

PREDIÇÃO DE MUDANÇAS DE SOFTWARE IMPACTANTES NO DESEMPENHO

Aline Rodrigues dos Santos, Paulo Roberto Farah

INTRODUÇÃO

Mudanças em software podem ocorrer devido a diversos fatores, sendo as alterações no código-fonte inevitáveis durante a evolução dos sistemas. À medida que os sistemas evoluem, tornam-se maiores e mais complexos, exigindo que os desenvolvedores gerenciem essas mudanças de forma estruturada e consciente de seus impactos (Elish e Al-Khiaty, 2013; Abbas, Albalooshi e Hammad, 2020). Diante disso, a predição de mudanças surge como uma alternativa para identificar elementos de código propensos a modificações (Malhotra e Khanna, 2019). Quando aplicada ao contexto de desempenho, essa abordagem permite identificar preventivamente alterações com alto risco de impactar a eficiência do sistema, viabilizando a priorização de testes e refatorações antes que eventuais degradações atinjam o ambiente de produção, uma vez que problemas de performance levam mais tempo para serem reportados e corrigidos em comparação com falhas funcionais (Jin et al., 2012; Han e Yu, 2016). Diante desse cenário, este trabalho tem como objetivo principal apresentar um estudo exploratório sobre os mecanismos de predição de mudanças aplicados à performance de software, detalhando os desafios práticos e as limitações técnicas encontradas durante a condução da pesquisa, bem como os direcionamentos para contornar tais restrições em trabalhos futuros.

DESENVOLVIMENTO

A metodologia foi estruturada em quatro etapas principais: (1) levantamento da literatura e seleção de projetos para análise; (2) preparação do ambiente experimental e execução inicial com a ferramenta PerfoRT; (3) identificação e caracterização das limitações técnicas; e (4) implementação de melhorias na ferramenta para viabilizar a coleta de dados. Cada etapa foi conduzida de forma iterativa, com ajustes contínuos no processo, com o objetivo de realizar uma análise comparativa entre o PerfoRT e outras ferramentas de análise.

Durante a execução da ferramenta, primeiramente foram selecionados projetos de baixa escala, nos quais a configuração da ferramenta e o poder computacional não seriam um empecilho, como o Commons-CSV (APACHE SOFTWARE FOUNDATION, 2024) e Tomcat (APACHE SOFTWARE FOUNDATION, 2025). Para esse sistema, a ferramenta realizou a análise corretamente, com exceção de um erro no banco de dados, no qual a ferramenta não populava a tabela de *methods*, que era de suma importância para a análise. Para tratar o erro, a ferramenta precisou ser adaptada ao projeto, identificando-se, assim, mais uma limitação. Dessa forma, finalizou-se a etapa de ensaio e foram selecionados projetos maiores para dar continuidade ao estudo.

Os projetos foram selecionados segundo os seguintes critérios: desenvolvimento em linguagem Java, utilização do Apache Maven para gerenciamento da compilação e conformidade com boas práticas arquitetônicas. Esses critérios garantiriam a reprodutibilidade dos testes e facilitariam a extração de métricas estáveis. Foram analisados projetos como PetClinic (SPRING PROJECTS, 2026) e TeaStore (DESK, 2018), classificados como “Códigos Ideais” pela sustentabilidade de seus padrões.

A ferramenta PerfoRT (Farah, 2023) instrumenta o código durante a execução de testes JUnit, rastreando eventos da JVM e mensurando o tempo de execução dos métodos, o consumo de CPU e

a alocação de memória. Inicialmente, foi configurada com Programação Orientada a Aspectos (AOP), por meio do agente AspectJ (-javaagent), e Java Flight Recorder (JFR), persistindo os dados em banco relacional. Essa configuração foi testada em diferentes cenários para validar a viabilidade da coleta de dados em larga escala.

Contudo, durante a etapa de extração massiva de dados com o PerfoRT, foram identificadas limitações relacionadas à compatibilidade do software e à capacidade de processamento dos computadores do laboratório, afetando a escalabilidade da ferramenta. Dentre elas:

- **Incompatibilidade de inicialização:** conflito entre AspectJ e JFR durante a inicialização da JVM.
- **Falhas silenciosas de persistência:** inconsistências no mapeamento entre dados estáticos e dinâmicos.
- **Gestão inadequada de contexto:** dependências de configurações locais (.env) que falhavam em diferentes ambientes.
- **Rigidez de caminhos:** problemas na construção do classpath ao transitar entre repositórios.

Paralelamente à configuração da ferramenta, foram criados e treinados modelos de aprendizado de máquina, utilizando Máquina de Vetores de Suporte (SVM) e Regressão Logística, com dados compostos por métricas de software. Embora os modelos tenham demonstrado potencial promissor, sua integração com o PerfoRT ainda era limitada, dado que o pipeline de coleta de dados ainda não estava disponível.

Diante dessas limitações identificadas, surgiram propostas de melhorias iterativas para o PerfoRT, consistindo na refatoração da arquitetura de instrumentação, na correção sistemática de cada limitação listada e na generalização dos mecanismos de localização de arquivos e configurações.

RESULTADOS

Como resultado da pesquisa, os achados foram divididos em dois segmentos: propostas de refatoração para o PerfoRT, baseadas na análise sistemática de suas limitações, e modelos de Machine Learning desenvolvidos para a predição de mudanças de desempenho.

A partir da análise sistemática das falhas identificadas durante a execução do PerfoRT, foram delineadas estratégias de refatoração e melhoria arquitetural. Para cada limitação caracterizada, apresenta-se a proposta técnica de resolução:

O PerfoRT apresentou incompatibilidade na inicialização da JVM causada pelo uso simultâneo do agente AspectJ (-javaagent) com Java Flight Recorder (JFR). Para viabilizar a execução dos testes, o JFR precisou ser temporariamente desativado, limitando a coleta de métricas de perfilamento. A refatoração deve implementar uma abordagem de instrumentação embutida, migrando da dependência de agentes externos (AspectJ) para código de instrumentação integrado diretamente ao bytecode dos métodos durante a compilação ou carregamento das classes. Dessa forma, eliminaria-se a necessidade de múltiplos agentes simultaneamente, permitindo que o JFR funcionasse sem conflitos e ampliando a cobertura de métricas capturadas. Como solução, propõe-se a migração da abordagem via agente externo para uma abordagem embutida, injetando código de perfilamento diretamente no bytecode das classes durante a compilação ou carregamento, eliminando a interferência entre agentes e reabilitando o uso do JFR em conjunto com as ferramentas.

A persistência dos dados dinâmicos apresentou forte vulnerabilidade a falhas relacionais. O agente interceptador (Tracer) condiciona a inserção de registros na tabela *methods* à existência prévia de um mapeamento estático da classe na tabela *files* para o hash de commit correspondente. Havendo divergência no padrão do qualified name ou falha no povoamento anterior pelo analisador estático, a requisição de busca retorna um identificador inválido (-1), resultando no descarte silencioso do evento sem o disparo de exceções tratáveis. Para solucionar o problema, caso o mapeamento prévio da classe não seja localizado na tabela *files*, o agente deverá realizar o registro dinâmico do arquivo/classe no banco ou gerar um log explícito de depuração, garantindo que o evento de execução não seja descartado silenciosamente e fornecendo rastreabilidade das exceções.

A concatenação iterativa de caminhos relativos ao manipular repositórios corrompia a construção do classpath necessário para a localização das classes compiladas (*target/classes*). Esse problema era agravado pela dependência de scripts executáveis intermediários (.sh) para disparar a suíte de testes JUnit, uma abstração que falha ao interagir com repositórios limpos sem esses scripts adaptativos integrados. Como solução, propõe-se a eliminação da dependência de scripts .sh externos, substituindo-os pela invocação direta e nativa da ferramenta de compilação (Maven/JUnit) por meio de código em Go (exec.CommandContext).

Paralelamente às melhorias realizadas no PerfoRT, foi desenvolvido o componente de Machine Learning para a predição de mudanças impactantes no desempenho. A pipeline de aprendizado foi estruturada em três etapas: Extração de features do PerfoRT conforme descrito em Farah e Vergilio (2023), que instrumentam o código durante testes JUnit, tratamento de dados, compreendendo normalização, redução de base de dados e balanceamento das classes utilizando SMOTE e, por fim, o treinamento e avaliação de modelos.

Foram desenvolvidos dois modelos de classificação: Máquinas de Vetores de Suporte (SVM) e Regressão Logística. Contudo, a integração desses modelos ao fluxo completo do PerfoRT ainda depende da estabilização da etapa de coleta e persistência das métricas dinâmicas, uma vez que os problemas técnicos identificados impossibilitaram a consolidação de um dataset completo para o treinamento com dados reais do projeto.

CONSIDERAÇÕES FINAIS

Diante do conjunto de limitações técnicas mapeadas na ferramenta PerfoRT e, consequentemente, da falta de *features*, não foi possível conduzir o estudo experimental conforme originalmente planejado, nem entregar o *pipeline* final de predição de mudanças em sua totalidade. A incapacidade de extrair e persistir métricas dinâmicas de forma estável impediu a consolidação da base de dados necessária para a validação das hipóteses da pesquisa.

Apesar da impossibilidade de concluir a coleta de dados primária, o projeto trouxe contribuições relevantes ao diagnosticar minuciosamente os gargalos arquitetônicos e os erros de execução do PerfoRT. O mapeamento dessas falhas permitiu direcionar os esforços para a resolução de dívidas técnicas e para o estudo de trabalhos correlatos na literatura, pavimentando o caminho para a reformulação da ferramenta e para a publicação de resultados parciais em eventos científicos futuros.

Palavras-chave: Predição de Mudança; Performance de Software; PerfoRT; Métricas Dinâmicas.

REFERÊNCIAS BIBLIOGRÁFICAS

FARAH, P. R.; VERGILIO, S. R. PerfoRT: A Tool for Software Performance Regression. In: ACM/SPEC INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING COMPANION, 2023, Coimbra, Portugal. Proceedings... New York: ACM, 2023. p. 111–114. DOI: <https://doi.org/10.1145/3578245.3585025>.

FARAH, Paulo Roberto. **PerfoRT**: Performance Testing and Reporting Tool. GitHub, 2026. Disponível em: <https://github.com/paulorfarah/PerfoRT>. Acesso em: 17 ago. 2026.

SPRING PROJECTS. **Spring PetClinic Sample Application**. Versão 2.0. San Francisco: GitHub, 2026. Disponível em: <https://github.com/spring-projects/spring-petclinic>. Acesso em: 17 ago. 2026.

DESKTOP SOFTWARE ENGINEERING RESEARCH GROUP (DESK). **TeaStore**: A micro-service reference and benchmark application. Versão 1.4.x. GitHub, 2018. Disponível em: <https://github.com/DescartesResearch/TeaStore>. Acesso em: 17 ago. 2026.

MALHOTRA, R.; KHANNA, M. Software change prediction: A systematic review and future guidelines. *e-Informatica Software Engineering Journal*, v. 13, n. 1, p. 227–259, 2019.

R. Abbas, F. A. Albaloooshi, and M. Hammad. Software change proneness prediction using machine learning. In 2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT), pages 1–7. IEEE, 2020.

APACHE SOFTWARE FOUNDATION. **Apache Commons CSV**: Apache Software Foundation, 2024. Disponível em: <https://github.com/apache/commons-csv>. Acesso em: 19 ago. 2026.

APACHE SOFTWARE FOUNDATION. **Apache Tomcat**: Apache Software Foundation, 2025. Disponível em: <https://tomcat.apache.org/>. Acesso em: 19 ago. 2026.

M. O. Elish and M. Al-Rahman Al-Khiaty. A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software. *Journal of Software: Evolution and Process*, 25(5):407–437, 2013.

G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu. Understanding and detecting real-world performance bugs. *ACM SIGPLAN Notices*, 47(6):77–88, 2012.

X. Han and T. Yu. An empirical study on performance bugs for highly configurable software systems. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '16, New York, NY, USA, 2016. Association for Computing Machinery.