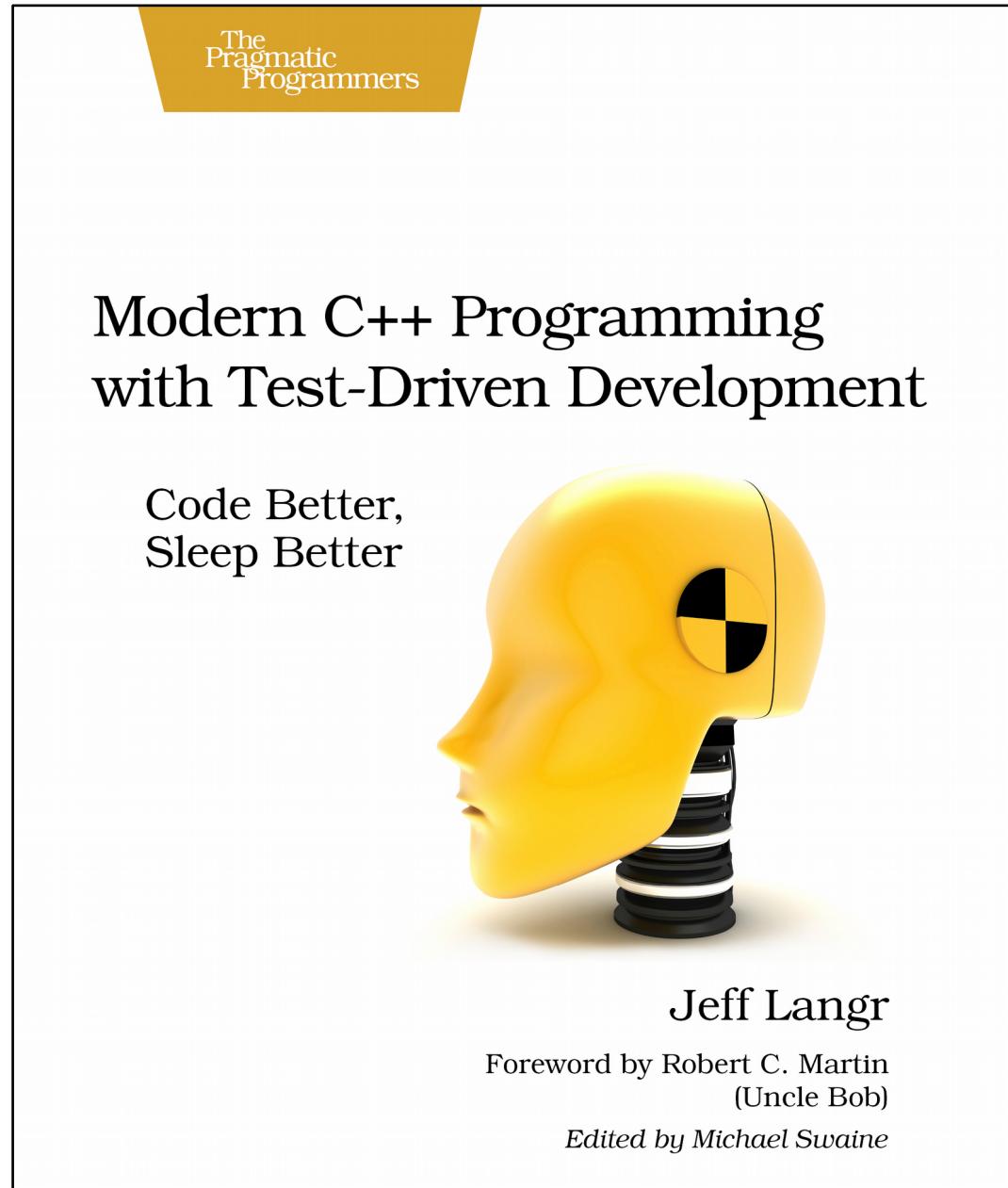


Test-driven development

Richèl Bilderbeek 

What is test-driven development?

- Developing code with the right amounts of tests



Why tests?

- Goal: code without bugs
- Untested code easily develops an error
- Code that cannot be tested has more errors

17/26
9/9

0800 Antran started
1000 " stopped - antran ✓ { 1.2700 9.037 847 025
13'00 (033) MP-MC 1.982 1.47000 9.037 846 995 corwck
033 PRO 2 2.130476415
corwck 2.130676415
Relays 6-2 in 033 failed special sped test
in relay " 10.000 test .
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multi Adder Test.

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

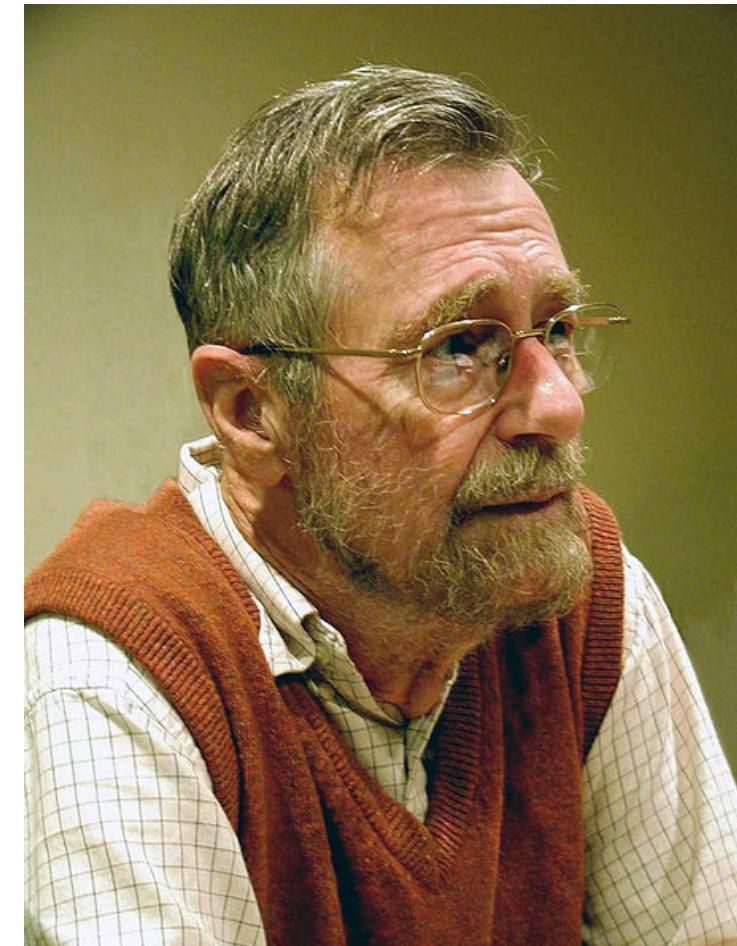
1600 Antran started.
1700 close down.

Relay 2145
Relay 3370

First actual case of bug being found.

How much tests?

- “Testing can be used to show the presence of errors, never their absence!” Edsger Dijkstra
- Too few: many errors are not discovered
- Too much: many correctnesses are checked multiple times
- What is the balance?



What is an error?

- Logic error
 - The programmer makes an incorrect reasoning
 - Example: a division by zero
- Runtime error
 - Hardware encounters a limit
 - Example: memory is full

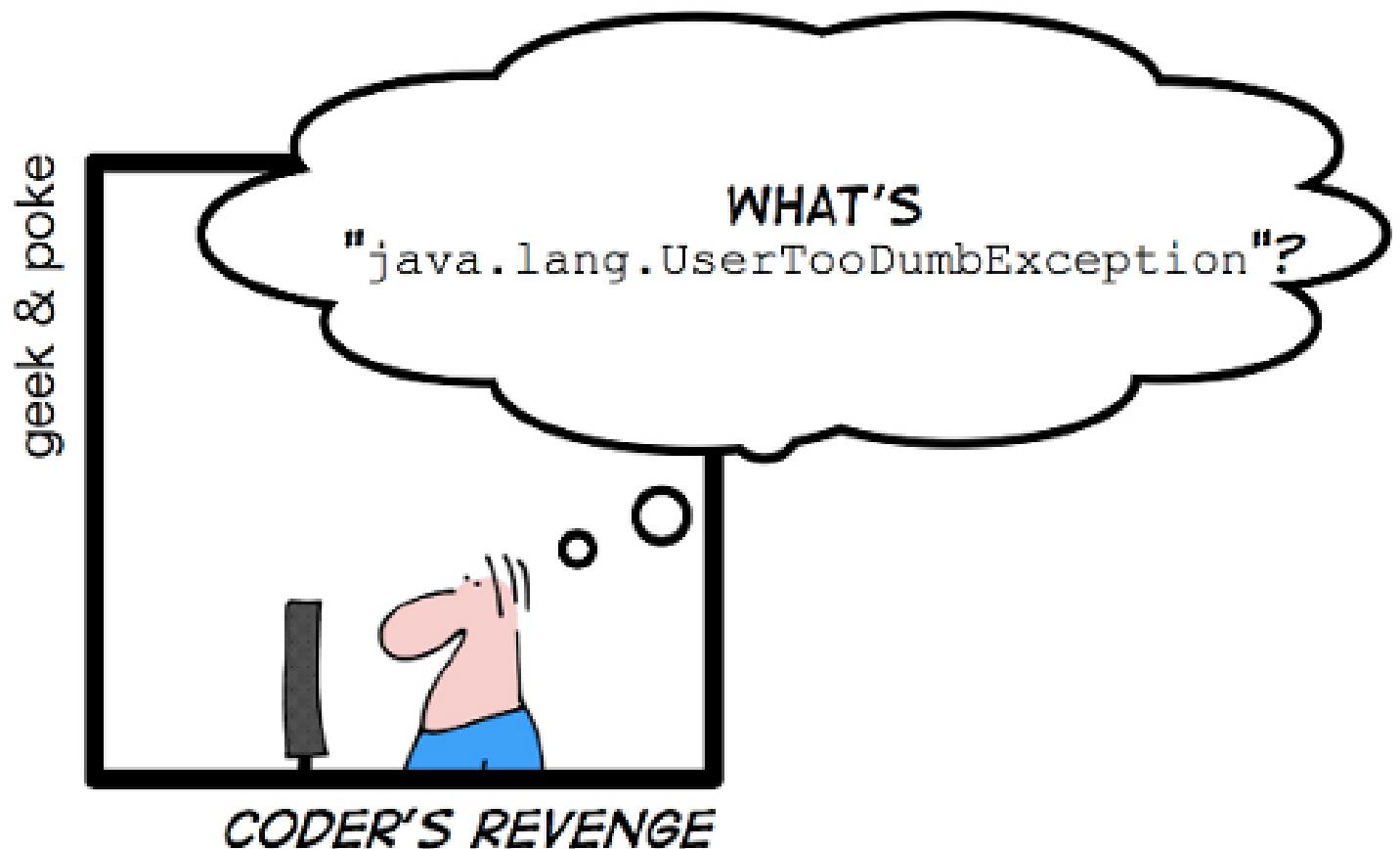
SIMPLY EXPLAINED



NullPointerException

What is not an error?

- The user gives incorrect/nonsense input
- Example: a command that does not exist

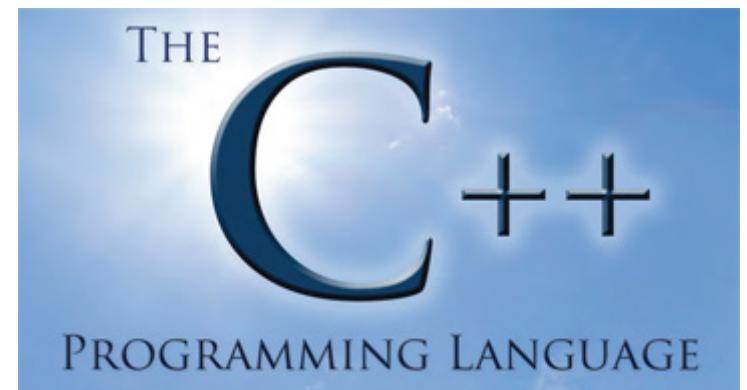


Logic error

- The programmer makes an incorrect reasoning
- Example: a function that returns the wrong answer
- Test:
 - call the function with different arguments
 - see if it returns the expected results

Example

```
bool IsPrime(const int x)
{
    /* calculation */
    /* return true or false */
}
```

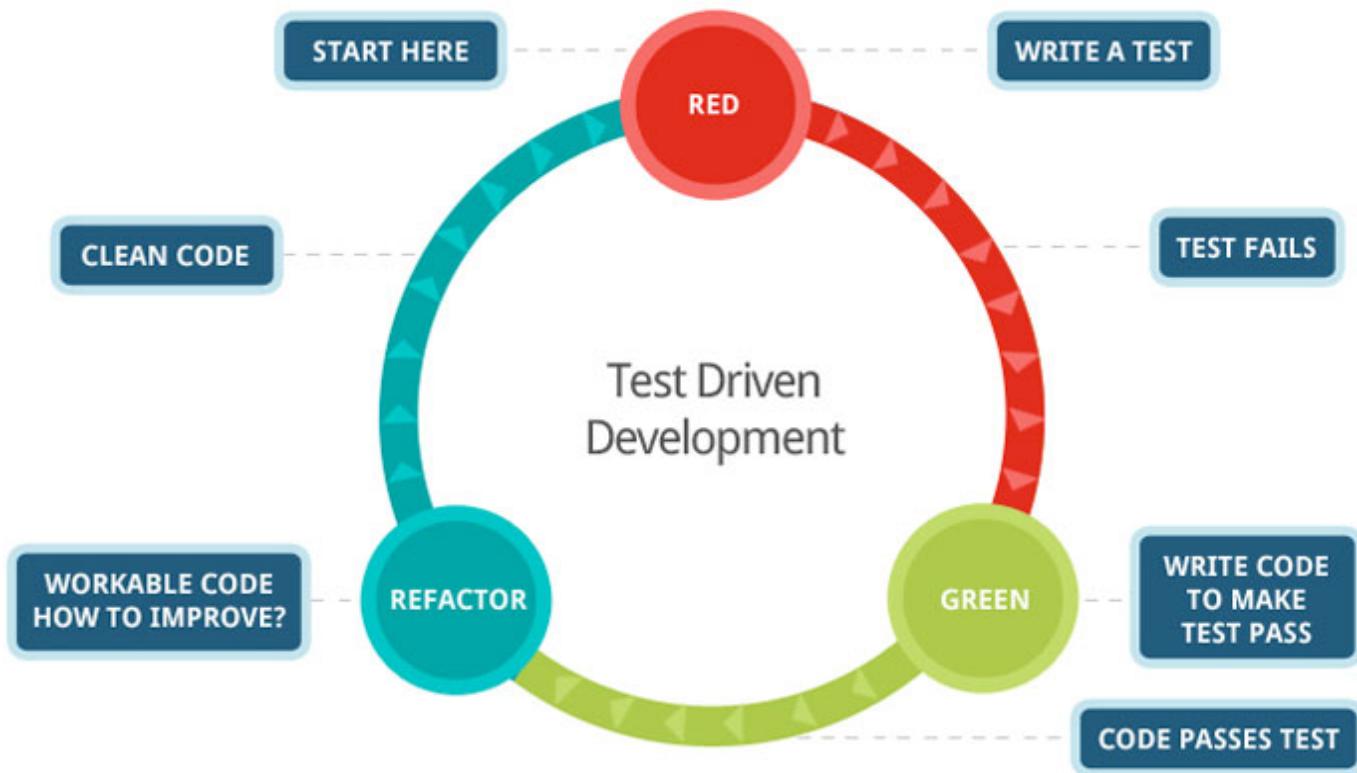


```
IsPrime <- function(x)
{
    # calculation
    # return TRUE or FALSE
}
```



Developing IsPrime

- 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

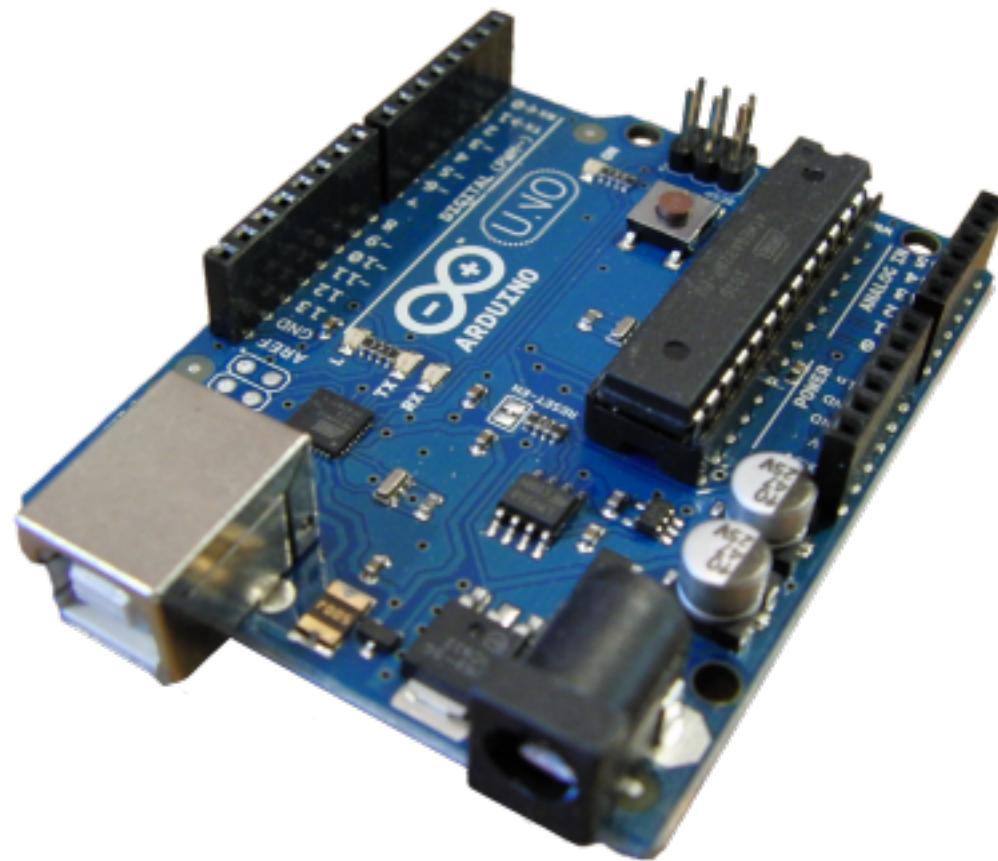


Development IsPrime

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



C++ on Arduino



Testing environment Arduino

```
void setup()
{
    Serial.begin(9600);
    if /* */ {
        Serial.println("Test failed!");
    }
}
```

Red: write a test that fails

```
void setup() {  
    Serial.begin(9600);  
    if (!IsPrime(2)) {  
        Serial.println(  
            "Two must be prime"  
        );  
    }  
}
```

Green: pass the test

```
bool IsPrime(const int x) {  
    return true;  
}  
  
void setup() {  
    Serial.begin(9600);  
    if (!IsPrime(2)) {  
        Serial.println("Two must be prime");  
    }  
}
```

Refactor

- Checking in

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

```
git commit -m "IsPrime: 2 is prime"
```

Red: write a test that fails

```
void setup() {  
    /* Previous code */  
    if (IsPrime(4)) {  
        Serial.println(  
            "Four is not prime"  
        );  
    }  
}
```

Green: pass the test

```
bool IsPrime(const int x) {  
    for (int i=2; i!=x; ++i)  
    {  
        if (x % i == 0) return false;  
    }  
    return true;  
}  
  
void setup() { /* */ }
```

Refactor

- Check in

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

```
git commit -m
```

```
"IsPrime: 4 is not prime"
```

Red: write a test that fails

```
void setup() {  
    /* Previous code */  
    if (IsPrime(1)) {  
        Serial.println(  
            "One is not prime"  
        );  
    }  
}
```

Green: pass the test

```
bool IsPrime(const int x) {  
    if (x == 1) return false;  
    /* rest of the code */  
}
```

Refactor

- Check in

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

```
git commit -m
```

```
"IsPrime: 1 is not prime"
```

Red: write a test that fails

```
void setup() {  
    /* Previous code */  
    if (IsPrime(0)) {  
        Serial.println(  
            "Zero is not prime"  
        );  
    }  
}
```

Green: pass the test

```
bool IsPrime(const int x) {  
    if (x <= 1) return false;  
    /* rest of the code */  
}
```

Refactor

- Check in

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

```
git commit -m
```

```
"IsPrime: zero and lower not prime"
```

Possible C++ end result

```
bool IsPrime(const int x) {  
    if (x <= 1) return false;  
    const int max = static_cast<int>(  
        sqrt(static_cast<double>(x))  
    ) + 1  
};  
for (int i=2; i!=max; ++i) {  
    if (x % i == 0) return false;  
}  
return true;  
}
```

Tests used

```
void setup()
{
    if (!IsPrime(2)) { /* ERROR */ }
    if ( IsPrime(4)) { /* ERROR */ }
    if ( IsPrime(1)) { /* ERROR */ }
    if ( IsPrime(0)) { /* ERROR */ }
}
```

C++ on a desktop PC



Testing environment desktop

```
#include <cassert>

int main()
{
    assert(/* something */);
}
```

Red: write a test that fails

```
#include <cassert>

int main()
{
    assert(IsPrime(2));
}
```

Green: pass the test

```
bool IsPrime(const int x) {  
    return true;  
}  
  
int main()  
{  
    assert(IsPrime(2));  
}
```

Refactor

- Checking in

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

```
git commit -m "IsPrime: 2 is prime"
```

Red: write a test that fails

```
int main()
{
    /* Previous code */
    assert(!IsPrime(4));
}
```

Green: pass the test

```
bool IsPrime(const int x) {  
    for (int i=2; i!=x; ++i)  
    {  
        if (x % i == 0) return false;  
    }  
    return true;  
}  
  
int main() { /* */ }
```

Refactor

- Check in

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

```
git commit -m
```

```
"IsPrime: 4 is not prime"
```

Red: write a test that fails

```
int main()
{
    /* Previous code */
    assert(!IsPrime(1));
}
```

Green: pass the test

```
bool IsPrime(const int x) {  
    if (x == 1) return false;  
    /* rest of the code */  
}
```

Refactor

- Check in

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

```
git commit -m
```

```
"IsPrime: 1 is not prime"
```

Red: write a test that fails

```
int main()
{
    /* Previous code */
    assert(!IsPrime(0));
}
```

Green: pass the test

```
bool IsPrime(const int x) {  
    if (x <= 1) return false;  
    /* rest of the code */  
}
```

Refactor

- Check in

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

```
git commit -m
```

```
"IsPrime: zero and lower not prime"
```

Possible C++ end result

```
bool IsPrime(const int x) {  
    if (x <= 1) return false;  
    const int max = static_cast<int>(  
        std::sqrt(static_cast<double>(x))  
    ) + 1  
};  
for (int i=2; i!=max; ++i) {  
    if (x % i == 0) return false;  
}  
return true;  
}
```

Tests used

```
int main()
{
    assert( IsPrime(2) );
    assert( !IsPrime(4) );
    assert( !IsPrime(1) );
    assert( !IsPrime(0) );
}
```

R on a desktop PC



Testing environment desktop

```
library(testit)

assert /* something */)
```

Red: write a test that fails

```
library(testit)

assert(IsPrime(2))
```

Green: pass the test

```
library(testit)

IsPrime <- function(x)
{
  return (TRUE)
}

assert(IsPrime(2))
```

Refactor

- Checking in

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

```
git commit -m "IsPrime: 2 is prime"
```

Red: write a test that fails

```
/* Previous code */  
assert( !IsPrime( 4 ) )
```

Green: pass the test

```
IsPrime <- 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
```

```
"IsPrime: 4 is not prime"
```

Red: write a test that fails

```
assert (!IsPrime ("Two") )
```

Green: pass the test

```
is_integer <- function(N) {  
  !grepl(  
    "[^[:digit:]]",  
    format(N, digits = 20, scientific = FALSE)  
  )  
}  
  
IsPrime <- function(x)  
{  
  if (!is_integer(x)) return (FALSE)  
  # Rest of code  
}
```

Refactor

- Check in

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

```
git commit -m
```

“IsPrime: words are not prime”

Possible R end result

```
library(testit)

is_integer <- function(N) {
  !grepl("[^[:digit:]]", format(N, digits = 20, scientific = FALSE))
}

IsPrime <- function(x)
{
  if (!is_integer(x)) return (FALSE)
  if (x == 2) return (TRUE)
  for (i in seq(2,sqrt(x)))
  {
    if (x %% i == 0) return (FALSE);
  }
  return (TRUE)
}
```

Tests used

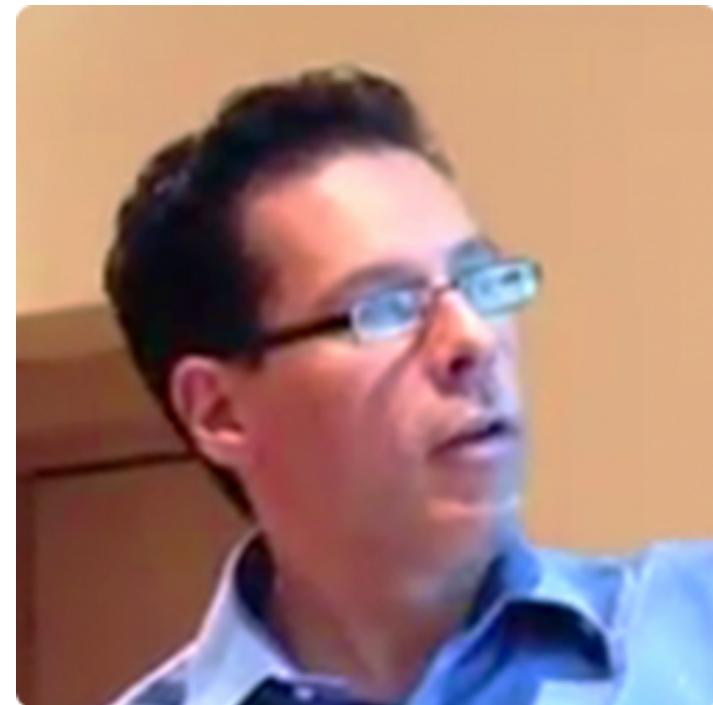
```
assert(IsPrime(2))  
assert(!IsPrime(4))  
assert(!IsPrime("Two"))
```

Final remarks

- Run-time speed
- Collaboration
- Limitation

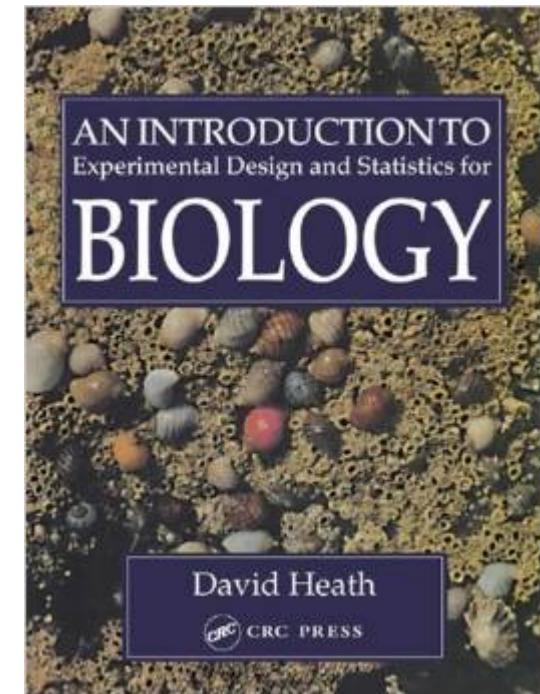
Run-time speed

- “It is far, far easier to make a correct program fast than it is to make a fast program correct.”
Herb Sutter
- The speed of a function can be measured
- The speed of a (possibly) improved function should be higher
- The (possibly) improved function should return the same correct answers



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(AreAboutEqual(expected, results));
```

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

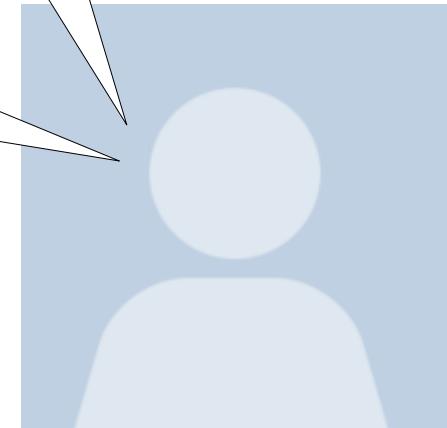
```
//From Heath, page 263, now with zero added
values = { 0.6, 1.4, 0.0, 4.0, 13.0, 14.5, 9.4, 11.4, 12.6, 4.0 };
expected = { 1.0, 2.0, 0.0, 3.5, 8.0, 9.0, 5.0, 6.0, 7.0, 3.5 };
results = GetRanks(values);
// Heath, page 263, with zero value
assert(AreAboutEqual(expected, results));
```

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



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(AreAboutEqual(expected, results)) ;
```

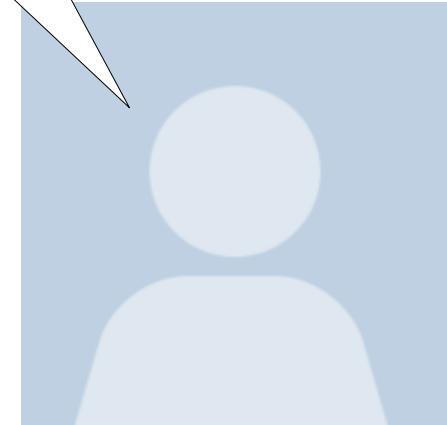
```
values      = { 0.0, 0.0, 0.0 } ;  
expected   = { 0.0, 0.0, 0.0 } ;  
results    = GetRanks(values) ;  
assert(AreAboutEqual(expected, results)) ;
```

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!



Eternal glory

This screenshot shows a GitHub repository page for [richelbilderbeek / ProjectRichelBilderbeek](#). The user has performed a search for the term "senbong G". The results show two matches from the file [Test/CppWilcoxonSignedRankTest/main.cpp](#). The code snippet is as follows:

```
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)
...
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};
```

The GitHub interface includes a navigation bar with links to Explore, Gist, Blog, Help, and a user profile for richelbilderbeek. It also features standard GitHub actions like Unwatch, Unstar, and Fork.

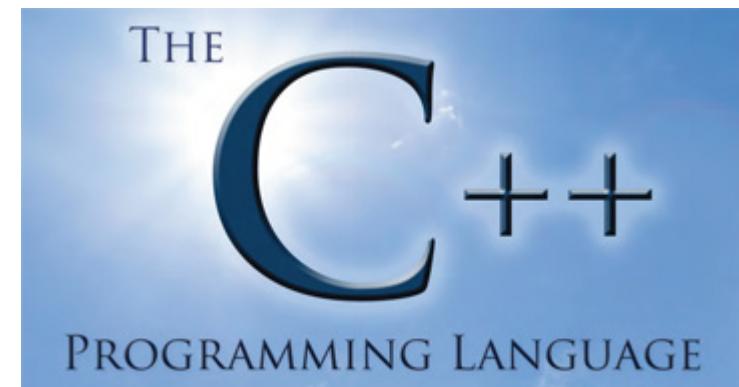
Limitations on Arduino

- You cannot test everything on Arduino:
 - Is the text displayed on the LCD screen?
 - Is the servo motor moving?
- What you can test:
 - What is the text I send to the LCD screen?
 - What is the angle I send to the servo motor?
- It is not doable to add sensors to test everything, as these sensors must also be tested



Limitations C++ on desktop PC

- Few
- So it is too easy to make brute force tests, just to be sure
- Solution:
 - Limit tests to 100 msec per class



Limitations R on desktop PC

- R does not have a debug and release mode
- Tests in functions will also slow down your released code!
- Solution:
 - Write no tests in function
 - Put tests in separate files
 - You will need more tests thanks to this



Conclusion

- Test-driven development
 - uses a systematic approach
 - delivers testable code
 - delivers code that has been tested to be correct
 - does not test more than needed
 - facilitates collaboration