Hands On With the C/C++ IDE

Severin Gehwolf, Jeff Johnston,
Bernhard Merkle, Andrew Overholt

Hands On With The C/C++ IDE

In this tutorial, attendees will be led through focussed examples that illustrate how to effectively use the C/C++ IDE.

A set of C/C++ projects will show users how to take advantage of the CDT to develop, build, debug, test, and profile their code within Eclipse.

Virtual Images: VirtualBox/VM-Ware with Fetora14

Easy Tutorial Setup: Use Virtual Images:

- HIGHLY RECOMMENDED: ready to go
- 4GB Virtual Box Image File
- Fedora14 preinstalled with Eclipse CDT Linux Tools
- Available for
  - Oracle Virtual Box
  - VM-Ware: VM-Ware Workstation/Player

Setup CDT on Windows → Wascana (Doug Schaefer)

Setup on OS X ? (better use our Virtual Box Image)

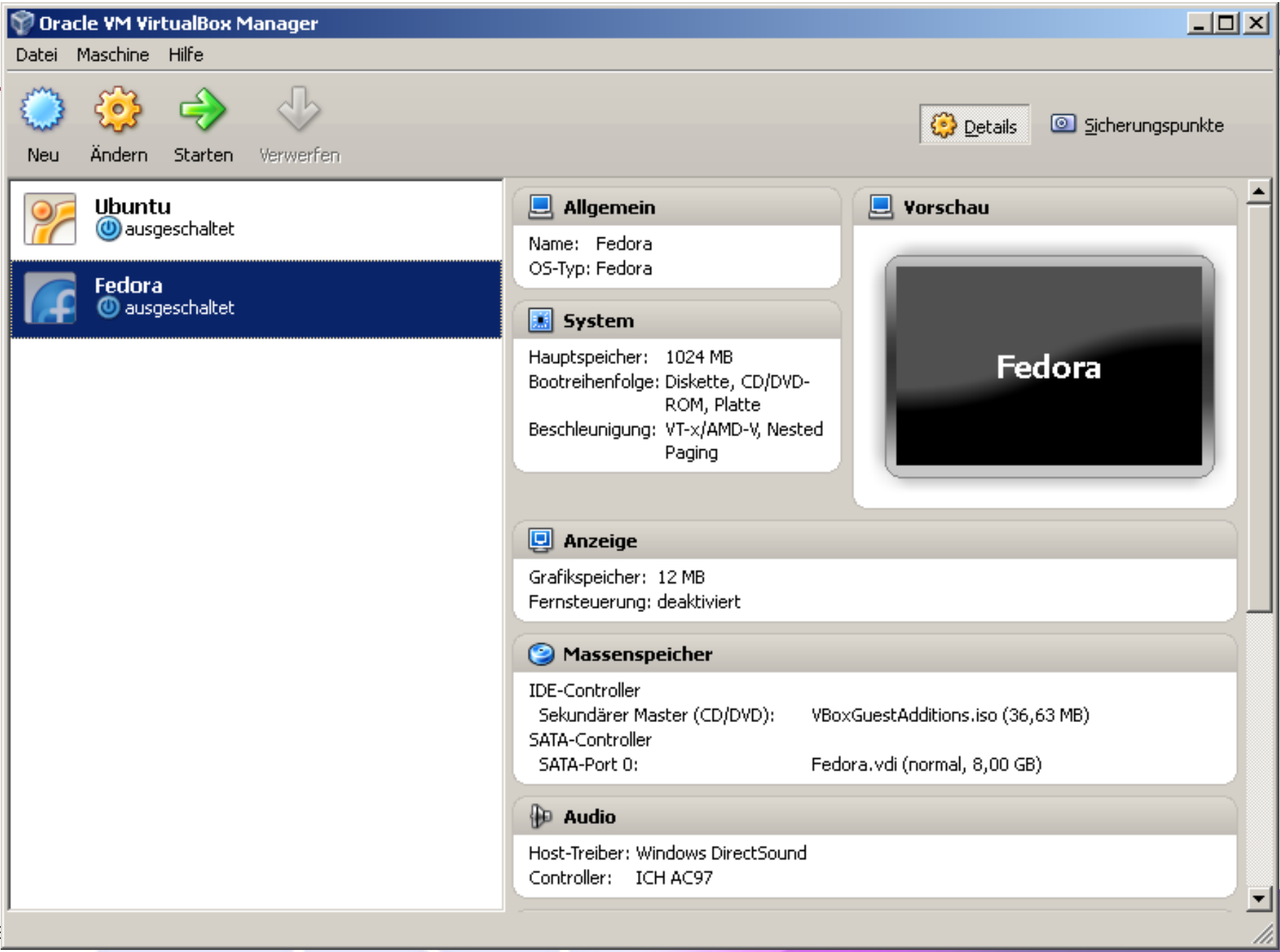
Tutorial Setup 101

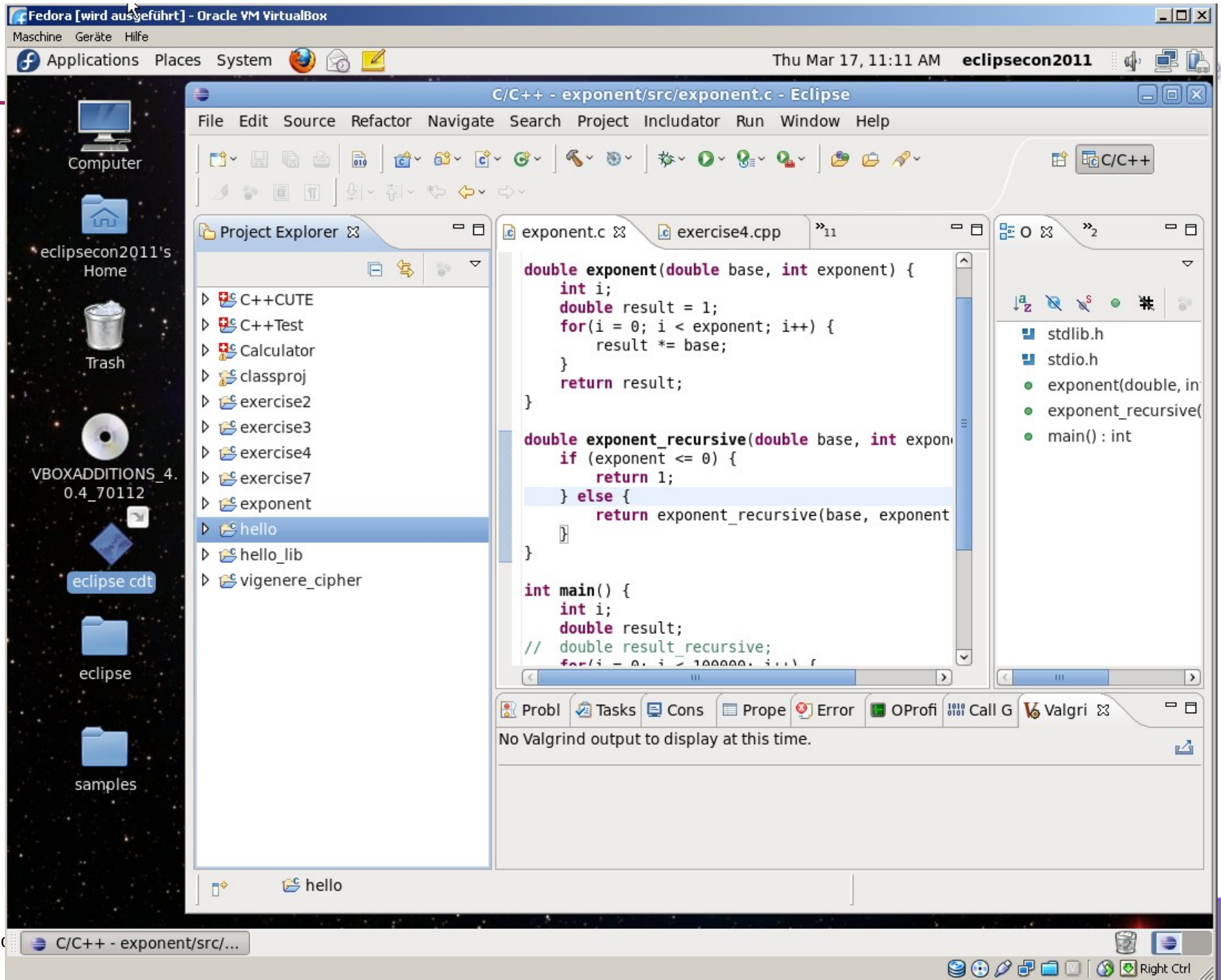
Copy **`VirtualBox-Image`** somewhere on HardDrive (4GB)

Install **`VirtualBox-Installer`** (for your OS)
We have Installers for Windows, MacOS, Linux, AMD/Intel

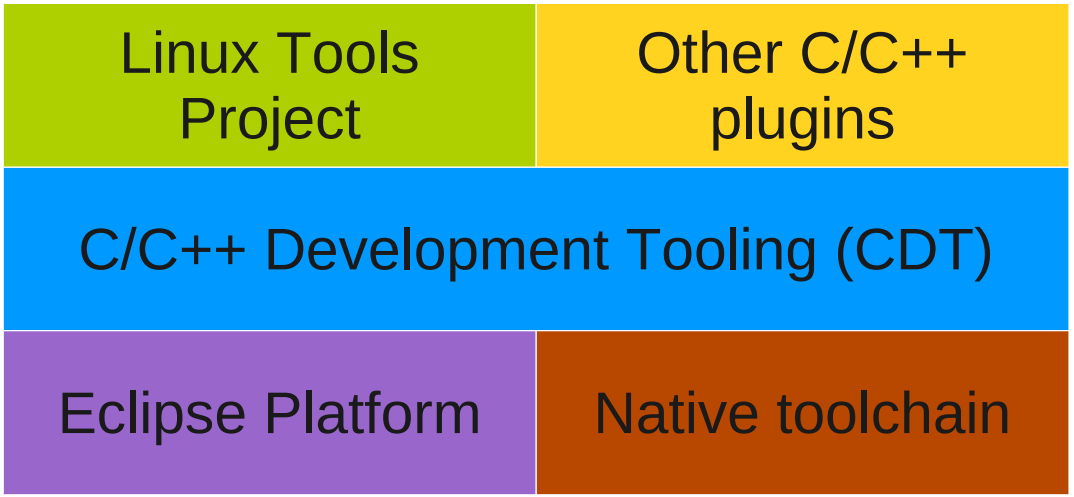
Startup VirtualBox

- Machine→Add… (Ctrl-A)
- Select Fedora.vbox (Copied in Step1)
- Startup the “Fedora” Virtual Machine
- Should log in automatically, but if not:
  - User “eclipsecon2011”, Password “eclipsecon2011”


Oracle VM VirtualBox Manager
Datei Maschine Hilfe
Neu
Ändern
Starten
Verwerfen
Details
Sicherungspunkte
Ubuntu
ausgeschaltet
Fedora
ausgeschaltet
Allgemein
Name: Fedora
OS-Typ: Fedora
System
Hauptspeicher: 1024 MB
Bootreihenfolge: Diskette, CD/DVD-ROM, Platte
Beschleunigung: VT-x/AMD-V, Nested Paging
Vorschau
Fedora
Anzeige
Grafikspeicher: 12 MB
Fernsteuerung: deaktiviert
Massenspeicher
IDE-Controller
Sekundärer Master (CD/DVD): VBoxGuestAdditions.iso (36,63 MB)
SATA-Controller
SATA-Port 0: Fedora.vdi (normal, 8,00 GB)
Audio
Host-Treiber: Windows DirectSound
Controller: ICH AC97


Fedora [wird ausgeführt] - Oracle VM VirtualBox
Maschine Geräte Hilfe
Applications Places System
Thu Mar 17, 11:11 AM eclipsecon2011
Computer
eclipsecon2011's Home
Trash
VBOXADDITIONS_4.0.4_70112
eclipse cdt
eclipse
samples
C/C++ - exponent/src/exponent.c - Eclipse
File Edit Source Refactor Navigate Search Project Includator Run Window Help
C/C++
Project Explorer
C++CUTE
C++Test
Calculator
classproj
exercise2
exercise3
exercise4
exercise7
exponent
hello
hello_lib
vigenere_cipher
exponent.c
exercise4.cpp
double exponent(double base, int exponent) {
int i;
double result = 1;
for(i = 0; i < exponent; i++) {
result *= base;
}
return result;
}
double exponent_recursive(double base, int expon
if (exponent <= 0) {
return 1;
} else {
return exponent_recursive(base, exponent
}
}
int main() {
int i;
double result;
// double result_recursive;
stdlib.h
stdio.h
exponent(double, in
exponent_recursive(
main() : int
Probl Tasks Cons Prope Error OProfi Call G Valgri
No Valgrind output to display at this time.
hello
C/C++ - exponent/src/...
Right Ctrl

Glossary and architecture



Exercises

- Discovering and fixing source code errors
- Configuring the build
- Working with breakpoints and data available while debugging
- Finding memory usage problems
- Tracking down performance bottlenecks
- Performing refactorings
- Integration with UnitTests
- Finding bugs and errors with static analysis

Overview

Test Driven Development

- Eclipse plugins for TDD: CUTE
- Implementing an example

Static Analysis (SA)

- 3 rules of Scott Meyers “Effective C++ 2nd” (Item 3, 11, 14)
- Tools for SA:
  - Lint, gcc –weffc++
- Eclipse plugins for SA:
  - Codan
  - Linticator
  - Includator

Overview

Test Driven Development

- Eclipse plugins for TDD: CUTE
- Implementing an example

Static Analysis (SA)

- 3 rules of Scott Meyers “Effective C++ 2nd” (Item 3, 11, 14)
- Tools for SA:
  - Lint, gcc –weffc++
- Eclipse plugins for SA:
  - Codan
  - Linticator
  - Includator

Project of IFS in Rapperswil, CH

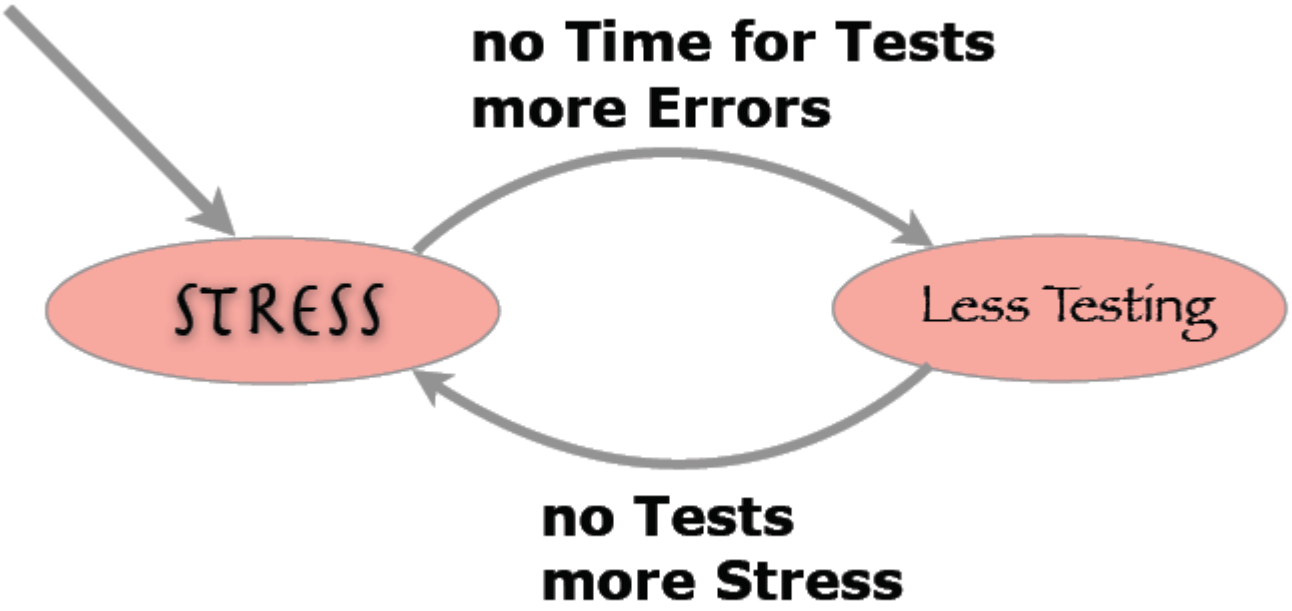
- http://www.cute-test.com

Features

- “The JUnit for C/C++ Programmers”
- CUTE = C(++) Unit Testing Easy

- Wizards to initialize and set up new tests
- Test navigator with green/red bar
- Diff-viewer for failing tests

Vicious Circle: Testing – Stress



Help:

- **Write test FIRST !**
- **Automate tests**
- **Run them often**

Unit-Testing 101

Test anything that **might** break

Test everything that **does** break

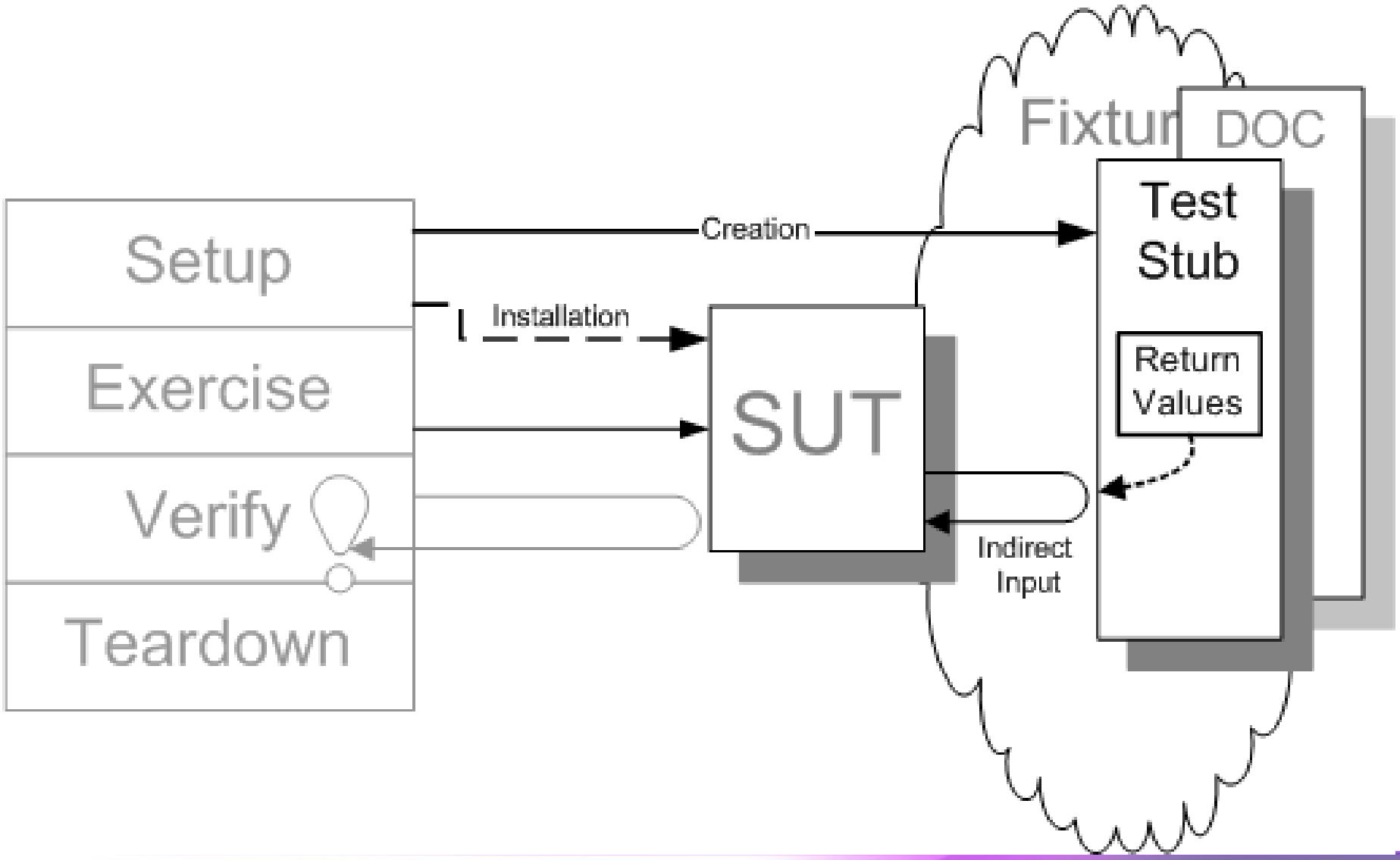
New code is guilty until proven innocent

Write at least as much test code as production code

Run local tests with each compile

Run all tests before check-in to repository

Structure of a typical Unit Testing Framework



Structure of a typical Unit Testing Framework

Test Assertion / Check statement

- used in

Test (Member-)Function

- defined in

TestCase Subclass bundling Tests

- its objects contained in

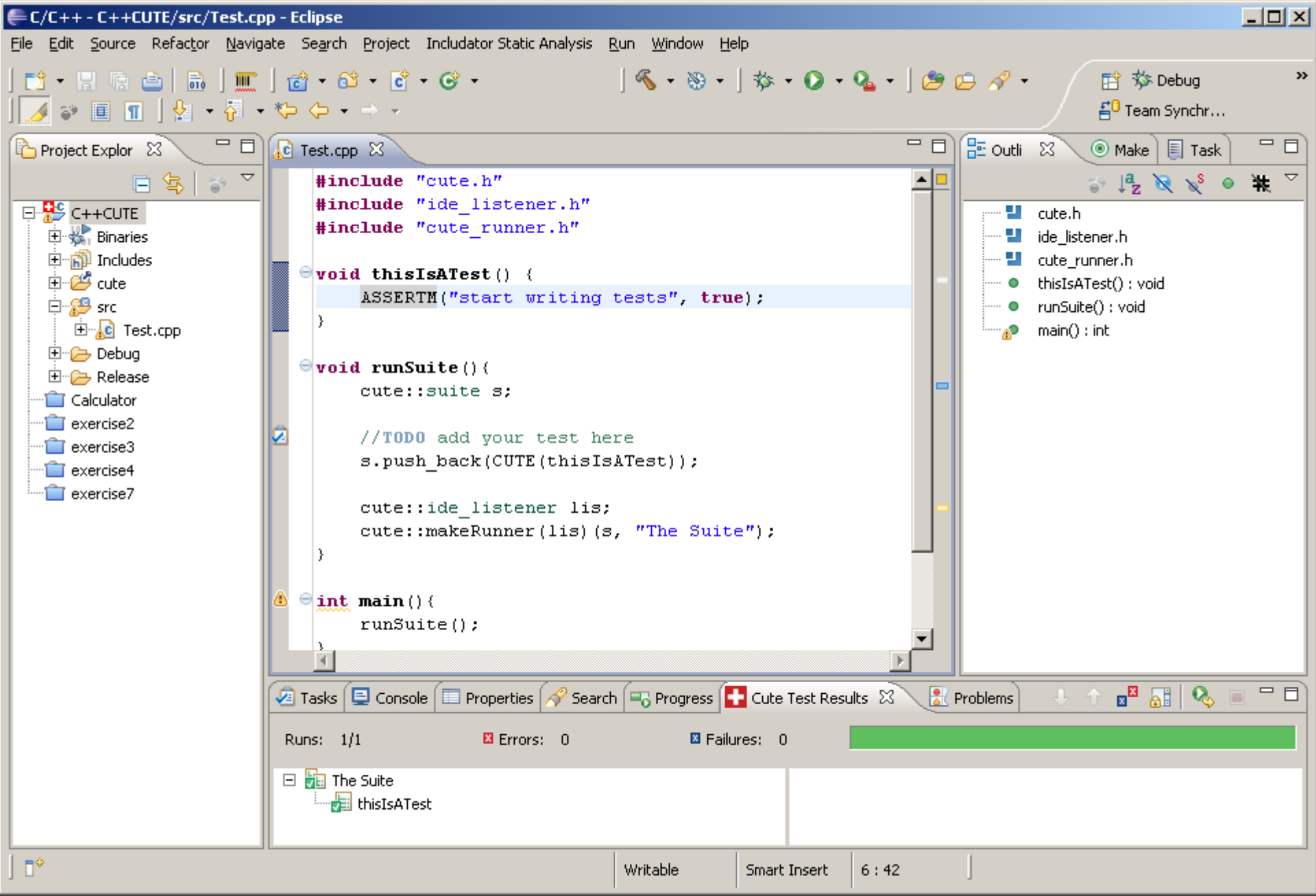
Test Suite collecting test objects

- executed by

Test Runner (often in a main() function)

- delivers result

OK or Failure


C/C++ - C++CUTE/src/Test.cpp - Eclipse
File Edit Source Refactor Navigate Search Project Includator Static Analysis Run Window Help
Debug
Team Synchr...
Project Explor
C++CUTE
Binaries
Includes
cute
src
Test.cpp
Debug
Release
Calculator
exercise2
exercise3
exercise4
exercise7
Test.cpp
#include "cute.h"
#include "ide_listener.h"
#include "cute_runner.h"
void thisIsATest() {
ASSERTM("start writing tests", true);
}
void runSuite(){
cute::suite s;
//TODO add your test here
s.push_back(CUTE(thisIsATest));
cute::ide_listener lis;
cute::makeRunner(lis)(s, "The Suite");
}
int main(){
runSuite();
Outli
Make
Task
cute.h
ide_listener.h
cute_runner.h
thisIsATest() : void
runSuite() : void
main() : int
Tasks
Console
Properties
Search
Progress
Cute Test Results
Problems
Runs: 1/1
Errors: 0
Failures: 0
The Suite
thisIsATest
Writable
Smart Insert
6 : 42

CUTE: Test Fixtures

```
#include "cute.h"
#include "cute_equals.h"

#include "CircularBuffer.h" // if you have this class separate

struct ATest {
  CircularBuffer<int> buf; // SUT == System Under Test

  ATest():buf(4){}
  void testEmpty(){ ASSERT(buf.empty());}
  void testNotFull(){ ASSERT(!buf.full());}
  void testSizeZero(){ ASSERT_EQUAL(0,buf.size());}
};

#include "cute_testmember.h"
....
s.push_back(CUTE_SMEMFUN(ATest,testEmpty));
s.push_back(CUTE_SMEMFUN(ATest,testNotFull));
s.push_back(CUTE_SMEMFUN(ATest,testSizeZero));
...
```

Using CUTE: it IS EASY !!! ☺

```
#include "cute.h"
```

`ASSERT(condition);`

- fails if condition is false

`ASSERT_EQUAL(expected,actual);`

- fails if exected is not equal to actual

add a message by appending M

- `ASSERTM(msg,condition)`
- `ASSERT_EQUALM(msg,exp,act)`

`FAIL(); FAILM(msg)`

- fails always, use to mark unwritten tests
- or for checking exceptions

My first CUTE Test

Create new C++ CUTE project

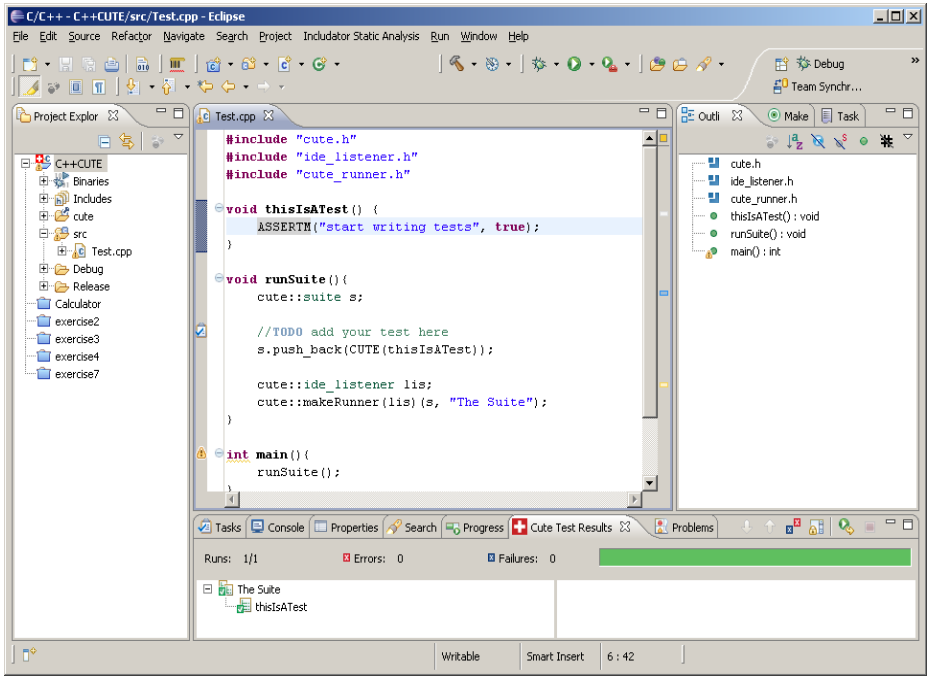
- In Project Explorer
  - New Project
  - C++ Project
  - CUTE Project
  - give project name

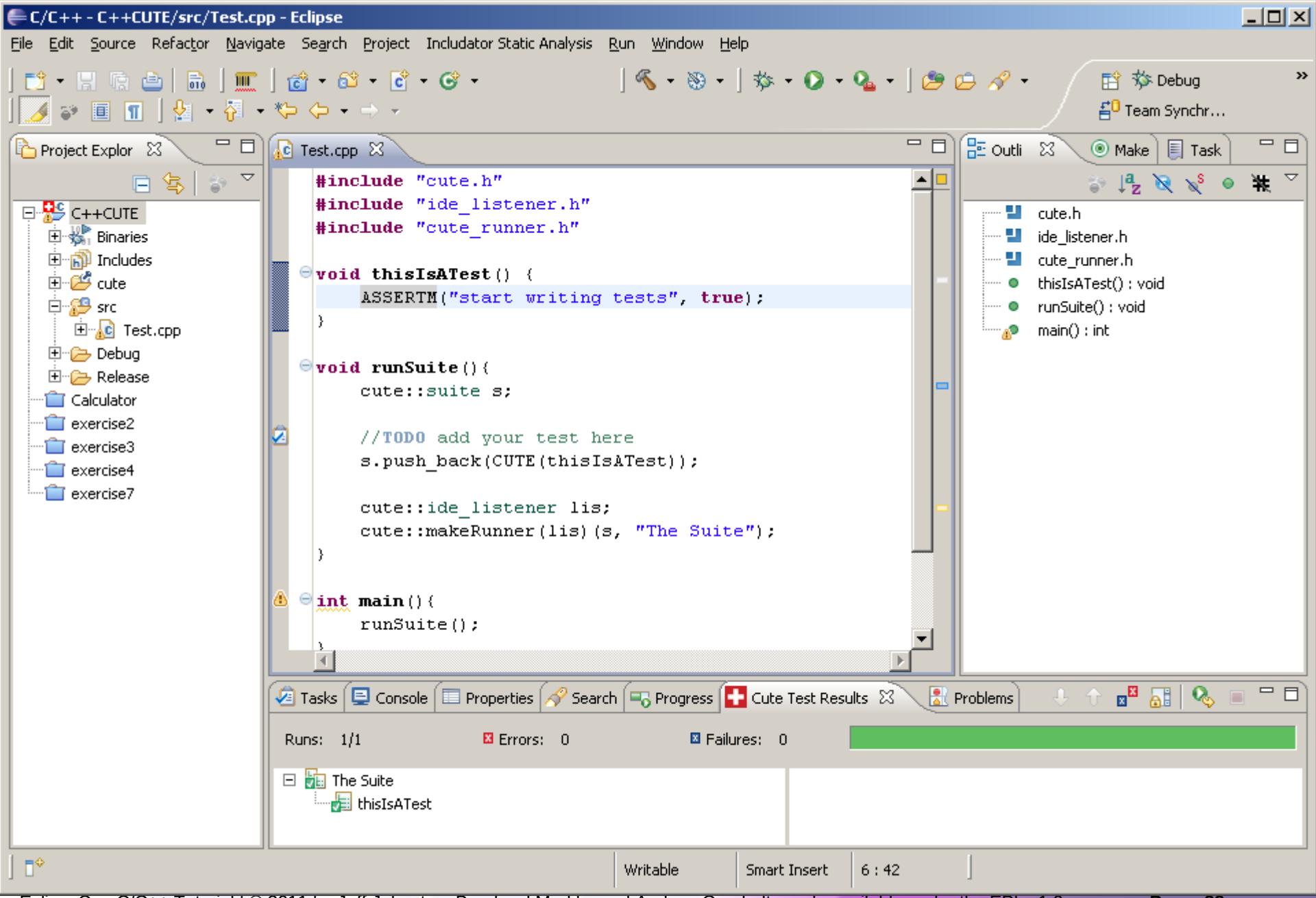
Let the project compile

Run binary as a CUTE Test

- Observe Result in CUTE
- Results Tab and Console
- Navigate to the failing test

Fix the Test and observe




C/C++ - C++CUTE/src/Test.cpp - Eclipse
File Edit Source Refactor Navigate Search Project Includator Static Analysis Run Window Help
Debug
Team Synchr...
Project Explor
C++CUTE
Binaries
Includes
cute
src
Test.cpp
Debug
Release
Calculator
exercise2
exercise3
exercise4
exercise7
Test.cpp
#include "cute.h"
#include "ide_listener.h"
#include "cute_runner.h"
void thisIsATest() {
ASSERTM("start writing tests", true);
}
void runSuite(){
cute::suite s;
//TODO add your test here
s.push_back(CUTE(thisIsATest));
cute::ide_listener lis;
cute::makeRunner(lis)(s, "The Suite");
}
int main(){
runSuite();
Outli
Make
Task
cute.h
ide_listener.h
cute_runner.h
thisIsATest() : void
runSuite() : void
main() : int
Tasks
Console
Properties
Search
Progress
Cute Test Results
Problems
Runs: 1/1
Errors: 0
Failures: 0
The Suite
thisIsATest
Writable
Smart Insert
6 : 42

Collecting multiple Tests

CUTE collects test objects in `cute::test_suite`

- this is just a `std::vector<cute::test>`

add your tests to your test suite

- `s.push_back(CUTE(testfunction));`
- `s.push_back(testfunctor());`

An overloaded operator+= could ease syntax:

- `s += CUTE(testfunction);`
- `s += testfunctor();`

CUTE: Example with expected Exception

– Test in C++

```
void testAnException() {
  std::vector<int> v; // arrange
  try {
    v.at(0); // act
    FAILM("expected out_of_range exception"); // assert
  }
  catch(std::out_of_range &) { }
}
```

– CUTE Version

```
void testAnException() {
  std::vector<int> v;
  ASSERT_THROWS(v.at(0),std::out_of_range);
}
```

Member Functions asTests in CUTE

`CUTE_SMEMFUN(TestClass,memfun)`

- instantiates a new object of TestClass and calls memfun on it ("simple“ member function)

`CUTE_MEMFUN(testobject,TestClass,memfun)`

- uses pre-instantiated testobject as target for memfun
  - this is kept by reference, take care of its scoping/lifetime
  - allows reuse of testobject for several tests and thus of a fixture provided by it.
  - allows for classes with complex constructor parameters

`CUTE_CONTEXT_MEMFUN(context,TestClass,memfun)`

- keeps a copy of context object and passes it to TestClass' constructor before calling memfun on it
  - avoids scoping problems
  - allows single-parameter constructors

TDD Example

- Start with a TEST FIRST !!!
- See Requirements R1…R4 for more details
- Requirement Priorities
  - High (++):
    must be completed to reach minimum usable subset
  - Medium (+):
    useful and should have, but could in principle live without
  - Low :
    optional, nice to have but definitely not essential

RE1 (++): Creation and Output of Strings

Objective

- Allow to create a string with a initial or a default value
- Allow to print its value on the console
- Allow to print the length of the string value

Details:

- String s1();
- String s2(“Hello world”);
- s1.print() results in “”
- s2.print () results in “Hello world”);
- s1.length() == 0;
- s2.length() == 11;

RE2 (+): Common String operations

Objective

- Allow common string manipulations, e.g. toUpper(), toLower(), trim()

Details

- String e("EclipseCon");
- e.toUpper() → ECLIPSECON
- e.toLower() → eclipsecon
- e.trim() → EclipseCon

RE3 (++): Support assignment, concatenation etc

Objective

- Extend with additional important convenience operations

Details

- String s1(“one”), String s2(“twenty”);
- s1 = s2; // results in s1 == “twenty”

- String s3 = s2 + s1; // results in ☺ S3 == “twentyone”

RE4 (): Additional operations

Objective

- Support additional convenience operations

Details

- `void clear()`
- `int compare(const MyString& other)`
- support for `operator` <, ==, > etc.
- `boolean contains(const MyString& other)`
- `starts/endsWith(const MyString& other)`
- `char operator[int pos]/char at(int pos)`

Overview

Test Driven Development

- Eclipse plugins for TDD: CUTE
- Implementing an example

Static Analysis (SA)

- 3 rules of Scott Meyers “Effective C++ 2nd” (Item 3, 11, 14)
- Tools for SA:
  - Lint, gcc –weffc++
- Eclipse plugins for SA:
  - Codan
  - Linticator
  - Includator

Possible levels of Static Analysis:

Micro-Level
- Code, MISRA-C
- e.g: =, ==, { },

Macro-Level
- Class-Design, Effective Rules for C++, Java, C#
- e.g: by reference, String concat, Exception-Handling

Architecture-Level:
- Layers, Graphs, Subsystems, Compoments, Interfaces
- e.g: Coupling, Dependency, etc…

Critical areas of C (C Standard)

...are described in Appendix F/ANSI or G/ISO

- Unspecified behaviour
- Undefined behaviour
- Implementation-defined behaviour
- Locale-specific behaviour

failures can be detected

- at compilation stage / static
- at run-time / dynamic

Unspecified behaviour

```
for ( i = 0; i < 100; a[i++] = b[i] )
{
     ...;
}
```

```
a * b + c;
...
(a * b) + c;
...
a * (b + c);

a * (f() + g());

a = i + b[++i];
a = 2 + b[3];   // valid compiler implementation
a = 3 + b[3];   // valid compiler implementation
```

Empirically determined misbehaviour

Errors of omission and addition

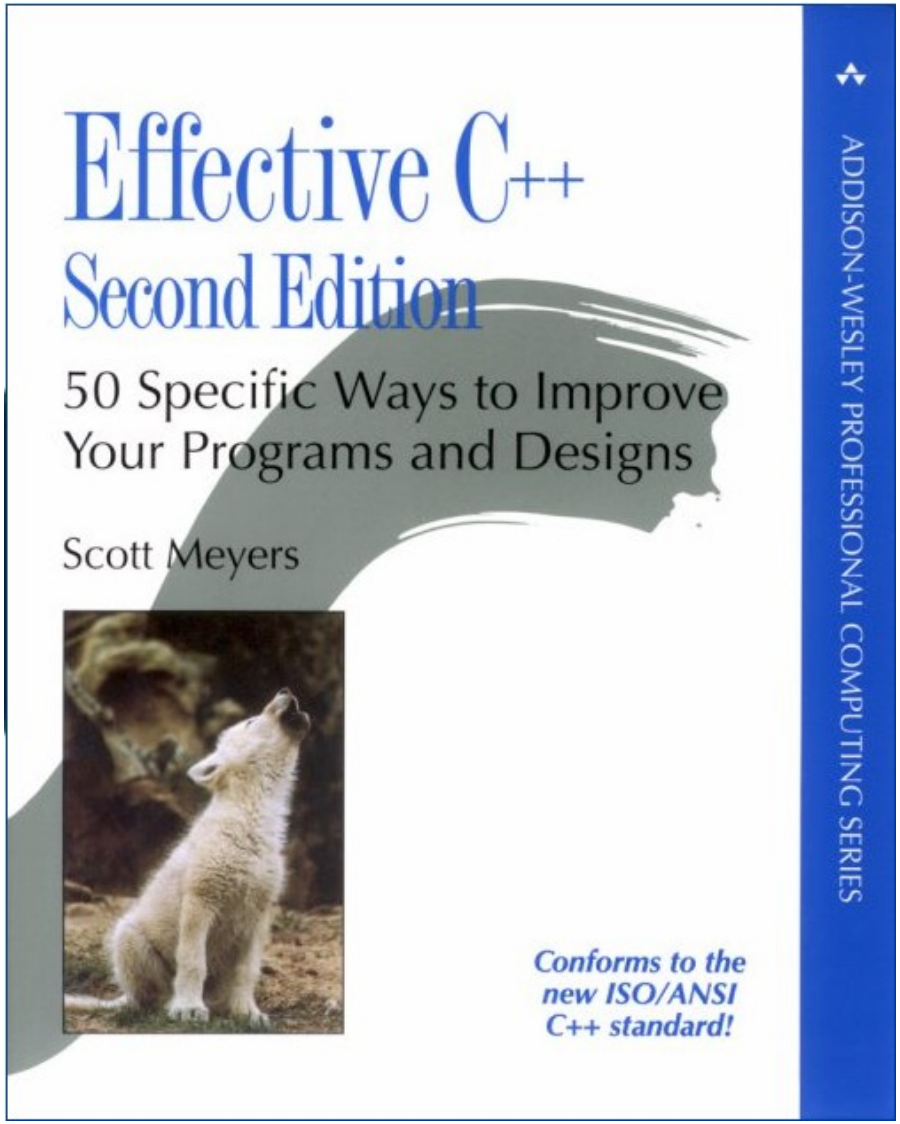
```
int a, b;
...
if ( a = b )
{
    ...
}
```

- occurs every 3306 lines in commerical C code

```
...
a == b;
...
```

- occurs every 12325 lines in commerical C code

```
...
if ( a == b );
{
    ...
}
```


Effective C++
Second Edition
50 Specific Ways to Improve Your Programs and Designs
Scott Meyers
Conforms to the new ISO/ANSI C++ standard!
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Overview: EC++ 2nd Edition

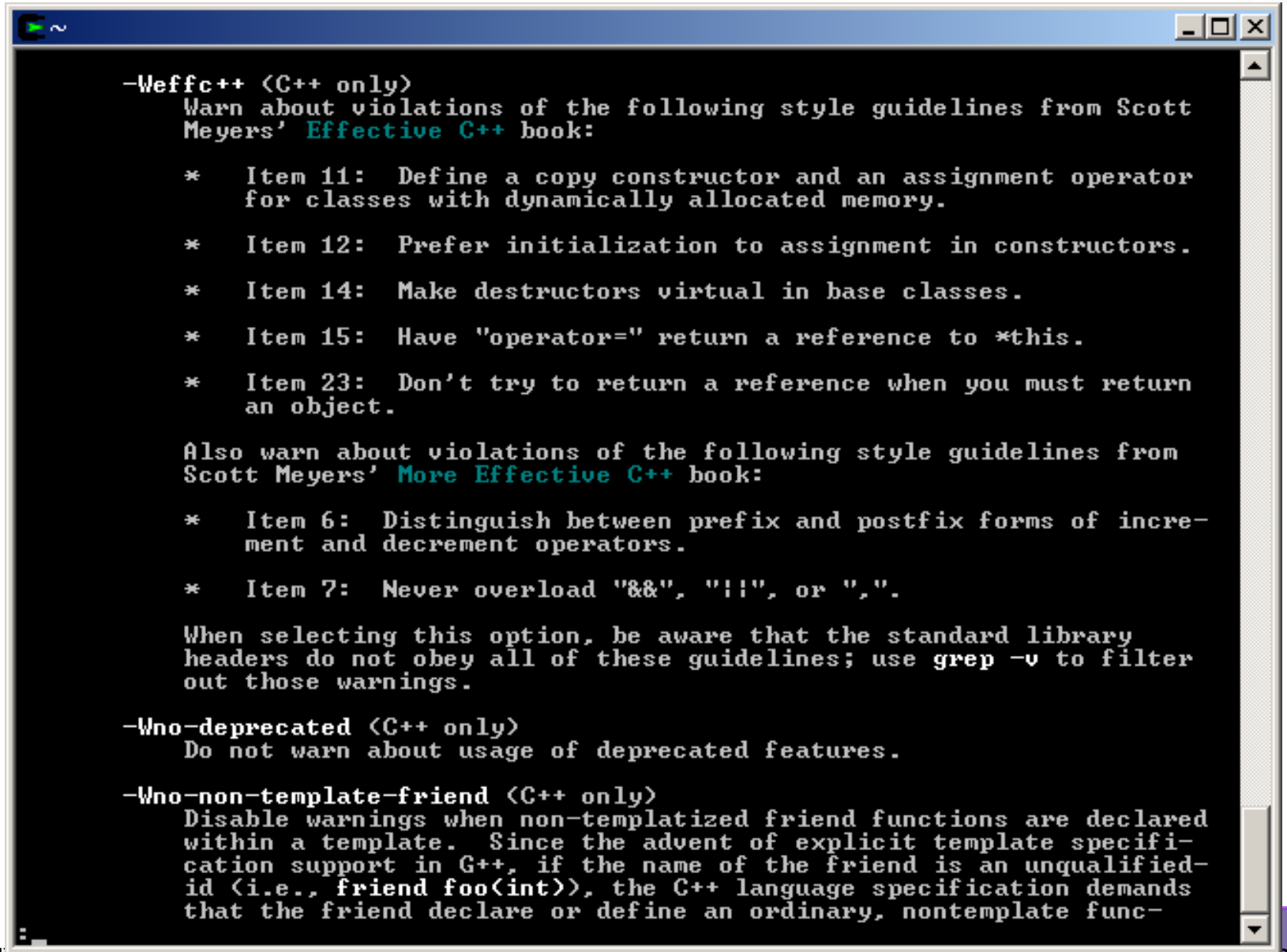
- Shifting from C to C++ (Item 1 - 4)

- Memory Management (Item 5 - 10)
- Constructors, Destructors, Assignment Operators (Item 11 - 17)

- Classes and Functions: Design and Declaration (Item 18 - 28)
- Classes and Functions: Implementation (Item 29 - 34)
- Inheritance and Object-Oriented Design (Item 35 - 44)

- Miscellany (Item 45 - 50)

Support of Effective C++ in tools: e.g. g++ -WeffC++



Item 3: Prefer **new /delete** to **malloc/free**

Problem with `malloc` and `free`

- they don't know about constructors and destructors

```
string *stringArray1 =
  static_cast<string*>(malloc(10 * sizeof(string)));

string *stringArray2 = new string[10];
```

- stringArray1 point to memory enough for 10 strings
- stringArray2 point to memory with 10 fully constructed strings

Advantages of `new` / `delete`

- always calls default ctor / dtor
  - Can also be a disadvantage (then forbid default ctor)
- they are typesafe

Item 3: Prefer **new /delete** to **malloc/free**

Same for deallocation of memory

```
free(stringArray1);
delete [] stringArray2;
```

- free only releases the memory, no dtor is called
- delete[] does what the programmer expects
- NOTE: delete and delete[] are discussed in Item5 (!!!)

Always use matching allocate / deallocate calls:

```
new --> delete
new[] --> delete[]
malloc --> free
```

You are asking for trouble if you violate this rule

Constructor(s), Destructor, and Assignment Oper

Ctor, Dtor, (Cctor), operator=

every class you write will have

- one or more constructors,
- a destructor, and
- an assignment operator

In fact, they already HAVE one if you don‘t define it (Item50)

these are your bread-and-butter functions

it's vital that you get them right

Example:

```
// a poorly designed String class
class String {
public:
  String(const char *value);
  ~String();
  ...                    // no copy ctor or operator=
private:
  char *data;
};
```

Item 11: cctor & operator= for classes with dny. memory

```
String::String(const char *value)
{
  if (value) {
    data = new char[strlen(value) + 1];
    strcpy(data, value);
  }
  else {
    data = new char[1];
    *data = '\0';
  }
}

inline String::~String() { delete [] data; }
```

Item 11: cctor & operator= for classes with dny. memory

```
String a("Hello");
String b("World");
b = a; //…
```

- problems during assignment:
  - multiple pointers on the SAME data
  - multiple deletes are called on the SAME data
- there is no client-defined `operator=`
- default assignment operator performs memberwise assignment from the members (just a bitwise copy)

```
void doNothing(String localString) {}

String s = "The Truth Is Out There";
doNothing(s); //…
```

- The case of the copy constructor differs a little from that of the assignment operator

solution to these kinds of pointer aliasing problems:

- write your own versions of
  - the copy constructor and
  - the assignment operator

  if you have any pointers in your class

- Inside those functions, you can either
  - copy the pointed-to data structures, every object has its own copy
  - implement some kind of reference-counting scheme

if you want to **inhibit** assignment or copy of this class

- You *declare* the functions (`private,` as it turns out), but you don't define (i.e., implement) them at all (Item 27)
- Or use boost:non_copyable

default and **delete** in C++0x

```
struct NC { // NonCopyable „old style“
     NC() {…};
private:
     NC(const NC&); // no impl !
     NC& operator=(const NC&); // no impl !
};

struct NC { // NonCopyable in C++0x
     NC() = default;
     NC(const NC&) = delete;
     NC& operator=(const NC&) = delete;
};
```

Declare a copy constructor and an assignment operator for classes with dynamically allocated memory (ressources)

Example:

```
// a poorly designed String class
class String {
public:
  String(const char *value);
  ~String();
  ...              // TODO !!! copy ctor AND operator=
private:
  char *data;
};
```

Item 14: have base classes have virtual dtors.

```
class Target {
public:
  Target() { ++numTargets; }
  Target(const Target&) { ++numTargets; }
  ~Target() { --numTargets; }

  static size_t numberOfTargets() { return numTargets; }
  virtual bool fire();
private:
  static size_t numTargets; // object counter
};

// Target.cpp init static member
size_t Target::numTargets = 0;

class EnemyTank: public Target {
public:
  EnemyTank() { ++numTanks; }
  EnemyTank(const EnemyTank& rhs): Target(rhs) { ++numTanks; }
  ~EnemyTank() { --numTanks; }

  static size_t numberOfTanks()  { return numTanks; }
  virtual bool fire();
private:
  static size_t numTanks;         // object counter for tanks
};
```

Item 14: have base classes have virtual dtors.

```
Target *targetPtr = new EnemyTank;
...
delete targetPtr;
```

Item 14: have base classes have virtual dtors.

```
Target *targetPtr = new EnemyTank;
...
delete targetPtr; //behaviour is undefined if no virtual dtor
```

- rule:
  declare a virtual destructor in a class if and only if that class contains at least one virtual function
- Efficiency in C++: declaring all destructors virtual is just as wrong as never declaring them virtual

- Finally, it can be convenient to declare pure virtual destructors in some classes
- one twist, however: you must provide a *definition* for the pure virtual destructor

Item 14: have base classes have virtual dtors.

- When you
  - try to delete a derived class object
  - through a base class pointer
  - and
  - the base class has a nonvirtual destructor
  - the results are undefined
- To avoid this problem you have only to make the destructor *virtual*
- If a class does *not* contain any virtual functions, that is often an indication that it is not meant to be used as a base class

Item 15: Have `operator=` return `*this`

- C++ and the creator strived to ensure that user-defined types would mimic the built-in types as closely as possible
- With built-in types, you can chain assignments together

```
int w, x, y, z;
w = x = y = z = 0;
```

- you should be able to chain together assignments for user-defined types, too

```
String w, x, y, z;
w = x = y = z = “hello“;

w = (x = (y = (z = "Hello")));
w.operator=(x.operator=(y.operator=(z.operator=("Hello"))));
```

Item 15: Have `operator=` return `*this`

`operator=`

- return type of must be acceptable as an input to the function
- define that return a reference to their left-hand argument, `*this`

```
String& String::operator=(const String& rhs)
{
  ...
  return *this;                 // return reference
                                // to left-hand object
}
```

Codan == CODe ANalysis

Tool Vendors

- **create plugins containing end-user checkers and templates**
- **integrate command line static analysis tools into CDT**

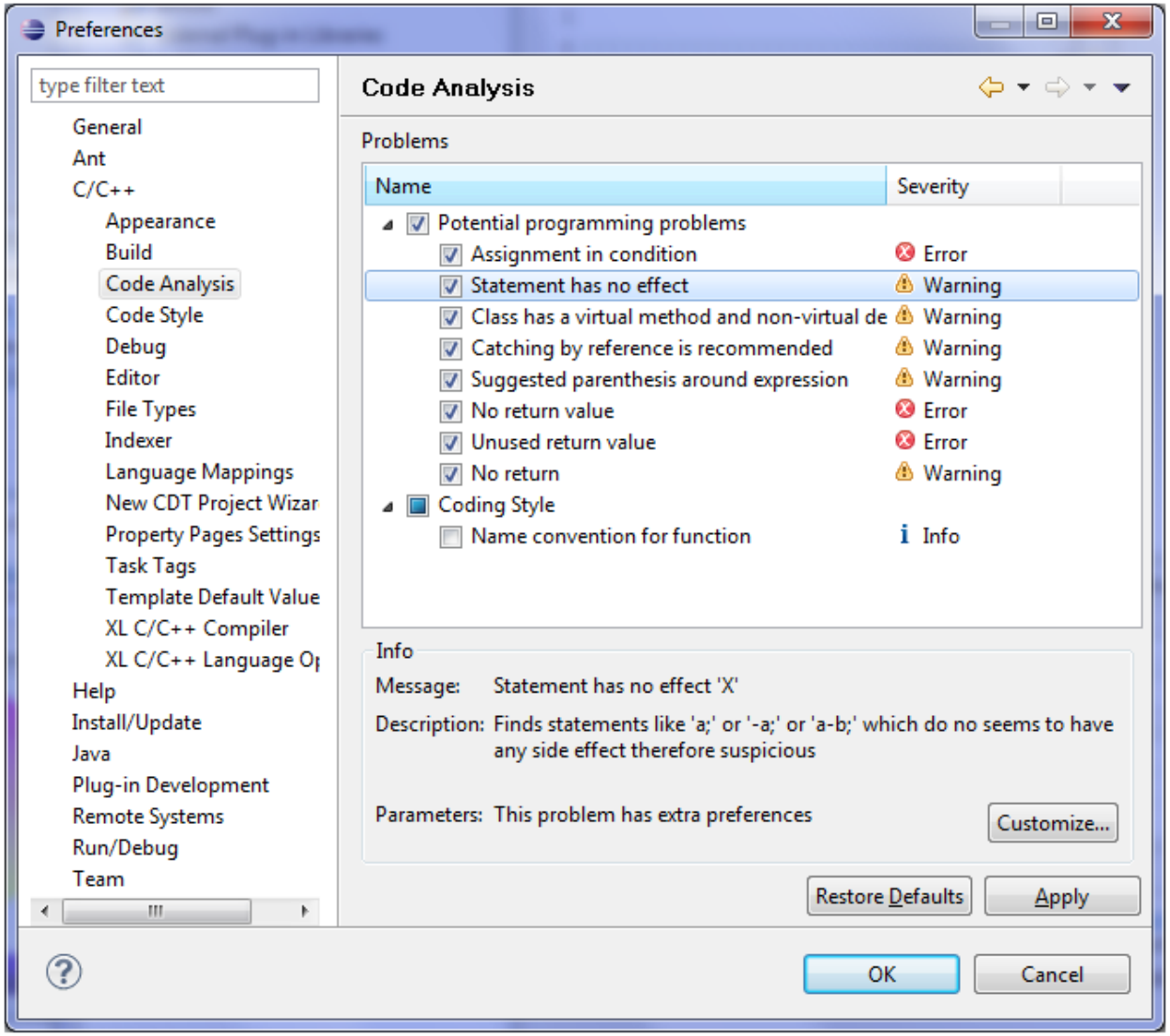
Software Architects, Process Enforcement

- **create customized new checkers, based on templates (no programming involved)**
- **To create problem profiles**

Developer, Tester, Code Inspector

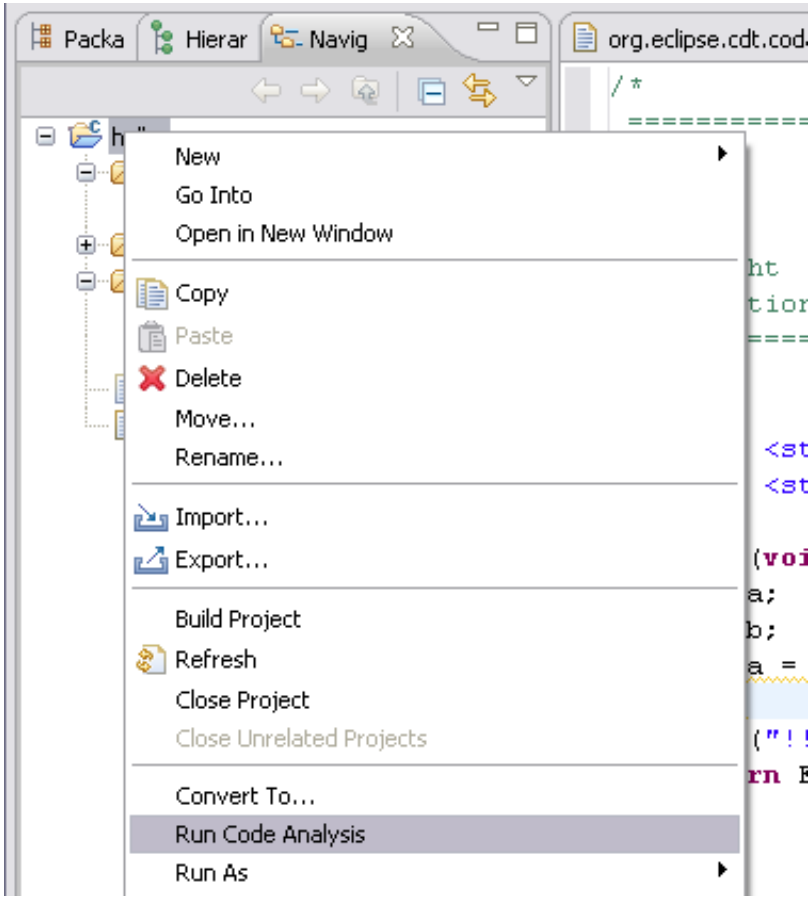
- **check for errors as you type and have a quick way to fix them**
- **find bugs, security violations, API violations, coding standard violations during code inspection and before code execution**

Codan: Severity + Enablement on Workspace/Project

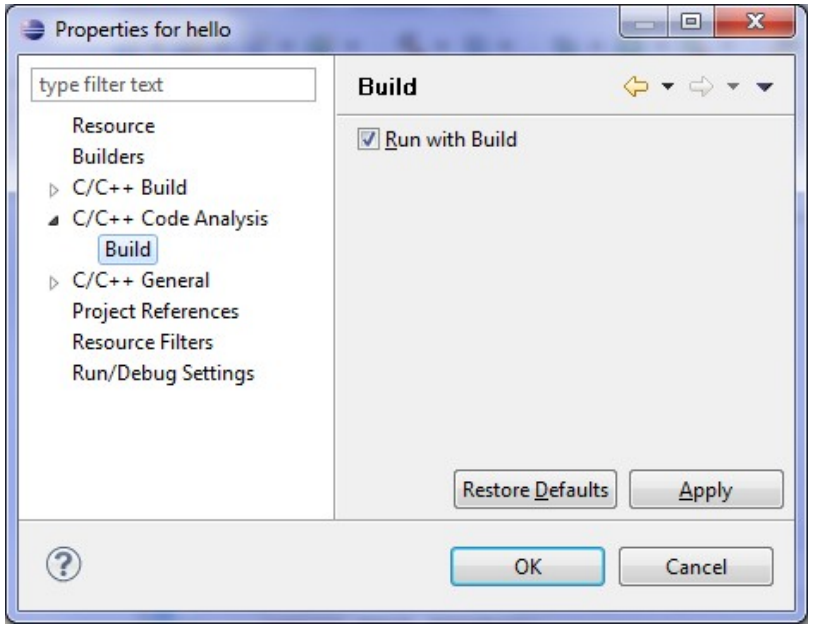


Codan: Launch Control

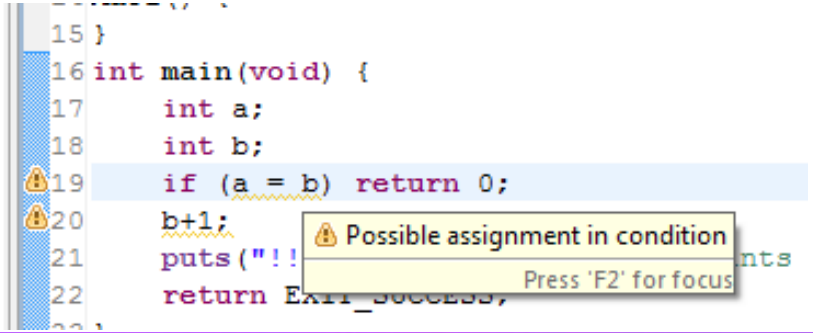
Run on demand from context menu



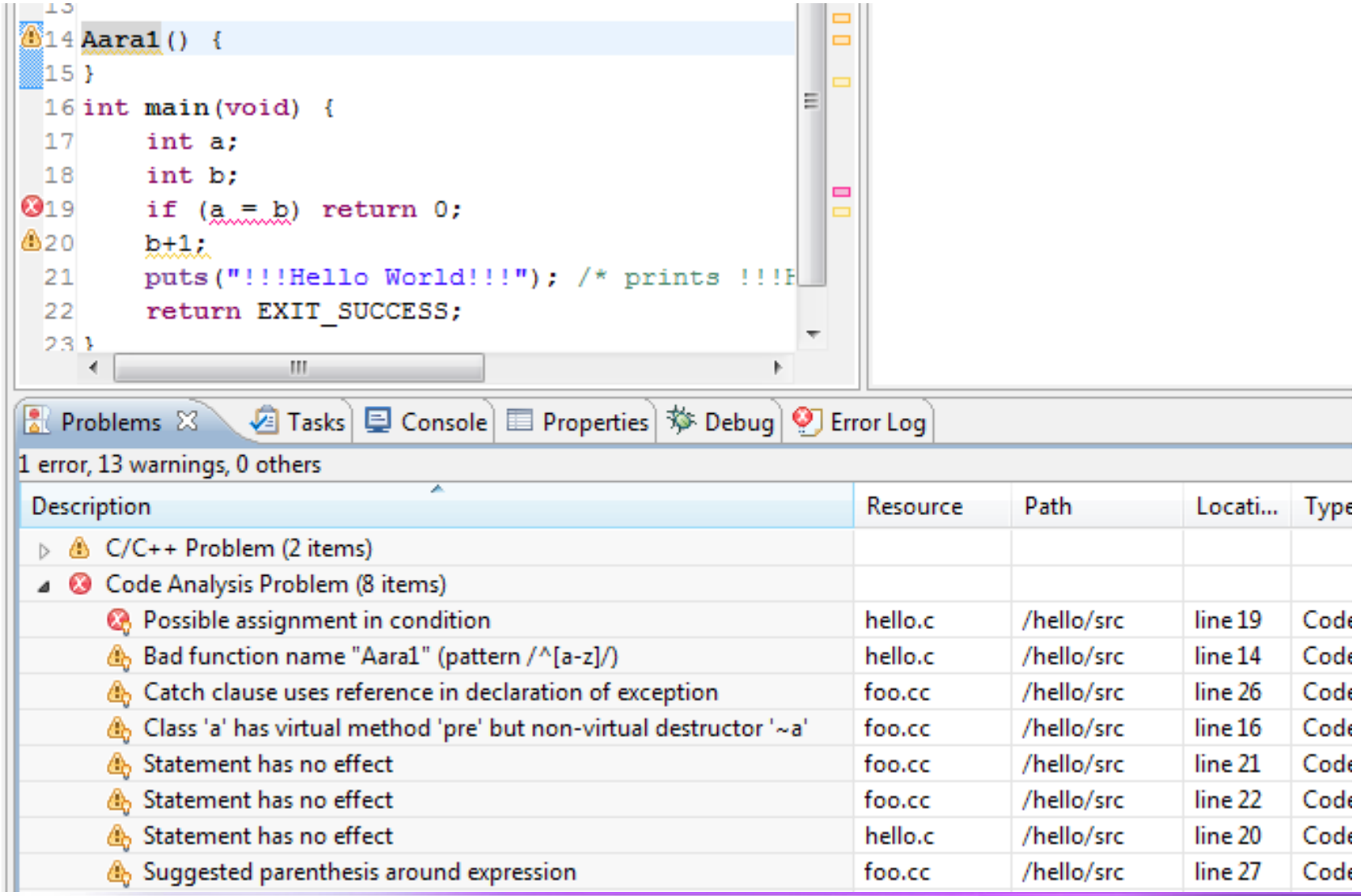
Run with Build



Run as you type



Codan: Problem Markers



Internal Checker

- Problem scope is userdefine (you found e.g. a bug)
- Pick a model to find that problem e.g.
  AST, Index, ControlFlow-, DataFlow-, Call-Graph
- Extend abstract checker for that model + implement check
- Create Extension for finding
- Create Autofix Action ?

External Checker

- Problem scope is defined by external tool
- Integrate output into eclipse concole/problems view (error parser)
- Offer Autofix Actions ?

Linticator

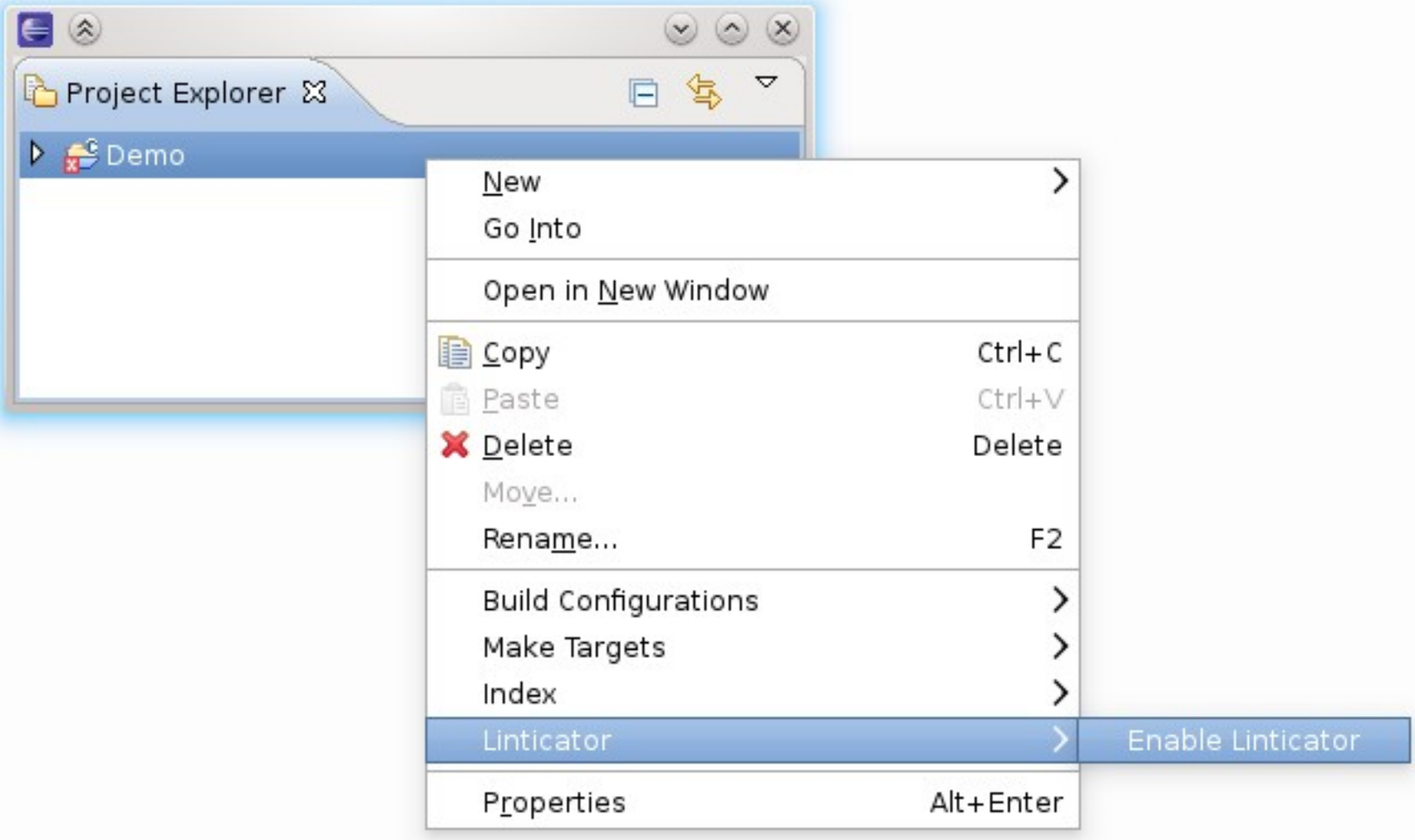
Project of IFS in Rapperswil, CH

- http://www.linticator.ch

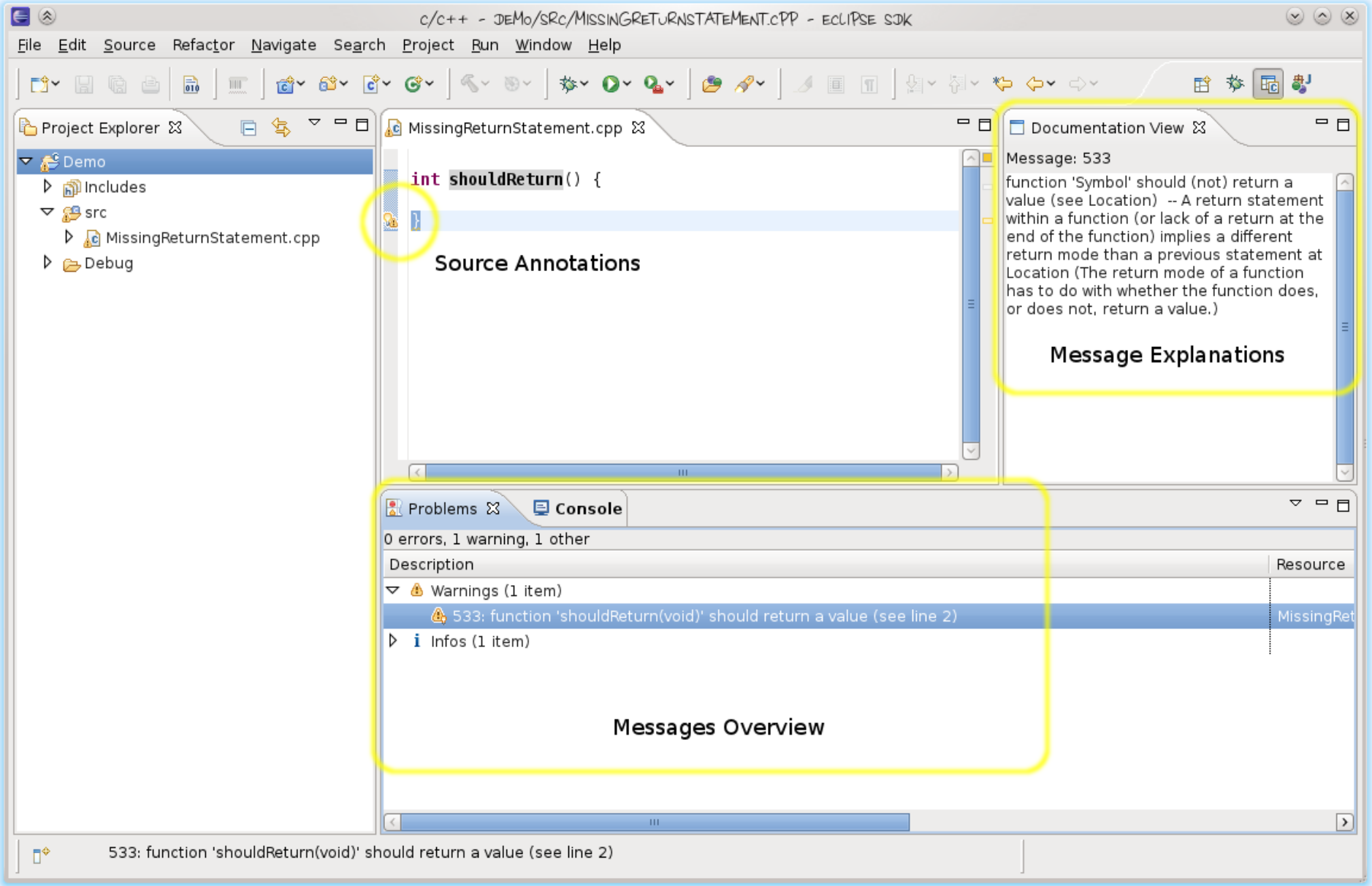
Features

- Autosetup + Project Configuration
- Problems Overview
- Message Explanation View
- Quickfixes
- Supressions

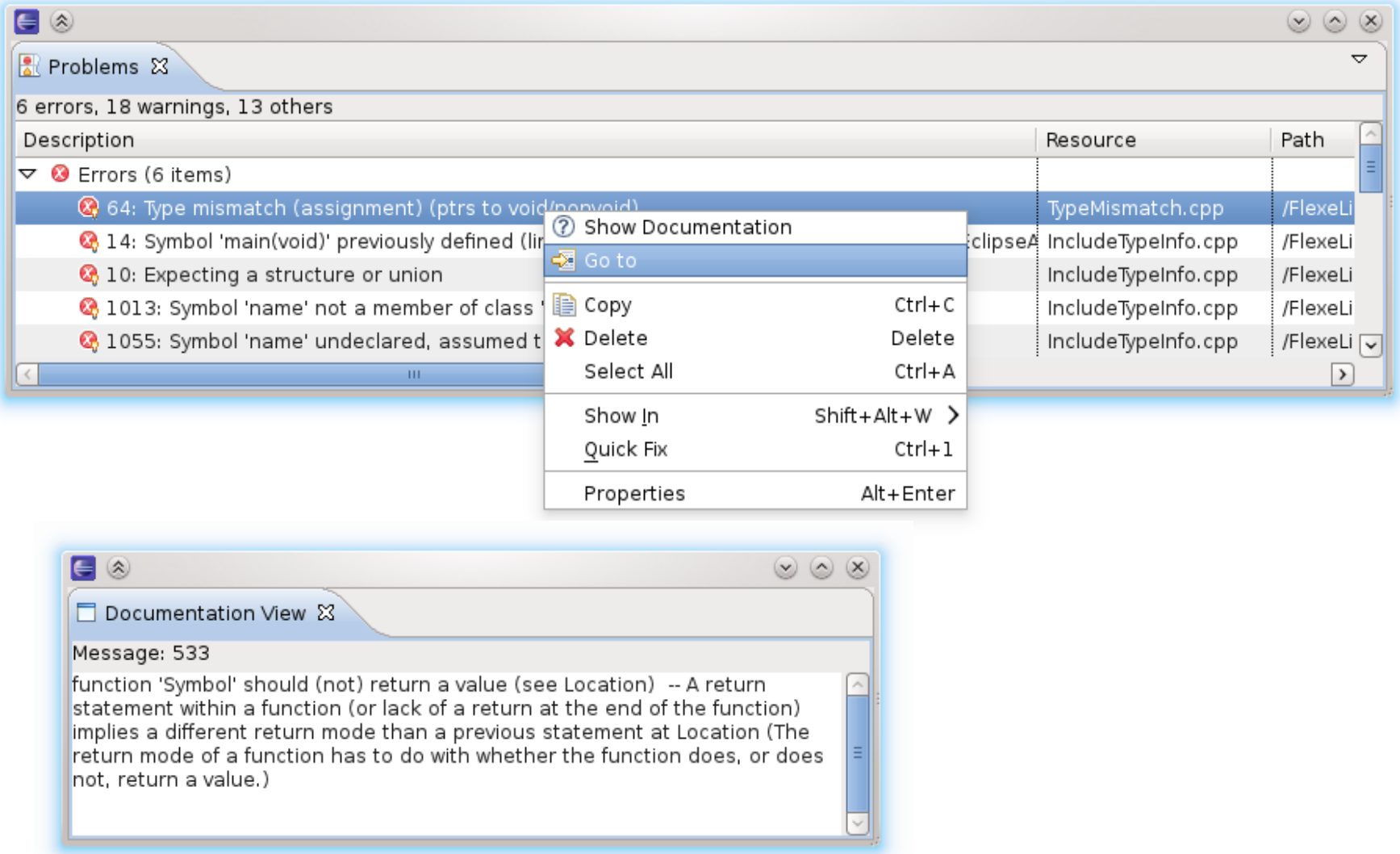
Linticator: Project Configuration



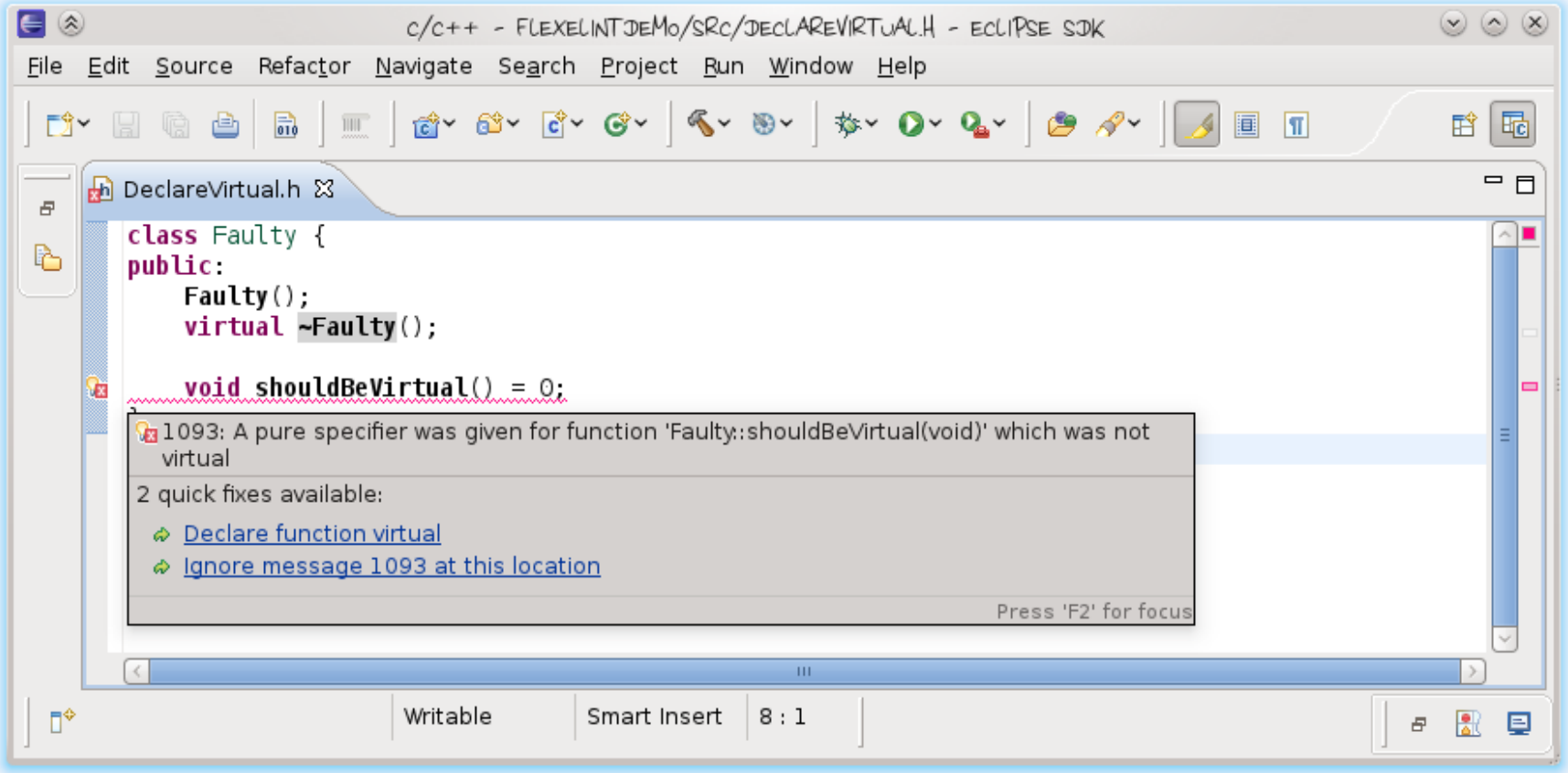
Linticator: Overview



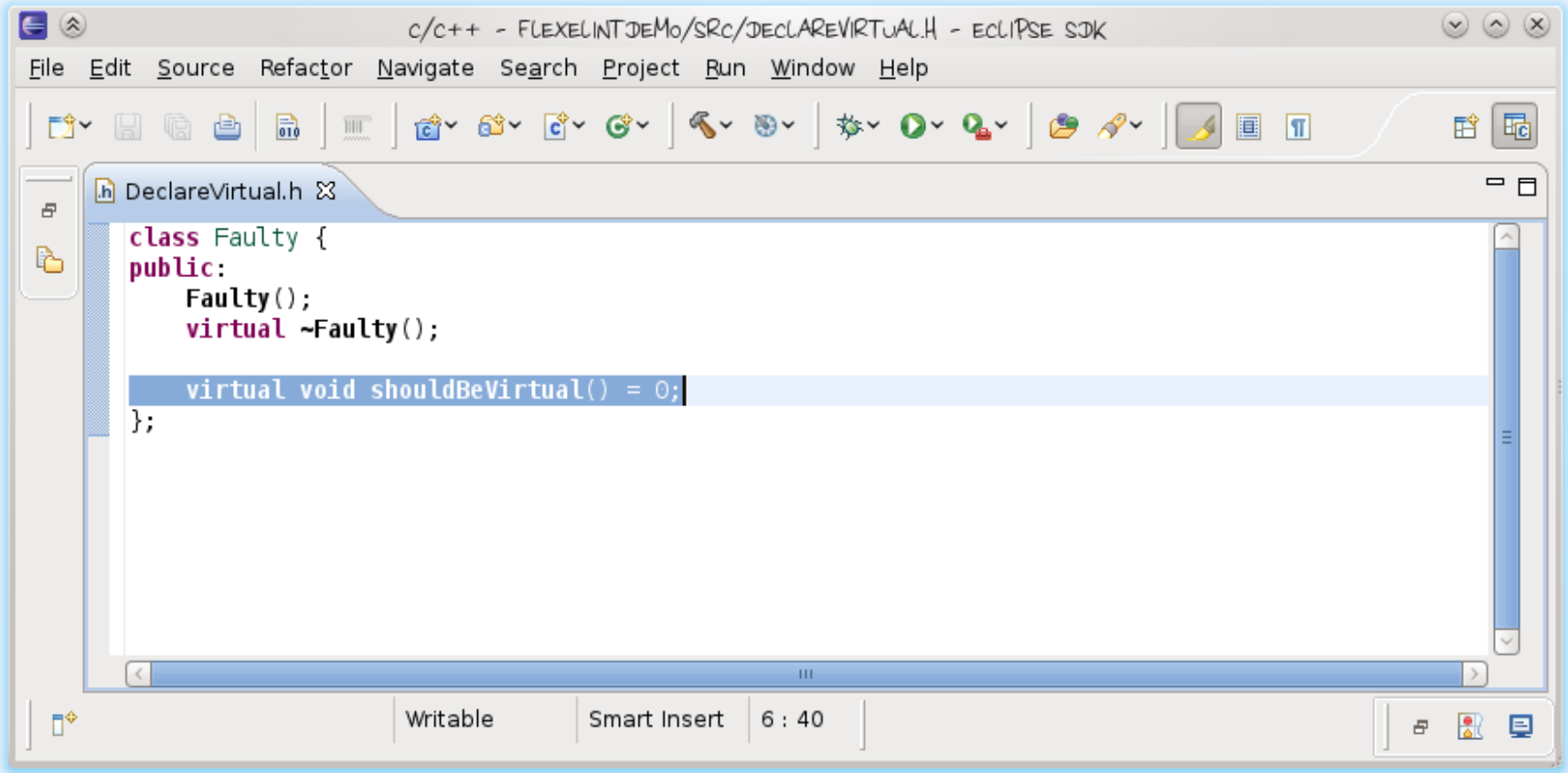
Linticator: Problems View + Message Explanation



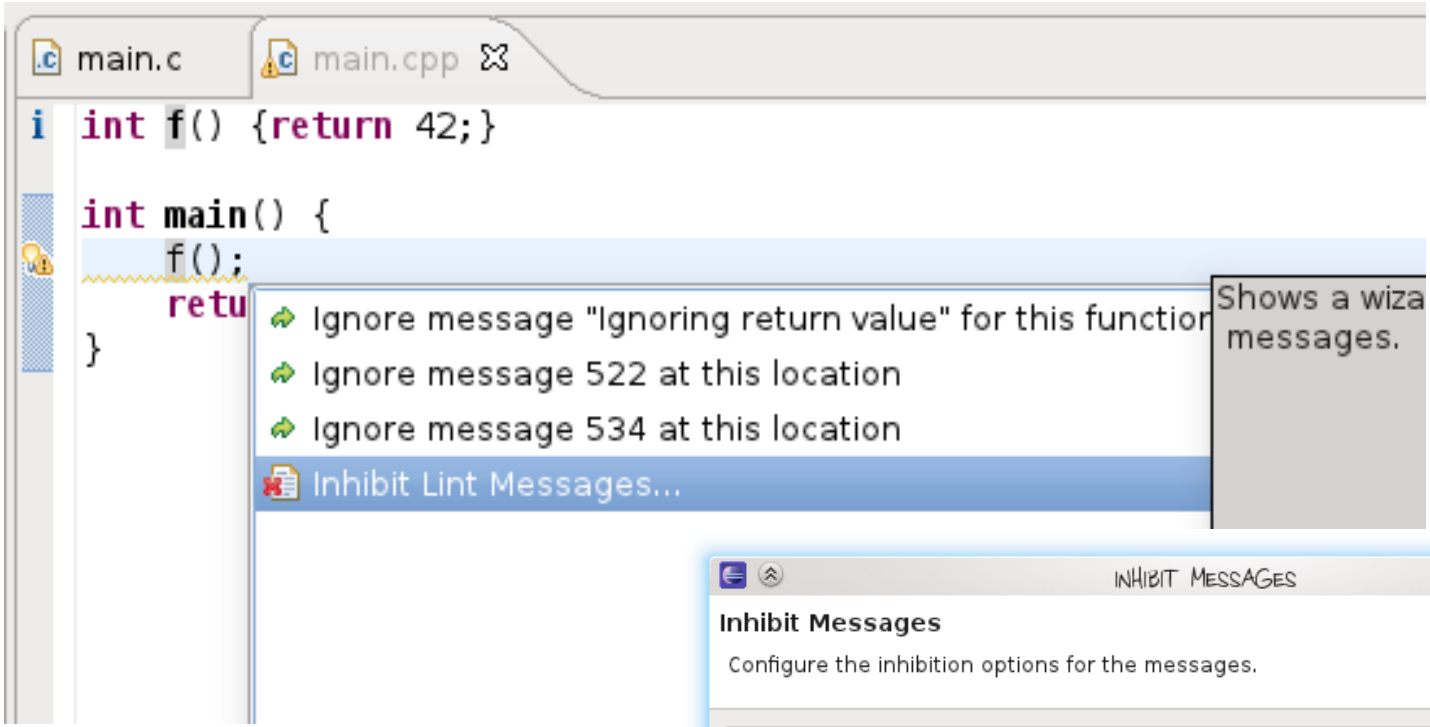
Linticator: Quickfix

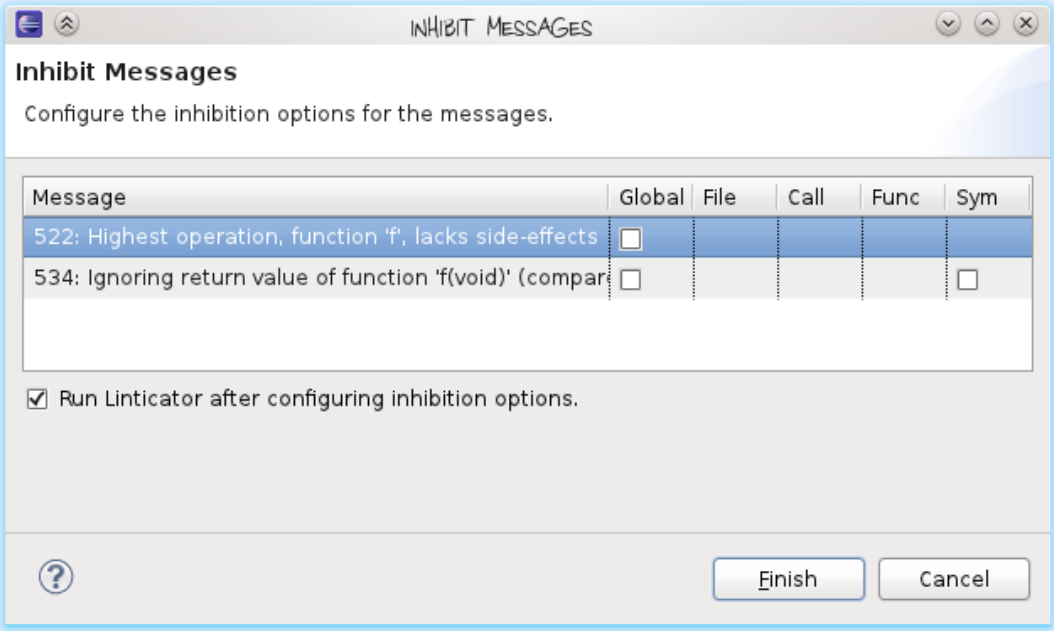


Linticator: Quickfix



Linticator: Supress Message





Includator

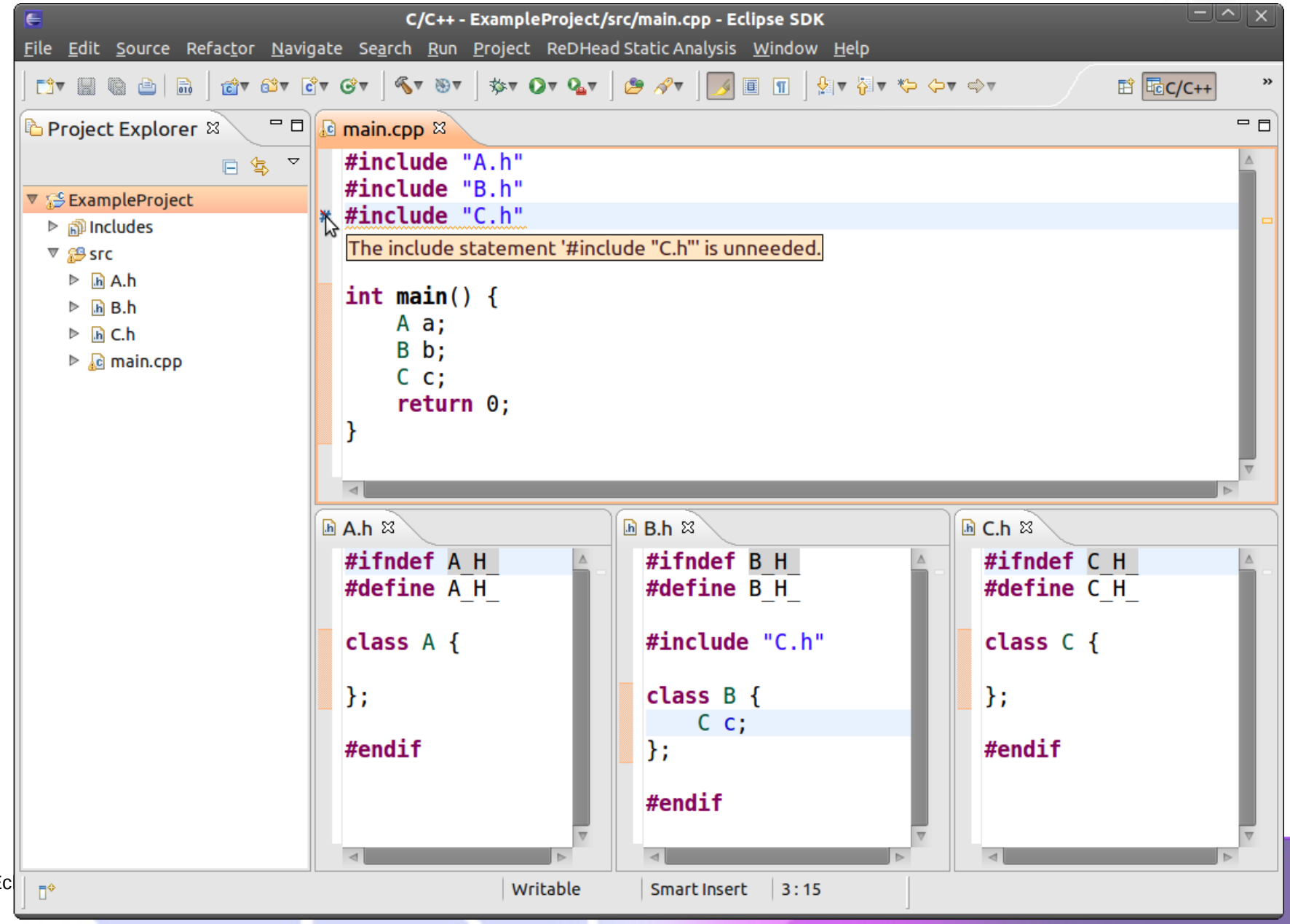
Project of IFS in Rapperswil, CH

- http://www.includator.ch

Features

- Find unused includes
- Directly include referenced files
- Organize includes
- Static code coverage
- Find unused files

Includator: Find unused includes



Includator: Directly include referenced files

This feature helps to automatically add include directives to a file under consideration, so that all files containing referenced declarations get included (directly). The features is based on the idea of John Lakos found in his book Large-Scale C++ Software Design (5th guidline)

Example

```cpp
  /* main.cpp */

#include "Y.h"

int main() {
  X x;
  return 0;
}
```

```cpp
  /* Y.h */

#include "X.h"

/* more code */
```

```cpp
  /* X.h */

class X { };

/* ... */
```

Here, the **Includator** makes the proposal to include file **X.h** directly into **main.cpp** independent of other, used or unused, types in **Y.h**.

Includator: Organize includes

This feature is similar to the one known form Eclipse JDT called *Organize Imports*. Its task is to find includes that should be added and/or includes that can be removed from a given file.

Example

```
1   /* main.cpp */
2
3  #include "Y.h"
4  #include "Z.h
5
6  int main() {
7    Y y;
8    X x;
9    return 0;
10 }
```

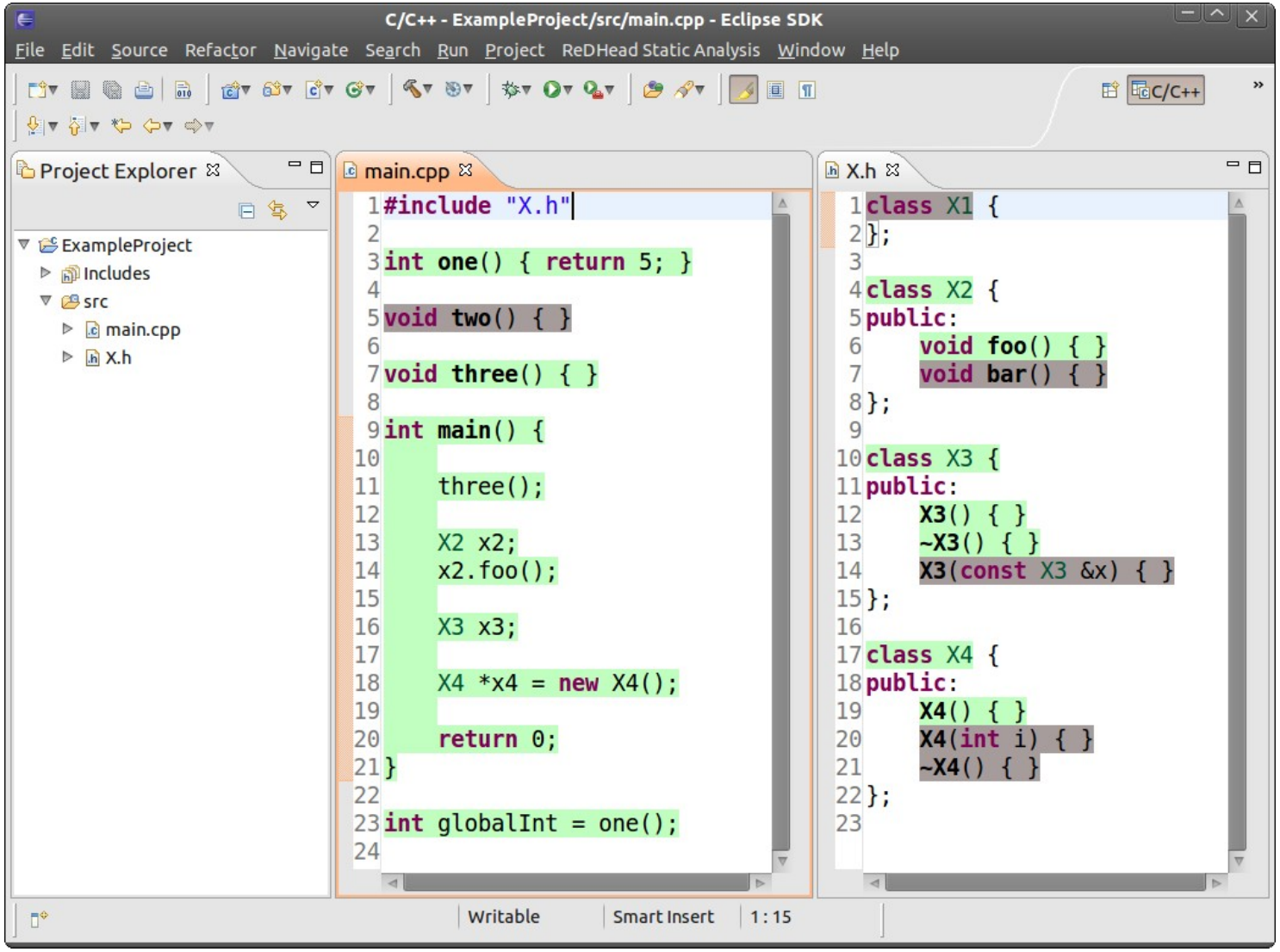
```
1   /* X.h */
2
3  class X { };
```

```
1   /* Y.h */
2
3  class Y { };
```

```
1   /* Z.h */
2
3  class Z { };
```

Here, the **Includator** makes the proposal to to include file **X.h** and to remove the include of **Z.h**.

Includator: Static code coverage



Includator: Find unused files

Finding unused files means to look at all the include dependencies in a given C++ project and find header files which are not included at all. This situation can often arise after unused includes directives have been removed with the **Includator**'s *find unused includes* or *organize includes* features.

Example

Consider the following project structure:

- project:
  - main.cpp
  - X.h
  - Y.h
  - Z.h

```
1  //main.cpp
2
3  #include "X.h"
4  #include "Y.h"
5
6  int main() {
7    X x;
8    Y y;
9    return 0;
10 }
```

Finding unused includes in the context of this project means to propose the deletion of file **Z.h**.

More information

Eclipse CDT: http://eclipse.org/cdt

Linux Tools Project: http://www.eclipse.org/linuxtools

Wascana: http://code.google.com/a/eclipselabs.org/p/wascana

CUTE: http://www.cute-test.com/

Linticator: http://www.linticator.ch

Includator: http://includator.ch/

Sconsolidator: http://www.sconsolidator.ch/

Conclusion

We hope you have enjoyed seeing some of the breadth and power of a few Eclipse C/C++ tools. All communities of developers writing these tools are active and always interested in feedback. Any level of participation is greatly appreciated and can be as easy as filing a bug, tweeting about a cool feature, or writing a blog post about how you set things up for your project.

Thank you.