appear with templates / generics
  - We have an implementation of constructive intensional negation

- Goals automatically reordered

- Extend CRISP to other languages: Java, Ada, C, Fortran, ...
Examples of Rules that Need Specific Analysis

**Rule HICPP 3.3.13:**
“Do not invoke virtual methods of the declared class in a constructor or destructor.”

**Rule HICPP 3.2.5:**
“Ensure destructors release all objects owned by the object”

**Rule HICPP 3.4.2:**
“Do not return non-const handles to class data from const member functions”
### Examples of Rules that Need Specific Analysis

**Rule HICPP 3.3.13:**

“Do not invoke virtual methods of the declared class in a constructor or destructor.”

**Rule HICPP 3.2.5:**

“Ensure destructors release all objects owned by the object”

**Rule HICPP 3.4.2:**

“Do not return non-const handles to class data from const member functions”
Examples of Rules that Need Specific Analysis

Rule HICPP 3.3.13:
“Do not invoke virtual methods of the declared class in a constructor or destructor.”

Rule HICPP 3.2.5:
“Ensure destructors release all objects owned by the object”

Rule HICPP 3.4.2:
“Do not return non-const handles to class data from const member functions”
Integration of Information from External Analyzers

Coding rules (in English)  C++ project source files

Coding rules formalized in CRISP$_{C++}$  g++' (project build)

Coding rule compiler  Project facts in Prolog

Coding rules compiled into Prolog  Ciao Prolog engine

Rule violations report

Guillem Marpons et al. (UPM)  Coding Rule Checking  Madrid December 2008  16 / 18
Integration of Information from External Analyzers

- Coding rules (in English)
- Coding rules formalized in CRISP$_{C++}$
- Coding rule compiler
- C++ project source files
- External Analyzer
- Translation

Knowledge Base about the compiled program

Ciao Prolog engine

Rule violations report
Open the Knowledge Base to External Tools
Thanks for your Attention!

- Questions?
- Comments?