Un environnement de développement C/C++ interactif

Cours Robots 2007, le 20 Mars 2007
Cyrille Berger

D'après une Présentation RIA, le 10 décembre 2004
Thomas Lemaire, Maxime Cottret, Sébastien Bosch

Plan

- petit historique : de vizir à jafar
- comparatif calife/jafar
- généralités sur Jafar
- module Jafar
- swig, STL, boost
- gestion des erreurs
- documentation
- tests unitaires

Historique

- vizir (ViZion In Robotique)
  - 1981 – 1986
  - projet ARA (bras manipulateur et tapis roulant…)
- calife
  - 1983 – 2005
  - 20^{aine} de thésards
  - Hilare Junior, Adam, Lama, Dala, Karma
- jafar
  - 2004 - ...

	Calife	***Jafar***
langage	C	C/C++
shell	Tcl	Tcl, Ruby,...
environnement modulaire	oui	oui
squelette de module	non	oui
version de module	non	version.revision
interface langage/shell	iwac	swig
chargement modules	statique (shell)	dynamique (module tcl)
aide en ligne (tcl/C)	oui/non	oui/bof
complétion	oui (built-in)	oui (eltclsh, tclreadline)
arguments par défaut	oui (iwak)	oui (C++)
erreur	variable globale	exceptions
test unitaire	non	oui (boost)
documentation	doc utilisateur, source	doxygen
compatibilité shell GenoM	non	oui
librairies externes	non	STL, openCV, boost (uBlas, unit_test, BGL,...)
installation	checkout	checkout + librairies

Environnement Jafar

- Modulaire
  - commande *jafar-module*
  - modèle de module dans `share/template_module`
  - `dépendance entre modules (bof...)`
- Interactif
  - shell tcl quelconque, ou shell ruby
  - `share/initJafar.tcl`
  - utilisation de packages tcl/ruby
- Documentation
  - doc/html/index.html

Pourquoi utiliser Jafar ?

- Jafar vs Genom
  - genom: utilisation finale dans les robots
  - jafar: aide au dévelopement d'algorithmes
- réutilisabilité
  - accès aux algorithmes existants
  - partage des vos nouveaux algorithmes
  - partage des fonctionnalités annexes (visualisation, lecture de données...)

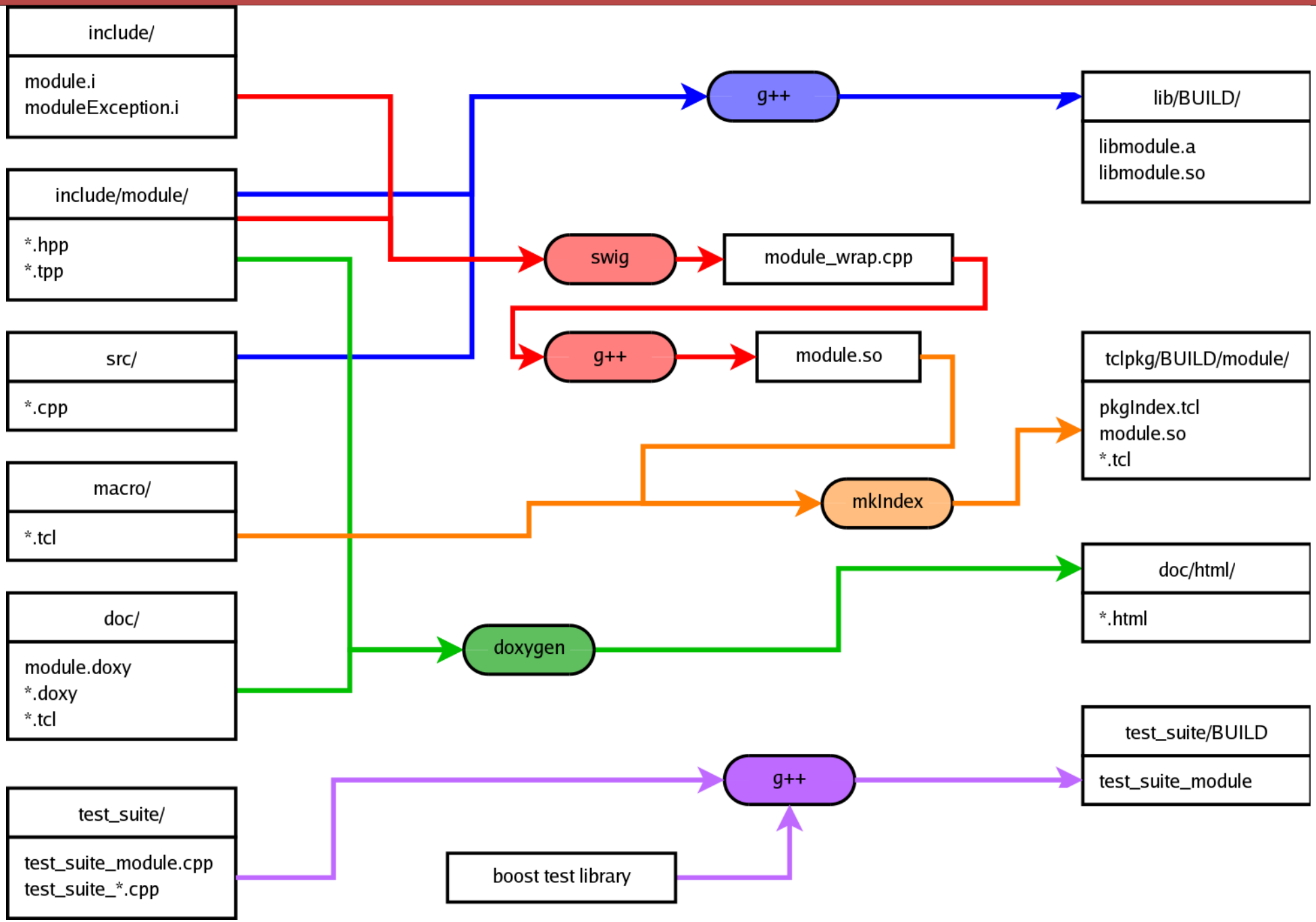
Module Jafar

- Aspects modulaires :
  - espace de nom C++ jafar::monmodule::
  - type d'erreurs MonModuleException
  - package tcl/ruby monmodule
  - module ruby Monmodule
  - espace de nom tcl monmodule::
  - group doxygen monmodule
  - librairies libmonmodule.so,.a
- commande *jafar-module*
  - création d'un nouveau module
  - mise à jour

jafar-module -c

```
pollux[modules] ../bin/jafar-module -c pipo
Creating module pipo
  template dir    : /home/tlemaire/Work/jafar/share/template_module
  destination dir : /home/tlemaire/Work/jafar/modules/pipo

Directory /home/tlemaire/Work/jafar/modules/pipo...ok
. ... ok
./macro ... ok
./Makefile ... ok
./configure ... ok
./aclocal.m4 ... ok
./src ... ok
./src/pipoException.cpp ... ok
./Makefile.module.in ... ok
./test_suite ... ok
./test_suite/test_suite_pipo.cpp ... ok
./COPYRIGHT ... ok
./include ... ok
./include/pipoException.i ... ok
./include/pipo.i ... ok
./include/pipo ... ok
./include/pipo/pipoException.hpp ... ok
./include/pipo/pipoTcl.hpp ... ok
./doc ... ok
./doc/pipo.doxy ... ok
./README ... ok
./configure.ac ... ok
./User.make ... ok
./.config ... ok
./.config/config.guess ... ok
./.config/missing ... ok
./.config/install-sh ... ok
./.config/user.ac ... ok
./.config/ltmain.sh ... ok
./.config/config.sub ... ok

Module pipo successfully created !
```

Module Jafar



- Modification du `User.make`
- Cibles principales :
  - all : config et module
  - config : lance le script configure avec les options du User.make
  - module : compile et installe les librairies et le module tcl
  - tcl-pkg : installe le module tcl
  - test : compile et lance la test_suite du module
  - clean
  - acf : régénère le script configure (User.ac)

User.make

```
# $Id: User.make 481 2004-11-25 16:08:53Z tlemaire $ #

# module version
MODULE_VERSION = 0
MODULE_REVISION = 1

# configure option
CONF_OPT = --with-boost

# modules dependencies
USE_MODULES = kernel jmath

# to add custom libraries
# concider also modifying .config/User.m4 to check for these libs at
# configure time

# LDFLAGS +=
LIBS += -lkernel -ljmath

CPPFLAGS += $(BOOST_CPPFLAGS) -DBOOST_UBLAS_USE_EXCEPTIONS
CPPFLAGS += -DNDEBUG
# CPPFLAGS += -DJFR_NDEBUG -DNDEBUG

CXXFLAGS += -g -ggdb -Wall
```

- SWIG = "Simple Wrapping Interface Generator"
  - supporte tcl, python, perl,...
  - C++: arguments par défaut, classe, héritage, template,...
  - récupère la documentation au format doxygen (python : *docstring*, tcl : ??)
- tcl
  - *package* tcl ou interpréteur
  - support des *namespace*
  - libswig: std::string, std::vector,...

Swig - Example

documentation doxygen

entêtes pour compiler module_wrap.cpp

entêtes a wrapper en tcl

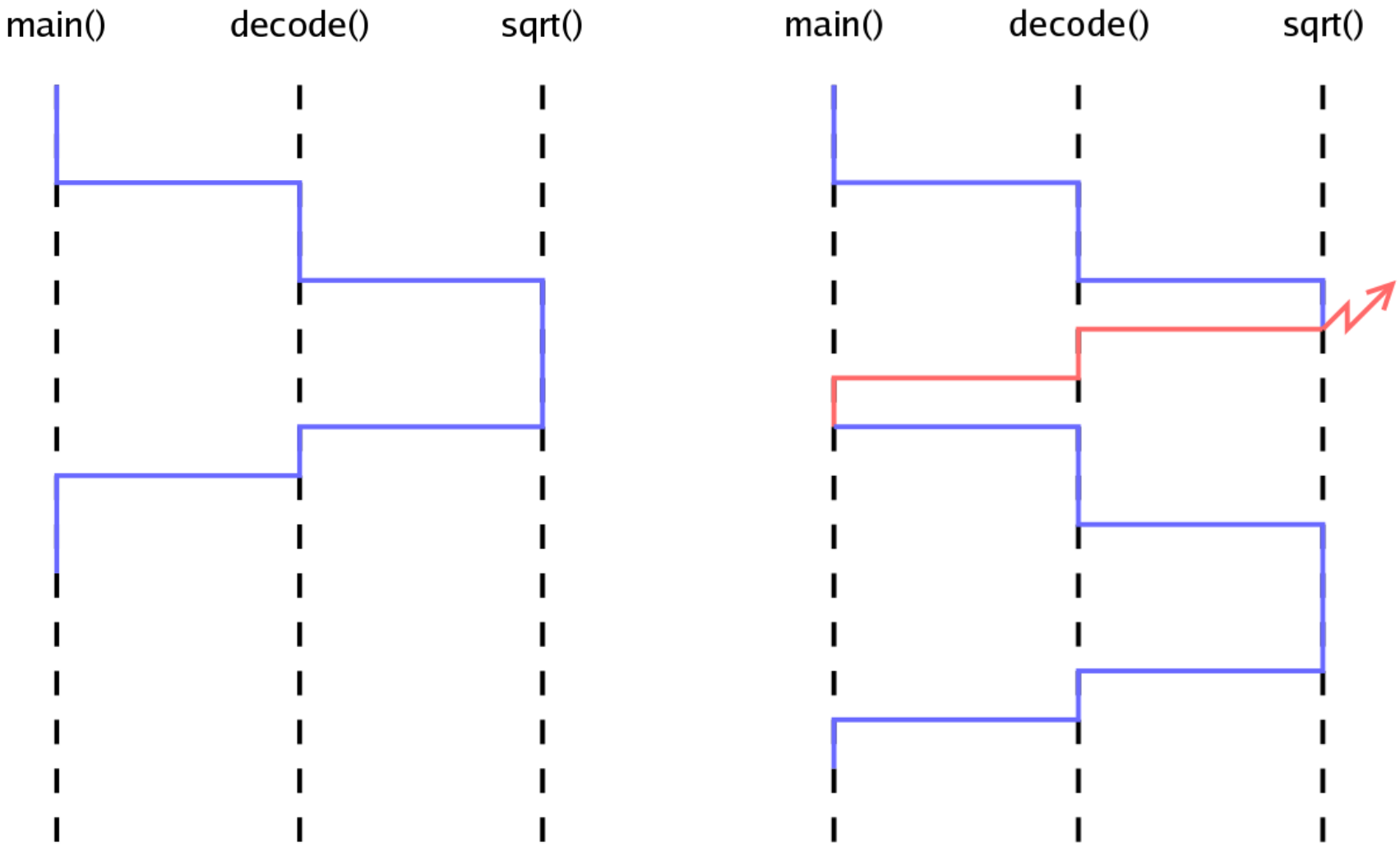
```
/* $Id: helloworld.i 363 2004-10-27 14:01:40Z tlemaire $ */

/** swig/tcl interface file for module helloworld.
 *
 * \file helloworld.i
 * \ingroup helloworld
 */

%module helloworld

%{

#include <string>

// uncomment to control debug output from tcl
#include "kernel/jafarDebug.hpp"

#include "helloworld/helloworldTcl.hpp"

#include "helloworld/helloWorld.hpp"

%}

%include "std_string.i"

%include "helloworldException.i"

// uncomment to control debug output from tcl
%include "kernel/jafarDebug.hpp"

%include "helloworld/helloworldTcl.hpp"
%template(print)
jafar::helloworld::print<jafar::helloworld::HelloWorld>;

%include "helloworld/helloWorld.hpp"
```

- Atouts :
  - *Standard* Template Library
  - intégré à g++
  - performante
  - documentée
- Défauts :
  - lenteur à la compilation (C++, template)
- Exemple :
  - string
  - conteneurs : list, vector, map,...
  - flux E/S

Librairies boost

- Déjà utilisée dans Jafar :
  - random, uBlas (module jmath)
  - date_time
  - unit test
  - smart_ptr
  - BGL (Boost Graph Library)
- Si vous avez besoin...
  - quaternion, octonion, serialization, tokenizer, tuple,...
  - http://www.boost.org
- Ne pas oublier : -DNDEBUG

- Repose sur le mécanisme des exceptions :



Les exceptions dans Jafar

- Informations communes :
  - ID, nom de module, nom de fichier, numéro de ligne, pile d'appel, message
- Les erreurs standards (classe `JafarException`)
  - programmation par contrat (pré-condition, post-condition, invariant)
  - erreur *run-time*
  - macros : `JFR_PRECOND`,...
- Les erreurs de modules
  - simples, classe `ModuleException`
  - complexes, classe utilisateur dérivant de `ModuleException`

Exceptions – Exemple (1/2)

documentation doxygen

types d'exceptions

```
/** Base class for all exceptions defined in the module
 * helloworld.
 *
 * @ingroup helloworld
 */
class HelloworldException : public jafar::kernel::Exception {
public:

  /** This enumeration defines exceptions id for the module
   * helloworld.
   */
  enum ExceptionId {
    EMPTY_HELLO,  /**< The hello string is empty */
    BAD_FORMAT    /**< bad format for string hello */
  };

  /** Constructor. You should not use this constructor directly,
   * prefer macros jfrThrowEx or jfrCreateEx which fill for you
   * parameters \c file_ and \c line_.
   *
   * @param id_ exception id
   * @param message_ message used for debug
   * @param file_ where the exception was thrown
   * @param line_ where the exception was thrown
   */
  HelloworldException(ExceptionId id_,
                      const std::string& message_,
                      const std::string& file_, int line_) throw();
  virtual ~HelloworldException() throw();
  ExceptionId getExceptionId() const throw();

protected:
  ExceptionId id;
  static std::string exceptionIdToString(ExceptionId id_) throw();
};
```

Exceptions – Exemple (2/2)

héritage

nouveaux attributs

```
/** hello attribute not initialized. This is an over simple
  * example on how to add your own exception.
  *
  * @ingroup helloworld
  */
class HelloworldFormatException : public HelloworldException {

 public:

   HelloworldFormatException(const std::string& hello_,
                             const std::string& message_,
                             const std::string& file_, int
line_)
                             throw();

   virtual ~HelloworldFormatException() throw();

   const std::string& getHello() const throw();

 private:

   /// the proposed hello string
   std::string hello;

};
```

Module *kernel*

- Module indispensable, tous les modules sont «linkés» à *libkernel :*
  - exception de base, erreurs génériques
  - message de debug
  - log de données
  - tic, toc, help, ask,...
- Autres modules disponibles :
  - *image (Sébastien, Maxime)*
  - *jmath, filter (Thomas)*

Documentation

- Documentation générale :
  - guide de l'utilisateur,
  - description d'un algorithme
- Documentation dans les sources (.hpp) :
  - fonctions, classes,...
  - extraite des sources et mise en forme
- Doxygen - www.doxygen.org
  - syntaxe à la *javadoc*
  - génération d'une documentation uniforme en html, hyperliens,...
  - ou en pdf (latex), rtf,...

Documentation – Exemple (1/2)

```
/** HelloWorld class. This class is a simple example.
 *
 * \ingroup helloworld
 */
class HelloWorld {

    private :

      /// contains the string to be displayed
      std::string hello;

      static bool checkHello(const std::string& hello_);

    public :

      /// Default constructor.
      HelloWorld();

      /** Constructor.
       * @param hello_ the initial string
       * @pre \a hello_ is not empty
       */
      HelloWorld(const std::string& hello_);

      /// Destructor.
      ~HelloWorld();

 }; // class HelloWorld
```

Documentation – Exemple (2/2)

```
/*!
  \addtogroup helloworld Module helloworld

\version 0.3

\author thomas.lemaire@laas.fr, maxime.cottret@laas.fr

This is a very minimal helloWorld module. It shows how to add a simple
class jafar::helloworld::HelloWorld and a user defined exception
jafar::helloworld::HelloEmptyException.

\section secHelloworldHistory History

  - 0.3 (2004-10-04) - Add test_suite
  - 0.2 (2004-09-20) - Add support for exceptions
  - 0.1 (2004-09-09) - Initial version

\section secHelloworldMacro Macro

  - demoHello.tcl is a very simple macro

\section secHelloworldExample Example use of helloWorld module

\subsection subsecHelloworldOld Using low level swig functions

\include helloworld_old.tcl

\subsection subsecHelloworldNew Using an object interface

\include helloworld_new.tcl
```

Tests unitaires

- *Boost test library*
- Test d'un algorithme :
  - évaluation de l'algorithme pour certaines entrées, comparaison avec le résultat attendu
- Test d'un module :
  - notion de *test_suite*
- Utilisation
  - Non régression (modifications de l'algorithme, mise à jour librairie externe,...)
  - Compilation sur une nouvelle plate-forme
  - lancement périodique des tests, rapport générés automatiquement,…

Tests unitaires – Exemple (1/3)

```
class test_HelloWorld {
 public:

   HelloWorld hw;
   HelloWorld hw2;

   test_HelloWorld() : hw(), hw2("Hello world") {}

   void test_init() {
     BOOST_CHECK_EQUAL(hw.getHello().size(), 0);
     BOOST_CHECK_EQUAL(hw2.getHello(), "Hello world");
   }

   void test_setHello() {
     hw.setHello("Hello world");
     BOOST_CHECK_EQUAL(hw.getHello(), "Hello world");

     BOOST_CHECK_THROW(hw.setHello(""), jafar::kernel::JafarException);

     BOOST_CHECK_THROW(hw.setHello("what ever"), HelloworldFormatException);
   }

   void test_clearHello() {
     hw.clearHello();
     BOOST_CHECK_EQUAL(hw.getHello().size(), 0);
   }
};
```

Tests unitaires – Exemple (2/3)

```
class test_suite_HelloWorld : public test_suite {

 public:

   test_suite_HelloWorld() : test_suite("test_suite_HelloWorld") {
     // add member function test cases to a test suite
     boost::shared_ptr<test_HelloWorld> instance( new test_HelloWorld() );

     test_case* test_case_init       =
       BOOST_CLASS_TEST_CASE( &test_HelloWorld::test_init, instance );
     test_case* test_case_setHello   =
       BOOST_CLASS_TEST_CASE( &test_HelloWorld::test_setHello, instance );
     test_case* test_case_clearHello =
       BOOST_CLASS_TEST_CASE( &test_HelloWorld::test_clearHello, instance );

     test_case_setHello->depends_on( test_case_init );
     test_case_clearHello->depends_on( test_case_init );

     add(test_case_init);
     add(test_case_setHello);
     add(test_case_clearHello);
   }
};
```

Tests unitaires – Exemple (3/3)

```
pollux[helloworld] gmake test

[...]

    ****************************************
    * Running tests for module helloworld *
    ****************************************

test_suite/sparc-solaris2.9/test_suite_helloworld
Running 3 test cases...

*** No errors detected
gmake[1]: Leaving directory `/home/tlemaire/Work/jafar/modules/helloworld'
```

Jafar au LAAS

- documentation :
  - http://www.laas.fr/~tlemaire/jafar
- repository subversion :
  - trois branches : jafarBackbone, jafarModules, tools
  - organisation standard (trunk/, branches/, tags/)