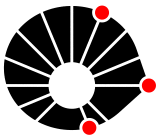

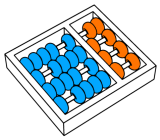

Auto-Diagnóstico da Adoção de Práticas Contínuas com Zeppelin

B. Robazza

Relatório Técnico - IC-PFG-26-03
Projeto Final de Graduação
2026 - Junho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Auto-Diagnóstico da Adoção de Práticas Contínuas com Zeppelin

Breno Robazza*

Resumo

A adoção de práticas de Engenharia Contínua de Software (ECS) tem se mostrado um diferencial competitivo na indústria. Para mensurar essa adoção, foi proposto o instrumento Zeppelin, baseado nos modelos *Stairway to Heaven* e *Eye of CSE*. Este trabalho descreve o desenvolvimento e a avaliação do módulo de Questionário para uma interface digital projetada para operacionalizar o instrumento Zeppelin. O desenvolvimento priorizou a criação de uma *Single Page Application* em React, implementando políticas de isolamento de contexto e mecanismos de persistência de dados. A solução foi avaliada por meio de testes automatizados de *front-end* e submetida à avaliação conceitual junto aos idealizadores da ferramenta. Os resultados confirmam que a interface *web* digitalizou o processo de diagnóstico, viabilizando integrações com motores de recomendação e de análise de dados.

1 Introdução

A necessidade de adaptação rápida às demandas de mercado impulsionou a evolução das metodologias ágeis em direção à Engenharia Contínua de Software (ESC) [5]. Nesse contexto, mensurar a maturidade dos processos internos das organizações de desenvolvimento de *software* tornou-se um desafio, culminando na criação de instrumentos de diagnóstico acadêmicos, como o Zeppelin [2]. Apesar de sua validade teórica, a aplicação puramente manual ou descentralizada desses instrumentos dificulta a escalabilidade, o armazenamento histórico e a análise cruzada de dados das empresas avaliadas.

Para mitigar essa limitação empírica, a motivação central deste projeto é digitalizar o instrumento em uma ferramenta. Este trabalho tem como objetivo construir e validar o módulo de questionário para o instrumento de autodiagnóstico Zeppelin, integrando-o a uma plataforma *web* de avaliação de maturidade. O foco do desenvolvimento reside em fornecer uma interface de usuário dinâmica e acessível para a coleta de respostas, permitindo a persistência das avaliações e preservando o isolamento dos dados organizacionais.

A principal contribuição deste trabalho é a materialização de uma arquitetura que traduz requisitos acadêmicos em restrições de *software*, viabilizando a geração futura de recomendações automatizadas com base nas respostas coletadas.

*Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP. Trabalho de Conclusão de Curso orientado pelo Prof. Dr. Breno Bernard Nicolau de França.

O restante deste documento está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica; a Seção 3 detalha os métodos de desenvolvimento adotados no trabalho; a Seção 4 expõe a arquitetura da solução proposta; a Seção 5 discute as avaliações do sistema; e a Seção 6 apresenta as conclusões e os trabalhos futuros.

2 Fundamentação Teórica

2.1 Engenharia Contínua de Software

Na indústria de *software* contemporânea, a velocidade de desenvolvimento tem se tornado uma das alavancas que definem a relevância de uma organização no mercado diante de seus concorrentes [5]. O lançamento frequente de capacidades tornou-se um diferencial competitivo, impulsionando a evolução acelerada de funcionalidades.

O mercado buscou se adequar a essa realidade por meio da adoção de metodologias ágeis; no entanto, as abordagens ágeis convencionais não são plenamente capazes de mitigar os conflitos decorrentes dessa aceleração nas entregas. Isso ocorre porque diferentes setores de uma mesma empresa costumam ser cobrados por metas divergentes: a área de negócios busca o lançamento de funcionalidades o quanto antes para antecipar o retorno financeiro; a equipe de desenvolvimento é pressionada a projetar, codificar, testar e implantar rapidamente, o que pode comprometer a qualidade interna do *software*; e a equipe de operações é cobrada pela estabilidade do ambiente de produção, enquanto modificações constantes tendem a elevar o risco de incidentes na infraestrutura. Esse desalinhamento de incentivos dá origem a barreiras interdepartamentais de comunicação e alinhamento [6][4], que fazem com que a implantação de novos recursos dependa de passagens de bastão manuais, propensas a erros. Para integrar esses processos, a ECS propõe unificar os fluxos e ciclos de desenvolvimento em atividades executadas com maior frequência e em ciclos curtos [5, 3].

A ECS estende as fronteiras das abordagens ágeis convencionais. Em vez de restringir seu escopo à equipe de desenvolvimento, ela agrupa uma série de práticas contínuas que alinham a estratégia de negócios, a engenharia de software e as operações de infraestrutura. Na prática, essa abordagem fundamenta-se na integração de práticas contínuas, tais como: o planejamento de negócios (*Continuous Planning*), a integração contínua (*Continuous Integration*), a implantação contínua (*Continuous Deployment*) e o *feedback* e monitoramento do uso em tempo real (*Continuous Real-Time User Monitoring*). A partir da adoção dessas práticas coordenadas, o ciclo completo entre a concepção de uma ideia, o projeto, a escrita do código, a publicação em produção, a análise de dados e a geração de novas demandas deixa de durar meses e passa a ocorrer em escala de horas, promovendo um alinhamento constante entre as áreas da organização.

2.2 O Modelo *Stairway to Heaven* (StH)

Para mapear o processo evolutivo das empresas na adoção de práticas contínuas, Olsson *et al.* [6] propuseram o modelo *Stairway to Heaven* (StH). Este modelo descreve a progressão em cinco degraus principais: Tradicional, Organização Ágil, Integração Contínua, Implantação Contínua e R&D como Sistema de Experimentos.

O primeiro estágio, denominado Desenvolvimento Tradicional, caracteriza-se por ciclos longos semelhantes ao modelo de Cascata e aos primeiros processos iterativos das décadas de 80 e 90 [7], com silos rígidos de arquitetura, desenvolvimento e testes, nos quais o *feedback* do cliente ocorre apenas ao fim do ciclo de desenvolvimento.

O segundo degrau, a Organização de R&D Ágil, introduz metodologias ágeis no desenvolvimento de produto, mas ainda mantém a validação final do sistema e a gestão de produtos sob uma abordagem sequencial tradicional.

O nível subsequente, de Integração Contínua (CI), estabelece a automação da compilação, dos testes e da integração do código, garantindo que o *software* esteja sempre testado e em estado funcional.

A Implantação Contínua (CD), por sua vez, parte do *software* validado em testes automatizados e publica em ambiente de produção de forma também automatizada, entregando valor ao cliente final logo após a validação.

O estágio de R&D como Sistema de Experimentos avança para a utilização de dados de uso real e telemetria coletados diretamente do ambiente de produção para realizar experimentos controlados, como em testes A/B, guiando as decisões de negócio e o *roadmap* do produto com base em dados empíricos.

2.3 O Instrumento Zeppelin

Como forma de diagnosticar a adoção prática desses modelos nas empresas, Santos Júnior *et al.* [2] conceberam o Zeppelin. Seu objetivo é avaliar o grau de adoção das práticas de engenharia contínua nas organizações por meio de uma abordagem flexível e não linear. O instrumento reconhece que a maturidade de uma empresa não evolui de forma estritamente sequencial, permitindo identificar, simultaneamente, a presença de práticas em diferentes níveis, como, por exemplo, a execução de rotinas de implantação contínua em produção, mesmo sem a consolidação completa da automação de testes em nível de integração contínua.

A dissertação de Araújo [1] contribuiu para o refinamento dos pesos atribuídos a cada prática no cálculo final de adoção do Zeppelin, por meio de uma calibragem baseada em análise de sensibilidade, possibilitando a geração de recomendações personalizadas voltadas à evolução gradual e contextualizada da organização.

2.4 Dimensões de Avaliação da ECS

Embora o modelo *Stairway to Heaven* (StH) proponha uma progressão linear e vertical, a adoção prática da ECS ocorre de forma multidimensional. Visualizar a evolução organizacional de maneira estritamente linear pode ser rígido demais para a realidade industrial, uma vez que uma organização pode possuir maturidade avançada na automação de implantação (CD) e, ao mesmo tempo, apresentar falhas no gerenciamento ágil ou manter práticas de planejamento de negócios tradicionais.

Para abordar essa complexidade, a literatura propõe abordagens baseadas em múltiplas dimensões concorrentes. O primeiro modelo de destaque nessa visão é o conceito *Continuous** (lê-se *Continuous Star*), proposto por Fitzgerald e Stöl [5]. Os autores observaram que o aspecto contínuo da engenharia de software não se restringe à integração de código

ou à implantação de artefatos em produção, constituindo um fluxo no qual quatro áreas se alinham de forma coordenada: negócios, desenvolvimento, operações e inovação. A partir dessa premissa, introduziram dois conceitos: o *BizDev*, referente ao alinhamento contínuo entre negócios e desenvolvimento, com vistas a planejar o produto em conformidade com o mercado; e o *DevOps*, caracterizado pela integração contínua entre as equipes de desenvolvimento e de operações de infraestrutura. Na prática da ECS, o *software* deixa de ser planejado de forma isolada e única para ser entregue meses depois; em vez disso, o planejamento, a segurança, a implantação e o *feedback* operacional operam em ciclos contínuos, em que o uso real do sistema retroalimenta diretamente o planejamento estratégico.

Enquanto o *Continuous** representa a filosofia de integração contínua das atividades organizacionais, uma abordagem prática para mensurar esse alinhamento surge com o *Eye of CSE* [8]. Trata-se de um modelo visual que organiza as práticas em uma lista de verificação composta por 33 elementos estruturados em nove categorias principais: Negócios, Gerência de Software, Desenvolvimento, Equipe, Solução Técnica, Qualidade, Conhecimento, Usuário/Cliente e Operação. Diferentemente da linearidade do StH, o *Eye of CSE* assume que a organização pode evoluir simultaneamente em múltiplos elementos de categorias distintas. Desse modo, a maturidade de engenharia contínua não é representada por um degrau único, mas sim pelo preenchimento de áreas que demandam atenção concorrente.

O instrumento Zeppelin baseia-se nas dimensões descritas no *Eye of CSE*, permitindo avaliar, por meio de um questionário integrado, práticas de negócios, qualidade e infraestrutura simultaneamente, resultando em um diagnóstico multidimensional da organização. Na estrutura conceitual descrita por Barcellos [3], essas dimensões diferenciam os processos centrais (*core*), voltados diretamente ao desenvolvimento e à entrega de *software* (Desenvolvimento Ágil, Integração Contínua e Implantação Contínua), dos processos transversais (*cross-cutting*), como a Medição de Software Contínua e a Gerência de Conhecimento Contínuo, que permeiam todas as atividades do ciclo de vida e sustentam as decisões estratégicas da organização.

2.5 A Escala de Maturidade

A fim de mensurar o grau de conformidade de cada prática nas organizações avaliadas, o Zeppelin adota uma escala de avaliação ordinal de cinco níveis [2]. Cada nível indica a maturidade de aplicação da prática:

- **Não Adotado (0%)**: A prática inexiste ou não é realizada de forma alguma.
- **Abandonado (10%)**: A prática chegou a ser iniciada ou testada no passado, mas foi descontinuada devido a barreiras internas, técnicas ou de processo.
- **Nível de Projeto/Produto (30%)**: A prática é realizada de forma isolada por projetos específicos ou times pontuais, sem haver uma padronização na empresa.
- **Nível de Processo (60%)**: A prática é formalizada, documentada e definida como um padrão a ser seguido por toda a organização.

- **Institucionalizado (100%)**: A prática está integrada à rotina e à cultura organizacional e é executada de forma sistemática por todas as equipes.

Cada percentual é utilizado como peso para calcular uma média ponderada do grau de adoção das práticas por categoria do *Eye of CSE* e estágios do StH. A dissertação de Araújo [1] refinou a distribuição desses pesos para que as notas gerais calculadas representem a maturidade da empresa com precisão, servindo de base para as recomendações de evolução.

3 Métodos

Este trabalho tem como objetivo criar uma interface digital para a ferramenta de autodiagnóstico Zeppelin, no âmbito de uma plataforma completa de avaliação de maturidade em práticas de engenharia contínua de software.

A metodologia adotada para o desenvolvimento do módulo de questionário e para a validação deste trabalho baseou-se, inicialmente, em uma revisão bibliográfica sobre ECS. O estudo dos artigos originais do Zeppelin [2] e da dissertação de calibragem de pesos de Araújo [1] fundamentou as atividades de todo o projeto, com o objetivo de compreender o histórico da ferramenta Zeppelin e os conceitos teóricos relacionados, o que permitiu uma análise crítica do desenvolvimento.

O processo de desenvolvimento seguiu um ciclo iterativo de prototipagem, utilizando a ferramenta Figma para o *design* das interfaces e estruturando uma arquitetura composta de *front-end* e *back-end*.

O *frontend* foi desenvolvido em React (versão 18) com a ferramenta de *build* Vite, enquanto o *backend* utilizou o *framework* Django (versão 4) integrado ao Django REST Framework. O desenvolvimento partiu de uma estrutura de *backend* pré-existente, desenvolvida pelo grupo de pesquisa de Santos Júnior *et al.* [2], voltada à gestão de questionários, sendo necessárias adaptações para expor novos *endpoints* e suportar as interações com a interface de usuário.

No *frontend*, as atividades concentraram-se na criação de toda a interface e dos fluxos para o preenchimento de questionários pelos usuários, indo desde telas gerais de interação (UI), com navegação dinâmica em abas correspondentes aos degraus do modelo StH, até funcionalidades para melhora de experiência do usuário (UX), como no salvamento automático de respostas de forma incremental para evitar perda de dados ou na validação de progresso para controle de avanço entre as abas.

Para testes e garantia de qualidade, o foco esteve na validação automatizada das interações do frontend, onde a lógica de preenchimento e acessibilidade do questionário está concentrada. Foi utilizado o *framework* `Vitest` com a `React Testing Library` para testar elementos da interface, incluindo tanto testes unitários quanto testes de fluxos de usuário. A execução local de todos os testes e análises de código foi unificada no comando `make verify`, que executa os `linters` e os testes automatizados antes de cada `commit` no repositório.

A estratégia de implantação baseia-se em um fluxo (*pipeline*) de entrega contínua, utilizando GitHub Actions, cujo ambiente de homologação consiste em uma máquina virtual disponibilizada na nuvem computacional do Instituto de Computação da UNICAMP, co-

nectada ao repositório do projeto no `Github`[1]. Este processo é iniciado a cada atualização da *branch* principal do repositório. Ainda, é mediado por contêineres Docker, nos quais a *action* de implantação atualiza o código a partir do repositório, reconstrói as imagens dos contêineres, executa as migrações do banco de dados e preenche o banco de perguntas do questionário.

4 Proposta de Solução

Para criar uma interface digital para a ferramenta de autodiagnóstico Zeppelin, foi elaborada uma arquitetura segmentada, composta pelos contêineres *front-end* e *back-end*, além da camada de dados (Figura 1). A solução completa possui diferentes funcionalidades voltadas ao fluxo completo da jornada dos usuários, sendo capaz de permitir que os usuários façam autenticação, cadastro da sua empresa atual, preenchimento dos questionários, verificação dos resultados individuais ou comparação com respostas anteriores e, por fim, para administradores, a capacidade de comparar resultados entre diferentes empresas.

Para isso, foram estabelecidos módulos distintos, tendo como foco deste projeto a criação do **Módulo de Questionário**. As seções a seguir detalharão a arquitetura geral e aprofundarão este módulo.

A arquitetura construída com base em contêineres, com separação em módulos, visa garantir a integridade dos dados e a manutenibilidade da solução. Ela foi desenhada para isolar contextos e dados, permitindo que múltiplos usuários interajam com o questionário simultaneamente, sem risco de exposição de informações sensíveis de diferentes organizações.

[1] O código-fonte do projeto está disponível em: `https://github.com/brenorobazza/zeppelin` Acesso em: jun. 2026.

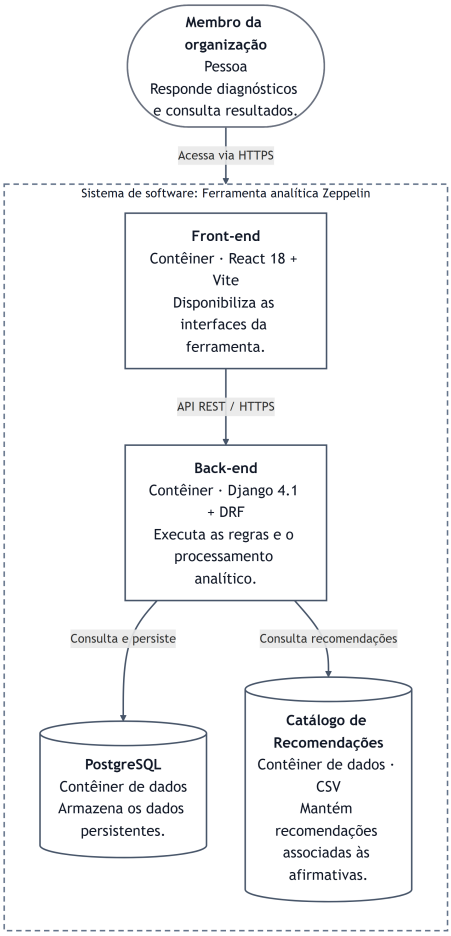

Figura 1: Diagrama de contêineres da solução Zeppelin.

Os componentes internos de cada contêiner são detalhados nas Figuras 2 e 3. Em ambos os diagramas, os componentes destacados em azul correspondem às partes desenvolvidas neste trabalho.

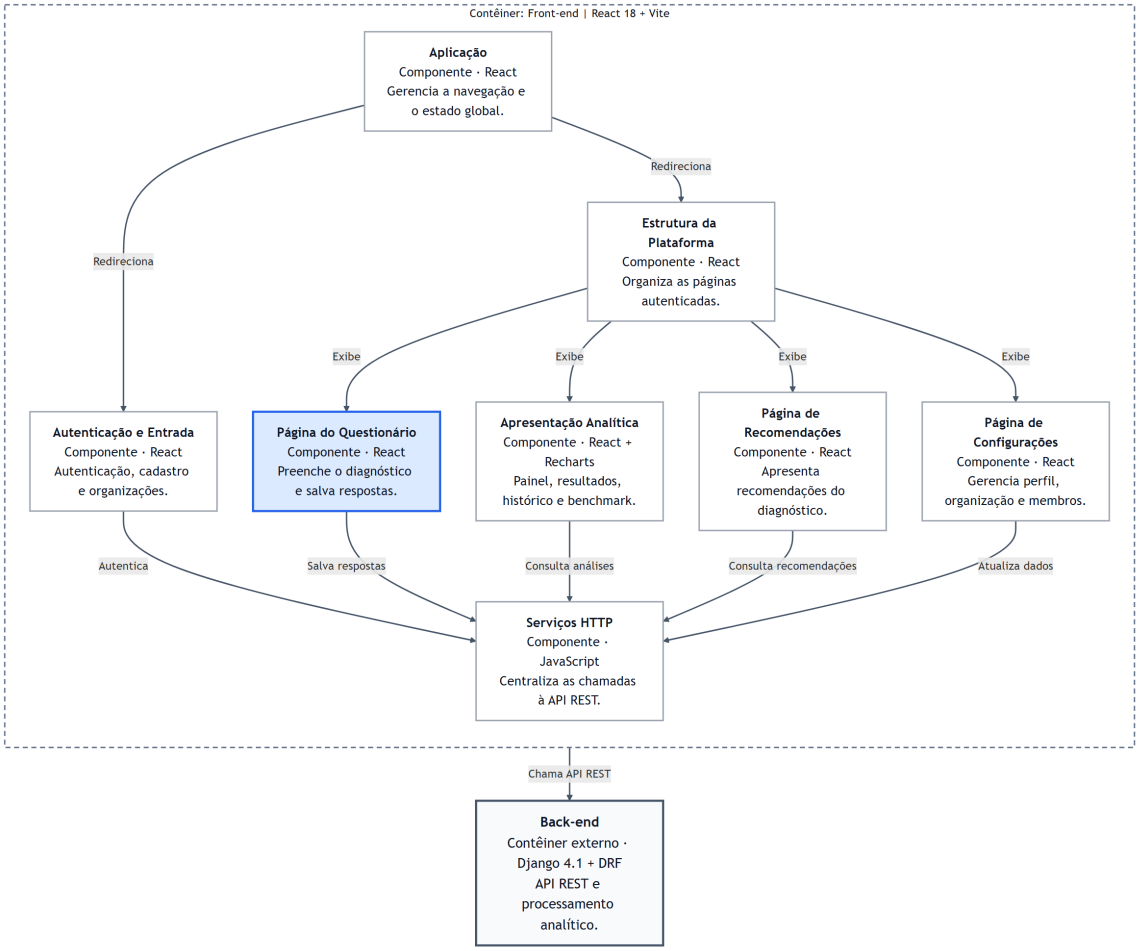

Figura 2: Componentes do contêiner *front-end*. Em azul, a *AssessmentPage*, componente do Módulo de Questionário desenvolvido neste projeto.

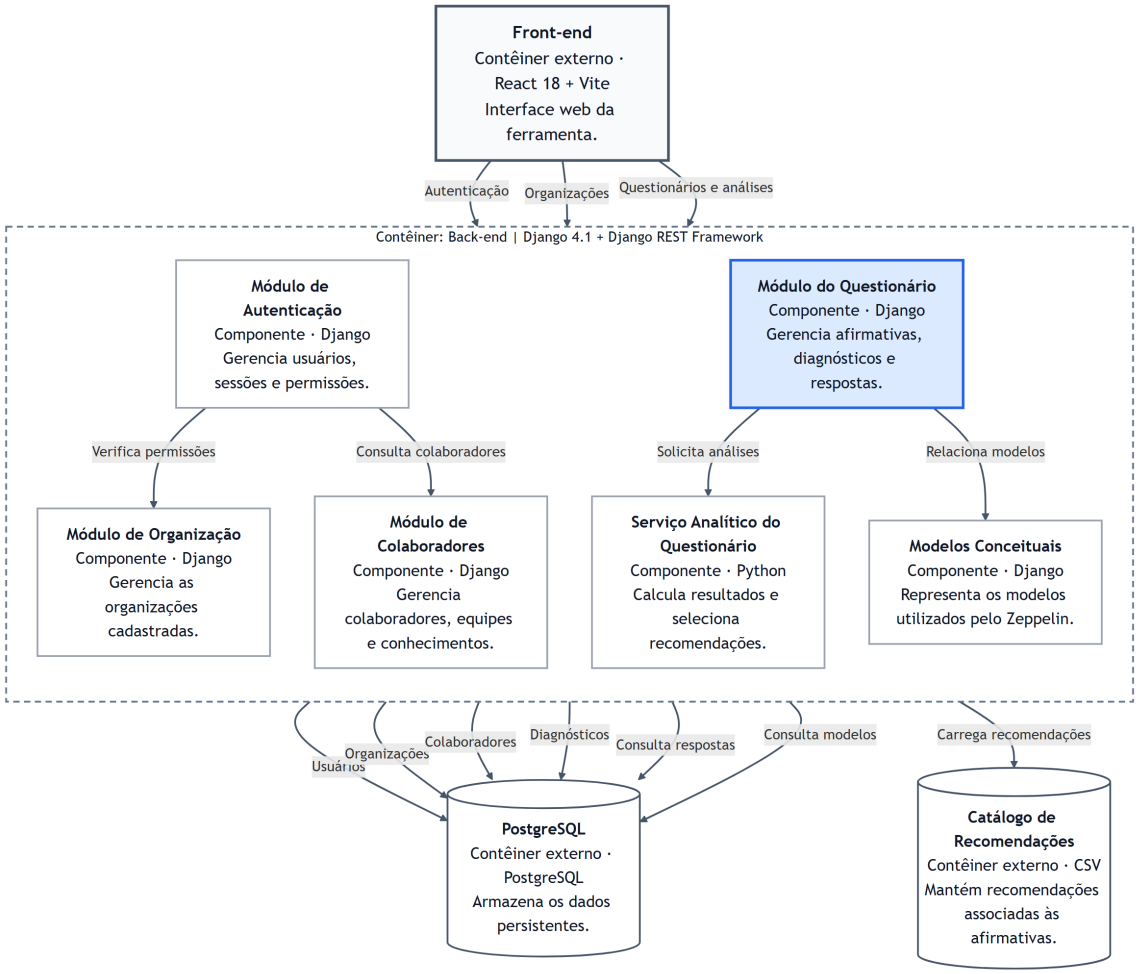

Figura 3: Componentes do contêiner *back-end*. Em azul, o Módulo do Questionário (`apps.questionnaire`), desenvolvido neste projeto.

4.1 Seleção de Tecnologias

A solução utiliza uma arquitetura cliente-servidor. O *front-end* é construído em JavaScript com o *framework* React, tratando-se de uma *Single Page Application* (SPA), que se comunica com o *back-end* por meio de uma API REST desenvolvida em Python com o *framework* Django. Toda a comunicação do *front-end* com o *back-end* exige o protocolo OAuth2 para garantir de isolamento dos dados, sendo necessários a emissão e o uso de *tokens* Bearer para validar o acesso aos *endpoints*. O banco de dados relacional (PostgreSQL) permite a persistência da solução.

4.2 Modelagem Conceitual

O modelo de dados (Figura 4) é composto de múltiplas entidades, como a entidade `Employee`, que vincula um usuário conectado a uma organização. O processo de preenchimento é organizado de maneira hierárquica: a entidade `Cycle` representa um período temporal de avaliação; a entidade `Assessment` instancia o questionário preenchido; a entidade `Response` armazena um valor numérico da resposta selecionada para cada pergunta, que, por sua vez, é representada pela entidade `Statement`.

Esta arquitetura traduz os conceitos do instrumento Zeppelin para o banco de dados relacional. Cada Statement armazenado no banco está associado a uma categoria do *Eye of CSE* e agrupado em degraus do modelo StH. Essa organização assegura a integridade das análises multidimensionais de maturidade, além de manter a fundamentação teórica atrelada a cada item do questionário e, portanto, às análises realizadas com base nele.

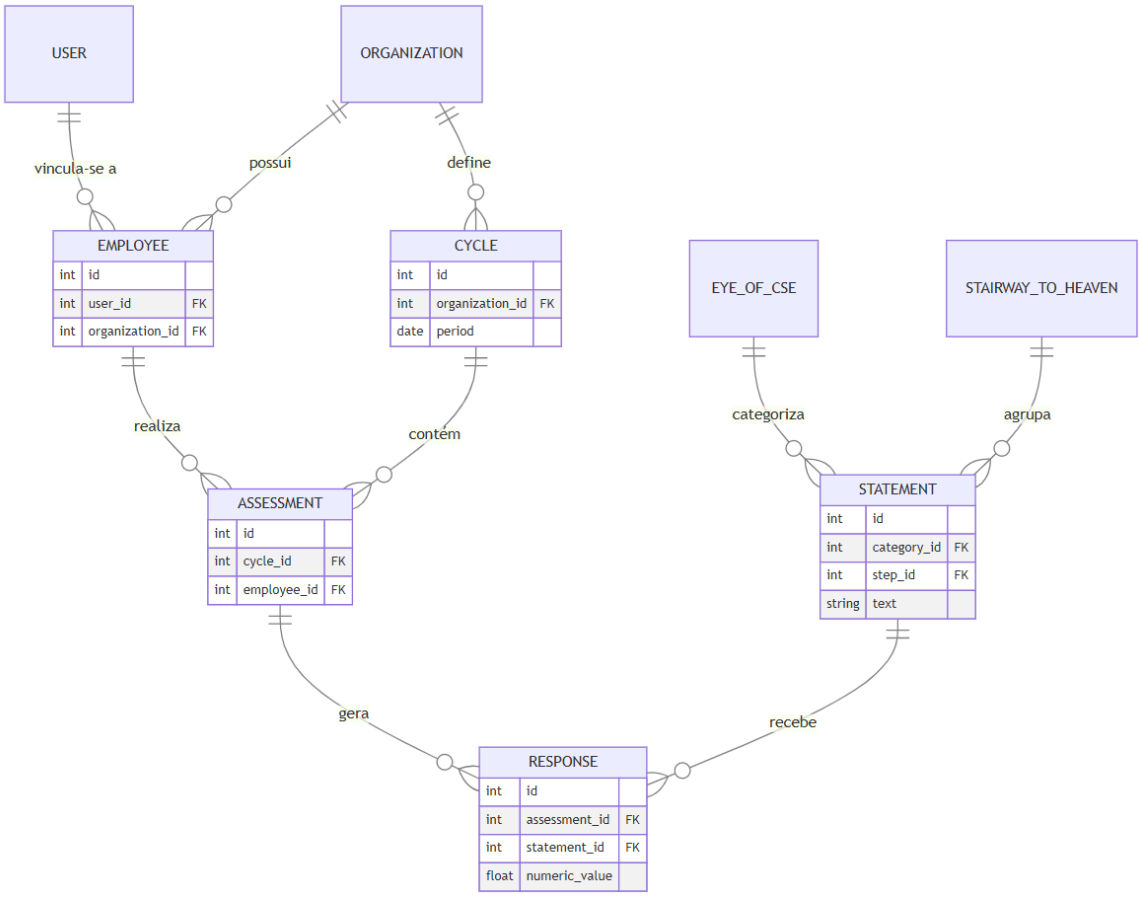

Figura 4: Diagrama relacional.

4.3 Implementação da Ferramenta

Para inserir as perguntas no banco de dados, foi estruturado um fluxo automatizado que serve como fonte de verdade. O arquivo `backend/fixtures/questions.json` é carregado pelo script `load_initial_questionnaire_data` durante a inicialização do contêiner, garantindo o alinhamento com a estrutura teórica.

A autenticação (criação de conta e acesso via OAuth2 — ver Figura 5) é o mecanismo técnico de acesso ao sistema.

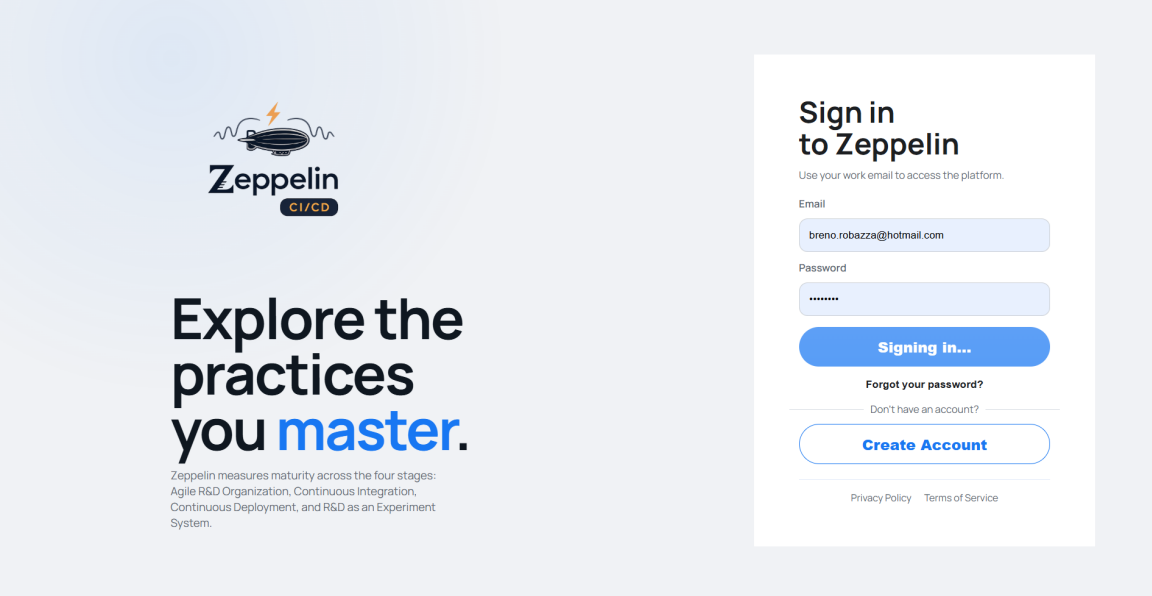

Figura 5: Página de Autenticação.

A interface (Figura 6) implementa uma navegação baseada em abas, nas quais cada aba corresponde a um degrau do modelo StH. Foi implementada uma validação de respostas que permite a transição livre entre as abas, porém exige que a avaliação em cada degrau ocorra sequencialmente, bloqueando o avanço para a questão seguinte caso a atual não tenha sido respondida.

O usuário pode selecionar a qual empresa, dentre as empresas associadas ao seu perfil, deseja responder ao questionário por meio da interface de listagem, que filtra todos os questionários já respondidos ou em processo de preenchimento (ver Figura 7).

Foi adotada uma estratégia de persistência incremental para prevenir a perda de progresso: a cada alternativa selecionada, a resposta é enviada imediatamente à API e, ao finalizar, uma variável de recarga no front-end é acionada para sincronizar a árvore de estado do React. O componente da tabela de histórico depende de uma propriedade chamada `refreshKey`. Quando o questionário é encerrado, essa chave é atualizada, atuando como gatilho que força a tabela a buscar os dados mais recentes no *back-end*. Dessa forma, o progresso recém-salvo é refletido imediatamente na listagem sem necessidade de recarregar

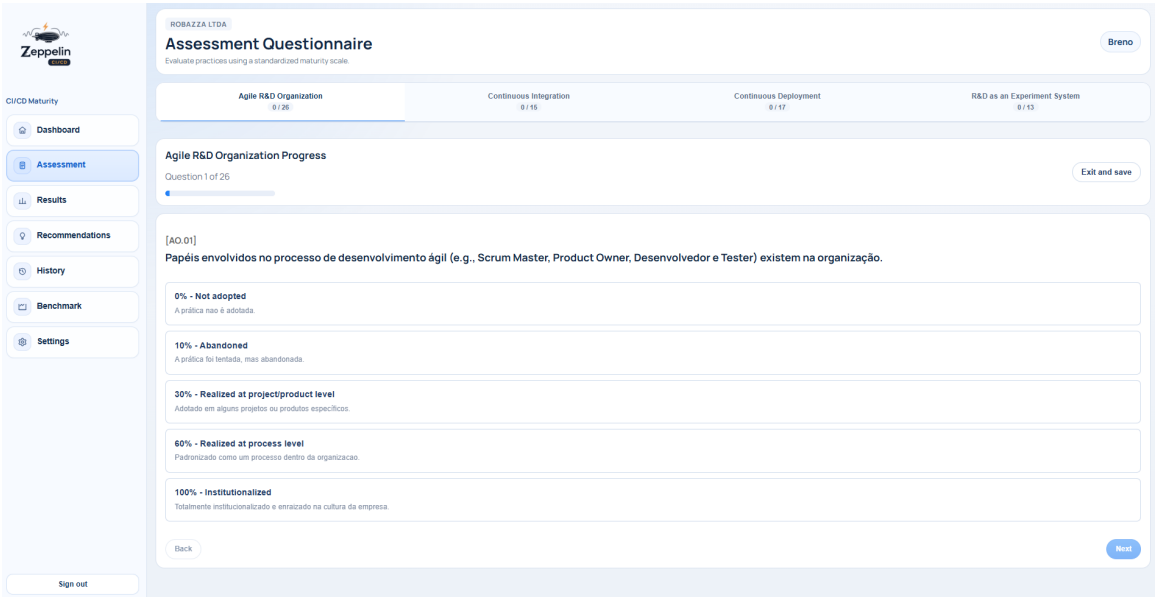

Figura 6: Página de Questionário - Preenchimento: Cada degrau é representado como uma aba do questionário. O usuário pode preencher as questões de um degrau de maneira progressiva, sem capacidade de pular questões, mas pode saltar para as questões de outro degrau.

a página.

Já, para garantir de isolamento de contexto, foi implementado um mecanismo de isolamento *multi-tenant* que permite selecionar, por meio de seletores na interface, qual organização será associada. Assim que o usuário inicia uma avaliação, os seletores globais localizados no cabeçalho (que normalmente permitiriam trocar de empresa) são desabilitados automaticamente e substituídos por valores estáticos (ver Figura 8). Isso impede mudanças de escopo durante a sessão, assegurando que o usuário não consiga enviar respostas da avaliação atual para a base de dados de uma empresa distinta.

E, no âmbito de experiência do usuário, foi adicionado suporte a eventos de teclado. As opções da escala ordinal (0%, 10%, 30%, 60%, 100%) foram mapeadas, respectivamente, para as teclas numéricas de 1 a 5, e a ação de submissão do formulário atual para a tecla *Enter*.

5 Avaliação da Ferramenta

A avaliação do módulo de questionário dividiu-se em duas frentes principais: a verificação técnica de *software*, garantindo o correto funcionamento e o isolamento da interface, e a validação conceitual do instrumento, submetendo a plataforma ao crivo de especialistas da área.

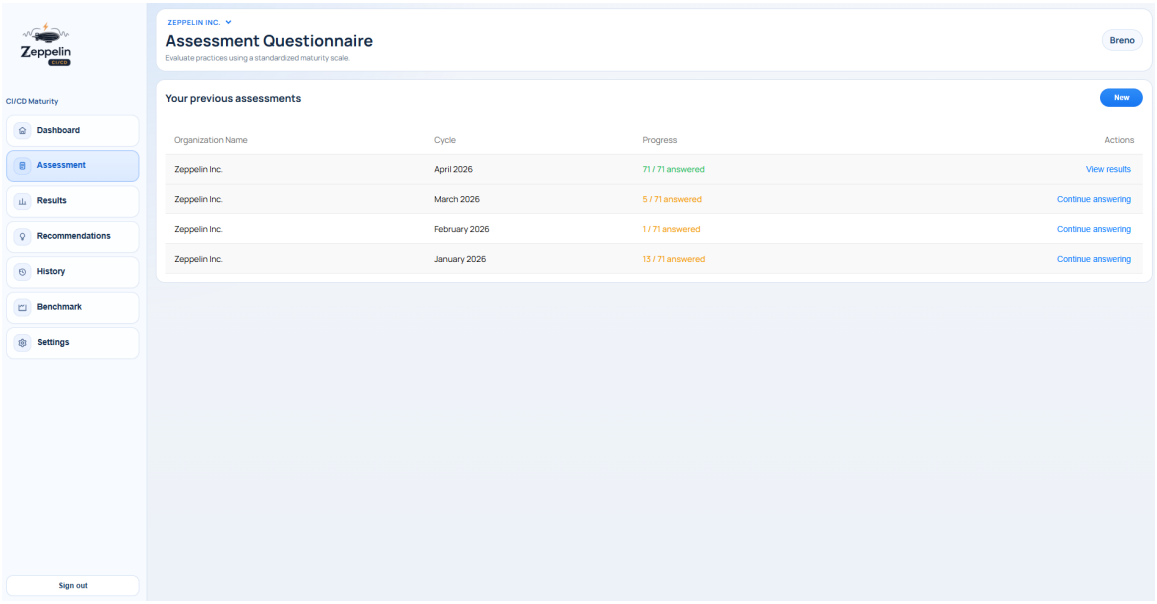

Figura 7: Página de Questionário - Listagem: O usuário pode visualizar todos os questionários já preenchidos ou em processo de preenchimento, filtrando por sua empresa escolhida. Clicar em um questionário já preenchido leva aos seus resultados, enquanto clicar em um formulário em processo de preenchimento leva à primeira questão não respondida entre os estágios.

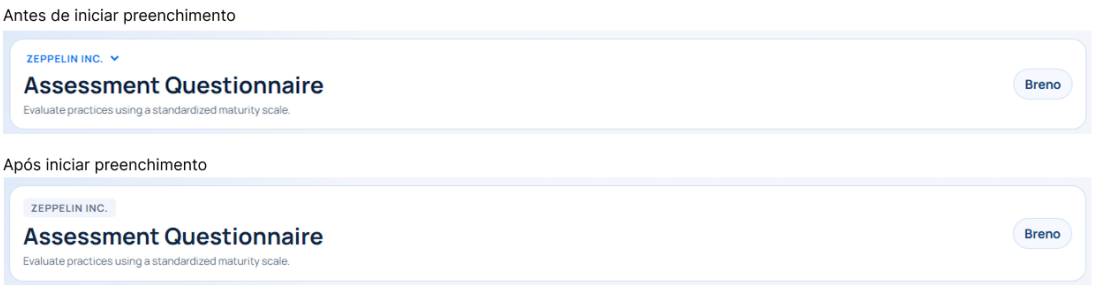

Figura 8: Isolamento *Multi-Tenant*. O seletor no cabeçalho torna-se desabilitado e exibe um valor estático para evitar alterações acidentais de escopo durante a avaliação.

5.1 Verificação Técnica Automatizada

O foco da validação técnica foi o comportamento do *front-end* e as integrações da interface. Para garantir a confiabilidade da navegação e das restrições de integridade dos dados, foi implementado um fluxo de testes utilizando `Vitest` em conjunto com a `React Testing`

`Library`.

Os testes incluem dezenove cenários automatizados que cobrem os componentes visuais, a lógica de roteamento em *loop* do questionário e as políticas de *layout*. Entre os testes principais, verificou-se o comportamento do isolamento de contexto (garantindo que os seletores sejam bloqueados durante uma avaliação ativa) e as restrições da interface de abas (impedindo que o usuário avance degraus do modelo StH sem responder às questões obrigatórias). Estes testes automatizados foram integrados ao fluxo de CI do repositório, no qual o comando `make verify` executa as validações antes de qualquer envio de código para o ambiente de produção.

5.2 Validação Conceitual com Especialistas

Além do processo de testes, a solução passou por uma avaliação qualitativa e conceitual junto aos autores originais do instrumento. A interface digital do questionário foi apresentada e testada por Paulo Sérgio dos Santos Júnior (criador original do Zeppelin) e por Aline Dias de Araújo (autora do refinamento e da calibração dos pesos).

O objetivo foi averiguar se a conversão do questionário para o formato *web* preservou a integridade do instrumento original. Durante a validação, os pesquisadores avaliaram a clareza da interface e a adesão à metodologia científica da ferramenta, atestando a aplicação correta das teorias do *Stairway to Heaven* e do *Eye of CSE*.

6 Conclusões e Trabalhos Futuros

Este trabalho atingiu o objetivo de construir uma interface *web* para o instrumento de autodiagnóstico Zeppelin, transformando a base teórica de avaliação em uma plataforma funcional. O desenvolvimento do módulo Questionário entregou uma interface que respeita a teoria original do instrumento e a fundamentação teórica da Engenharia Contínua de Software, mapeando os degraus do *Stairway to Heaven* e as categorias do *Eye of CSE* diretamente para a arquitetura de banco de dados e para a navegação *front-end*.

Do ponto de vista técnico, a implementação de restrições de integridade, como o isolamento de contexto e o salvamento incremental, mostrou-se eficaz para assegurar a integridade na coleta de dados. A aprovação da interface pelos idealizadores originais do instrumento atesta o sucesso da conversão do modelo para o meio digital sem perda de rigor metodológico.

Trabalhos futuros desta solução incluem, por exemplo, a integração nativa e contínua deste módulo com os motores de Recomendação e Análise de Dados, permitindo que as respostas aqui coletadas gerem *dashboards* e planos de ação automatizados, além da aplicação da plataforma em cenários reais, ampliando a base empírica sobre a adoção de práticas contínuas em organizações de *software*.

Referências

[1] ARAUJO, Aline Dias de. *Diagnosis and Recommendations for CI/CD Practices based on the Zeppelin Instrument*. 2025. Dissertação (Mestrado em Ciência da Computação) – Instituto de Computação, Universidade Estadual de Campinas (UNICAMP), Campinas, SP, 2025.

[2] SANTOS JÚNIOR, Paulo Sérgio dos; BARCELLOS, Monalessa Perini; RUY, Fabiano Borges. Tell me: Am I going to Heaven? A Diagnosis Instrument of Continuous Software Engineering Practices Adoption. In: *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering (EASE '21)*. ACM, 2021. p. 30–39. DOI: `https://doi.org/10.1145/3463274.3463324`.

[3] BARCELLOS, Monalessa Perini. Towards a Framework for Continuous Software Engineering. In: *Proceedings of the 34th Brazilian Symposium on Software Engineering (SBES 2020)*. ACM, 2020. p. 626–631. DOI: `https://doi.org/10.1145/3422392.3422469`.

[4] DE FRANÇA, Breno Bernard N.; JERONIMO, Helvio; TRAVASSOS, Guilherme Horta. Characterizing DevOps by hearing multiple voices. In *Proceedings of the XXX Brazilian Symposium on Software Engineering (SBES 2016)*. ACM, 2016, September. pp. 53-62.

[5] FITZGERALD, Brian; STOL, Klaas-Jan. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, v. 123, p. 176–189, 2017. DOI: `https://doi.org/10.1016/j.jss.2015.06.063`.

[6] OLSSON, Helena Holmström; ALAHYARI, Hiva; BOSCH, Jan. Climbing the "Stairway to Heaven" – A Multiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In: *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2012. p. 392–399.

[7] PFLEEGER, Shari L; Atlee, J. M. Software engineering: theory and practice. *Pearson Education*. India. 2010.

[8] JOHANSSEN, Jan Ole; KLEEBAUM, Anja; PAECH, Barbara; BRUEGGE, Bernd. Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners. *Journal of Software: Evolution and Process*, v. 31, n. 5, p. e2169, 2019. DOI: `https://doi.org/10.1002/smr.2169`.