Asserções e Programação por Contrato

Instituto Politécnico de Beja
Escola Superior de Tecnologia e de Gestão
Licenciatura em Engenharia Informática
Unidade Curricular de Programação Orientada por Objectos
João Paulo Barros

Beja, 08 de Abril de 2019

Curas para os Erros

- Ler e pensar novamente.
- Adicionar "prints"
- Utilizar um *debugger*
- Imprimir o código e ler com atenção.
- Explicar a outra pessoa ou objecto(!)
- ...

O que é melhor do que a Cura?

Prevenção!

Erros nos programas.
o que fazer?

1. Corrigi-los
2. Evitar que surjam

- "Clean Code",
  e.g. Regras de Estilo
- Asserções e
  programação por contrato
- ...
- Programação conduzida
  por testes

Asserções

- Pré-condições (verificar antes)
- Pós-condições (verificar depois)
- Invariantes de classe (verificar “sempre”)

Também denominadas: ***contratos***

> *A contract is a specification of properties of a software element that affect its use by potential clients.*
>
> *Bertrand Meyer, “ Touch of Class: Learning to Program Well with Object and Contracts”, Springer Verlag, 2009*

Um pequeno Exemplo (uma classe Clock)

```
class Clock
{
    private int hours;
    private int minutes;
    private int seconds;

    /** creates clock with given time ...*/
    public Clock(int hours, int minutes, int seconds)...

    /** Sets a new time ...*/
    public void setTime(int hours, int minutes, int seconds)...

    /** Sets a new time ...*/
    public void setTime(int onlySeconds)...

    /** Adds time in seconds ...*/
    public void addSeconds(int onlySeconds)...

}
```

Um pequeno Exemplo (uma classe Clock)

```
/** Sets a new time */
    public void setTime(int hours, int minutes, int seconds)
    {
        assert(0 <= hours && hours <= 23);
        assert(0 <= minutes && minutes <= 59);
        assert(0 <= seconds && seconds <= 59);

        this.hours   = hours;
        this.minutes = minutes;
        this.seconds = seconds;
}
```

Um pequeno Exemplo (uma classe Clock)

```
/** Alternative definition. Sets a new time */
public void setTime(int hours, int minutes, int seconds)
        throws IllegalArgumentException
{
    if (0 <= hours && hours <= 23 &&
        0 <= minutes && minutes <= 59 &&
        0 <= seconds && seconds <= 59) == false)
    {
        throw new IllegalArgumentException("ilegal time");
    }

    this.hours   = hours;
    this.minutes = minutes;
    this.seconds = seconds;
}
```

Um pequeno Exemplo (uma classe Clock)

```
/** Sets a new time in seconds */
public void setTime(int onlySeconds)
{
    assert(onlySeconds >= 0);

    this.hours   = onlySeconds / 3600;
    this.minutes = (onlySeconds % 3600) / 60;
    this.seconds = (onlySeconds % 3600) % 60;

    assert(0 <= this.hours   && this.hours <= 23);
    assert(0 <= this.minutes && this.minutes <= 59);
    assert(0 <= this.seconds && this.seconds <= 59);
}
```

Um pequeno Exemplo (uma classe Clock)

```
/** Adds time in seconds */
public void addSeconds(int onlySeconds)
{
    assert(onlySeconds >= 0);

    int s =  this.hours * 3600 +
             this.minutes * 60 +
             this.seconds +
             onlySeconds;
    this.setTime(s);

    assert(0 <= this.hours   && this.hours <= 23);
    assert(0 <= this.minutes && this.minutes <= 59);
    assert(0 <= this.seconds && this.seconds <= 59);
}
```

Invariantes de Classe

- Verdadeiras em qualquer estado "estável" (não transitório)

Exemplo:

```
assert(0 <= this.hours   && this.hours <= 23);
assert(0 <= this.minutes && this.minutes <= 59);
assert(0 <= this.seconds && this.seconds <= 59);
```

Conclusão

Prevenção

- Asserções
    - pré-condições
    - pós-condições
    - invariantes de classe
- Testes Unitários
    - Automação
    - Regressão

Em java

```
assert(1 <= month && month <= 12);
```

ou

```
assert(1 <= month && month <= 12)  : "month out of range";
```

Têm de ser *activadas*:

```
javac Time.java
java -ea Time
```

Programação Conduzida por Testes

Instituto Politécnico de Beja
Escola Superior de Tecnologia e de Gestão
Licenciatura em Engenharia Informática
Unidade Curricular de Programação Orientada por Objectos
João Paulo Barros

Beja, 2 de Março de 2015

Erros nos programas. O que fazer?

1. Corrigi-los
2. Evitar que surjam

- "Clean Code", e.g. Regras de Estilo
- Asserções e programação por contrato
- ...
- Programação conduzida por testes

Testes Unitários

- *Test automation*
- *Regression testing*

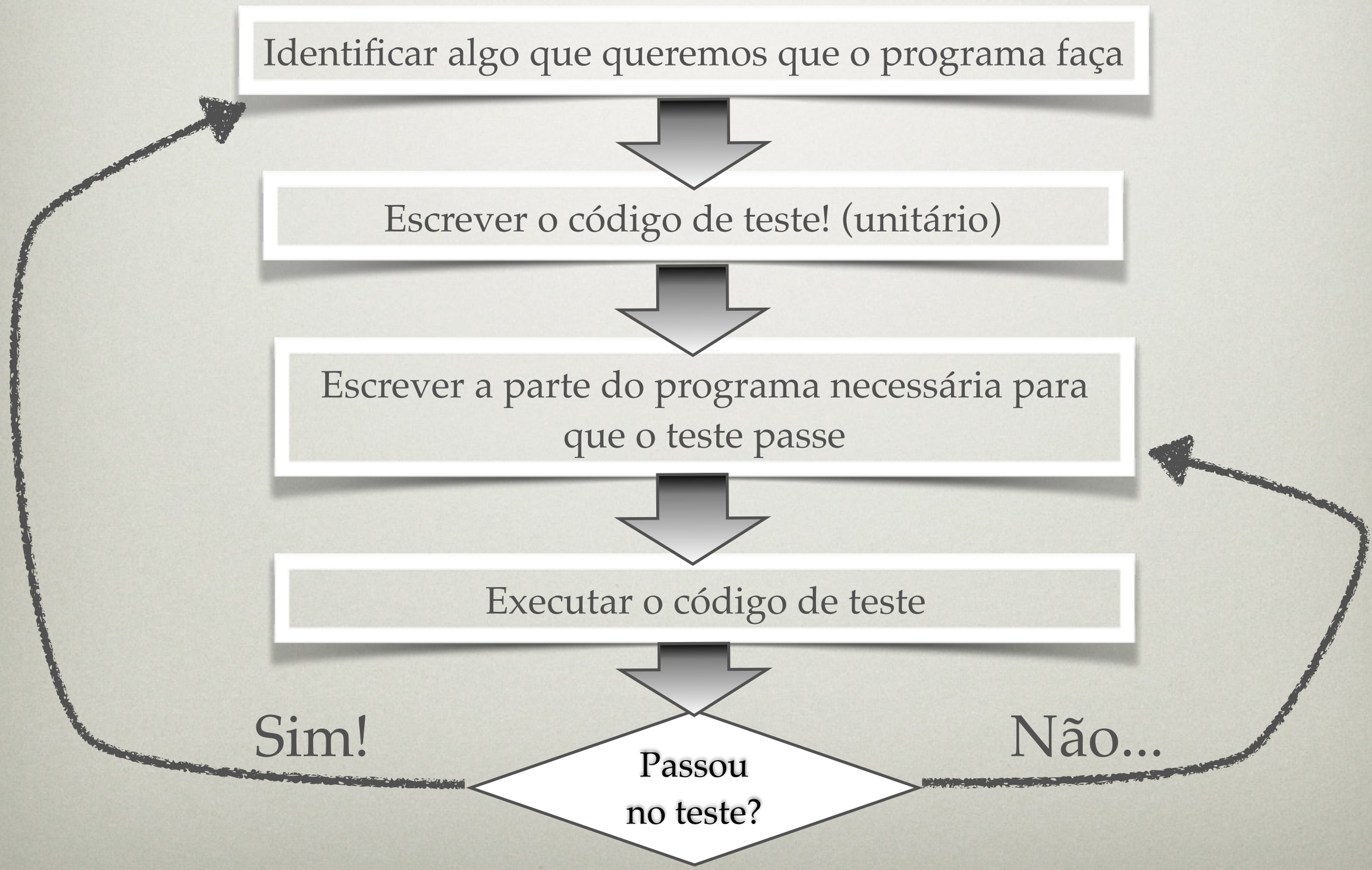
E Como Funciona?...

Solução

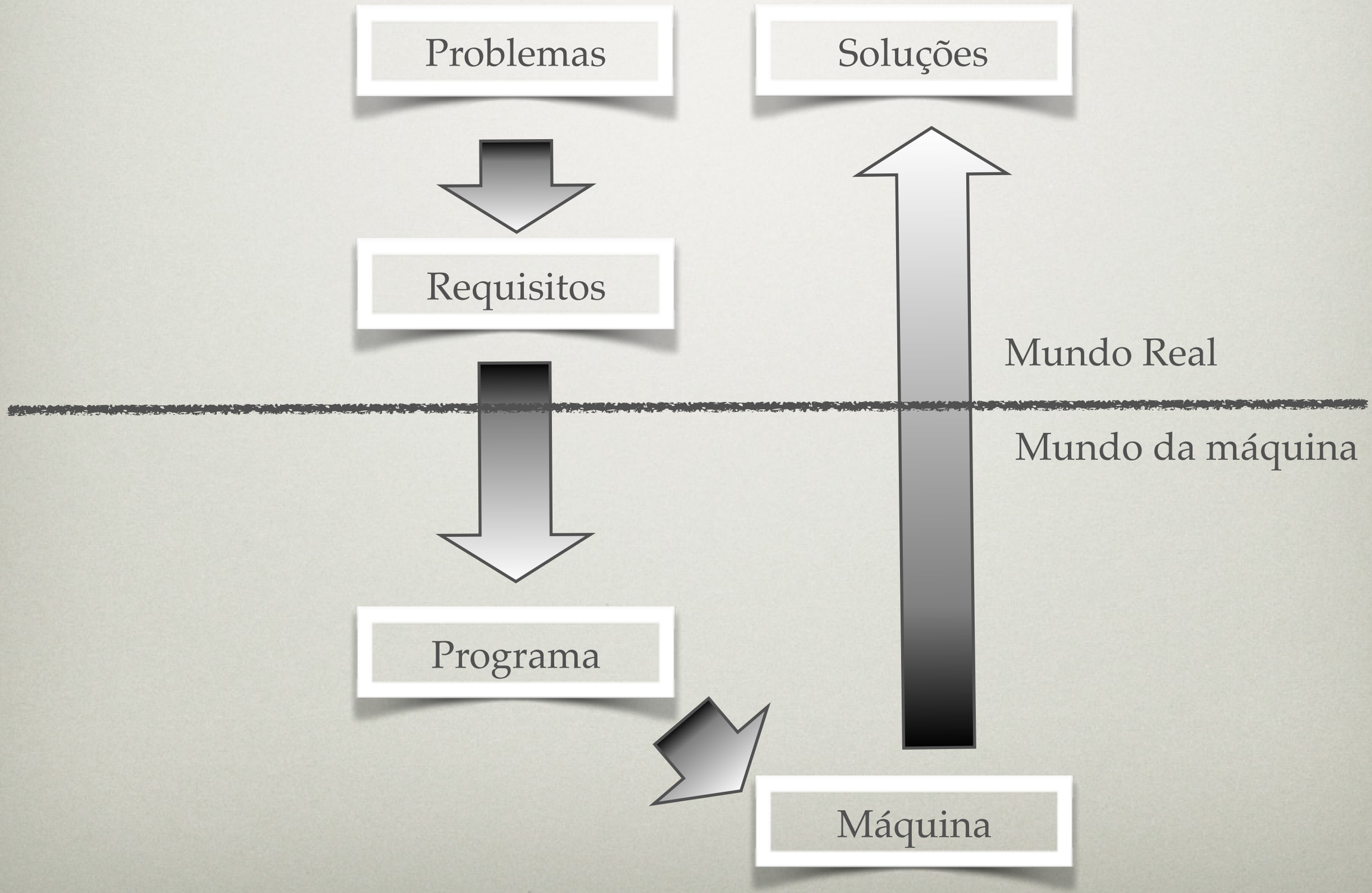
- Perguntar ao código se ele funciona bem.
- Utilizar o computador para fazer essa pergunta.
- Todos os testes podem ser executados em qualquer altura.

Programação conduzida por testes

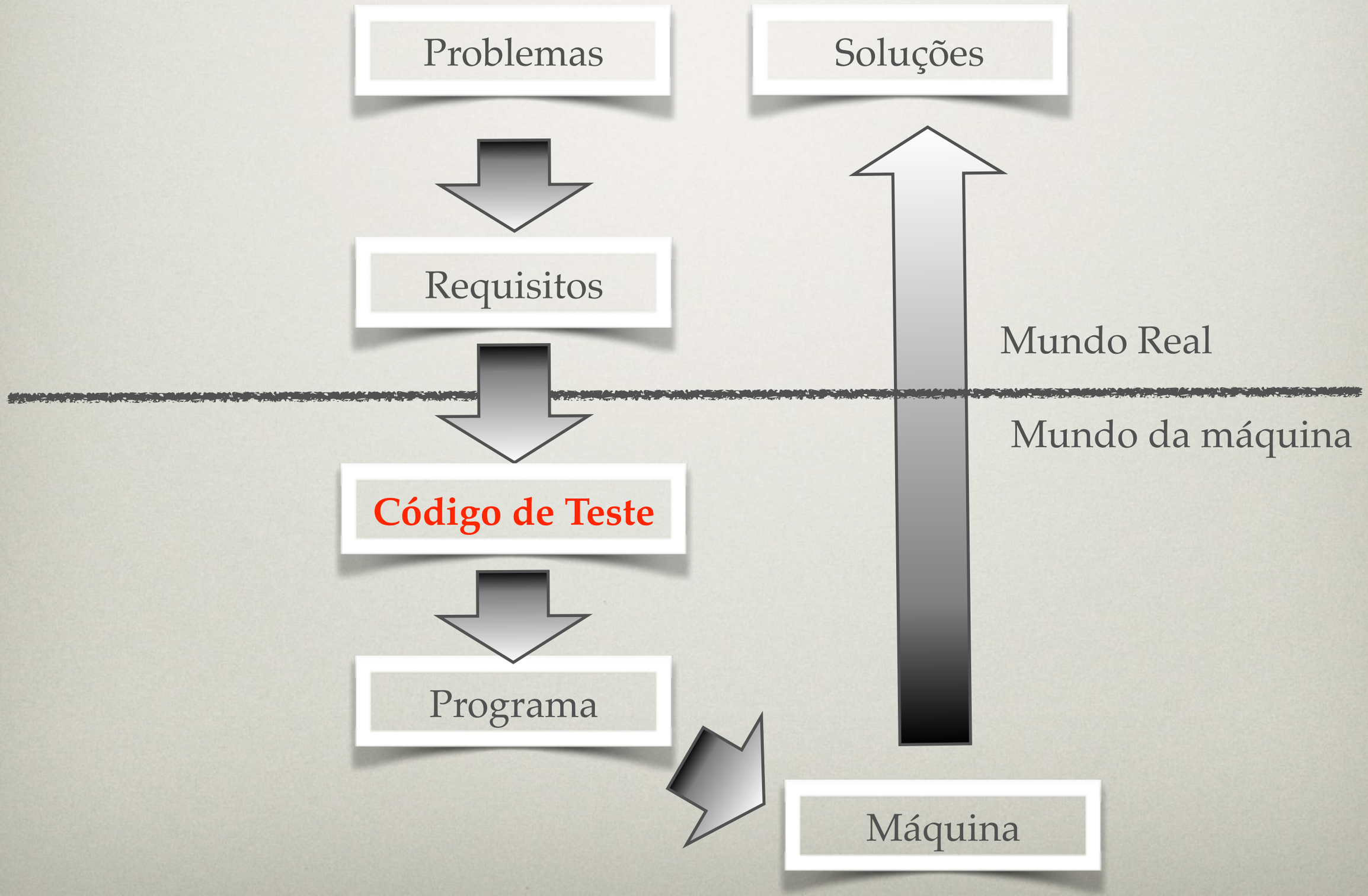
FUNCIONAMENTO



E o que realmente Significa?...



E o que realmente Significa?...



Teste Unitário

1. Um teste unitário é uma porção de código
2. Especifica uma pequena parte do que o código que está a ser escrito deve fazer.
3. Permite que um teste automático seja executado.

Concluindo e Resumindo

1. Escrever o teste.
2. Escrever o código.
3. Não escrever outro código antes do teste passar.
4. Os testes são a única coisa que o programa realmente sabe sobre o seu programa.

Pat and Dale (a short story)

(adaptado deAndy Hunt and Dave Thomas, “Pragmatic Unit Testing in Java with JUnit”, 2003 The Pragmatic Programmers, ISBN 0-9745140-1-2. pp. 2-3)

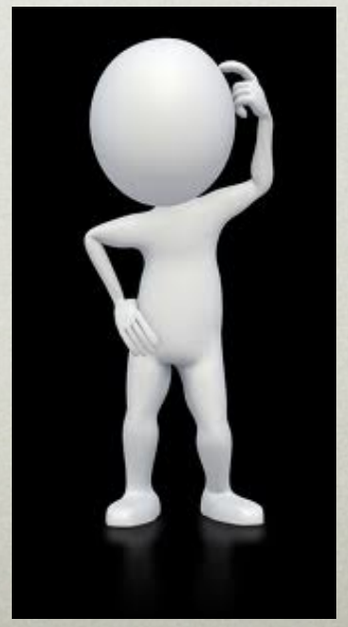
Once upon a time —maybe it was last Tuesday — there were two developers, **Pat** and **Dale**. They were both up against the same deadline, which was rapidly approaching. **Pat** was pumping out code pretty fast; developing class after class and method after method, stopping every so often to make sure that the code would compile.

Pat and Dale (a short story)

(…)
Pat kept up this pace right until the night before the deadline, when it would be time to demonstrate all this code.

Pat ran the top-level program, but didn't get any output at all. Nothing...
Time to step through using the debugger. Hmm. That can't be right, thought Pat. There's no way that this variable could be zero by now.

So Pat stepped back through the code, trying to track down the history of this elusive problem…

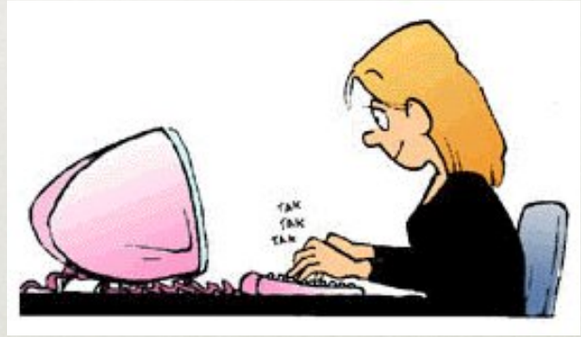


Pat and Dale (a short story)

(…)
It was getting late now. That bug was found and fixed, but Pat found several more during the process. And still, there was no output at all. Pat couldn't understand why. It just didn't make any sense….

Pat and Dale (a short story)

Dale, meanwhile, wasn't churning out code nearly as fast. Dale would write a short test and a new routine to go along with it. Nothing fancy, just a simple test to see if the routine just written actually did what it was supposed to do.



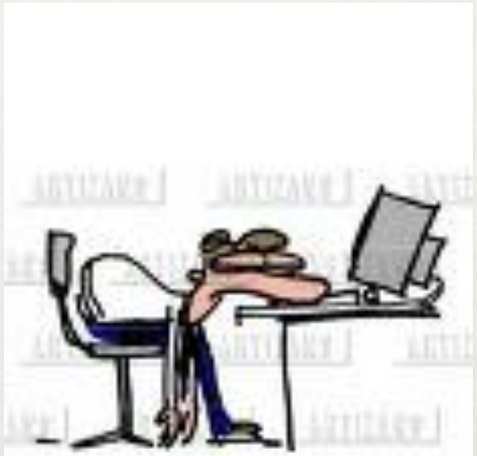
It took a little longer to think of the test, and write it, but Dale **refused to move on until the new routine could prove itself**. Only then would Dale move up and write the next test and the respective routine, and so on.

Pat and Dale (a short story)

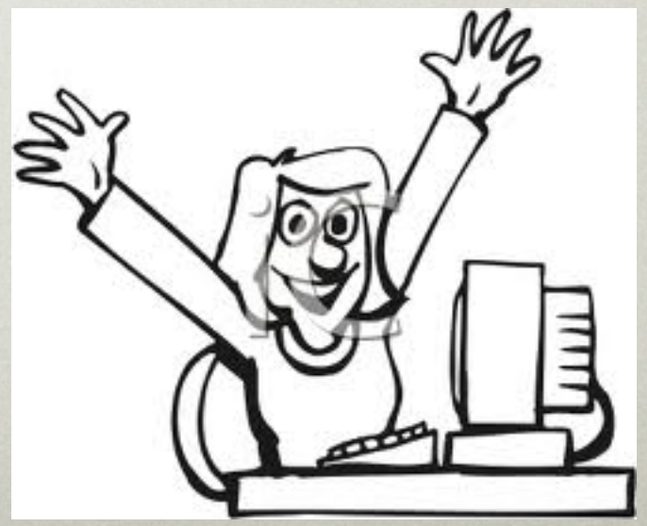
Dale rarely used the debugger, if ever, and was somewhat puzzled at the picture of Pat, head in hands, muttering various evil-sounding curses at the computer with wide, bloodshot eyes staring at all those debugger windows.

Pat and Dale (a short story)

The deadline came and went, and Pat didn't make it...



Dale‘s code was integrated and ran almost perfectly. One little glitch came up, but it was pretty easy to see where the problem was. Dale fixed it in just a few minutes.



Pat and Dale (the end)

Now comes the punch line: Dale and Pat are the same age, and have roughly the same coding skills and mental prowess. The only difference is that **Dale believes very strongly in unit testing, and tests every newly-crafted method before relying on it or using it from other code.**

Pat does not. **Pat "knows" that the code should work as written, and doesn't bother to try it until most of the code has been written**. But by then it's too late, and it becomes very hard to try to locate the source of bugs, or even determine what's working and what's not.