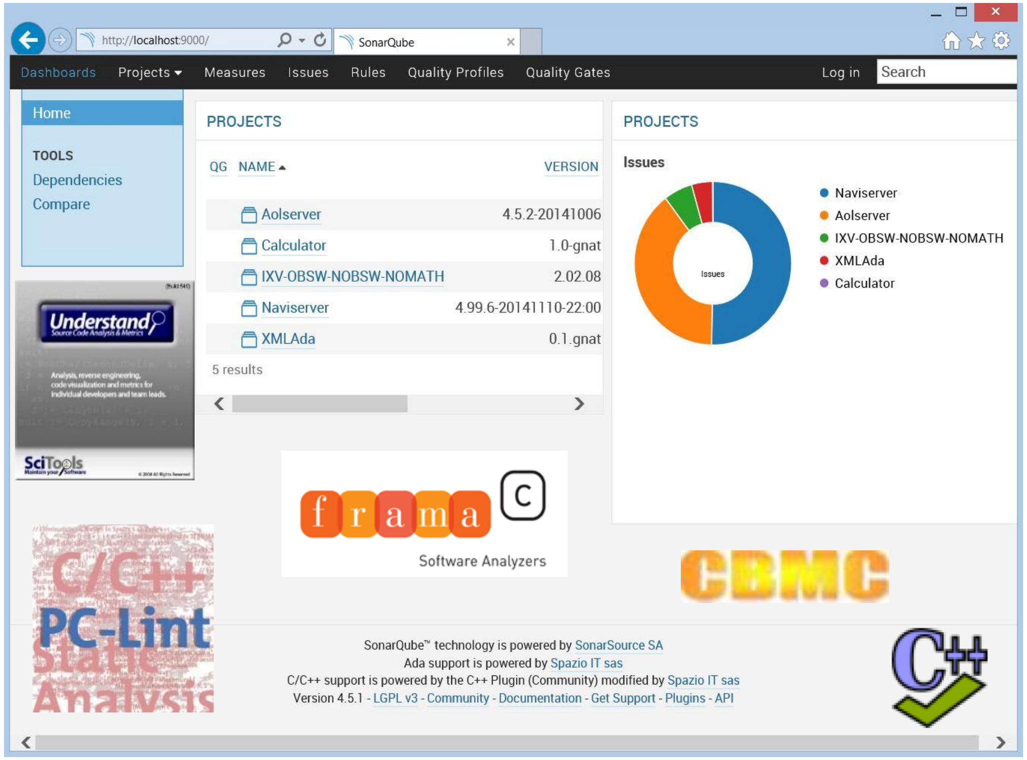
SPAZIO IT

IXV OBSW SOURCE CODE STATIC ANALYSIS



PROJECTS

QG	NAME ▲	VERSION
	Aolserver	4.5.2-20141006
	Calculator	1.0-gnat
	IXV-OBSW-NOBSW-NOMATH	2.02.08
	Naviserver	4.99.6-20141110-22:00
	XMLAda	0.1.gnat

5 results

PROJECTS

Issues

SonarQube™ technology is powered by SonarSource SA
Ada support is powered by Spazio IT sas
C/C++ support is powered by the C++ Plugin (Community) modified by Spazio IT sas
Version 4.5.1 - LGPL v3 - Community - Documentation - Get Support - Plugins - API

Maurizio Martignano
Spazio IT – Soluzioni Informatiche s.a.s
Via Manzoni 40
46030 San Giorgio di Mantova, Mantova
http://www.spazioit.com

Agenda

- IXV OBSW and Spazio IT Code Quality Platform
- Bounded Model Checking and Abstract Intepretation (a proposed methodology)
- Analyses results
- Processes
- Spazio IT Code Quality Platforms @ AIRBUS Helicopters
- Future Work

IXV OBSW and
Spazio IT Code Quality Platform

IXV OBSW and Spazio IT Code Quality Platform

- Spazio IT was requested to perform an activity of V&V on the entire IXV OVSW.

- To this purpose Spazio IT integrated the open source code quality platform SonarQube (http://www.sonarqube.org) with the following tools:
  - CppCheck (http://cppcheck.sourceforge.net/) - open source – a C/C++ static analyser
  - PC-Lint (http://www.gimpel.com/) – proprietary - a rich pattern matching source code static analyzer (mostly used for MISRA C 2004 compliancy checks)

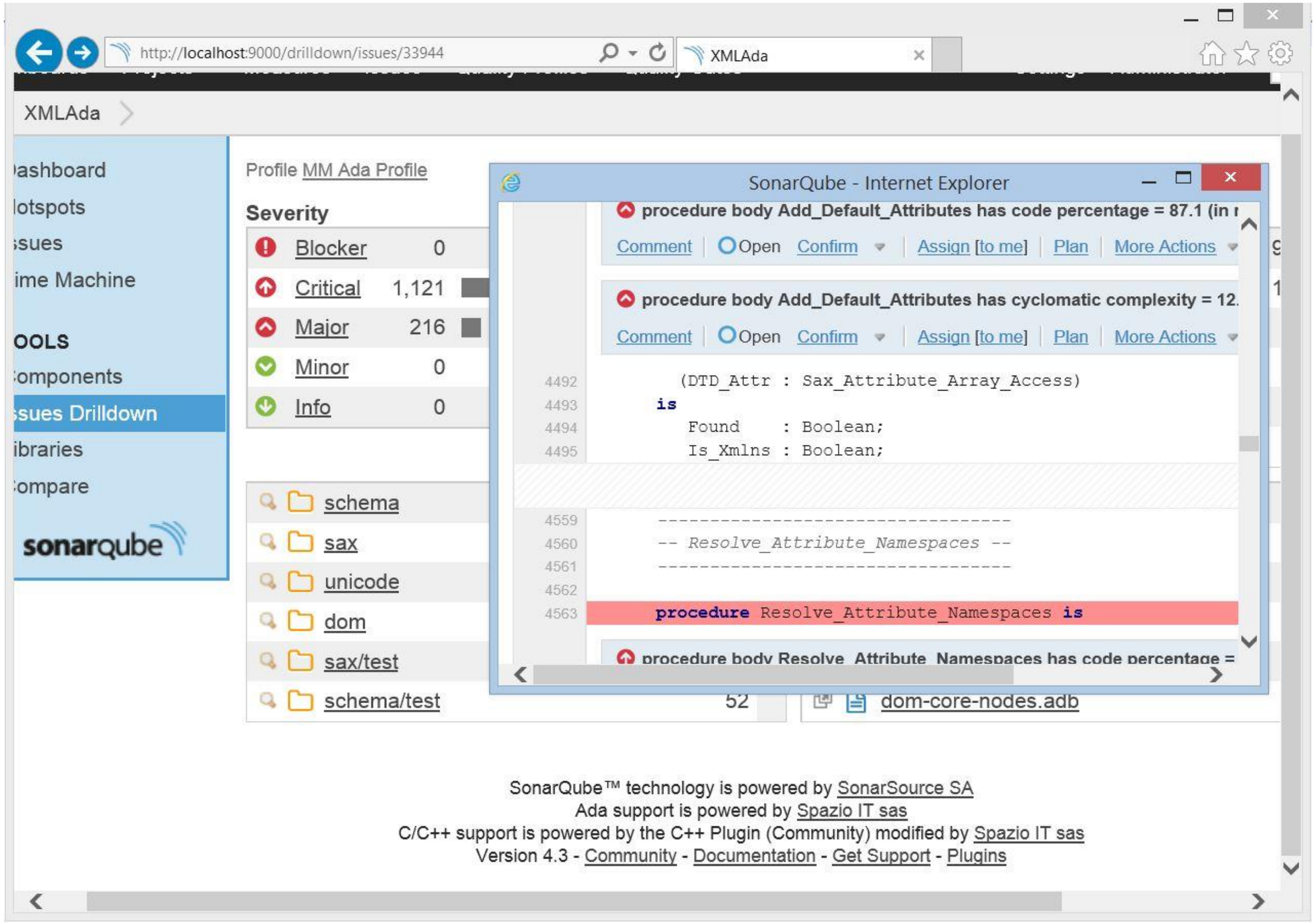
IXV OBSW and Spazio IT Code Quality Platform

- Spazio IT also integrated the following tools to see if they were applicable to the IXV OBSW and could provide additional information:

  - CBMC (http://www.cprover.org/cbmc/) – open source – a C prover based on bounded model checking

  - Frama-C (http://frama-c.com/) – open source – a framework for the static analysis of C code – especially its “value analysis” (i.e. abstract interpretation) and “weakest precondition calculus” plugins.

SonarQube



Profile MM Ada Profile

Severity

●	Blocker	0
●	Critical	1,121
●	Major	216
●	Minor	0
●	Info	0

schema	
sax	
unicode	
dom	
sax/test	
schema/test	52

dom-core-nodes.adb

procedure body Add_Default_Attributes has code percentage = 87.1 (in r

Comment | ○Open Confirm | Assign [to me] | Plan | More Actions

procedure body Add_Default_Attributes has cyclomatic complexity = 12.

Comment | ○Open Confirm | Assign [to me] | Plan | More Actions

```
     (DTD_Attr : Sax_Attribute_Array_Access)
   is
      Found     : Boolean;
      Is_Xmlns  : Boolean;
```

```
   -----------------------------------
   -- Resolve_Attribute_Namespaces --
   -----------------------------------

   procedure Resolve_Attribute_Namespaces is
```

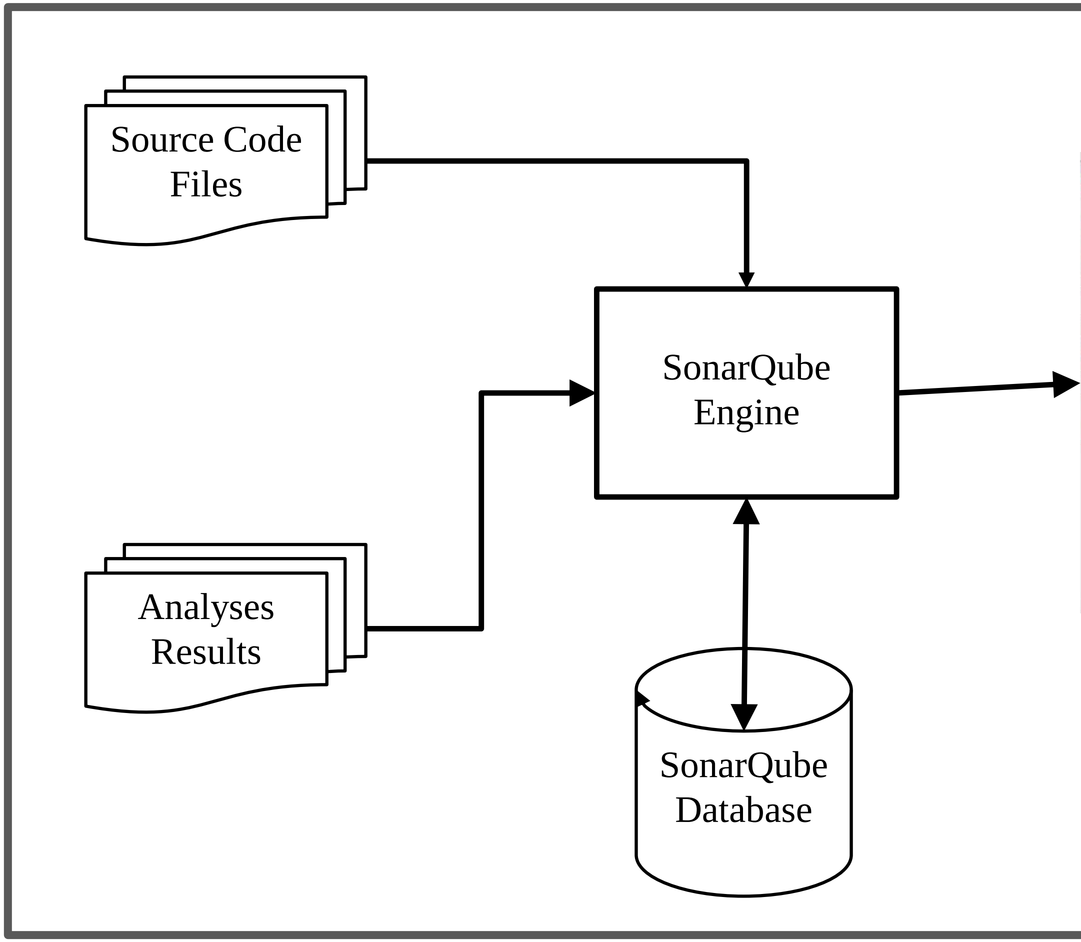
procedure body Resolve_Attribute_Namespaces has code percentage =

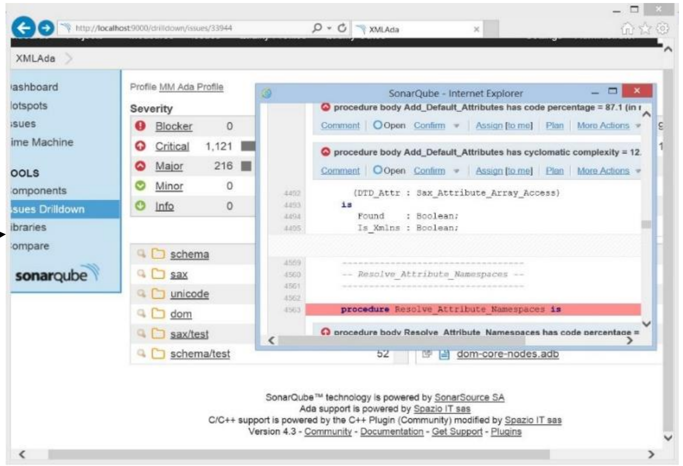
SonarQube™ technology is powered by SonarSource SA
Ada support is powered by Spazio IT sas
C/C++ support is powered by the C++ Plugin (Community) modified by Spazio IT sas
Version 4.3 - Community - Documentation - Get Support - Plugins

SonarQube – What is it?

- SonarQube is an open source Web Application (http://www.sonarqube.org) which
  - Takes in input a set of source code files and a set of analyses results (produced by external tools).
  - Stores both sources and results in a database.
  - Makes available the gathered information via a dynamic website where the results are shown in the context of the code itself.

SonarQube – What is it?





Profile MM Ada Profile

Severity

Blocker	0	
Critical	1,121	▇
Major	216	▇
Minor	0	
Info	0	

schema
sax
unicode
dom
sax/test
schema/test 52

dom-core-nodes.adb

SonarQube - Internet Explorer

procedure body Add_Default_Attributes has code percentage = 87.1 (in r

Comment | ○Open Confirm ▾ | Assign [to me] | Plan | More Actions ▾

procedure body Add_Default_Attributes has cyclomatic complexity = 12.

Comment | ○Open Confirm ▾ | Assign [to me] | Plan | More Actions ▾

```
      (DTD_Attr : Sax_Attribute_Array_Access)
   is
      Found    : Boolean;
      Is_Xmlns : Boolean;
```

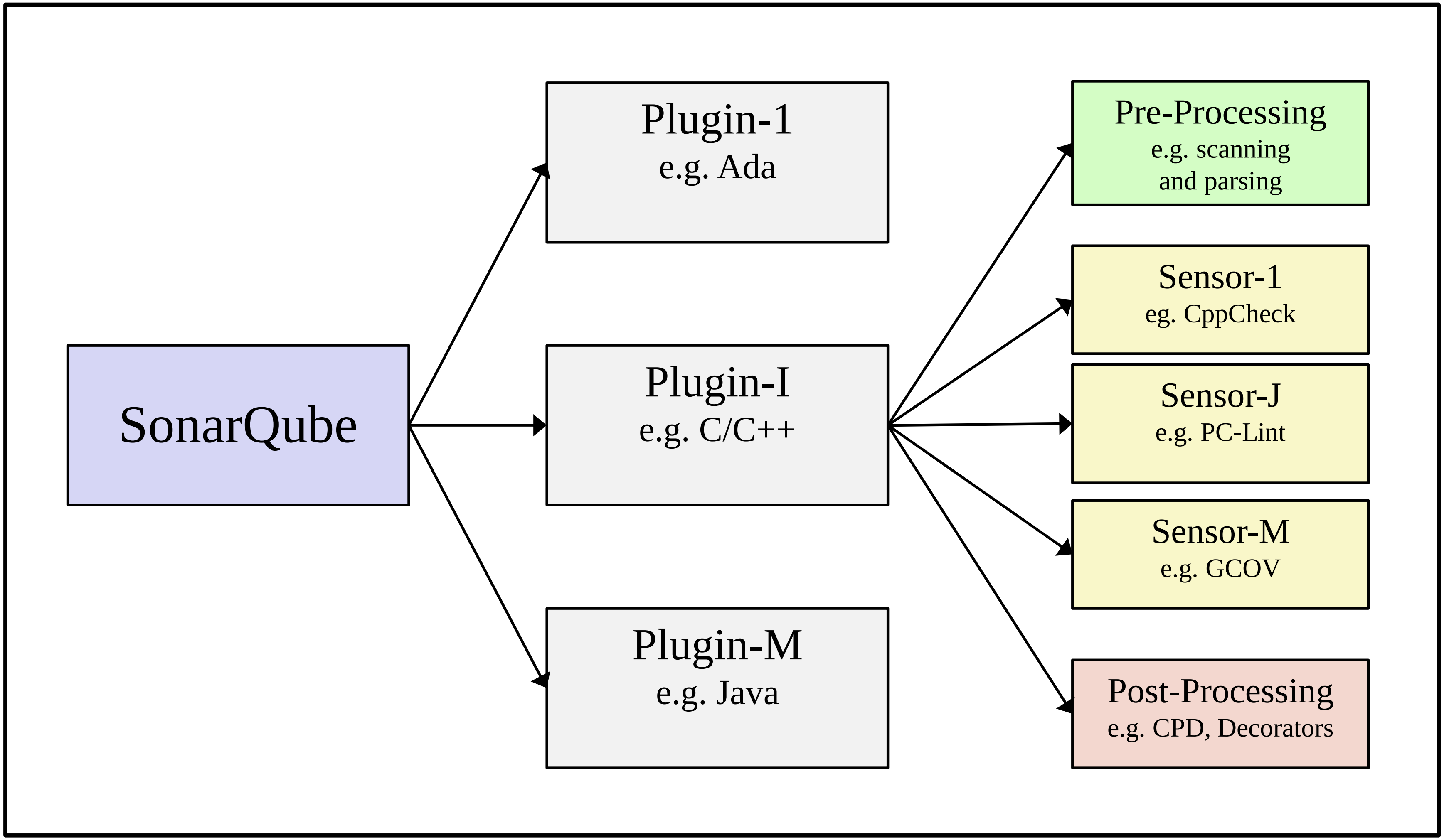
```
   -----------------------------------
   -- Resolve_Attribute_Namespaces --
   -----------------------------------

   procedure Resolve_Attribute_Namespaces is
```

procedure body Resolve_Attribute_Namespaces has code percentage =

SonarQube™ technology is powered by SonarSource SA
Ada support is powered by Spazio IT sas
C/C++ support is powered by the C++ Plugin (Community) modified by Spazio IT sas
Version 4.3 - Community - Documentation - Get Support - Plugins

SonarQube / Plugins / Sensors



SonarQube

Plugin-1
e.g. Ada

Plugin-I
e.g. C/C++

Plugin-M
e.g. Java

Pre-Processing
e.g. scanning
and parsing

Sensor-1
eg. CppCheck

Sensor-J
e.g. PC-Lint

Sensor-M
e.g. GCOV

Post-Processing
e.g. CPD, Decorators

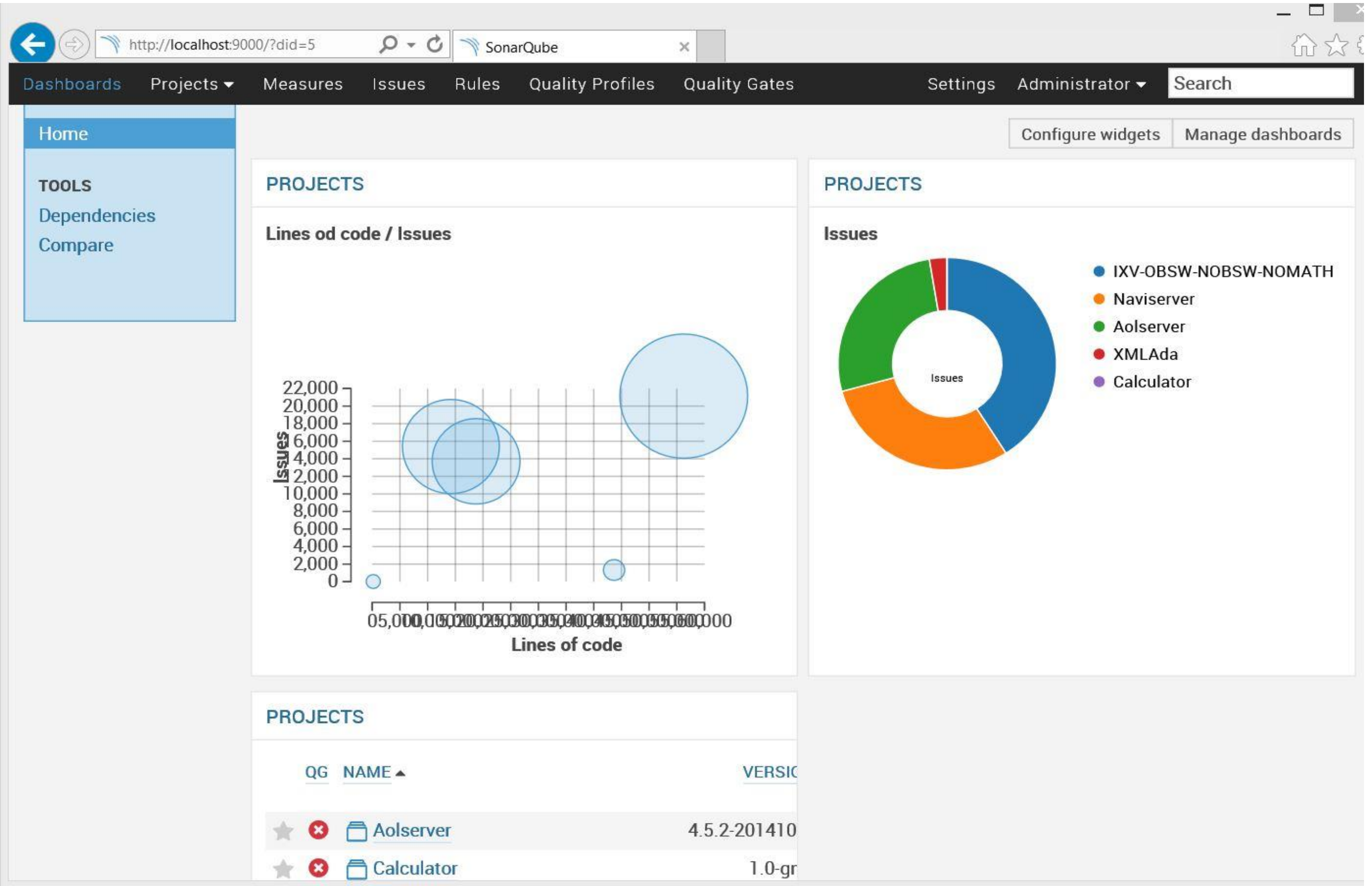
SonarQube – There’s more

- Analyses on the same code base can be performed at different moments in time and SonarQube keeps track of the changes/evolution.

- The problems found during analyses (a.k.a. issues) can be managed directly from within the system itself, e.g.
  - Identifying false positives
  - Assigning issues to developers
  - Checking their status (if they have been solved)
  - …

SonarQube - Screenshots

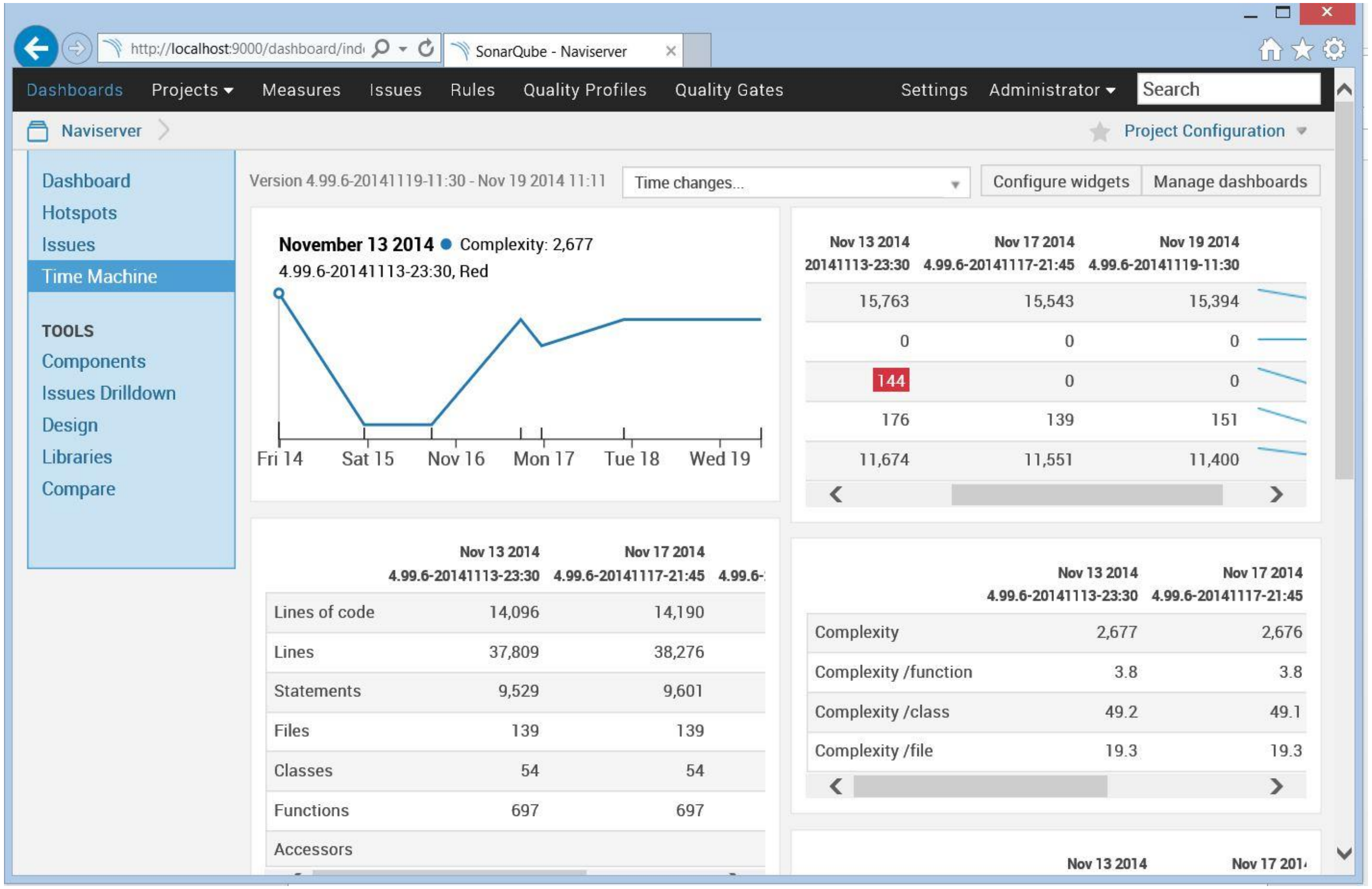
http://sonarsrv.spazioit.com/



QG	NAME ▲	VERSIC
★ ⊗	Aolserver	4.5.2-201410
★ ⊗	Calculator	1.0-gr

SonarQube - Screenshots

http://sonarsrv.spazioit.com/



Version 4.99.6-20141119-11:30 - Nov 19 2014 11:11

Nov 13 2014 20141113-23:30	Nov 17 2014 4.99.6-20141117-21:45	Nov 19 2014 4.99.6-20141119-11:30
15,763	15,543	15,394
0	0	0
144	0	0
176	139	151
11,674	11,551	11,400

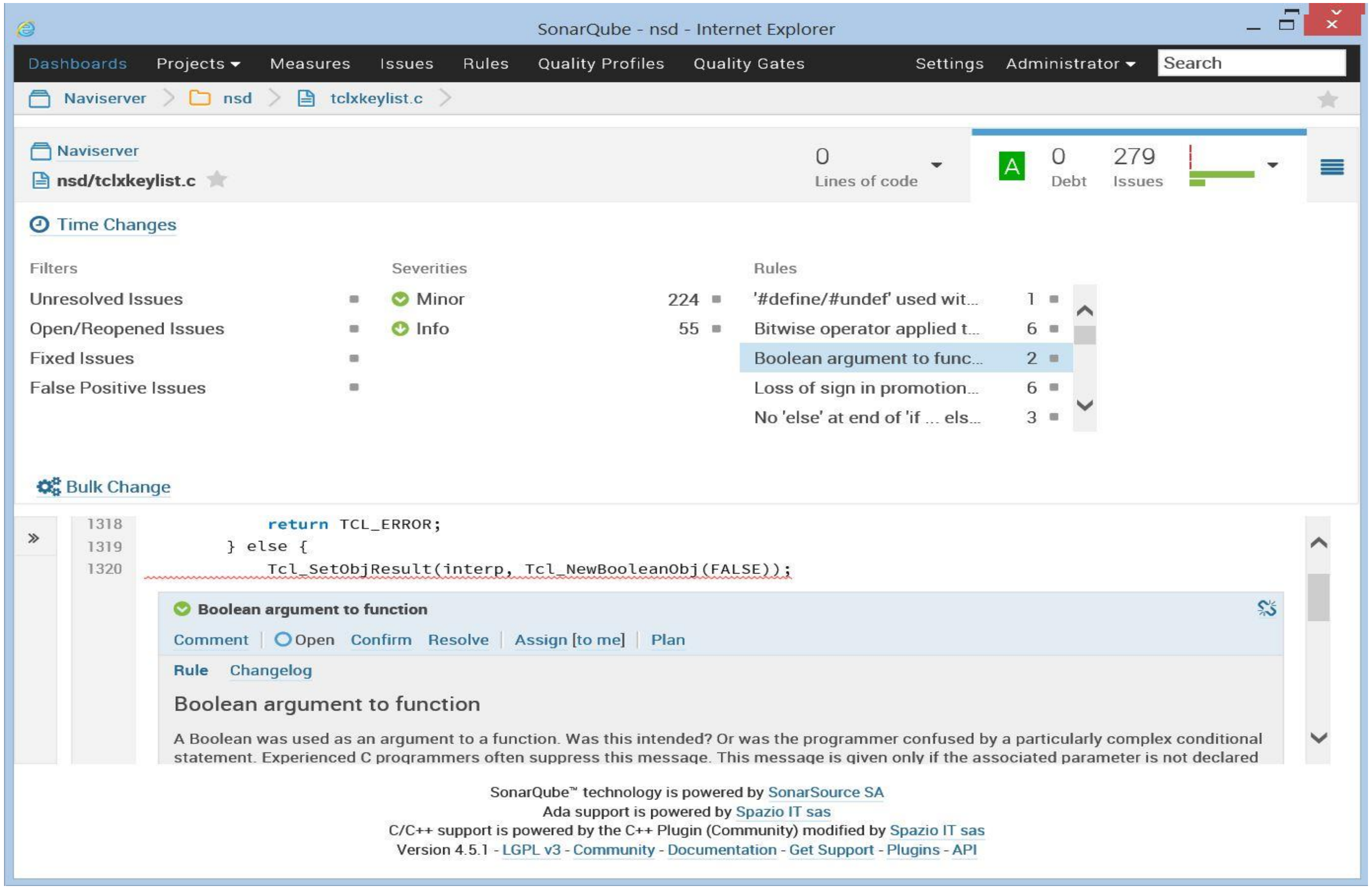
	Nov 13 2014 4.99.6-20141113-23:30	Nov 17 2014 4.99.6-20141117-21:45	4.99.6-
Lines of code	14,096	14,190	
Lines	37,809	38,276	
Statements	9,529	9,601	
Files	139	139	
Classes	54	54	
Functions	697	697	
Accessors			

	Nov 13 2014 4.99.6-20141113-23:30	Nov 17 2014 4.99.6-20141117-21:45
Complexity	2,677	2,676
Complexity /function	3.8	3.8
Complexity /class	49.2	49.1
Complexity /file	19.3	19.3

Nov 13 2014 Nov 17 201

SonarQube - Screenshots

http://sonarsrv.spazioit.com/



Naviserver
nsd/tclxkeylist.c

0 Lines of code

A 0 Debt 279 Issues

Time Changes

Filters		Severities		Rules	
Unresolved Issues		Minor	224	'#define/#undef' used wit...	1
Open/Reopened Issues		Info	55	Bitwise operator applied t...	6
Fixed Issues				Boolean argument to func...	2
False Positive Issues				Loss of sign in promotion...	6
				No 'else' at end of 'if ... els...	3

Bulk Change

```
            return TCL_ERROR;
        } else {
            Tcl_SetObjResult(interp, Tcl_NewBooleanObj(FALSE));
```

Boolean argument to function

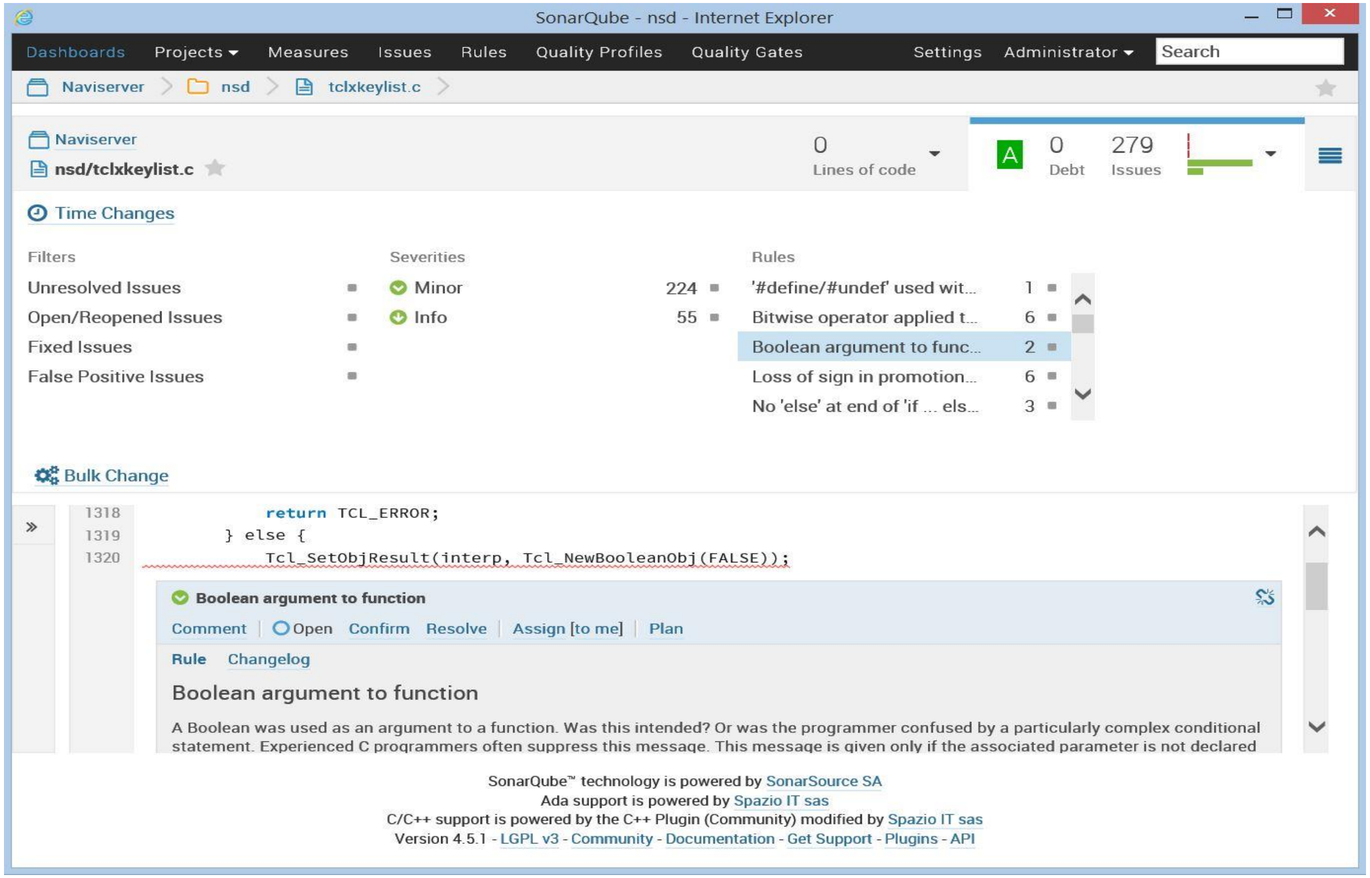
Boolean argument to function

A Boolean was used as an argument to a function. Was this intended? Or was the programmer confused by a particularly complex conditional statement. Experienced C programmers often suppress this message. This message is given only if the associated parameter is not declared

SonarQube™ technology is powered by SonarSource SA
Ada support is powered by Spazio IT sas
C/C++ support is powered by the C++ Plugin (Community) modified by Spazio IT sas
Version 4.5.1 - LGPL v3 - Community - Documentation - Get Support - Plugins - API

SonarQube - Screenshots

http://sonarsrv.spazioit.com/



Naviserver
nsd/tclxkeylist.c

0 Lines of code	A	0 Debt	279 Issues	

Time Changes

Filters		Severities		Rules	
Unresolved Issues		Minor	224	'#define/#undef' used wit...	1
Open/Reopened Issues		Info	55	Bitwise operator applied t...	6
Fixed Issues				Boolean argument to func...	2
False Positive Issues				Loss of sign in promotion...	6
				No 'else' at end of 'if ... els...	3

Bulk Change

```
            return TCL_ERROR;
        } else {
            Tcl_SetObjResult(interp, Tcl_NewBooleanObj(FALSE));
```

Boolean argument to function

Comment | ○ Open | Confirm | Resolve | Assign [to me] | Plan
Rule | Changelog

Boolean argument to function

A Boolean was used as an argument to a function. Was this intended? Or was the programmer confused by a particularly complex conditional statement. Experienced C programmers often suppress this message. This message is given only if the associated parameter is not declared

SonarQube™ technology is powered by SonarSource SA
Ada support is powered by Spazio IT sas
C/C++ support is powered by the C++ Plugin (Community) modified by Spazio IT sas
Version 4.5.1 - LGPL v3 - Community - Documentation - Get Support - Plugins - API

CppCheck

- “Cppcheck is a static analysis tool for C/C++ code

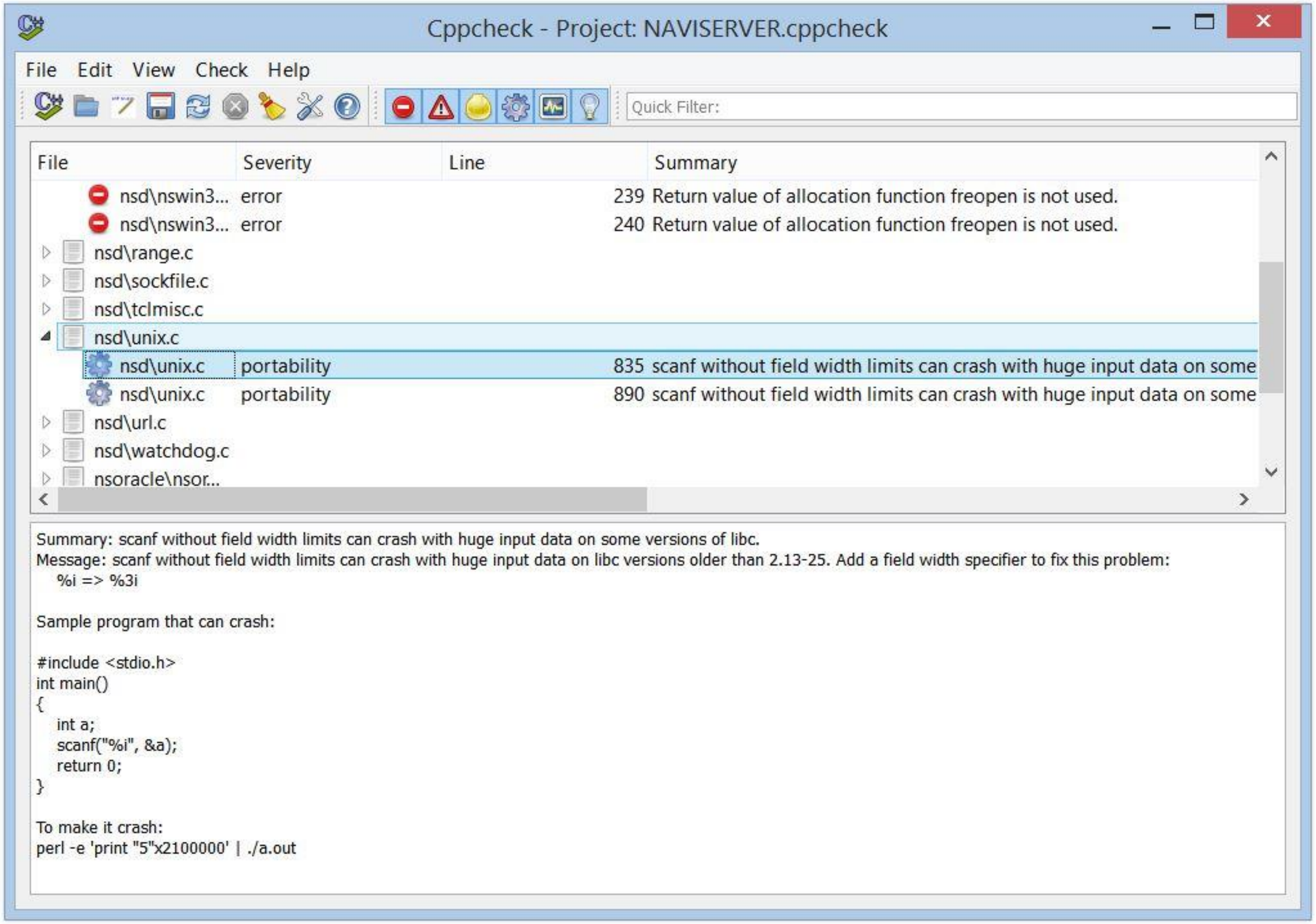
- Unlike C/C++ compilers and many other analysis tools it does not detect syntax errors in the code.

- Cppcheck primarily detects the types of bugs that the compilers normally do not detect (from CppCheck website http://cppcheck.sourceforge.net/)”

CppCheck

- CppCheck produces messages belonging to 6 different categories:
  1. “Error - used when bugs are found
  2. Warning - suggestions about defensive programming to prevent bugs
  3. Style - stylistic issues related to code clean-up (unused functions, redundant code, “constness”, and such)
  4. Portability - portability warnings. 64-bit portability. code might work different on different compilers. etc.
  5. Performance - suggestions for making the code faster. These suggestions are only based on common knowledge.
  6. Information - informational messages about checking problems.”

CppCheck


Cppcheck - Project: NAVISERVER.cppcheck
File Edit View Check Help
Quick Filter:

File	Severity	Line	Summary
nsd\nswin3...	error	239	Return value of allocation function freopen is not used.
nsd\nswin3...	error	240	Return value of allocation function freopen is not used.
nsd\range.c			
nsd\sockfile.c			
nsd\tclmisc.c			
nsd\unix.c			
nsd\unix.c	portability	835	scanf without field width limits can crash with huge input data on some
nsd\unix.c	portability	890	scanf without field width limits can crash with huge input data on some
nsd\url.c			
nsd\watchdog.c			
nsoracle\nsor...			

```
Summary: scanf without field width limits can crash with huge input data on some versions of libc.
Message: scanf without field width limits can crash with huge input data on libc versions older than 2.13-25. Add a field width specifier to fix this problem:
    %i => %3i

Sample program that can crash:

#include <stdio.h>
int main()
{
    int a;
    scanf("%i", &a);
    return 0;
}

To make it crash:
perl -e 'print "5"x2100000' | ./a.out
```

PC-Lint

- PC-Lint (http://www.gimpel.com) is a static analyzer deriving from the old Unix utility “lint”.

- It supports many checks:
  - Barr Group's Top 10 Bug-Killing Rules (http://www.barrgroup.com/webinars/10rules)
  - Dan Saks (http://www.dansaks.com/)
  - Scott Meyers C++ books
    - (More) Effective C++ - edition 1992
    - (More) Effective C++ - edition 1996
    - (More) Effective C++ - edition 1996

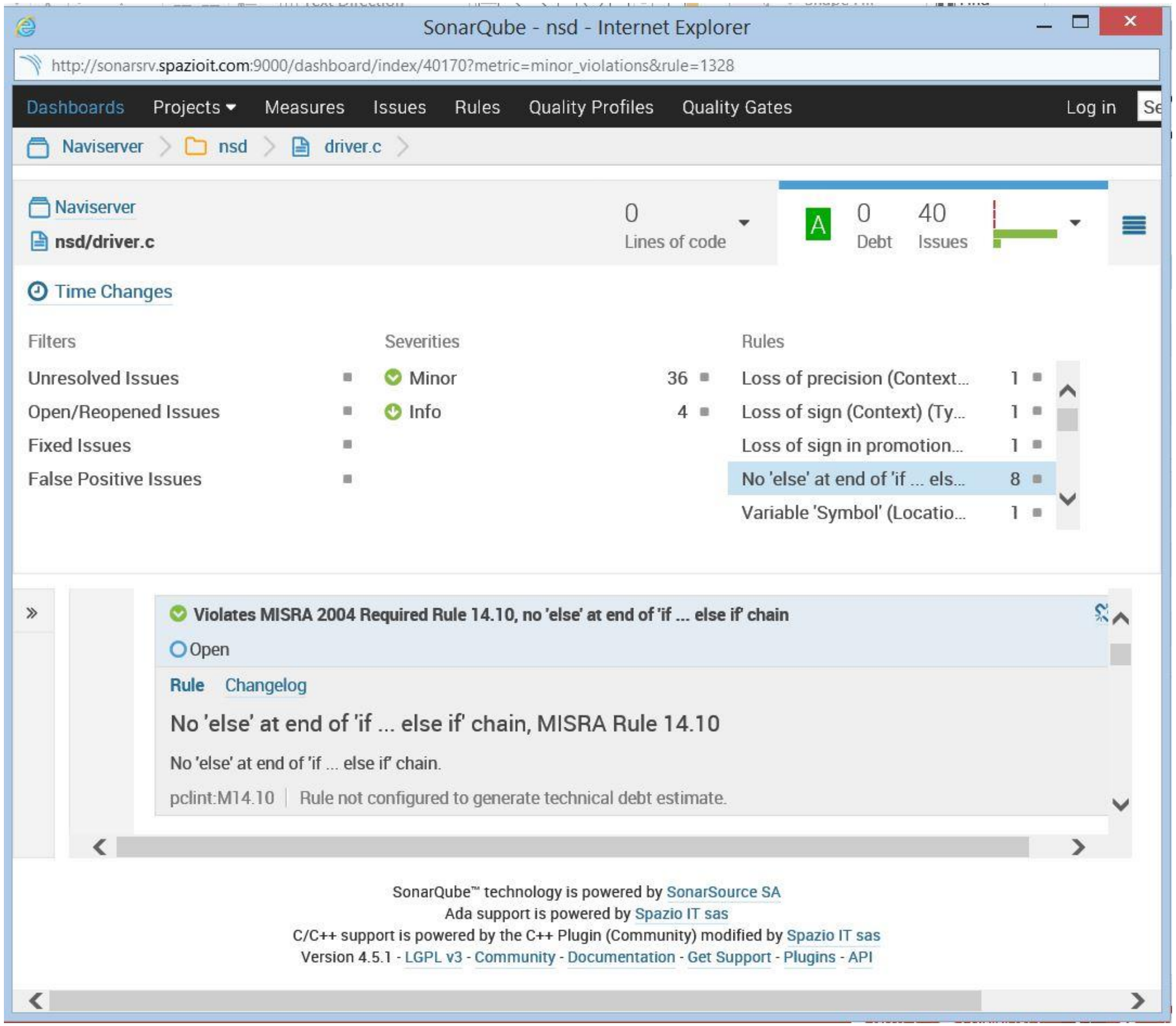
PC-Lint

- MISRA (http://www.misra.org.uk/)
  - Latest MISRA C (TM)
  - MISRA C 1998 (TM)
  - MISRA C 2004 (TM)
  - MISRA C 2012 (TM)
  - MISRA C++ 2008
  - MISRA C++ 2008 using 9000 level messages
- Barr’s / Netrino Embedded C Coding Standard (http://www.netrino.com/taxonomy/term/3)
- Porting from 32-bit to 64-bit
  - from 32-bit to LP64 model
  - from 32-bit to LLP64 model
  - from 32-bit to ILP64 model

PC-Lint

- In order to “reduce the noise”, it needs a very careful setup / initial configuration (e.g.)
  - the proper memory model
  - the C/C++ include files (libraries)
  - the behaviour of standard macros and pragmas like “assert”, “pragma pack”,
  - the set of active checks
  - …

PC-Lint


SonarQube - nsd - Internet Explorer
http://sonarsrv.spazioit.com:9000/dashboard/index/40170?metric=minor_violations&rule=1328
Dashboards
Projects
Measures
Issues
Rules
Quality Profiles
Quality Gates
Log in
Naviserver
nsd
driver.c

Naviserver

nsd/driver.c

0 Lines of code

A 0 Debt 40 Issues

Time Changes

Filters		Severities		Rules	
Unresolved Issues		Minor	36	Loss of precision (Context...	1
Open/Reopened Issues		Info	4	Loss of sign (Context) (Ty...	1
Fixed Issues				Loss of sign in promotion...	1
False Positive Issues				No 'else' at end of 'if ... els...	8
				Variable 'Symbol' (Locatio...	1

Violates MISRA 2004 Required Rule 14.10, no 'else' at end of 'if ... else if' chain

○ Open

Rule Changelog

No 'else' at end of 'if ... else if' chain, MISRA Rule 14.10

No 'else' at end of 'if ... else if' chain.

pclint:M14.10 | Rule not configured to generate technical debt estimate.

SonarQube™ technology is powered by SonarSource SA
Ada support is powered by Spazio IT sas
C/C++ support is powered by the C++ Plugin (Community) modified by Spazio IT sas
Version 4.5.1 - LGPL v3 - Community - Documentation - Get Support - Plugins - API

CBMC

- "CBMC is a Bounded Model Checker for C and C++ programs. It supports C89, C99, most of C11 and most compiler extensions provided by GCC and Visual Studio. (…)
- It allows verifying array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions.(..).
- The verification is performed by unwinding the loops in the program and passing the resulting equation to a decision procedure. (http://www.cprover.org/cbmc/)"

CBMC

- CBMC first converts a C program into a model, some kind of “symbolic executable”.
- Then the “symbolic executable” is “executed” and the execution generates “decision conditions” (about some property being true or false) that are expressed as CNF formulae.
- Finally these formulae are passed to an external SAT solver for their evaluation and verification (http://www.dwheeler.com/essays/minisat-user-guide.html).

Frama-C

- Frama-C (http://frama-c.com/), like SonarQube, rather than being a specific tool, is "is an extensible and collaborative platform dedicated to source-code analysis of C software."
- Frama-C relies on CIL (C Intermediate Language) to generate an abstract syntax tree. The abstract syntax tree supports annotations written in ANSI/ISO C Specification Language (ACSL).

Frama-C

- Like SonarQube also Frama-C has its own Plugins. Spazio IT has used two Frama-C Plugins:
  1. Value analysis (http://frama-c.com/value.html) - which computes a value or a set of possible values for each variable in a program. This plugin uses abstract interpretation techniques and many other plugins make use of its results.
  2. WP (Weakest Precondition - http://frama-c.com/wp.html) - to verify properties in a deductive manner

Methodology

Basic Core

Basic Core

- Identify which checks need to be executed on the code, i.e.
  - for the compiler, which compiler warnings (possibly all of them) need to be verified;
  - for CppCheck, which type pf messages (errors, warnings, performance messages, and so on) need to be verified
  - for PC-Lint, which rule sets have to be used (e.g. MISRA C 2004), and for each rule set, which actual rules make sense and need to be verified

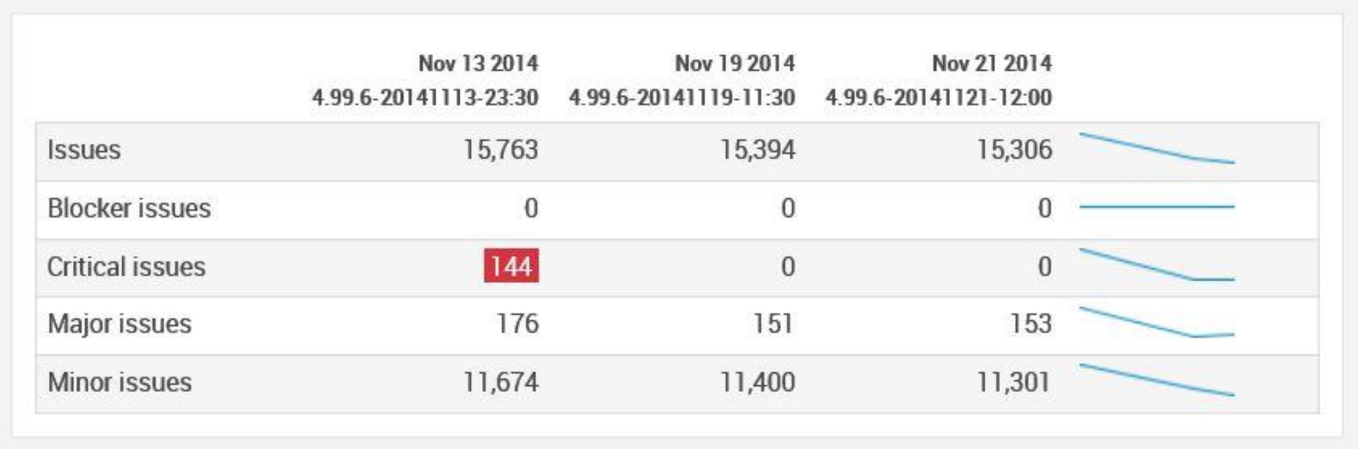
Basic Core

- Configure carefully the tools (in terms of tools options, selected memory model – e.g. LP64 vs. LLP64, location of the sources, location of the include files, and so on…
- Tune/optimize the configuration identified in point 1 by running few analysis sessions to verify that the proper information is generated (and disable the production of useless, noisy outputs – this may require the development of some filtering scripts).
- Run the analyses whenever it makes sense in the lifetime of a project (or during operations), and possibly on a regular basis.

Basic Core

- At every run the code:
  - should compile;
  - should compile without generating any of the selected warning;
  - should pass CppCheck analyses without generating any of the selected messages;
  - should pass PC-Lint analyses without violating any of the selected rules/guidelines.

Basic Core



	Nov 13 2014 4.99.6-20141113-23:30	Nov 19 2014 4.99.6-20141119-11:30	Nov 21 2014 4.99.6-20141121-12:00
Issues	15,763	15,394	15,306
Blocker issues	0	0	0
Critical issues	144	0	0
Major issues	176	151	153
Minor issues	11,674	11,400	11,301

- Gerard J. Holzmann , “Mars Code”, Communications of the ACM, Vol. 57 No. 2, Pages 64-73, 10.1145/2560217.2560218 (http://cacm.acm.org/magazines/2014/2/171689-mars-code/fulltext)

Bounded Model Checking
Abstract Interpretation

When looking carefully into the magic ball…

what we eventually see…

is us. ☺ ☺ ☺

Bounded Model Checking
Abstract Interpretation

- CBMC and Frama-C Plugins (Value Analysis and WP) organize their computation into two phases:
  - Generation of a model of the code under analysis
  - "Symbolic execution" or "logic verification" of the model itself.
- The computation resources required by phase one grow in a polynomial way with the complexity of code under analysis (number of files, packages, classes, functions, parameters, variables, lines of code, loops, constructs and so o…)
- The computation resources required by phase two grow exponentially with the complexity of the code under of analysis.

Bounded Model Checking
Abstract Interpretation

- So, for not so small, real code bases
  - either we stick to phase one
  - or we split the system under analysis into reasonable, «manageable» chunks.

Infinite Loops

- Never ending loop

```
#include <stdio.h>

int main() {
  int i = 0;
  int n = 10;

  for (i = 0; i < n; i++) {
    printf("Iteration #% 2d.\n", i + 1);
    if (i == 5) i = 0;
  }
  return 0;
}
```

Infinite Loops

- CBMC reaction (phase one)

```
…
Unwinding loop c::main.0 iteration 1205 file loops.c line 7
function main thread 0
Unwinding loop c::main.0 iteration 1206 file loops.c line 7
function main thread 0
Unwinding loop c::main.0 iteration 1207 file loops.c line 7
function main thread 0
Unwinding loop c::main.0 iteration 1208 file loops.c line 7
function main thread 0
Unwinding loop c::main.0 iteration 1209 file loops.c line 7
function main thread 0
…
```

Infinite Loops

- Frama-C reaction (phase two)

```
…
[value] Done for function printf
[value] computing for function printf <- main.
        Called from loops.c:8.
[value] Done for function printf
[value] computing for function printf <- main.
        Called from loops.c:8.
[value] Done for function printf
[value] Recording results for main
[value] done for function main
[value] ====== VALUES COMPUTED ======
[value] Values at end of function main:
  NON TERMINATING FUNCTION
…
```

Infinite Loops

- Both CBMC and Frama-C would detect this

```
#include <stdio.h>

int main() {
  int i = 0;
  int n = 10;
  // int *pn = &n;
  int *pn;

  for (i = 0; i < (*pn); i++) {
    printf("Iteration #% 2d.\n", i + 1);
  }
  return 0;
}
```

Loops Checking and Cyclic Tasks

- If a CBMC analysis is conducted on a set of files using as entry point a given function (say ‘foo’) and the analysis finishes without having to limit neither the “unwinding” nor the “depth”, then also the function ‘foo’ finishes, as well as any other function that is called by ‘foo’ (both directly or indirectly) and that belongs to the code under analysis.

Loops Checking and Cyclic Tasks

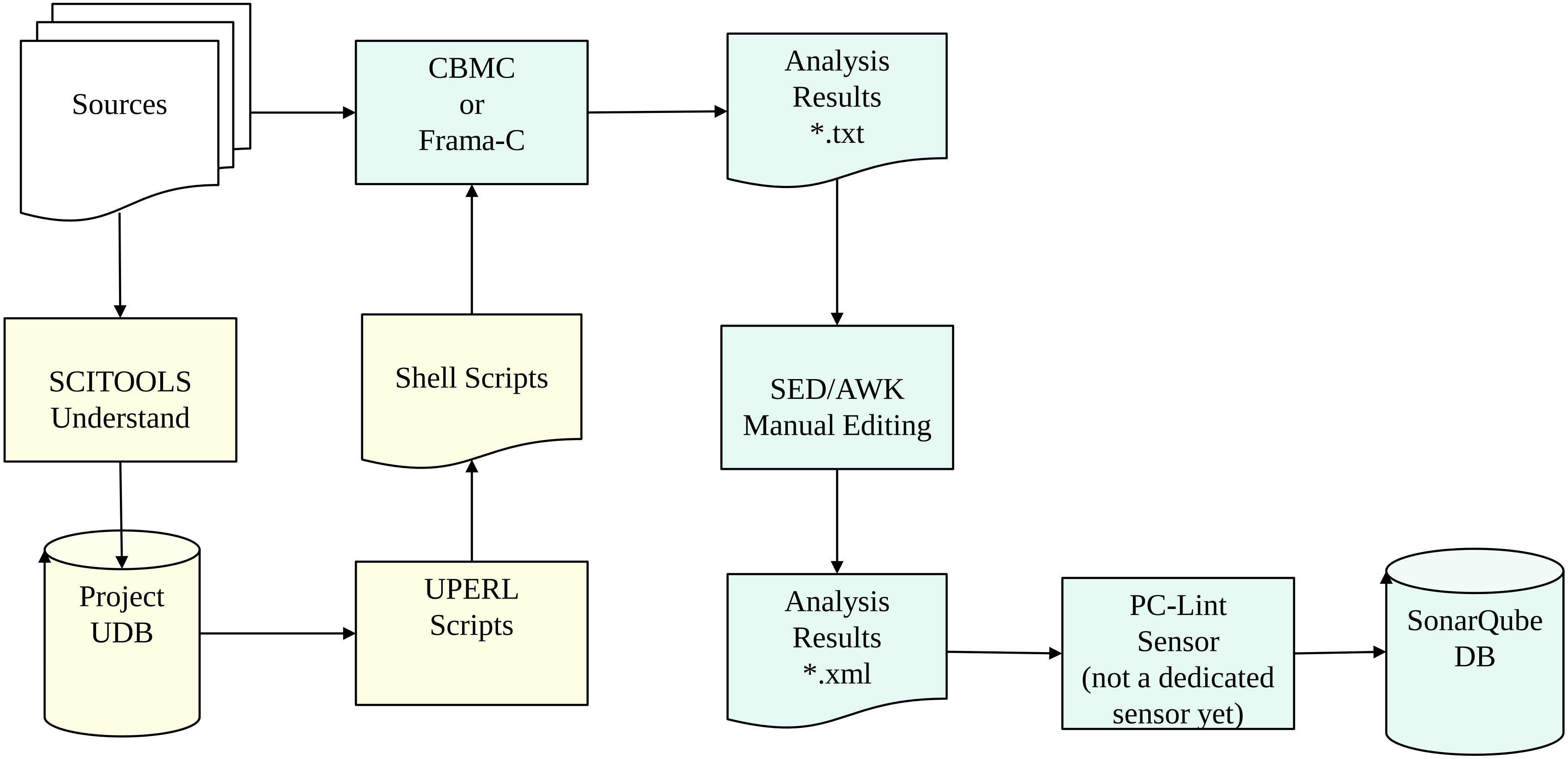
- Analysing the IXV OBSW it is possible to see that every cyclic task (cyclic thread) in the system is characterised by three functions:
  1. XXX_Init – initialise the data required by the task
  2. XXX_StartThread – start the thread
  3. XXX_Cycle – this is the function that gets called at every cycle.
- So, if the CBMC analyses of all the XXX_Cycle functions finish, then (at least in the cyclic tasks) there is no never-ending loop. And this is what was proved.

Manageable Chunks

- Acting locally (at function level) → Identifying Manageable Chunks

- Acting locally:
  - pointer checks
  - memory leak checks
  - signed/unsigned overflow
  - float overflow
  - …

Manageable Chunks



Manageable Chunks

- The output produced by CBMC and Frama-C is similar to the one produced by static analysers based on pattern matching (like PC-Lint), similar but not the same, rather complementary.

- Frama-C Value Analysis Plugin did not bring interesting results when used at local level, at function level.

Some Few Results Examples

IXV
OBSW
(made «generic»)

Some Few Results Examples

- Uninitialized Variables
  - Compiler (gcc, clang, Visual Studio)
  - PC-Lint

- Array Index out of bounds
  - PC-Lint in all code bases but only in simple cases
  - CBMC and Frama-C in all possible cases but in small portions of code

Some Few Results Examples

- Constant Value Boolean Expression (MISRA C 2004 Rule 13.7)

```
 void foo(unsigned int arg) {
  …
  if (arg < 0) {
    printf(“Error: input parameter can only be a positive
integer\n”);
    return;
  }
  …

  if (arg > UINT_MAX) …
```

Some Few Results Examples

```
case SListIdx:
       tablePtr = &itPtr->sets;
       if (tablePtr != NULL) {
 if (arg > UINT_MAX) …
```

  - PC-Lint

- Combining Signing and Unsigned Integers (MISRA C 2004 Rules 10.1, 10.3, 10.4)
  - PC-Lint

Some Few Results Examples

- Implicit integer type conversion (and promotion) (MISRA C 2004 10.1, 10.3, 10.4, 10.6, 10.7, 10.8)
  - PC-Lint

- Floating point comparison (MISRA C 2004 Rule 13.3)

```
double a;
double b;
if (a == b) …
```

Some Few Results Examples

```
bool_t My_DoubleEquals (double64_t first, double64_t second)
{
  bool_t isEqual = FALSE;
  double64_t difference = My_AbsValue(first - second);
  if (difference >= T_EPSILON)
    {
      isEqual = FALSE;
    }
  else
    {
      isEqual = TRUE;
    }
  return isEqual;
}
```

- PC-Lint

Some Few Results Examples

- Problems with pointers

```
#include <stdio.h>
/* Why is this wrong? */
int main(void) {
  int x, *p;
  x = 10;
  *p = x;
  printf("*p = %d.\n", *p);
  return 0;
}
```

  - PC-Lint
  - CBMC / Frama-C

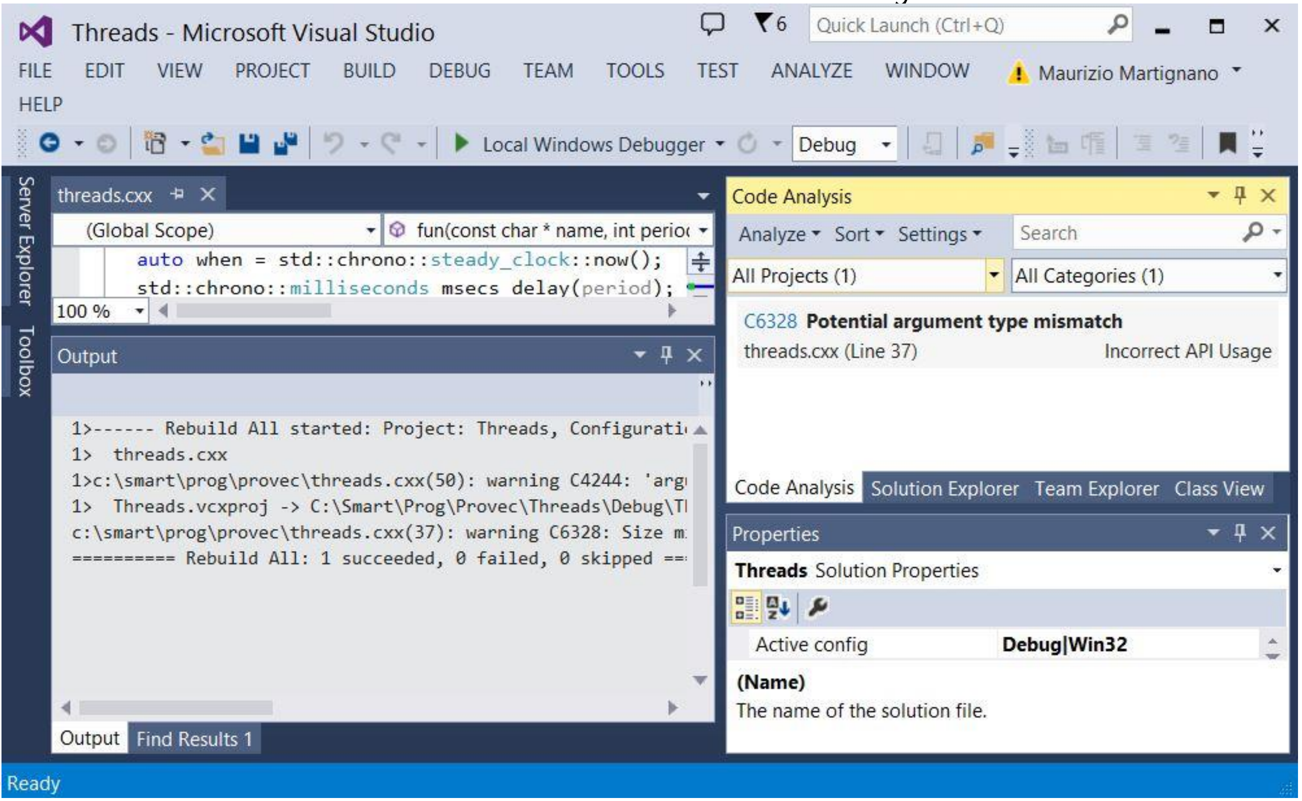
Some Few Results Examples

- Divisions by Zero / Overflows
  - PC-Lint
  - CBMC / Frama-C
  - **Traps** ←

Processes

Who does what?

- All nowadays Integrated Development Environments (IDEs) like GNAT GPS 2014, Visual Studio 2013, Eclipse Luna, offer some form of Code Analysis.



```
auto when = std::chrono::steady_clock::now();
std::chrono::milliseconds msecs delay(period);
```

```
1>------ Rebuild All started: Project: Threads, Configurati
1>  threads.cxx
1>c:\smart\prog\provec\threads.cxx(50): warning C4244: 'arg
1>  Threads.vcxproj -> C:\Smart\Prog\Provec\Threads\Debug\T
c:\smart\prog\provec\threads.cxx(37): warning C6328: Size m
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==
```

C6328 **Potential argument type mismatch**
threads.cxx (Line 37) Incorrect API Usage

Threads Solution Properties

Active config	**Debug|Win32**

(Name)
The name of the solution file.

Who does what?

- IDE’s analysis tools are to be used by software developers during their everyday work.

- SonarQube analyses are more for the «quality people» and they are not supposed to be executed everyday, but rather at specific /well defined moments in the software development life cycle.

When?

- SonarQube analyses should be performed after any «significant» delivery in a software development project, e.g. using ECSS 40 terminology, at:
  - CDR
  - QR
  - AR
- In maintenance projects SonarQube analyses should be performed after any «significant» new delivery, e.g. supposing a versioning like:
  major.minor[.build[.revision]]
  After every «minor» delivery.

AIRBUS Helicopters

AIRBUS Helicopters

- Since mid 2012 Spazio IT has been working for AIRBUS Helicopters and has developed an Ada Plugin supporting both:
  - Adacore GNAT (http://www.adacore.com)
  - Atego APEX Ada (http://www.atego.com)

  compilation tools chains

- Spazio IT platform has been adopted by the group maintaining the software of the NH90 and Tiger helicopters.

frama C
Software Analyzers
CBMC
PC-Lint
C/C++
Static
Analysis

Ada vs. C/C++

Ada
C/C++

Ada vs. C/C++

- The majority if not all the problems mentioned earlier would never occur if Ada is used.

- In safety critical applications, the additional costs deriving by the adoption of Ada are partly compensated by the savings gained when performing Verification & Validation activities.

- What is the added value of using a code quality platform like the one developed by Spazio IT in the case of Ada?

- The answer is: METRICS.

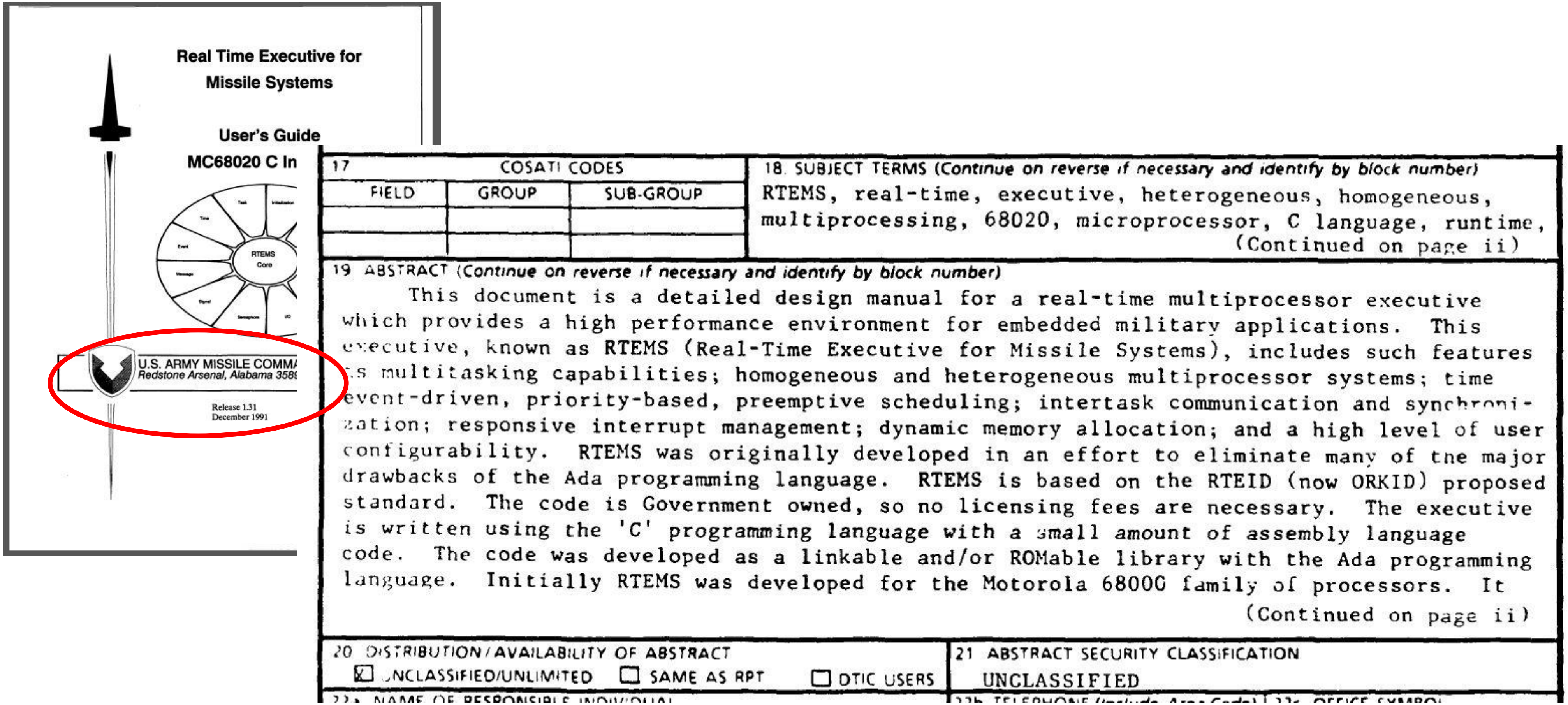
Ada vs. C/C++

- Metrics like the lines of code, the % of comments, the cyclomatic complexity, the nesting and so on… are all correlated somehow to the readability and maintainability of the code.

- Being able to "see" these metrics in the context of the code allows developers and maintainers to immediately identify "host-spots", that is portions of code requiring attention.

- On top of that, the "time-machine" of SonarQube allows checking the evolution of these "hot-spots" with time.

Ada vs C/C++: origins and explanations



Real Time Executive for
Missile Systems

User's Guide
MC68020 C In

RTEMS
Core
Task
Initialization
Time
Event
Message
Signal
Semaphore
I/O

U.S. ARMY MISSILE COMMA
Redstone Arsenal, Alabama 358

Release 1.31
December 1991

17 COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	RTEMS, real-time, executive, heterogeneous, homogeneous, multiprocessing, 68020, microprocessor, C language, runtime,
			(Continued on page ii)

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This document is a detailed design manual for a real-time multiprocessor executive which provides a high performance environment for embedded military applications. This executive, known as RTEMS (Real-Time Executive for Missile Systems), includes such features as multitasking capabilities; homogeneous and heterogeneous multiprocessor systems; time event-driven, priority-based, preemptive scheduling; intertask communication and synchronization; responsive interrupt management; dynamic memory allocation; and a high level of user configurability. RTEMS was originally developed in an effort to eliminate many of the major drawbacks of the Ada programming language. RTEMS is based on the RTEID (now ORKID) proposed standard. The code is Government owned, so no licensing fees are necessary. The executive is written using the 'C' programming language with a small amount of assembly language code. The code was developed as a linkable and/or ROMable library with the Ada programming language. Initially RTEMS was developed for the Motorola 68000 family of processors. It

(Continued on page ii)

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) 22c OFFICE SYMBOL

Future Work

Future Work

- Continuous Integration
  - Analyses should be run in an automatic way (e.g. via integration with systems like Jenkins - http://jenkins-ci.org/)
- More research on Abstract Interpretation
  - E.g. Clang Static Analyzer (http://clang-analyzer.llvm.org/ a working example is available at http://sonarsrv.spazioit.com:8181/) vs. MathWorks Polyspace (http://www.mathworks.com/products/polyspace/)
- Education
  - C Awareness Campaign
- Code Quality Competence Centre
  - Training, Services, Platforms

Thank you for your time!

Software

Spazio IT