Hands On With the C/C++ IDE

Andrew Overholt, Bernhard Merkle

Hands On With The C/C++ IDE

In this tutorial, attendees will be led through focused 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

eclipsecon™

Easy Tutorial Setup: Use Virtual Images:

- HIGHLY RECOMMENDED: ready to go
- 4GB Virtual Box Image File
- Fedora 14 pre-installed with Eclipse CDT Linux Tools
- Available for
  - Oracle Virtual Box

Tutorial Setup 101

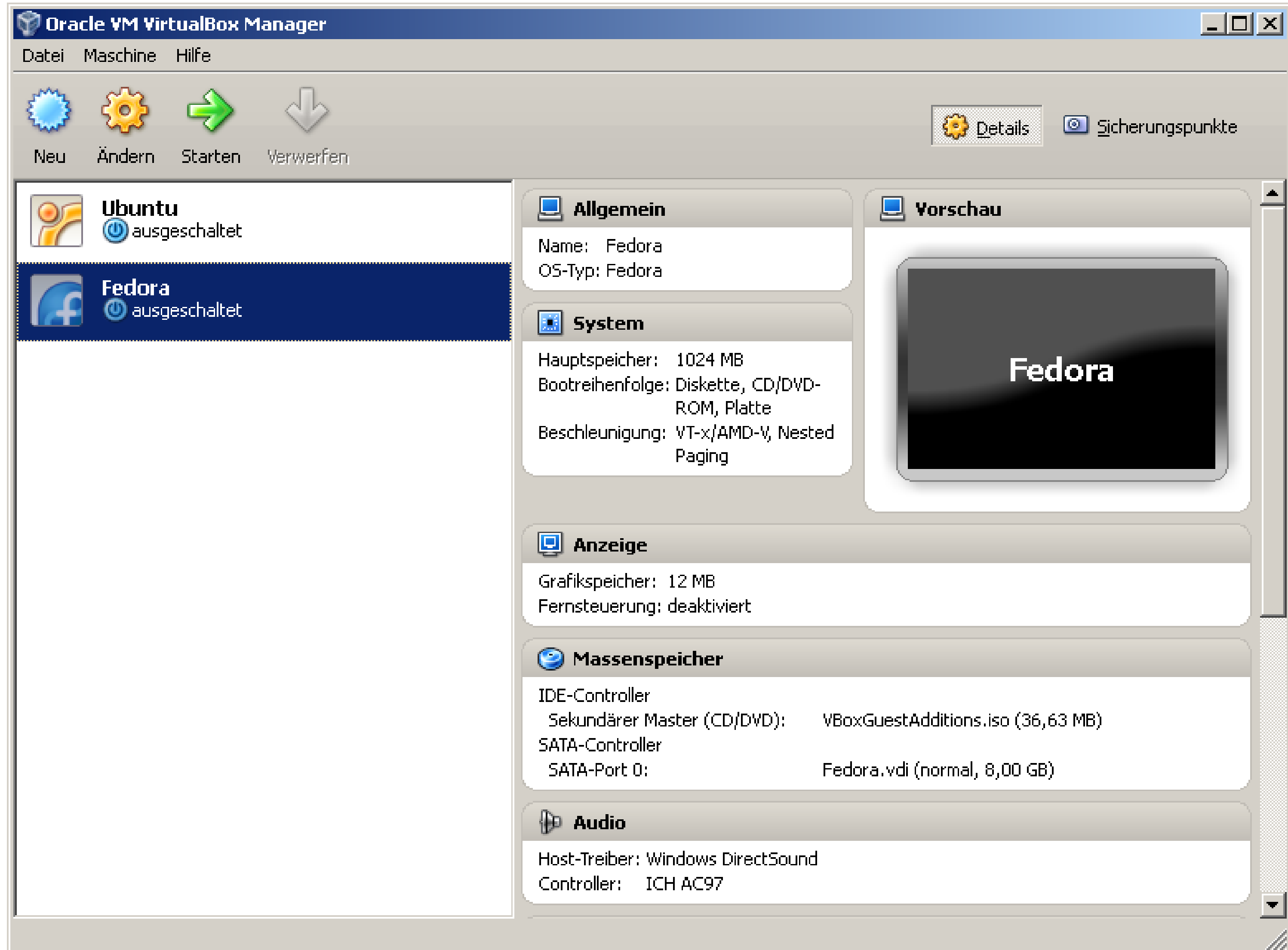
Copy **`VirtualBox-Image`** somewhere on HardDrive (4GB)

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

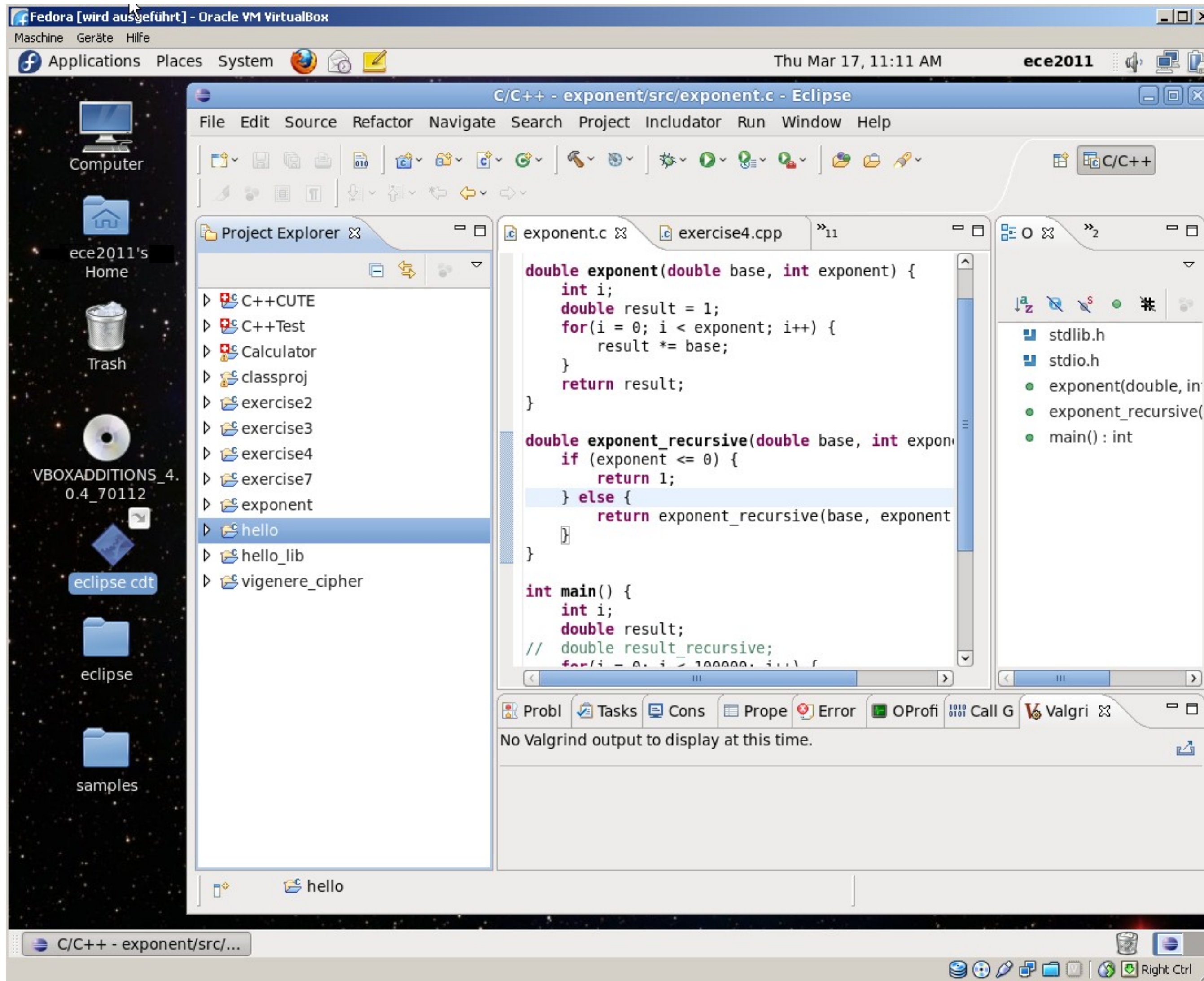
Startup VirtualBox

- Machine→Add… (Ctrl-A)
- Select Fedora.vbox (Copied in Step1)
- Startup the “Fedora” Virtual Machine
- Login: User “ece2011”, Password “ece2011”

Tutorial Setup 101

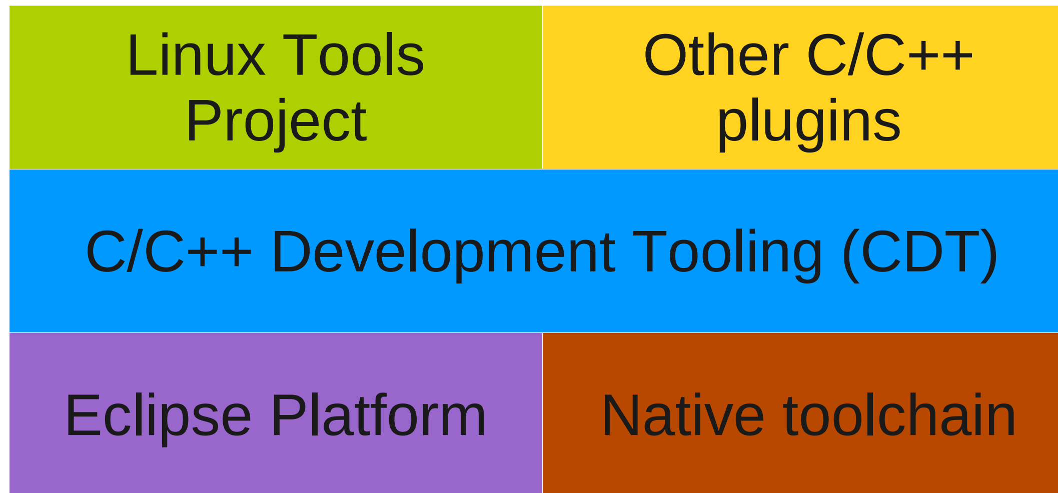


Tutorial Setup 101



EclipseCon C/C++ Tutorial | © 2011 by Bernhard Merkle, and Andrew Overholt, made available under the EPL v1.0

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

eclipsecon™

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

eclipsecon™

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

eclipsecon™

Project of IFS in Rapperswil, CH

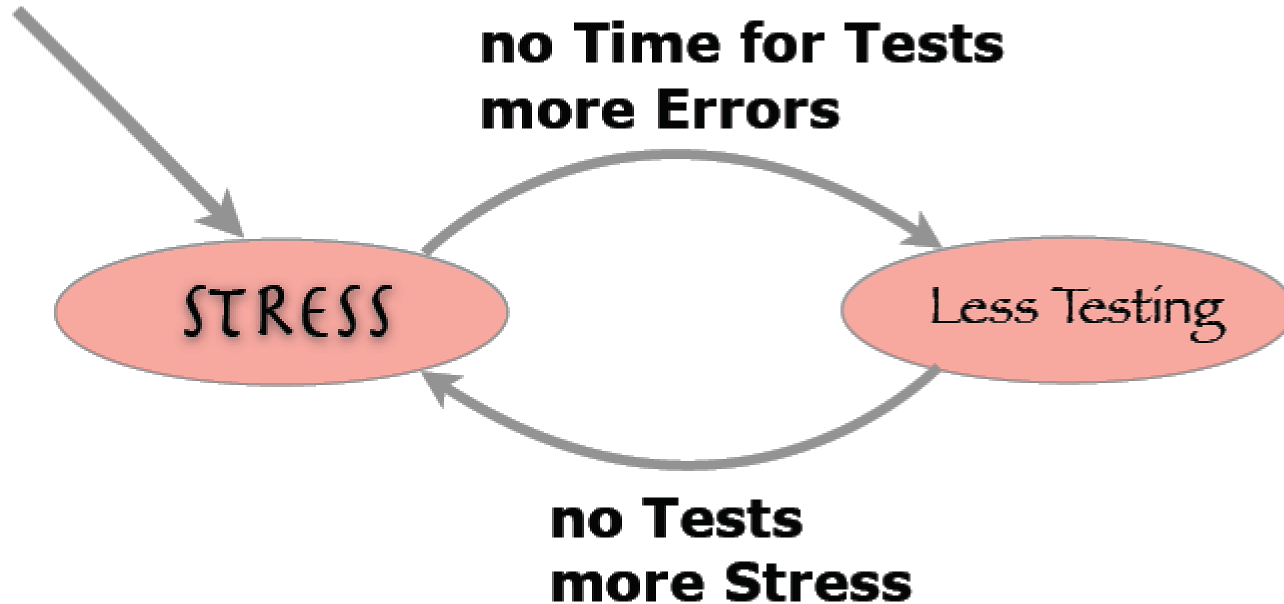
- http://www.cute-test.com

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

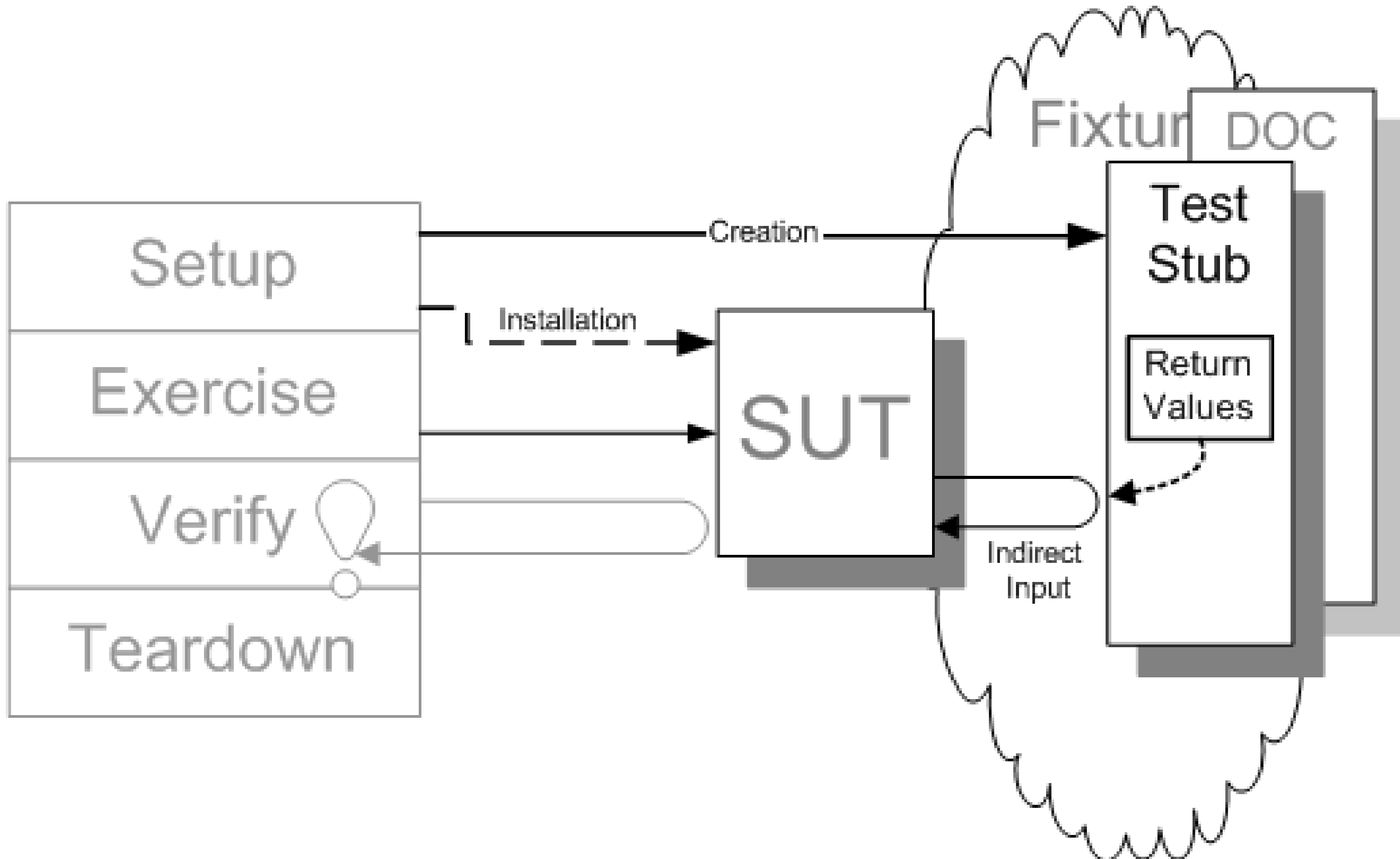
Vicious Circle: Testing – Stress



Help:

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

Structure of a typical Unit Testing Framework



eclipseCON™

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

Using CUTE: it IS EASY !!! ☺

eclipsecon™

```
#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

Collecting multiple Tests

eclipsecon™

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();`

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

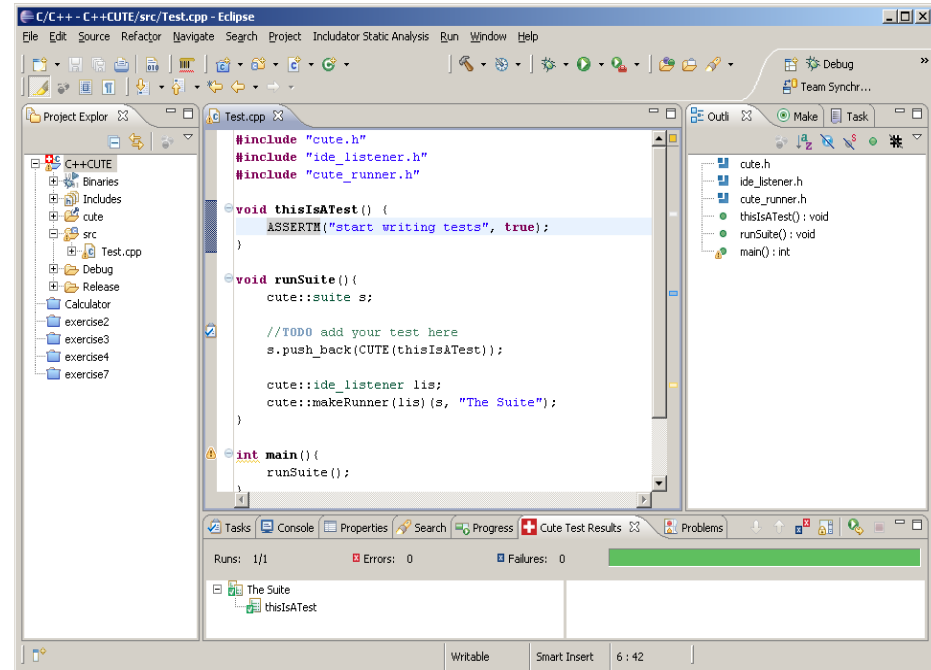
CUTE: Test Fixtures

eclipsecon™

```
#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));
...
```

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

Project Explorer: 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();
}
```

Outline: 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

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

eclipsecon™

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:

eclipsecon™

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

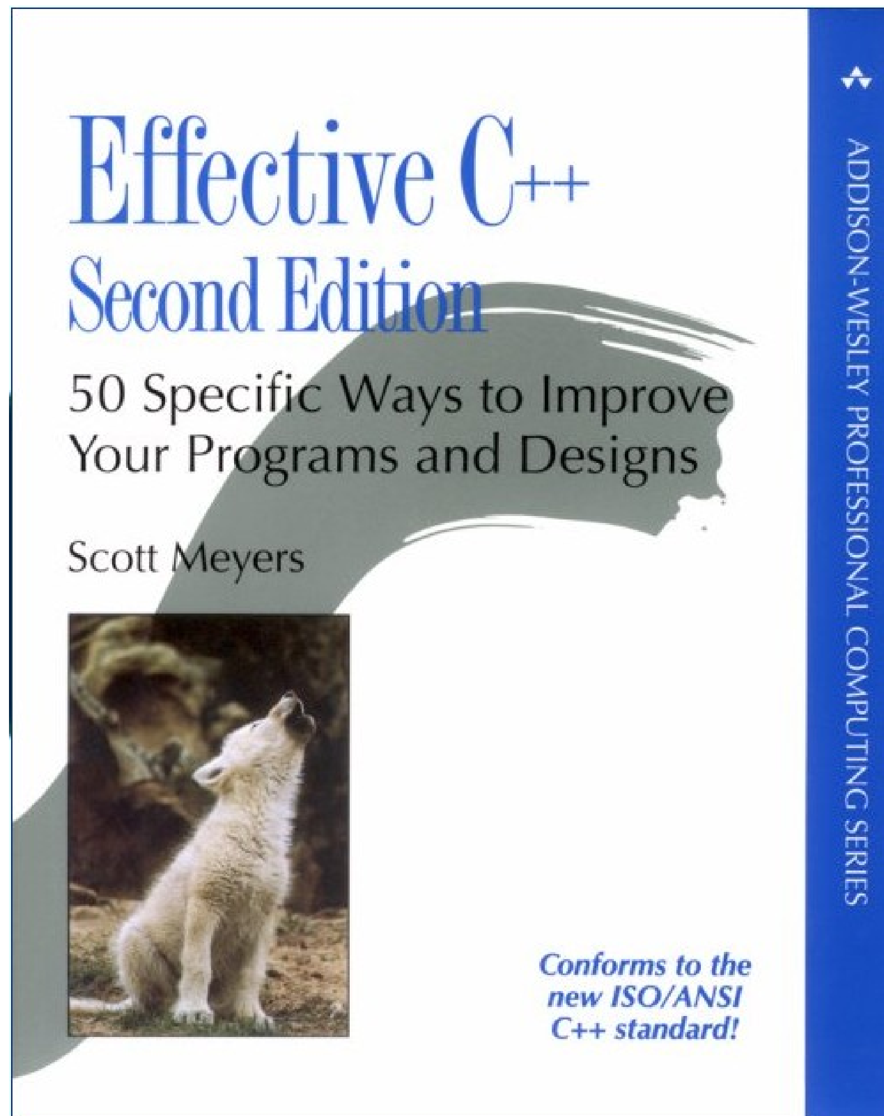
```
int a, b;
...
if ( a = b )
{
    ...
}
```

- 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
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES
Conforms to the
new ISO/ANSI
C++ standard!

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++

```
-Weffc++ (C++ only)
    Warn about violations of the following style guidelines from Scott
    Meyers' Effective C++ book:

    *   Item 11:  Define a copy constructor and an assignment operator
        for classes with dynamically allocated memory.

    *   Item 12:  Prefer initialization to assignment in constructors.

    *   Item 14:  Make destructors virtual in base classes.

    *   Item 15:  Have "operator=" return a reference to *this.

    *   Item 23:  Don't try to return a reference when you must return
        an object.

    Also warn about violations of the following style guidelines from
    Scott Meyers' More Effective C++ book:

    *   Item 6:  Distinguish between prefix and postfix forms of incre-
        ment and decrement operators.

    *   Item 7:  Never overload "&&", "||", or ",".

    When selecting this option, be aware that the standard library
    headers do not obey all of these guidelines; use grep -v to filter
    out those warnings.

-Wno-deprecated (C++ only)
    Do not warn about usage of deprecated features.

-Wno-non-template-friend (C++ only)
    Disable warnings when non-templatized friend functions are declared
    within a template.  Since the advent of explicit template specifi-
    cation support in G++, if the name of the friend is an unqualified-
    id (i.e., friend foo(int)), the C++ language specification demands
    that the friend declare or define an ordinary, nontemplate func-
:
```

Constructor(s), Destructor, and Assignment Oper

eclipsecon™

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

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

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

eclipseCON™

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

eclipsecon™

Preferences

type filter text

- General
- Android
- Ant
- C/C++
  - Appearance
  - Build
  - Code Analysis
  - Code Style
  - Debug
  - Editor
  - File Types
  - Indexer
  - Language Mappings
  - New CDT Project Wiza
  - Property Pages Settin
  - Task Tags
  - Template Default Value
- Checkstyle
- Ecore Tools Diagram
- ECP
- Graphiti Test Preferences
- Help
- Includator
- Install/Update
- Java
- Library Hover
- Linticator
- Logic Diagrams
- Model Validation
- Mwe2
- Plug-in Development
- Run/Debug
- Team
  - CVS
  - File Content

Code Analysis

Problems

Name	Severity
Potential Programming Problems	
Assignment in condition	Warning
Statement has no effect	Warning
Class has a virtual method and non-virtual destruc	Warning
Catching by reference is recommended	Warning
Suggested parenthesis around expression	Warning
No return value	Error
Unused return value	Error
No return	Warning
Assignment to itself	Error
Suspicious semicolon	Warning
No break at end of case	Warning
Unused variable declaration in file scope	Warning
Unused function declaration	Warning
Unused static function	Warning
Coding Style	
Name convention for function	Info
Return with parenthesis	Warning
Syntax and Semantic Errors	
Symbol is not resolved	Error
Invalid overload	Error
Ambiguous problem	Error
Circular inheritance	Error
Invalid redeclaration	Error
Invalid redefinition	Error
Member declaration not found	Error
Label statement not found	Error
Invalid template argument	Error

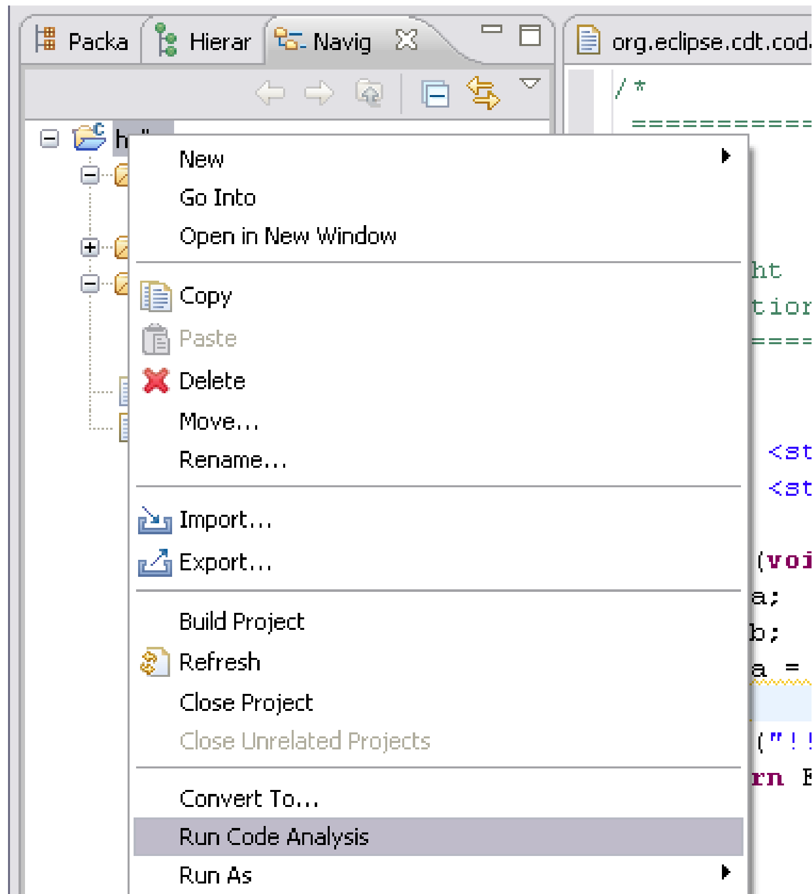
Customize Selected...

Restore Defaults Apply

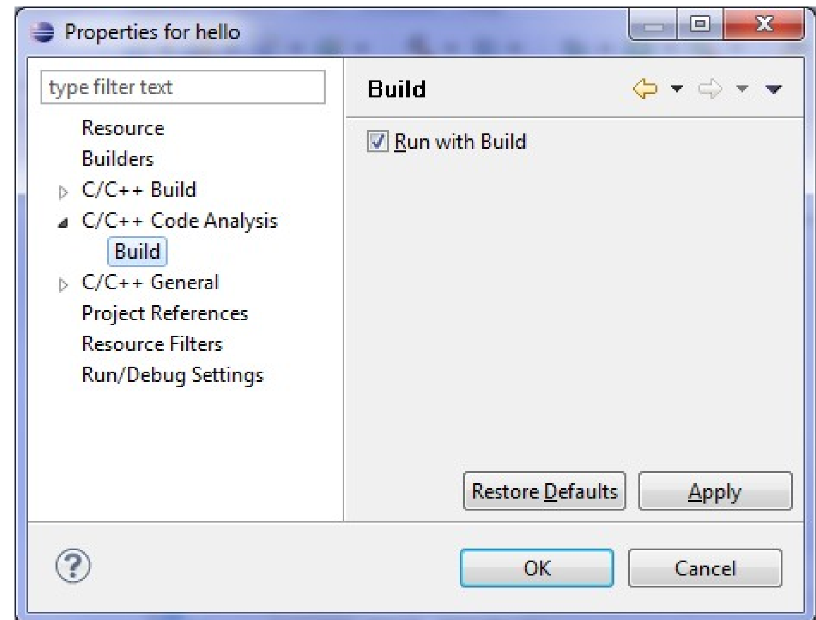
OK Cancel

Codan: Launch Control

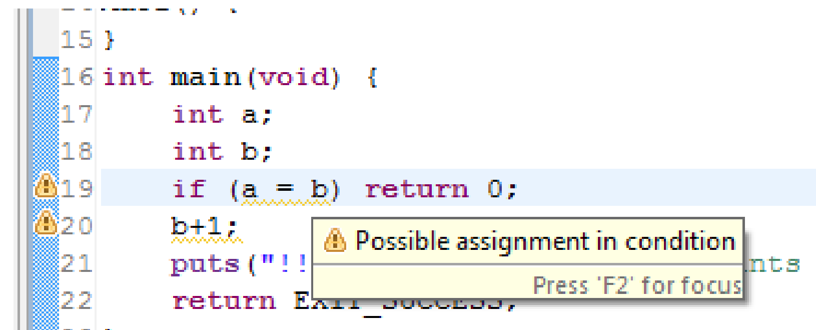
Run on demand from context menu



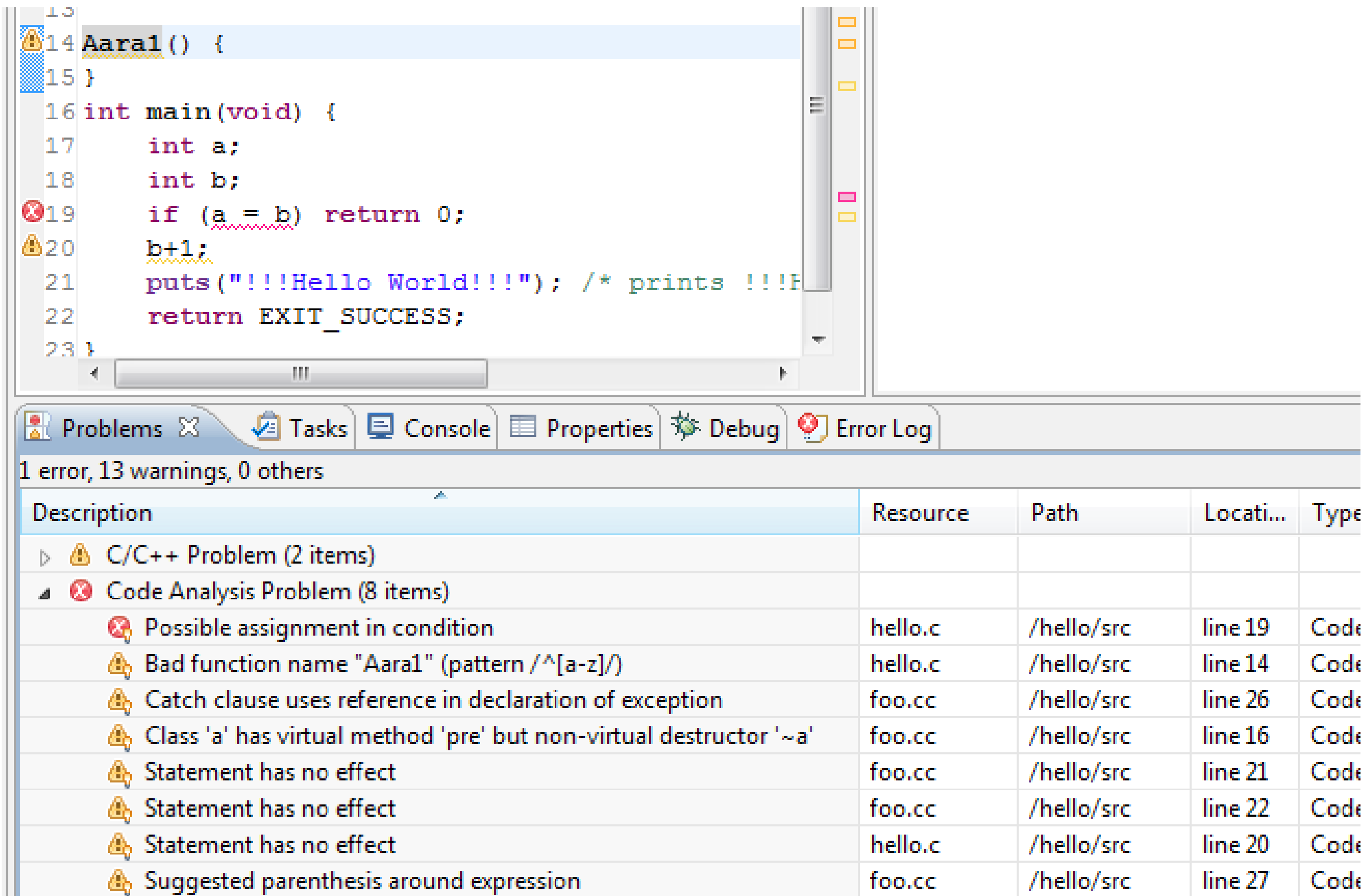
Run with Build



Run as you type



Codan: Problem Markers



Codan: How the write own checkers

Internal Checker

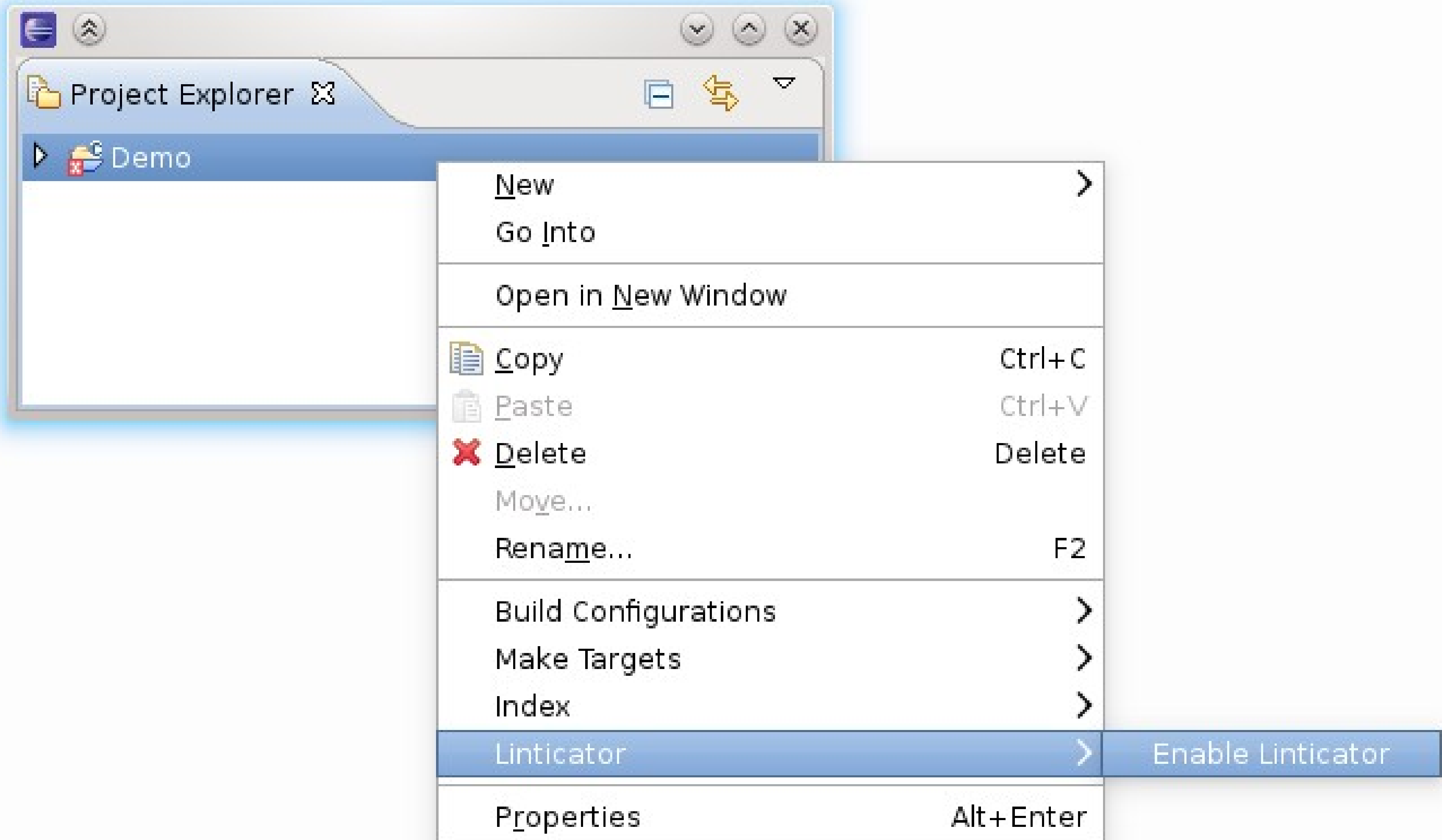
- Problem scope is userdefine (you found e.g. a bug)
- Pick a model to find that problem e.g.
  AST, Index, ControlFlow-, DataFlow-, Call-Graph
- Extend abstract checker for that model + implement check
- Create Extension for finding
- Create Autofix Action ?

External Checker

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

- 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

C/C++ - Demo/src/MissingReturnStatement.cpp - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- Demo
  - Includes
  - src
    - MissingReturnStatement.cpp
  - Debug

MissingReturnStatement.cpp

```
int shouldReturn() {

}
```

Source Annotations

Documentation View

Message: 533

function 'Symbol' should (not) return a value (see Location) -- A return statement within a function (or lack of a return at the end of the function) implies a different return mode than a previous statement at Location (The return mode of a function has to do with whether the function does, or does not, return a value.)

Message Explanations

Problems | Console

0 errors, 1 warning, 1 other

Description	Resource
Warnings (1 item)	
533: function 'shouldReturn(void)' should return a value (see line 2)	MissingRet
Infos (1 item)	

Messages Overview

533: function 'shouldReturn(void)' should return a value (see line 2)

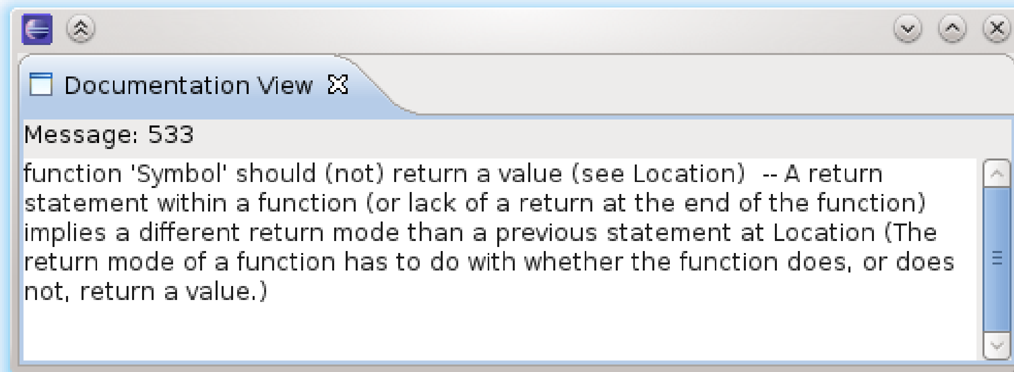
Linticator: Problems View + Message Explanation

Problems

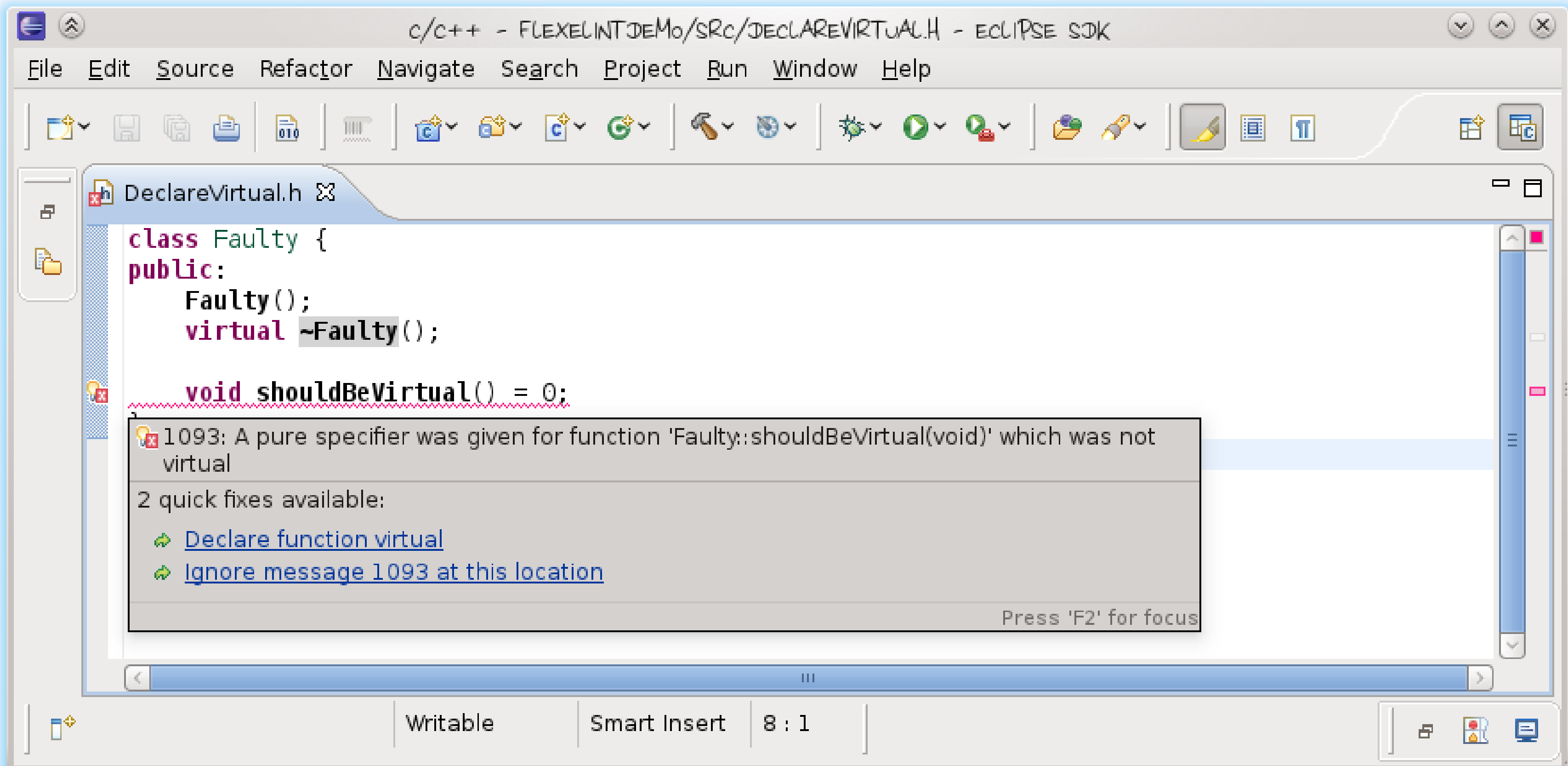
6 errors, 18 warnings, 13 others

Description	Resource	Path
Errors (6 items)		
64: Type mismatch (assignment) (ptrs to void/nonvoid)	TypeMismatch.cpp	/FlexeLi
14: Symbol 'main(void)' previously defined (lin ... clipseA	IncludeTypeInfo.cpp	/FlexeLi
10: Expecting a structure or union	IncludeTypeInfo.cpp	/FlexeLi
1013: Symbol 'name' not a member of class '	IncludeTypeInfo.cpp	/FlexeLi
1055: Symbol 'name' undeclared, assumed t	IncludeTypeInfo.cpp	/FlexeLi

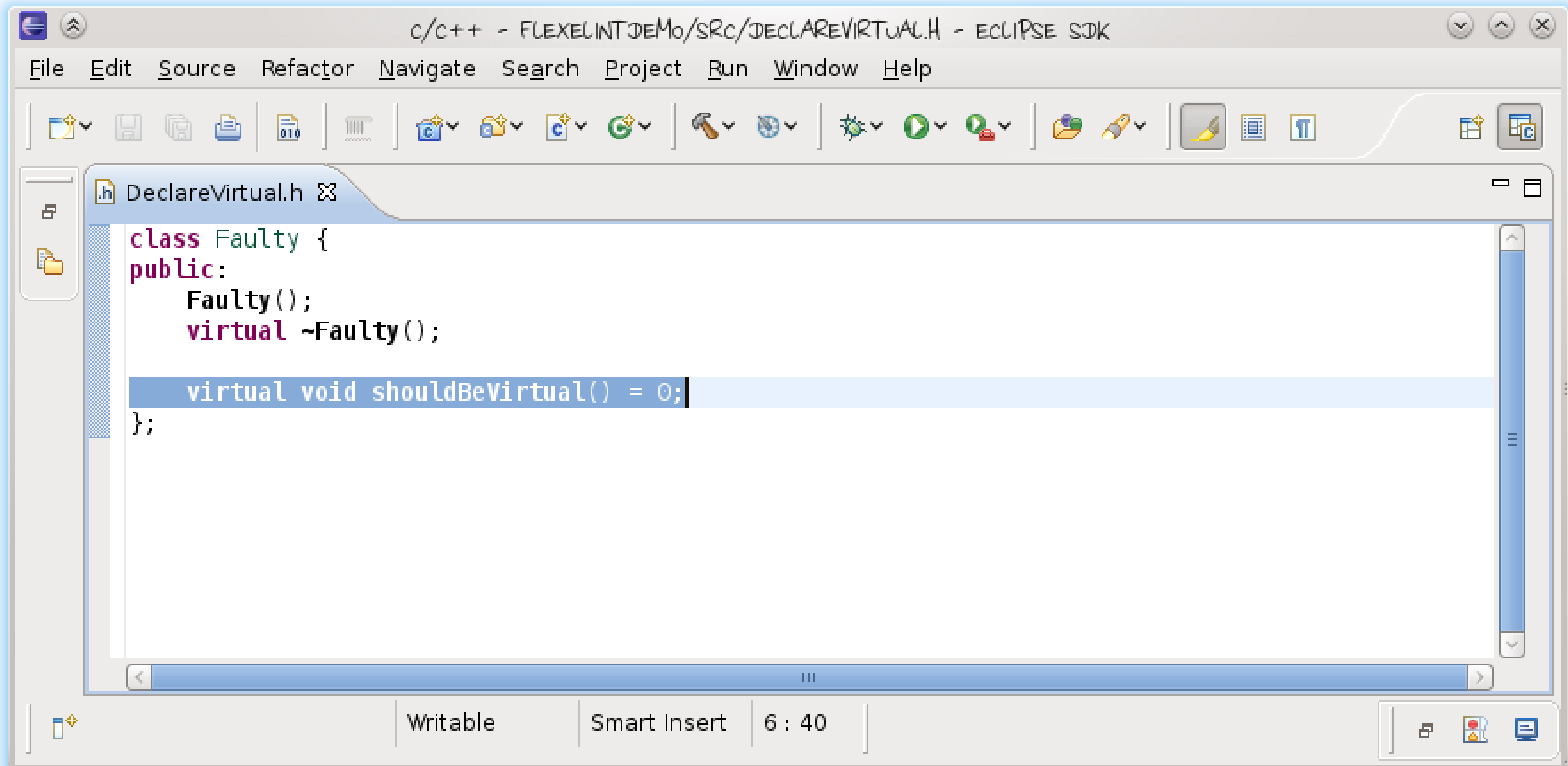
Show Documentation
Go to
Copy Ctrl+C
Delete Delete
Select All Ctrl+A
Show In Shift+Alt+W
Quick Fix Ctrl+1
Properties Alt+Enter



Linticator: Quickfix



Linticator: Quickfix



Linticator: Supress Message

main.c | main.cpp

```
int f() {return 42;}

int main() {
    f();
    retu
}
```

- Ignore message "Ignoring return value" for this functior
- Ignore message 522 at this location
- Ignore message 534 at this location
- Inhibit Lint Messages...

Shows a wiza messages.

INHIBIT MESSAGES

Inhibit Messages

Configure the inhibition options for the messages.

Message	Global	File	Call	Func	Sym
522: Highest operation, function 'f', lacks side-effects	☐				
534: Ignoring return value of function 'f(void)' (compar	☐				☐

☑ Run Linticator after configuring inhibition options.

Finish | Cancel

Includator

eclipseCON™

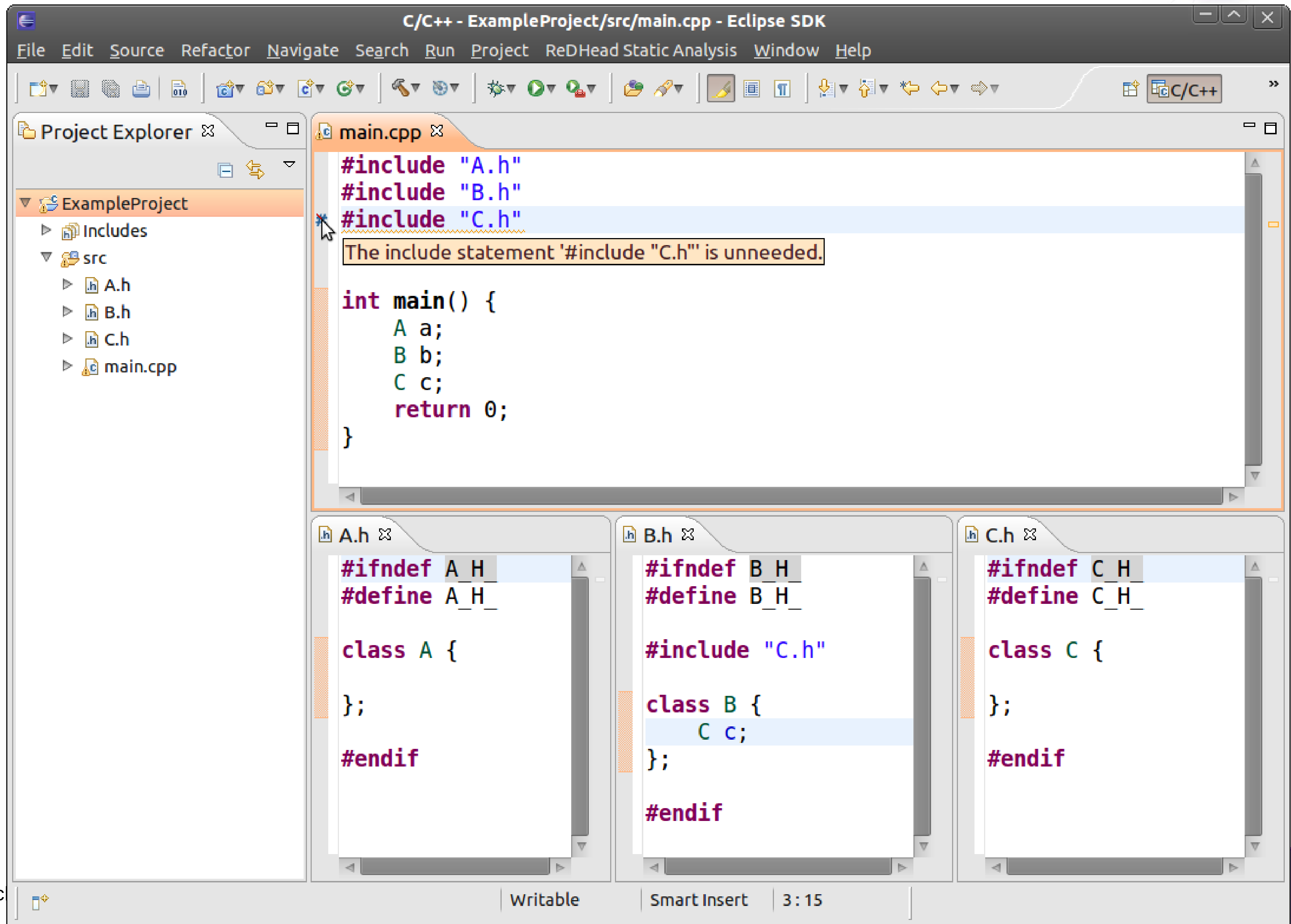
Project of IFS in Rapperswil, CH

- 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

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

```
1   /* Y.h */
2
3 #include "X.h"
4
5 /* more code */
```

```
1   /* X.h */
2
3 class X { };
4
5 /* ... */
```

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 }
```

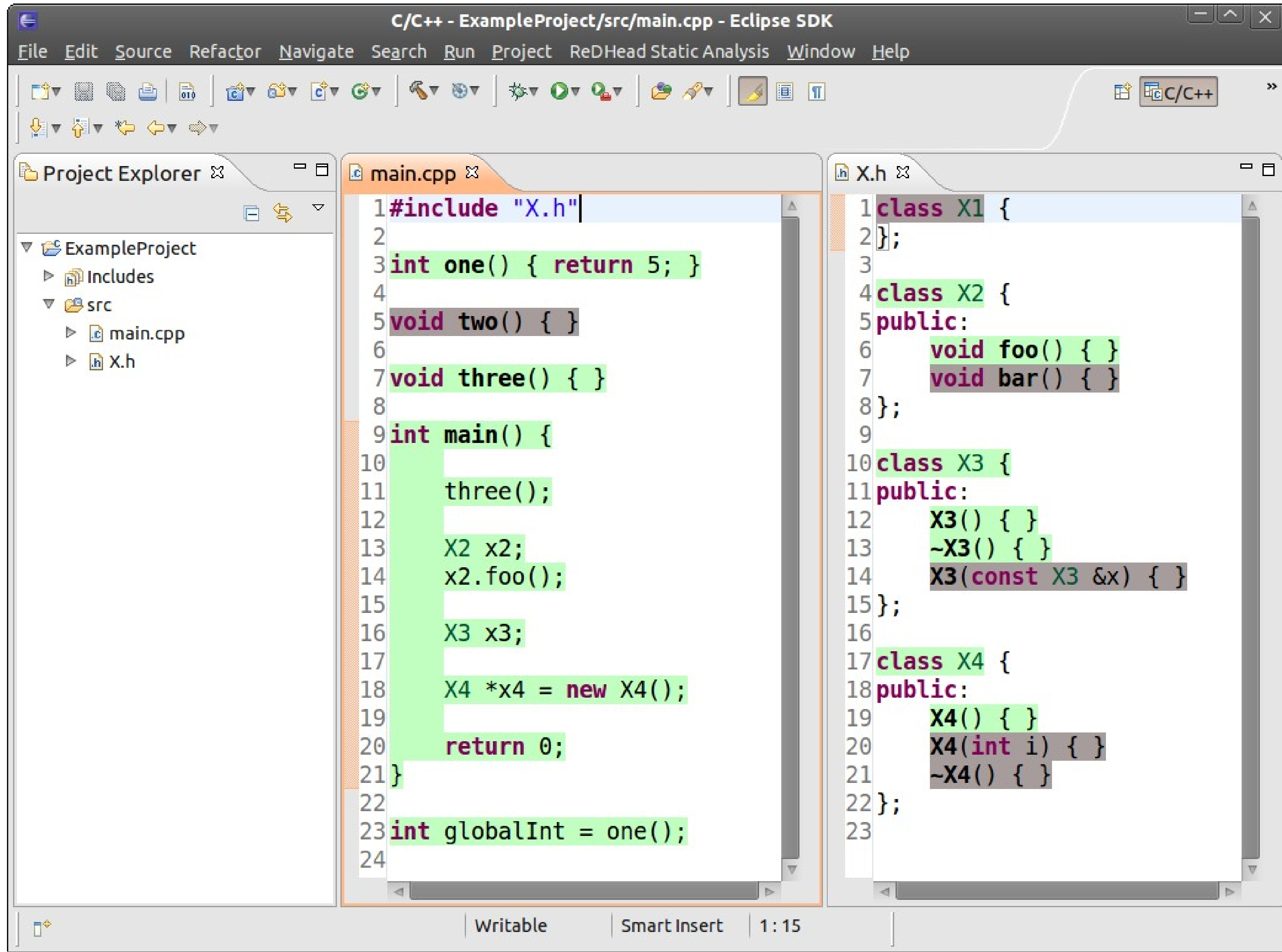
```
1   /* X.h */
2
3 class X { };
```

```
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

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

Linticator: http://www.linticator.ch

Includator: http://includator.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.