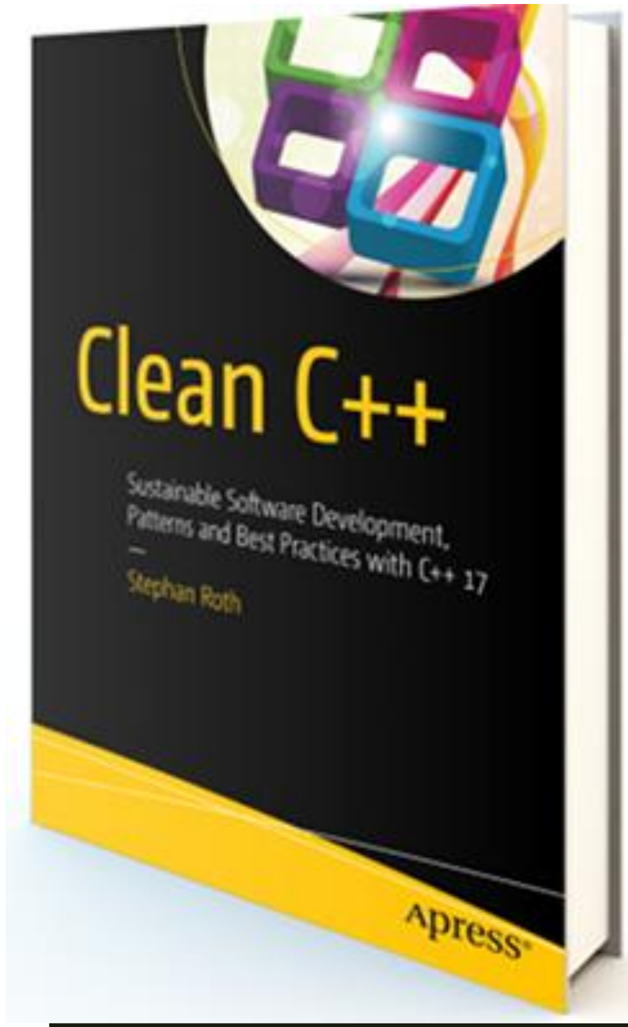
Clean C++ - Sustainable Software Development Patterns and Best Practices with C++17

by Stephan Roth


Clean C++
Sustainable Software Development, Patterns and Best Practices with C++ 17
Stephan Roth
Apress

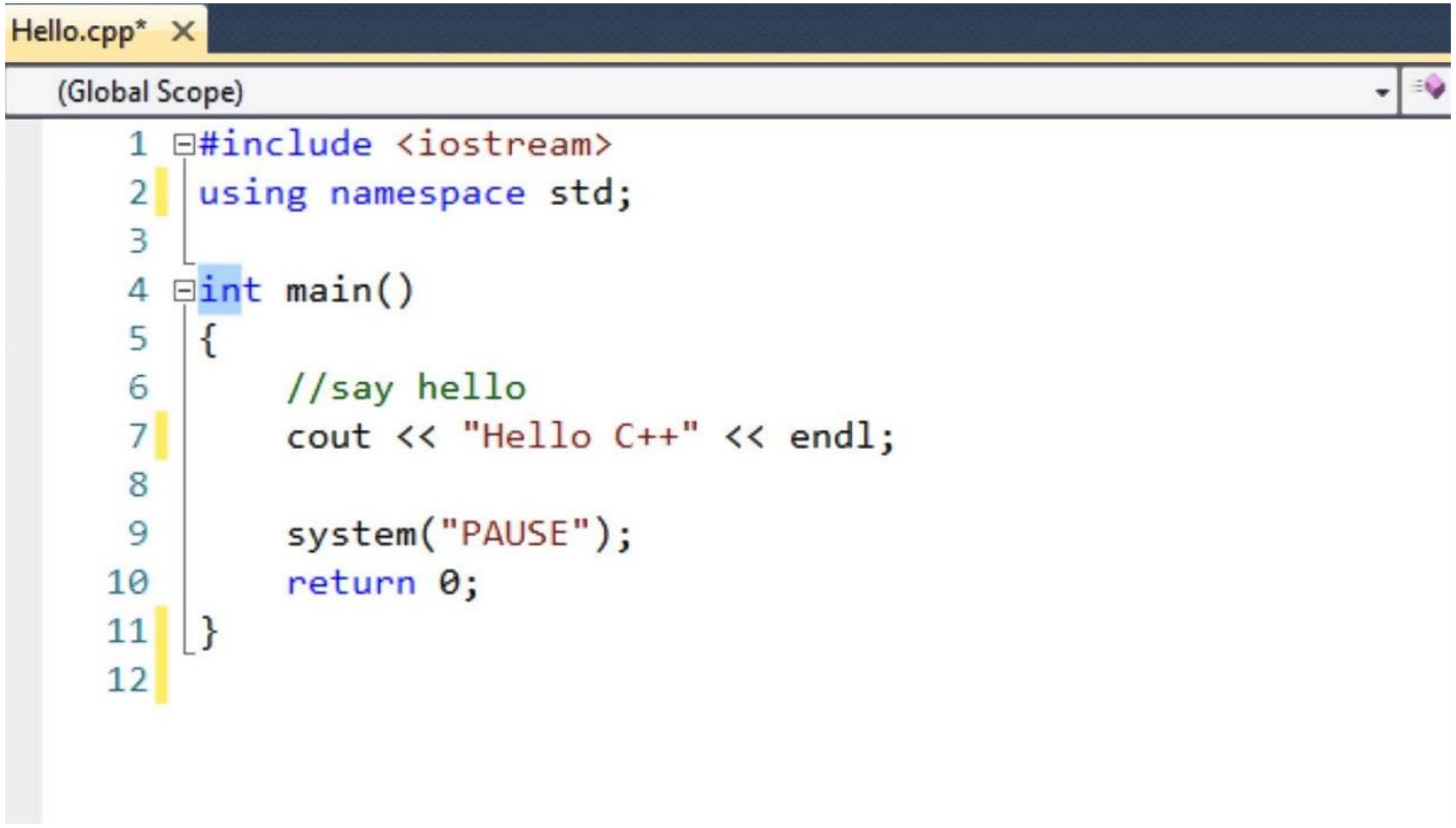
About the Author

Stephan Roth is a consultant and trainer for Systems and Software Engineering . He has developed sophisticated applications, especially for distributed systems with ambitious performance requirements, and graphical user interfaces using C++ and other programming languages.

About Clean C++

Stephan Roth: “My book *Clean C++ – Sustainable Software Development: Patterns and Best Practices With C++17* , which has been published with Apress Media LLC (New York) in September 2017, is about writing maintainable, extensible, and durable software with modern C++.”

Clean C++ is not a C++ primer!



Hello.cpp*

(Global Scope)

```
#include <iostream>
using namespace std;

int main()
{
    //say hello
    cout << "Hello C++" << endl;

    system("PAUSE");
    return 0;
}

```

Contents at a Glance

Chapter 1: Introduction

How it is done is as important as having it done.

—Eduardo Namur

This section includes topics such as:

- ❑ Software Entropy
- ❑ Clean Code
- ❑ C++11 – The Beginning of a New Era
- ❑ Who this book is for
- ❑ Conventions used in this book
- ❑ UML Diagrams

Why C++?

C makes it easy to shoot yourself in the foot. C++ makes it harder, but when you do, you blow away your whole leg!

—Bjarne Stroustrup

Don’t tolerate “broken windows” in your code – fix them!

Chapter 2: Build a Safety Net

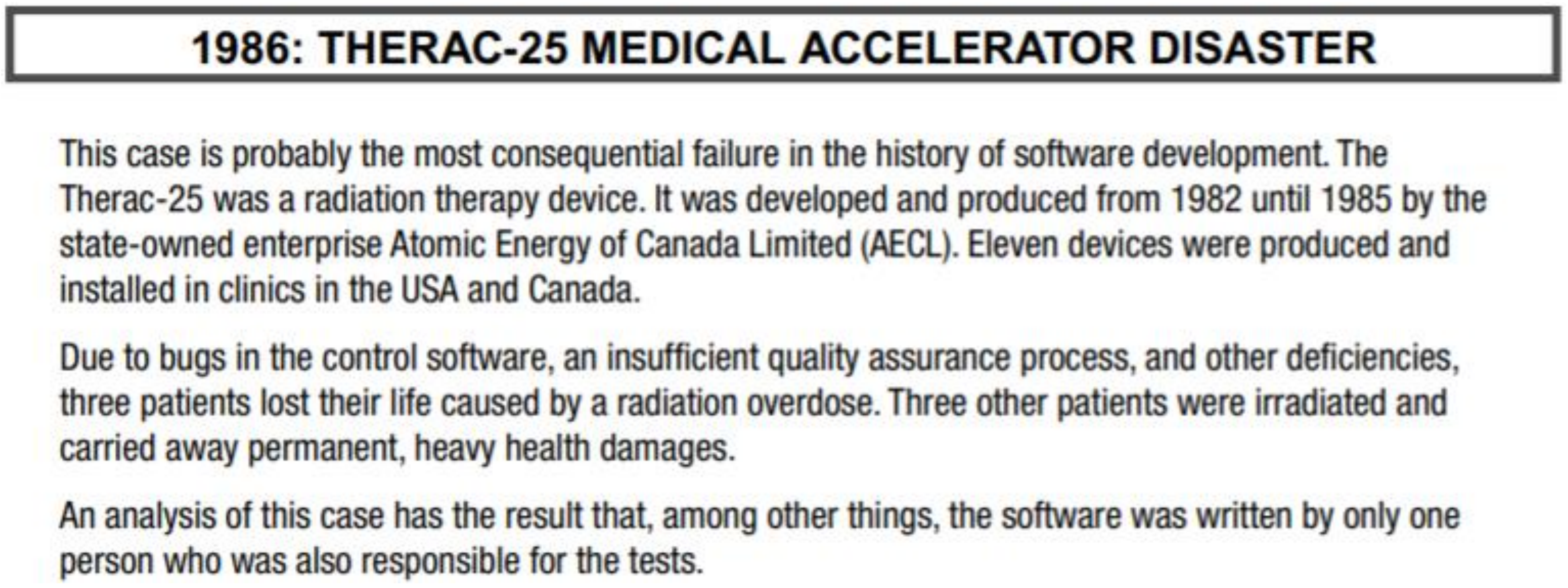
Testing is a skill. While this may come as a surprise to some people it is a simple fact.

—Mark Fewster and Dorothy Graham, Software Test Automation, 1999

This section includes topics such as:

- ❑ The need for testing
- ❑ Introduction into testing
- ❑ Unit tests
- ❑ What about QA?
- ❑ Rules for good unit tests

The Need for Testing



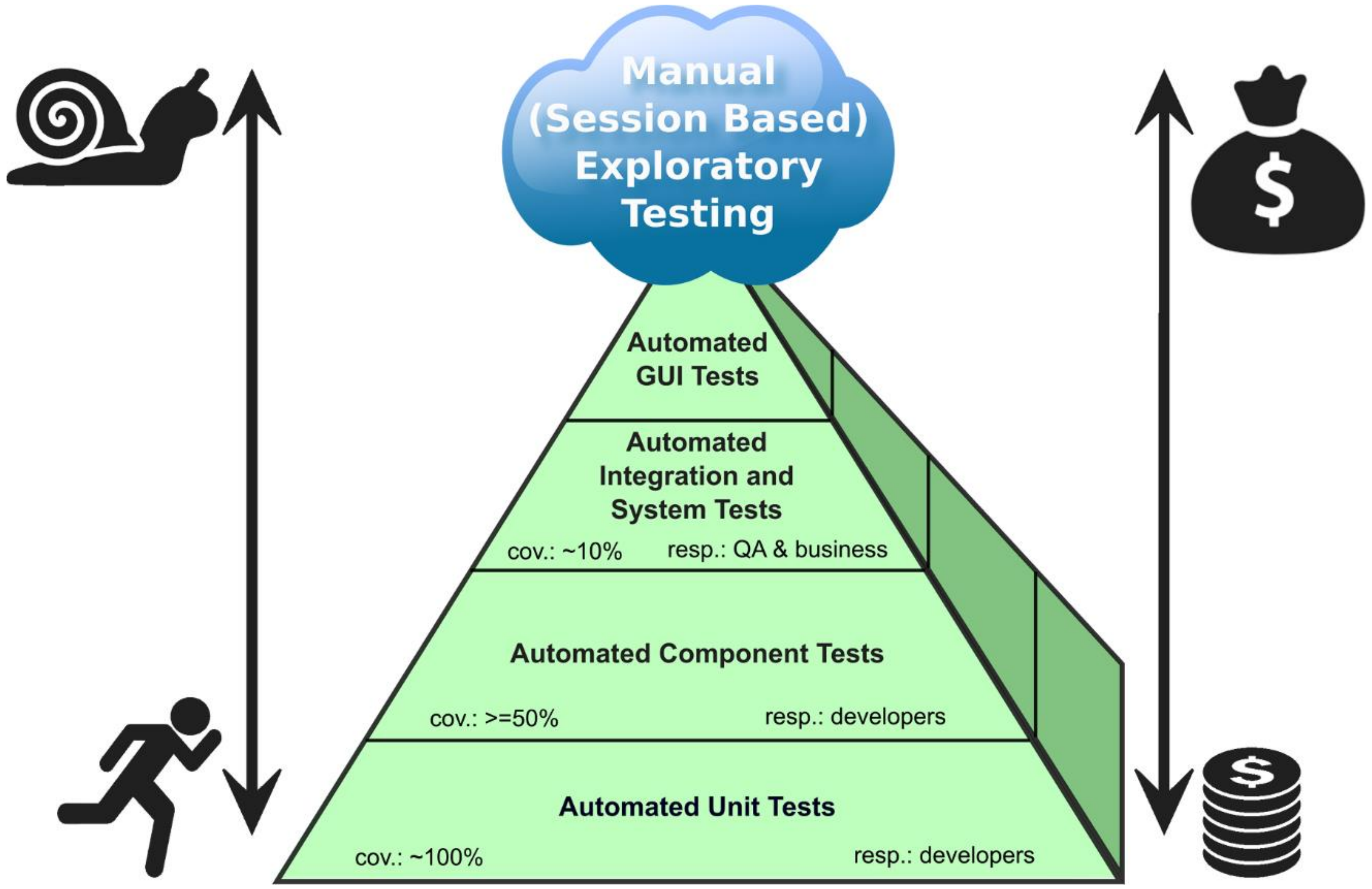
1986: THERAC-25 MEDICAL ACCELERATOR DISASTER

This case is probably the most consequential failure in the history of software development. The Therac-25 was a radiation therapy device. It was developed and produced from 1982 until 1985 by the state-owned enterprise Atomic Energy of Canada Limited (AECL). Eleven devices were produced and installed in clinics in the USA and Canada.

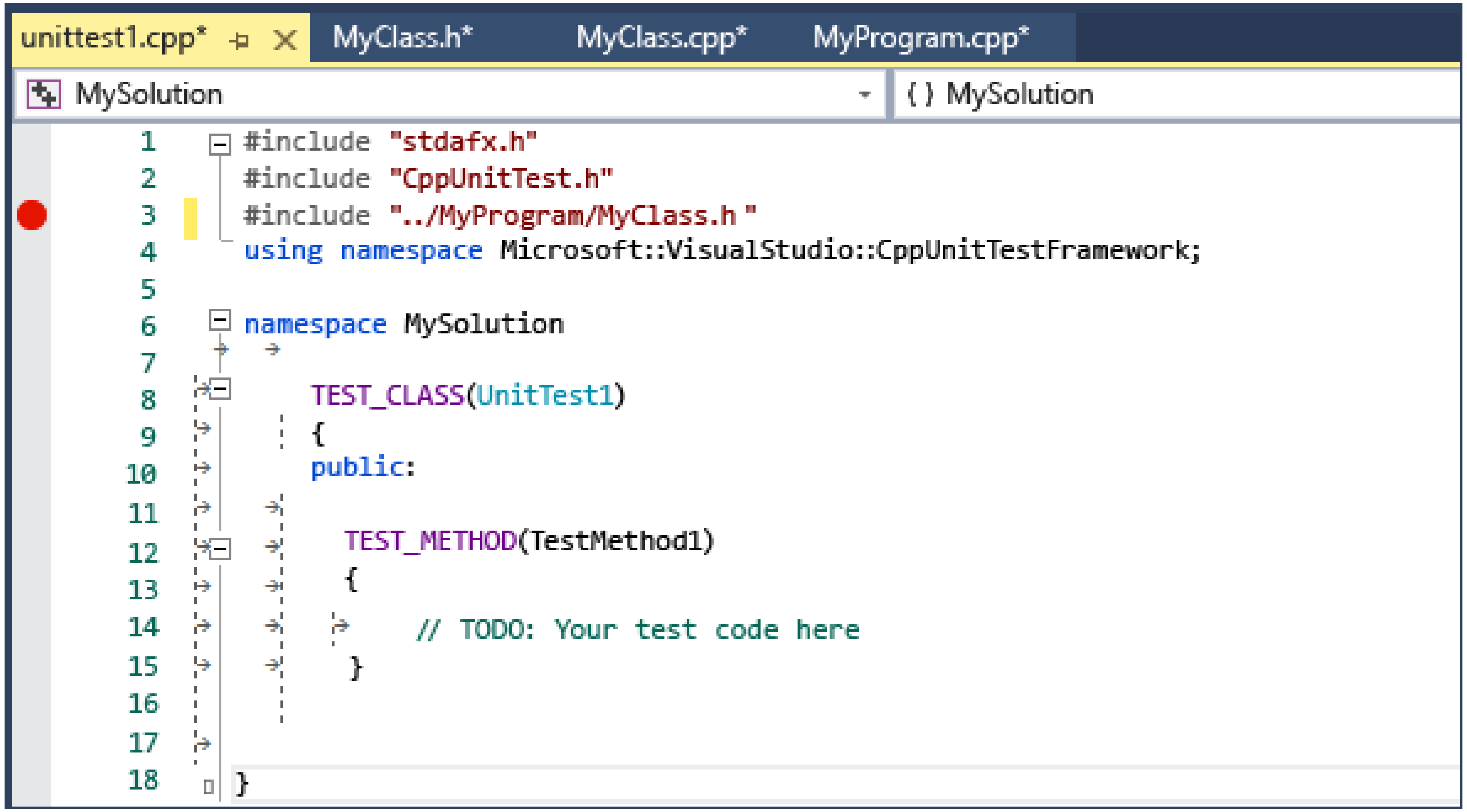
Due to bugs in the control software, an insufficient quality assurance process, and other deficiencies, three patients lost their life caused by a radiation overdose. Three other patients were irradiated and carried away permanent, heavy health damages.

An analysis of this case has the result that, among other things, the software was written by only one person who was also responsible for the tests.

Test Pyramid



Unit Tests



```
#include "stdafx.h"
#include "CppUnitTest.h"
#include "../MyProgram/MyClass.h "
using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MySolution


    TEST_CLASS(UnitTest1)
    {
    public:

        TEST_METHOD(TestMethod1)
        {
            // TODO: Your test code here
        }


}
```

Rules for Good Unit Tests

- Unit test naming

<PreconditionAndStateOfUnitUnderTest>_<TestedPartOfAPI>_<ExpectedBehavior>

- Unit test independence
- One assertion per test
- Independent initialization of Unit test environment
- Exclude getters and setters
- Exclude Third-Party Code
- Exclude External Systems
- Don’t Mix Test Code with Production Code
- Tests Must Run Fast

Chapter 3: Be Principled

I would advise students to pay more attention to the fundamental ideas rather than the latest technology. The technology will be out-of-date before they graduate. Fundamental ideas never get out of date.

—David L. Parnas

This section includes topics such as:

- ❑ KISS
- ❑ YAGNI
- ❑ DRY
- ❑ Information hiding
- ❑ Strong cohesion
- ❑ Loose coupling
- ❑ PLA
- ❑ The Boy Scout Rule

- ✓ KISS - Keep it simple and stupid
- ✓ YAGNI - You Aren’t Gonna Need It!
- ✓ DRY - Don’t repeat yourself!
- ✓ The Boy Scout Rule - Always leave the campground cleaner than you found it.

Chapter 4: Basics of Clean C++

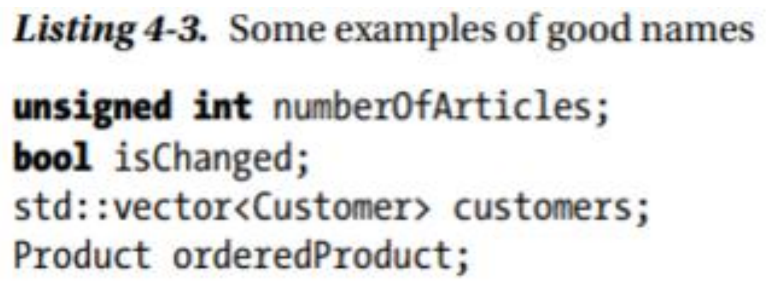
Programs must be written for people to read, and only incidentally for machines to execute.

—Hal Abelson and Gerald Jay Sussman, 1984

This section includes topics such as:

- ❑ Good names
- ❑ Comments
- ❑ Functions
- ❑ About Old C-style in C++ Projects

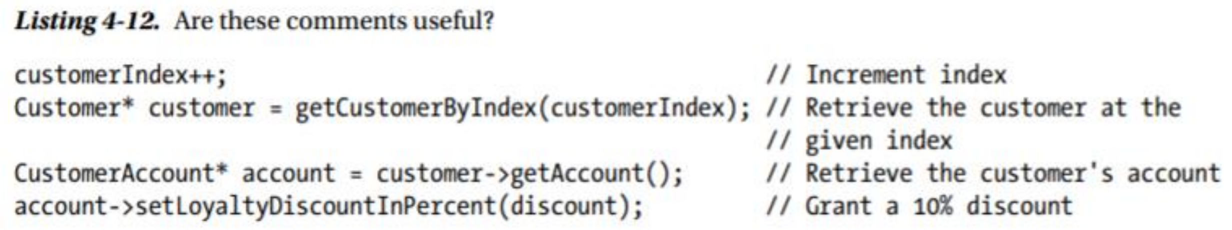
- Good names



Listing 4-3. Some examples of good names

```
unsigned int numberOfArticles;
bool isChanged;
std::vector<Customer> customers;
Product orderedProduct;
```

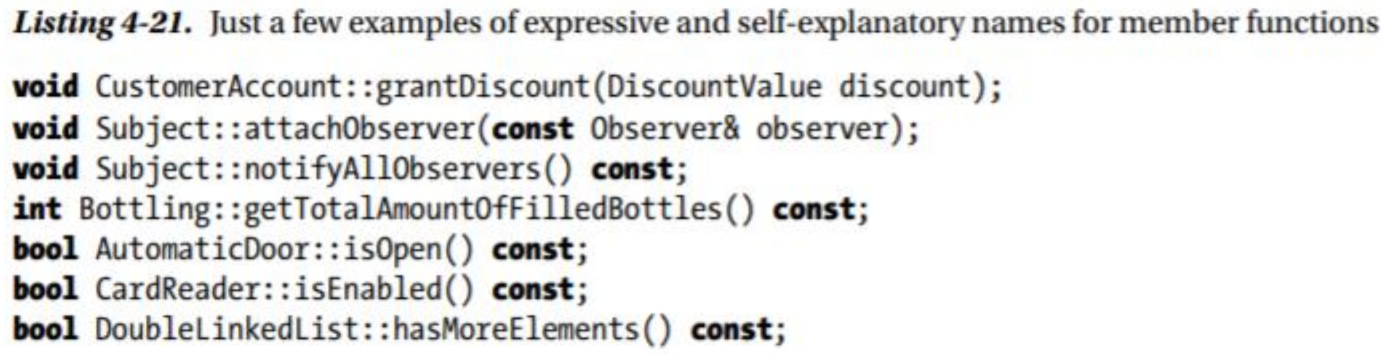
- Comments



Listing 4-12. Are these comments useful?

```
customerIndex++;                                          // Increment index
Customer* customer = getCustomerByIndex(customerIndex); // Retrieve the customer at the
                                                          // given index
CustomerAccount* account = customer->getAccount();       // Retrieve the customer's account
account->setLoyaltyDiscountInPercent(discount);          // Grant a 10% discount
```

- Functions



Listing 4-21. Just a few examples of expressive and self-explanatory names for member functions

```
void CustomerAccount::grantDiscount(DiscountValue discount);
void Subject::attachObserver(const Observer& observer);
void Subject::notifyAllObservers() const;
int Bottling::getTotalAmountOfFilledBottles() const;
bool AutomaticDoor::isOpen() const;
bool CardReader::isEnabled() const;
bool DoubleLinkedList::hasMoreElements() const;
```

Thank you for your attention!