

Engenharia de Software: da teoria a prática em 3,6s

Alcançando atributos qualitativos com boas práticas de codificação



Realização



Apoio



Patrocínio



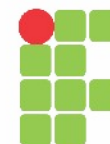
Quem eu sou

Márcio Torres

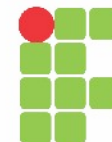
Programador

Professor

Pai



De onde eu vim? Para onde vou?





HELLO, WORLD!

LIST

```
10 HOME
20 INVERSE
30 PRINT "HELLO, WORLD!"
40 NORMAL
50 PRINT CHR$ (7)
```

■



Date: 12-11-1990
Time: 16:21:17

Master Menu

Page 1 of 2
Version 1.00

1. Filing Assistant
2. Reporting Assistant
3. Professional Finance Program
4. UP Planner
5. DOS Commands
6. Managing Your Money
7. Dr. Halo
8. Word Perfect
9. Printmaster

↑ or ↓ point to option
ENTER select option

F1 Help
F7 Quit
F9 Menu Maintenance Pgdn Next page

Database file

Format for Screen
Query

Catalog
View

Quit dBASE III PLUS

ASSIST

<C:>

Opt: 1/6

Move selection bar - ↑↓. Select - ←→. Leave menu - ↔. Help - F1. Exit - Esc.
Select a database file.

AMD 

AMD-K6™-2

AMD-K6-2/500AFX

2.2V CORE/3.3V I/O

A 0010APJW

© 1998 AMD

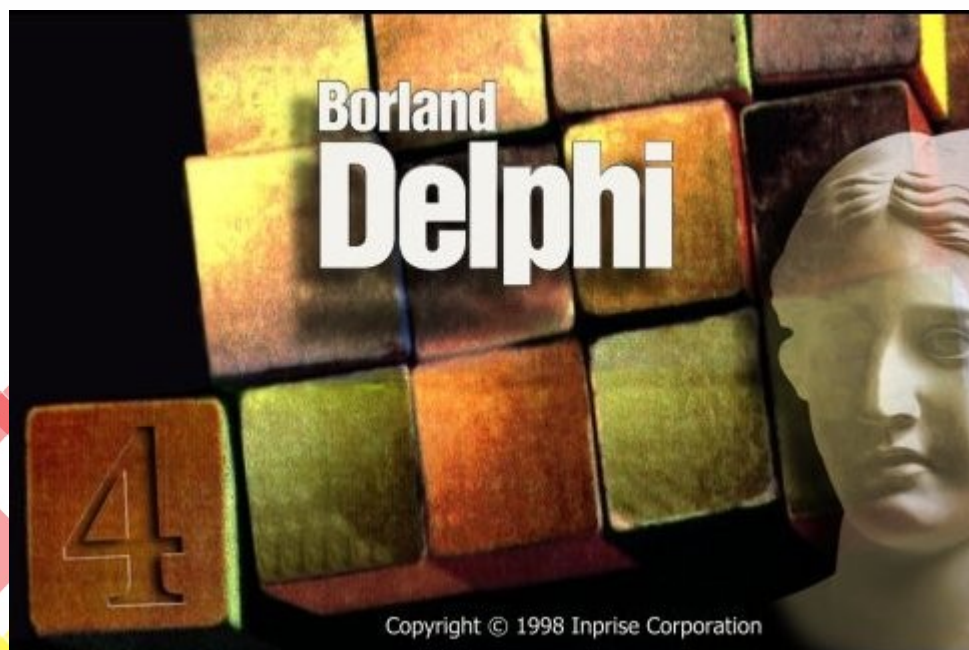
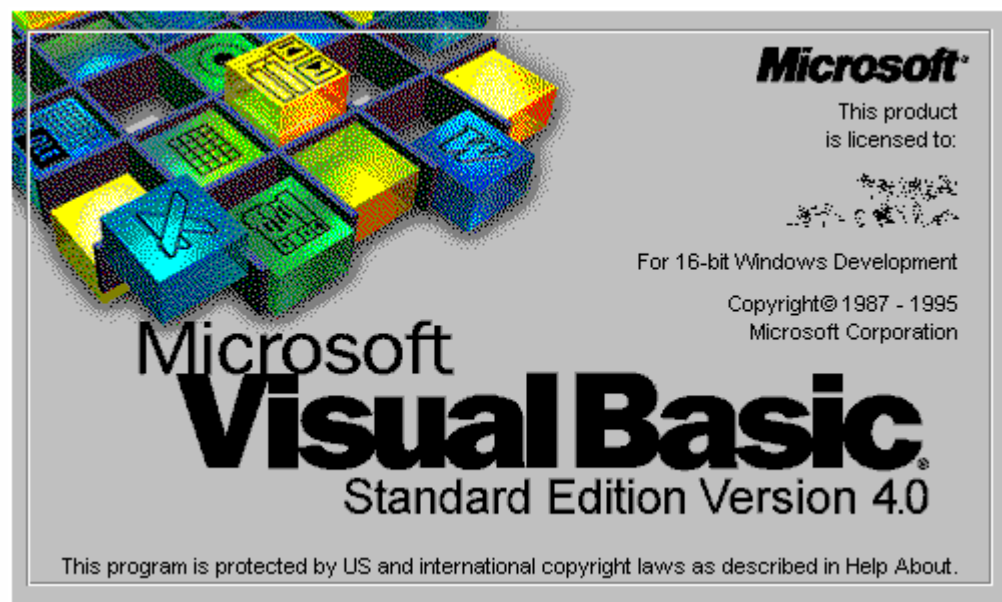
Designed for

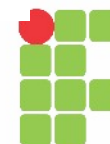
Microsoft®
Windows NT®
Windows® 95

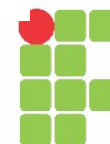
ASSEMBLED IN MALAYSIA

26351

N





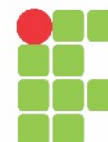


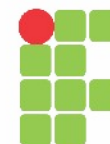
Microsoft®

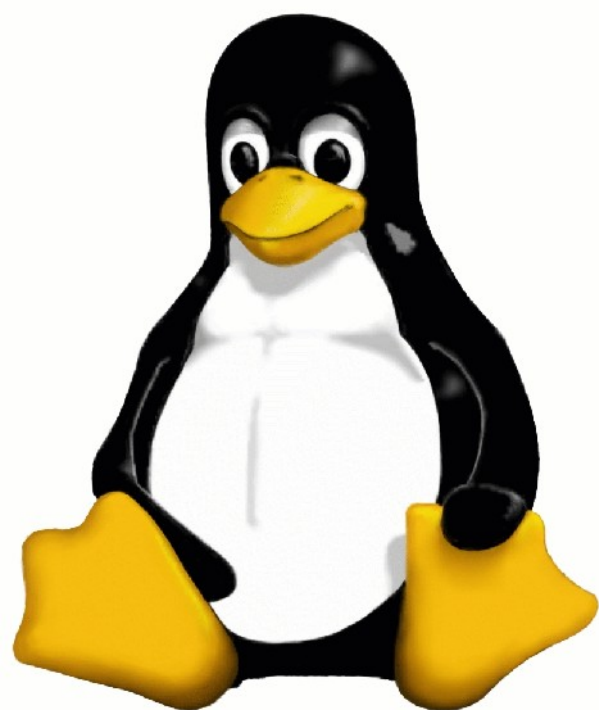


Microsoft®
Windows Me
Millennium
Edition

Getting ready to run Windows for the first time.



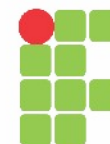




ubuntu



Porque é tão difícil manter um software?





OBJECT-ORIENTED ANALYSIS AND DESIGN WITH APPLICATIONS

THIRD EDITION

GRADY BOOCH, ROBERT A. MAKSIMCHUK,
MICHAEL W. ENGLE, BOBBI J. YOUNG, Ph.D.,
JIM CONALLEN, KELLI A. HOUSTON

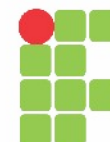


ANALYSIS PATTERNS

REUSABLE OBJECT MODELS

MARTIN FOWLER

Forewords by
Ward Cunningham and Ralph Johnson



APPLYING UML AND PATTERNS

An Introduction to Object-Oriented Analysis and Design
and Iterative Development

THIRD EDITION



"People often ask me which is the best book to introduce them to the world of OO design. Ever since I came across it, *Applying UML and Patterns* has been my unreserved choice."
—Martin Fowler, author of *UML Distilled* and *Refactoring*

CRAIG LARMAN

Foreword by Philippe Kruchten

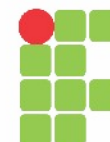
Covers through Version 2.0 OMG UML Standard

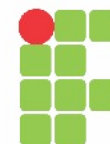
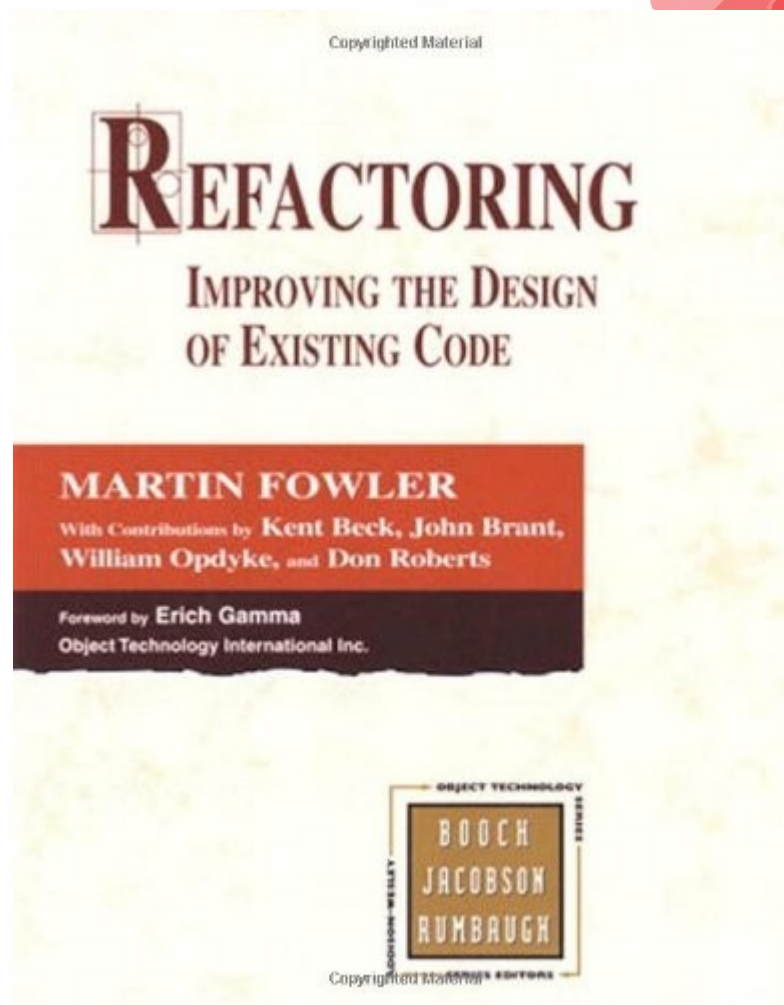
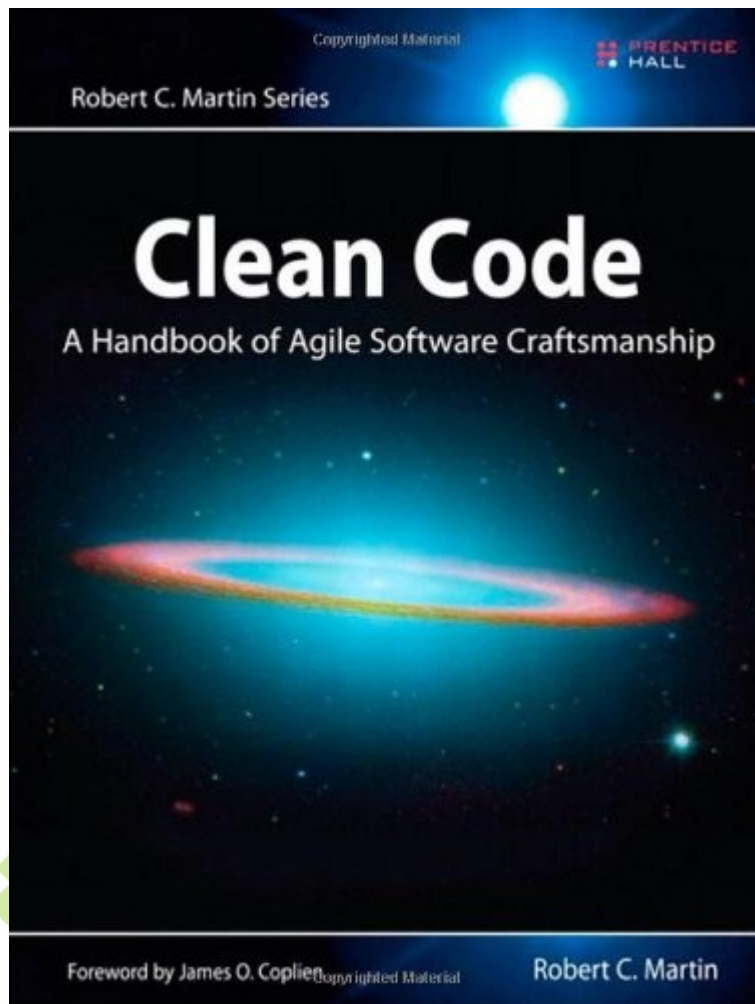
UML DISTILLED THIRD EDITION

A BRIEF GUIDE TO THE STANDARD
OBJECT MODELING LANGUAGE

MARTIN FOWLER

Forewords by Cris Kobryn, Grady Booch,
Ivar Jacobson, and Jim Rumbaugh



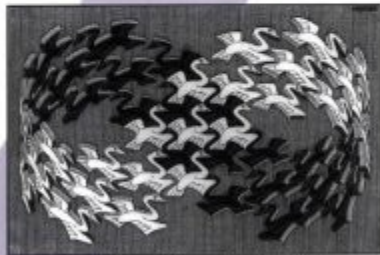


Copyrighted Material

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 M.C. Escher / Gordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch



Copyrighted Material



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

The Addison-Wesley Signature Series

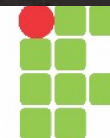
BOOK A MARTIN FOWLER SIGNATURE
Making

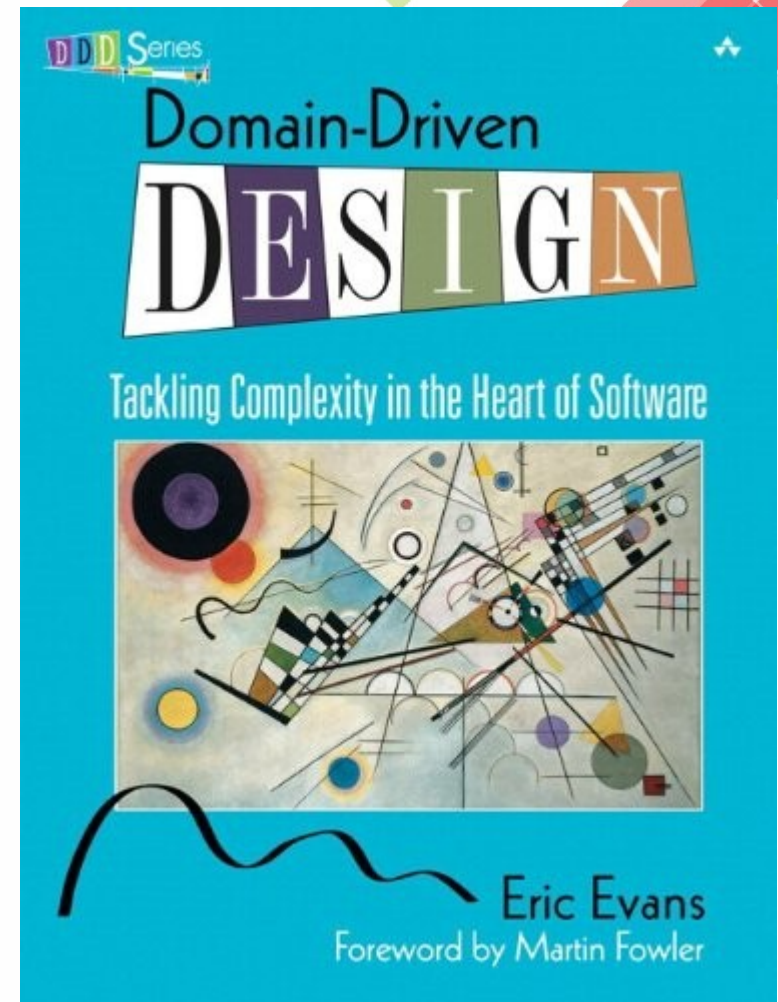
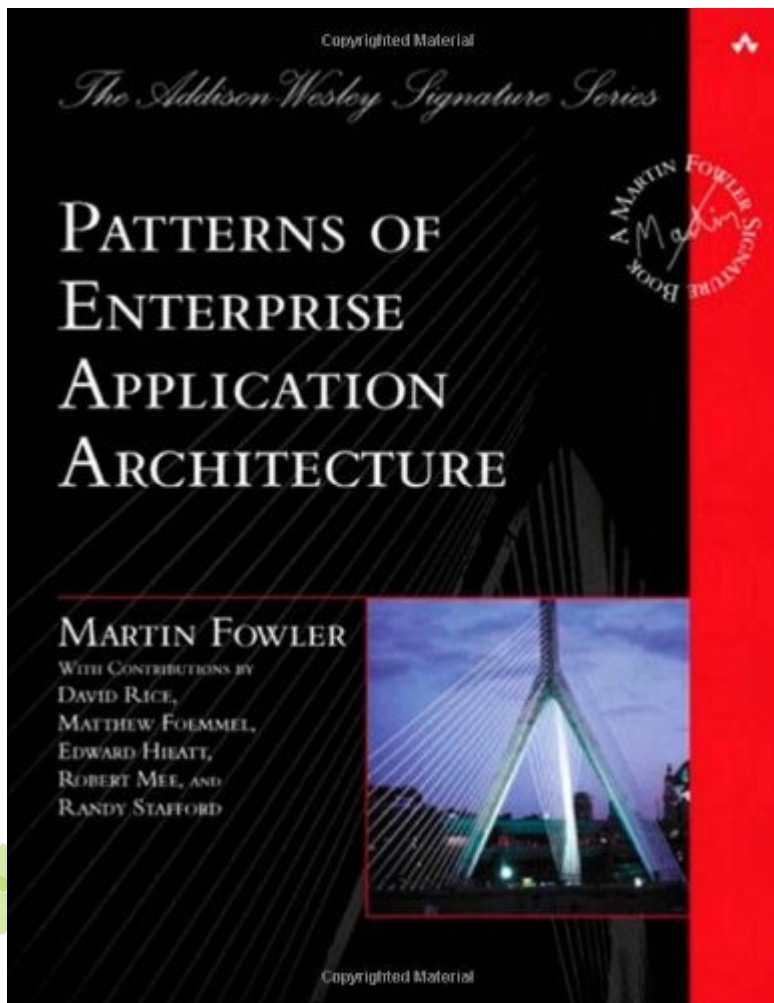
REFACTORING TO PATTERNS

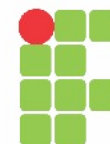
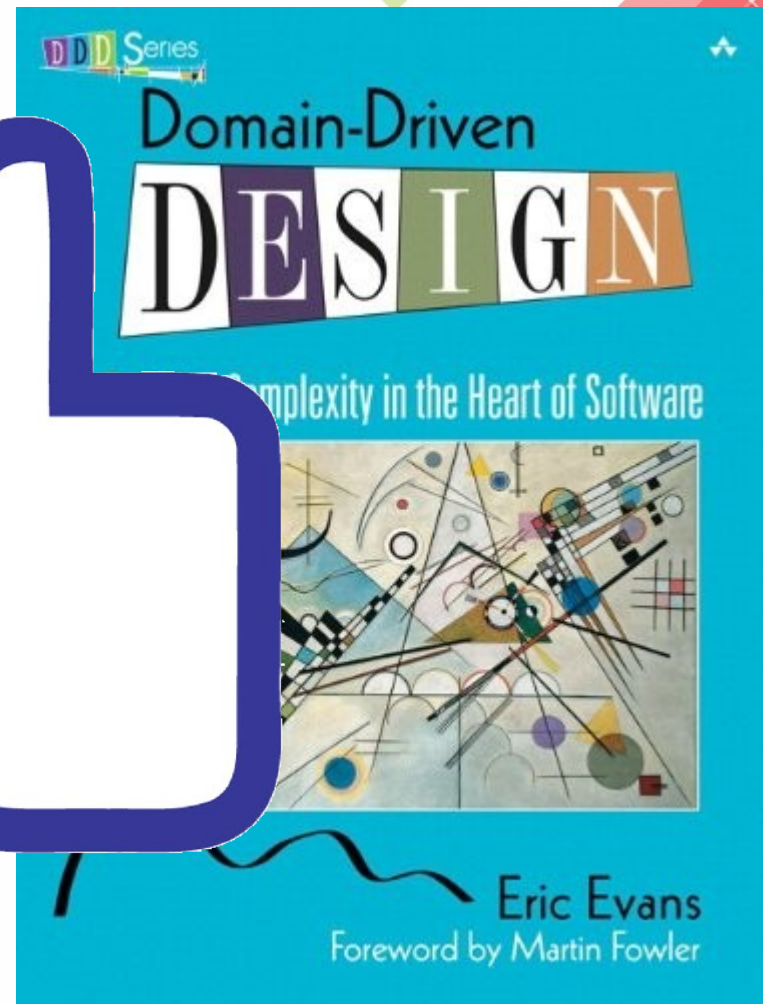
JOSHUA Kerievsky

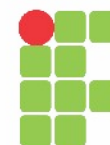
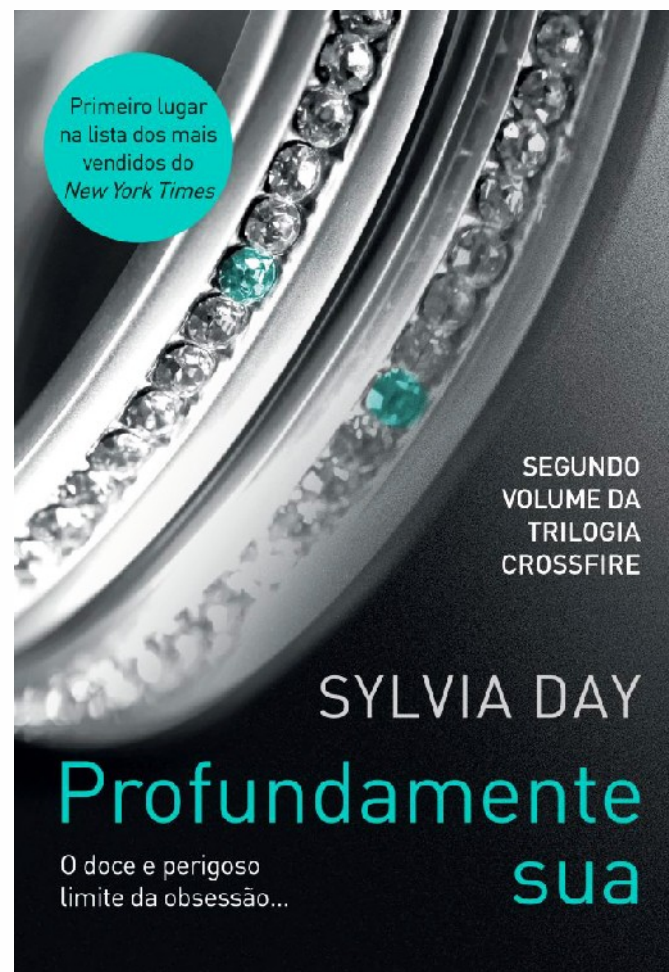
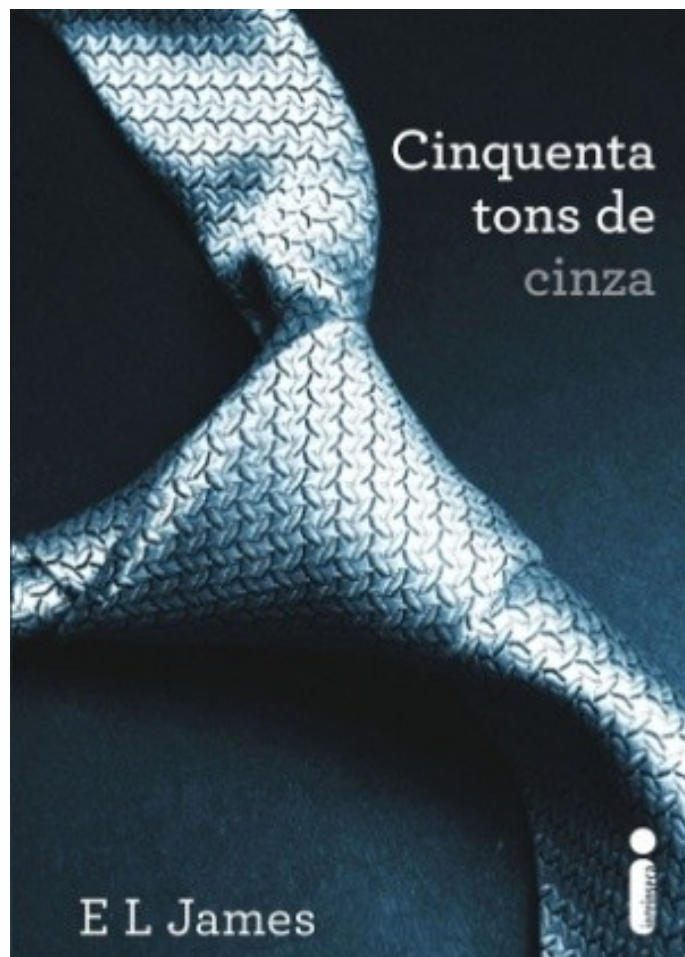


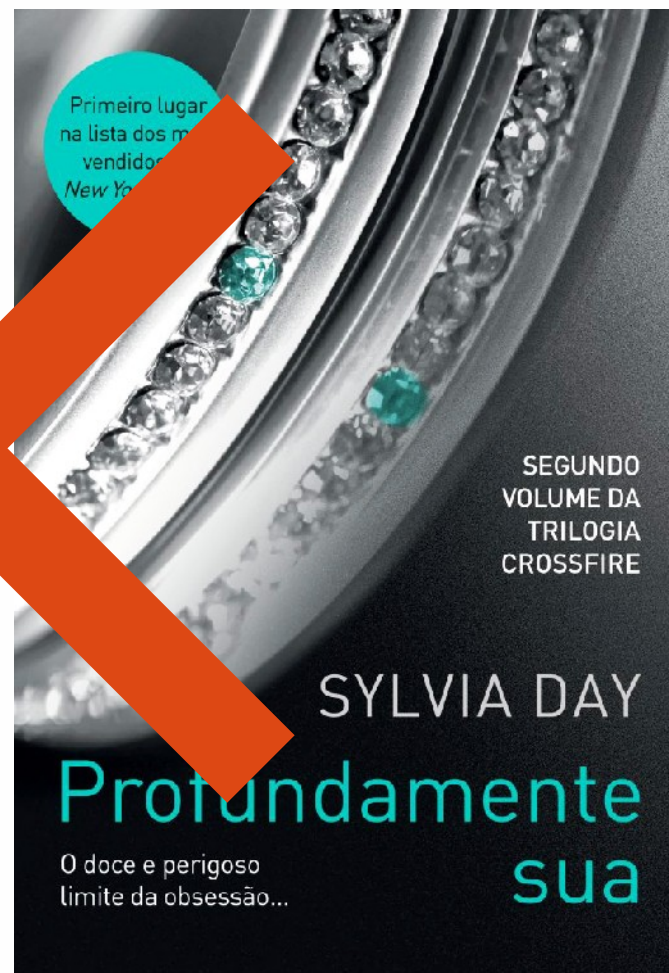
Forewords by Ralph Johnson and Martin Fowler
Afterword by John Brant and Don Roberts

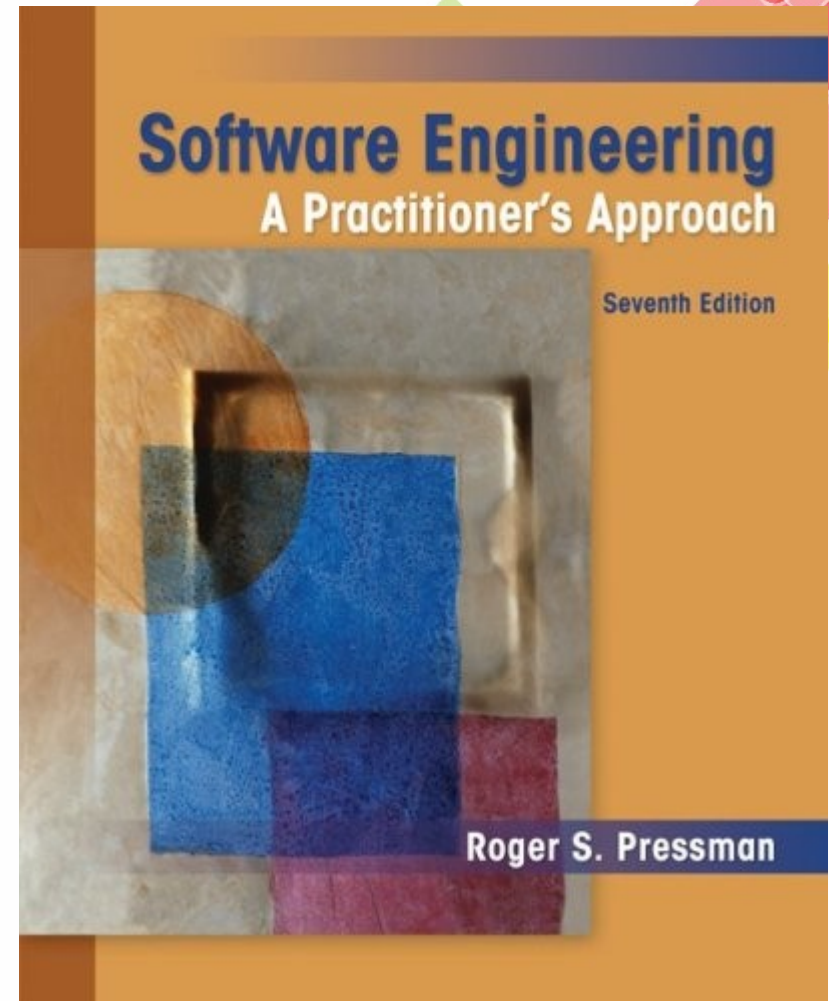
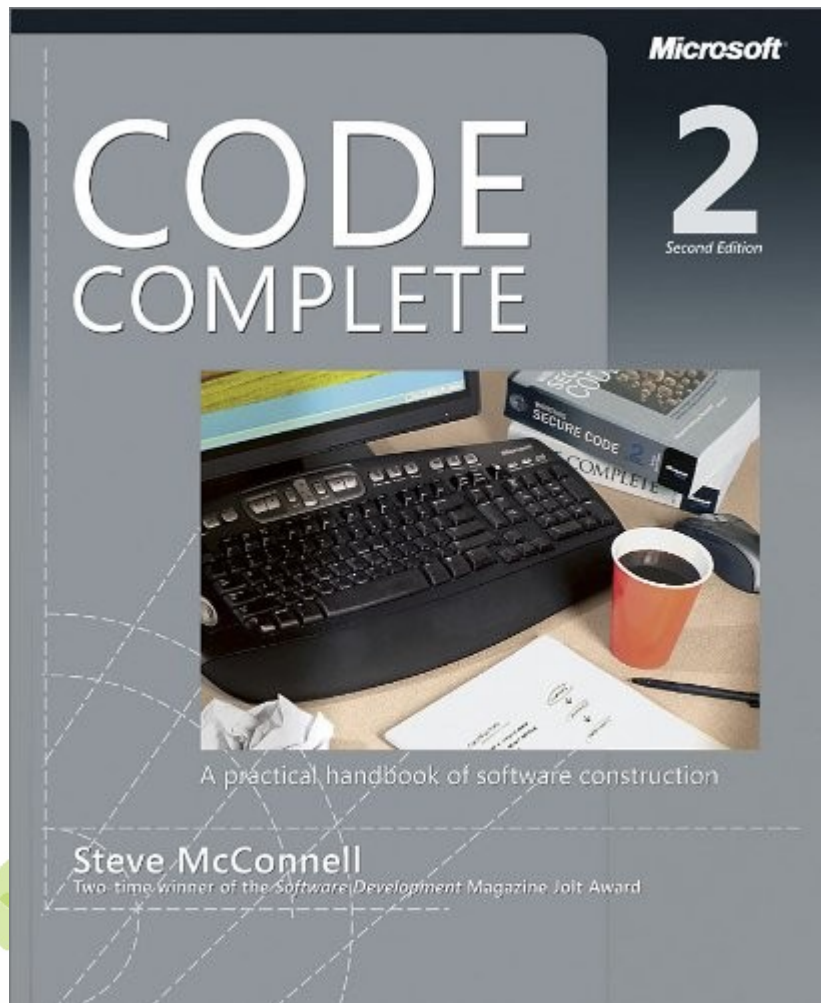






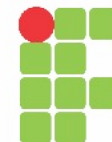




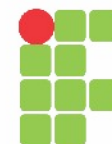


Então, foi assim que me interessei pela
Engenharia de Software.

O que eu faço agora?

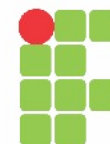


IFRS (a.k.a. antigo CTI)



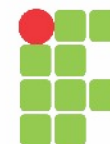
Cursos Técnicos

- Automação Industrial
- Eletrotécnica
- Fabricação Mecânica
- Geoprocessamento
- Refrigeração e Climatização
- Enfermagem

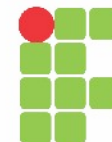


Cursos Superiores

- Tecnologia em Análise e Desenvolvimento de Sistemas
- Tecnologia em Construção de Edifícios
- Tecnologia em Refrigeração e Climatização
- Licenciatura para a Educação Profissional e Tecnológica



Fases de Desenvolvimento (Disciplinas)



Concepção, Requisitos

Análise, Especificação

Projeto de Alto Nível, Arquitetura

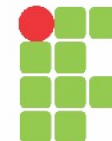
Projeto de Baixo Nível, Design

Implementação, Codificação

Teste, Integração, Homologação

Implantação

Operações, Manutenção



Técnico

Concepção, Requisitos

Análise, Especificação

Projeto de Alto Nível, Arquitetura

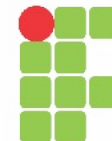
Projeto de Baixo Nível, Design

Implementação, Codificação

Teste, Integração, Homologação

Implantação

Operações, Manutenção



Tecnólogo

Concepção, Requisitos

Análise, Especificação

Projeto de Alto Nível, Arquitetura

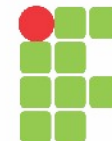
Projeto de Baixo Nível, Design

Implementação, Codificação

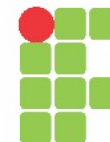
Teste, Integração, Homologação

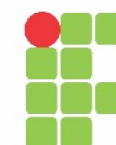
Implantação

Operações, Manutenção



Pontos de vista pragmáticos, aplicados

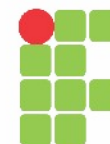








Planejar e Projetar Softwares



++Programador == Projetista

Law of the instrument

“Dê a um garotinho um martelo, e ele achará que tudo ao seu alcance necessita de uma martelada”



Engenharia de Software

Disciplinas *design & construction*

Concepção, Requisitos

Análise, Especificação

Projeto de Alto Nível, Arquitetura

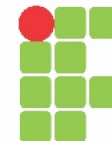
Projeto de Baixo Nível, Design

Implementação, Codificação

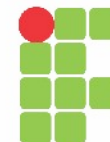
Teste, Integração, Homologação

Implantação

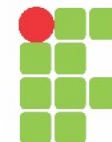
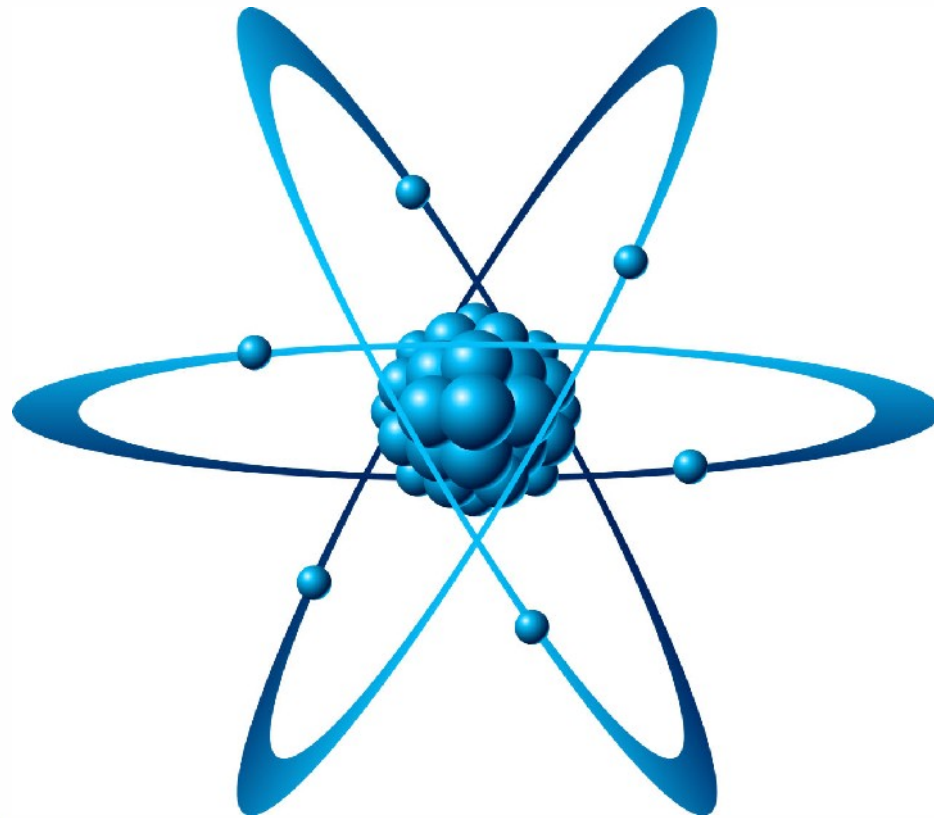
Operações, Manutenção



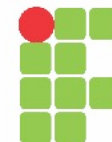
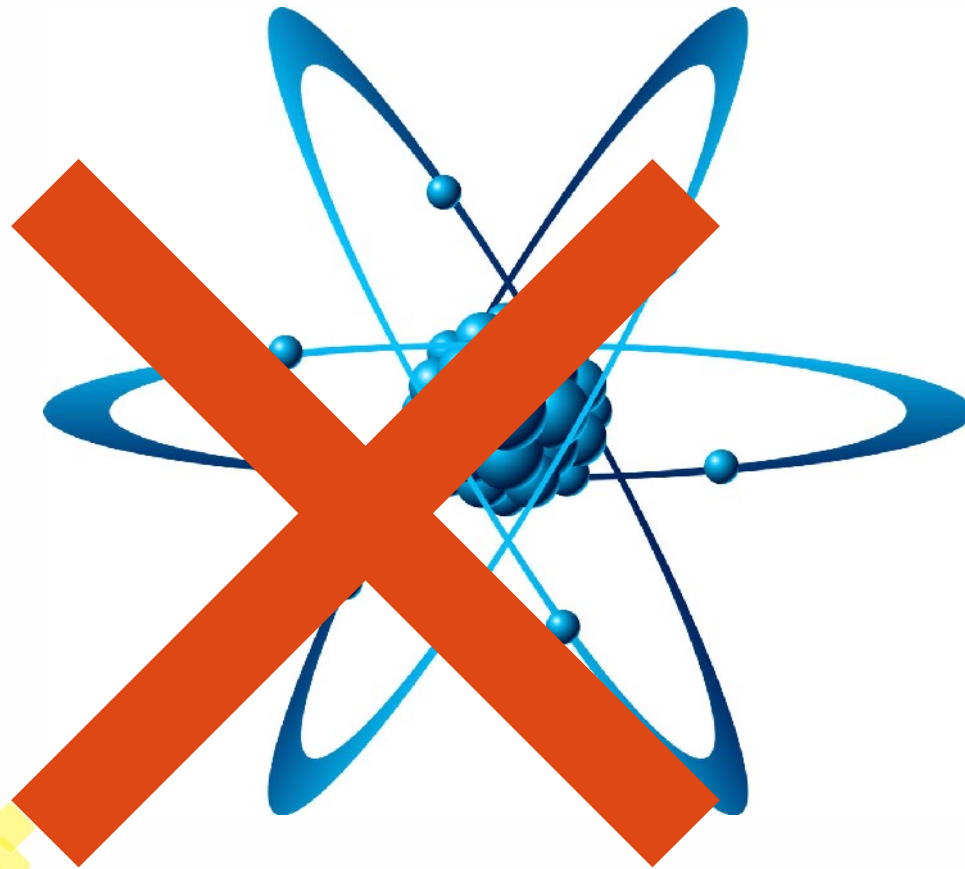
ES != outras Engenharias



Outras Engenharias constróem sobre matéria
e se baseiam nas leis da física



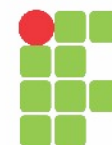
Engenharia de Software



O que você está me
dizendo? Que eu posso
desviar de balas?



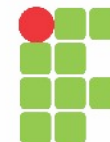
Não Neo. Quando você estiver pronto, você não precisará.





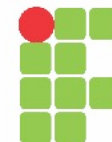
Objetivo em comum

Qualidade



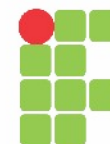
Atributos Qualitativos para Softwares (ISO 9126)

- Funcionalidade
- Confiabilidade
- Usabilidade
- Eficiência
- Manutenibilidade
- Portabilidade



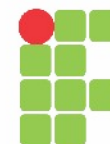
Atributos Qualitativos

- Funcionalidade
- Confiabilidade
- Usabilidade
- Eficiência
- Manutenibilidade
- Portabilidade

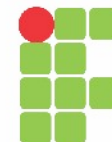


Atributos Qualitativos

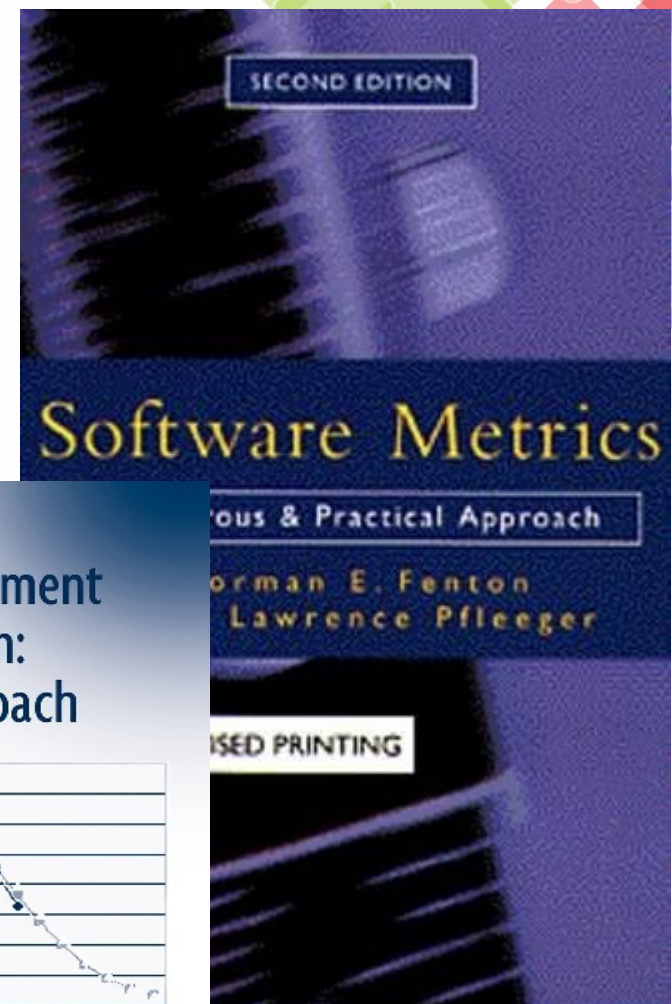
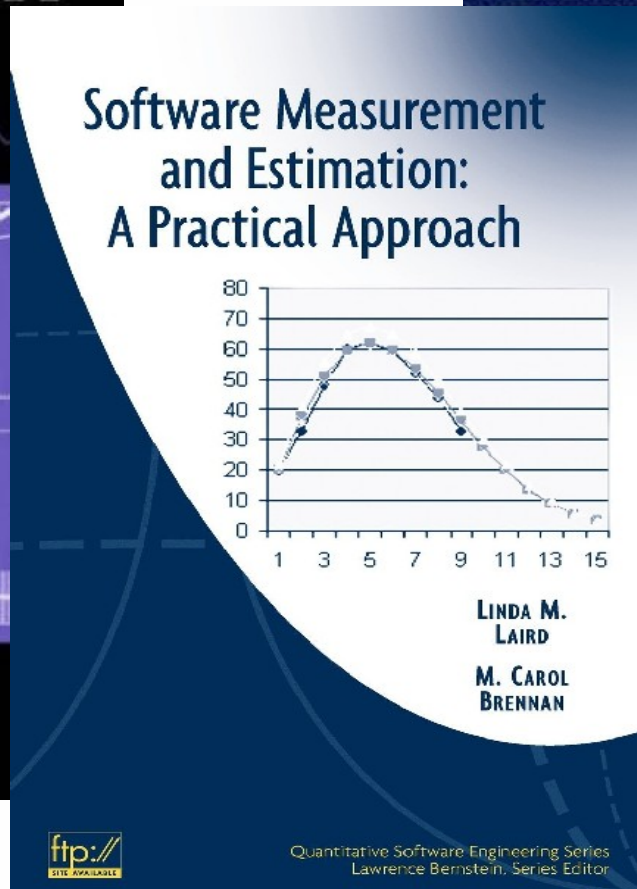
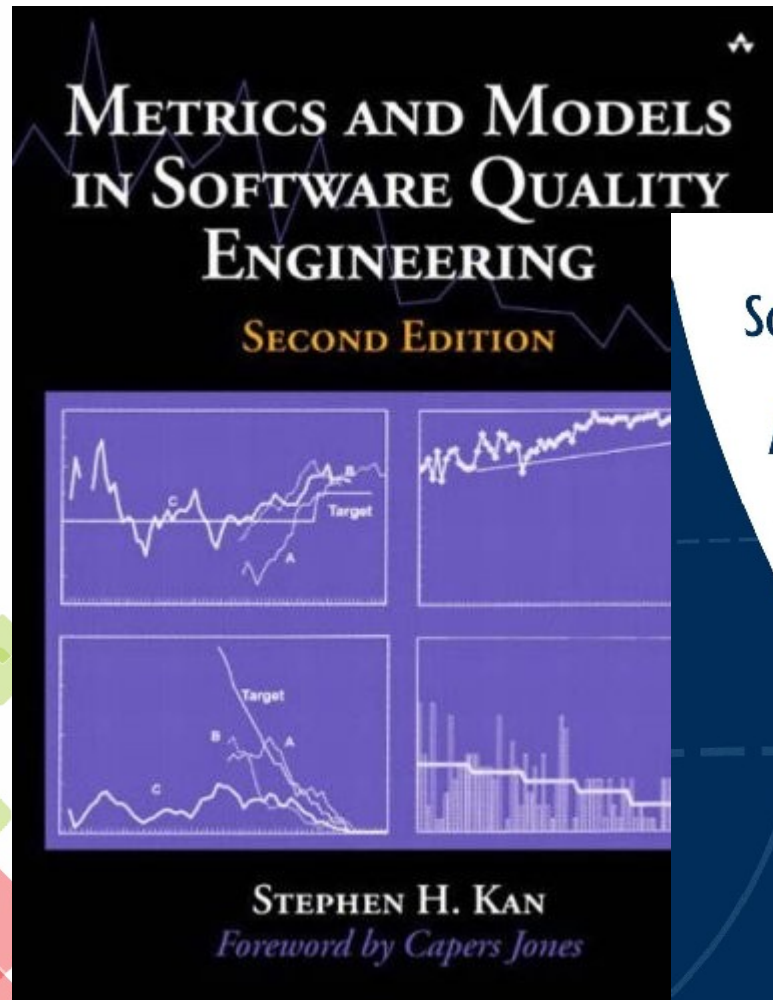
- Confiabilidade
 - Maturidade
- Eficiência
 - Utilização de recursos
- Manutenibilidade
 - Estabilidade
 - Modificabilidade



Um problema: dificuldade de medir

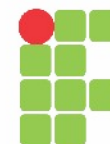


Medições/Métricas

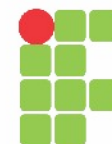


Otimizações e Métricas

- Otimizações de estrutura (projeto)
 - Métricas de acoplamento, coesão, complexidade ciclomática, etc
- Otimizações de algoritmo
 - Métricas de complexidade computacional, notações assintóticas, etc
- Otimizações de dados
 - Normalização, desnormalização, consistência, disponibilidade, particionamento, etc



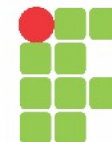
Por exemplo: vamos falar de performance ...



Mas lembre ...

“premature optimization is the root of all evil”

– Donald Knuth



Profiling (instrumentação)

VisualVM 1.0

File Applications View Tools Window Help

Applications

- Local
 - VisualVM
 - java2d.Java2Demo (pid 3812)
 - [snapshot] 01:30:42 odp.
 - Remote
 - Snapshots

Start Page x java2d.Java2Demo (pid 3812) x

Overview Monitor Threads Profiler [snapshot] 01:30:42 odp. x

java2d.Java2Demo (pid 3812)

Profiler Snapshot

View: Methods

Call Tree - Method	Time [%]	Time	Invocations
All threads		1068 ms (100%)	1
Intro		950 ms (100%)	1
java2d.Intro\$Surface.run ()		950 ms (100%)	1
Self time		822 ms (86,4%)	1
java2d.Intro\$Surface\$Scene.paus		128 ms (13,6%)	1
AWT-EventQueue-0		117 ms (100%)	1
javax.swing.SystemEventQueueUtilities		116 ms (99,6%)	21
java2d.Intro\$Surface.paint (java..		86.1 ms (73,4%)	18
Self time		30.0 ms (25,6%)	21
java2d.Java2Demo\$J2DIcon.paint		0.566 ms (0,5%)	3
sun.awt.windows.WComponentPee		0.097 ms (0,1%)	1
java2d.Java2Demo\$J2DIcon.getIc		0.021 ms (0%)	3
java2d.Java2Demo\$J2DIcon.getIc		0.004 ms (0%)	3
java2d.Java2Demo\$4.windowDeicon		0.260 ms (0,2%)	1
java2d.Java2Demo\$4.windowIconific		0.241 ms (0,2%)	1

Call Tree Hot Spots Combined Info

Profiling

The screenshot shows the Chrome Developer Tools Profiler interface. The top toolbar includes icons for Elements, Resources, Network, Scripts, Timeline, Profiles, Audits, and Console. The 'Profiles' tab is active, displaying a list of CPU profiles. The left sidebar shows 'CPU PROFILES' with 'Profile 4' selected. The main panel displays a table of profiling data for 'Profile 4'.

Self	Total	Function
98.41%	100.00%	▼ lin_solve navier-stokes.js:128
96.36%	97.87%	▼ project navier-stokes.js:239
56.57%	57.33%	▶ vel_step navier-stokes.js:284
39.79%	40.55%	▶ runNavierStokes navier-stokes.js:35
1.37%	1.44%	▼ diffuse navier-stokes.js:157
1.37%	1.44%	▼ dens_step navier-stokes.js:277
1.37%	1.44%	▼ FluidField.update navier-stokes.js:329
1.37%	1.44%	▼ runNavierStokes navier-stokes.js:35
1.37%	1.44%	▶ Measure base.js:202
0.68%	0.68%	▼ dens_step navier-stokes.js:277
0.68%	0.68%	▼ runNavierStokes navier-stokes.js:35
0.68%	0.68%	▶ Measure base.js:202

The bottom status bar shows 'Heavy (Bottom Up)' and a percentage icon, indicating the current view of the profile.



How Loading Time Affects Your Bottom Line

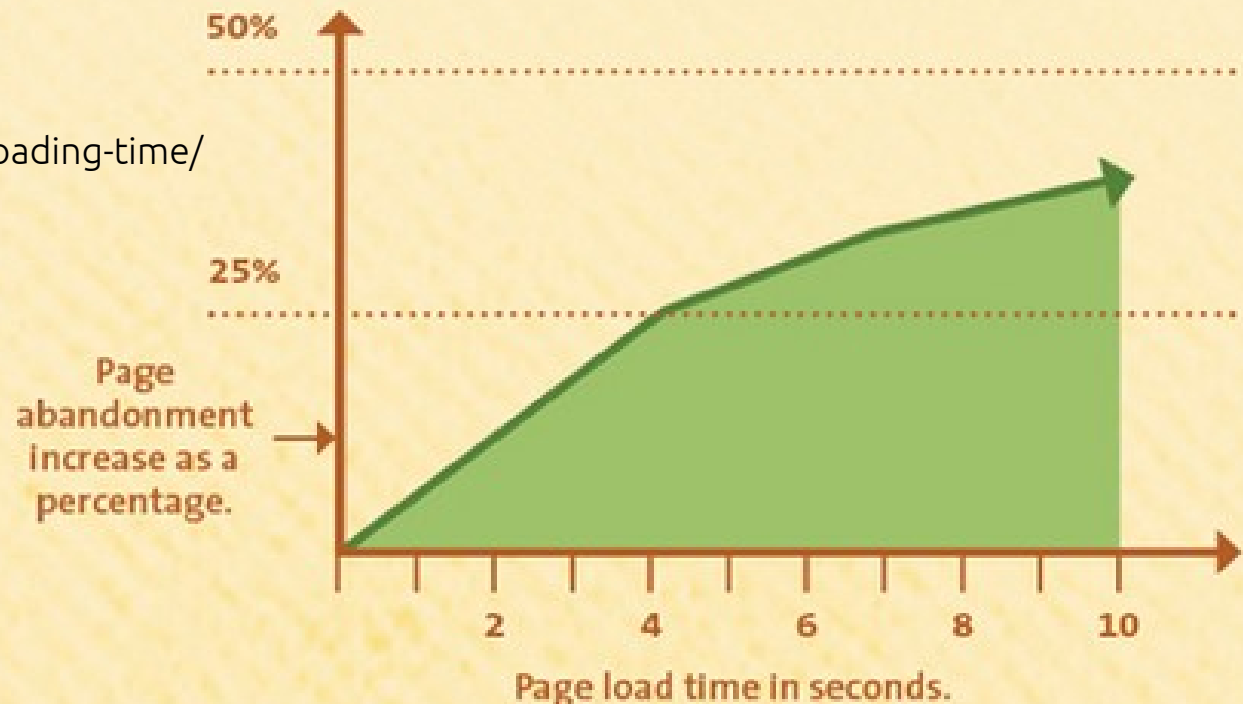
<http://blog.kissmetrics.com/loading-time/>



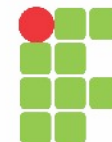
EVERY SECOND COUNTS

Loading time is a major contributing factor to page abandonment. The average user has no patience for a page that takes too long to load, and justifiably so.

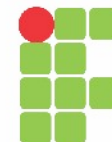
Observation: slower page response time results in an increase in page abandonment, as demonstrated in the following chart.



Escalabilidade!

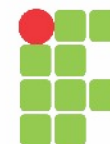


Vamos a parte prática: exemplos de boas práticas de projeto e implementação com a linguagem Java (e outras que forem aparecendo)



Práticas para obter performance

Regra geral: conheça as especificidades da linguagem

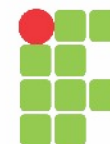


Números

- Não use os *wrappers*, prefira os primitivos

```
// em vez de:  
Integer n = 10;
```

```
// prefira:  
int n = 10;
```



Strings

- Não use new para instanciar Strings, declare literalmente
- Não concatene Strings em loop, use StringBuilder
- Não converta números para Strings concatenando-os com ""

```
// em vez de:  
String s = new String("teste");
```

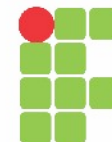
```
// prefira:  
String s = "teste"
```

```
// em vez de:  
s = s + "string"
```

```
// prefira:  
s.append("string")
```

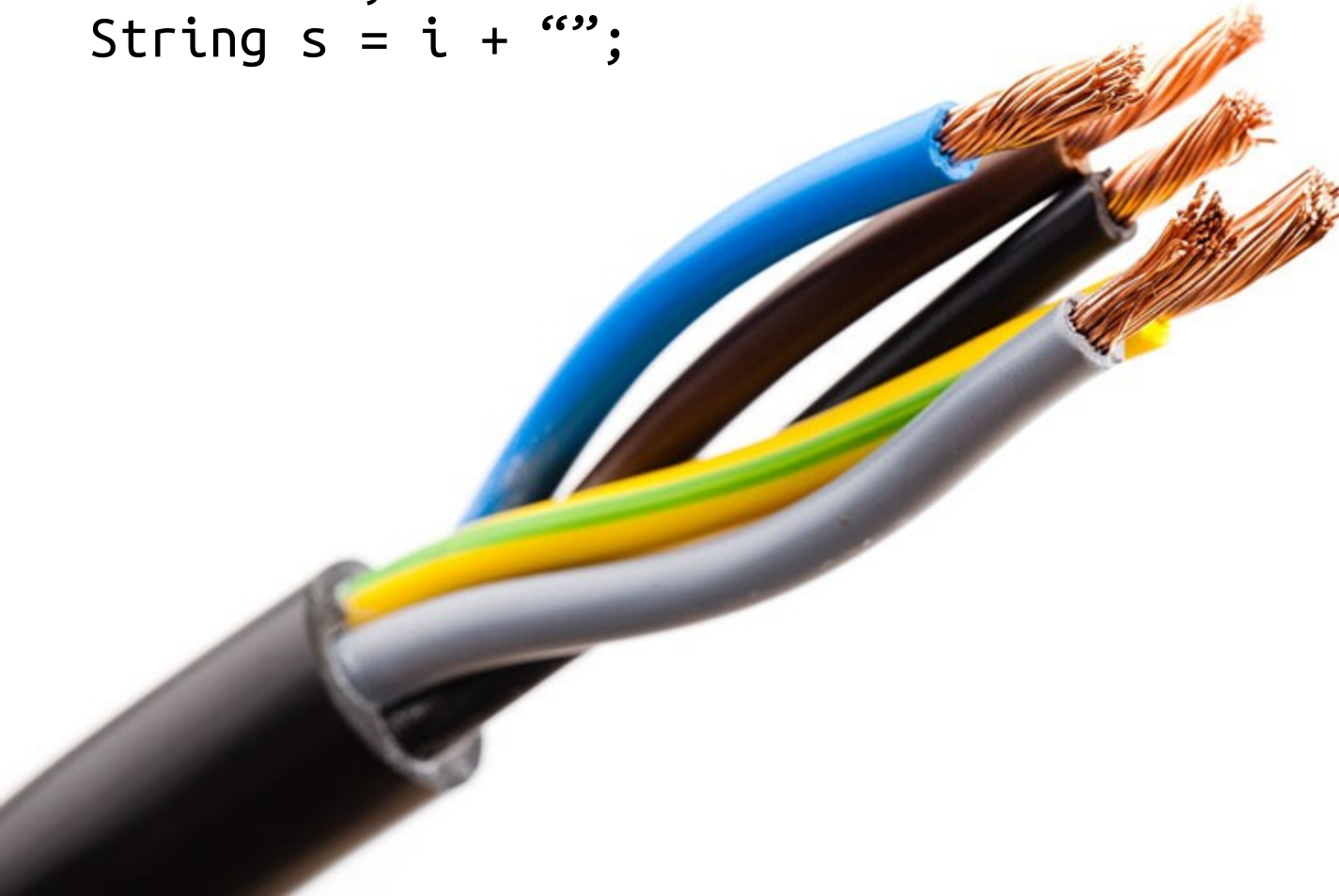
```
// em vez de:  
String s = 10 + "";
```

```
// prefira:  
String s =  
String.valueOf(10);
```



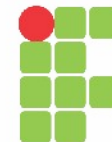
Kludges, Hacks, ou *gambiarrras*

```
int i = 5;  
String s = i + "";
```



demonstrar ...

{Java: [Strings, Numeros]}

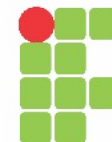


Em JavaScript e PHP

- Evite consultar o length de um array dentro do for, atribua antes

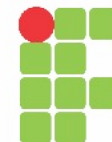
```
// em vez de:  
for ($i = 0; $i < count($elementos); $i++)
```

```
// prefira:  
for ($i = 0, $count = count($elementos);  
    $i < $count; $i++)
```



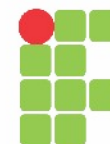
demonstrar ...

{PHP: iterando-arrays}



Práticas para obter confiabilidade e estabilidade

Regra geral: programe defensivamente

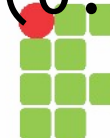


Números

- Não use float ou double quando precisão for necessária, prefira BigDecimal

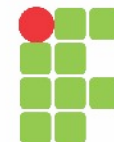
```
// em vez de:  
double v1 = 1.1;  
double v2 = 0.1;  
double v3 = v1 + v2;
```

```
// prefira:  
BigDecimal v1 = BigDecimal.valueOf(1.1);  
BigDecimal v2 = BigDecimal.valueOf(0.1);  
BigDecimal v3 = v1.add(v2);
```



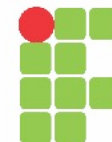
Strings

- Não use `==` para comparar Strings, use `equals`
- Prefira `literal.equals(variavel)` em vez de `variavel.equals(literal)`



demonstrar ...

{Java: [Strings, Numeros]}



Yoda Conditions



**CALM YOU
SHALL KEEP
AND
CARRY ON
YOU MUST**

YES, HMMMM

Yoda Condition Notation em C/C++

- Compare o literal com a variável em condicionais de igualdade (útil em PHP, JavaScript, C, C++)

```
// em vez de:  
if (v == 1) ...  
while (v == 1) ...
```

```
// prefira:  
if (1 == v) ...  
While (1 == v) ...
```



```
if (5 == count)
```

Coding Conventions



comp.lang.c FAQ list • Question 17.4

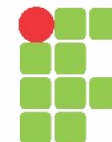
Q: Why do some people write `if(0 == x)` instead of `if(x == 0)`?

A: It's a trick to guard against the common error of writing

```
if(x = 0)
```

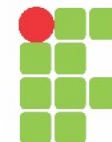
If you're in the habit of writing the constant before the `==`, the compiler will complain if you accidentally type

```
if(0 = x)
```



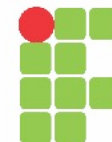
demonstrar ...

{CCPP: [comparacao]}



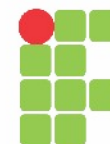
Práticas para obter modificabilidade

Regra geral: escreva códigos fáceis de entender



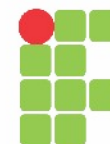
Custo para modificar

Custo para Entender + Custo para Implementar



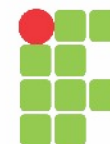
Custo para modificar

Custo para Entender + Custo para Implementar



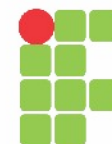
Dê bons nomes

- Evite nomes com só uma letra, com exceção se é um padrão conhecido
- Dê nomes que revelem o propósito



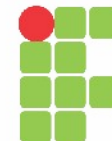
Para métodos

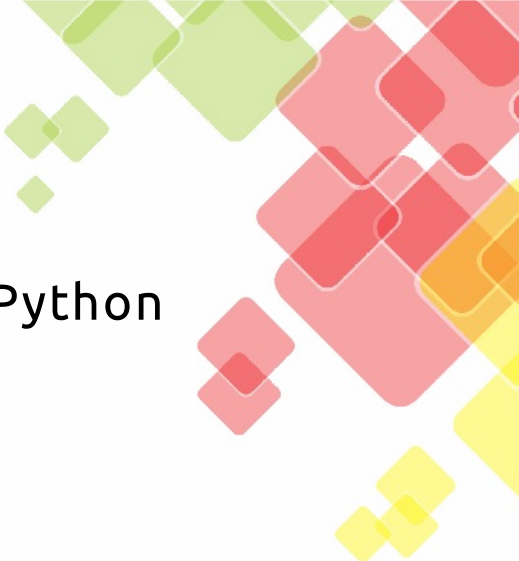
- Evite muitos parâmetros, caso necessário use um objeto parâmetro
- Projete para que nulos não sejam passados nos argumentos, aproveite a sobrecarga
- Evite parâmetros booleanos, strings ou numéricos para identificar opções



```
// em vez de:  
int m = 10;  
// prefira:  
int minutesToWait = 10;
```

```
// em vez de:  
convert(String str, int case)  
// prefira:  
convert(String str, LetterCase case)
```





Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Zen of Python



Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although **never** is often better than **right** now.

If the implementation is hard to explain, it's a bad idea.

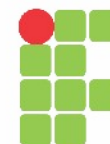
If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

Zen of Python

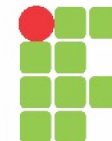


Não comente seu código ...



... pelo menos, não assim:

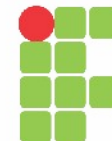
```
class TransferStat {  
    // bytes recebidos  
    int br;  
  
    // construtor  
    public TransferStat() {  
    }  
    ...
```



... pelo menos, não assim:

```
class TransferStat {  
    // bytes recebidos  
    int br;  
  
    // construtor  
    public TransferStat() {  
    }  
    ...
```

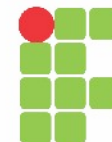
```
class TransferStat {  
    int bytesReceived;  
    public TransferStat() {  
    }  
    ...
```



Bom código é sua própria, e melhor, documentação. Sempre que estiveres por adicionar um comentário, pergunte a si mesmo:

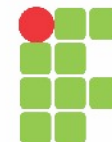
“Como eu posso melhorar este código para que esse comentário não seja necessário?”

– Steve McConnell



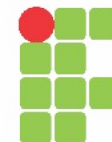
demonstrar ...

{Java: [Methods]}



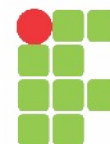
Práticas para obter bons sistemas

Regra geral: contrate bons programadores



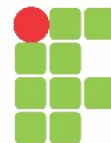

```
if ($numero_nota >= 1 && $numero_nota < 10) {  
    $numero_nota = '00000000' . $numero_nota;  
} else if ($numero_nota >= 10 && $numero_nota < 100) {  
    $numero_nota = '0000000' . $numero_nota;  
} else if ($numero_nota >= 100 && $numero_nota < 1000) {  
    $numero_nota = '000000' . $numero_nota;  
} else if ($numero_nota >= 1000 && $numero_nota < 10000) {  
    $numero_nota = '00000' . $numero_nota;  
} else if ($numero_nota >= 10000 && $numero_nota < 100000) {  
    $numero_nota = '0000' . $numero_nota;  
} else if ($numero_nota >= 100000 && $numero_nota < 1000000) {  
    $numero_nota = '000' . $numero_nota;  
} else if ($numero_nota >= 1000000 && $numero_nota < 10000000) {  
    $numero_nota = '00' . $numero_nota;  
} else if ($numero_nota >= 10000000 && $numero_nota < 100000000) {  
    $numero_nota = '0' . $numero_nota;  
} else if ($numero_nota >= 100000000 && $numero_nota < 1000000000) {  
    $numero_nota = '' . $numero_nota;  
}
```

<https://gist.github.com/marciojrtores/6190421>

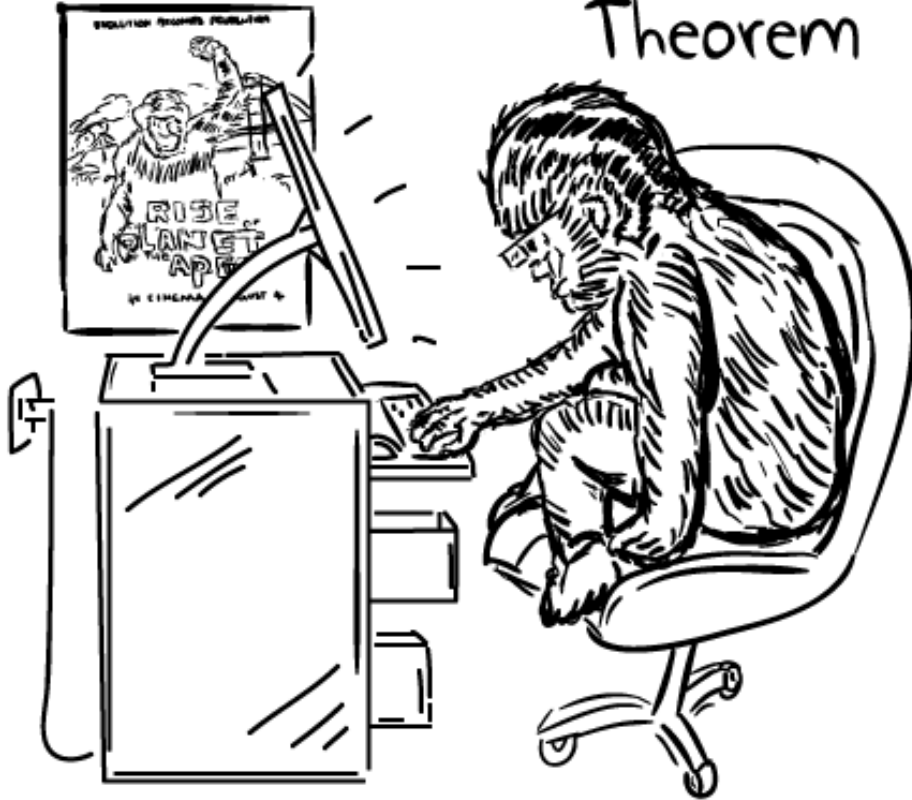


Teorema do Macaco Infinito

Um macaco digitando aleatoriamente em um teclado por um intervalo de tempo infinito irá certamente criar um texto qualquer escolhido, como por exemplo a obra completa de William Shakespeare.

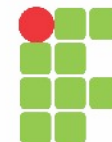


The Infinite Code-Monkey Theorem



After an infinite amount of time, a hypothetical programming monkey would eventually give up on solving $P = NP$

Não seja um programador macaco, aprenda a projetar seus programas

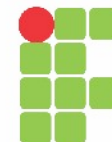


Princípio DRY: Don't Repeat Yourself

"cada pedaço de conhecimento deve ter uma única não ambígua e autêntica representação em um sistema"

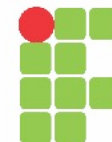
Um dos piores, senão o pior, problemas no desenvolvimento de softwares:

código duplicado



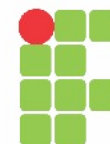
demonstrar ...

{Python: [dry-ok, dry-not-ok]}



Existem inúmeras práticas e princípios de projeto e implementação.

<http://academico.riogrande.ifrs.edu.br/~marcio.torres/site/livros/aps-novo.pdf>



Perguntas?



marcio.torres@riogrande.ifrs.edu.br

