DEPURAÇÃO DE SOFTWARE

Prof. Marcelo Medeiros Eler

marceloeler@usp.br

EACH | campus USP LESTE
Escola de Artes, Ciências e Humanidades
Universidade de São Paulo

USP

OBJETIVOS DA AULA

- Apresentar conceitos principais e algumas abordagens que oferecem apoio automatizado à atividade de depuração de software

CONCEITOS

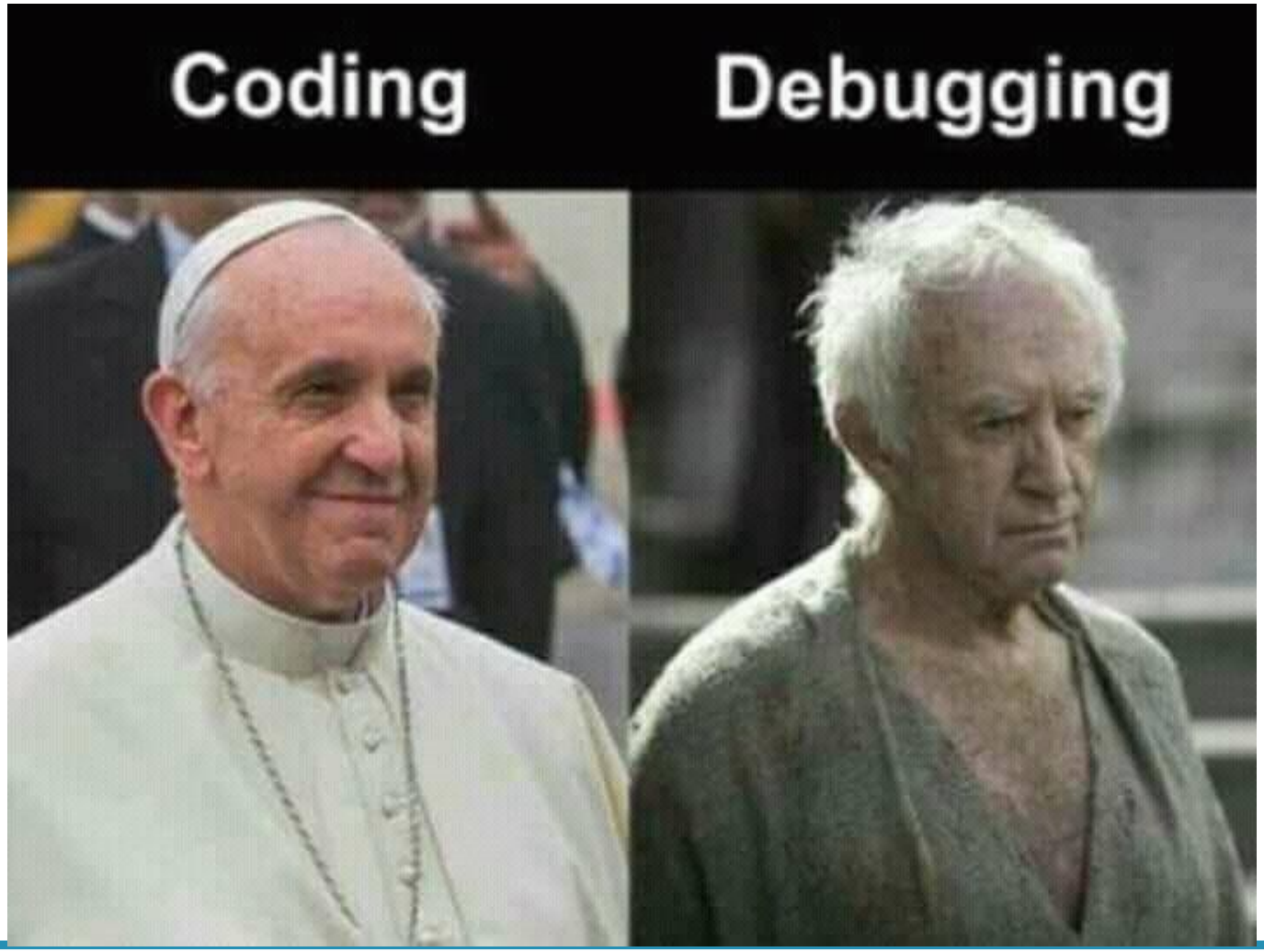
- Depuração
  - Processo de localizar a origem/causa de defeitos

CONCEITOS

- Depuração
  - Tarefa custosa.
  - Realizada ainda hoje, em grande parte, de maneira manual.
  - Ou com ferramentas criadas no final dos anos 50.

UM POUCO DE HUMOR

UM POUCO DE HUMOR



HUMOR

CONCEITOS

- Depuração
  - Tarefa custosa.
  - Realizada ainda hoje, em grande parte, de maneira manual.
  - Ou com ferramentas criadas no final dos anos 50.

Modelos de depuração

- Criação e validação de hipóteses [Araki et al., 1991]
  - Programadores elaboram hipóteses sobre a possíveis localização de defeitos.
  - Confirmação (ou refutação) de hipóteses → novas hipóteses.
  - Novas hipóteses são confirmadas (ou refutadas) → defeito.

Modelos de depuração

- Predador-caça [Lawrance et al., 2013]:
  - O programador (predador) navega pelo código...
  - observando dicas e avaliando a chance (cheiro)...
  - de chegar ao defeito (presa).

UM POUCO DE HUMOR

Modelos de depuração

Estudos indicam que:

- Modelo predador-caça é prevalente;
- Criação e validação de hipóteses também é utilizada.

TÉCNICAS DE DEPURAÇÃO

- Rastreamento e Inspeção
- Fatiamento de Programas (Program slicing)
- Depuração Baseada em Cobertura
- Reparo Automático de Programas

RASTREAMENTO E INSPEÇÃO

- Rastrear pontos do programa (eventos).
- Inspecionar o valor das variáveis escolhidas (estado parcial do programa).
- Formas básicas:
  - inserção de comandos de escrita (logs) em pontos estratégicos.
  - bibliotecas para logging (e.g., Log4J).

RASTREAMENTO E INSPEÇÃO

- Logging requer escrever código, compilar e rodar o programa novamente.
- Os depuradores simbólicos tornam essas tarefas transparentes e fornecem mais recursos.

DEPURADORES SIMBÓLICOS

- Recebem notificações sempre que eventos de interesse ocorrem.
- Eventos de interesse:
  - Breakpoints;
  - Watchpoints.
- Inspeção:
  - Stepping : execução passo-a-passo;
  - Acesso ao estado do programa: valores de variáveis, pilha de execução.

DEPURADORES SIMBÓLICOS

- Permitem inspecionar:
  - O estado da pilha de execução;
  - Os valores de variáveis;
  - Posições de memórias.
- Eles também permitem:
  - Alterar valores de variáveis;
  - Avaliar expressões;
  - Monitorar posições de memória (watchpoints).

FATIAMENTO DE PROGRAMAS

- Uma fatia é um subconjunto do programa
- Ajuda a visualização do código relevante a uma execução de interesse
- Fatias de programa removem partes irrelevantes do código
- Fatias podem ser criadas estática ou dinamicamente.
- Auxiliam na localização de defeitos

DEFINIÇÃO DE FATIAS

- P é o programa
- V é o conjunto de variáveis em uma localização do programa – linha n
- Uma Fatia F(V,n) produz com porções do programa que contribuem ou influenciam o valor de V antes da execução da linha n
- F(V,n) é chamado de critério de fatiamento

CONDIÇÕES DE FATIAMENTO

- F(V,n) tem que ser derivada de P com a remoção de instruções de P
- F(V,n) precisa ser sintaticamente correta
- O valor de V antes da linha n da execução F(V,n) deve ser igual ao valor de V da execução P antes da linha n

EXEMPLO

```
main() {
1. int mx, mn, av;
2. int tmp, sum, num;
3.
4. tmp = readInt():
5. mx = tmp;
6. mn = tmp;
7. sum = tmp;
8. num = 1;
9.
10. while(tmp >= 0)
11. {
12. if (mx < tmp)
13. mx = tmp;
14. if (mn > tmp)
15. mn = tmp;
16. sum += tmp;
17. ++num;
18. tmp = readInt();
19. }
20.
21. av = sum / num;
22. printf("\nMax=%d", mx);
23. printf("\nMin=%d", mn);
24. printf("\nAvg=%d", av);
25. printf("\nSum=%d", sum);
26. printf("\nNum=%d", num);
}
```

F(num,26)

```
main() {
2. int tmp, num;
4. tmp = readInt():
8. num = 1;
10. while(tmp >= 0)
11. {
17. ++num;
18. tmp = readInt();
19. }
26. printf("\nNum=%d", num);
 }
```

F(av,24)

```
main() {
1. int av;
2. int tmp, sum, num;
4. tmp = readInt():
7. sum = tmp;
8. num = 1;
10. while(tmp >= 0)
11. {
16. sum += tmp;
17. ++num;
18. tmp = readInt();
19. }
21. av = sum / num;
24. printf("\nAvg=%d", av);
}
```

PROCESSO DE FATIAMENTO

- Selecione o critério de fatiamento
  - Variável ou um conjunto de variáveis e uma linha de Código
- Gere a fatia do programa
- Execute testes e faça a depuração na fatia do programa
- Modifique a fatia e faça a fusão com o programa original

FATIAMENTO DE PROGRAMAS

- É possível aplicar fatiamento dinâmico
  - Contexto de uma execução (caso de teste)
- Limitações
  - As fatias podem ficar muito grandes em programas grandes
- Otimizações
  - Uso de heurísticas para reduzir mais o espaço de busca

DEPURAÇÃO BASEADA EM COBERTURA

- Utiliza informações da execução de casos de teste para auxiliar a localização de defeitos
  - Cobertura de código
- O objetivo é identificar trechos de código que tem maior chance de conter defeitos

DEPURAÇÃO BASEADA EM COBERTURA

- Testes que passam
  - Código executado => menos suspeito
  - Código não executado => mais suspeito
- Testes que falham
  - Código executado => mais suspeito
  - Código não executado => menos suspeito

DEPURAÇÃO BASEADA EM COBERTURA

- Coeficientes

	Testes que falharam	**Testes que passaram**
Executou	CEF	CEP
Não executou	CNF	CNP

- Tarantula

$$\frac{CEF/(CEF+CNF)}{CEF/(CEF+CNF) + CEP/(CEP+CNP)}$$

DEPURAÇÃO BASEADA EM COBERTURA

Linha	Bloco	Código
1	1	int max(int array [], int length)
2	1	{
3	1	int i = 0;
4	1	int max = array[++i]; **//array[i++];**
5	2	while(i < length){
6	3	{
7	3	if(array[i] > max)
8	4	max = array[i];
9	5	i = i + 1;
10	5	}
11	6	return max;
12	6	}

DEPURAÇÃO BASEADA EM COBERTURA

- Casos de teste

T_n	Entrada	Saída	Resultado
t_1	max([1,2,3] , 3)	3	✓
t_2	max([5,5,5] , 3)	5	✓
t_3	max([2,10,1] , 3)	10	✓
t_4	max([4,2,3] , 3)	3	✗
t_5	max([4] , 1)	Exceção	✗

DEPURAÇÃO BASEADA EM COBERTURA

- Cálculo do grau de suspeição

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*					0	1	0	0	
4	1	*					0	1	0	0	
5	2	*					0	1	0	0	
7	3	*					0	1	0	0	
8	4	*					0	1	0	0	
9	5	*					0	1	0	0	
11	6	*					0	1	0	0	
-		⊟					-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Cálculo do grau de suspeição

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*	*				0	2	0	0	
4	1	*	*				0	2	0	0	
5	2	*	*				0	2	0	0	
7	3	*	*				0	2	0	0	
8	4	*	*				0	2	0	0	
9	5	*	*				0	2	0	0	
11	6	*	*				0	2	0	0	
-		⊟	⊟				-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Cálculo do grau de suspeição

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*	*	*			0	3	0	0	
4	1	*	*	*			0	3	0	0	
5	2	*	*	*			0	3	0	0	
7	3	*	*	*			0	3	0	0	
8	4	*	*	*			0	3	0	0	
9	5	*	*	*			0	3	0	0	
11	6	*	*	*			0	3	0	0	
-		⊔	⊔	⊔			-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Cálculo do grau de suspeição

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*	*	*	*		0	3	0	1	
4	1	*	*	*	*		0	3	0	1	
5	2	*	*	*	*		0	3	0	1	
7	3	*	*	*	*		0	3	0	1	
8	4	*	*	*	*		0	3	0	1	
9	5	*	*	*	*		0	3	0	1	
11	6	*	*	*	*		0	3	0	1	
-		⊔	⊔	⊔	▫		-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Cálculo do grau de suspeição

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*	*	*	*	*	0	3	1	2	0,5
4	1	*	*	*	*	*	0	3	1	2	0,5
5	2	*	*	*	*		0	3	0	1	0,33
7	3	*	*	*	*		0	3	0	1	0,33
8	4	*	*	*	*		0	3	0	1	0,33
9	5	*	*	*	*		0	3	0	1	0,33
11	6	*	*	*	*		0	3	0	1	0,33
-		␣	␣	␣	▫	▫	-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Cálculo do grau de suspeição

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	**1**	*	*	*	*	*	**0**	**3**	**1**	**2**	**0,5**
4	**1**	*	*	*	*	*	**0**	**3**	**1**	**2**	**0,5**
5	2	*	*	*	*		0	3	0	1	0,33
7	3	*	*	*	*		0	3	0	1	0,33
8	4	*	*	*	*		0	3	0	1	0,33
9	5	*	*	*	*		0	3	0	1	0,33
11	6	*	*	*	*		0	3	0	1	0,33
-		␣	␣	␣	▫	▫	-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

Linha	Bloco	Código
1	1	int max(int array [], int length)
2	1	{
3	1	int i = 0;
4	1	int max = array[++i]; **//array[i++];**
5	2	while(i < length){
6	3	{
7	3	if(array[i] > max)
8	4	max = array[i];
9	5	i = i + 1;
10	5	}
11	6	return max;
12	6	}

DEPURAÇÃO BASEADA EM COBERTURA

- Coeficiente de Jaccard $\frac{CEF}{(CEF+CNF+CEP)}$

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*	*	*	*	*	0	3	1	2	0,33
4	1	*	*	*	*	*	0	3	1	2	0,33
5	2	*	*	*	*		0	3	0	1	0,25
7	3	*	*	*	*		0	3	0	1	0,25
8	4	*	*	*	*		0	3	0	1	0,25
9	5	*	*	*	*		0	3	0	1	0,25
11	6	*	*	*	*		0	3	0	1	0,25
-		␣	␣	␣	▫	▫	-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Coeficiente de Jaccard $\frac{CEF}{(CEF+CNF+CEP)}$

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*	*	*	*	*	0	3	1	2	0,33
4	1	*	*	*	*	*	0	3	1	2	0,33
5	2	*	*	*	*		0	3	0	1	0,25
7	3	*	*	*	*		0	3	0	1	0,25
8	4	*	*	*	*		0	3	0	1	0,25
9	5	*	*	*	*		0	3	0	1	0,25
11	6	*	*	*	*		0	3	0	1	0,25
-		⊟	⊟	⊟	▫	▫	-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Coeficiente de Ochiai $\frac{CEF}{\sqrt{(CEF+CNF)*(CEF+CEP)}}$

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*	*	*	*	*	0	3	1	2	0,517
4	1	*	*	*	*	*	0	3	1	2	0,517
5	2	*	*	*	*		0	3	0	1	0,5
7	3	*	*	*	*		0	3	0	1	0,5
8	4	*	*	*	*		0	3	0	1	0,5
9	5	*	*	*	*		0	3	0	1	0,5
11	6	*	*	*	*		0	3	0	1	0,5
-		␣	␣	␣	▫	▫	-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Coeficiente de Ochiai $\frac{CEF}{\sqrt{(CEF+CNF)*(CEF+CEP)}}$

Linha	Bloco	T1	T2	T3	T4	T5	CNP	CEP	CNF	CEF	Grau de Suspeição
3	1	*	*	*	*	*	0	3	1	2	0,517
4	1	*	*	*	*	*	0	3	1	2	0,517
5	2	*	*	*	*		0	3	0	1	0,5
7	3	*	*	*	*		0	3	0	1	0,5
8	4	*	*	*	*		0	3	0	1	0,5
9	5	*	*	*	*		0	3	0	1	0,5
11	6	*	*	*	*		0	3	0	1	0,5
-		⊔	⊔	⊔	▫	▫	-	-	-	-	-

DEPURAÇÃO BASEADA EM COBERTURA

- Comparação
  - Tarantula vs Jaccard vs Ochiai

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A practical evaluation of spectrum-based fault localization ☆

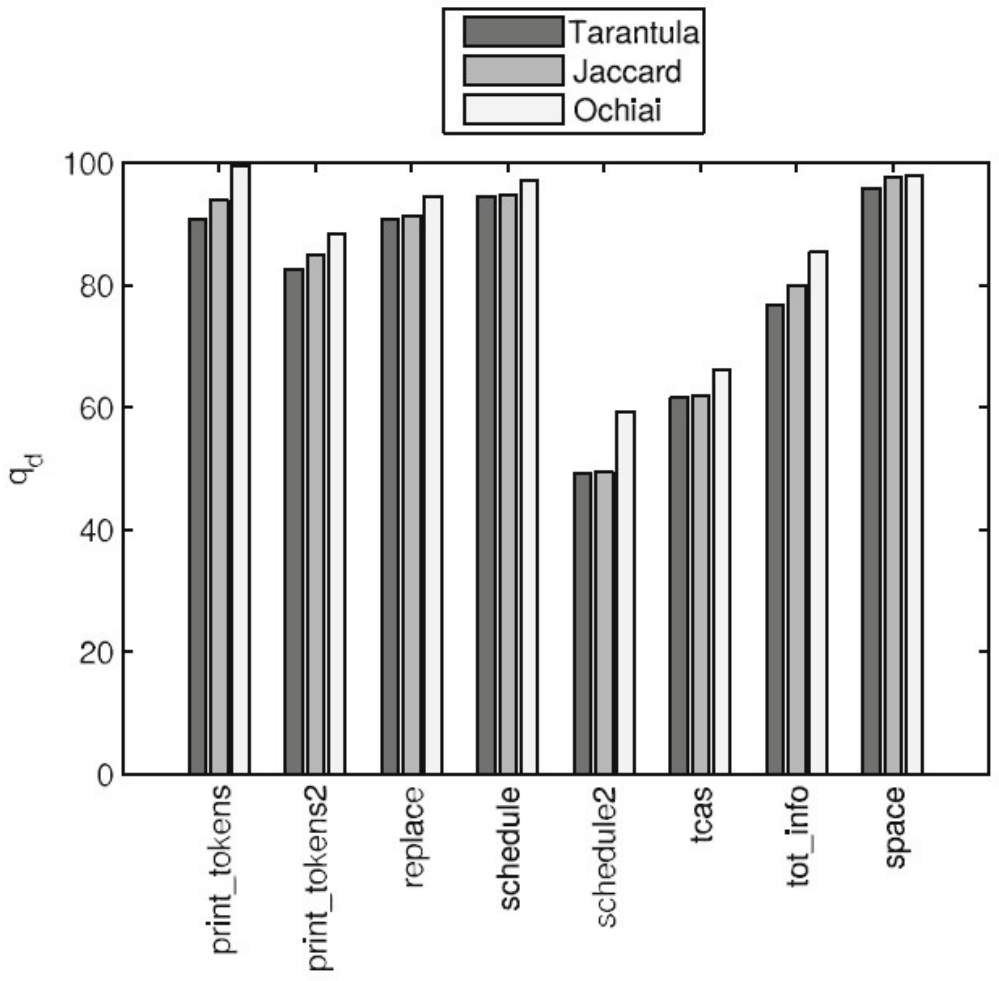
Rui Abreu [a,*], Peter Zoeteweij [a], Rob Golsteijn [b], Arjan J.C. van Gemund [a]

[a] Delft University of Technology, The Netherlands
[b] NXP Semiconductors, The Netherlands

DEPURAÇÃO BASEADA EM COBERTURA

- Comparação
  - Tarantula vs Jaccard vs Ochiai
- Benchmark
  - Siemens suíte (7 programas)
  - Programa Space

DEPURAÇÃO BASEADA EM COBERTURA



DEPURAÇÃO BASEADA EM COBERTURA

- Limitações
  - Teste suítes precisam ser robustas
  - Linha defeituosa mal classificada faz o desenvolvedor desistir
  - Dificuldade com funcionalidades ausentes

REPARO AUTOMÁTICO DE PROGRAMAS

- Gerar versões modificadas do programa original (defeituoso) com o objetivo de corrigir um defeito
- É o inverso do teste de mutação, que tem o objetivo de gerar versões defeituosas do programa original

REPARO AUTOMÁTICO DE PROGRAMAS

- Uso de operadores de reparo
  - Adicionar/Remover/Substituir Código
  - Adicionar pré-condição
  - Substituir condições
  - Inserir checkers
    - Range
    - Class cast
    - Null pointer

REPARO AUTOMÁTICO DE PROGRAMAS

- Exemplos

```
for (int i=num; i < state.parenCount; i++)
        state.parens[i].length = 0;
state.parenCount=num;
```

```
for (int i=num; i < state.parenCount; i++)
        if (state!=null && state.parens[i]!=null)
                state.parens[i].length = 0;
state.parenCount=num;
```

REPARO AUTOMÁTICO DE PROGRAMAS

- Exemplos

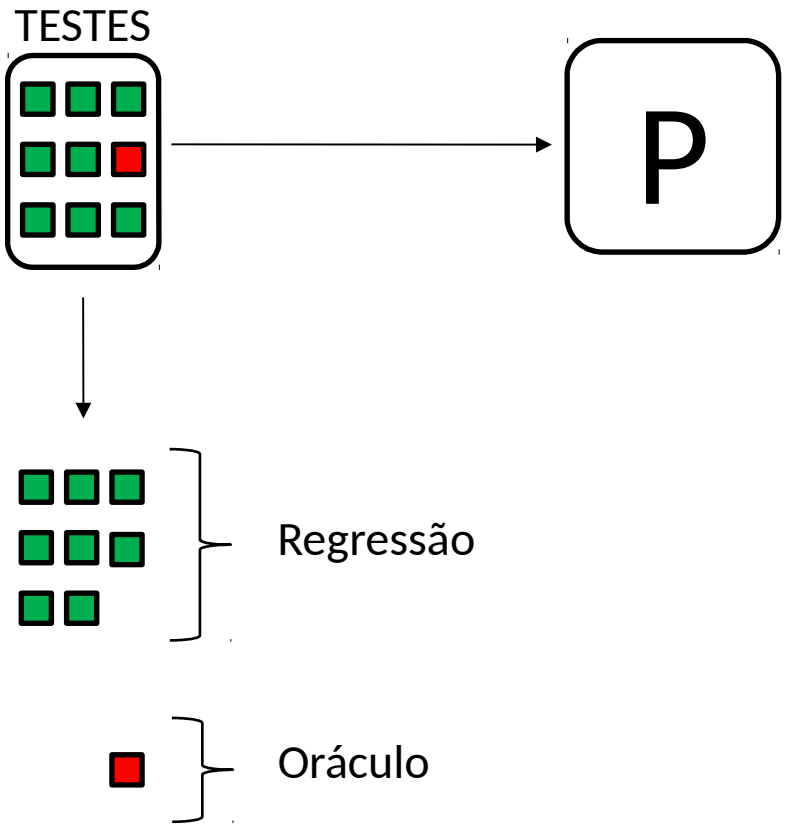
```
Comment[] getLeadingComments(){
   If (this.leadingComments!=null){
      ...
   }
   return trailingComments;
}
```



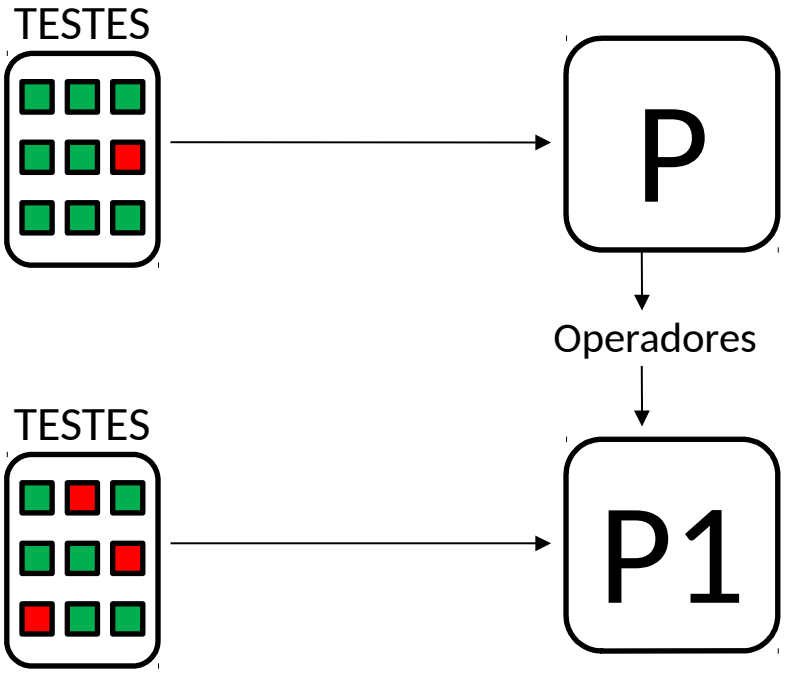
```
Comment[] getLeadingComments(){
   If (this.leadingComments!=null){
      ...
   }
   return leadingComments;
}
```

REPARO AUTOMÁTICO DE PROGRAMAS

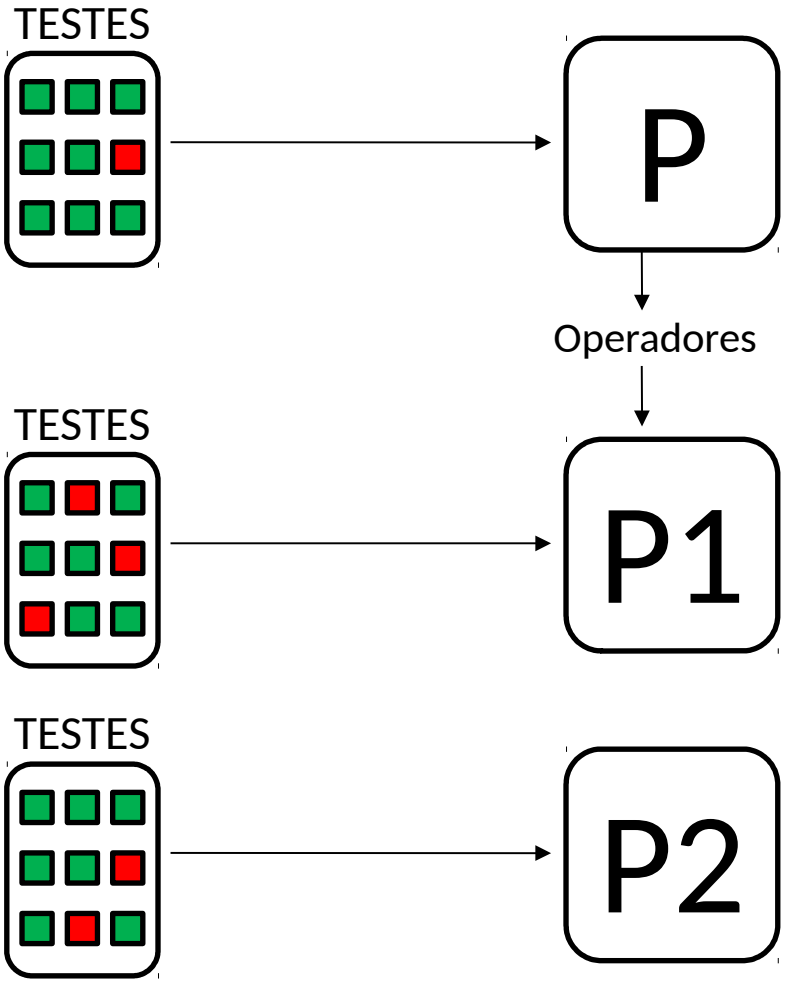
REPARO AUTOMÁTICO DE PROGRAMAS



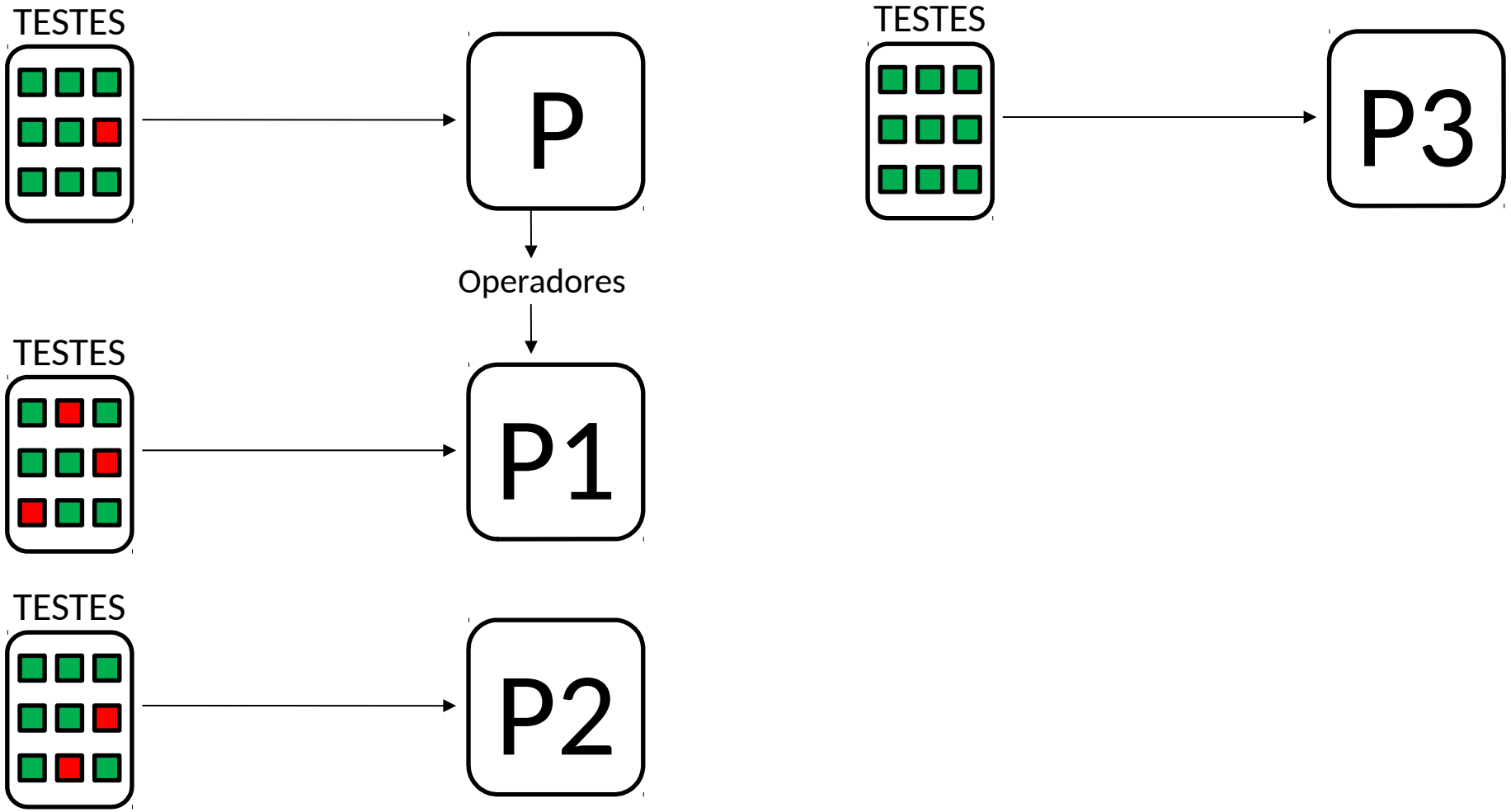
REPARO AUTOMÁTICO DE PROGRAMAS



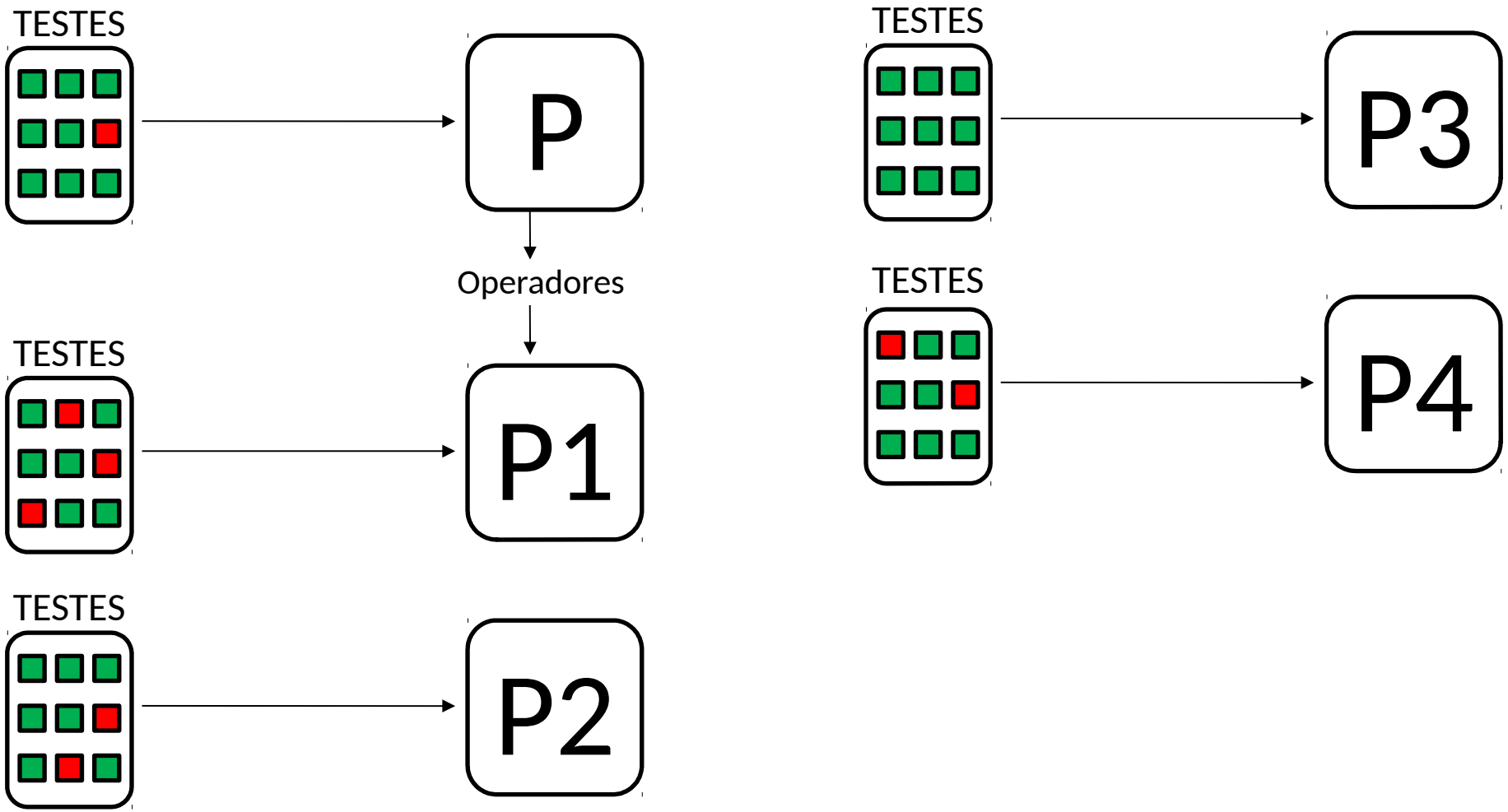
REPARO AUTOMÁTICO DE PROGRAMAS



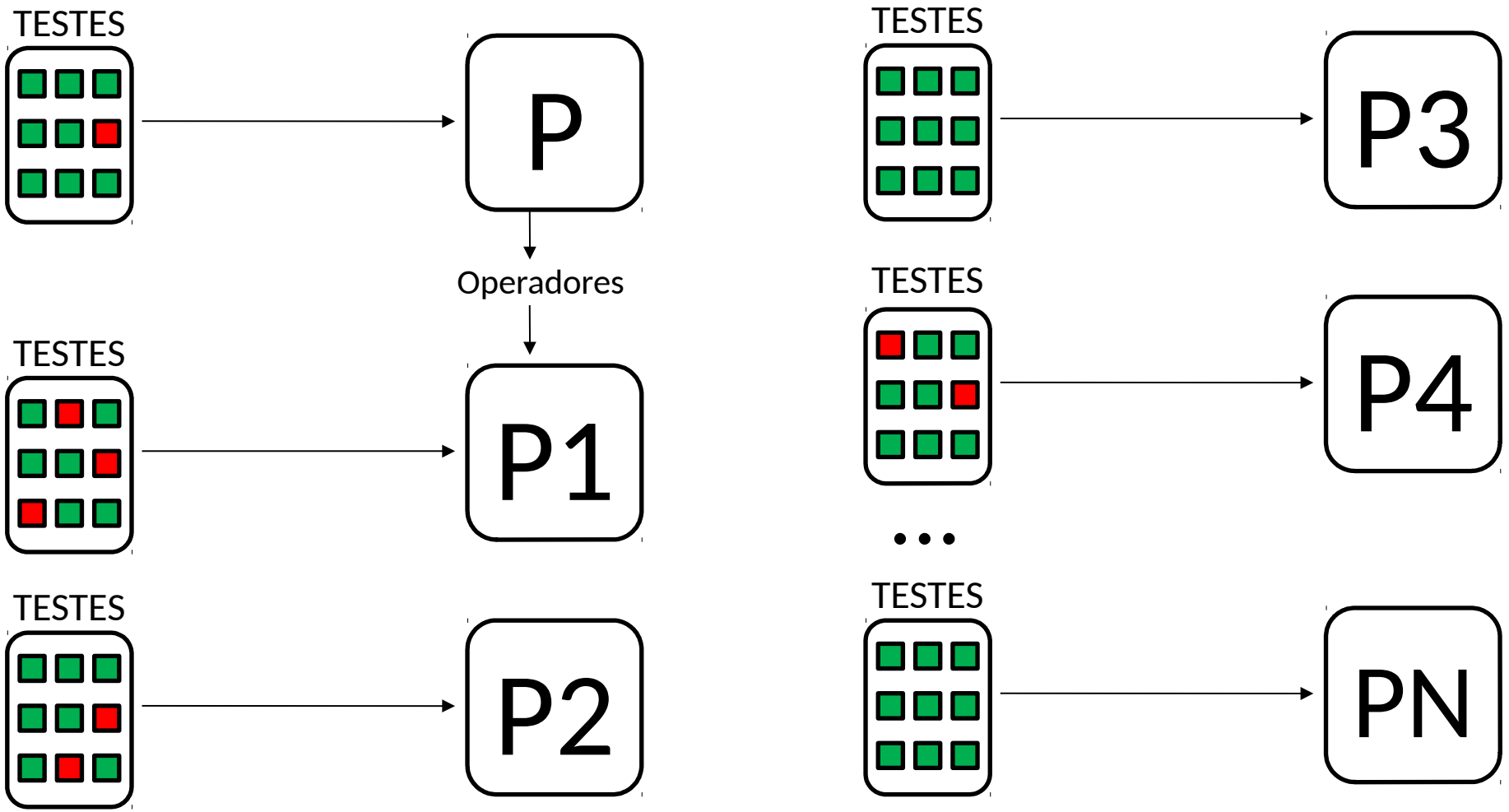
REPARO AUTOMÁTICO DE PROGRAMAS



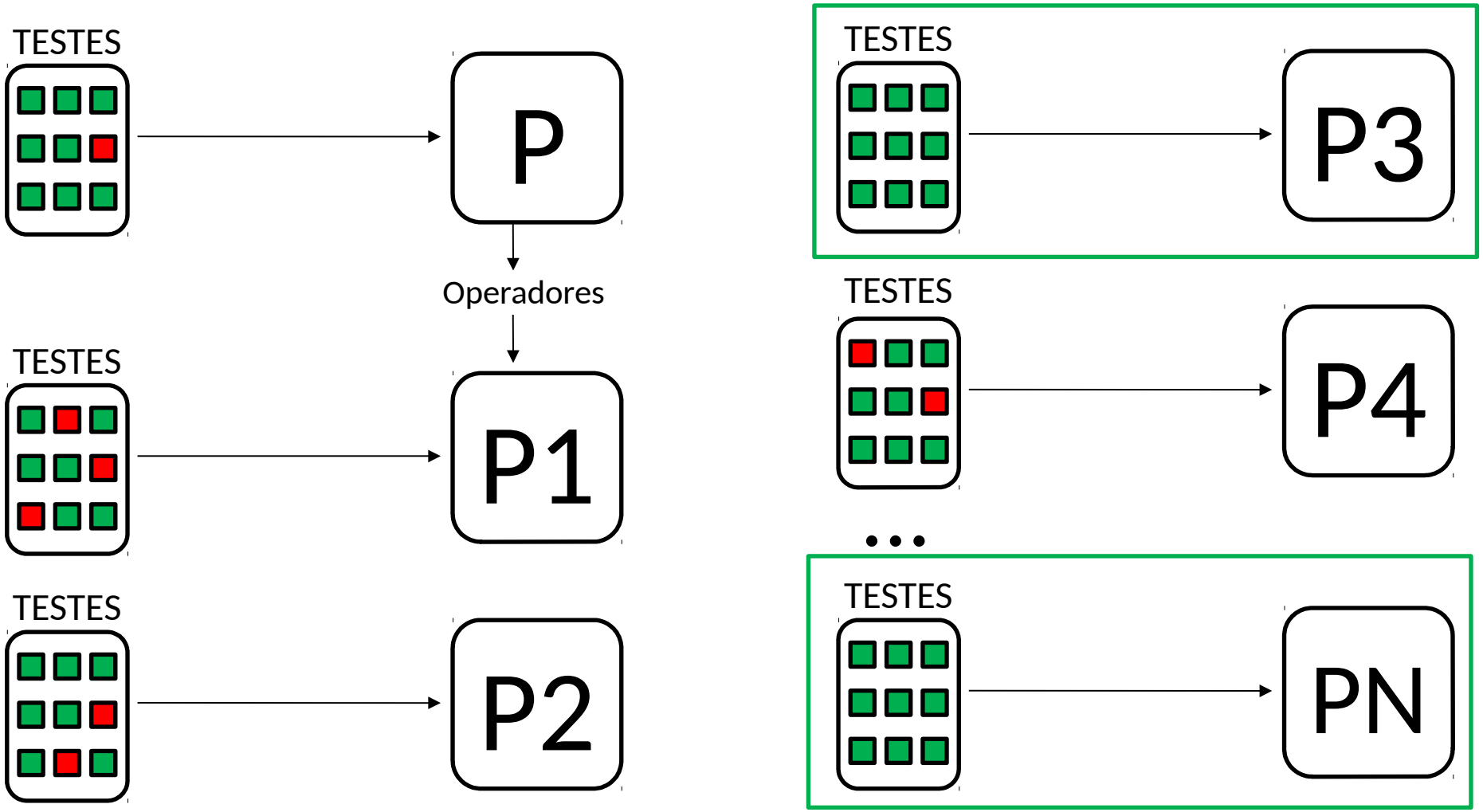
REPARO AUTOMÁTICO DE PROGRAMAS



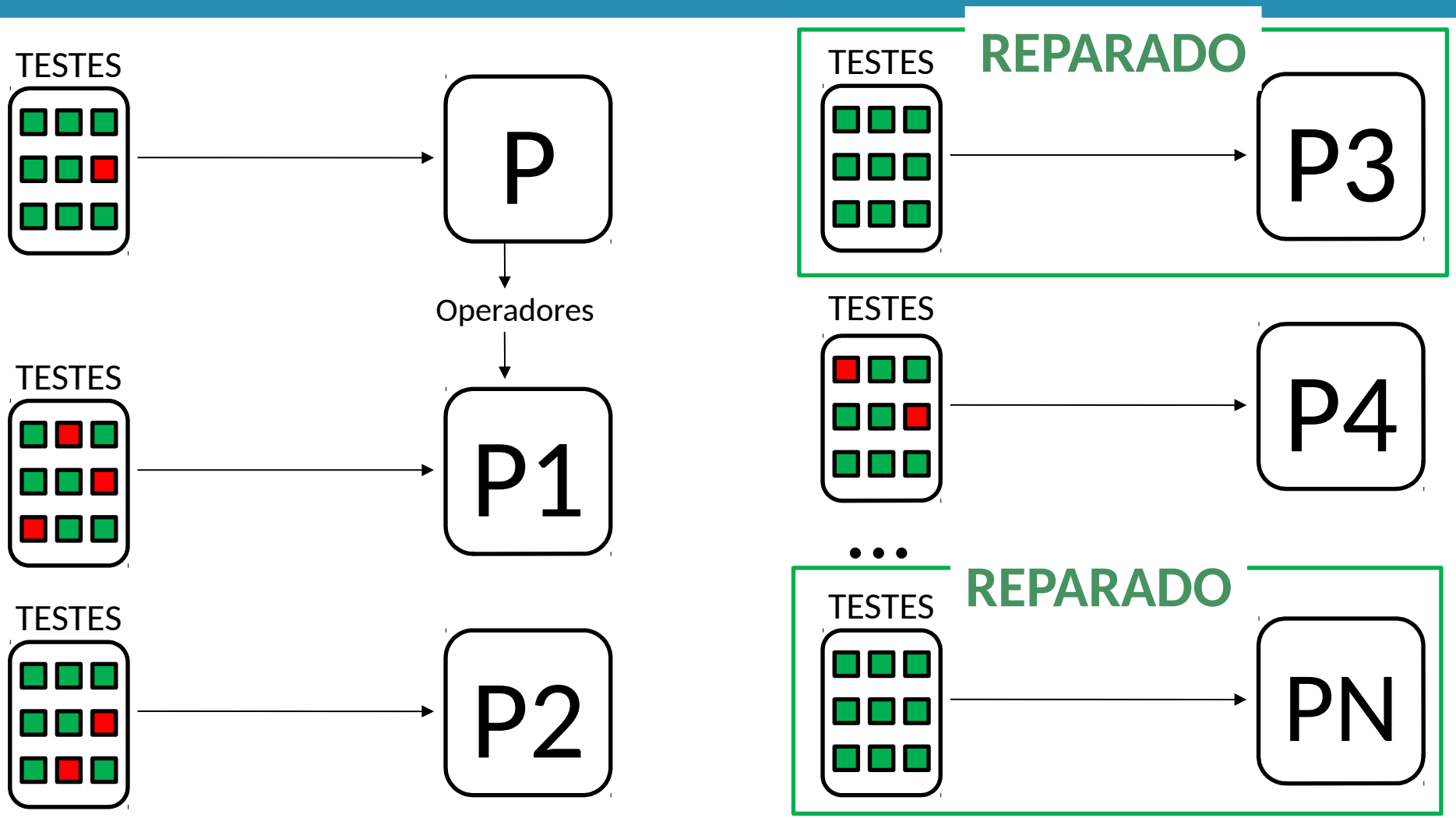
REPARO AUTOMÁTICO DE PROGRAMAS



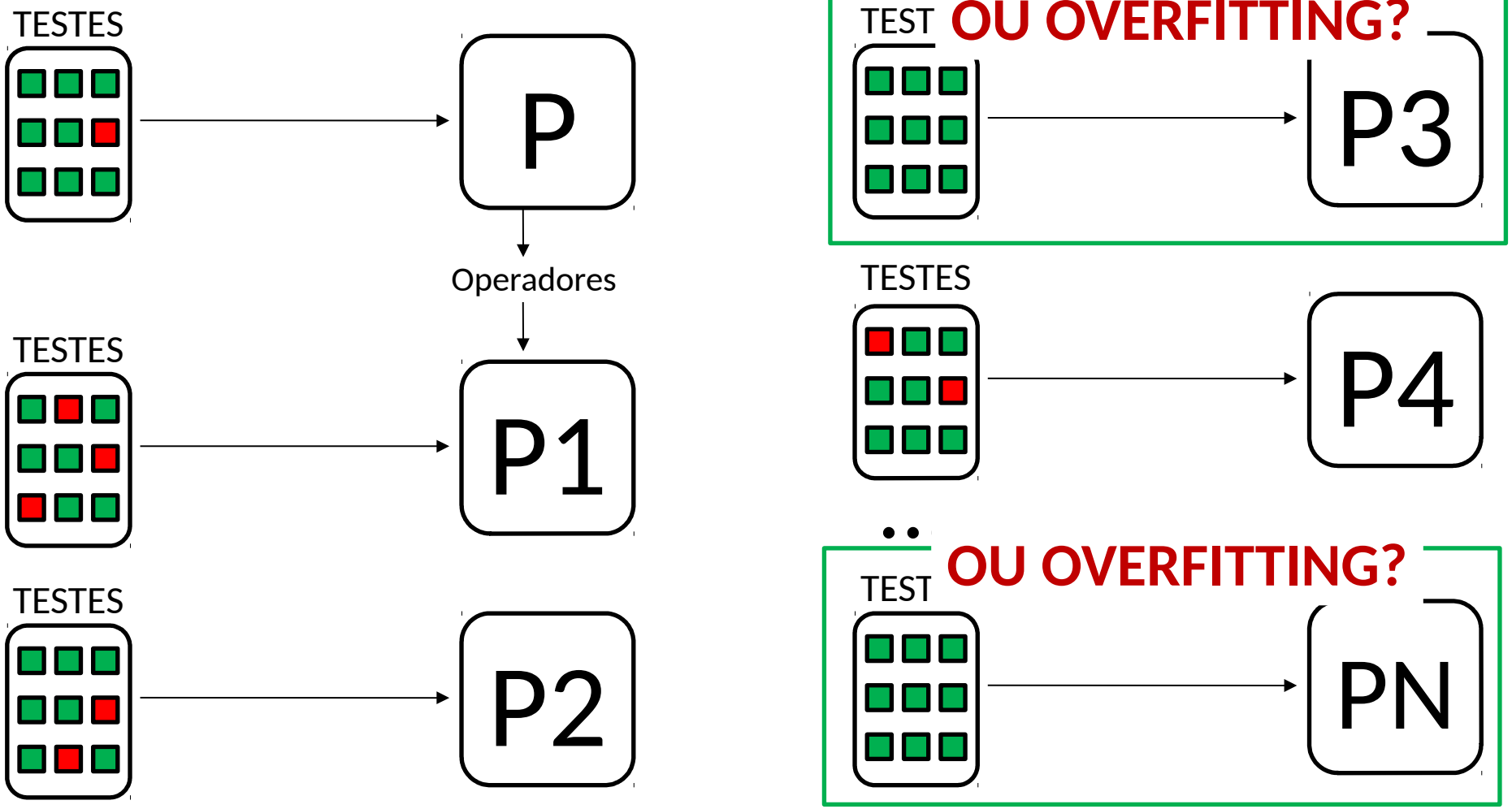
REPARO AUTOMÁTICO DE PROGRAMAS



REPARO AUTOMÁTICO DE PROGRAMAS



REPARO AUTOMÁTICO DE PROGRAMAS



REPARO AUTOMÁTICO DE PROGRAMAS

- Técnicas
  - Programação Genética
  - Síntese de Programas
  - Padrões de Reparo

REPARO AUTOMÁTICO DE PROGRAMAS

- Combinação de técnicas
  - Localização de defeitos
  - Reparo automático
- Avaliação das técnicas
  - Geralmente com programas menores
  - Poucos benchmarks robustos e consolidados

REPARO AUTOMÁTICO DE PROGRAMAS

- Limitações
  - Precisa de suítes de teste robustas
  - Dificuldades com alterações mais complexas
  - Quantidade de versões geradas
  - Overfitting

DESAFIOS

- Problema do engajamento
- Problema do reconhecimento.
- Problema dos múltiplos defeitos.
- Problema do defeito multilinhas.
- Problema da escalabilidade.

DESAFIOS

- Problema do engajamento
  - Desenvolvedor investiga cinco, no máximo 10 elementos suspeitos.
  - Desenvolvedor abandona a técnica se não observa progresso.
  - Modelos não capturam como as técnicas são usadas, nem o que faz o programador usá-las.

DESAFIOS

- Problema do reconhecimento.
  - Entendimento perfeito do defeito (perfect bug understanding)
  - O que é necessário para o programador reconhecer o defeito?

DESAFIOS

- Problema dos múltiplos defeitos.
  - Programas possuem mais de um defeito
  - Uma falha pode ser resultado da combinação de dois ou mais defeitos.

DESAFIOS

- Problema do defeito multilinhas.
  - Defeitos não estão apenas em uma linha
  - Uma falha é causada por um bug espalhado em várias linhas.
  - O que signifca localizar o defeito?

DESAFIOS

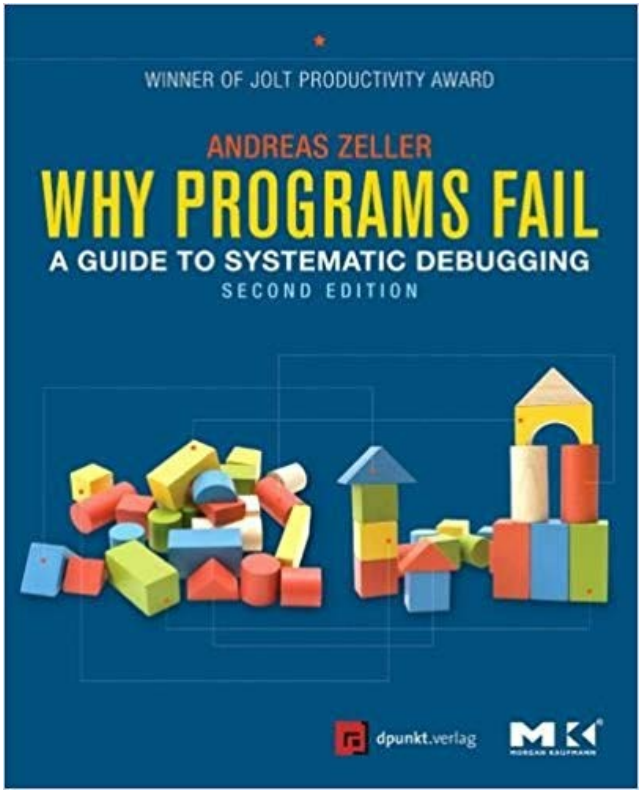
- Problema da escalabilidade.
  - Programas grandes e com execuções longas.
  - Não deveria ser necessário hardware especial.
  - O tempo de resposta não pode “desengajar” o programador.

LEITURA RECOMENDADA

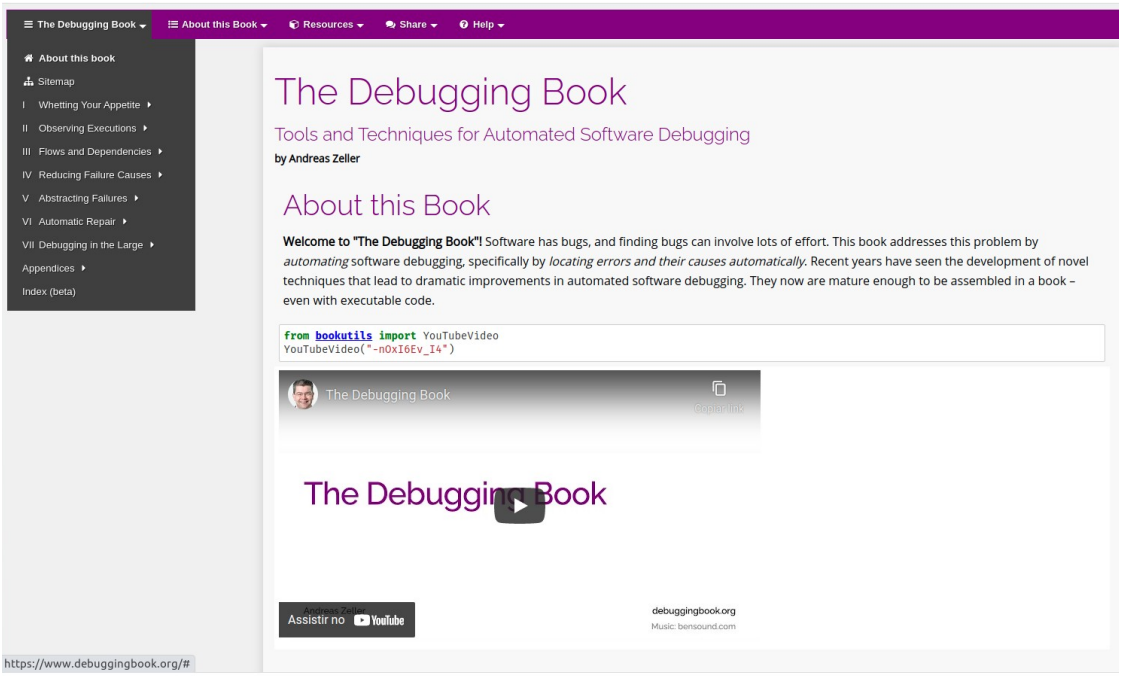
- W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. **A Survey on Software Fault Localization.** IEEE Trans. Softw. Eng. 42, 8 (August 2016), 707-740. DOI: https://doi.org/10.1109/TSE.2016.2521368

- Martin Monperrus. 2018. **Automatic Software Repair: A Bibliography.** ACM Comput. Surv. 51, 1, Article 17 (January 2018). DOI: https://doi.org/10.1145/3105906

Referência recomendada



https://www.debuggingbook.org/



DEPURAÇÃO DE SOFTWARE

Prof. Marcelo Medeiros Eler

marceloeler@usp.br

EACH | campus USP LESTE
Escola de Artes, Ciências e Humanidades
Universidade de São Paulo

USP