UFMG

lab-soft
software engineering laboratory

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Teste para Sistemas de Software com Abundância de Recursos

LabSoft

Fischer Jônatas Ferreira

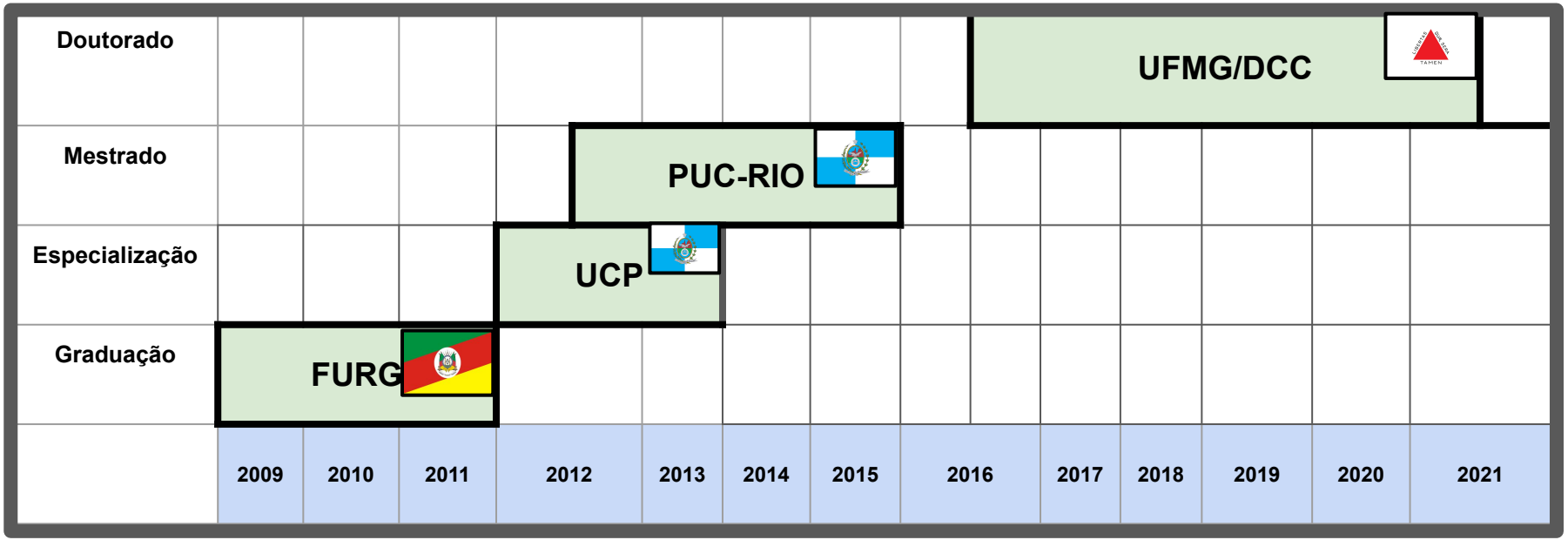
12 de março, 2026

Roteiro do projeto

- Apresentação pessoal
- Contextualização: teste para sistemas configuráveis
- Apresentação de resultados alcançados
- Alguns desdobramentos da pesquisa
- Proposta de projeto de pesquisa

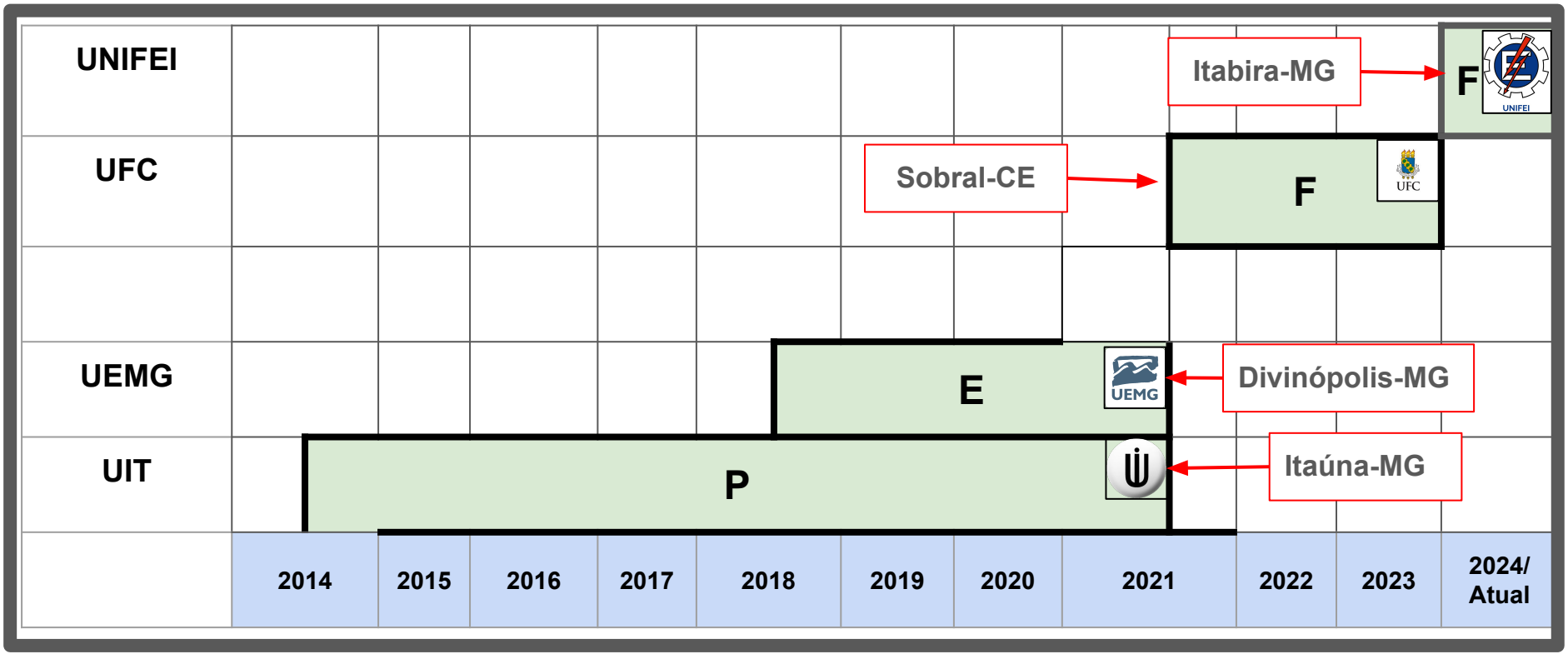
Formação e algumas atividades

Formação: Cursos Realizados por Estados



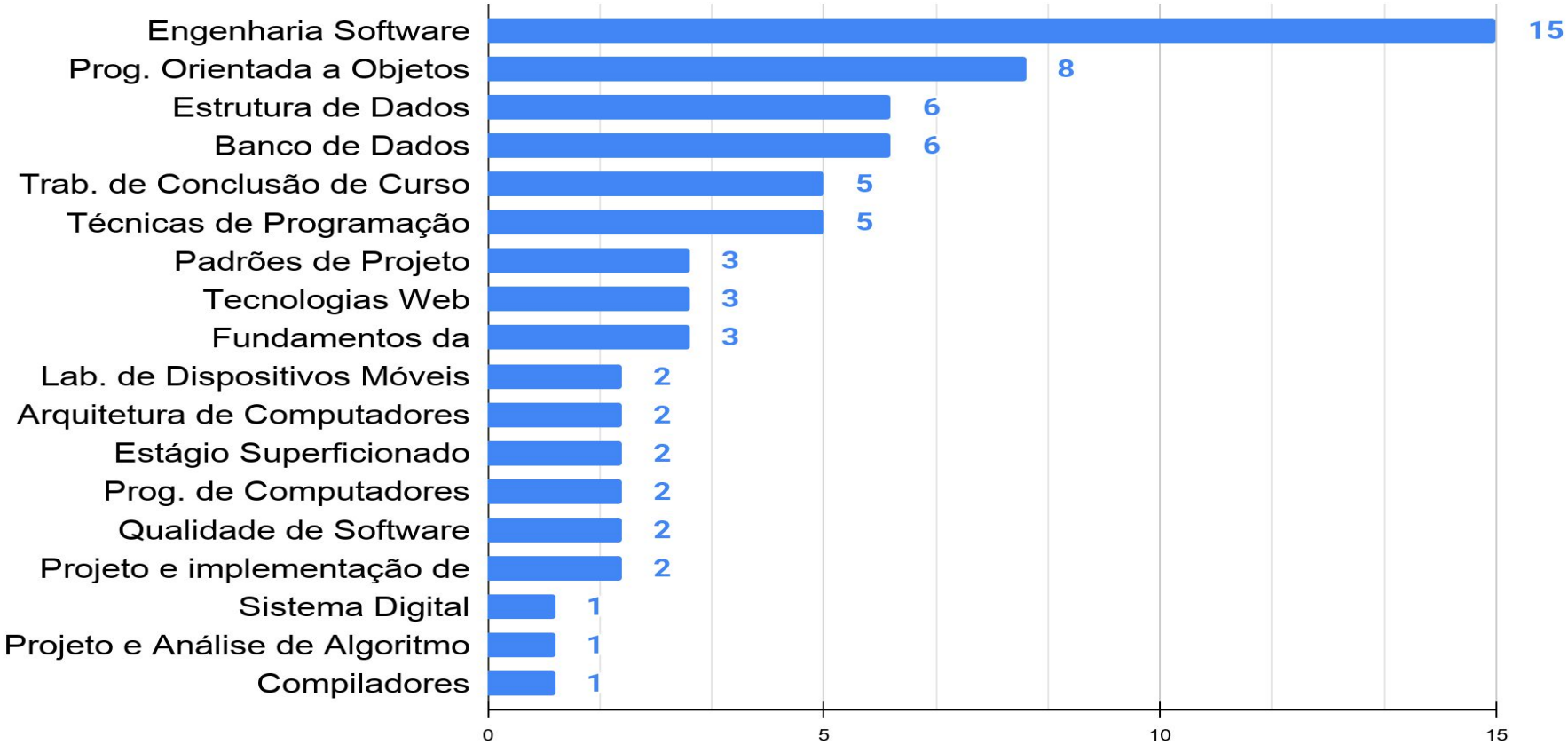
Pós-doc DCC/UFMG: 2025/atualmente

Docência no Ensino Superior



F = Instituição **F**ederal, **E** = Instituição **E**stadual e **P** = Instituição **P**rivada

Docência no Ensino Superior



Projetos atualmente

- FAPEMIG - Demanda universal
- CNPq Universal
- Mistérios Público de Minas Gerais (MPMG) SPEED (DCC/UFMG) e o GSI/MPMG

Pesquisa

PPGEEC - (UFC) - 06 orientações de mestrado em andamento

05 orientações de mestrado concluídas

PPGCC: Coorientação (Pesquisa: Euler)

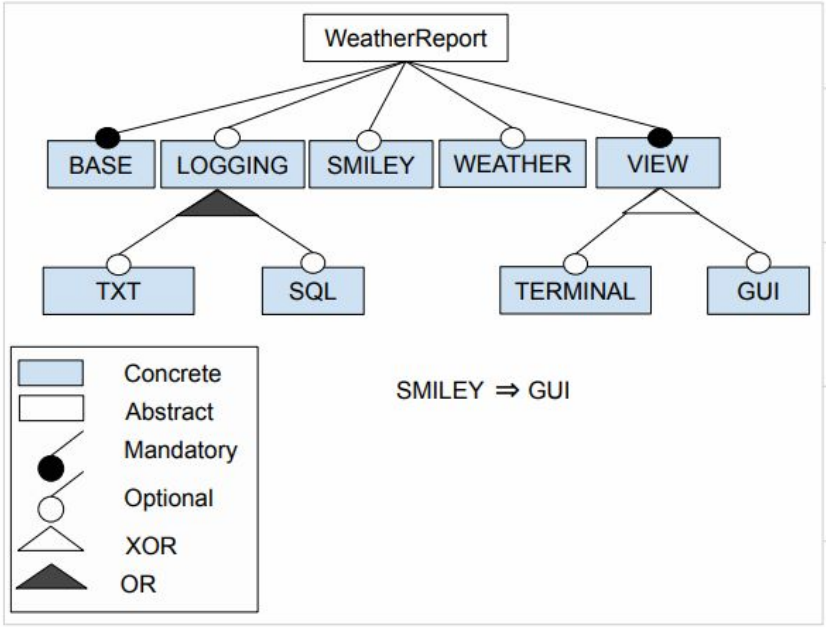
TCP-Member:

- SAST (2021/ atualmente)
- EduComp
- WEI

Editorial Board: SBC Reviews

Pesquisa sobre Teste para Sistemas Configuráveis de Software

Contextualização: Sistemas Configuráveis de Software



Configuration 1

BASE, -LOGGING, -TXT, -SQL -SMILEY, -WEATHER, **VIEW**, **TERMINAL,** -GUI

Configuration 2

BASE, LOGGING, **TXT**, -SQL -SMILEY, **WEATHER**, **VIEW**, **TERMINAL,** -GUI

Configuration 3

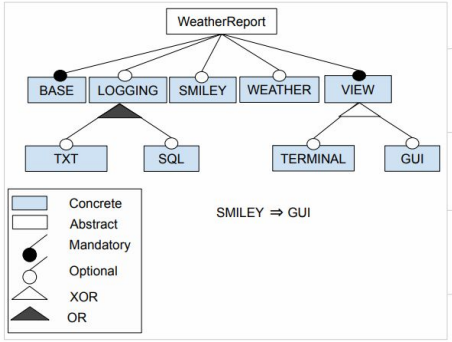
BASE, LOGGING, -TXT, **SQL,** -SMILEY, **WEATHER**, **VIEW,** -TERMINAL, **GUI**

Configuration 4

BASE, LOGGING, **TXT, SQL, SMILEY**, **WEATHER**, **VIEW,** -TERMINAL, **GUI**

Contextualização: Sistemas Configuráveis de Software

Alto nível de Reuso de Software



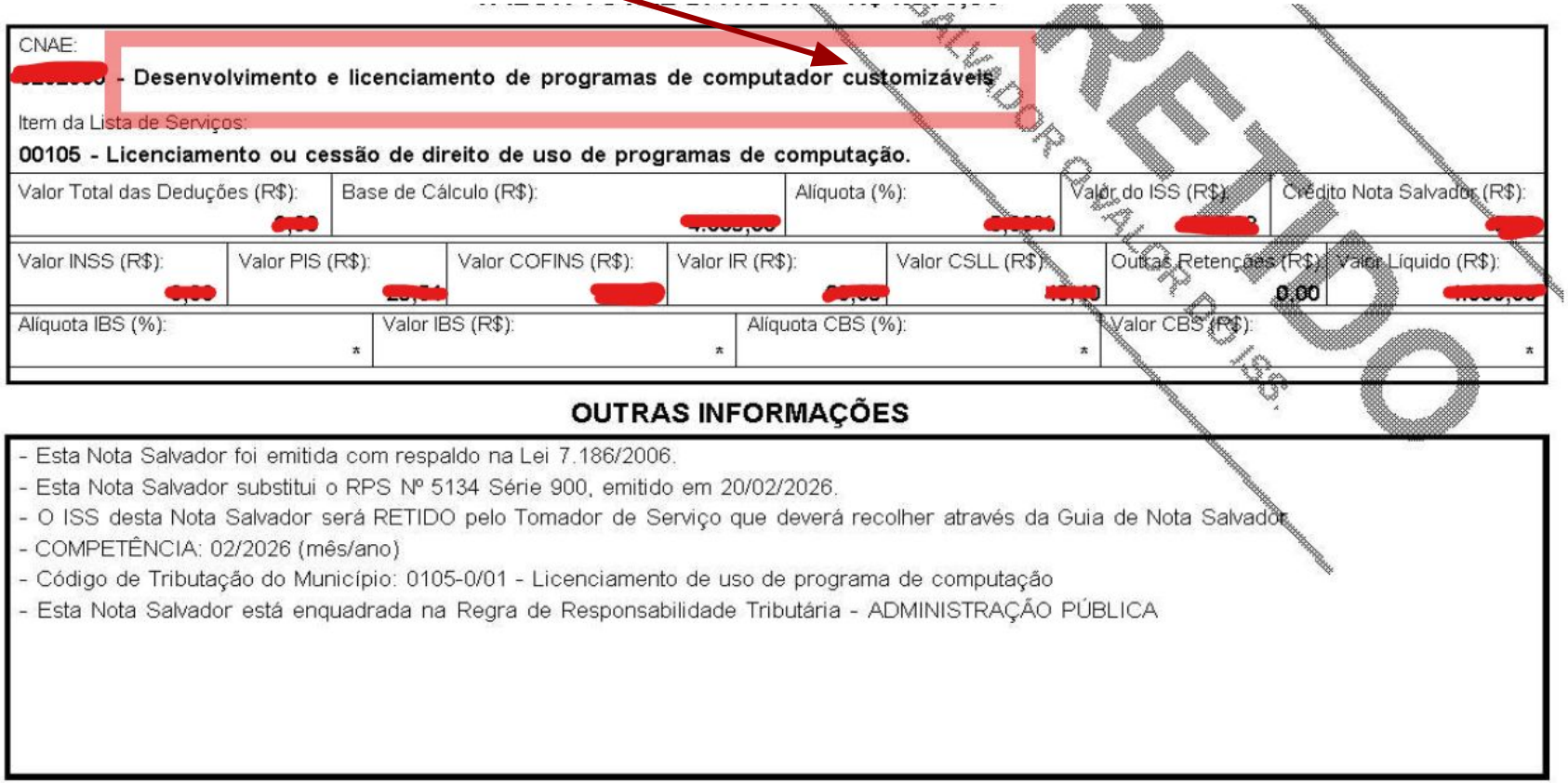
Configuration 1
BASE, -LOGGING, -TXT, -SQL -SMILEY, -WEATHER, **VIEW**, **TERMINAL**, -GUI

Configuration 2
BASE, **LOGGING**, **TXT**, -SQL -SMILEY, **WEATHER**, **VIEW**, **TERMINAL**, -GUI

Configuration 3
BASE, **LOGGING**, -TXT, **SQL**, -SMILEY, **WEATHER**, **VIEW**, -TERMINAL, **GUI**

Configuration 4
BASE, **LOGGING**, **TXT**, **SQL**, **SMILEY**, **WEATHER**, **VIEW**, -TERMINAL, **GUI**

Contextualização: Sistemas Configuráveis de Software



CNAE:
[redacted] - **Desenvolvimento e licenciamento de programas de computador customizáveis**

Item da Lista de Serviços:
00105 - Licenciamento ou cessão de direito de uso de programas de computação.

Valor Total das Deduções (R$):	Base de Cálculo (R$):	Alíquota (%):	Valor do ISS (R$):	Crédito Nota Salvador (R$):
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Valor INSS (R$):	Valor PIS (R$):	Valor COFINS (R$):	Valor IR (R$):	Valor CSLL (R$):	Outras Retenções (R$):	Valor Líquido (R$):
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	0,00	[redacted]

Alíquota IBS (%):	Valor IBS (R$):	Alíquota CBS (%):	Valor CBS (R$):
*	*	*	*

RETIDO

OUTRAS INFORMAÇÕES

- Esta Nota Salvador foi emitida com respaldo na Lei 7.186/2006.
- Esta Nota Salvador substitui o RPS Nº 5134 Série 900, emitido em 20/02/2026.
- O ISS desta Nota Salvador será RETIDO pelo Tomador de Serviço que deverá recolher através da Guia de Nota Salvador
- COMPETÊNCIA: 02/2026 (mês/ano)
- Código de Tributação do Município: 0105-0/01 - Licenciamento de uso de programa de computação
- Esta Nota Salvador está enquadrada na Regra de Responsabilidade Tributária - ADMINISTRAÇÃO PÚBLICA

Contextualização: Implementações

Duas principais abordagens:

- Annotative (compile-time)
  Representada por diretivas #ifdef-like
  (Linux Kernel)

- Variability Encoding (execution-time)
  Representada por variáveis
  (Android Family)

Exemplo de Implementação de Sistema Configurável

Método: Criar texto

entrada: c = [:weather:]

[:weather☺

```
1 public String createText(String c) {
2   if (Configuration.SMILEY)
3     c = c.replace(":]", getSmiley(":]"));
4   if (Configuration.WEATHER)
5     c = c.replace("[:weather:]", temperature);
6   return c;
7 }
```

Exemplo de Implementação de Sistema Configurável

Método: Criar texto

entrada: c = [:weather:]

30.0º C

```
1 public String createText(String c) {
2   if (Configuration.SMILEY)
3     c = c.replace(":]", getSmiley(":]"));
4   if (Configuration.WEATHER)
5     c = c.replace("[:weather:]", temperature);
6   return c;
7 }
```

Problema da Interação de Features com Falhas

Método: Criar texto

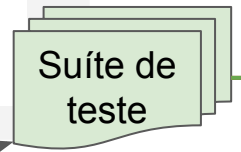
entrada: c = [:weather:]

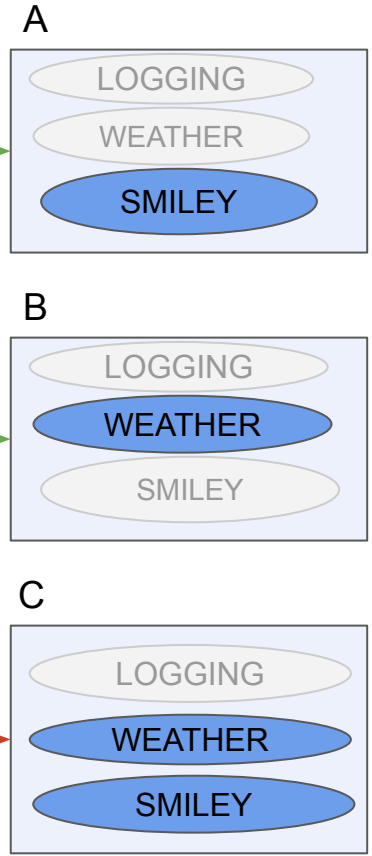
```
1 public String createText(String c) {
2   if (Configuration.SMILEY)
3     c = c.replace(":]", getSmiley(":]"));
4   if (Configuration.WEATHER)
5     c = c.replace("[:weather:]", temperature);
6   return c;
7 }
```

[:weather☺

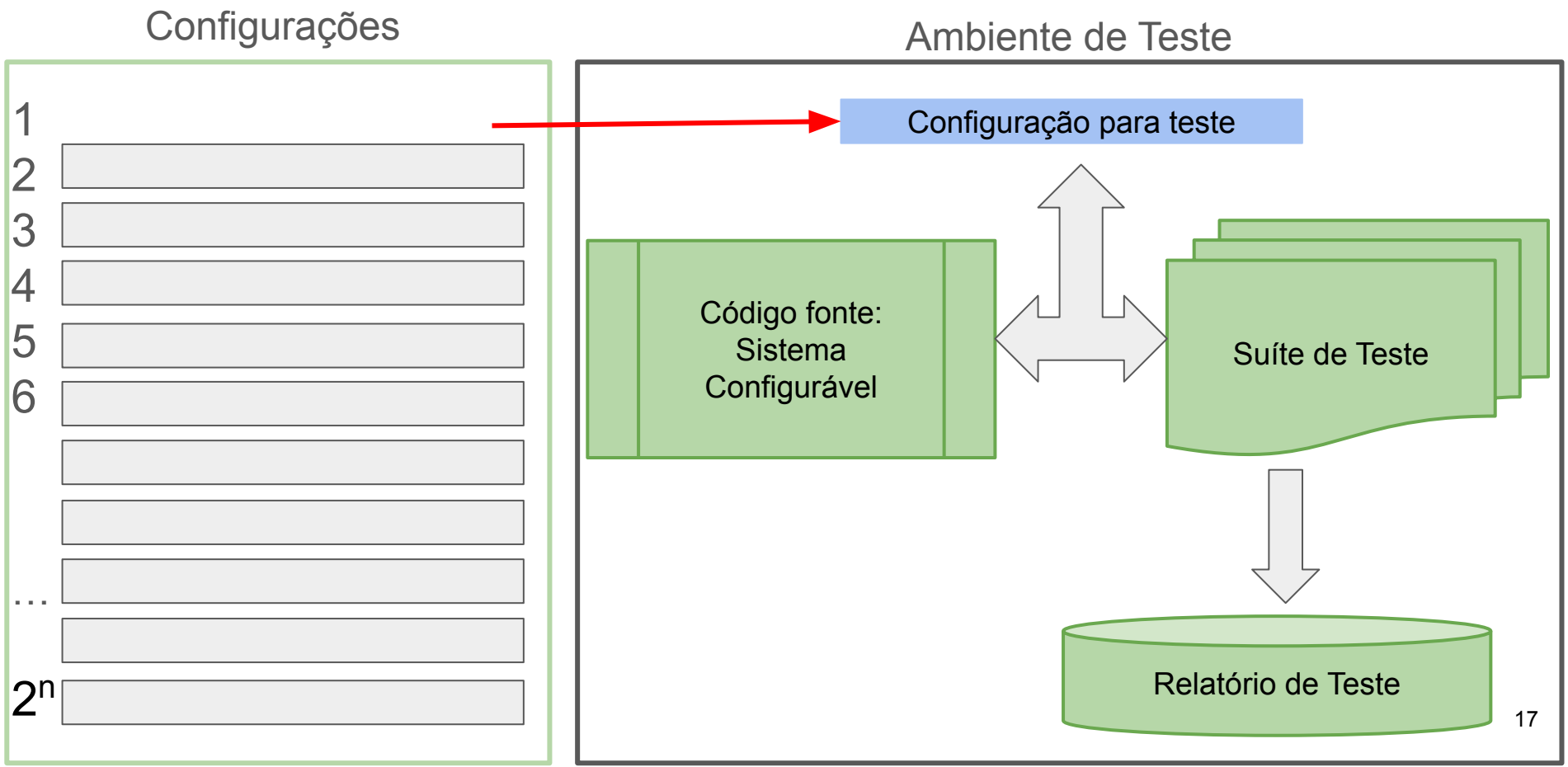
Caso de teste para Wheather

```
1 @Test
2 public void weatherTest () {
3  if (Configuration.WEATHER)
4   assertEquals(wr.createText("[:weather:]"), "30.0°C");
5 }
```

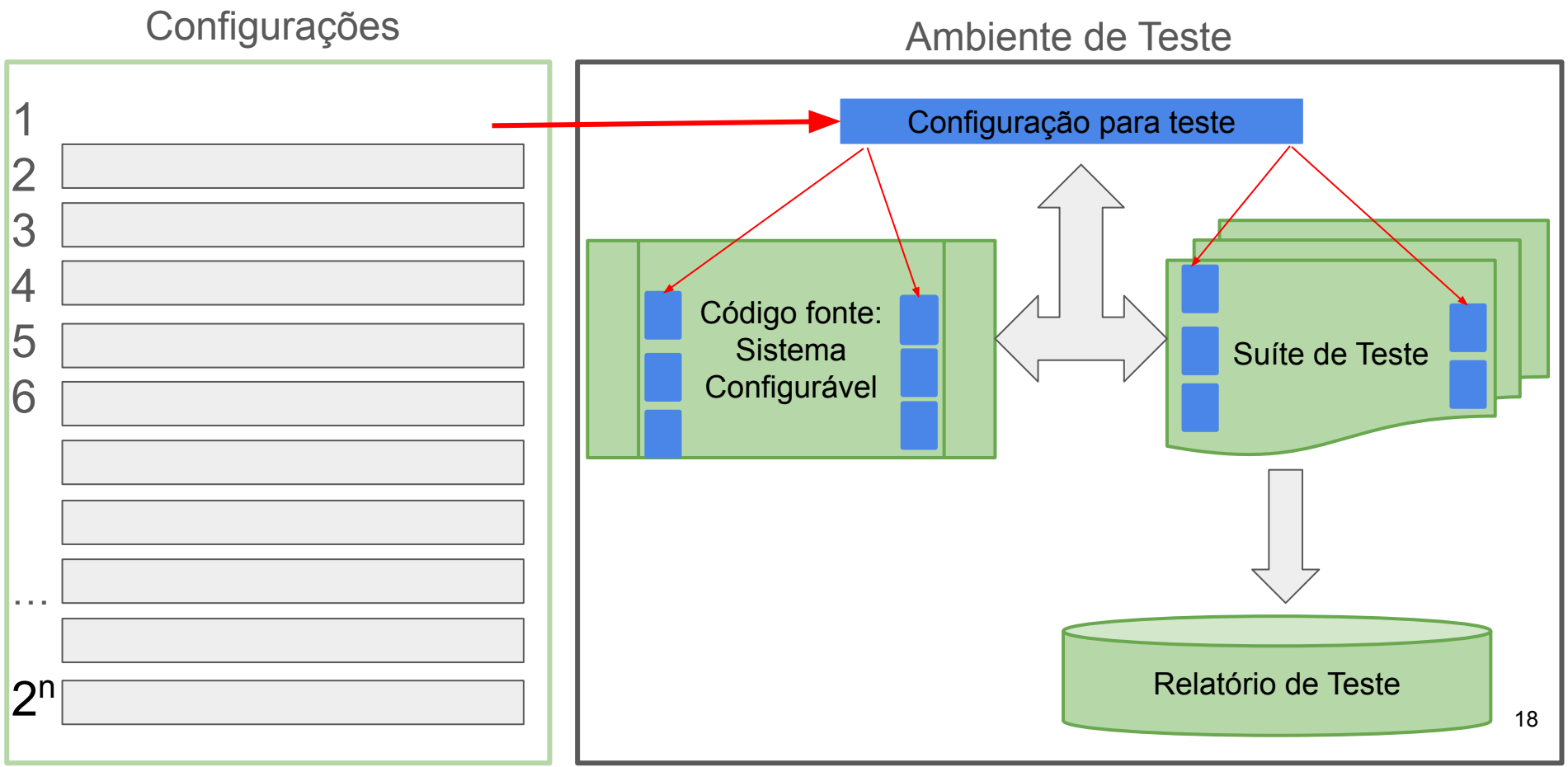




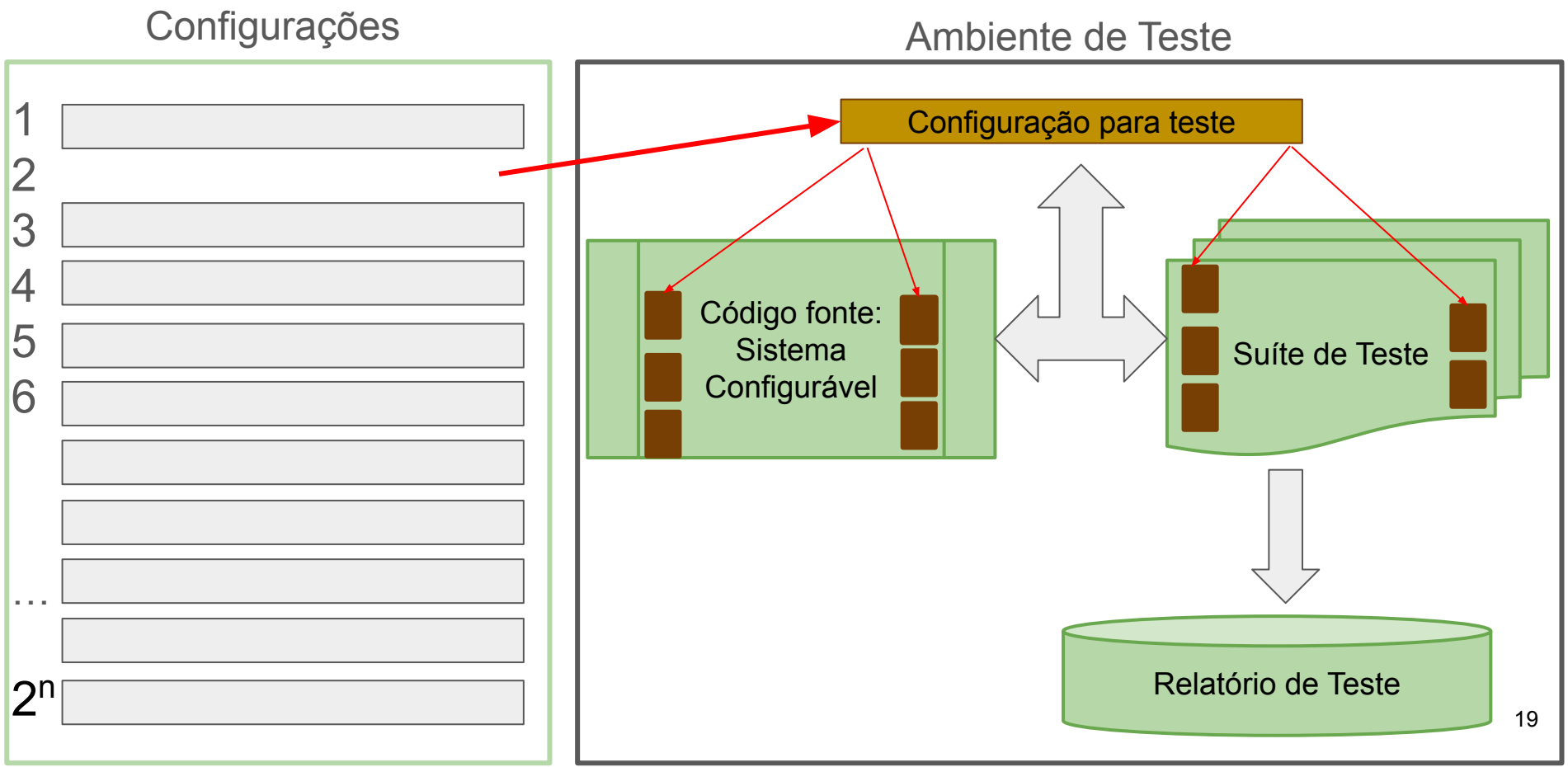
Processo de Teste para Sistemas Configuráveis



Processo de Teste para Sistemas Configuráveis



Processo de Teste para Sistemas Configuráveis



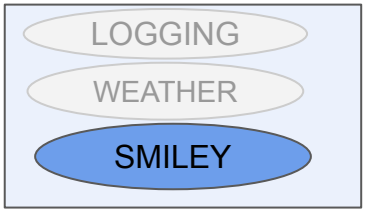
Problema da Interação de Features com Falhas

O número de diferentes configurações a representar e verificar pode chegar a 2^n.

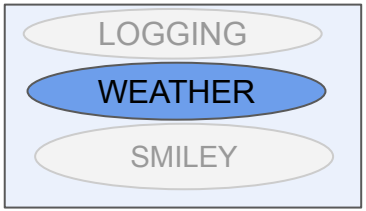
n = número de features

- Lidar com as combinações de features é um problema NP-Completo (MIN-SET-COVER)

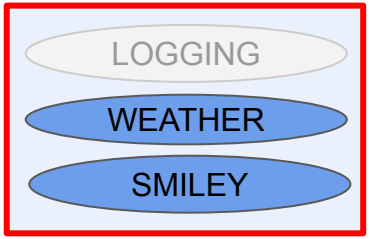
A



B



C



Problema de como Lidar com a Explosão Combinatória de Configurações para Teste

Estratégias Sound

Estratégias Sampling

Estratégias de Teste por Amostragem

A aplicação prática de estratégias de teste para sistemas configuráveis permanece ainda muito custoso

Estratégias	Configurações
One-Disabled	**{!R1, R2, R3}, {R1, !R2, R3}, {R1, R2, !R3}**
One-Enabled	**{R1, !R2, !R3}, {!R1, R2, !R3}, {!R1, !R2, R3}**
Most-Enabled-Disabled	**{R1, R2, R3}, {!R1, !R2, !R3}**
Random	**{!R1, !R2, R3}, {!R1,R2,R3}, {R1, !R2, !R3}, {R1,R2, !R3}, {R1,R2,R3}**
Pairwise	**{R1,R2, !R3}, {R1, !R2,R3}, {!R1,R2,R3}, {!R1, !R2, !R3}**

Lacunas

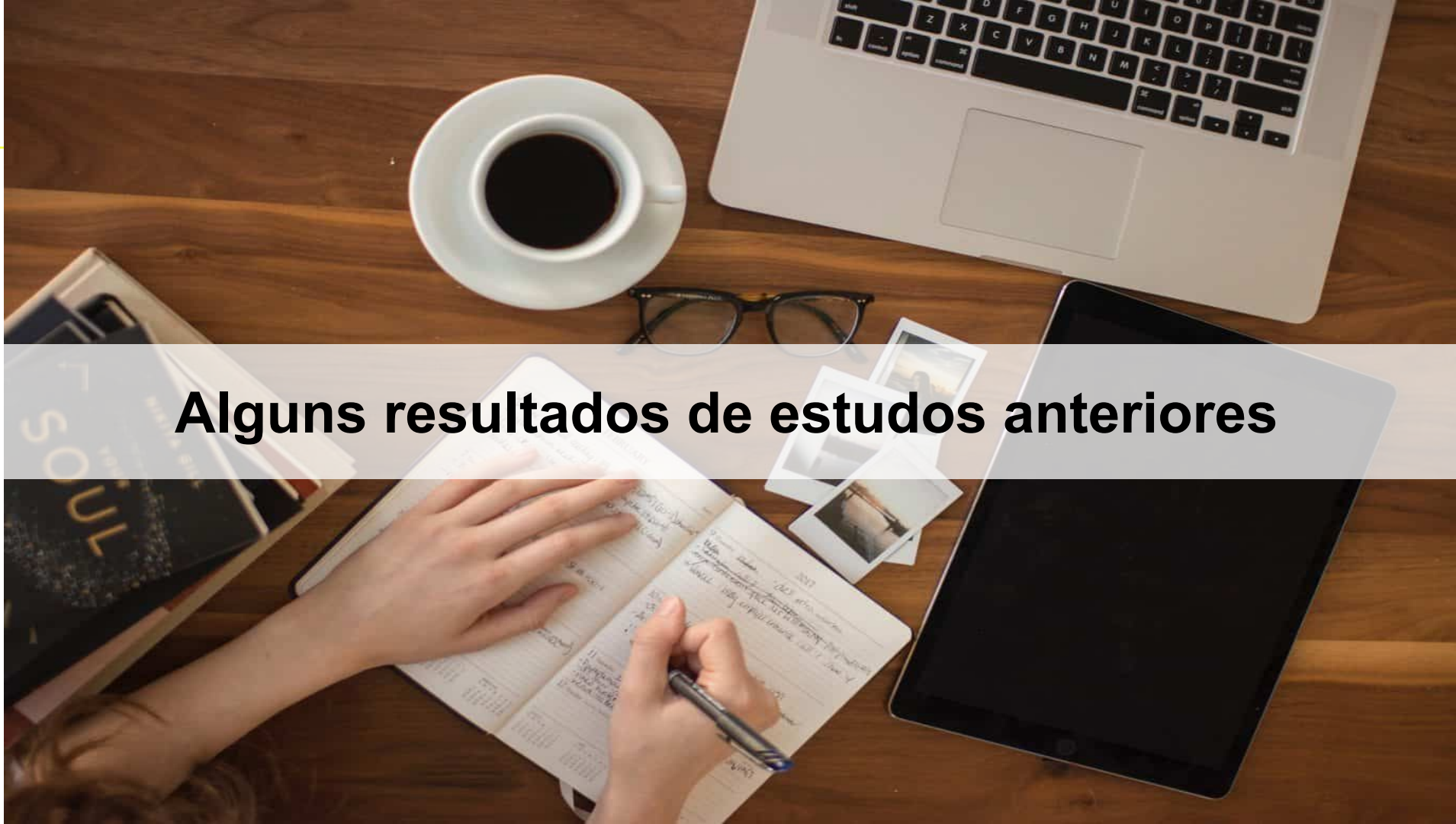
Várias abordagens de testes foram propostas, mas suas potenciais aplicações práticas permanecem amplamente inexploradas

Altos custos para testar Sistemas Configuráveis

Lidar com explosão combinatória de configurações para testes

Problema de interação de features

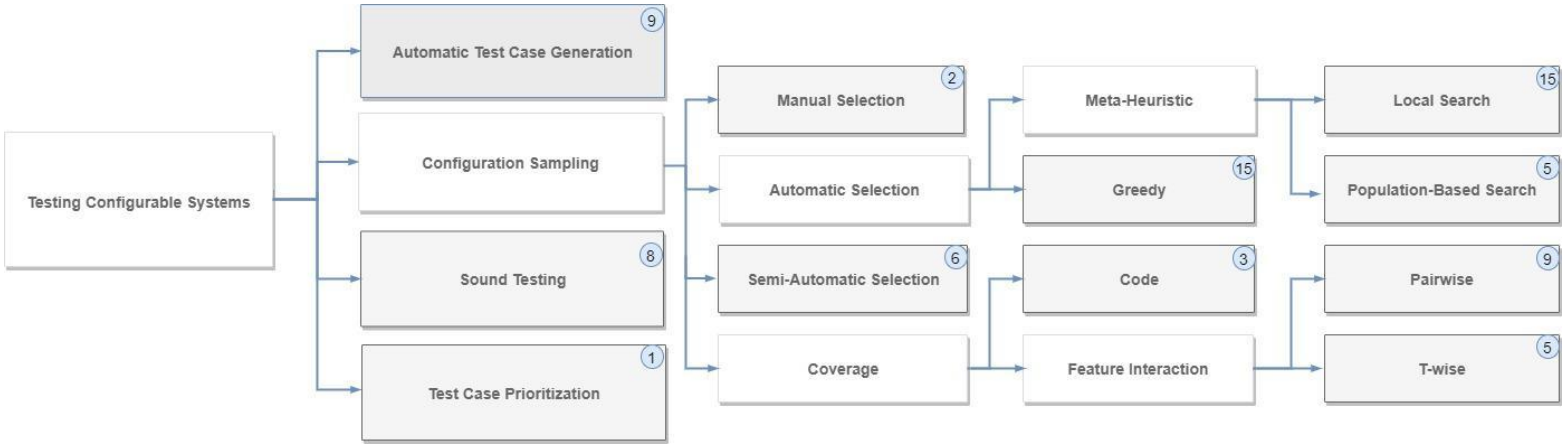
Maior entendimento e classificação das falhas em Sistemas Configuráveis

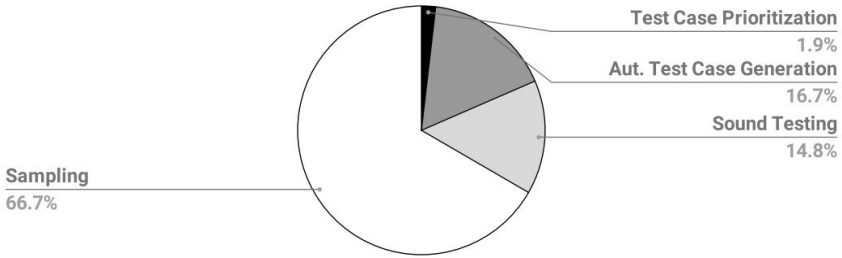


Alguns resultados de estudos anteriores

Estudos Anteriores

As principais estratégias utilizadas para testar ferramentas de sistemas configuráveis

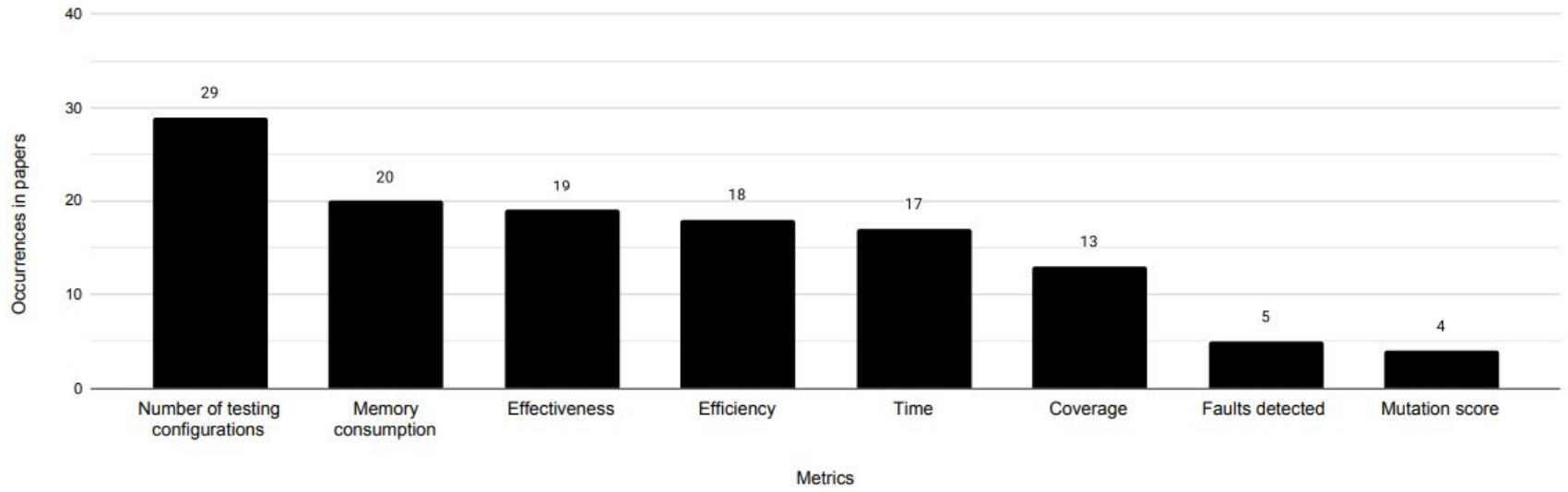




Estudos Anteriores

Visão geral da avaliação de ferramentas de teste para sistemas configuráveis

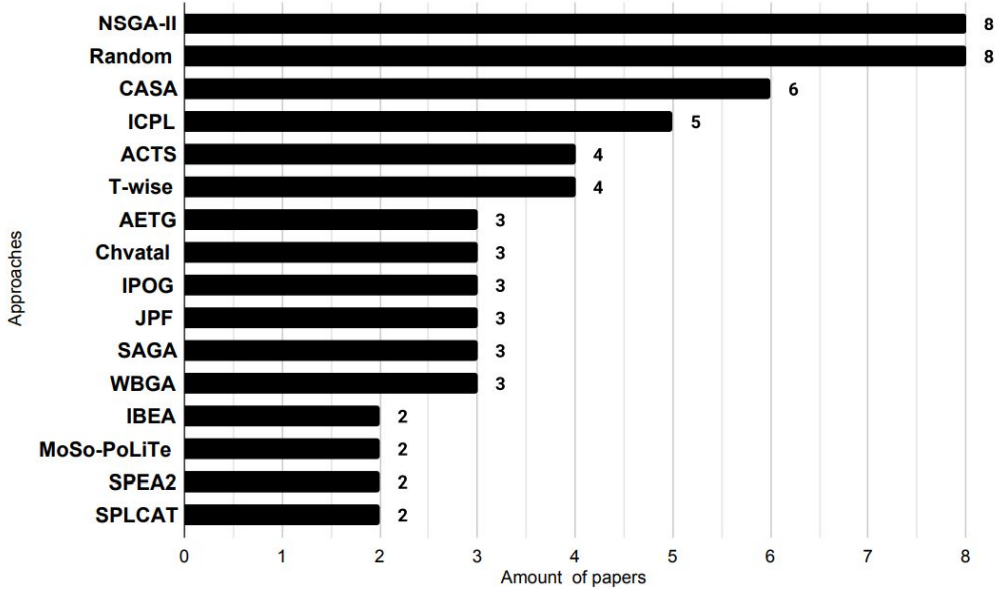
Métricas de avaliação encontradas



Estudos Anteriores

Visão geral da avaliação de ferramentas de teste para sistemas configuráveis

Ferramentas encontradas



Estudos Anteriores

Ferramentas e estratégias de teste selecionadas

- Brute force
- Random
- CASA: versions 1-, 2-, 3-, and 4-wise
- Chvatal: versions 1-, 2-, 3-, and 4-wise
- ICPL: versions 1-, 2-, and 3-wise
- IncLing: 2-wise
- YASA: versions 1-, 2-, 3-, and 4-wise

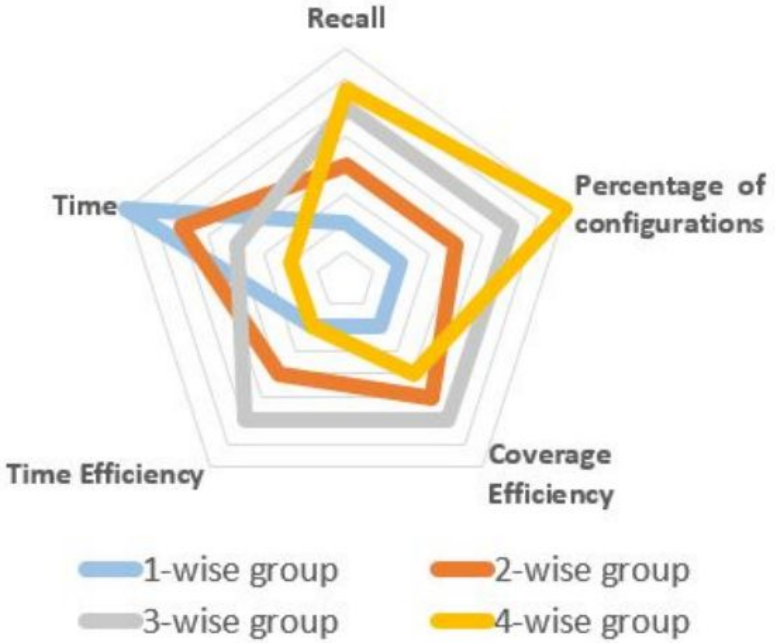
Aquisição dos dados

Estudos Anteriores

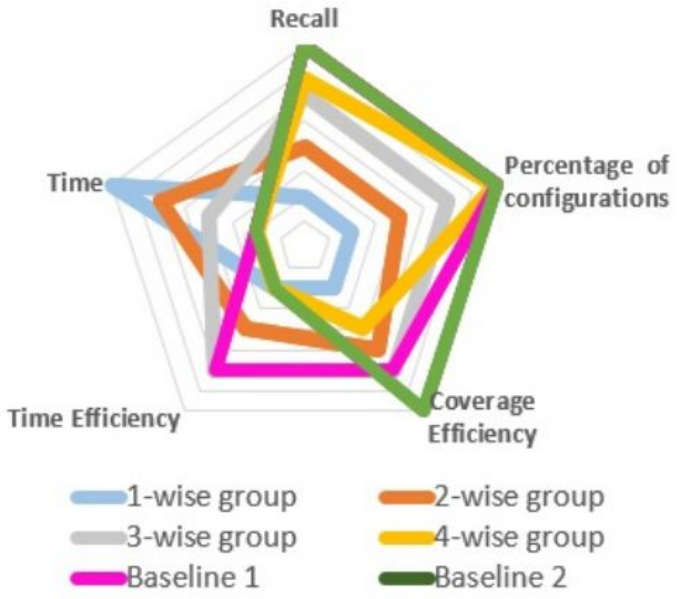
A estratégia de cobertura mais eficiente

Grupo 1: Chvatal-T1 é a estratégia de teste mais eficiente em termos de cobertura para 3 sistemas configuráveis.
Grupo 2: CASA-T2 e Chvatal-T2 são as estratégias de teste mais eficientes em termos de cobertura para 5 sistemas configuráveis.
Grupo 3: Chvatal-T3 é a estratégia de teste mais eficiente em termos de cobertura para 5 sistemas configuráveis.
Grupo 4: Chvatal-T4 é a estratégia de teste mais eficiente em termos de cobertura para 6 sistemas configuráveis.

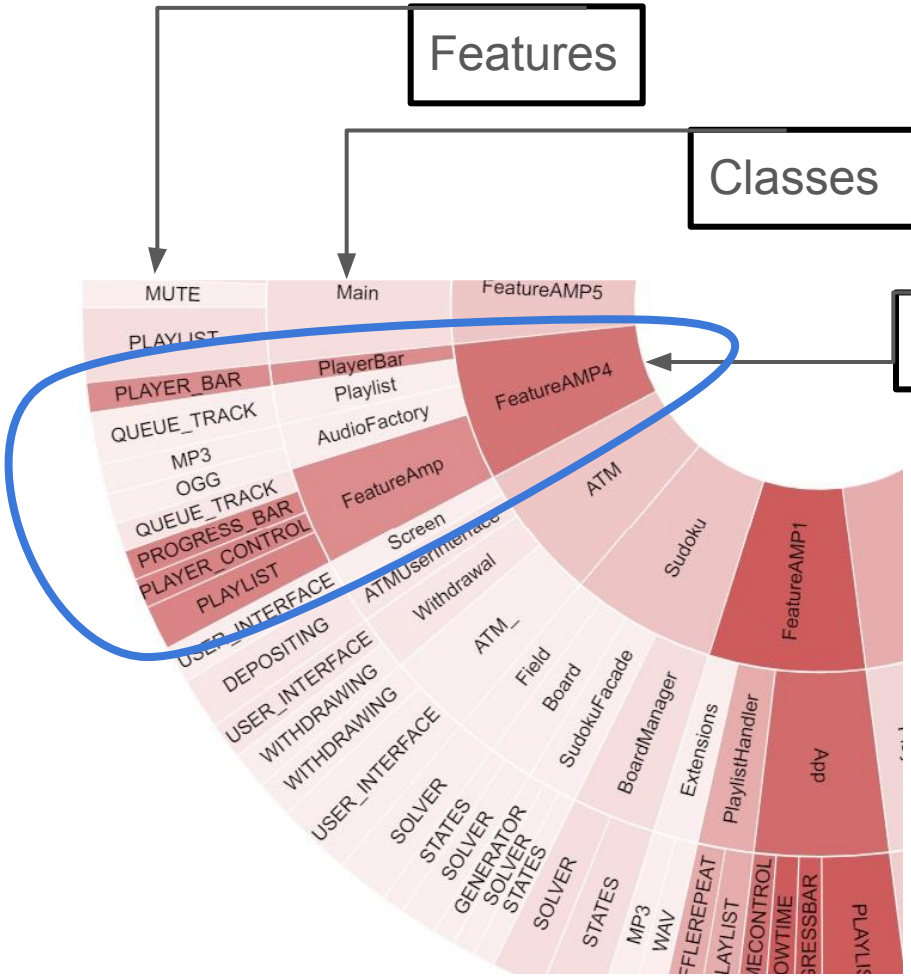
A comparação entre grupos t-wise



T-wise Groups



T-wise and baselines



As falhas se concentram em algumas features e classes dos sistemas configuráveis

Ferreira et. al. Evaluating T-wise testing strategies in a community-wide dataset of configurable software systems 2021. JOURNAL OF SYSTEMS AND SOFTWARE (JSS)

Estudos Anteriores

- Pelo menos uma falha em 16 dos 30 sistemas no conjunto de dados em questão.
- As falhas geralmente se concentram em poucas classes e funcionalidades.
- Classes e features propensas a falhas apresentam características distintas das classes e features livres de falhas.

  Essas diferenças podem ser encontradas com base em métricas de código-fonte comumente utilizadas.

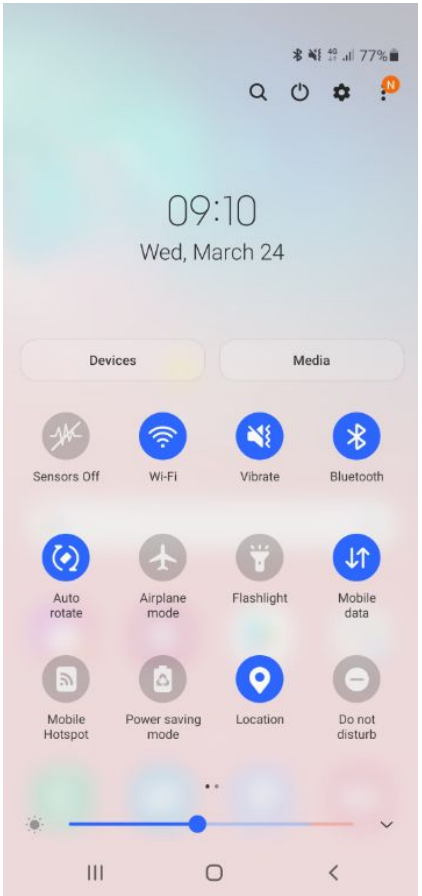
Estudos Anteriores

1. Criação e expansão de conjuntos de testes
2. Criação de casos de teste em classes altamente acopladas
3. Lidar com a explosão combinatória de configurações
4. Amostragem de configurações para teste
5. Execução dos conjuntos de testes
6. Avaliação da qualidade dos conjuntos de testes automatizados
7. Métricas para conjunto de testes
8. Lidar com falsos positivos nos testes
9. Rastreamento de falhas de interação de funcionalidades até suas origens
10. Identificação de dívidas técnicas em casos de teste de sistemas configuráveis

Tipos de sistemas de software com abundância de recursos

Aplicações para dispositivos móveis

- Capacidades de Comunicação (Wi-Fi, Bluetooth, GPS)
- Sensores (Acelerômetro, Giroscópio, Magnetômetro)
- Opções Controladas pelo Usuário (Economia de Bateria, Não Perturbe, Rotação Automática)



Tipos de sistemas de software com abundância de recursos

$$Effectiveness = \frac{FailingSettings}{TotalSettings}$$

Effectiveness of testing strategies

	Random		One-Disabled		One-Enabled		Most-Enab-Dis		Pairwise	
Application	FS	E	FS	E	FS	E	FS	E	FS	E
CovidNow	17	0.57	1	0.07	13	0.93	1	0.50	-	-
Lockwise	30	1.00	14	1.00	14	1.00	2	1.00	8	1.00
Mixin-Messenger	5	0.17	-	-	12	0.86	1	0.50	2	0.25
Nl-covid19	26	0.87	8	0.57	14	1.00	1	0.50	6	0.75
OwnTracks	14	0.47	14	1.00	14	1.00	2	1.00	8	1.00
PocketHub	3	0.10	1	0.07	-	-	-	-	-	-
SpaceXFollower	30	1.00	14	1.00	14	1.00	2	1.00	8	1.00
Threema	14	0.47	13	0.93	1	0.07	1	0.50	4	0.50
Vocable	5	0.17	1	0.07	13	0.93	1	0.50	4	0.50
WordPress-Android	18	0.60	2	0.14	12	0.86	1	0.50	4	0.50

Tipos de sistemas de software com abundância de recursos

Sistemas de Internet das Coisas (IoT)

- Dispositivos Inteligentes (Smartphones, Termostatos), Protocolos de Comunicação (MQTT, CoAP), Gateways

Tipos de sistemas de software com abundância de recursos

Identificação de Conflitos em Merges

- Diferenças entre versões do código, Ferramentas de Controle de Versão (Git, SVN), Algoritmos de Detecção de Conflitos

Tipos de sistemas de software com abundância de recursos

Sistemas Baseados em Nuvem

- Máquinas Virtuais, Armazenamento Escalável, Redes Virtuais, Ferramentas de Gerenciamento de Contêineres

Proposta de pesquisa

Objetivo principal

- Propor diretrizes para apoiar testadores de sistemas com abundância de recursos.

Plano principal

Combinar o conhecimento existente sobre teste de sistemas de software tradicionais com conhecimento específico sobre interação de recursos.

Alguns Objetivos Específicos

- Observar a ocorrência de interações de recursos nos diferentes tipos de sistema de software, além do domínio de sistemas configuráveis, aplicações móveis e IoT.
- Verificar se as estratégias de teste para sistemas configuráveis e aplicações móveis são eficazes em outros domínios, tais como: IoT

Alguns Objetivos Específicos

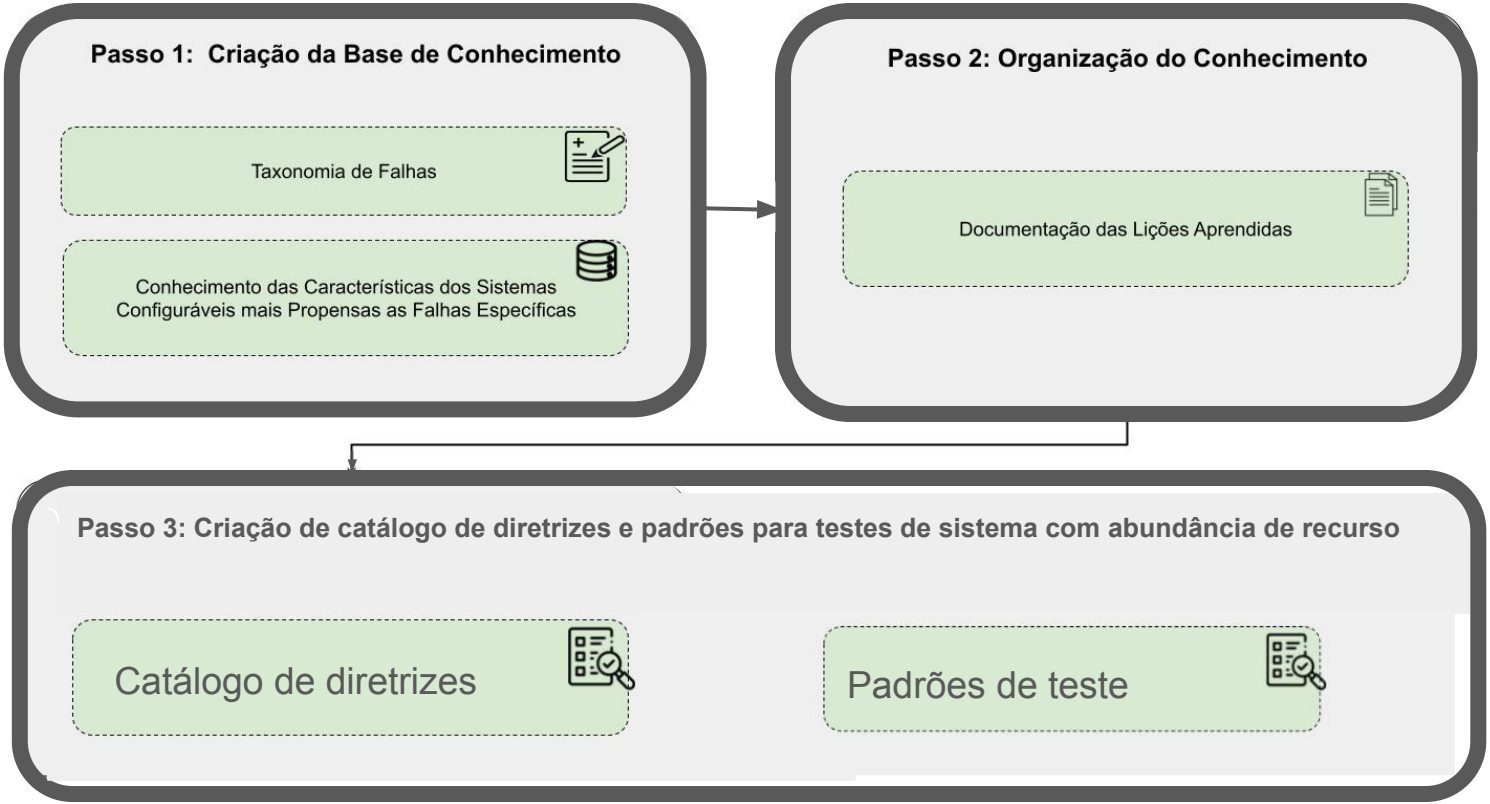
- Criar um dataset de sistemas de software com abundância de recursos, de diferentes domínios, a partir de repositórios públicos de software.

- Criar um catálogo de diretrizes e boas práticas (padrões) para o teste de sistemas de software com abundância de recursos com base em evidências empíricas obtidas nos estudos.

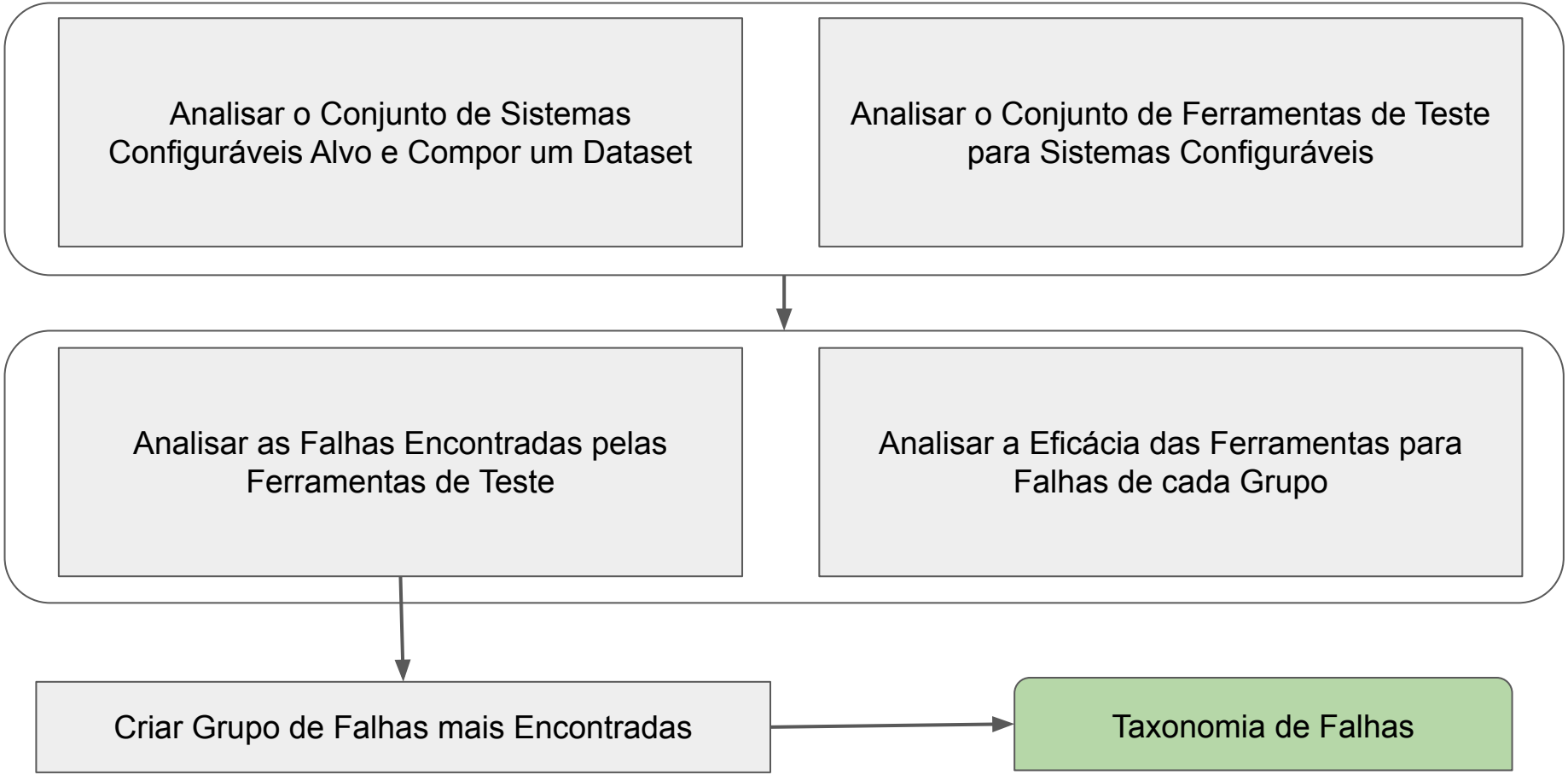
Resultado e Contribuição Principal

- Sumarização das lições aprendidas com os estudos realizados
- Um catálogo de diretrizes e padrões para o desenvolvimento e teste de sistemas de software com abundância de recursos.

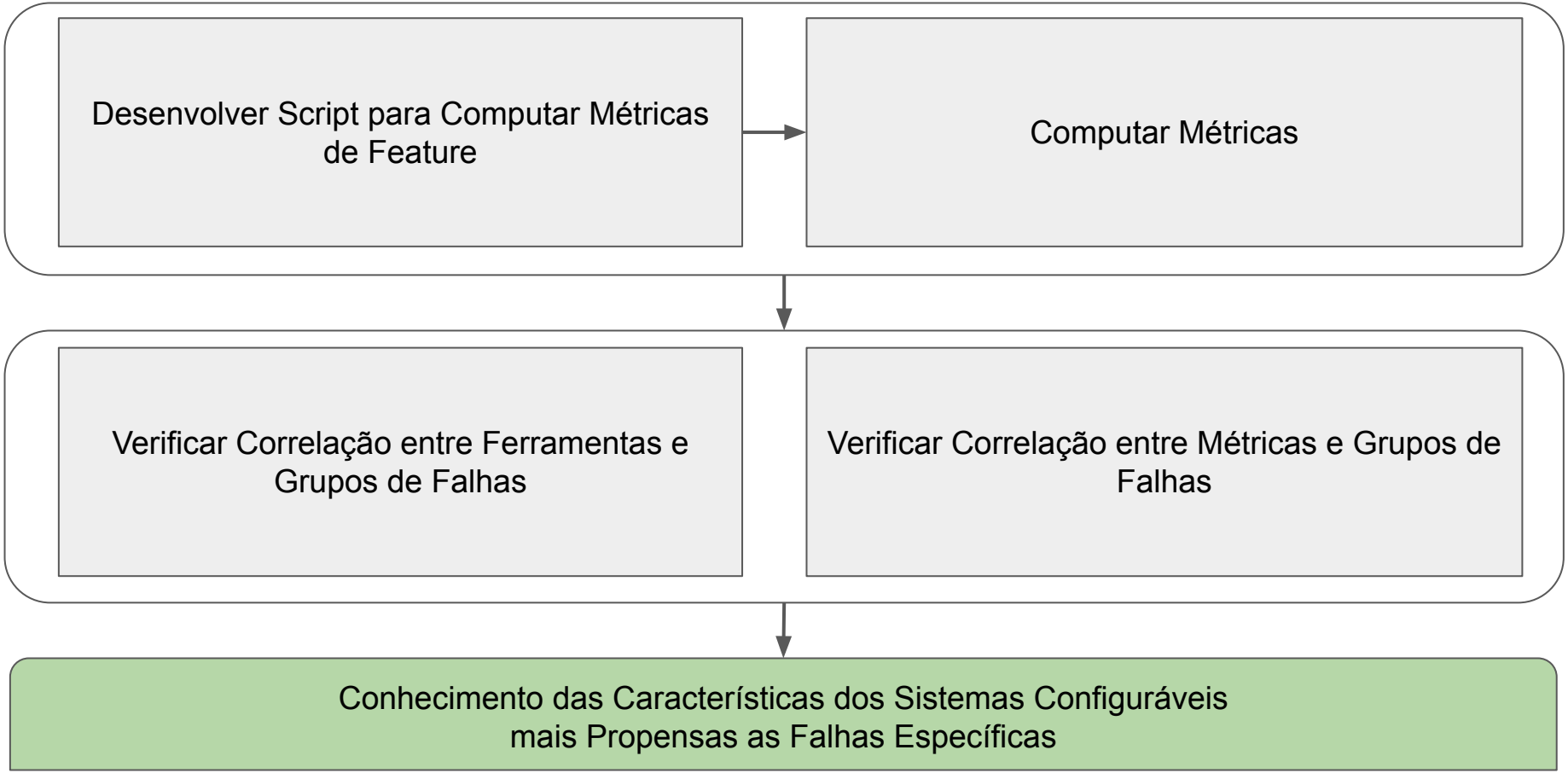
Passos do Projeto



Passo 1 - Criação da Base de Conhecimento



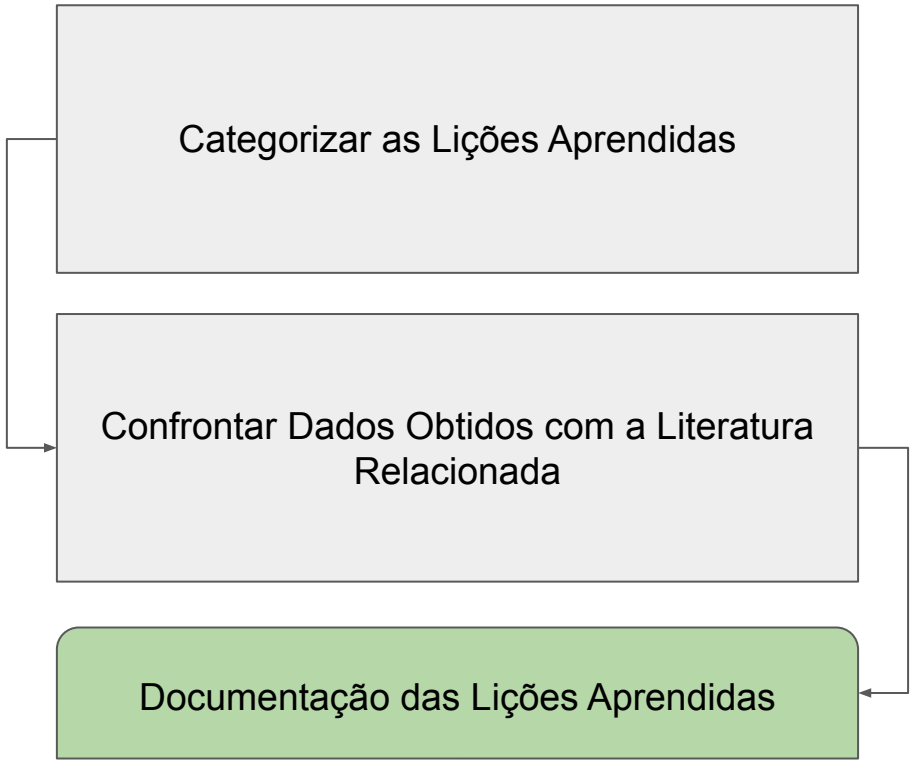
Passo 1 - Criação da Base de Conhecimento



Questão de Pesquisa

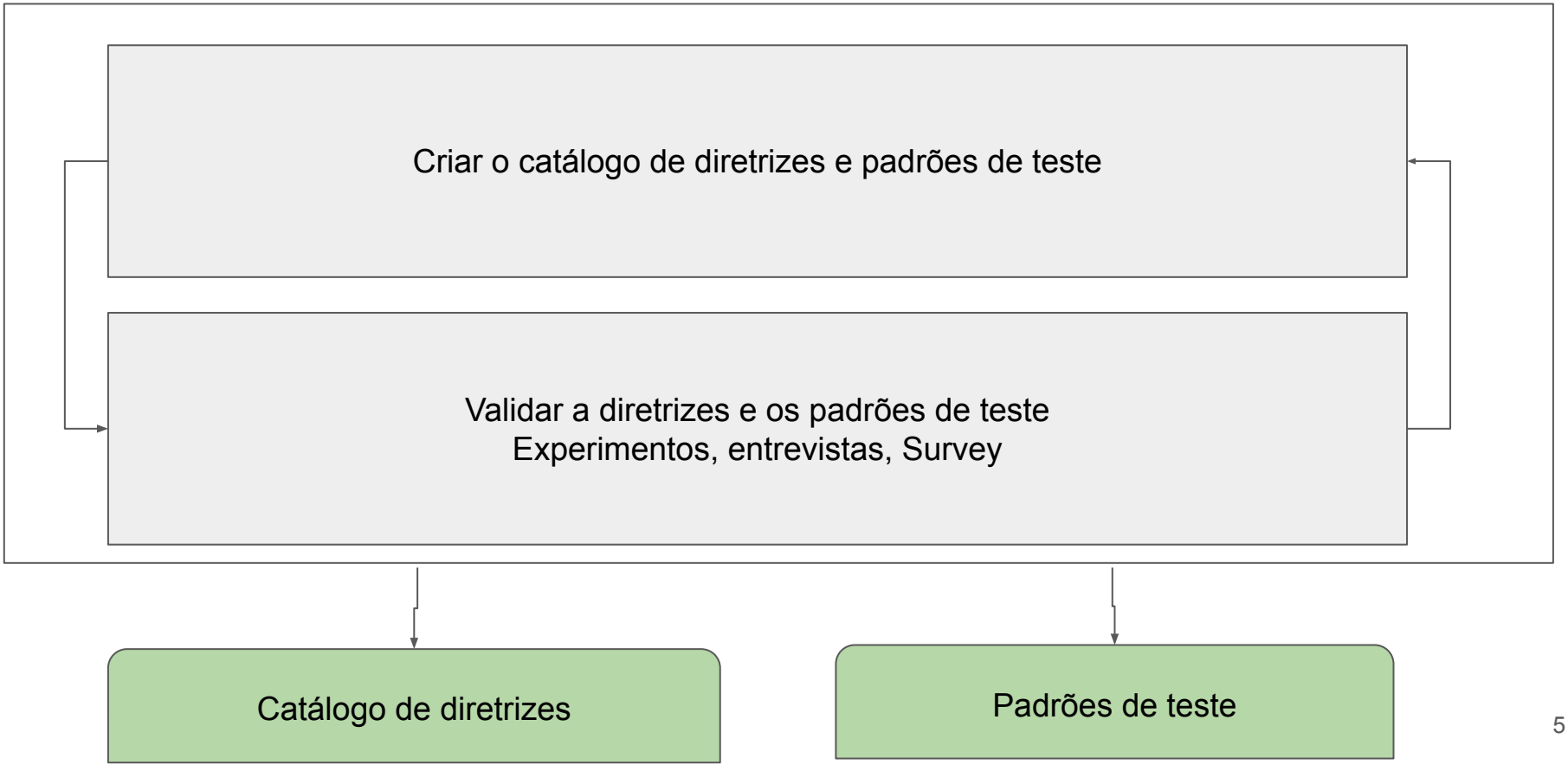
Quais são os grupos das falhas mais recorrentes em sistemas com abundância de recursos?

Passo 2 - Organização do Conhecimento



OE4: Sumarizar as **lições aprendidas** para criação do catálogo e padrões de teste

Passo 3 - Criação do Catálogo de diretrizes e padrões de teste



Esforço para execução da pesquisa (opinião pessoal)

Criação/expansão da suíte de teste

Depuração no código e suíte de teste

Tempo de execução do experimentos

Encontrar o sistemas para compor o dataset

Instrumentalização do sistema alvo para execução do experimento

Código de teste gerado por LLM

Prompt 1: Considere o código Java abaixo. Este código utiliza dois recursos chamados SMILEY e WEATHER. Existe uma suspeita de que um recurso pode estar influenciando o outro recurso. Desta forma, é necessária a criação de um caso de teste que verifique a interação entre os dois recursos. Assim, forneça uma implementação de um caso de teste utilizando o framework JUnit para testar o código em Java. Considere que a entrada dada para o código Java é [:weather:], ou seja: c =[:weather:].

```
1  public class WeatherReport {
2    private Date date;
3    private String temperature;
4    public WeatherReport(Date currentDate, String currentTemperature){
5      this.date = currentDate;
6      this.temperature = currentTemperature;
7    }
8    public String createText(String c) {
9      if (Configuration.SMILEY)
10       c = c.replace(":]", getSmiley(":]"));
11     if (Configuration.WEATHER)
12       c = c.replace("[:weather:]", temperature);
13     return c;
14   }
15 }
```

Código de teste gerado por LLM

```java
public class WeatherReportInteractionTest {

    @Test
    public void testInteractionBetweenWeatherAndSmiley() {
        // Ativa os dois recursos
        Configuration.SMILEY = true;
        Configuration.WEATHER = true;

        String expectedTemperature = "25ºC";
        WeatherReport report = new WeatherReport(new Date(), expectedTemperature) {
            // Stub para controlar o retorno do smiley no teste
            public String getSmiley(String s) {
                return "😊";
            }
        };

        String input = "[:weather:]";

        String result = report.createText(input);

        // O comportamento esperado do WEATHER sozinho seria substituir por 25ºC
        // Mas, devido à interação, isso não acontece
        assertNotEquals(expectedTemperature, result);

        // Mostra o resultado produzido após a interferência do SMILEY
        assertEquals("[:weather😊", result);
    }
}
```

Tarefas de pesquisa

- Revisão sistemática da literatura para identificar ferramentas recentes de testes sistemas configuráveis (em andamento)

CASA-T1, CASA-T2, CASA-T3, CASA-T4, Chvatal-T1, Chvatal-T2, Chvatal-T3, Chvatal-T4, ICPL-T1, ICPL-T2, ICPL-T3, IncLing-T2, YASA-T1, YASA-T2, YASA-T3, and YASA-T4, LEDGE-25%, P2: PLEDGE-50% e P3: PLEDGE-70%

Tarefas de pesquisa

- Criação de dataset de sistemas em abundância de recursos

- Minerar dados de repositórios públicos para coletar e analisar informações sobre código-fonte e dados de teste relacionados a sistemas de software com abundância de recursos.

Tarefas de pesquisa

- Avaliação quantitativa de sistemas com abundância de recursos

- Projetar e criar experimentos com estes sistemas para identificar falhas de interação de recursos.
  - Utilizar ferramentas de testes para sistemas configuráveis aplicado a sistemas com abundância de recursos

Obrigado! Dúvidas?

Fischer Jônatas Ferreira

fischer.ferreira@unifei.edu.br
https://lattes.cnpq.br/1412888913678183

UFMG

lab-soft
software engineering laboratory

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Teste para Sistemas de Software com Abundância de Recursos

LabSoft
Fischer Jônatas Ferreira

12 de março, 2026