Da Validação Manual à Qualidade Contínua: Um Relato de Evolução de Testes Automatizados em um ERP Legado

Icaro Santos de Oliveira
Universidade Estadual do Ceará (UECE)
Fortaleza, Brazil
icaro.santos@aluno.uece.br

F. F. Rodrigues de Oliveira
Universidade Estadual do Ceará (UECE)
Fortaleza, Brazil
felipeo.rodrigues@aluno.uece.br

Ismayle S. Santos
Universidade Estadual do Ceará (UECE)
Fortaleza, Brazil
ismayle.santos@uece.br

Resumo

Este artigo relata a evolução das práticas de garantia de qualidade em um sistema ERP corporativo legado de grande porte, baseado em Protheus, ao longo de seis meses de transição de validações manuais *ad-hoc* para um *pipeline* de qualidade contínua integrado ao CI/CD. A intervenção, conduzida em estágios incrementais, combinou (i) automação de interface de usuário com o *framework* Test Interface Robot (TIR) para rotinas críticas do módulo de Recursos Humanos, e (ii) testes de camada de serviço com Playwright sob política *documentation-first*, com relatórios consolidados pelo Allure Report. Diferentemente de relatos focados em uma única camada ou ferramenta isolada, este estudo descreve a coevolução entre infraestrutura técnica, governança via Gitflow e um Board de Qualidade dedicado à gestão de dívida técnica de teste. Reportamos resultados preliminares — mais de 30 rotinas críticas de folha de pagamento e mais de 1.300 testes de API automatizados — e lições sobre *flakiness* em sistemas legados de alta latência, contribuindo para a literatura de evolução de práticas de teste em contextos industriais.

Palavras-chave

Automação de testes, ERP legado, Evolução de software, Protheus, TIR, Playwright, CI/CD, Relato industrial

1 Introdução

Sistemas de planejamento de recursos empresariais (ERP) sustentam atividades de missão crítica em organizações de grande porte, nas quais módulos heterogêneos — financeiro, Recursos Humanos (RH), suprimentos e manutenção — devem interoperar de forma confiável e auditável [1, 6]. Em uma companhia de economia mista brasileira de prestação de serviços públicos essenciais, um ERP baseado em Protheus Webapp sustenta processos administrativos e operacionais de missão crítica que afetam diretamente a continuidade do serviço aos cidadãos. Antes do esforço aqui relatado, as validações de mudanças no sistema baseavam-se em procedimentos manuais e *ad hoc* — eficazes para verificações pontuais, mas insuficientes para regressões frequentes e para garantir desempenho previsível em janelas críticas, como o fechamento mensal de folha de pagamento. Esse modelo gera ciclos de *feedback* longos, riscos a cada *release* e dependência de conhecimento tácito para triagem de defeitos. As necessidades principais do setor público — auditabilidade, conformidade regulatória e exigência de continuidade de serviço — tornam a transição para uma estratégia sistemática de qualidade um requisito operacional, não apenas técnico.

Embora a literatura sobre garantia de qualidade em sistemas web modernos e microsserviços seja extensa [3, 4], relatos sobre a evolução de práticas de teste em ERPs legados de larga escala — especialmente sob restrições do setor público — permanecem escassos. Estudos recentes em contextos análogos focaram em uma camada por vez: automação de UI em sistemas judiciais [12] ou validação de APIs REST em sistemas governamentais setoriais [9]. ERPs Protheus impõem desafios específicos pouco explorados: alta latência das interfaces Webapp, fragilidade de seletores em formulários complexos, regras de negócio interdependentes entre módulos (por exemplo, RH integrado a financeiro e tributário) e janelas de execução longas que tornam testes E2E inviáveis em todo *pull request*. Este artigo aborda essa lacuna ao reportar uma evolução **multicamada** e **governada** das práticas de teste em um ERP corporativo do setor público.

Apresentamos um relato industrial de seis meses com as seguintes contribuições: (i) **programa em estágios** — descrição replicável da transição de validações manuais para automação multicamada, combinando o TIR [13] para UI e Playwright com Allure Report para API, ambos integrados a Jenkins via *pipeline-as-code*; (ii) **governança institucional** — papel do Gitflow e de um Board de Qualidade dedicado à gestão de *flakiness* e dívida técnica de teste; (iii) **resultados preliminares** — mais de 30 rotinas críticas de RH automatizadas e mais de 1.300 testes de API integrados ao *pipeline*, totalizando mais de 340 *commits* no repositório de UI e mais de 190 no de API; e (iv) **lições aprendidas** sintetizadas em três pilares transferíveis para outros contextos de modernização de ERPs.

A Seção 2 apresenta o *background* e os trabalhos relacionados. A Seção 3 detalha o desenho do estudo, o programa de automação e a infraestrutura de governança. A Seção 4 reporta resultados e lições aprendidas. A Seção 5 discute ameaças à validade e as considerações finais.

2 Background e Trabalhos Relacionados

Esta seção situa o presente trabalho em relação à literatura de automação de testes e à experiência industrial recente. A Subseção 2.1 sintetiza os conceitos fundamentais que motivam as decisões técnicas descritas na Seção 3. A Subseção 2.2 compara o relato com estudos industriais recentes, evidenciando o nicho endereçado.

2.1 Background

A pirâmide de testes preconiza a priorização de verificações em camadas inferiores (unitárias e de serviço) em relação a interfaces gráficas, visando estabilidade e *feedback* ágil [1]. Em ERPs legados, a automação de UI permanece necessária para validar jornadas complexas entre módulos, sendo simultaneamente vulnerável à fragilidade de seletores e a comportamentos assíncronos [2, 10]. Essa fragilidade origina testes *flaky* — execuções não determinísticas que minam a confiança no *pipeline* [8]. A literatura indica que o

padrão *Page Objects* e a segregação de testes E2E para jornadas de alto valor, com deslocamento da regressão para APIs, mitigam esse efeito [7]. A integração contínua transforma verificações episódicas em portões de qualidade visíveis [6], e a unificação de evidências em relatórios estruturados facilita a triagem colaborativa de defeitos [5].

No contexto específico de sistemas ERP, esses desafios se acentuam por três fatores. Primeiro, a interdependência de dados entre módulos (por exemplo, um lançamento de férias no módulo de RH que repercute no módulo financeiro e no tributário) torna as asserções de teste inerentemente multi-tabelares, elevando o risco de falsos positivos quando o estado do banco de dados não é controlado por completo. Segundo, ERPs como o Protheus utilizam interfaces web proprietárias (Webapp/SmartClient) com um DOM que não segue os padrões convencionais de aplicações web modernas, tornando seletores CSS e XPath genéricos particularmente problemáticos nesse contexto. Terceiro, janelas de execução longas —fluxos de folha de pagamento que levam dezenas de minutos para completar — inviabilizam abordagens de *feedback* rápido e exigem infraestrutura dedicada para execução de testes de regressão em escala.

2.2 Trabalhos Relacionados

Relatos industriais recentes oferecem pontos de comparação. Souza et al. [12] descrevem uma jornada de quase uma década em um sistema jurídico público, com automação de UI via Selenium IDE organizada em estágios de maturidade. Morais et al. [9] apresentam o Zettalenium, *framework* próprio para APIs REST em sistema governamental, destacando padronização e auditabilidade. Brasil et al. [3] aplicam *mutation testing* em uma *fintech* com 14 microsserviços. Chaves et al. [4] relatam a operação de uma *device farm* física para testes Android em escala. A Tabela 1 sintetiza as diferenças em relação ao presente trabalho.

Tabela 1: Relatos Industriais de Automação de Testes e o Presente Trabalho.

Trabalho	Domínio	Camada	Ferramentas	Foco
Souza et al. (2025)	Setor público	Interface	Selenium IDE	Estágios de maturidade
Morais et al. (2025)	Setor público	Serviço	Zettalenium	Framework próprio
Brasil et al. (2024)	Fintech	Unidade	PIT/Pitest	Teste de mutação
Chaves et al. (2024)	Mobile	Interface	STELA + Jira	*Device farm*
Este trabalho	**Setor público**	**Interface + Serviço**	**TIR, Playwright, Allure**	**Evolução multicamada governada**

Camada refere-se ao nível da pirâmide de testes endereçado: interface (E2E via GUI), serviço (API) ou unidade.

Diferentemente desses relatos — que tipicamente focam em uma camada ou técnica específica —, o presente trabalho descreve a **coevolução de duas camadas** em um mesmo contexto institucional, com mecanismos de governança que sustentam a evolução ao longo do tempo. O foco em Protheus Webapp via TIR endereça um nicho técnico — ERPs legados de alta latência em setor público — pouco explorado na literatura.

3 Programa de Automação em Estágios

Esta seção descreve o percurso metodológico da intervenção. A subseção 3.1 apresenta o desenho do estudo e as condições de confidencialidade dos dados. A subseção 3.2 define as questões de pesquisa. As subseções 3.3 a 3.5 detalham, respectivamente, os estágios da evolução, a governança adotada e a infraestrutura de CI/CD.

3.1 Desenho do Estudo e Confidencialidade

Conduzimos um estudo de caso industrial [11] em uma companhia de economia mista brasileira de prestação de serviços públicos. A intervenção compreendeu seis meses consecutivos, durante os quais uma equipe dedicada de qualidade — composta por um engenheiro de QA sênior e dois analistas de teste — implementou progressivamente as duas frentes de automação (UI e API) em paralelo com as atividades de evolução do ERP.

O protocolo de coleta de dados triangulou múltiplas fontes: (i) telemetria de execução dos testes (*logs* do TIR, artefatos do Allure Report); (ii) metadados do *pipeline* de CI (Jenkins); (iii) registros de defeitos e decisões de quarentena do Board de Qualidade; e (iv) histórico de *commits* e *pull requests* nos dois repositórios de teste. As questões de pesquisa foram operacionalizadas via estatística descritiva sobre esses artefatos.

A Figura 1 sintetiza a arquitetura metodológica do estudo, reunindo em um único esquema os três elementos que se articulam ao longo desta seção: o programa de automação em estágios (à esquerda), as fontes de dados e instrumentos de coleta (ao centro) e as questões de pesquisa que esses dados operacionalizam (à direita). A leitura horizontal do diagrama revela a lógica de rastreabilidade adotada: cada estágio de automação produz artefatos distintos — *logs* de execução do TIR no Estágio 1 e relatórios Allure no Estágio 2 — que convergem nos metadados do *pipeline* de CI e, a partir daí, alimentam a análise de RQ1 e RQ2. Essa separação entre fontes de evidência garante que as duas questões de pesquisa sejam respondidas por dados complementares, e não pelos mesmos artefatos, reduzindo o risco de viés de confirmação na interpretação dos resultados.

Todo o material apresentado neste artigo passou por revisão de anonimização conduzida pela coordenação de segurança da informação da organização; as condições de disponibilidade dos artefatos são detalhadas ao final deste artigo.

3.2 Questões de Pesquisa

O estudo é guiado por duas questões de pesquisa (RQs):

RQ1 (Viabilidade e Cobertura): De que forma a adoção do TIR e o mapeamento extensivo de APIs viabilizam a automação de fluxos críticos de RH em um ERP legado de alta complexidade?

RQ2 (Estabilidade e Infraestrutura): *Quais padrões recorrentes de instabilidade emergem ao escalar uma suíte de testes em um sistema legado de alta latência, e quais mitigações se mostraram eficazes na prática?* Trata-se de uma investigação qualitativa, baseada na caracterização de padrões observados e das intervenções adotadas, e não de uma medição estatística de taxas de *flakiness*.

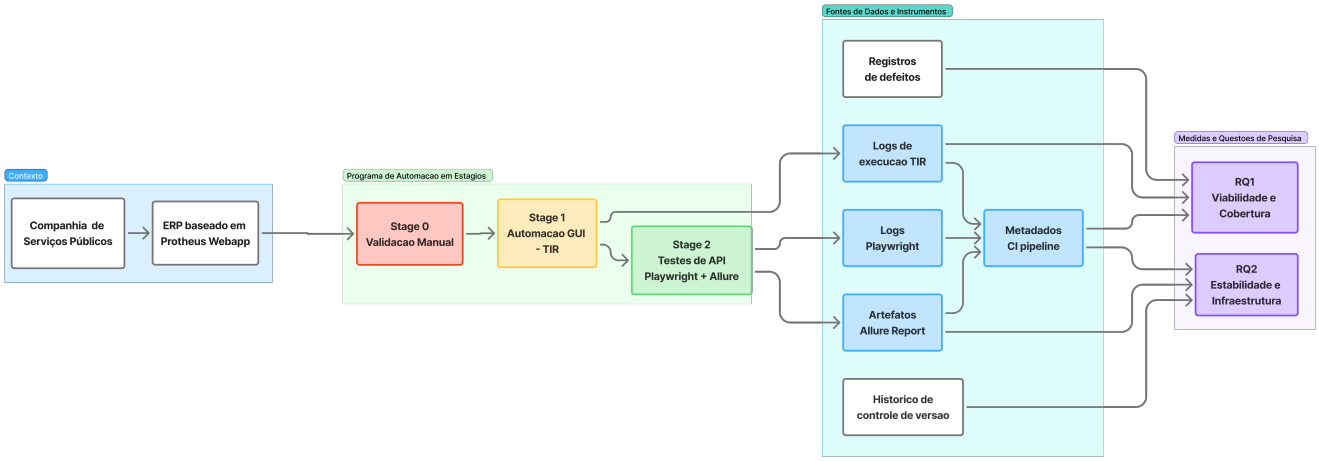

Figura 1: Fluxo metodológico do estudo. O programa de automação em estágios (centro-esquerda) opera sobre o ERP Protheus Webapp (contexto), gerando dados em múltiplas fontes (centro-direita) que operacionalizam as duas questões de pesquisa (base).

3.3 Estágios da Evolução

A intervenção foi estruturada em três estágios evolutivos partindo de um *baseline* manual, conforme ilustrado na Figura 1.

Estágio 0 — Validação manual *ad-hoc*. Antes da iniciativa, as validações dependiam exclusivamente de procedimentos manuais executados pelos próprios analistas de negócio, sem rastreabilidade sistemática de defeitos ou cobertura de regressão. Cada *release* era validado pontualmente, com alto risco de regressões não detectadas. Esse cenário serviu como linha de base para avaliação dos ganhos das etapas seguintes.

Estágio 1 — Automação de UI com TIR. Iniciou-se pela camada de interface por dois motivos estratégicos: gerar visibilidade imediata para os *stakeholders*, ao demonstrar que o processo crítico de fechamento de folha podia ser reproduzido sem intervenção humana; e endereçar as jornadas de mais alto risco antes de avançar para camadas internas.

O *framework* TIR [13] foi adotado por oferecer integração nativa com o DOM do Protheus Webapp, reduzindo a volatilidade de seletores comparada ao uso de Selenium genérico. Os *scripts* foram estruturados com o padrão *Page Objects* para isolar a lógica de teste da volatilidade da interface. Para acelerar a criação dos *scripts*, utilizou-se o TIR Record Protheus 1.0.0, uma extensão de navegador que captura interações do usuário no Protheus WebApp e as traduz automaticamente em código TIR, reduzindo significativamente o tempo de autoria inicial. O Código 1 exemplifica a estrutura de um caso de teste TIR para o cadastro de funcionário, ilustrando como o *framework* abstrai as interações com a interface do Protheus.

Listing 1: Estrutura de um caso de teste TIR para cadastro de funcionário no Protheus (simplificado e sem dados reais).

```
from tir import Webapp
import unittest

class TEST010(unittest.TestCase)::
    @classmethod
    def setUpClass(inst):
        inst.oHelper = Webapp()
        inst.oHelper.Setup("SIGA", "01/01/2025")
        inst.oHelper.Program("SISRH")

    def test_cadastro_mensalista_001(self):
        self.oHelper.SetButton("Incluir")
        self.oHelper.SetValue("NOME", "FUNC TESTE")
        self.oHelper.SetValue("ADMISSAO", "123123")
        self.oHelper.SetValue("FUNCAO", "M")
        self.oHelper.SetButton("Confirmar")
        self.oHelper.AssertTrue()

    @classmethod
    def tearDownClass(inst):
        inst.oHelper.TearDown()

if __name__ == '__main__':
    unittest.main()
```

Ao término do período, a suíte cobria mais de 30 rotinas críticas de RH (módulo do sistema), incluindo cadastro de funcionários em múltiplas categorias (mensalista, estagiário, aprendiz e autônomo), cálculos de adiantamento e folha de pagamento, gestão de férias coletivas e programadas, rescisões, integração de cálculos, fechamento mensal, apuração de ponto eletrônico, além de rotinas de gestão de usuários e adiantamentos. A suíte completa demanda de 3 a 4 horas de execução — reflexo direto da alta latência do sistema legado — e os resultados são visualizados em um *dashboard* web dedicado (Portal da Qualidade), executado via *containers* Podman.

Estágio 2 — Testes de API com Playwright e Allure. A validação da camada de serviços foi deliberadamente isolada em um projeto separado, utilizando Playwright com TypeScript. A decisão de segregar as suítes decorreu de uma política *documentation-first*: nenhuma API é automatizada sem que seu contrato esteja formalmente documentado, garantindo estabilidade dos contratos de serviço. Essa abordagem reduziu o acoplamento entre a suíte e a implementação, tornando os testes mais resilientes a mudanças de interface. Os relatórios são consolidados pelo Allure Report — organizado em *epics*, *features* e *stories* — e publicados automaticamente pelo *plugin* do Jenkins. Ao fim do período, a suíte contava com mais de 1.300 testes de API, com tempo médio de execução de 15 minutos por *job*.

3.4 Governança e Sustentabilidade

A sustentabilidade do programa além do esforço inicial exigiu a institucionalização de práticas de governança. O Gitflow foi adotado como estratégia de ramificação para todos os ativos de teste: cada nova automação parte de uma *feature branch*, passa por revisão de código entre pares da equipe de QA e integra a *branch* principal somente após aprovação. Essa prática garantiu aderência ao padrão *Page Objects* e reduziu a introdução de *scripts* frágeis no *pipeline*.

Um Board de Qualidade dedicado foi criado com responsabilidades explícitas: (i) revisão periódica da suíte com foco na taxa de *flakiness*; (ii) priorização de novas automações de acordo com o risco de negócio das rotinas; e (iii) gestão da *quarantine lane* — mecanismo que isola testes instáveis, impedindo que bloqueiem *merges* enquanto são investigados. A combinação de Gitflow e Board de Qualidade converteu a automação de um esforço pontual em uma prática institucional sustentável.

3.5 Infraestrutura de CI/CD

O *pipeline* foi estruturado como código (*Jenkinsfile*) no Jenkins, com estágios independentes para cada frente de automação. Os testes de UI (TIR) são executados em ambiente Linux virtualizado via Podman, superando a incompatibilidade inicial entre o ambiente de desenvolvimento local (Windows) e os *runners* de CI (Linux). Os testes de API (Playwright) são disparados em todo *pull request* que toque o *back-end*, com relatórios consolidados pelo *plugin* Allure do Jenkins. A infraestrutura cobre três ambientes: desenvolvimento, teste e homologação.

4 Resultados Preliminares e Lições Aprendidas

Nesta seção, apresentamos os resultados organizados por questão de pesquisa (Subseções 4.1 e 4.2), seguidos das lições aprendidas (Subseção 4.3). Para cada RQ, adotamos a estrutura *achado — interpretação — relevância*, distinguindo o dado observado da reflexão sobre seu significado e sua transferibilidade.

4.1 RQ1: Viabilidade e Cobertura

Achado. A suíte de UI cobre mais de 30 rotinas críticas do processo de folha de pagamento, abrangendo cadastro de funcionários em múltiplas categorias, transferências, cálculos de adiantamento, benefícios, plano de saúde, férias coletivas e programadas, rescisões, integração de cálculos e fechamento mensal, além de gestão de usuários e adiantamento de viagem. A suíte de API ultrapassa 1.300 testes automatizados. A Tabela 2 resume a escala alcançada por estágio. A Tabela 3 detalha as rotinas automatizadas por módulo do ERP. Em termos de esforço de desenvolvimento, o repositório de automação de UI totaliza 76 arquivos Python e mais de 4.000 linhas de código de teste, distribuídas entre *scripts* de caso de teste, módulos utilitários de manipulação de datas e funções de validação específicas por rotina.

Nota sobre a unidade de contagem. Distinguimos dois níveis de granularidade. Um *programa* corresponde a uma rotina do ERP (por exemplo, o cadastro de funcionários), enquanto um *cenário de teste* corresponde a uma variação de negócio automatizada dentro daquele programa (por exemplo, cadastro de mensalista, de estagiário e de aprendiz, três cenários sobre o mesmo programa). A Tabela 3 lista os 17 programas automatizados, que juntos totalizam mais de 30 cenários de teste distintos, número reportado ao longo deste artigo.

Tabela 2: RQ1 — Escala e Cobertura por Estágio.

Métrica	Est. 0	Est. 1	Est. 2
Rotinas de RH automatizadas	Manual	30+	—
Testes de API automatizados	0	—	1.300+
Commits no repositório	—	340+	190+
Arquivos Python (*codebase*)	—	76	—
Linhas de código de teste	—	4.000+	—
Tempo de execução (*job*)	N/A	3–4 h	~15 min

Tabela 3: RQ1 — Rotinas automatizadas por módulo do ERP no Estágio 1.

Módulo	Código	Descrição
SISRH	TEST010	Cadastro de funcionários (Mensalista, Estagiário, Aprendiz)
SISRH	TEST265	Cadastro de Contrib. Individual (Autônomo)
SISRH	TEST180	Transferência de funcionários
SISRH	TEST340	Cadastro de sindicatos
SISRH	TEST400	Validação de períodos
SISRH	TEST020	Cálculo por roteiros (Adiantamento / Folha)
SISRH	TEST016	Cálculo de plano de saúde
SISRH	TEST019	Integração de cálculos
SISRH	TEST040	Cálculo de rescisão
SISRH	TEST060	Férias coletivas/programadas
SISRH	TEST120	Fechamento mensal do período
SISRH	TEST011	Substituição de funcionário
SISRH	TESTCALC	Cálculo de benefícios (VA, VR, VT)
SISRH	TESTP010	Leitura e apontamento de marcações de ponto
SISRH	TEST090	Fechamento mensal do ponto
SISCONF	TEST510	Gestão de usuários
SISADV	TESTR02	Adiantamento de viagem

Interpretação. A escala da cobertura de UI é compatível com relatos análogos — Souza et al. [12] reportam aproximadamente 30 fluxos em sistema de complexidade similar. A adoção do TIR mostrou-se decisiva: sua integração nativa com o DOM do Protheus Webapp reduziu a volatilidade de seletores que inviabilizaria o uso de Selenium genérico. Na frente de APIs, a política *documentation-first* viabilizou cobertura ampla sem a instabilidade típica de testes acoplados à implementação. Cabe ressaltar que a cobertura alcançada engloba o fluxo completo de geração de folha de pagamento — da preparação do período ao fechamento mensal definitivo — representando a jornada mais complexa e de maior risco para a organização.

Por que importa. Esses resultados validam que ERPs Protheus — frequentemente considerados refratários à automação por sua complexidade e alta latência — são sistematicamente automatizáveis quando se adota um *framework* especializado para a camada de UI e uma política contratual rigorosa para a camada de API.

4.2 RQ2: Estabilidade e Infraestrutura

Achado. A instabilidade observada nos testes de UI decorreu predominantemente de três padrões, descritos a seguir com exemplos concretos.

O primeiro padrão, denominado *warning dialog inesperado*, ocorria quando o Protheus exibia diálogos modais de aviso durante a execução de determinadas rotinas (por exemplo, confirmações de período ou alertas de integridade de dados). O TIR não esperava

esses diálogos por padrão, causando falhas por *timeout* em fluxos intermediários. A mitigação consistiu em adicionar tratamentos explícitos de *warning dialogs* nos *scripts*, utilizando métodos da API do TIR para detectar e fechar janelas modais antes de prosseguir.

O segundo padrão, denominado *incompatibilidade de versão*, manifestou-se na atualização anual obrigatória do Protheus. A TOTVS disponibiliza periodicamente uma nova versão do *backoffice*, e os contratantes do ERP devem implementá-la para cumprir obrigações contratuais. Essa atualização alterou o comportamento de determinados formulários e seletores do DOM, quebrando *scripts* que funcionavam na versão anterior.

O terceiro padrão, denominado *divergência de plataforma*, decorreu de diferenças de comportamento entre o ambiente local de desenvolvimento (Windows) e o ambiente de CI (Linux virtualizado via Podman). Determinados fluxos que passavam localmente falhavam no *runner* de CI por diferenças de resolução de tela, tempos de resposta e configuração do Web Agent do Protheus. A mitigação envolveu a padronização do ambiente de execução via Podman e a adição de *waits* adaptativos nos *scripts*.

A Tabela 4 compara os parâmetros de estabilidade entre os dois estágios.

Tabela 4: RQ2 — Parâmetros de Estabilidade por Estágio.

Parâmetro	Est. 1 (UI)	Est. 2 (API)
Tempo de execução	3–4 horas	~15 minutos
Origem da instabilidade	Infra / versão ERP	Contrato / dados
Dependência do ambiente de teste	Crítica	Moderada
Esforço de manutenção	Alto (sincronismo)	Baixo (documentação)

Interpretação. A origem da *flakiness* é sistêmica, não lógica: os *scripts* estavam corretos, mas o ambiente de execução introduzia variabilidade [8, 10]. A adição de sincronizações explícitas e o tratamento de *warning dialogs* eliminaram as ocorrências recorrentes desses dois padrões nas execuções subsequentes, conforme observado pela equipe no acompanhamento das execuções noturnas. Ressaltamos que essa constatação é qualitativa: a instrumentação sistemática de taxas de falha por *sprint* teve início após o período coberto por este relato. A *quarantine lane*, gerenciada pelo Board de Qualidade, isolou os testes instáveis remanescentes, preservando a confiabilidade do sinal de qualidade nos *merges*. Esses padrões são consistentes com a taxonomia de causas de *flakiness* relatada por Luo et al. [8], na qual a dependência de ordem de execução e a sensibilidade ao ambiente figuram como causas predominantes.

4.3 Lições Aprendidas

A especialização do framework *de UI é condição necessária para a automação em ERPs Protheus.* O TIR foi decisivo por sua integração nativa com o DOM do Protheus Webapp. Ferramentas genéricas exigem adaptações extensas para lidar com os seletores voláteis do sistema, tornando a manutenção proibitiva em escala. O padrão *Page Objects* complementou essa estabilidade ao isolar a lógica do teste das variações de interface, concentrando o esforço de manutenção em pontos bem definidos e transferíveis a outros sistemas com interfaces igualmente instáveis.

A política documentation-first *é o principal mecanismo de estabilidade na camada de API.* A exigência de documentação formal antes de qualquer automação criou contratos de serviço estáveis que blindaram os testes contra mudanças de implementação. Ao mesmo tempo, forçou as equipes de desenvolvimento a manter a documentação atualizada, gerando benefícios além da qualidade de teste: rastreabilidade e governança de APIs. Essa lição é transferível a qualquer organização que adote automação de API em sistemas com múltiplos consumidores internos.

A transição de testers manuais para QA-coders demanda investimento deliberado em cultura, não apenas em ferramentas. O gargalo inicial não era a curva de aprendizado do TIR, mas a familiaridade da equipe com controle de versão e revisão de código. A institucionalização do Gitflow — com *feature branches*, revisão entre pares e padrões de nomenclatura para testes — foi o principal vetor de nivelamento. O Board de Qualidade consolidou essa cultura ao tornar a gestão da suíte uma responsabilidade coletiva e explícita, não individual.

A suíte de testes é, ela própria, um artefato de software que coevolui com o sistema sob teste. Ao longo dos seis meses, os repositórios de automação acumularam mais de 530 *commits*, refletindo um ciclo contínuo de manutenção evolutiva: atualizações do Protheus exigiram refatoração de seletores; mudanças em regras de negócio de RH demandaram revisão de asserções; e a própria infraestrutura de execução migrou de ambientes locais para *containers*. Esse padrão evidencia que a dívida técnica de teste em sistemas legados não é um passivo estático, mas um fluxo que exige governança permanente — observação alinhada à literatura de evolução de software, na qual a manutenção de artefatos de verificação acompanha a taxa de mudança do sistema verificado.

5 Considerações Finais e Trabalhos Futuros

Esta seção sintetiza as ameaças à validade (Subseção 5.1), as conclusões do relato (Subseção 5.2) e as direções de trabalhos futuros (Subseção 5.3).

5.1 Ameaças à Validade

Quanto à **validade interna**, este é um estudo de caso único conduzido em uma organização com pilha tecnológica e restrições institucionais específicas. Mudanças organizacionais ou atualizações do ERP podem alterar as condições de reprodução dos resultados. As métricas quantitativas de *flakiness* (taxa por *sprint* e MTTR de testes) estão em fase de coleta sistemática, o que limita a precisão das afirmações sobre taxas de instabilidade. Além disso, a equipe de QA era composta por três profissionais, e a generalização para equipes de tamanho diferente requer cautela.

Quanto à **validade externa**, a especificidade do Protheus Webapp como alvo do TIR limita a generalização direta para outros ERPs ou sistemas web. Contudo, os princípios de automação em estágios, a política *documentation-first* e a governança via Gitflow são transferíveis para outros sistemas corporativos de alta criticidade que buscam modernizar suas práticas de qualidade. Os três padrões de *flakiness* identificados na RQ2 são inerentes a qualquer sistema legado com interfaces proprietárias, sugerindo aplicabilidade além do Protheus.

Quanto à **validade de constructo**, a cobertura foi medida em termos de rotinas automatizadas (UI) e testes criados (API), não de cobertura de código ou critérios de adequação formal. Essas métricas capturam a abrangência funcional, mas não garantem a ausência de lacunas em caminhos de execução menos frequentes.

5.2 Conclusão

Este artigo relatou a evolução das práticas de garantia de qualidade em um ERP Protheus de grande porte, ao longo de seis meses de transição de validações manuais para um programa de automação multicamada integrado ao CI/CD. Os resultados — mais de 30 rotinas de RH e 1.300+ testes de API automatizados — demonstram a viabilidade técnica e a escalabilidade da abordagem em sistemas legados de alta complexidade. A governança via Gitflow e o Board de Qualidade revelaram-se tão críticos quanto as ferramentas para a sustentação do programa ao longo do tempo. As lições aprendidas — especialização do *framework*, política *documentation-first* e investimento em cultura de engenharia — são aplicáveis a outros contextos institucionais que enfrentem desafios análogos de modernização de qualidade em ERPs.

5.3 Trabalhos Futuros

Como direções de trabalhos futuros, identificamos quatro frentes. Primeiro, pretende-se instrumentar métricas quantitativas de *flakiness* — taxa de falhas não determinísticas por *sprint* e MTTR de testes — para embasar decisões de quarentena com evidências numéricas em vez de avaliação qualitativa. Segundo, planeja-se expandir a automação para módulos adicionais do ERP, incluindo eSocial e financeiro, ampliando a cobertura multicamada para rotinas de diferentes domínios de negócio. Terceiro, a introdução de testes de carga via k6 como portão de desempenho no *pipeline* de homologação integraria requisitos não funcionais ao programa de qualidade, complementando as duas frentes funcionais atuais. Quarto, a adoção do PROBAT — conjunto de ferramentas nativas do TlppCore para testes unitários em programas TLPP/AdvPL — viabilizaria a cobertura da base da pirâmide de testes no ecossistema Protheus, fechando o ciclo de automação desde testes unitários até testes E2E.

Disponibilidade de Artefatos

Os artefatos de teste descritos neste relato foram desenvolvidos no contexto operacional de uma companhia de economia mista, sujeita a normas de governança corporativa e sigilo de dados. Os *scripts*, *logs* de execução e configurações de *pipeline* não são, portanto, disponibilizados publicamente. Todo o material apresentado neste artigo passou por revisão de anonimização conduzida pela coordenação de segurança da informação da organização, assegurando a ausência de credenciais, endereços internos, dados pessoais ou configurações proprietárias. Para mitigar parcialmente a limitação de reprodutibilidade, o Código 1 apresenta um exemplo representativo e anonimizado da estrutura dos casos de teste, e a Seções 3 descreve a arquitetura, as ferramentas e as decisões de configuração com detalhe suficiente para replicação da abordagem em contextos institucionais análogos.

Agradecimentos

Agradecemos à **CAGECE** (Companhia de Água e Esgoto do Ceará) pela parceria e pelo apoio à realização deste trabalho, e ao **Governo do Estado do Ceará** pelo apoio institucional que viabilizou este trabalho. Esta pesquisa contou com apoio, em parte, do **CNPq** (Conselho Nacional de Desenvolvimento Científico e Tecnológico). Ferramentas de IA generativa foram utilizadas para revisão gramatical e de clareza textual do manuscrito, sem geração de conteúdo técnico, análises ou resultados. As opiniões expressas são de responsabilidade dos autores e não refletem, necessariamente, as posições das instituições apoiadoras.

Referências

[1] Paul Ammann and Jeff Offutt. 2017. *Introduction to Software Testing* (2 ed.). Cambridge University Press, Cambridge, UK.

[2] Antonia Bertolino. 2007. Software Testing Research: Achievements, Challenges, Dreams. In *Future of Software Engineering (FOSE'07)*. IEEE Computer Society, Los Alamitos, CA, USA, 85–103. doi:10.1109/FOSE.2007.25

[3] Antonio Ricardo Alexandre Brasil, Maria Clara Talhate De Souza, Fernanda Domenicali, Renan Machado, and Jose Urbano Duarte Junior. 2024. Mutation Testing as a Quality Assurance Technique in a Fintech Company: An Experience Report in The Industry. In *Proceedings of the XXIII Brazilian Symposium on Software Quality (SBQS)*. ACM, Salvador, BA, Brasil, 446–451. doi:10.1145/3701625.3701629

[4] Lennon Chaves, Flavia Camila Morais de Oliveira, Leonardo Tiago, and Renata Gabriella Vieira Castro. 2024. Benefits and Challenges of a Physical Device Farm for Automated Software Testing: A Case Study of the STELA Tool Implementation. In *Anais do IX Simpósio Brasileiro de Testes de Software Sistemático e Automatizado (SAST 2024)*. Sociedade Brasileira de Computação, Porto Alegre, RS, Brasil, 89–91. doi:10.5753/sast.2024.3880

[5] Floris MA Erich, Chintan Amrit, and Maya Daneva. 2017. A qualitative study of DevOps usage in practice. *Journal of software: Evolution and Process* 29, 6 (2017), e1885.

[6] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional, Boston, MA, USA.

[7] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2018. Pesto: Automated migration of DOM-based Web tests towards the visual approach. *Software Testing, Verification And Reliability* 28, 4 (2018), e1665.

[8] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An Empirical Analysis of Flaky Tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. Association for Computing Machinery, New York, NY, USA, 643–653. doi:10.1145/2635868.2635920

[9] João Victor de Morais, Neumar Costa Malheiros, and Ricardo Terra. 2025. Automação de Testes de APIs REST no SisNIR. In *Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado (SAST 2025)*. Sociedade Brasileira de Computação, Porto Alegre, RS, Brasil, 138–140. doi:10.5753/sast.2025.13084

[10] Rudolf Ramler, Michael Felderer, Takashi Kitamura, and Darko Marinov. 2016. Industry-Academia Collaboration in Software Testing: An Overview of TAIC PART 2016. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, Piscataway, NJ, USA, 238–239. doi:10.1109/ICSTW.2016.19

[11] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. 2012. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, Hoboken, NJ, USA. doi:10.1002/9781118181034

[12] Bruno Pedraça de Souza, Rebeca Campos Motta, Jessica Soares da Costa, Elaine Nunes, Gustavo Rocha Barreto Pinto, and Rafael Maiani de Mello. 2025. A Journey of Functional Test Evolution in the Public Sector. In *Anais do X Simpósio Brasileiro de Testes de Software Sistemático e Automatizado (SAST 2025)*. Sociedade Brasileira de Computação, Porto Alegre, RS, Brasil, 147–149. doi:10.5753/sast.2025.13869

[13] TOTVS. 2026. TIR: Test Interface Robot — Python module for automated testing of Protheus web interfaces. https://github.com/totvs/tir. Acesso em maio de 2026.