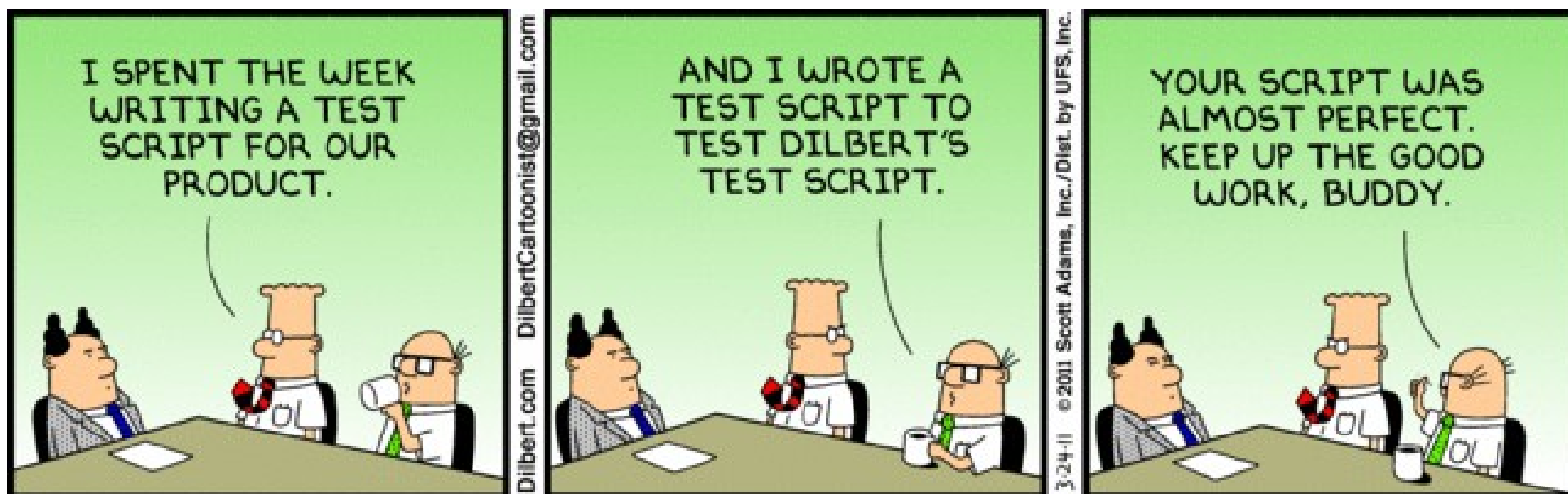


# Test-driven development in R

Richèl Bilderbeek

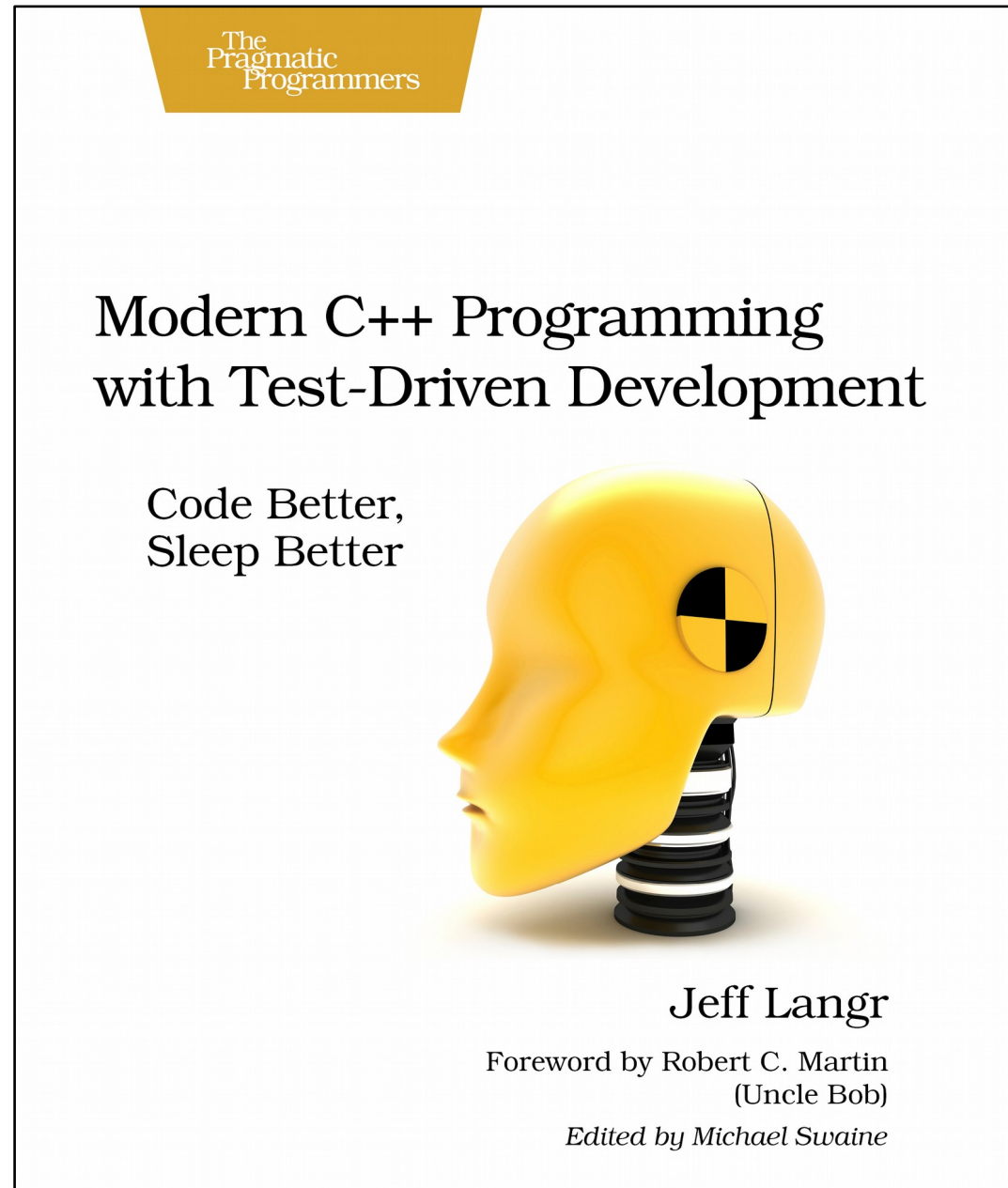


<http://github.com/richelbilderbeek/PresentationsAboutR>



# What is test-driven development?

- Letting tests being the guide in developing new code
- A workflow for growing high-quality code



# Why tests?

- High-quality code
- Increase productivity
- Improves software architecture
- Documentation
- Collaboration

9/9

0800 Antan started  
1000 " stopped - antan ✓ { 1.2700 9.037 847 025  
1300 (032) MP-MC 1.98267000 9.037 846 995 correct  
2.130476415 (033) PRO 2 2.130476415 4.615925059(-2)  
correct 2.130676415  
Relays 6-2 in 033 failed spiral speed test  
in relay " 11.000 test.

1100 Started Cosine Tape (Sine check)  
1525 Started Multi-Adder Test.

1545 Relay #70 Panel F  
(moth) in relay.

First actual case of bug being found.

1630 Antan started.  
1700 closed down.

Relay 2145  
Relay 3370

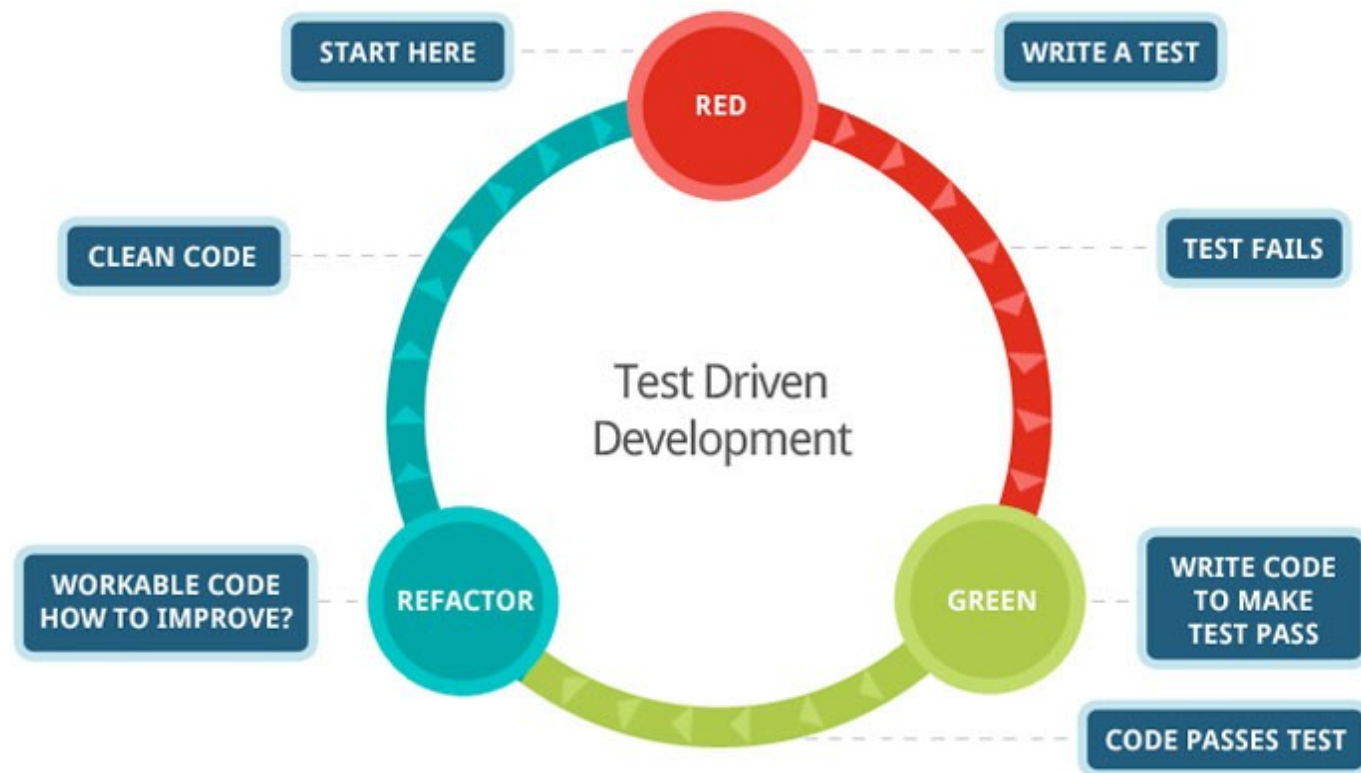
# Example

```
is_prime <- function(x) {  
  # check input arguments  
  # calculation  
  # return TRUE or FALSE  
}
```



# Developing is\_prime

- Cycle test-driven development:
  - Red: write a test that fails
  - Green: pass the test
  - Refactor: improve the new code, clean up mess, check in code





# Development is\_prime

- As small steps as possible
- One step at a time



# Red: write a test that fails

- Create the testing architecture, in console:

```
devtools::use_test("is_prime")
```

- Create a test that fails, in file 'test-is\_prime.R':

```
context("is_prime")  
  
test_that("is_prime works", {  
  expect_equal(is_prime(2), TRUE)  
})
```

# Green: pass the test

```
#' Determines if the input is prime
#' @param x input
#' @return TRUE or FALSE
#' @export
is_prime <- function(x) {
  return (TRUE)
}
```



# Refactor

- Checking in

```
git add --all :/
```

```
git commit -m
```

```
    "is_prime: 2 is prime"
```



## Red: write a test that fails

```
test_that("is_prime works", {  
  # Previous code  
  expect_equal(is_prime(4), FALSE)  
})
```

# Green: pass the test

```
is_prime <- function(x) {  
  if (x == 2) return(TRUE)  
  for (i in seq(2, x - 1)) {  
    if (x %% i == 0) {  
      return(FALSE)  
    }  
  }  
  return(TRUE)  
}
```

# Refactor

- Check in

```
git add --all :/
```

```
git commit -m
```

```
    "is_prime: 4 is not prime"
```



## Red: write a test that fails

```
test_that("is_prime works", {  
  # Previous code  
  expect_equal(is_prime(1), FALSE)  
})
```

# Green: pass the test

```
is_prime <- function(x)
{
  if (x <= 1) return(FALSE)
  if (x == 2) return(TRUE)
  for (i in seq(2, x - 1)) {
    if (x %% i == 0) { return(FALSE) }
  }
  return(TRUE)
}
```

# Refactor

- Check in

```
git add --all :/
```

```
git commit -m
```

```
    "is_prime:  $\leq 1$  is not prime"
```





## Red: write a test that fails

```
test_that("is_prime works", {  
  # Other test  
  expect_error(  
    is_prime("Hello"),  
    "testit: input must be numeric"  
  )  
})
```

# Green: pass the test

```
is_prime <- function(x)
{
  if (!is.numeric(x)) {
    stop("is_prime: input should be numeric")
  }
  # Other code
}
```

# Refactor

- Check in

```
git add --all :/
```

```
git commit -m
```

```
    "is_prime: input must be numeric"
```



# Red: write a test that fails

```
test_that("is_prime works", {  
  # Other tests  
  expect_error(  
    is_prime(c(1,2,3)),  
    "is_prime: input should be one number"  
  )  
})
```

# Green: pass the test

```
is_prime <- function(x)
{
  if (!is.numeric(x)) {
    stop("is_prime: input should be numeric")
  }
  if (length(x) != 1) {
    stop("is_prime: input should be one number")
  }
  # Other code
}
```

# Refactor

- Check in

```
git add --all :/
```

```
git commit -m
```

```
    "is_prime: input one number"
```



# Red: write a test that fails

```
test_that("is_prime works", {  
  # Other tests  
  expect_equal(  
    is_prime(c(1, 2, 3)),  
    c(FALSE, TRUE, TRUE)  
  )  
})
```



# Green: pass the test

```
#' Calculates if the input is prime
#' @param x input values
#' @return vector of TRUEs and/or FALSEs
#' @export
is_prime <- function(x) {
  r <- rep(TRUE, times = length(x))
  for (i in seq(1, length(x))) {
    r[i] <- is_prime_single(x[i])
  }
  r
}
```

```
is_prime_single <- function(x) {
  # What used to be is_prime
}
```

# Refactor

- Check in

```
git add --all :/
```

```
git commit -m
```

```
    "is_prime: allow multiple inputs"
```



# Final is\_prime

```
#' Calculates if the input is prime
#' @param x input values
#' @return vector of TRUEs and/or FALSEs
#' @export
is_prime <- function(x) {
  r <- rep(TRUE, times = length(x))
  for (i in seq(1, length(x))) {
    r[i] <- is_prime_single(x[i])
  }
  r
}

#' Calculates if the input is prime
#' @param x input value
#' @return TRUE or FALSE
#' @export
is_prime_single <- function(x) {
  if (!is.numeric(x)) {
    stop("is_prime: input must be integer")
  }
  if (length(x) != 1) {
    stop("is_prime_single: input must be a single value")
  }
  if (x == 2) return(TRUE)
  for (i in seq(2, x - 1)) {
    if (x %% i == 0) return(FALSE)
  }
  return(TRUE)
}
```

# Complete test suite

```
context("is_prime")

test_that("is_prime works", {
  expect_equal(is_prime(2), TRUE)
  expect_equal(is_prime(4), FALSE)
  expect_equal(is_prime(1), FALSE)
  expect_equal(is_prime(0), FALSE)
  expect_equal(is_prime(-1), FALSE)
  expect_error(is_prime("hello"), "is_prime: input must be integer")
  expect_equal(is_prime(c(1,2,3)), c(FALSE, TRUE, TRUE))
})
```

# Too complex to write a test for?

- Do test simple cases as usual
- Use vignettes to convince the reader it works for more complex input

```
devtools::use_vignette("demo")
```

# Vignette

```
---  
title: "is_prime demonstration"  
author: "Richel Bilderbeek"  
date: "`r Sys.Date()`"  
output: rmarkdown::html_vignette  
vignette: >  
  %\VignetteIndexEntry{Richel Bilderbeek}  
  %\VignetteEngine{knitr::rmarkdown}  
  %\VignetteEncoding{UTF-8}
```

```
---
```

Density of primes should decrease:

```
```{r}  
library(my_is_prime)  
xs <- seq(1,100)  
ys <- is_prime(xs)  
plot(xs, ys)  
```
```

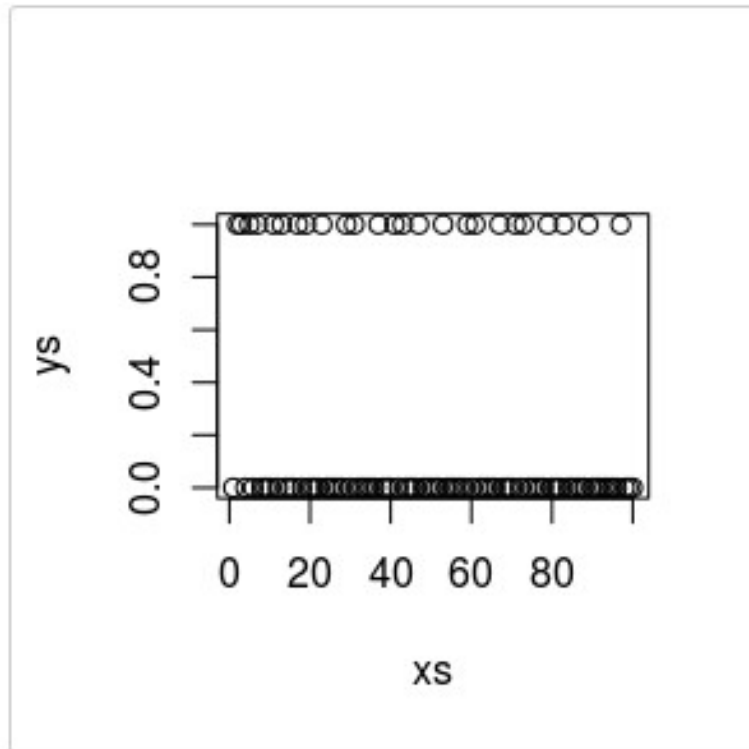
# is\_prime demonstration

*Richel Bilderbeek*

**2016-04-17**

Density of primes should decrease:

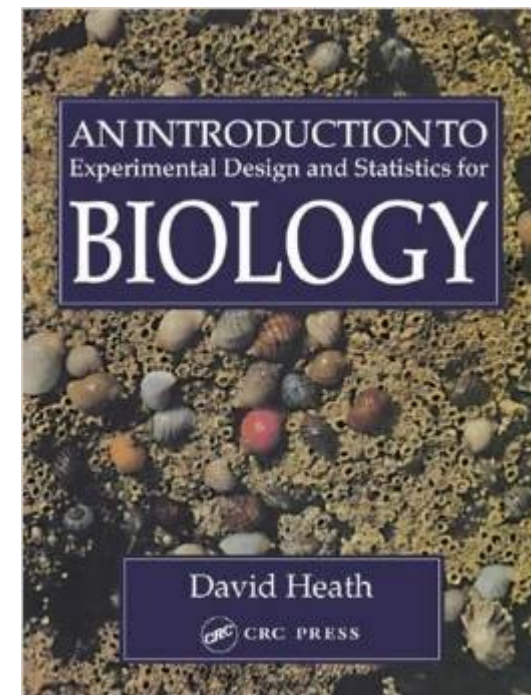
```
library(testr)
xs <- seq(1,100)
ys <- xs
for (i in seq(1,length(ys))) { ys[i] <- is_prime(ys[i]) }
plot(xs, ys)
```





# Collaboration

- Tests can be used as a mean to let others contribute
- Example:
  - I implemented a Wilcoxon's signed rank test, especially a function to get the rank of each value, following Heath
  - Later, Senbong G contacted me and helped us fix my function
- In this example, syntax is not important



# C++ tests

```
values    = { 10.0, 20.0, 30.0 };  
expected = {  1.0,  2.0,  3.0 };  
results = GetRanks(values);  
assert(expected == results);
```

```
values    = { 30.0, 10.0, 20.0 };  
expected = {  3.0,  1.0,  2.0 };  
results = GetRanks(values);  
assert(expected == results);
```

# C++ tests

```
//From Heath, page 263
values    = { 0.6, 1.4, 4.0, 13.0, 14.5, 9.4, 11.4, 12.6, 4.0 };
expected  = { 1.0, 2.0, 3.5, 8.0, 9.0, 5.0, 6.0, 7.0, 3.5 };
results = GetRanks(values);
// Heath, page 263, no zero value
assert(expected == results);
```

```
//From Heath, page 263, now with zero added
values    = { 0.6, 1.4, 0.0, 4.0, 13.0, 14.5, 9.4, 11.4, 12.6, 4.0 };
expected  = { 1.0, 2.0, 0.0, 3.5, 8.0, 9.0, 5.0, 6.0, 7.0, 3.5 };
results = GetRanks(values);
// Heath, page 263, with zero value
assert(expected == results);
```

# Email from Senbong G

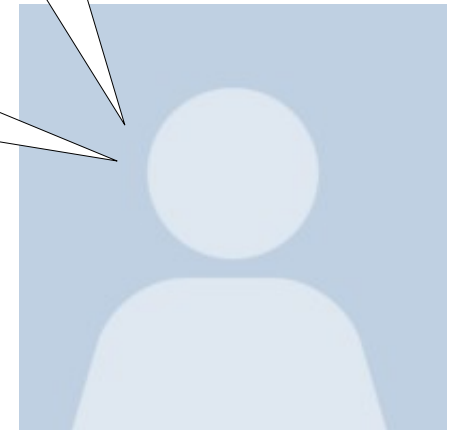
I think I found a bug

How can that be?  
Can you convince me with a test?

Sure, the tests on the  
next slide all fail



Richel Bilderbeek



Senbong G

# C++ tests

```
values    = { 1.0, 1.0, 1.0, 2.0, 2.0, 3.0, 3.0, 3.0};  
expected = { 2.0, 2.0, 2.0, 4.5, 4.5, 7.0, 7.0, 7.0};  
results = GetRanks(values);  
assert(expected == results);
```

# Email from Senbong G

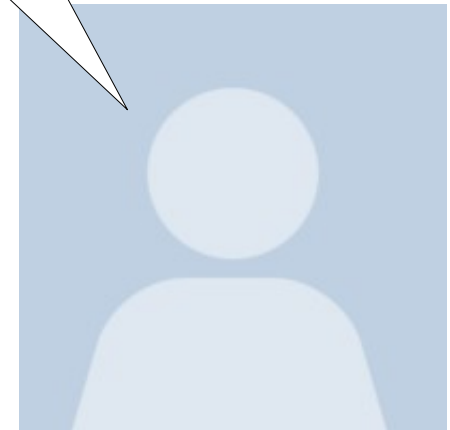
Blimey, you're right!  
I'll fix it within a week

No need to,  
I already did,  
here is the code

Thanks!




Richel Bilderbeek







Senbong G


# Eternal glory



ExploreGistBlogHelp

 richelbilderbeek +



 richelbilderbeek / **ProjectRichelBilderbeek**

Unwatch 4Unstar 8Fork 3

<> Code 1

! Issues 1

Languages

C++ 1

Search all of GitHub

Search

**Test/CppWillcoxonsSignedRankTest/main.cpp** C++  
Showing the top two matches. Last indexed on 2 Sep 2014.

```
154     return (lowest != max ? lowest : above);
155 }
156
157
158 //Thanks to Senbong G for detecting and fixing a bug in this code
159 const std::vector<double> GetRanks(const std::vector<double> &v)
160 ...
290     assert(AreAboutEqual(expected, results) && "Heath, page 263, with zero value");
291 }
292 {
293     //Thanks to Senbong G for adding this test
294     const std::vector<double> values = { 1.0, 1.0, 1.0, 2.0, 2.0, 3.0, 3.0, 3.0};
```



# Conclusion

- Test-driven development
  - uses a systematic approach
  - has guidelines for a lower and upper limit of tests
  - facilitates collaboration