Extração e Visualização de Métricas Evolutivas em LPS: um Estudo de Caso com o Eclipse IDE

Kevin G. Strey
Dept. de Engenharia de Software
Universidade do Estado de Santa Catarina (Udesc)
Ibirama, SC, Brasil
kevin.strey@edu.udesc.br

Willian D. F. Mendonça
Faculdade Donaduzzi
Toledo, PR, Brasil
williandouglasferrari@gmail.com

Paulo Roberto Farah
Dept. de Engenharia de Software
Universidade do Estado de Santa Catarina (Udesc)
Ibirama, SC, Brasil
paulo.farah@udesc.br

Resumo

Detectar padrões de mudança durante a evolução de sistemas de Linha de Produtos de Software (LPS) é um desafio, especialmente quando as funcionalidades são desenvolvidas em paralelo em múltiplos repositórios. Para compreender melhor essa evolução, analisamos métricas estruturais e evolutivas do projeto EclipseIDE, observando como o projeto evolui ao longo do tempo. A análise foi realizada em três etapas. Primeiramente, mapeou-se o repositório central que agrega os repositórios distribuídos das funcionalidades. Em seguida, extraímos métricas estruturais e evolutivas em nível de método para todas as funcionalidades. Por fim, as funcionalidades foram categorizadas com base em seu tamanho absoluto, tamanho médio dos métodos e grau de mudança ao longo do tempo, revelando a formação de três grupos: funcionalidades enxutas e estáveis, médias e gigantes voláteis. Também foram identificados padrões nas métricas BOM, CSB e LOC, indicando eventos de refatoração, como a adição de novos métodos, a refatoração estrutural do sistema e a consolidação de repositórios. Essa coleta de métricas e a identificação de grupos que sofrem mais modificações contribuem para evitar vieses de dados na previsão de mudanças e para apoiar a priorização de testes em sistemas de LPS.

Palavras-chave

Linha de Produto de Software, Mineração de Repositórios de Software, Visualização de Software

1 Introdução

A Linha de Produto de Software (LPS) é uma abordagem de reuso sistemático em que uma família de produtos relacionados são desenvolvidos compartilhando funcionalidades comuns. Essas funcionalidades atendem às necessidades específicas de um determinado segmento de mercado e permite customizar os produtos para diferentes clientes. Para isso, a LPS é desenvolvida a partir ativos centrais compartilhados, com extensões e variações organizadas em unidades chamadas *features*, de acordo com uma abordagem previamente estabelecida [5].

O código-fonte associado às *features* de uma LPS normalmente se fragmenta ao longo de diversos arquivos ou repositórios, dificultando a identificação e o agrupamento dos elementos de código pertencentes a cada *feature*. Essa característica torna desafiadora a extração de métricas evolutivas no nível de *feature*, especialmente quando se considera a evolução de sistemas com arquitetura de LPS. Portanto, o principal problema que justifica o presente trabalho é a dificuldade de se extrair métricas evolutivas de sistemas LPS.

Diante disso, esse trabalho apresenta uma ferramenta capaz de construir um mapeamento temporal entre releases de repositórios, extrair suas métricas estruturais e evolutivas no escopo de método e gerar gráficos para visualizar a evolução das features LPS.

Essa característica de temporalidade permite que métricas coletadas a partir do código fonte desses repositórios possam ser utilizadas para analisar a evolução das características do sistema como um todo, por meio de diferentes visualizações.

A extração das métricas evolutivas permite, por exemplo, adotar estratégias como a priorização ou seleção para testes de regressão mais eficientes. Além disso, permite o treinamento de modelos de predição de defeitos, por meio da identificação das features que mais sofreram modificações ao longo do tempo.

A LPS analisada no presente trabalho é o Eclipse IDE, um editor de código modular baseado em plug-ins, frequentemente utilizado como estudo de caso na literatura de LPS [2, 3]. Cada módulo do projeto é desenvolvido em um repositório independente no GitHub e referenciado por um repositório central, responsável por mapear quais features e versões irão compor sua release final [6].

A visualização da evolução de software emprega recursos visuais para representar e compreender as mudanças ocorridas ao longo do tempo, sendo uma prática consolidada na literatura de Engenharia de Software [19]. Diante disso, para a geração dos gráficos, utilizamos os conjuntos de métricas evolutivas [7] e as métricas estruturais CK [4]. Juntas, elas são utilizadas, por exemplo, em pesquisas de predição de mudanças e qualidade de software, pois refletem as refatorações feitas na estrutura do sistema ao longo do tempo e as modificações realizadas entre releases de um sistema analisado [1, 8].

Para nortear as análises das métricas evolutivas e estruturais, elencaram-se as seguintes perguntas de pesquisa:

RQ1 Como o ecossistema Eclipse evoluiu globalmente ao longo de suas releases em termos de tamanho (LOC total) e Média de LOC por método ativo?

RQ2 É possível agrupar as diferentes features do sistema em perfis evolutivos distintos (clusters) baseados em suas métricas estruturais e evolutivas?

RQ3 Como refatorações impactam a trajetória das métricas de features individuais?

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados, contextualizando o estudo na literatura existente. A Seção 3 detalha a metodologia, descrevendo a arquitetura da ferramenta, o processo de mapeamento temporal e a coleta das métricas. Na Seção 4, os resultados são apresentados e discutidos, respondendo às perguntas de pesquisa formuladas. A Seção 5 traz as ameaças à validade do trabalho. Por fim, a Seção 6 expõe as conclusões derivadas da análise e aponta

direcionamentos para trabalhos futuros, seguida das informações sobre a disponibilidade dos artefatos.

2 Trabalhos Relacionados

A literatura relacionada evidencia que a proposta deste trabalho situa-se na interseção de três linhas de pesquisa consolidadas em Engenharia de Software. A primeira linha investiga a evolução de LPS, concentrando-se principalmente em processos de reengenharia [9, 15], refatoração, gerenciamento da variabilidade[25], sincronização entre artefatos e propagação de mudanças ao longo do ciclo de vida da linha de produto [10]. Embora alguns desses trabalhos empreguem métricas quantitativas para apoiar avaliações específicas [20], o foco predominante está na evolução dos artefatos e dos modelos de variabilidade, e não na caracterização quantitativa da evolução histórica da LPS a partir do seu histórico de desenvolvimento.

A segunda linha de pesquisa abrange a mineração de repositórios de software (MSR) e a análise de métricas evolutivas, áreas já bem fundamentadas por mapeamentos extensos na literatura [13, 24]. Dentro desse contexto, uma vertente expressiva utiliza dados extraídos do histórico de desenvolvimento primordialmente para estudos empíricos de qualidade de software, como a predição de defeitos e a identificação de gargalos no processo de desenvolvimento [22]. Outra vertente busca caracterizar a própria natureza da evolução do software ao longo do tempo, seja através do desenvolvimento de ferramentas para o rastreamento longitudinal de métricas de componentes [21], seja pelo uso de *Visual Analytics* para apoiar a compreensão de manutenções acumuladas [11].

A terceira linha compreende estudos empíricos longitudinais conduzidos no ecossistema de software do Eclipse. Trabalhos nessa vertente investigam, por exemplo, a evolução do tamanho e da complexidade do sistema e de seus subprojetos para validar leis de evolução de software [17], bem como a extração e a análise de padrões de reuso de features ao longo de múltiplas releases [23]. Há, inclusive, iniciativas que modelam o ecossistema do Eclipse explicitamente sob a perspectiva de uma LPS com o objetivo de prever arquivos propensos a falhas utilizando métricas de mudança [14]. Contudo, mesmo nesses cenários de interseção, observa-se uma escassez de abordagens que integrem métricas estruturais e evolutivas de forma combinada para caracterizar sistematicamente a evolução das features ao longo das releases do projeto.

3 Metodologia

Para responder às perguntas de pesquisa estabelecidas e viabilizar a análise histórica da plataforma Eclipse, a metodologia deste trabalho foi estruturada para superar o desafio da fragmentação do código das features em múltiplos repositórios. Dessa forma, para possibilitar a investigação da evolução global (RQ1), o agrupamento por perfis evolutivos (RQ2) e a avaliação do impacto de grandes refatorações (RQ3), desenvolveu-se uma abordagem de extração de dados, descrita a seguir.

A abordagem de coleta proposta é organizada em um pipeline de dois estágios, implementado em duas ferramentas complementares: o Repository_mapper (Python) e o EvoMetrics Collector (Java). O primeiro estágio é responsável por construir um mapeamento temporal entre as releases do repositório central Eclipse Simrel[1] (Simultaneous Release) e os commits dos repositórios individuais de cada feature. O segundo estágio consome esse mapeamento para realizar a extração evolutiva de métricas no escopo de método sobre o código-fonte de cada módulo. As métricas coletadas para análise foram:

Lines Of Code (LOC): Quantidade total de linhas de código, removendo comentários e linhas em branco.

Média de LOC por método ativo: Razão entre LOC total e a quantidade de métodos ativos (LOC do método > 0) de cada módulo. Trata-se de uma métrica estrutural simplificada, usada para medir o tamanho médio de cada método em cada módulo do sistema.

Birth Of Method (BOM): Indica a release do módulo em que o método apareceu pela primeira vez. Utilizado para fazer a contagem de métodos nascidos por release.

Changes since the birth (CSB): É uma métrica que representa o volume total acumulado de modificações obtido através da contagem de linhas inseridas e linhas removidas entre a versão antiga e a versão atual do método em uma release específica. Além dessas métricas utilizadas no presente trabalho, a ferramenta coleta um total de 12 métricas evolutivas e 29 métricas estruturais CK, detalhadas no Readme[2] no repositório da ferramenta.

3.1 Repositório-Base e Linha do Tempo da Coleta

No primeiro estágio de coleta se utiliza o repositório central simrel.build[3], mantido pela Eclipse Foundation. Este repositório atua como um repositório de integração, onde cada tag git do simrel.build corresponde a uma release oficial da plataforma Eclipse (por exemplo, JunoSR0[4], Neon.3[5], 2024-03[6]) e registra, no commit associado a essa tag, a configuração de agregação que especifica quais versões de cada módulo compõem aquela release.

As tags do simrel.build são enumeradas em ordem cronológica com base na data do commit que cada tag referencia, estabelecendo assim uma sequência temporal completa das releases da plataforma, desde as releases nomeadas com codinomes (Juno, Kepler, Luna, Mars, Neon, Oxygen, Photon), até as releases com nomenclatura baseada em data (2018-09…2026-06). Essa sequência cronológica constitui o eixo temporal sobre o qual toda a coleta é organizada.

No repostitório central, as features e dependências são organizadas através de arquivos `.b3aggrcon`, antes da release Oxygen, e em arquivos `.aggrcon` a partir dessa release. Contudo, para delimitar o escopo de coleta e análise das features, utilizou-se como base o feature-model proposto no trabalho de Johansen et al. [12], do qual foram removidos os repositórios Jubula e RSE, pois seu sistema de controle de versão não é o GitHub. Desse modo, as features coletadas foram: JDT, PDE, CDT, GEF, EMF, BIRT, DataTools, EclipseLink, EGit, GMF Runtime, Mylyn, RAP, PTP, Scout, WebTools, Window Builder, m2e(MAVEN), CVS e SVN(Subversive).

Os repositórios das features são clonados localmente a partir do GitHub. A ferramenta mantém um mapa de URLs, informado

[1] https://github.com/eclipse-simrel/simrel.build
[2] https://github.com/KevinStrey/feature-model-eclipse/blob/main/README.md
[3] https://github.com/eclipse-simrel/simrel.build
[4] https://github.com/eclipse-simrel/simrel.build/tree/JunoSR0
[5] https://github.com/eclipse-simrel/simrel.build/tree/Neon.3
[6] https://github.com/eclipse-simrel/simrel.build/tree/2024-03

manualmente, que associa cada nome de repositório à sua URL de clonagem. Quando o EvoMetrics Collector (Java) identifica que um repositório necessário não existe localmente, ele realiza a clonagem automática via git clone antes de iniciar a extração.

3.2 Processo de Mapeamento Temporal

O mapeamento entre as releases do simrel.build e os commits dos repositórios individuais é realizado através de uma estratégia baseada em proximidade temporal. O processo funciona da seguinte forma:

Resolução da data da release: Para cada tag do simrel.build, a ferramenta identifica o commit associado à tag e extrai sua data. Esta data representa o momento em que a configuração daquela release foi consolidada.

Busca do commit mais próximo por data: Para cada repositório de módulo, a ferramenta executa uma busca no histórico Git do repositório local pelo commit cuja data é anterior ou igual à data do commit da release no simrel.build. Essa abordagem de mapeamento temporal garante que o commit selecionado represente o estado do módulo no momento (ou imediatamente antes) da consolidação da release.

Resolução da tag (versão): Uma vez identificado o commit do módulo, a ferramenta tenta associá-lo a uma tag git do repositório do módulo. Se o commit estiver associado a uma tag (direta ou ancestralmente), o nome da tag é registrado como a versão do módulo naquela release. Caso não exista uma tag associável, o campo *version* é registrado como *null*, mas o commit ainda é considerado válido.

Geração do artefato de saída: Para cada release processada, a ferramenta gera um arquivo JSON contendo: o nome da release, a data do commit no simrel.build, o hash do commit do simrel.build, e um dicionário de mapeamentos onde cada chave é o nome de exibição do módulo e o valor contém a versão (tag), o status (SUCCESS ou NOT FOUND), o hash do commit mapeado e o nome do repositório. Por exemplo, para a release 2024-03[7], o mapeamento do módulo JDT registra o commit c648d350[8]... com a tag I20240308-1800 no repositório eclipse.jdt.core.

3.3 Consumo do Mapeamento pela Ferramenta de Extração

O segundo estágio da coleta é realizado pelo módulo Java (EvoMetrics Collector), que consome os arquivos JSON gerados na etapa de mapeamento. A classe MappingLoader carrega todos os arquivos de mapeamento do diretório de saída e os deserializa em objetos ReleaseMapping, cada um contendo o nome da release, a data e o dicionário de FeatureMapping (com campos version, status, commit e repository).

A partir da leitura desses mapeamentos, a ferramenta precisa reorganizar os dados para facilitar a análise histórica. Originalmente, os arquivos JSON gerados na etapa anterior são organizados por Release, isto é, cada arquivo de Release contém o estado de várias Features naquele momento (Release → Feature → Commit).

Para analisar a evolução de uma feature específica ao longo do tempo, o módulo Java inverte essa hierarquia, transformando-a em uma estrutura centrada na Feature: Feature → (Release → Commit).

Dessa forma, o pipeline completo estabelece as seguintes etapas de mapeamento e coleta: tag do repositório central (simrel.build) → data do commit da release → commit mais recente do módulo até aquela data → tag/versão do módulo (quando disponível) → commits intermediários entre releases → extração de métricas evolutivas. A Figura 1 apresenta um diagrama com o fluxo de coleta completo realizado pela ferramenta.

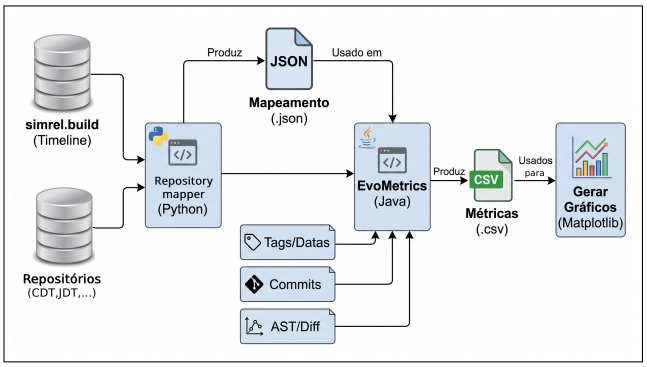

Figura 1: Diagrama do fluxo de coleta da ferramenta

4 Resultados

Esta seção apresenta as análises das métricas coletadas para as 61 releases do projeto Eclipse, com o objetivo de compreender a evolução do projeto e responder às perguntas de pesquisa formuladas.

4.1 RQ1 - Como o ecossistema Eclipse evoluiu globalmente ao longo de suas releases em termos de tamanho (LOC total) e Média de LOC por método ativo?

A Figura 2 apresenta a evolução do tamanho de todas as features do ecossistema em termos de LOC. Isso permite observar o crescimento de algumas features do sistema, como EclipseLink, DataTools e JDT. Mas também há casos de redução não expressiva da quantidade de linhas, como a BIRT, CDT CVS e Window Builder. As demais features apresentaram baixa variabilidade, apresentando LOC estável ao longo do tempo.

Considerando todas as features e releases do período analisado, e computando exclusivamente os métodos ativos (LOC > 0) nas versões em que as features já possuíam código implementado, a média global de tamanho por método manteve-se em 10,47 linhas, com desvio padrão de 2,83 (variando entre o mínimo de 7,16 e o máximo de 19,85)[9].

[7] https://github.com/eclipse-simrel/simrel.build/commit/7409f370c1e29953a07a86faf4154f2ac38b1599

[8] https://github.com/eclipse-jdt/eclipse.jdt.core/commit/c648d35069922c6cebf7dbd773d7fe509f3c3afa

[9] O gráfico com todas as features pode ser visualizado em https://github.com/KevinStrey/feature-model-eclipse/blob/main/6-Analises/LOC_metodos/mean_loc_all_features.pdf

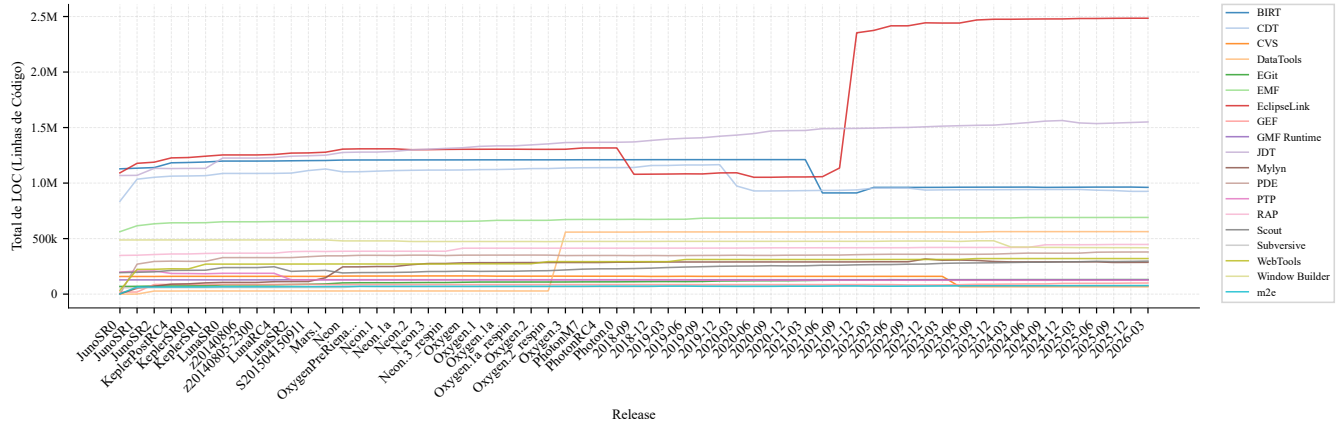

Figura 2: Evolução do Tamanho Total (LOC) para todas as features do ecossistema Eclipse.

Resposta à RQ1

Em sua maioria as features do ecossistema Eclipse foram desenvolvidas nas primeiras releases e se mantiveram estáveis ao longo do tempo. No entanto, os valores absolutos de LOC das features EclipseLink, JDT e DataTools aumentaram, indicando a inclusão de novas funcionalidades.

4.2 RQ2 - É possível agrupar as diferentes features do sistema em perfis evolutivos distintos (clusters) baseados em suas métricas estruturais e evolutivas?

Com base nas características individuais de LOC, Média de LOC por método ativo e CSB de cada módulo, aplicamos o algoritmo de clusterização K-Means para agrupar as features em perfis evolutivos distintos.

Para evitar viés de dimensionalidade e garantir que todas as métricas tivessem exatamente o mesmo peso durante o cálculo das distâncias Euclidianas, os dados brutos foram previamente submetidos a uma normalização Min-Max. Esse processo redimensionou os valores de cada uma das três métricas para um intervalo contínuo padronizado entre 0 e 1.

Para determinar o número ideal de agrupamentos para o algoritmo K-Means, aplicou-se o Método do Cotovelo. O método consistiu em calcular a Soma dos Quadrados Intra-Cluster para valores de K variando de 1 a 10. O ponto de inflexão da curva (cotovelo), onde os ganhos marginais de redução de variância passam a ser progressivamente menores, consolidou-se na região entre $K = 3$ (WCSS = 1.08) e $K = 4$ (WCSS = 0.70)[10].

Diante disso, o modelo de K-Means foi então configurado para particionar o espaço de atributos normalizados definindo $K = 3$, para criar perfis fáceis de compreender, formando três clusters de features bem delineados:

C1: Enxutas e Estáveis: Composto pela maioria das features (como PDE, Mylyn, WebTools e Scout), este grupo caracteriza-se por um tamanho reduzido de base de código e baixos níveis de alteração ao longo de sua vida útil.

C2: Médias: As features DataTools, Window Builder, EGit e EMF compõem este grupo intermediário. Apresentam tamanho moderado e taxas de modificações mais ativas.

C3: Gigantes Voláteis: Este grupo é composto pelos pilares da plataforma: JDT, CDT, BIRT e EclipseLink. São features com grande volume de código e que sofrem constantes refatorações gerando um alto número de modificações acumuladas (CSB).

A identificação desses perfis indica que a evolução de um sistema LPS não é uniforme e que diferentes features operam sob regimes de manutenção distintos. A Figura 3 mostra um gráfico de dispersão 3D que permite visualizar os clusters de features.

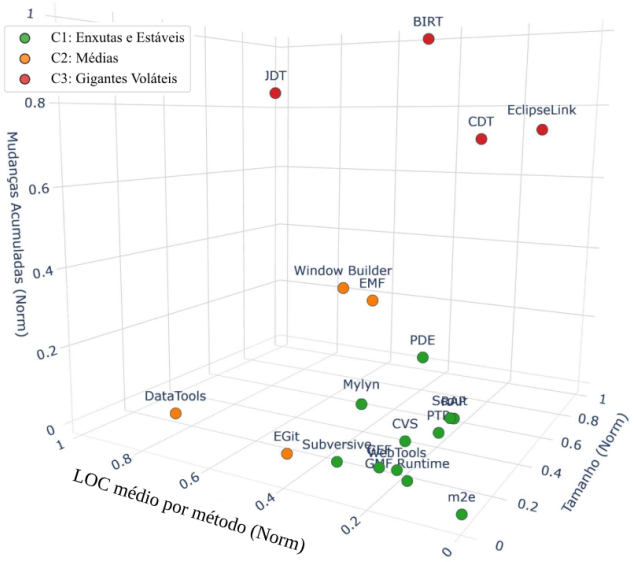

Figura 3: Gráfico de dispersão dos clusters (eixos normalizados)

[10]Os resultados estão no repositório https://github.com/KevinStrey/feature-model-eclipse/blob/main/5-Views/resultados/02_clusterizacao/elbow_method_with_values.png

Resposta à RQ2

A aplicação do algoritmo K-Means revelou a formação de três perfis evolutivos distintos: **C1 (Enxutas e Estáveis)**, com bases de código reduzidas e um valor baixo de LOC por método ativo; **C2 (Médias)**, com tamanho moderado e manutenção ativa; e **C3 (Gigantes Voláteis)**, caracterizadas por alto volume de código e frequentes refatorações estruturais.

4.3 RQ3 - Como grandes refatorações impactam a trajetória das métricas de features individuais?

Para compreender como grandes eventos de refatoração impactam a trajetória das métricas de software, selecionamos casos de estudo de um cluster evolutivo distinto para observação, permitindo identificar diferentes tipos de refatorações estruturais.

4.3.1 O caso das Gigantes Voláteis. A Figura 4 apresenta o CSB médio por método de todas as features do sistema. Observa-se que, na release 2019-12, o módulo *EclipseLink* sofre uma queda abrupta na sua média de CSB, chegando a zero. Para confirmar a anomalia, avaliou-se a métrica BOM. Como mostrado na Figura 5, há um pico massivo de aproximadamente 119 mil métodos nascidos na release 2019-12.

Contudo, ao cruzar esses dados com o tamanho absoluto do sistema (Figura 2), percebe-se que o LOC Total do EclipseLink em 2019-12 permaneceu constante. Essa assinatura com Pico de BOM, Queda de CSB e LOC Constante indica que não houve injeção de código novo, mas sim uma refatoração estrutural do sistema. Como a ferramenta de extração utiliza o caminho do arquivo como identificador, a migração em massa de pacotes[11] é interpretada como exclusão e nascimento simultâneo dos métodos, zerando o histórico (CSB) e inflando o nascimento (BOM). Como o código é essencialmente o mesmo, o LOC Total não sofre alteração.

Em contraste, na release 2021-12, o *EclipseLink* apresenta um novo pico de nascimentos (BOM), mas dessa vez acompanhado por um salto substancial no LOC Total. Essa segunda assinatura visual, com pico de BOM e LOC crescente, revela que houve uma adição de código novo, diferenciando-se completamente do evento de 2019-12.

4.3.2 O caso das Features Médias. Um evento de refatoração também ocorreu no grupo de features médias, especificamente com o módulo *DataTools*. Conforme ilustrado na Figura 5, durante a release *Oxygen.3*, houve um aumento notável na quantidade de novos métodos (BOM). Como consequência, o CSB (Figura 4) caiu para quase zero. No entanto, diferentemente do caso de refatoração pura em 2019-12 do *EclipseLink*, o tamanho absoluto do sistema (LOC Total) do *DataTools* aumentou consideravelmente nesse mesmo período (Figura 2).

A inspeção do histórico do repositório revela que essa assinatura gráfica com pico de BOM, LOC crescente e queda de CSB foi causada por uma consolidação de repositórios[12]. Anteriormente o DataTools era segmentado em múltiplos repositórios independentes hospedados na infraestrutura própria da Fundação Eclipse, mas no final de 2017 a equipe migrou seu sistema de compilação (para Maven/Tycho) e executou *merges* estruturais para unificar todos os subprojetos dentro de um único repositório público.

4.3.3 O caso das Features Enxutas e Estáveis. As features classificadas no cluster menor e mais estável (como Mylyn, Subversive, PDE, Scout e EGit) apresentam um padrão evolutivo em que seu LOC total permanece praticamente constante ao longo do tempo. Ao contrário das variações presenciadas nos módulos médios e grandes, os gráficos dessas features não apresentam grandes refatorações, refletindo bases de código maduras que exigem pouca intervenção.

A hipótese para essa estabilidade é justificada pelo domínio de aplicação em que operam. Diferentemente dos módulos centrais do ecossistema, como o JDT (Java) e o CDT (C/C++), que precisam sofrer alterações continuamente para suportar mudanças nas especificações de suas respectivas linguagens e novos recursos de compilação, as features menores geralmente atuam apenas como clientes de integração para protocolos externos (como repositórios SVN/Git ou rastreadores de bugs ALM). Uma vez que o núcleo da ferramenta é implementado, seu escopo dispensa refatorações grandes, mantendo a estabilidade da feature ao longo do tempo.

Resposta à RQ3

Refatorações que modificam o nome dos métodos ou que alteram sua localização diluem artificialmente o CSB e podem simular uma reinicialização no histórico de alteração dos métodos. Já quando há aumento de LOC e BOM, isso costuma indicar adição de métodos novos no sistema. Diante disso, a análise isolada de métricas pode ser enganosa e a comparação do comportamento de múltiplas métricas é essencial.

5 Ameaças à Validade

No que tange às ameaças à validade, a principal limitação deste estudo refere-se à validade externa, ou seja, à capacidade de generalização dos resultados. A análise restringiu-se ao ecossistema do Eclipse IDE, visto que foi o único projeto LPS em Java, com histórico de versionamento acessível e que atendeu aos requisitos da pesquisa. Além disso, a arquitetura atual da ferramenta de extração foi projetada especificamente para cenários em que o modelo de variabilidade ocorre através de features distribuídas em múltiplos repositórios. Consequentemente, a versão atual não suporta a análise de projetos desenvolvidos em outras linguagens de programação ou que adotem modelos de variabilidade baseados em anotações (como o uso de `#ifdef`).

Ademais, surge também a ameaça de construto, devido à técnica de extração baseada em proximidade temporal, ao invés do mapeamento estrito por arquivos de agregação (.aggrcon). Embora a extração baseada em arquivos de agregação represente o estado

[11] O commit no qual ocorre a maior refatoração é o https://github.com/eclipse-ee4j/eclipselink/commit/5a965ba8e80113a204ad19ef875e651a38af6e59, responsável por reestruturar milhares de arquivos para adequação ao projeto EE4J/Jakarta.

[12] Comprovado publicamente pelos logs de consolidação no repositório oficial (e.g., commit `503b5d22`) disponível em https://github.com/eclipse-datatools/datatools/commit/503b5d223de415f063b28bc0fbb93421e0012e77

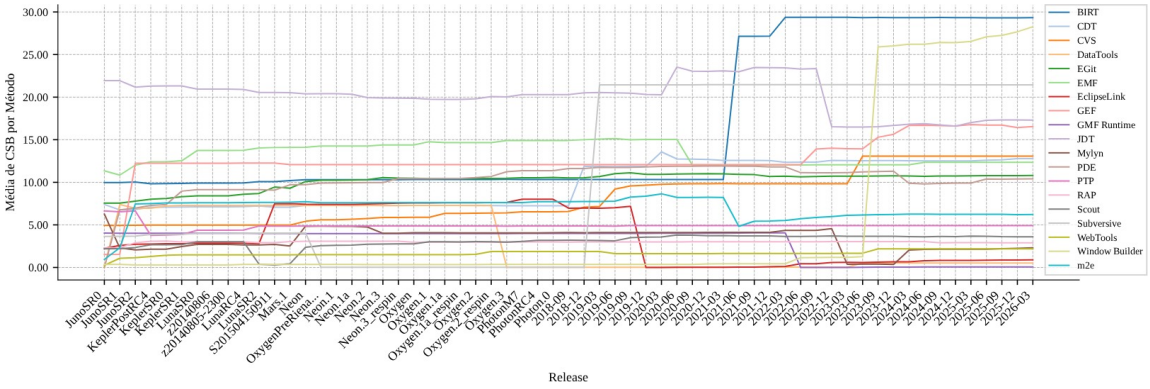

Figura 4: Média de mudanças desde o nascimento (CSB) por método para todas as features.

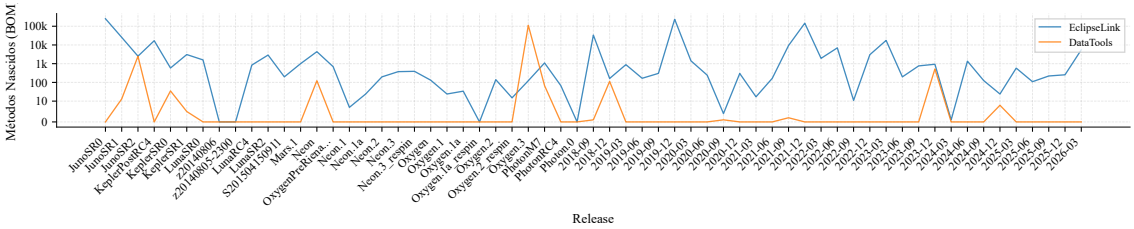

Figura 5: Contagem de métodos nascidos (BOM) para as features EclipseLink e DataTools (escala logarítmica).

exato do binário distribuído aos usuários, essa abordagem resultou na quebra de rastreabilidade e dados incompletos inviabilizando sua utilização. A abordagem temporal, por sua vez, pode introduzir um pequeno viés de sincronização, por conta de poder considerar o começo do desenvolvimento da feature da próxima release. No entanto, ela garantiu uma linha do tempo contínua que preserva as métricas do código e representa suficientemente bem os padrões de desenvolvimento e refatorações ocorridas na história do sistema.

Por fim, vale ressaltar que, embora as alterações nos caminhos dos métodos utilizados para sua identificação acabem inflando as métricas de BOM e CSB, este viés é suprimido no momento da agregação em nível de feature para a clusterização. Para isso, o cálculo do CSB foi modelado através da soma dos valores máximos cumulativos alcançados por todos os métodos associados a uma feature, por exemplo, se um método sofre 5 modificações antes de ter seu caminho alterado e mais 3 modificações em seu novo diretório, a agregação somará os picos das duas assinaturas totalizando 8 modificações, não prejudicando a validade da clusterização.

6 Conclusão e Trabalhos Futuros

Este trabalho apresentou uma nova ferramenta para a extração, visualização e análise de métricas evolutivas e estruturais em projetos de LPS, além de realizar um estudo de caso com o projeto Eclipse. A análise multidimensional através das métricas de Tamanho (LOC), Tamanho médio dos métodos ativos, Mudanças Acumuladas (CSB) e Nascimento de Métodos (BOM) permitiu uma nova visão da evolução do sistema, revelando padrões de manutenção e evolução nas diferentes features.

A partir disso, a capacidade de identificar padrões de manutenção e agregar as features de um sistema em clusters tem implicações importantes para pesquisadores em áreas como a predição de defeitos. Modelos preditivos de falhas baseiam-se fortemente no histórico de mudanças do código para identificar áreas de risco. As reestruturações de pacotes realizadas nas features médias e grandes, criam uma falsa percepção de estabilidade, pois zeram o CSB artificialmente, fazendo com que métodos antigos e complexos sejam classificados por um modelo de inteligência artificial como recém-criados ao invés de modificados. Isso pode enviesar drasticamente a predição de bugs caso eventos de refatoração estrutural não sejam discriminados das verdadeiras expansões funcionais [16, 18].

Sendo assim, conclui-se que utilizar métricas estruturais e evolutivas em conjunto, permite analisar o perfil evolutivo de cada feature e identificar padrões de refatoração que, se considerados isoladamente, podem conduzir a conclusões equivocadas em relação à natureza da modificação realizada.

Como trabalhos futuros, pode-se analisar a correlação com outras métricas estruturais do conjunto CK, tais acoplamento e coesão, para compreender como as mudanças na estrutura interna do código afetam diretamente a evolução dos diferentes clusters. Além disso, é possível expandir a ferramenta para realizar a extração de métricas evolutivas em repositórios centralizados e em diferentes mecanismos de implementação de variabilidade.

Disponibilidade de Artefatos

https://github.com/KevinStrey/feature-model-eclipse

Referências

[1] Mojeeb Al-Khiaty, Radwan Abdel-Aal, and Mahmoud Elish. 2017. Abductive network ensembles for improved prediction of future change-prone classes in object-oriented software. *International Arab Journal of Information Technology (IAJIT)* 14, 6 (2017).

[2] Yasser Ali Alshehri. 2020. Can We Predict the Change in Code in a Software Product Line Project? *Journal of Software Engineering and Applications* 13, 06 (2020), 91–103. doi:10.4236/jsea.2020.136007

[3] Yasser Ali Alshehri. 2022. Predicting change in newly created files in a software product line project. *Software: Practice and Experience* 52, 12 (2022), 2499–2512. doi:10.1002/spe.2911

[4] S. R. Chidamber and C. F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.* 20, 6 (June 1994), 476–493. doi:10.1109/32.295895

[5] P. Clements and L. Northrop. 2002. *Software Product Lines: Practices and Patterns*. Addison-Wesley. https://books.google.com.br/books?id=tHGFQgAACAAJ

[6] ECLIPSE FOUNDATION. 2026. Eclipse IDE: The Eclipse IDE repository. https://github.com/eclipse-ide

[7] Mahmoud O Elish and Mojeeb Al-Rahman Al-Khiaty. 2013. 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 (2013), 407–437.

[8] Paulo Roberto Farah, Rogério Silva, and Silvia Vergilio. 2023. An Assessment of Machine Learning Algorithms and Models for Prediction of Change-Prone Java Methods. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (Campo Grande, Brazil) *(SBES '23)*. Association for Computing Machinery, New York, NY, USA, 322–331. doi:10.1145/3613372.3613395

[9] David Faust and Chris Verhoef. 2003. Software product line migration and deployment. *Software: Practice and Experience* 33, 10 (2003), 933–955.

[10] Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sergio Soares, Fabiano Ferrari, Safoora Khan, Fernando Castor Filho, et al. 2008. Evolving software product lines with aspects: an empirical study on design stability. In *Proceedings of the 30th international conference on Software engineering*. 261–270.

[11] Antonio González-Torres, Francisco J García-Peñalvo, and Roberto Therón. 2013. Human–computer interaction in evolutionary visual software analytics. *Computers in Human Behavior* 29, 2 (2013), 486–495.

[12] Martin Fagereng Johansen, Øystein Haugen, Franck Fleurey, Erik Carlson, Jan Endresen, and Tormod Wien. 2012. A Technique for Agile and Automatic Interaction Testing for Product Lines. In *Testing Software and Systems*, Brian Nielsen and Carsten Weise (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 39–54.

[13] Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice* 19, 2 (2007), 77–131.

[14] Sandeep Krishnan, Chris Strasburg, Robyn R Lutz, and Katerina Goševa-Popstojanova. 2011. Are change metrics good predictors for an evolving software product line?. In *Proceedings of the 7th international conference on predictive models in software engineering*. 1–10.

[15] Miguel A Laguna and Yania Crespo. 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming* 78, 8 (2013), 1010–1034.

[16] Willian Mendonça, Wesley Assunção, and Silvia Vergilio. 2024. Feature-oriented Test Case Prioritization Strategies: An Evaluation for Highly Configurable Systems. *ACM* (09 2024), 72–83. doi:10.1145/3646548.3672592

[17] Tom Mens, Juan Fernández-Ramil, and Sylvain Degrandsart. 2008. The evolution of Eclipse. In *2008 IEEE International Conference on Software Maintenance*. IEEE, 386–395.

[18] Feifei Niu, Junqian Shao, Christoph Mayr-Dorn, Liguo Huang, Wesley K. G. Assunção, Chuanyi Li, Jidong Ge, and Alexander Egyed. 2025. Refactoring ≠ Bug-Inducing: Improving Defect Prediction with Code Change Tactics Analysis. arXiv:2507.19714 [cs.SE] https://arxiv.org/abs/2507.19714

[19] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonça, and Nico Zazworka. 2013. Software evolution visualization: A systematic mapping study. *Information and Software Technology* 55, 11 (2013), 1860–1883. doi:10.1016/j.infsof.2013.05.008

[20] Edson A Oliveira Junior, Itana Gimenes, Jose C Maldonado, Paulo C Masiero, and Leonor Barroca. 2013. Systematic evaluation of software product line architectures. *Journal of Universal Computer Science* 19, 1 (2013), 25–52.

[21] Joshua Oosterman, Warwick Irwin, and Neville Churcher. 2011. Evojava: A tool for measuring evolving software. In *Proceedings of the Thirty-Fourth Australasian Computer Science Conference-Volume 113*. 117–126.

[22] Dmytro Polishchuk. 2026. QModel: A Time-Aware GitHub Mining Framework for Empirical Software Quality Studies. *Research Square* (2026).

[23] Amjed Tahir, Sherlock A Licorish, and Stephen G MacDonell. 2017. Feature evolution and reuse-an exploratory study of eclipse. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 582–587.

[24] Andy Zaidman, Martin Pinzger, and Arie van Deursen. 2010. Software Evolution.

[25] Bo Zhang and Martin Becker. 2012. Code-based variability model extraction for software product line improvement. In *Proceedings of the 16th International Software Product Line Conference-Volume 2*. 91–98.