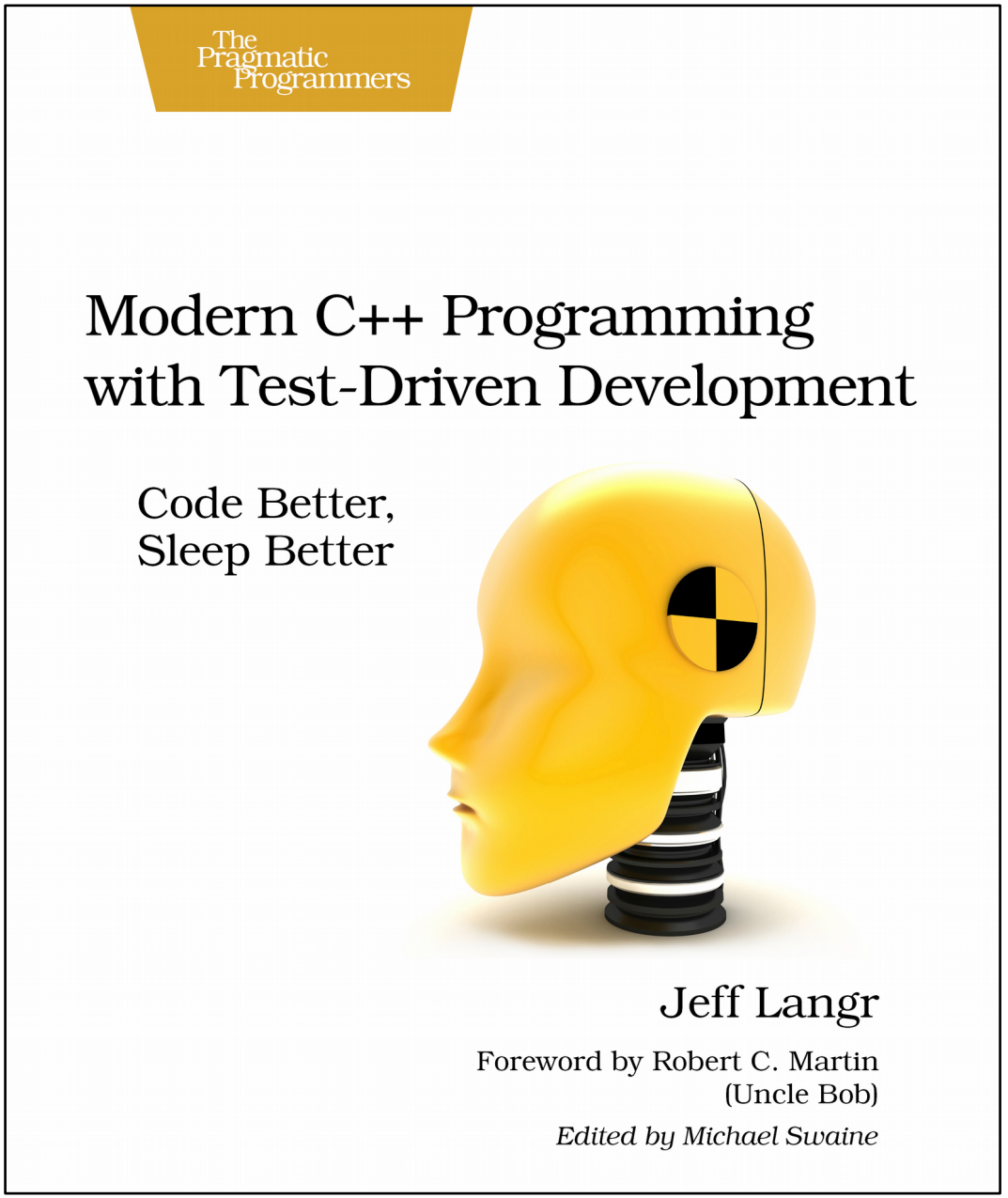
Test-driven development

Richèl Bilderbeek

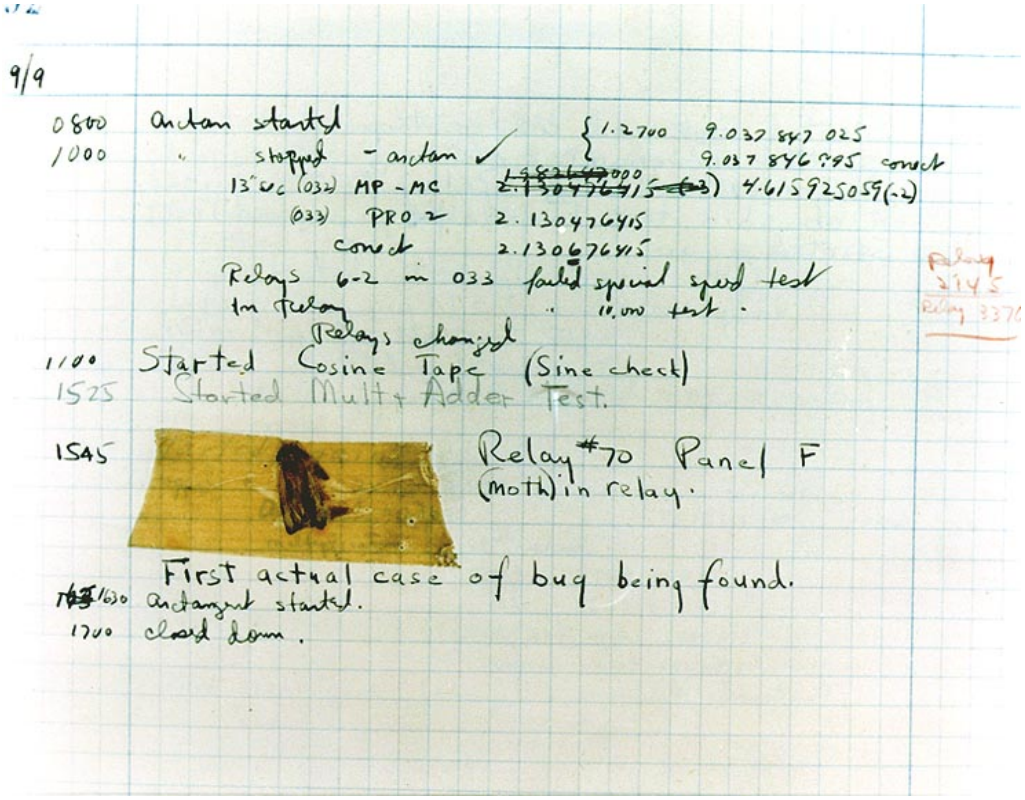
What is test-driven development?

- 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

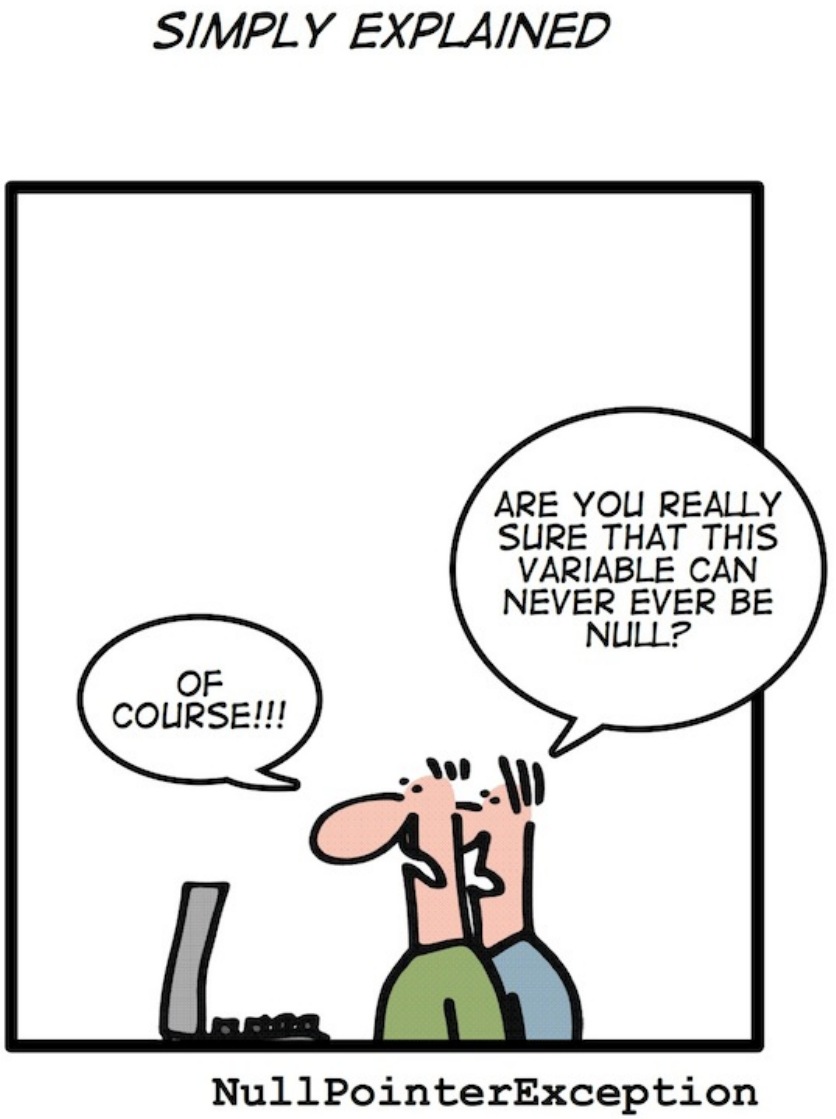


How much tests?

- “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?

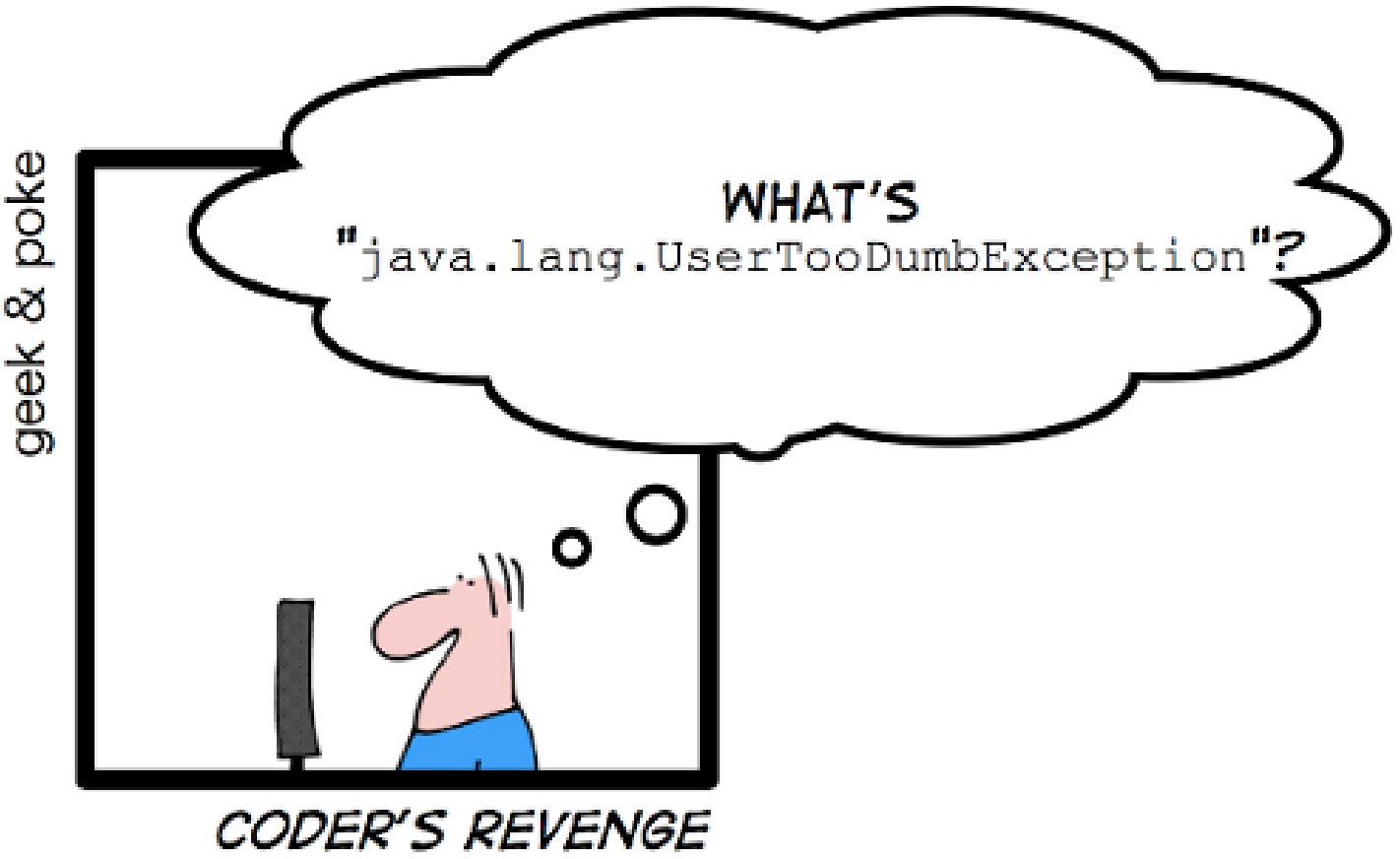
What is an error?

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



What is not an error?

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



Logic error

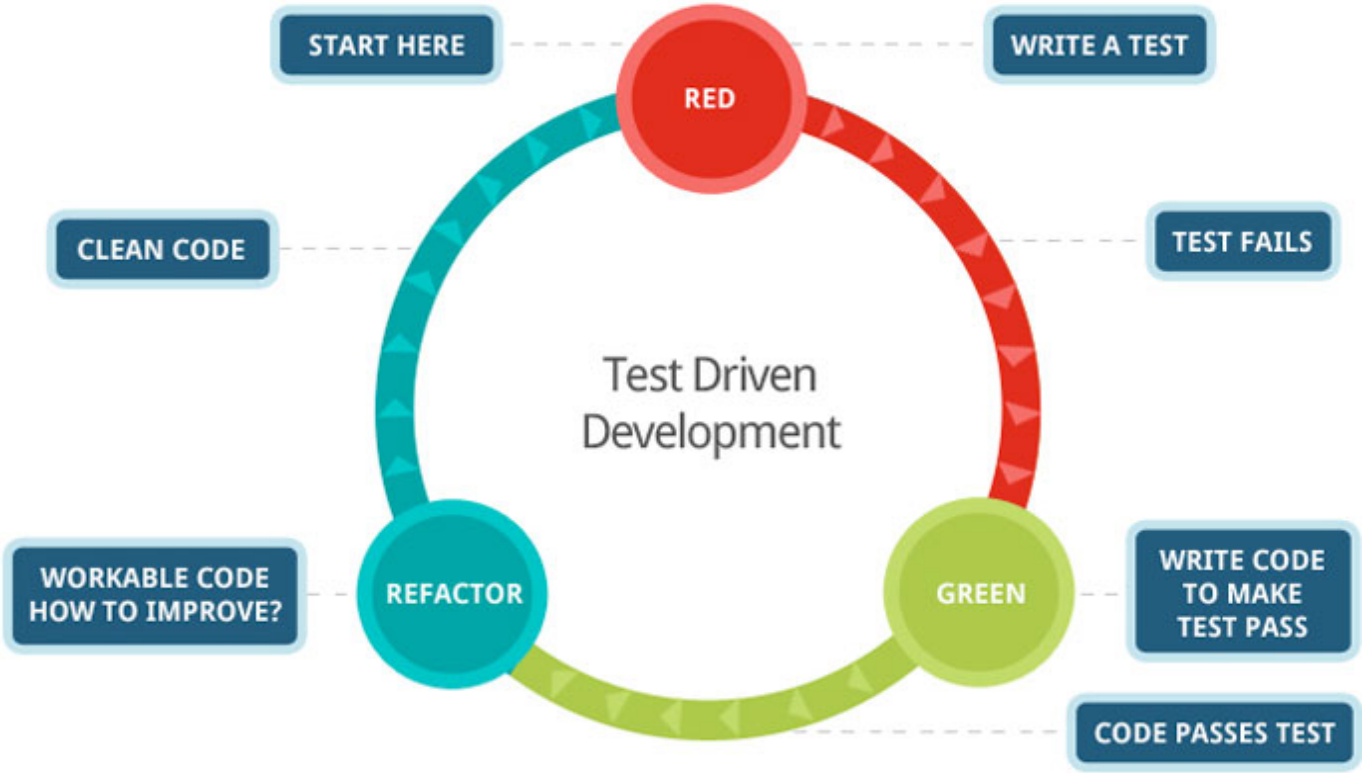
- The programmer makes an incorrect reasoning
- Example: a function that returns the wrong answer
- Test:
  - call the function with different arguments
  - see if it returns the expected results

Example

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

Developing IsPrime

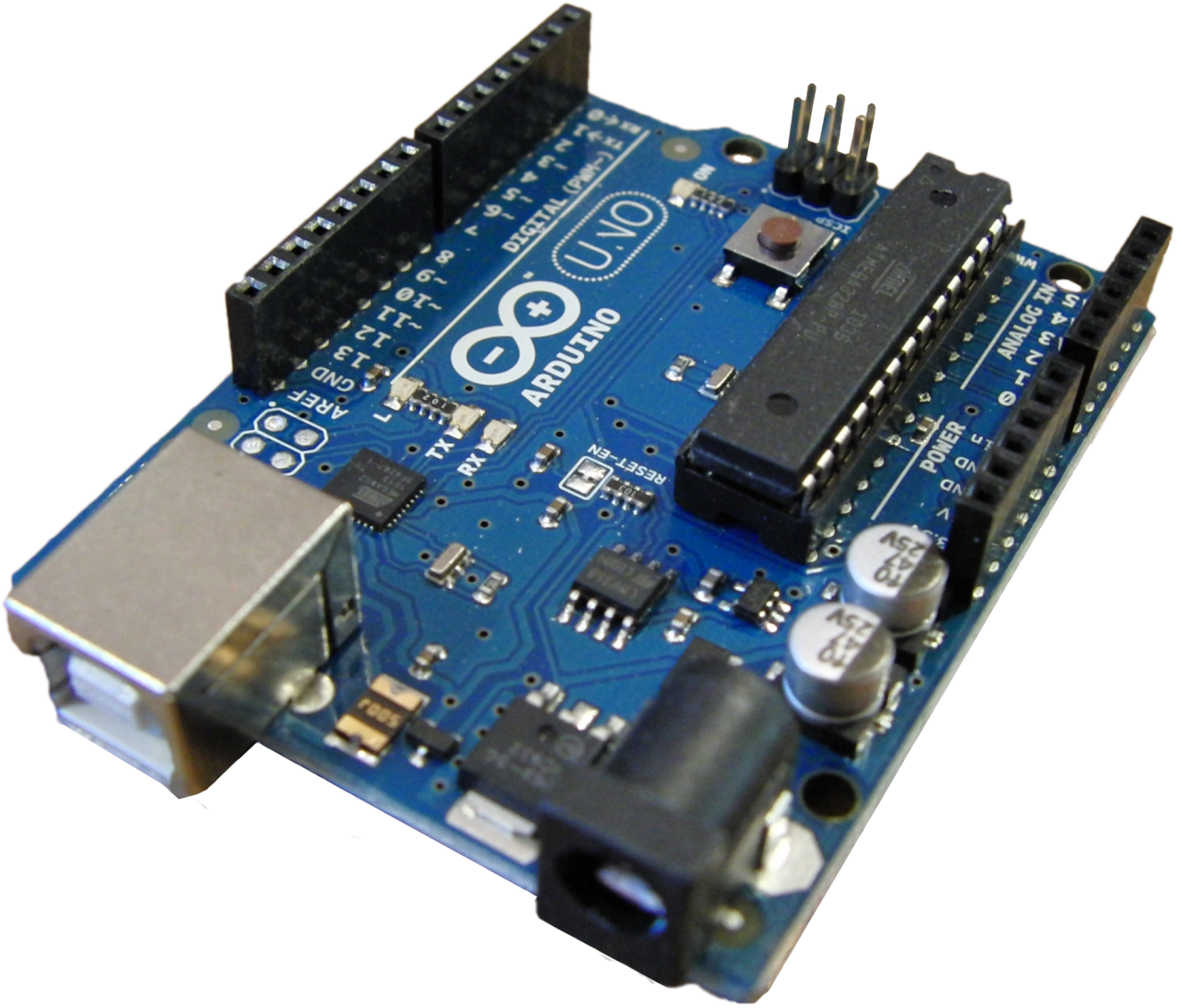
- Cycle test-driven development:
  - Red: write a test that fails
  - Green: pass the test
  - Refactor: improve the new code, clean up mess, check in code



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”
```

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”
```

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

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;
}
```

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 on desktop PC

- Few

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
  - allows rapid development