AdaptSuite
Ferramenta para Seleção Automática de Testes

André Pestana
Orientador: Prof. Dr. Alfredo Goldman
gold@ime.usp.br

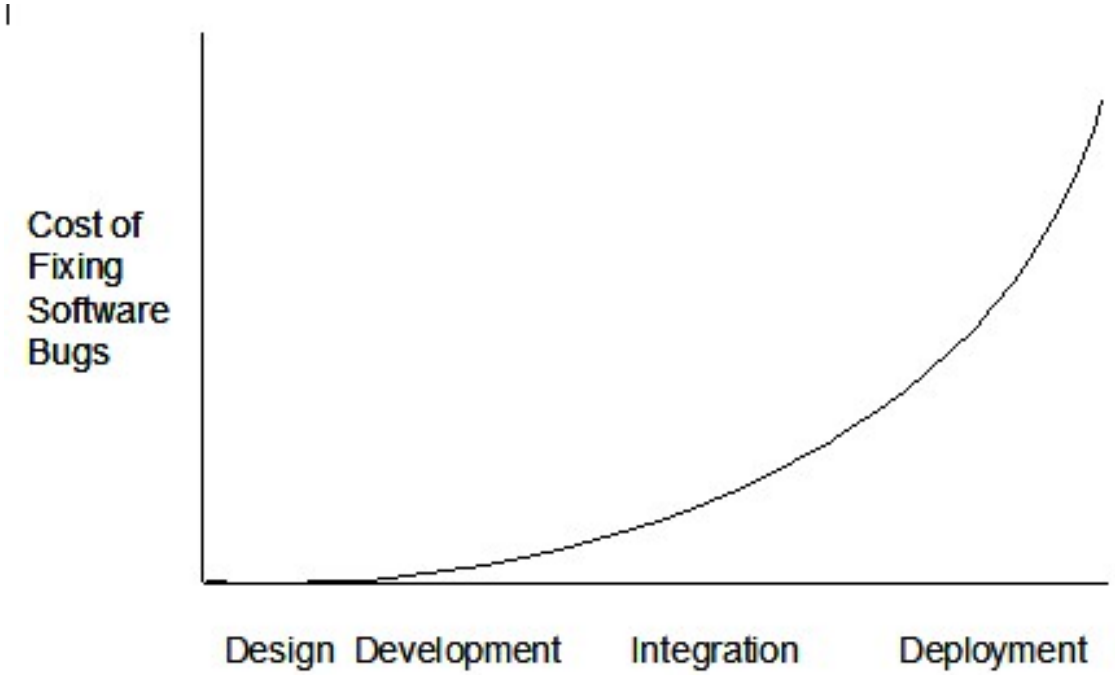
Universidade de São Paulo

andre.pestana@usp.com

Conteúdo

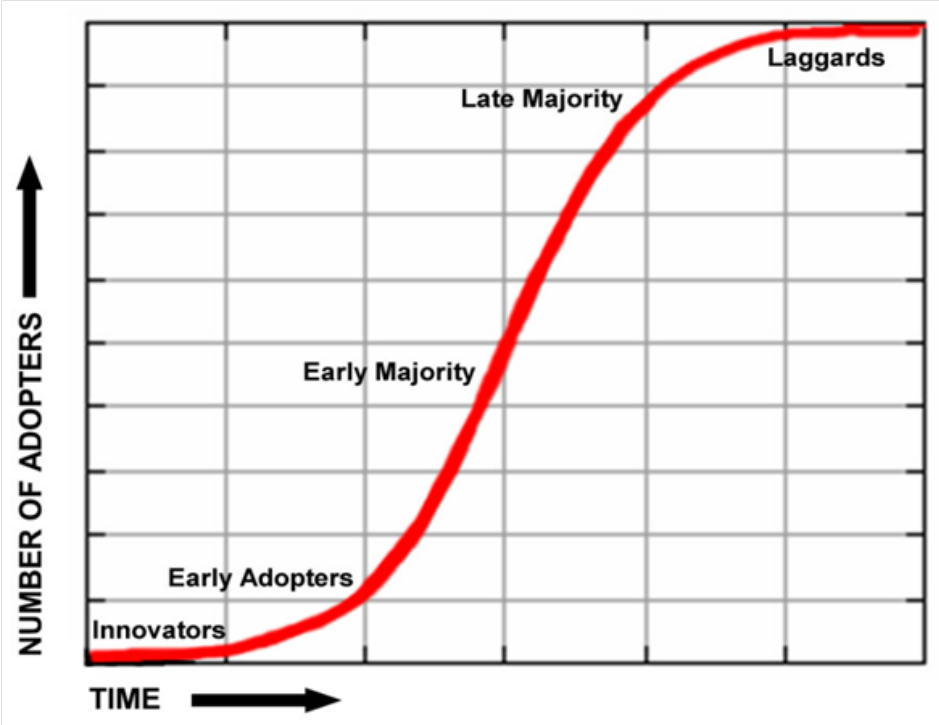
Engenharia Tradicional

- Testes são deixados para o final. [1]
- São caros e demorados. [3]
- Etapas anteriores atrasam.



Métodos Ágeis

- Constante desenvolvimento.
- Manuntenção vs. Implementação. [3] [8]
- Tempo proporcional ao número de usuários. [4]



Metodologia

Ideia

Encontrar suite otimal [7] dentro de um tempo limite

Princípio

Cada teste tem seu valor e tempo de execução.

Algorítmo

- Problema da Mochila (MAC0338)
- Resolvido de maneira dinâmica

$t[i, Y] = t[i-1, Y]$ **se** $w[i] > Y$

$t[i, Y] = \max\{t[i-1, Y], t[i-1, Y-w[i]] + v[i]\}$ **se** $w[i] \leq Y$

i \ Y	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	500	500
2	0	0	400	400	500	500
3	0	300	400	400	700	800
4	0	300	400	450	750	850

$\Theta(nW)$

Obtendo mochila

```
MOCHILA (w, n, W, t)
1   Y ← W
2   para i ← n decrescendo até 1 faça
3       se t[i, Y] = t[i−1, Y]
4           então x[i] ← 0
5           senão x[i] ← 1
6                     Y ← Y − w[i]
7   devolva x
```

Consumo de tempo é $\Theta(n)$.

Valor de um Teste

1. Quantidade de falhas registradas pelo teste [5]
2. Cobertura em linhas
3. Última vez que o teste foi executado
4. Frequencia com que o teste é executado pela ferramenta

Prioridades

Prioridades entre 0 e 5 para cada variável.

Valor de um Teste

Definição Formal

Seja T um teste qualquer, E_T a quantidade de erros referente ao teste T, C_T a cobertura, U_T a última vez que foi executado, F_T e P_1, P_2, P_3, P_4 os pesos desses parâmetros que podem ser modificados pelo usuário. O valor V_T associado à esse teste é definido como:

$$V_T = (P_1 * E_T) * (P_2 * C_T) * (P_3 * U_T) * (P_4 * F_T) \quad (1)$$

Instanciando Objeto

```
import main.java.org.adaptsuite.suite.AdaptSuiteBuilder;

@RunWith(AllTests.class)
public class Seconds3Suite {
    public static TestSuite suite(){
        return new AdaptSuiteBuilder().sec(3).build();
    }
}
```

Tempo Máximo

Pode-se usar os comandos *mili sec*, *min*.

Todos os testes serão executados se nada for passado.

Iniciando com Prioridades

```
import main.java.org.adaptsuite.suite.AdaptSuiteBuilder;

@RunWith(AllTests.class)
public class Seconds3Suite {
    public static TestSuite suite(){
        Map <String, Long> relevance = new HashMap<String, Long>();
        relevance.put(AdaptSuiteBuilder.getErrorConstant(), 2L);
        relevance.put(AdaptSuiteBuilder.getFrequencyConstant(), 4L);
        relevance.put(AdaptSuiteBuilder.getLastExecutionConstant(), 0L);
        return new AdaptSuiteBuilder().sec(3).build(relevance);
    }
}
```

Constantes disponíveis são:

- getErrorConstant
- getCoverageConstant
- getLastExecutionConstant
- getFrequencyConstant

Testes

Teste 1

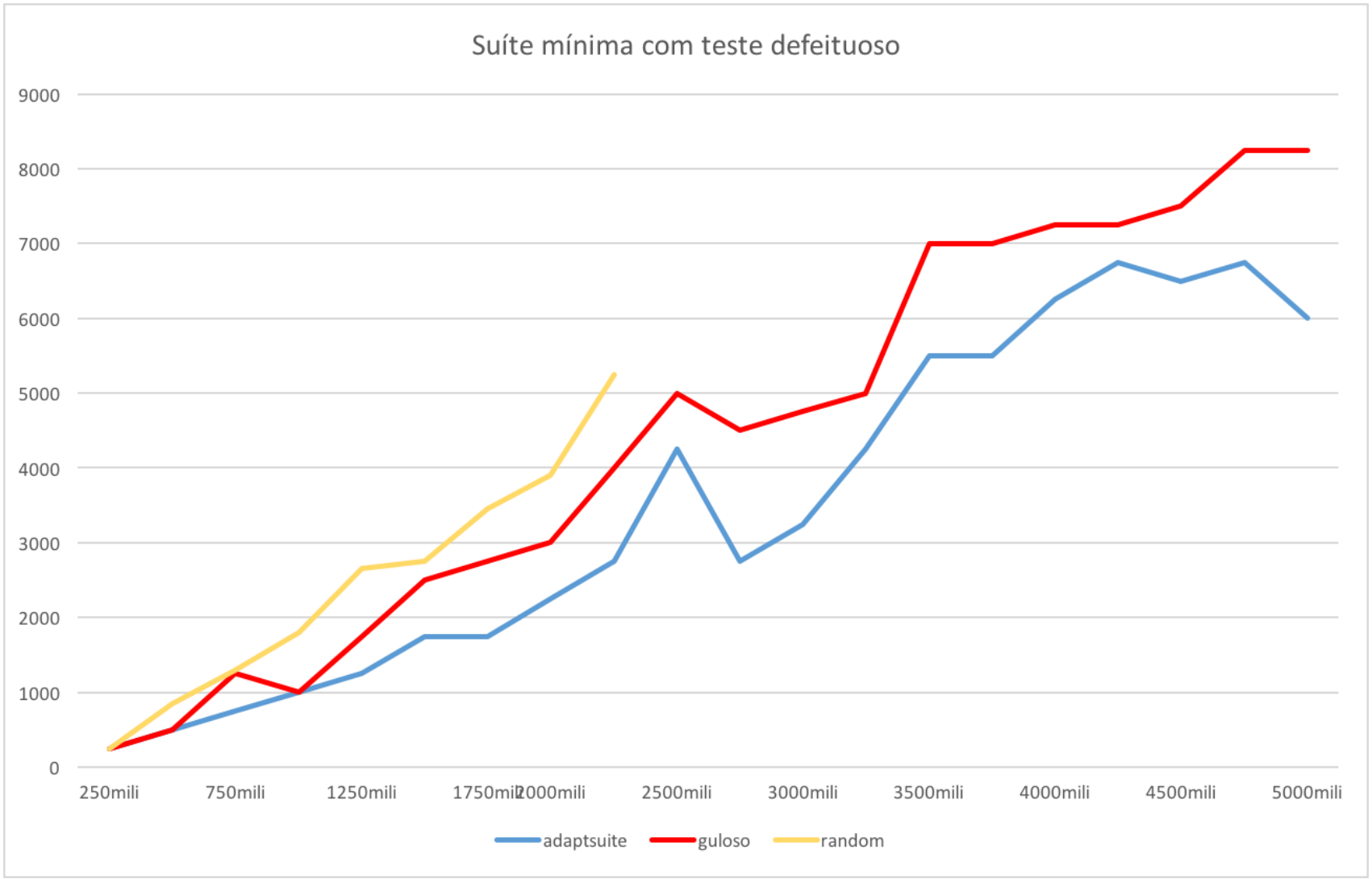
Encontrar um teste defeituoso.

Teste 2

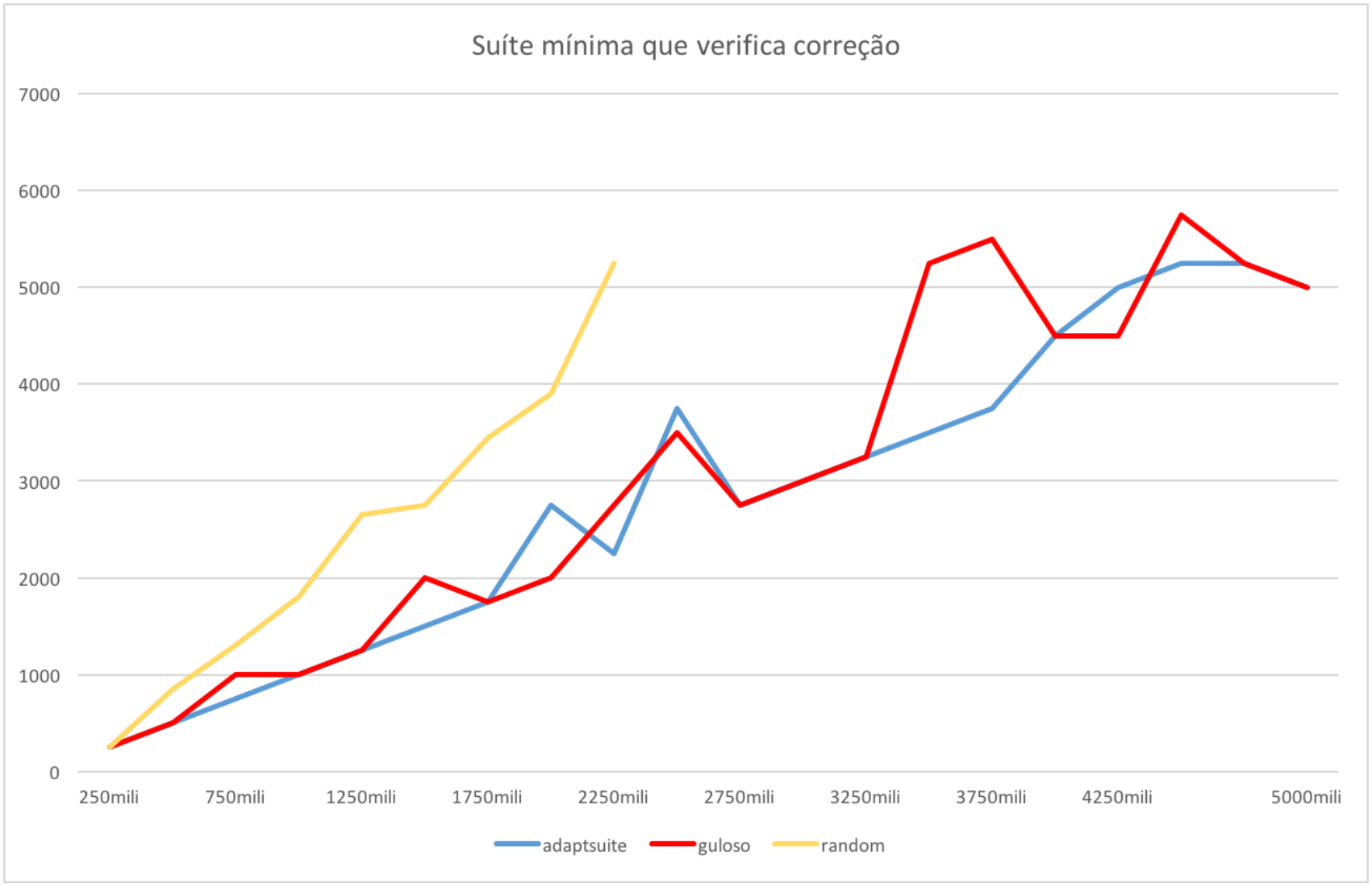
Verificar que um teste foi corrigido.

- 20 testes (0.25s à 5s)
- adaptsuite vs guloso vs aleatório
- Teste de duração n exercitado com suite de tamanho $2n$

Teste 1

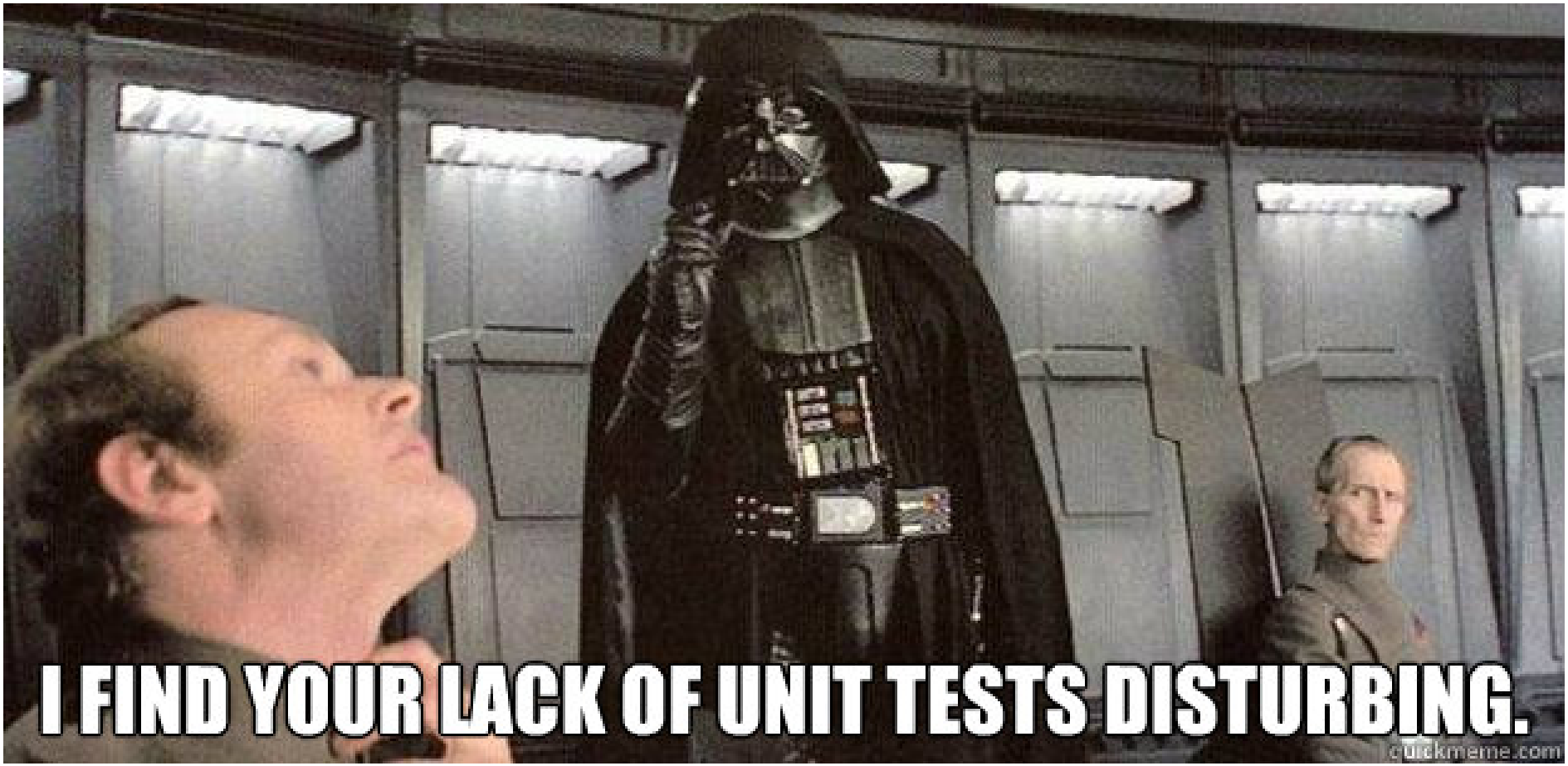

Suíte mínima com teste defeituoso
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
250mili
750mili
1250mili
1750mili
2000mili
2500mili
3000mili
3500mili
4000mili
4500mili
5000mili
adaptsuite
guloso
random

Teste 2


Suíte mínima que verifica correção
7000
6000
5000
4000
3000
2000
1000
0
250mili
750mili
1250mili
1750mili
2250mili
2750mili
3250mili
3750mili
4250mili
5000mili
adaptsuite
guloso
random

Conclusões

1. Solução proposta é mais eficiente em detectar falhar e confimar que as mesmas foram corrigidas.
2. Sempre teste o código. De preferência com critérios.
3. Há muito o que se pesquisar na área.


I FIND YOUR LACK OF UNIT TESTS DISTURBING.
quickmeme.com

Referências I

Catal, C., Mishra, D.

Test case prioritization: a systematic mapping study.

Springer Science+Business Media, 2012.

Elbaum, S., Kallakuri, P., Malishevsky, A. G., Rothermel, G., Kanduri, S.

Understanding the effects of changes on the cost-effectiveness of regression testing techniques.

Journal of Software Testing, Verification, and Reliability, 2003.

Pressman, R. S.

Engenharia de Software - Uma Abordagem Profissional.

AMGH Editora, 7 edição, 2011.

Rogers, E. M.

Diffusion of innovations.

New York Free Press 5.ed., 2003

Referências II

Rothermel, G., Untch, R. H., Chu, C., Harrold, M.J.

Prioritizing test cases for regression testing, volume 27.

IEEE Transactions on Software Engineering, 2001.

Wu, K., Fang, C., Chen, Z., Zhao, Z.

Understanding the effects of changes on the cost-effectiveness of regression testing techniques.

Test Case Prioritization Incorporating Ordered Sequence of Program Elements, 2012.

Yu, Y. T., Lau, M. F.

Fault-based test suite prioritization for specification-based testing, volume 54.

Information and Software Technology, 2012.

Mens, T., Demeyer S.

Software Evolution.

Editora Springer, 1 edição, 2008.

Dúvidas?