UNIVERSIDADE FEDERAL DO CEARÁ

***CAMPUS* SOBRAL**

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

MESTRADO ACADÊMICO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

AUGUSTO BRAZ DE SOUSA

UMA VALIDAÇÃO DAS LEIS DE LEHMAN: UMA INVESTIGAÇÃO QUANTITATIVA NO DESENVOLVIMENTO DE SOFTWARE MODERNO

SOBRAL

2024

AUGUSTO BRAZ DE SOUSA

UMA VALIDAÇÃO DAS LEIS DE LEHMAN: UMA INVESTIGAÇÃO QUANTITATIVA NO DESENVOLVIMENTO DE SOFTWARE MODERNO

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia Elétrica e de Computação do Programa de Pós-Graduação em Engenharia Elétrica e de Computação do *CAMPUS* SOBRAL da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia Engenharia Elétrica e de Computação. Área de Concentração: Sistemas de Informação

Orientador: Prof. Dr. Fischer Jonatas Ferreira

SOBRAL

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S696v Sousa, Augusto Braz de.
Uma validação das Leis de Lehman: uma investigação quantitativa no desenvolvimento de software moderno / Augusto Braz de Sousa. – 2024.
48 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Campus de Sobral, Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Sobral, 2024.
Orientação: Prof. Dr. Fischer Jonatas Ferreira.

1. Leis de Lehman. 2. Evolução de software. 3. GQM. 4. Diretrizes práticas. 5. Qualidade de software. I. Título.

CDD 621.3

AUGUSTO BRAZ DE SOUSA

UMA VALIDAÇÃO DAS LEIS DE LEHMAN: UMA INVESTIGAÇÃO QUANTITATIVA NO DESENVOLVIMENTO DE SOFTWARE MODERNO

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia Elétrica e de Computação do Programa de Pós-Graduação em Engenharia Elétrica e de Computação do *CAMPUS* SOBRAL da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia Engenharia Elétrica e de Computação. Área de Concentração: Sistemas de Informação

Aprovada em: 23 de Julho de 2024

BANCA EXAMINADORA

Prof. Dr. Fischer Jonatas Ferreira (Orientador)
Universidade Federal de Itajubá (UNIFEI)

Prof. Dr. Iális Cavalcante de Paula Júnior
Universidade Federal do Ceará (UFC)

Prof. Dr. Antonio Emerson Barros Tomaz
Universidade Federal do Ceará (UFC)

Prof. Dr. Eduardo Jabbur Machado
Universidade Federal de Itajubá (UNIFEI)

Dedico este trabalho a todos aqueles que, em meio às adversidades, nunca deixaram de acreditar no poder transformador da educação e às mentes curiosas que se recusam a aceitar os limites impostos, encontrando na busca pelo conhecimento um caminho para a liberdade e a realização.

AGRADECIMENTOS

A Deus, por me conceder saúde, força e sabedoria para enfrentar todos os desafios ao longo desta jornada acadêmica.

Aos meus pais, Maria Elizete Braz de Sousa e Tarcísio Cavalcante de Sousa, pelo amor, apoio incondicional e ensinamentos que sempre me guiaram.

Ao Prof. Dr. Fischer Jônatas Ferreira, meu orientador, pela paciência, orientação, e por compartilhar seu vasto conhecimento. Sua dedicação e apoio foram fundamentais para a realização deste trabalho.

Agradeço a todos os professores do Programa de Pós-graduação em Engenharia Elétrica e de Computação pela excelência no ensino e pelas contribuições durante todo o período de estudos. Seus ensinamentos foram essenciais para o meu crescimento acadêmico e profissional.

E aos meus irmãos de comunidade do Caminho Neocatecumenal, pelo apoio espiritual, amizade e pelo encorajamento, que me deram forças para continuar e perseverar.

“A fé e a ciência não estão em oposição, mas devem caminhar juntas, cada uma iluminando a outra.”

(Papa Francisco)

RESUMO

Este estudo investiga a aplicabilidade e relevância das Leis de Lehman no contexto atual de desenvolvimento de *software*, utilizando uma abordagem quantitativa para analisar três sistemas modernos: Dropwizard, K-9 Mail e LanguageTool. Através do *framework Goal Question Metric* (GQM), a pesquisa valida empiricamente as Leis de Lehman, que descrevem princípios fundamentais da evolução do *software*, como mudança contínua, aumento da complexidade e conservação da estabilidade organizacional. O estudo é dividido em duas etapas principais. A primeira etapa envolve a análise das métricas de evolução dos três sistemas de *software* selecionados, verificando a conformidade com as Leis de Lehman. A segunda etapa utiliza as lições aprendidas para desenvolver diretrizes práticas que podem ser aplicadas no desenvolvimento e manutenção de *software*. Os resultados confirmam que as Leis de Lehman são aplicáveis e relevantes para os sistemas de *software* modernos, proporcionando uma base teórica para orientar práticas de desenvolvimento e manutenção. As métricas analisadas demonstram padrões de crescimento e complexidade que corroboram as leis propostas por Lehman. Além disso, o estudo oferece uma sumarização das lições aprendidas e diretrizes práticas para desenvolvedores e equipes de manutenção, visando melhorar a adaptabilidade, reduzir a complexidade e garantir a qualidade contínua dos sistemas. Essas diretrizes são baseadas nas evidências empíricas coletadas e oferecem recomendações para a evolução eficiente do *software*. As principais contribuições deste trabalho incluem a validação empírica das Leis de Lehman, a aplicação do *framework* GQM como base metodológica e a criação de diretrizes práticas baseadas nas lições aprendidas.

Palavras-chave: leis de Lehman; evolução de software; GQM; diretrizes práticas; qualidade de software.

ABSTRACT

This study investigates the applicability and relevance of Lehman's Laws in the current context of software development, using a quantitative approach to analyze three modern systems: Dropwizard, K-9 Mail, and LanguageTool. Through the Goal Question Metric (GQM) framework, the research empirically validates Lehman's Laws, which describe fundamental principles of software evolution, such as continuous change, increasing complexity, and conservation of organizational stability. The study is divided into two main stages. The first stage involves analyzing the evolution metrics of the three selected software systems, verifying their compliance with Lehman's Laws. The second stage utilizes the lessons learned to develop practical guidelines that can be applied to software development and maintenance. The results confirm that Lehman's Laws are applicable and relevant to modern software systems, providing a theoretical foundation for guiding development and maintenance practices. The analyzed metrics demonstrate growth and complexity patterns that corroborate Lehman's proposed laws. Additionally, the study offers a summary of the lessons learned and practical guidelines for developers and maintenance teams, aiming to improve adaptability, reduce complexity, and ensure the continuous quality of systems. These guidelines are based on empirical evidence collected and offer recommendations for the efficient evolution of software. The main contributions of this work include the empirical validation of Lehman's Laws, the application of the GQM framework as a methodological basis, and the creation of practical guidelines based on the lessons learned.

Keywords: lehman's laws; software evolution; GQM; practical guidelines; software quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CBO	*Coupling Between Objects*
CK	*Chidamber & Kemere Metrics*
CSV	Comma-separated Values
ESSC	Categoria de evolução das características do sistema de software
GQM	*Goal-Question-Metric*
LCOM	*Lack of Cohesion in Methods*
LOC	Lines of Code
NASA	*National Aeronautics and Space Administration*
OERC	Restrições de recursos organizacionais ou econômicos
RFC	*Response For a Class*
SBSI	Simpósio Brasileiro de Sistemas de Informação
WMC	*Weighted Methods per Class*

SUMÁRIO

1 INTRODUÇÃO

No cenário atual de desenvolvimento de *software*, a evolução constante é uma necessidade intrínseca para garantir a adaptação e a relevância dos sistemas frente às mudanças nas necessidades das organizações, ambientes e usuários (LEHMAN, 1996). Todavia, acompanhar essa evolução e atestar a qualidade do *software* ao longo do tempo é um desafio complexo, que demanda análises aprofundadas e abordagens sólidas.

Nesse contexto, surge o desafio de validar e compreender a aplicação das Leis de Lehman e Ramil (2001), que descrevem princípios observados na evolução de *software*, incluindo mudança contínua, aumento da complexidade, conservação da estabilidade organizacional, entre outras. Embora essas leis tenham sido propostas há décadas, é crucial verificar sua relevância e aplicabilidade no desenvolvimento de *software* atualmente, onde novos métodos, ferramentas e técnicas estão surgindo constantemente (KOUR; SINGH, 2016).

A validação dessas leis é fundamental para oferecer orientações aos desenvolvedores e equipes de manutenção de *software*, auxiliando-os a entender melhor como os sistemas evoluem ao longo do tempo e como garantir a qualidade contínua dos produtos desenvolvidos. Além disso, compreender a aplicação dessas leis pode fornecer resultados para o aprimoramento de práticas de desenvolvimento e manutenção dos sistemas.

Assim, a problemática central deste estudo reside na capacidade dos desenvolvedores e equipes de manutenção de *software* em garantir a evolução contínua e a qualidade dos sistemas em um ambiente de constantes mudanças. A hipótese principal é que as Leis de Lehman continuam a ser aplicáveis e relevantes para os sistemas de *software* modernos, fornecendo uma base para guiar práticas de desenvolvimento e manutenção.

1.1 Objetivos

O objetivo deste estudo é realizar uma análise da evolução de três sistemas de *software* modernos, definidos como aqueles que foram desenvolvidos ou amplamente atualizados nos últimos cinco anos, incorporando práticas de desenvolvimento ágil e integração contínua, a fim de apreciar a validade das Leis de Lehman no contexto atual, utilizando o *framework Goal Question Metric* GQM e criar diretrizes a partir do resumo das lições aprendidas.

Através de uma abordagem baseada em métricas e análises empíricas, busca-se verificar se as leis propostas por Lehman (1979) são aplicáveis aos sistemas atuais, definidos

neste estudo como aplicações de médio a grande porte, tanto em ambiente web quanto *desktop*, que foram desenvolvidas ou amplamente atualizadas nos últimos anos. O estudo foca em soluções que estão ativamente evoluindo para atender às demandas tecnológicas contemporâneas, sem incluir sistemas legados. Com isso, pretende-se fornecer evidências práticas para embasar os resultados obtidos. Este estudo está intrinsecamente ligado à capacidade dos envolvidos na manutenção do *software* de entenderem e fazerem as mudanças necessárias.

1.2 Trabalho Proposto

Tendo em vista a hipótese formulada, parte-se para a realização do experimento para verificar se a mesma tem aplicabilidade real ou não. O experimento foi realizado em duas etapas principais com objetivos distintos.

A primeira etapa é voltada para a verificação da aplicabilidade das Leis de Lehman na evolução de sistemas de *software* modernos. Para isso, foram utilizados três sistemas específicos: Dropwizard, K-9 Mail e LanguageTool.

No segundo momento do experimento, a preocupação foi relativa à formulação de diretrizes práticas para o desenvolvimento e manutenção de sistemas de *software* com base nas lições aprendidas. A análise das métricas e dos dados coletados nos sistemas estudados permitiu o desenvolvimento de diretrizes que visam melhorar a adaptabilidade, reduzir a complexidade e garantir a qualidade contínua dos sistemas.

1.3 Organização do trabalho

Além do capítulo de introdução, este documento apresenta mais quatro capítulos que estão organizados da seguinte forma. O Capítulo 2 apresenta a fundamentação teórica e trabalhos relacionados. O Capítulo 3 descreve a validação das Leis de Lehman no contexto dos três *softwares* e a análise dos dados. O Capítulo 4 integra os resultados das análises dos três sistemas e desenvolve diretrizes práticas para o desenvolvimento e manutenção de *software* com base nas lições aprendidas. Finalmente, o Capítulo 5 apresenta as conclusões e trabalhos futuros, resumindo as principais descobertas do estudo, discutindo suas implicações e sugerindo direções para futuras pesquisas.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capitulo explora três importantes abordagens para avaliação e medição de qualidade de *software* e oferece uma visão abrangente dessas abordagens e como elas podem ser aplicadas para medir e melhorar a qualidade de sistemas computacinais em diferentes contextos.

2.1 Leis de Lehman

A conhecida contribuição de Lehman gira em torno de suas teorias sobre a evolução do *software*, desenvolvidas e aperfeiçoadas ao longo de muitos anos (LEHMAN, 1996). De acordo com a visão de Lehman, a instalação de um programa provoca mudanças no ambiente em que ele é executado. Portanto, um programa projetado para operar no mundo real não pode ser completamente especificado por duas razões: é impossível prever todas as complexidades do ambiente real em que será utilizado e, igualmente importante, o próprio programa afetará o ambiente assim que começar a ser utilizado. O ambiente no qual um sistema de *software* está inserido inevitavelmente evoluirá ao longo do tempo, muitas vezes em direções imprevisíveis (CHIDAMBER; KEMERER, 1994). Dessa forma, o ambiente do programa (incluindo seus usuários) torna-se a entrada para um *loop* de *feedback* que impulsiona sua evolução constante.

Embora um programa possa atender perfeitamente às necessidades de um usuário em determinado momento, ele requer ajustes explícitos à medida que as circunstâncias mudam para continuar a atender a essas necessidades (GODFREY; MICHAEL, 2008). A evolução é uma característica essencial do *software* no mundo real, porque conforme os requisitos mudam, também mudam os critérios de satisfação (COOK *et al.*, 2006). Lehman apontou que aperfeiçoar com sucesso um sistema de *software* existente é uma tarefa extremamente desafiadora. Ele resumiu suas observações sobre os sistemas de *software*, como são conhecidos hoje, nas Leis da Evolução de *software*.

Lehman (1980) definiu três diferentes categorias de sistemas chamados programas tipo S, programas tipo P e programas tipo E. No domínio dos programas do tipo S, encontra-se aqueles que podem ser formalmente especificados; os programas do tipo P não podem ser especificados, mas um processo iterativo é empregado para encontrar uma solução funcional; e os programas do tipo E, por outro lado, integram-se perfeitamente ao mundo real tornando-se parte dele, modificando-o e exigindo um sistema de *feedback* no qual o programa e seu ambiente evoluem de maneira coordenada.

De acordo com Lehman e Ramil (2001), as leis da evolução de *software* se concentram principalmente nos sistemas do tipo E e podem ser enumeradas como segue a Tabela 1:

Tabela 1 – Leis de Evolução de Software

Número e Ano	Nome
I -1974	MUDANÇA CONTINUA
II - 1974	AUMENTO DA COMPLEXIDADE
III - 1974	AUTO-REGULAÇÃO
IV - 1978	CONSERVAÇÃO DA ESTABILIDADE ORGANIZACIONAL
V - 1991	CONSERVAÇÃO DA FAMILIARIDADE
VI - 1991	CRESCIMENTO CONTÍNUO
VII - 1996	DECLÍNIO DA QUALIDADE
VIII - 1971/1996	SISTEMA DE*Feedback*

Fonte: o autor.

Herraiz *et al.* (2013) apontam que tais leis são classificadas em três categorias amplas. Na Categoria de evolução das características do sistema de software (ESSC), estão incluídas as seguintes leis:

- LEI DA MUDANÇA CONTÍNUA: o sistema deve se adaptar constantemente ou perderá efetividade ao longo do tempo.
- LEI DO CRESCIMENTO CONTÍNUO: o sistema precisa continuar se desenvolvendo para manter a satisfação dos usuários.
- AUMENTO DA COMPLEXIDADE: a menos que os desenvolvedores intervenham, o sistema se tornará cada vez mais complexo.
- DECLÍNIO DA QUALIDADE: a qualidade do sistema diminuirá a menos que os desenvolvedores o adaptem rigorosamente às mudanças no ambiente operacional.

Duas leis relacionadas a Restrições de recursos organizacionais ou econômicos (OERC) incluem as leis de:

- CONSERVAÇÃO DA FAMILIARIDADE: à medida que o sistema evolui e cresce, os profissionais envolvidos, incluindo desenvolvedores e usuários, devem manter sua expertise para garantir o sucesso da evolução.
- CONSERVAÇÃO DA ESTABILIDADE ORGANIZACIONAL: a taxa média de trabalho dos profissionais ao utilizar um sistema em evolução deve permanecer constante para garantir a estabilidade organizacional.

As duas leis adicionais são:

- AUTO-REGULAÇÃO: a evolução global do sistema tipo E é regulada por meio de

feedback.

- SISTEMA DE*Feedback*: para obter melhorias significativas no sistema por meio da evolução, os desenvolvedores devem considerar a evolução como um sistema de apreciação, levando em consideração a entrada dos usuários.

Essas leis desempenham um papel fundamental na compreensão da evolução de sistemas de *software* e são aplicáveis em diversos contextos. Ao reconhecer e seguir essas leis, os desenvolvedores podem orientar-se de maneira mais eficiente durante o processo de desenvolvimento e evolução de sistemas de *software*.

2.2 *Goal-Question-Metric* (GQM)

A abordagem GQM pressupõe que as organizações devem estabelecer metas claras para si mesma e seus projetos, a fim de realizar medições com propósito (ROSENBERG *et al.*, 2000). Isso envolve vincular essas metas aos dados que definem operacionalmente tais objetivos e fornecer uma estrutura para interpretar esses dados de acordo com as metas declaradas. Na concepção de Caldiera e Rombach (1994), é fundamental discriminar, de forma geral, as necessidades de informação da organização para que possam ser quantificadas sempre que possível, permitindo a análise dos dados quantificados para verificar o alcance das metas.

Este método foi originalmente desenvolvido para avaliar deficiências em um grupo de itens no ambiente do *National Aeronautics and Space Administration* (NASA). Um experimento de estudo de caso foi conduzido por Runeson e Höst (2009) e conforme relatos de Giray (2021), várias abordagens experimentais foram subsequentemente exploradas.

A aplicação do GQM leva à criação de um sistema de medição concentrado em um grupo específico de questões, juntamente com um conjunto de regras para interpretar os dados coletados. O modelo de medição possui três níveis:

1 - Nível conceitual: define-se uma meta para um objeto, considerando modelos de qualidade, pontos de vista e ambiente específicos. Os objetos medidos podem ser:

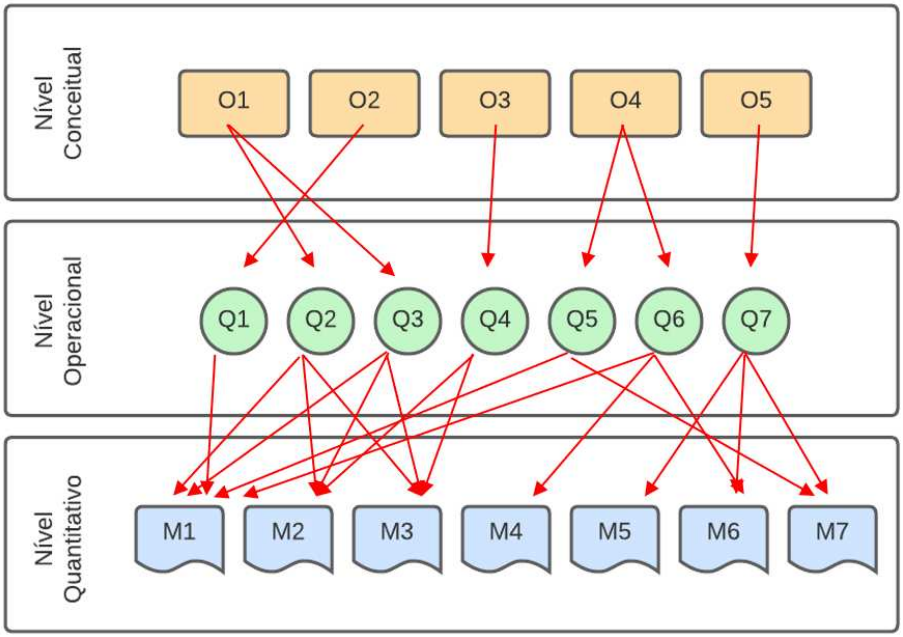
- Produtos: artefatos, entregáveis e documentos produzidos durante o ciclo de vida do sistema, como especificações, *designs* e programas;
- Processos: atividades relacionadas ao *software* que ocorrem ao longo do tempo, como especificação, projeto, teste e entrevistas;
- Recursos: elementos usados pelos processos para produzir suas saídas, como equipe, hardware, *software* e espaço físico.

2 - Nível Operacional: elabora-se um conjunto de perguntas para caracterizar como avaliar ou alcançar uma meta específica, com base em um modelo. Essas perguntas descrevem o objeto medido (produto, processo e recurso) em relação a um problema de qualidade e determinam sua qualidade a partir do ponto de vista escolhido.

3 - Nível Quantitativo: associa-se um conjunto de dados, métricas, a cada pergunta para respondê-las de forma quantitativa.

Um modelo GQM é uma estrutura hierárquica Figura 1 que começa com uma meta (especificando o propósito da medição, o objeto medido, o problema e o ponto de vista). A meta é detalhada em uma série de perguntas, que geralmente desmembram o problema em seus principais componentes. Cada pergunta é refinada em métricas, algumas objetivas e outras subjetivas. A mesma métrica pode ser usada para responder diferentes perguntas dentro da mesma meta. Vários modelos GQM podem compartilhar perguntas e métricas, garantindo que diferentes pontos de vista sejam adequadamente considerados durante a medição (ou seja, a métrica pode ter valores diferentes dependendo do ponto de vista).

Figura 1 – Estrutura básica do GQM.



Fonte: o autor.

2.3 Métricas de *Software*

Para Fenton e Norman (2014), as métricas de *software* desempenham um papel essencial no desenvolvimento, avaliação e manutenção de sistemas de *software*. Elas oferecem

uma abordagem quantitativa para medir diversos aspectos, proporcionando uma compreensão mais detalhada para desenvolvedores, gerentes de projeto e *stakeholders* (BOSCH *et al.*, 2021). Estas métricas são utilizadas para avaliar características específicas de um sistema, programa ou processo de desenvolvimento. Elas proporcionam uma visão objetiva e mensurável, permitindo uma análise mais precisa e fundamentada (FENTON; NORMAN, 2014).

Métricas de Código-Fonte: as métricas de código-fonte são focadas na avaliação das características do próprio código, oferecendo percepções específicas sobre a qualidade, eficiência e complexidade do sistema de informação (STOL; FITZGERALD, 2018).

Chidamber & Kemere Metrics (CK) é um conjunto de métricas de *software* desenvolvido por Chidamber e Kemerer (1994) com o objetivo de medir a complexidade e o tamanho de sistemas orientados a objetos. Essas métricas são amplamente utilizadas na indústria de *software* para avaliar a qualidade e a manutenibilidade de um código.

Métricas de Processo: as métricas de processo concentram-se na avaliação do desempenho e eficácia do processo de desenvolvimento em si. Essas métricas fornecem uma visão holística das práticas e atividades realizadas durante o ciclo de vida do *software*. De acordo com Hall *et al.* (2012), essas métricas são essenciais para entender como os processos influenciam a qualidade e a produtividade do desenvolvimento de *software*.

3 UM ESTUDO EXPLORATÓRIO SOBRE A VALIDAÇÃO DAS LEIS DE LEHMAN

Este capitulo apresenta um estudo do uso do *framework* GQM para validar as Leis de Lehman no contexto de três *softwares* atuais. O estudo, aceito e publicado no Simpósio Brasileiro de Sistemas de Informação (SBSI) 2024, segue uma abordagem empírica e está estruturado da seguinte forma: a Seção 3.1 detalha a metodologia utilizada no estudo. A Seção 3.2 apresenta a análise dos resultados obtidos e a validação das Leis de Lehman. A Seção 3.3 discute os trabalhos relacionados. Por fim, a Seção 3.4, são abordadas as ameaças à validade do estudo.

3.1 Configuração do Estudo

O estudo realizado adota uma abordagem metodológica para avaliar a validade das leis de Lehman nos dias atuais e verificar seu cumprimento. Trata-se de um estudo empírico, que envolve a coleta e análise de dados para responder às perguntas de pesquisa propostas (MOLLÉRI *et al.*, 2019). Foram executados os seguintes passo:

Passo 1: Sistemas alvos do estudo. Este estudo se concentrou na análise de três sistemas distintos — Dropwizard, K-9 Mail e LanguageTool — com o objetivo específico de validar a aplicabilidade das Leis de Lehman sobre a evolução dos *softwares*. A seleção desses sistemas proporcionou um contexto diversificado para examinar como as leis se manifestam em diferentes ambientes.

Dropwizard: um *framework* de desenvolvimento de aplicativos *Web* em Java, hospedado no GitHub. Com mais de 8.075 *commits* na última versão, o Dropwizard exibe uma ativa contribuição dos desenvolvedores, evidenciada por 3.500 *forks* e mais de 8.400 estrelas. Foram selecionadas 14 versões do sistema com base no versionamento semântico "*Major.Minor.Patch*", focando nas versões que representam mudanças significativas para a análise da evolução do *software*.

K-9 Mail: um aplicativo de e-mail *open-source* para Android, marcado por uma presença ativa no GitHub com 364 *watchers*, 2.400 *forks* e 8.700 estrelas. O K-9 Mail, com 13.189 *commits* e 293 *releases*, foi escolhido para este estudo devido ao engajamento significativo da comunidade e contribuição de 301 desenvolvedores. A seleção de 10 versões representativas foi guiada pelo versionamento semântico, visando identificar as mudanças evolutivas mais impactantes no *software*.

LanguageTool: um verificador de estilo e gramática para mais de 25 idiomas, com

155 *watchers*, 1.200 *forks* e 10.400 estrelas no GitHub. Com 73.426 *commits* distribuídos em 73 *releases* e contribuições de 183 desenvolvedores, o LanguageTool foi incluído para oferecer uma perspectiva adicional na aplicação das Leis de Lehman. Dez versões foram escolhidas com base no critério de versionamento semântico, para abranger as etapas significativas do seu desenvolvimento.

A diversidade dos sistemas — um *framework* de desenvolvimento *web*, um aplicativo de e-mail para Android e um verificador de gramática multilíngue — permite uma análise comparativa das tendências de evolução em diferentes áreas do desenvolvimento de sistemas (FERREIRA *et al.*, 2021). O uso do versionamento semântico para a seleção de versões específicas assegura que as análises estejam alinhadas com mudanças significativas dentro de cada sistema, segundo a ótica dos seus desenvolvedores, reforçando o foco na validação das leis de evolução propostas por Lehman.

Passo 2: Objetivo e questões de pesquisa. Apoiado pelo modelo GQM, o objetivo do estudo foi definido. Analisar **(objeto)** sistemas com o propósito de **(finalidade)** comparar no que diz respeito à **(propriedade)** evolução desses sistemas em relação às oito leis de Lehman do ponto de vista **(quem usa os dados)** dos desenvolvedores no contexto **(ambiente da análise)** do processo de desenvolvimento.

A estrutura apresentada na Tabela 2 é fundamental para sistematizar a análise conforme o modelo GQM, enfocando a aplicabilidade das oito Leis de Lehman nas versões dos *softwares*. No nível conceitual, os objetivos (O1 a O8) derivam diretamente das Leis de Lehman, formando a base para a verificação dos resultados deste estudo. Essas leis descrevem os princípios fundamentais do desenvolvimento de sistemas e sua evolução ao longo do tempo. Progressivamente, no nível operacional, são formuladas as questões de pesquisa (QP1 a QP11) que direcionam o foco da análise para a verificação da pertinência das leis em discussão. Finalizando, no nível quantitativo, são especificadas métricas (M1 a M12) destinadas a responder às questões de pesquisa estabelecidas. Estas métricas permitem uma análise quantitativa detalhada. A seleção dessas métricas foi feita com o propósito de fornecer dados empíricos capazes de sustentar ou contestar a manifestação das dinâmicas propostas pelas Leis nos sistemas analisados. A correlação entre as questões de pesquisa e as métricas é claramente definida, garantindo que cada questão seja abordada de maneira empírica e mensurável.

Passo 3: Coleta e processamento de dados. Os sistemas analisados neste estudo são de código aberto, com seus códigos-fonte hospedados em repositórios Git no GitHub,

Tabela 2 – Estrutura do GQM para avaliar as leis de Lehman

NÍVEL CONCEITUAL		
Código	Meta	
O1	Mudança Continua	
O2	Aumento da Complexidade	
O3	Auto-Regulação	
O4	Conservação da Estabilidade Organizacional	
O5	Conservação da Familiaridade	
O6	Crescimento Contínuo	
O7	Declínio da Qualidade	
O8	Sistema de *Feedback*	
NÍVEL OPERACIONAL		
Código	Perguntas	Metas
QP1	Aconteceu evolução no decorrer das versões?	O1
QP2	Existe uma tendência de aumento ou diminuição no número de commits ao longo do tempo por versões?	O1
QP3	O acoplamento do sistema diminuiu na sua evolução?	O2
QP4	Como a complexidade evoluiu ao longo das diferentes versões do software?	O2
QP5	Houve um aumento na métrica RFC em comparação com versões anteriores?	O2
QP6	Quantas classes foram excluídas e adicionadas em cada versão do software?	O3
QP7	Como a taxa de mudança em sistemas influencia a capacidade de compreensão dos envolvidos no processo?	O4
QP8	Houve algum impacto na adição de novas funcionalidades e atualizações?	O5
QP9	O software consegue expandir/evoluir?	O6
QP10	Como a coesão do sistema evoluiu ao longo do tempo, afetando o Declínio Inevitável?	O7
QP11	Ocorre *Feedback* nos sistemas?	O8
NÍVEL QUANTITATIVO		
Código	Métricas	Perguntas
M1	Lines of Code (LOC)	QP1, QP9
M2	Commits por versões	QP2, QP7, QP9
M3	Analisar o número de classes excluídas e adicionadas por versão	QP6
M4	Resposta para classe – RFC	QP5
M5	*Lack of Cohesion in Methods* (LCOM)	QP10
M6	CBO	QP3, QP10
M7	Métodos de classe	QP2, QP9
M8	WMC	QP4
M9	Complexidade Ciclomática	QP4, QP6
M10	Proporção comentários para linhas de código	QP10, QP11
M11	Mudança Incremental	QP7, QP8, QP9, QP10, QP11
M12	*issues* abertas e fechadas	QP8, QP10, QP11

Fonte: elaborado pelo autor (2023).

de onde os dados foram coletados. A coleta de dados e a análise do código-fonte seguiram um procedimento estruturado. Inicialmente, as versões selecionadas de cada *software* foram extraídas dos respectivos repositórios Git. Posteriormente, as ferramentas Understanding [1] e a calculadora CK [2] foram empregadas para o cálculo das métricas de código-fonte necessárias para este estudo. Cada versão foi submetida a uma análise por meio dessas ferramentas, resultando em dados armazenados em arquivos no formato Comma-separated Values (CSV), escolhido por sua capacidade de armazenar dados tabulares de forma simples e pela compatibilidade com a maioria das ferramentas de análise de dados.

Os dados foram coletados ao longo do período de desenvolvimento dos *softwares* com base nos padrões estabelecidos pelo versionamento semântico adotado pelo projeto, seguindo

[1] https://documentation.scitools.com/pdf/understand.pdf
[2] https://github.com/mauricioaniche/ck

o padrão "*Software*.2.5.33". Assim, segundo Cosentino *et al.* (2017), nessa abordagem cada número na sequência de versão tem um significado específico, permitindo a seleção precisa de qual versão considerar. Considerando que o primeiro número da sequência, chamado "*major*", indica uma grande alteração incompatível com as versões anteriores, todas as versões em que ocorrem grandes alterações foram incluídas na seleção. Esta seleção destina-se a conter versões significativamente modificadas para permitir uma análise aprofundada das mudanças e implicações dessas mudanças.

O segundo número na sequência, chamado de "*minor*", refere-se a alterações médias no *software*, como a inclusão de novas funções. Nesse cenário, a escolha recaiu sobre o primeiro e o último "*minor*"de cada "*major*". Essas versões são relevantes para analisar a evolução do *software* ao longo do tempo, considerando as funcionalidades adicionadas e as melhorias implementadas. Por fim, o último número da sequência representa as correções realizadas no *software*, conhecido como "*patch*". Neste estudo, foi decidido selecionar a primeira e a última versão de cada "*minor*", permitindo uma análise abrangente das correções aplicadas em cada conjunto de funções.

Para simplificar a análise dos dados, foi adotado o método de registrar cada métrica em um arquivo separado. Nesse formato, as linhas representam as classes, enquanto as colunas representam as diferentes versões do sistema. Essa abordagem proporciona uma visão clara das informações coletadas, oferecendo uma compreensão das características do sistema em estudo. As métricas de processo foram coletadas do repositório GitHub e armazenadas da mesma maneira que as demais métricas. Após a consolidação, os dados foram exportados para uma planilha no *Google Drive* [3]. Neste estudo, a análise apoia-se em uma lista de métricas que estão no nível quantitativo da Tabela 2.

O procedimento completo de coleta e análise dos dados teve uma duração aproximada de cinco meses, variando conforme o tamanho e a complexidade de cada sistema analisado. A reprodução deste estudo em outros *softwares* com versões de código disponíveis no GitHub pode ser realizada seguindo as mesmas etapas descritas, com os devidos ajustes na seleção de versões conforme necessário.

Algumas limitações devem ser consideradas. Ferramentas como Understanding e a calculadora CK podem requerer configurações específicas ou ajustes adicionais, dependendo da estrutura do código-fonte do *software* em questão. Além disso, sistemas com uma grande

[3] https://drive.google.com/drive/folders/1-3dxlVS5kZWpiHYOS74Fez-27pXFbytf?usp=sharing

quantidade de código ou múltiplas versões podem demandar um tempo maior de análise. Assim, recomenda-se que o pesquisador familiarize-se previamente com as ferramentas mencionadas e realize testes preliminares para ajustar o fluxo de trabalho conforme necessário.

Passo 4: Análise dos dados. Para realizar essa análise, gráficos de dispersão foram elaborados com base nos dados processados das métricas selecionadas. Esses gráficos cobrem diferentes períodos temporais para cada sistema: para o Dropwizard, foram analisadas 14 versões do período de 2011 a 2022; para o K-9 Mail, 10 versões de 2009 a 2022 foram consideradas; e para o LanguageTool, o estudo incluiu 10 versões de 2013 a 2022. O processo de criação dos gráficos e análise dos dados foi conduzido de forma manual, utilizando o Python como ferramenta principal para a manipulação e visualização dos dados. Esse procedimento, apesar de sistemático, demandou um tempo considerável, devido à necessidade de ajustar e processar as métricas de cada versão dos sistemas.

Em cada sistema, linhas de regressão foram aplicadas para cada métrica, permitindo uma avaliação da evolução dessas métricas ao longo do tempo e proporcionando uma visão comparativa sobre a aplicabilidade das Leis de Lehman na evolução de cada um desses *softwares*. A fórmula para a regressão linear simples, que relaciona uma variável dependente y a uma única variável independente x utilizada nesse estudo, é expressa como:

$$y = \beta 0 + \beta 1x + \varepsilon x = 0 \tag{3.1}$$

onde a variável y representa as métricas, enquanto a variável x são as versões dos sistemas que são usadas para prever ou explicar a variação em y. O termo β_0 é o intercepto, indicando o valor estimado de y quando x é igual a zero. O coeficiente β_1 é o coeficiente (ou gradiente), representando a mudança média em y para uma unidade de mudança em x. O termo ε é o termo de erro, representando a variabilidade não explicada no modelo. Este termo de erro reflete a diferença entre os valores previstos pelo modelo e os valores reais observados, indicando a variância nos dados que não é explicada pelo modelo.

Este estudo também inclui a avaliação dos intervalos de confiança de 95% para os parâmetros estimados. Esses intervalos de confiança fornecem uma estimativa do grau de incerteza associado aos parâmetros β_0 e β_1. Especificamente, um intervalo de confiança de 95% indica que, se o estudo fosse repetido várias vezes, em 95% dos casos, os intervalos calculados conteriam o verdadeiro valor do parâmetro. Assim, esses intervalos fornecem uma faixa plausível na qual os verdadeiros valores dos parâmetros residem, com um nível de confiança de 95%.

3.2 Discussão dos Resultados

Nesta seção, conduz uma análise das métricas previamente selecionadas, conforme discutido na Seção 3.1, a fim de avaliar a validade das Leis de Lehman no contexto atual. Por meio dessa análise, propociona-se uma descrição dos resultados obtidos a partir da validação empírica da evolução dos *softwares* utilizando as referidas leis.

3.2.1 LEI 1: Mudança Continua

Lehman (1996) relata que esta lei enfatiza a necessidade inerente de sistemas se adaptarem e se ajustarem continuamente para acompanhar as mudanças nas necessidades dos usuários, ambientes e organizações ao longo do tempo. Essa adaptabilidade contínua é fundamental para manter a eficácia e relevância do sistema, pois a falta de adaptabilidade resulta na perda de sua capacidade de atender adequadamente às necessidades em mudança, levando a um declínio progressivo em sua utilidade e valor para os usuários e para a organização ().

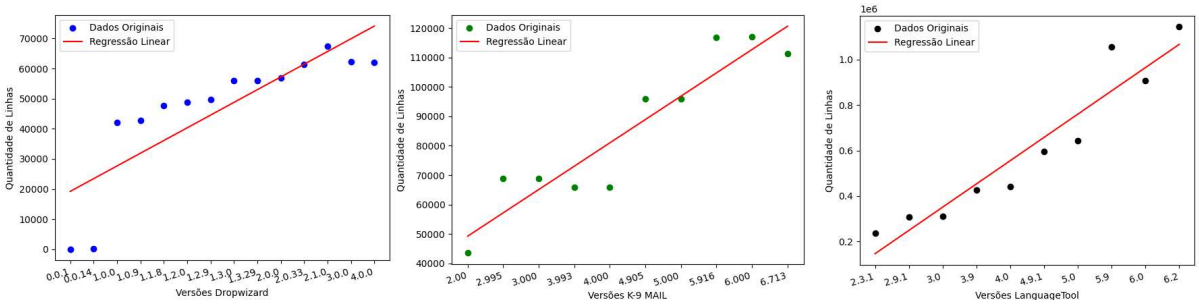
Duas perguntas foram formuladas para obter uma visão mais completa sobre a mudança contínua: QP01 e QP02.

Respondendo a QP1. A análise das linhas de código (M1) como mostrado na Figura 2, revela padrões distintos de crescimento em cada sistema. No caso do Dropwizard, observa-se um aumento médio de 4.220 linhas por versão, indicando uma evolução contínua. Por outro lado, tanto o K-9 Mail quanto o LanguageTool apresentam aumentos mais significativos, com coeficientes de gradiente de 7.934 e 102.295, respectivamente. O LanguageTool exibe o crescimento mais rápido, sugerindo uma expansão contínua em tamanho e complexidade.

Analisando as mudanças entre versões específicas, notam-se variações significativas em termos percentuais. Por exemplo, no LanguageTool, a transição da versão 5.0 para a 5.9 resulta em mais que o dobro das linhas de código. Isso evidencia um período de intenso desenvolvimento, indicando um crescimento linear nas linhas de código à medida que as versões avançam, corroborando estudos anteriores (FREUND *et al.*, 2006).

Respondendo a QP2. Conforme a Figura 3, todos os sistemas mostram um aumento no número de *commits* por versão (M2), com o LanguageTool apresentando o maior crescimento. Este aumento sugere uma intensificação nas atividades de desenvolvimento e uma tendência de crescimento, demonstrando uma contínua adaptabilidade e evolução dos sistemas. Ao considerar as variações percentuais no número de *commits* entre as versões, identifica-se que períodos

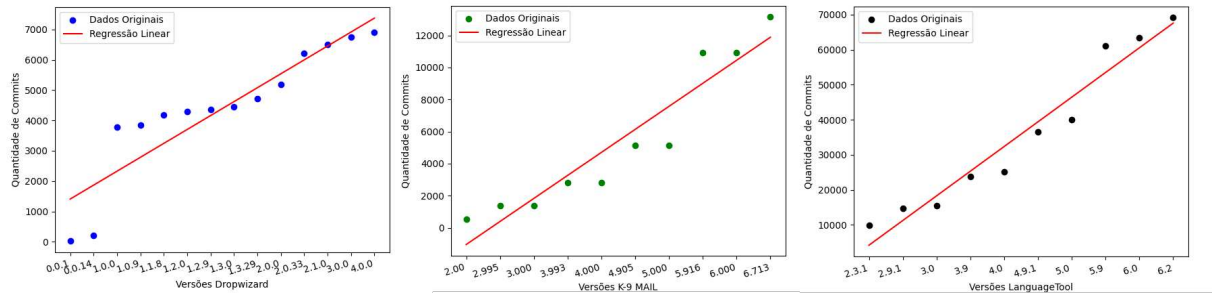
Figura 2 – Evolução da Quantidade de Linhas por Versão.



Fonte: elaborado pelo autor (2023).

específicos apresentam aumentos notáveis. Por exemplo, no K-9 Mail, a transição da versão 4.905 para a 5.000 evidencia um aumento considerável no número de *commits*, refletindo um período de mudanças significativas e possivelmente de introdução de novas funcionalidades ou correções importantes. Já no Dropwizard, o período da versão 0.0.1 para a versão 1.0.0, observa-se um crescimento notável de 15.6% no número de *commits*. Esse fato é pelo o grande número de contribuidores nesse período.

Figura 3 – Quantidade de Commits por Versão.

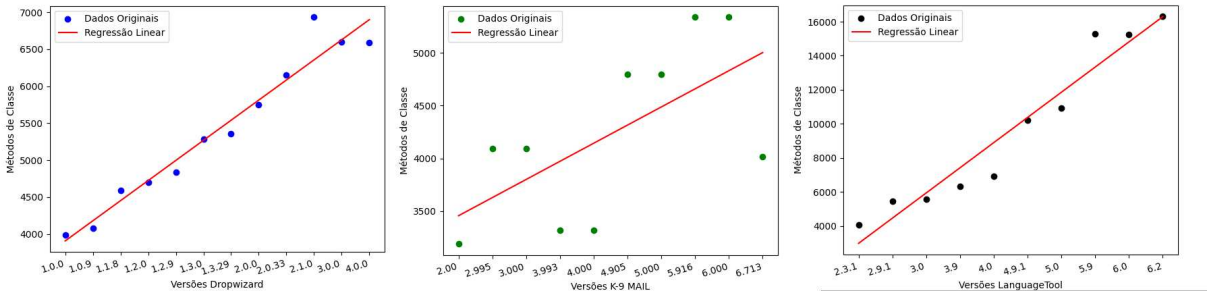


Fonte: elaborado pelo autor (2023).

Diante desses resultados, conclui-se que há uma tendência geral de aumento no número de *commits* ao longo do tempo por versões. Os coeficientes positivos indicam que, em média, a quantidade de *commits* tende a aumentar à medida que a versão dos sistemas avança. Complementando as análises anteriores, a Figura 4 mostra um aumento nos métodos de classe (M7) para cada sistema, com o LanguageTool mostrando um crescimento acelerado. Este padrão indica uma expansão contínua em termos de complexidade e funcionalidades.

A análise dos dados de LOC (M1), *commits* (M2) e métodos de classe (M7) fornece evidências claras de crescimento linear e contínuo em todos os sistemas estudados. Estes resultados confirmam a aplicabilidade da Lei de Mudança Contínua nos sistemas atuais, validando a teoria de que a constante adaptação é fundamental para manter a relevância e eficácia dos

Figura 4 – Métodos de classe por Versão.



Fonte: elaborado pelo autor (2023).

sistemas em ambientes dinâmicos.

3.2.2 LEI 2: Aumento da Complexidade

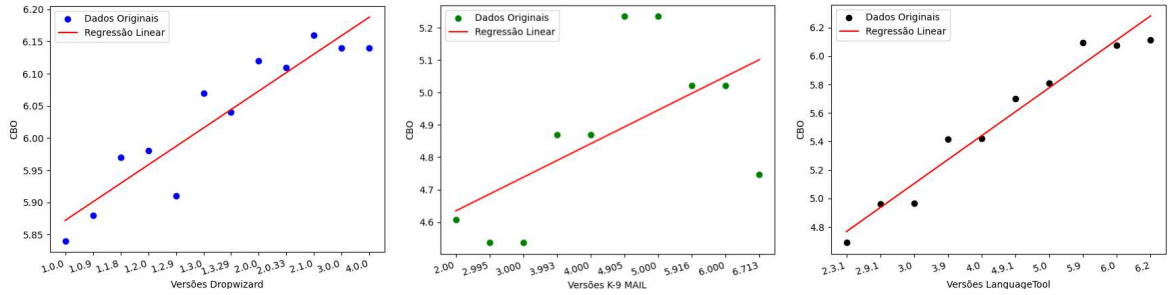
A lei do Aumento da Complexidade, afirma que, a menos que os desenvolvedores intervenham, um sistema tende a se tornar cada vez mais complexo (LEHMAN, 1996).

Respondendo a QP3. A análise da métrica CBO (M6) em diferentes versões dos sistemas revela padrões variados de acoplamento entre classes. No Dropwizard, observou-se um aumento leve no acoplamento, com um gradiente de 0,0287 e um crescimento percentual de 5,14%, da versão 5.84 para 6.14. O K-9 Mail apresenta um padrão mais complexo, com um gradiente de 0,0519 e um intervalo de confiança que inclui zero, sugerindo esforços pontuais para reduzir o acoplamento, possivelmente em resposta a um aumento na complexidade. Já o LanguageTool mostra um aumento significativo no acoplamento, com um gradiente de 0,1681 e um aumento percentual de cerca de 30,29% do CBO, da versão 4.69 para 6.11.

Esses resultados corroboram a lei do Aumento da Complexidade, indicando uma tendência natural de aumento na complexidade do *Software* ao longo do tempo. No entanto, o caso do K-9 Mail destaca que intervenções conscientes, como refatoração e revisão de *design*, podem mitigar essa tendência, sublinhando a importância de práticas contínuas de gestão da complexidade no desenvolvimento de *Software*.

Respondendo a QP4. Com base nos resultados da análise de regressão linear apresentados nas Figuras 6 (M9) e 7 (M8) no Dropwizard, constatou-se um aumento na complexidade do fluxo de controle e na estrutura das classes, com um aumento significativo na Complexidade Ciclomática (M9) e um crescimento modesto no WMC (M8). No K-9 Mail, a Complexidade Ciclomática apresentou um padrão incerto, enquanto a WMC demonstrou uma tendência de decréscimo, indicando esforços para redução da complexidade. O LanguageTool

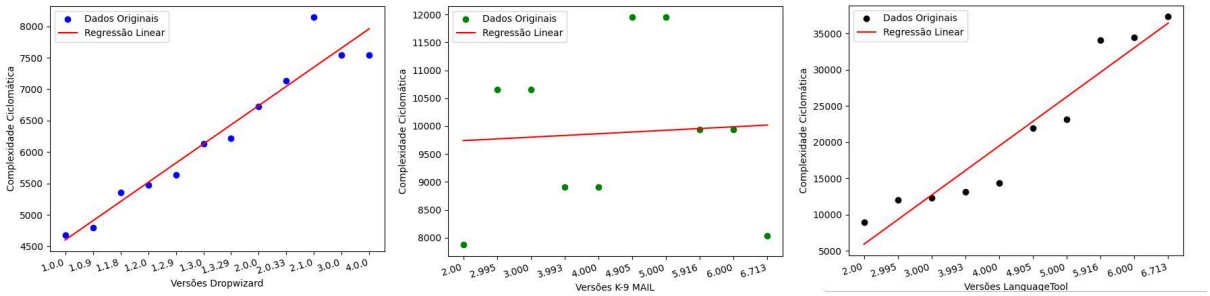
Figura 5 – CBO por Versão.



Fonte: elaborado pelo autor (2023).

mostrou um aumento acentuado em ambas as métricas, sinalizando um crescimento contínuo na complexidade.

Figura 6 – Complexidade Ciclomática.

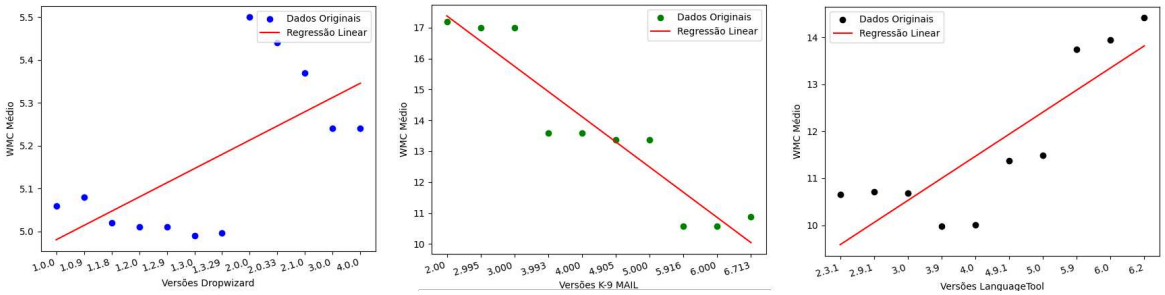


Fonte: elaborado pelo autor (2023).

A análise combinada das métricas de Complexidade Ciclomática e WMC nos três sistemas de informação revela padrões distintos de evolução da complexidade. No Dropwizard, observa-se um aumento geral na complexidade. No K-9 Mail, há uma indicação de esforços para gerenciar e potencialmente reduzir a complexidade estrutural, apesar de uma tendência menos definida na complexidade de controle de fluxo. Por fim, o LanguageTool demonstra um aumento consistente e significativo na complexidade em ambas as métricas, refletindo uma evolução contínua e acentuada da complexidade do sistema. Portanto, embora exista uma tendência geral de aumento da complexidade nos sistemas das versões, as abordagens adotadas para gerenciar essa complexidade diferem entre os projetos analisados.

Respondendo a QP5. A análise da métrica RFC (M4), Figura 8, nos três sistemas revela uma tendência geral de aumento na complexidade e interconectividade das classes. No primeiro sistema, foi observado um aumento consistente na RFC, com um coeficiente de gradiente significativo de 412 e um intervalo de confiança reforçando a tendência ascendente, evidenciado pelo aumento da RFC de 7.610 para 11.679. No segundo sistema, apesar de um

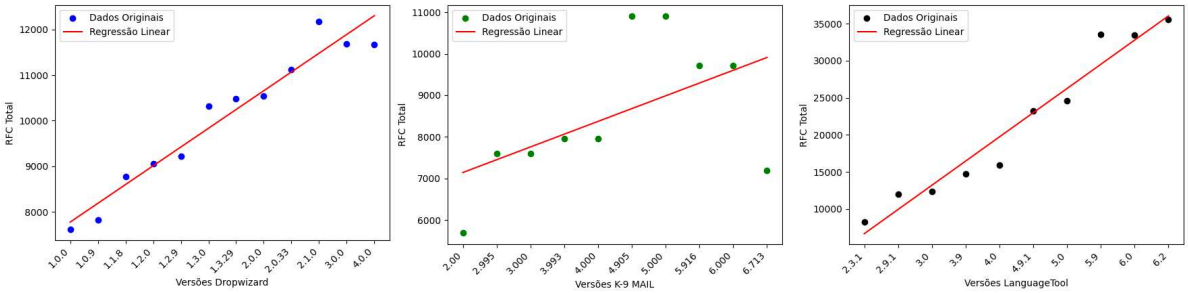
Figura 7 – WMC Médio.



Fonte: elaborado pelo autor (2023).

gradiente positivo de 307,59, o intervalo de confiança inclui valores negativos, trazendo alguma incerteza à tendência de aumento, que se reflete no crescimento da RFC de 5684 para 7.196. No LanguageTool, o aumento na RFC foi mais acentuado, com um gradiente de 3.261,88 e um intervalo de confiança que confirma uma tendência clara de aumento, com a RFC subindo de 8.229 para 35.509. Esses resultados apontam para um crescimento geral na complexidade e interações entre classes, medido pela métrica RFC, nos sistemas de informação analisados ao longo do tempo.

Figura 8 – RFC Total por Versão.



Fonte: elaborado pelo autor (2023).

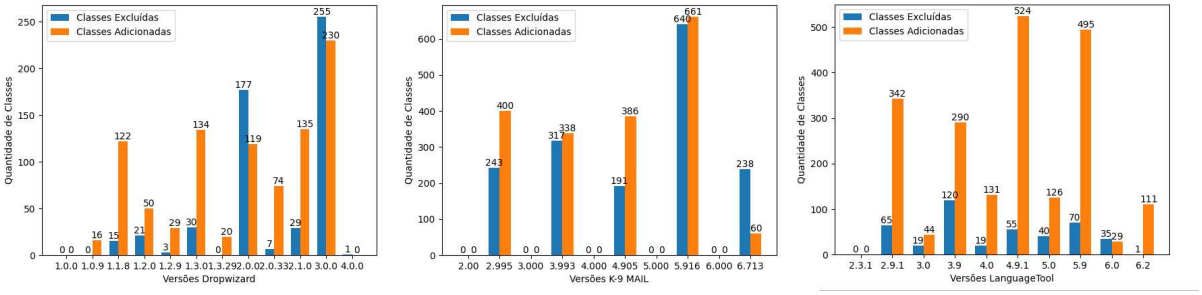
Em suma, a análise dessas métricas, sugere que, para o Dropwizard e o LanguageTool, a Lei 2 de Lehman é validada, evidenciando um aumento na complexidade ao longo do tempo. No K-9 Mail, a situação é menos clara, sugerindo que esforços para controlar a complexidade podem ter tido um impacto significativo. Essas descobertas destacam como a complexidade do sistema de informação é um fenômeno dinâmico, sujeito a variações dependendo de práticas específicas de desenvolvimento e manutenção adotadas pelos projetos do sistema.

3.2.3 LEI 3: Auto-Regulação

A Lei da Auto-Regulação em sistemas indica que ajustes são realizados dentro de limites definidos para manter a estabilidade e familiaridade dos produtos de *Software*. A auto-regulação é analisada neste estudo por meio do crescimento do número de LOC (M1), Figura 2, variação no número de métodos por classe (M7), Figura 4, e a frequência de classes adicionadas ou excluídas (M3), Figura 9.

Respondendo a QP6. A análise das alterações de classes nos sistemas Dropwizard, K-9 Mail e LanguageTool revela padrões dinâmicos que refletem a auto-regulação dos sistemas. No Dropwizard, uma atividade intensa de refatoração é indicada pela exclusão de 255 classes e adição de 230 na versão 3.0.0, correlacionada com mudanças na Complexidade Ciclomática. O K-9 Mail apresenta alterações notáveis, particularmente nas versões 2.995 e 5.916, com exclusões e adições substanciais de classes (243 excluídas e 400 adicionadas; 640 excluídas e 661 adicionadas, respectivamente), sugerindo revisões significativas em resposta a desafios ou mudanças nas necessidades dos usuários (COOK *et al.*, 2006). O LanguageTool mostra um aumento contínuo no número de classes adicionadas, especialmente nas versões 4.9.1 e 5.9, podendo indicar um aumento de funcionalidades.

Figura 9 – Classes excluídas e adicionadas por versão.



Fonte: elaborado pelo autor (2023).

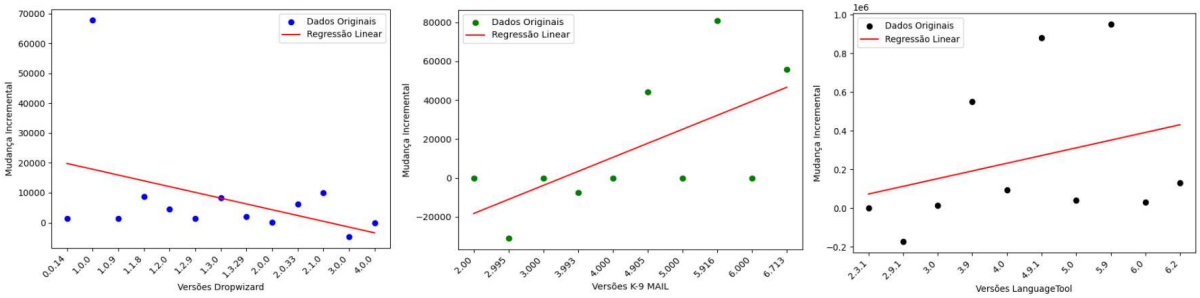
Esses padrões de alterações de classes nos sistemas de informação indicam uma natureza auto-regulatória, evidenciando a adaptação e resposta constante dos sistemas às necessidades internas e externas, alinhando-se com a Lei de Auto-Regulação de Lehman sobre a evolução dos sistemas de *Software*.

3.2.4 LEI 4: Conservação da Estabilidade Organizacional

De acordo com a Lei da Conservação da Estabilidade Organizacional, de Lehman (1996), a taxa de mudança em sistemas de informação tende a ser equilibrada pela capacidade de compreensão e manutenção por parte dos envolvidos, o que mantém a taxa de atividade global estável, apesar das transformações contínuas.

Respondendo a QP7. Analisando a evolução da mudança incremental (M11), Figura 10 e os dados de *Commits* (M2), Figura 3, observa-se padrões variados. No primeiro sistema, destacam-se grandes mudanças incrementais entre versões específicas, sugerindo flutuações na estabilidade organizacional. No K-9 Mail, um aumento substancial na mudança incremental entre certas versões sugere períodos de atividade de desenvolvimento intensificada. O LanguageTool apresenta um aumento massivo na mudança incremental em um intervalo específico, indicando um período de evolução acelerada do sistema.

Figura 10 – Mudança incremental por versão.



Fonte: elaborado pelo autor (2023).

A análise combinada das métricas de *Commits* e mudança incremental nos sistemas revela uma complexa dinâmica de mudança e estabilidade. Portanto, a validação da Lei de CONSERVAÇÃO DA ESTABILIDADE ORGANIZACIONAL de Lehman nestes sistemas é parcial. Em outras palavras, os padrões de desenvolvimento desses sistemas não seguem estritamente a lei, que prevê uma taxa de atividade de desenvolvimento constante ao longo do tempo.

3.2.5 LEI 5: Conservação da Familiaridade

A Lei da Conservação da Familiaridade destaca a importância de manter a familiaridade dos usuários e dos desenvolvedores com um sistema ao longo do tempo, mesmo com a adição de novas funcionalidades e atualizações.

Respondendo a QP8. Esta lei pode ser validada observando a mudança incremental

(M11), Figura 10 e as métricas de *issues* (M12). No Dropwizard, o fechamento de 7.766 *issues* sugere uma evolução contínua, mas com desafios na gestão de componentes externos, potencialmente impactando a familiaridade da equipe com o sistema. O K-9 Mail, com 733 *issues* abertas e 6.644 fechadas, revela um sistema em constante adaptação, enfrentando desafios para manter a familiaridade devido às mudanças frequentes. O LanguageTool, com 1.934 *issues* abertas e 7.927 fechadas, também exibe intensa atividade de manutenção e desafios na gestão da complexidade.

Esses dados indicam um esforço contínuo para evoluir e adaptar os sistemas, equilibrando a necessidade de inovação com a manutenção de um nível gerenciável de familiaridade, refletindo uma validação parcial da Lei da CONSERVAÇÃO DA FAMILIARIDADE.

3.2.6 LEI 6: Crescimento Contínuo

A Lei de Crescimento Contínuo sugere que a funcionalidade de um sistema de informação tende a crescer com a adição de novas funcionalidades e melhorias solicitadas pelos usuários. Esse crescimento é frequentemente refletido no aumento do tamanho do código. Originalmente, os *softwares* são desenvolvidos com base em requisitos iniciais, mas devem adaptar-se às demandas emergentes para satisfazer os usuários, promovendo assim o crescimento contínuo do sistema.

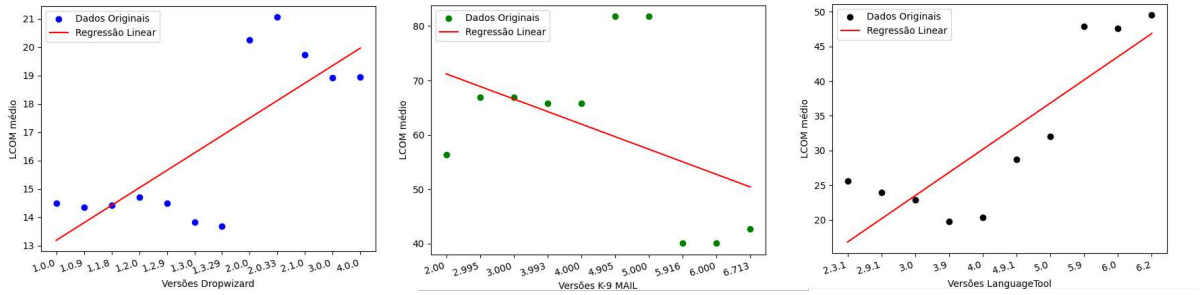
Respondendo a QP9: no estudo atual, o crescimento dos sistemas foi medido utilizando métricas como o Número de Linhas de Código (M1), Quantidade de *Commits* (M2) e Número de Métodos por Versão (M7). O aumento observado nessas métricas indica um crescimento em tamanho e funcionalidade nos sistemas analisados, corroborando a Lei de CRESCIMENTO CONTÍNUO.

3.2.7 LEI 7: Declínio da Qualidade

A análise dos gráficos das métricas LCOM (M5) e CBO (M6) permite uma avaliação mais precisa da evolução da coesão do sistema ao longo do tempo e seu impacto no declínio inevitável da qualidade do *software*.

Respondendo a QP10. Analisando a métrica LCOM (M5), Figura 11, observa-se que no primeiro sistema, há um aumento gradual indicando um declínio na coesão das classes ao longo do tempo, com o LCOM médio aumentando de 14,48 para 18,94. O segundo sistema, por outro lado, mostra uma melhoria na coesão com um gradiente negativo no LCOM, reduzindo

Figura 11 – Evolução de LCOM por Versão.



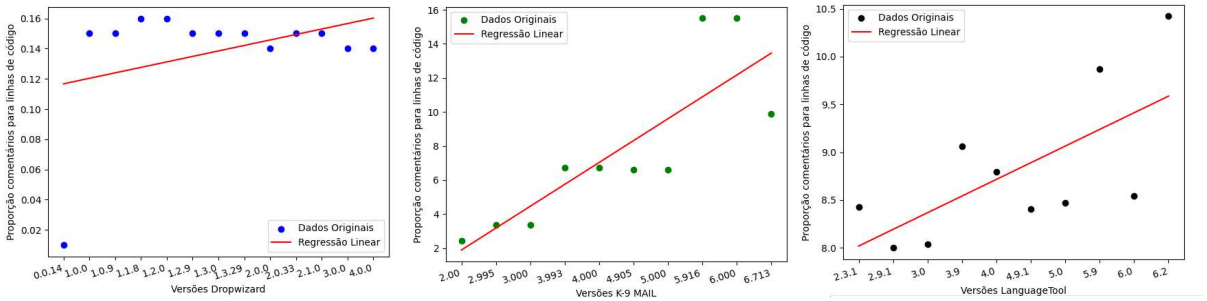
Fonte: elaborado pelo autor (2023).

de 56,37 para 42,77. O LanguageTool exibe um aumento significativo no LCOM de 25,58 para 49,50, indicando um declínio na coesão. Esses padrões sugerem variações na qualidade do sistema, com o Dropwizard e o LanguageTool enfrentando um declínio na coesão, enquanto o K-9 Mail apresenta uma melhoria.

Além disso, a análise da proporção de comentários nos códigos dos sistemas (M10) revela uma melhoria modesta no primeiro sistema (de 0.01 para 0.14), um aumento significativo no K-9 Mail (de 2,44 para 9,91), e no LanguageTool (de 8,43 para 10,42), indicando uma melhoria na documentação dos sistemas.

A análise das métricas nos três sistemas sugere que Lei DECLÍNIO DA QUALIDADE, é validada. Portanto, embora existam esforços para contrariar o declínio da qualidade, os sistemas continuam enfrentando o desafio de equilibrar a expansão e a manutenção da qualidade à medida que evoluem.

Figura 12 – Proporção comentários para linhas de código.



Fonte: elaborado pelo autor (2023).

3.2.8 LEI 8: Sistema de Feedback

Para a validação dessa lei, considera-se a interação das métricas de *issues* (M12), mudança incremental (M11), Figura 10 e a proporção de comentários para linhas de código

(M10), Figura 12.

Respondendo a QP11. No Dropwizard, a análise das *issues* mostra uma diversidade de problemas reportados, incluindo *bugs* e melhorias, sugerindo um processo ativo de *feedback* e adaptação. A mudança incremental, que exibe variações significativas, pode indicar respostas a demandas externas e internas. A proporção de comentários para linhas de código, embora tenha aumentado modestamente, reflete um esforço para melhorar a documentação e a compreensão do código, possivelmente como resposta a *feedbacks* internos. Para o K-9 Mail, o número substancial de *issues* abertas e fechadas evidencia um forte *loop* de *feedback* com os usuários. As mudanças incrementais significativas em certas versões podem ser vistas como uma resposta direta a este *feedback*. Adicionalmente, o aumento na proporção de comentários sugere uma ênfase na clareza e manutenção do código, um aspecto crucial no processo de *feedback* interno entre os desenvolvedores. O LanguageTool, com um número considerável de *issues* abertas e fechadas, também demonstra um ciclo ativo de *feedback* com seus usuários. As mudanças incrementais e o aumento na proporção de comentários refletem uma resposta ao *feedback* interno e externo, indicando um esforço contínuo para adaptar e aprimorar o sistema.

Portanto, a validação da Lei do SISTEMA DE *Feedback* é evidente nesses sistemas, refletindo a importância crítica dos *loops* de *feedback* multi-nível na evolução contínua e no aprimoramento dos sistemas de software.

3.3 Trabalhos Relacionados

O trabalho de Lee *et al.* (2007) teve foco em uma análise da evolução do *software JFreeChart*, utilizando métricas de tamanho, acoplamento de *fan-in/out* e coesão. O estudo observou que o grupo de classes adicionadas ao longo do tempo apresentou melhor qualidade do que o grupo de classes removidas. Além disso, o estudo constatou que as mudanças no *software JFreeChart* seguem as leis 1ª, 2ª e 6ª de evolução de *software* de Lehman, mas não seguem a 7ª lei.

Kour e Singh (2016) realizaram um estudo empírico sobre a evolução de sistemas de *software* construídos com metodologias ágeis. O estudo avaliou métricas de produto e processo em projetos ágeis, concluindo que as leis de evolução de *software* de Lehman também se aplicam a esses tipos de sistemas.

Cao *et al.* (2019) conduziram um estudo empírico em seis projetos de *blockchain* de código aberto, analisando 504 versões. O estudo verificou a aplicabilidade das leis de Lehman às

aplicações *blockchain* ao longo do tempo e constatou que algumas leis não foram confirmadas. Além disso, identificou tendências de revisões centralizadas e crescimento não uniforme em projetos de *blockchain*.

Outros pesquisadores, Nummenmaa *et al.* (2013), aplicaram as leis de evolução de *software* de Lehman ao desenvolvimento de jogos. O estudo mostrou que muitas das leis se aplicam aos jogos, mas algumas podem ser mais relevantes do que outras, dependendo do tipo de evolução do jogo.

O estudo apresentado neste artigo se diferencia dos trabalhos anteriores por sua abordagem de validação das Leis de Lehman utilizando a estrutura de GQM. Através de uma revisão dos trabalhos relacionados, foi possível identificar pesquisas sobre evolução e qualidade do *software* JFreeChart, evolução de sistemas de *software* ágeis, evolução de aplicações *blockchain* e aplicação das leis de evolução de *software* ao desenvolvimento de jogos. A Tabela 3 demonstra o comparativo entre esses estudos, destacando as leis validadas e não validadas.

Tabela 3 – Resumo de comparação

Leis de Lehman	Este estudo	(LEE *et al.*, 2007)	(KOUR; SINGH, 2016)	(CAO *et al.*, 2019)	(NUMMENMAA *et al.*, 2013)
I	Validado	Validado	Validado	Validado	Validado
II	Validado	Validado	Validado	Validado	Validado
III	Validado	Validado	Validado	Validado	Parc. Validado
IV	Parc. Validado	Validado	Validado	Validado	Validado
V	Parc. Validado	Validado	Validado	Validado	Validado
VI	Validado	Validado	Validado	Validado	Parc. Validado
VII	Validado	Validado	Validado	Não Validado	Validado
VIII	Validado	Não Validado	Validado	Validado	Validado

Fonte: elaborado pelo autor (2023).

3.4 Ameaças à Validade

Com o objetivo de assegurar a confiabilidade e a capacidade de generalização dos resultados, as ameaças à validade neste estudo foram cuidadosamente consideradas e tratadas. Quatro tipos principais de ameaças à validade, conforme identificados por Wohlin *et al.* (2012), foram abordados.

Validade Interna: as ameaças à validade interna referem-se a fatores que podem influenciar a relação causal entre as variáveis estudadas. Para atenuar esta ameaça, procedimentos rigorosos de controle e manipulação das variáveis foram implementados durante a coleta e análise dos dados. A aplicação de métodos estatísticos apropriados, incluindo análise de regressão, foi empregada para minimizar a influência de variáveis confundidoras, proporcionando uma análise

mais precisa das relações identificadas.

Validade Externa: as ameaças à validade externa tratam da capacidade de generalizar os resultados para outros contextos além do ambiente específico do estudo. Neste trabalho, a análise concentrou-se em três sistemas de informação distintos. Esta diversidade de sistemas contribui para atenuar as ameaças à validade externa, pois abrange uma variedade de contextos e cenários de uso. Ao incluir múltiplos sistemas, aumenta-se a probabilidade de que as conclusões sejam aplicáveis em uma gama mais ampla de situações.

Validade de Construção: as ameaças à validade de construção dizem respeito à adequação das medidas utilizadas para avaliar as variáveis do estudo. Para mitigar essa ameaça, selecionaram-se métricas reconhecidas na literatura e os critérios de avaliação foram definidos com precisão. A consistência na aplicação das métricas foi mantida ao longo da análise.

Validade de Conclusão: as ameaças à validade de conclusão estão relacionadas à interpretação correta dos resultados. Para minimizar essa ameaça, uma análise sistemática dos dados foi conduzida e as conclusões foram baseadas em evidências coletadas. Discussões com outros pesquisadores e revisão por pares foram essenciais para validar as conclusões do estudo.

3.5 Considerações Finais

Este capítulo investigou a evolução de sistemas de *software* para verificar a aplicabilidade das leis propostas por Lehman, fornecendo dados e análises que enriquecem a compreensão da dinâmica evolutiva desses sistemas. A análise de métricas de evolução e qualidade em projetos de sistemas revelou resultados que proporcionam uma visão das características evolutivas.

À luz das conclusões apresentadas, este capítulo representa uma contribuição relevante para o campo de investigação da evolução de *software* e a validação das Leis de Lehman. Suas descobertas proporcionam uma base para futuras pesquisas e práticas relacionadas ao desenvolvimento e evolução de sistemas de software, visando aprimorar sua qualidade e sustentabilidade ao longo do tempo.

No próximo capítulo, serão elaboradas diretrizes práticas para o desenvolvimento e manutenção de sistemas de *software*. As diretrizes serão baseadas nas observações e conclusões derivadas da análise das métricas dos três sistemas de *software*.

4 SUMARIZAÇÃO DAS LIÇÕES APRENDIDAS

No cenário atual de desenvolvimento de sistemas, as Leis de Lehman fornecem um quadro teórico essencial para entender a evolução contínua do *software*. Este capítulo tem como objetivo sumarizar as lições aprendidas na análise de três sistemas – Dropwizard, K-9 Mail e LanguageTool – e organizar essas lições em diretrizes práticas para desenvolvedores e pesquisadores que lidam com a evolução de *software* e conformidade com as Leis de Lehman.

4.1 Lei 1: Mudança Contínua

Lições Aprendidas:

- Evidência de Evolução: Todos os sistemas mostraram um crescimento contínuo no número de linhas de código e *commits* por versão, refletindo a necessidade de adaptação contínua para atender às demandas dos usuários;
- Práticas Recomendadas: Estabelecer processos regulares de revisão e atualização de código para garantir que os sistemas se adaptem às novas necessidades e tecnologias emergentes.

Implementar processos regulares de revisão e atualização de código é essencial para manter a relevância do sistema. Por exemplo, no estudo de Sejfia (2019), observou-se que sistemas que não realizam essas práticas tendem a se tornar obsoletos rapidamente. Além disso, a integração contínua e a entrega contínua (CI/CD) podem ser estratégias eficazes para garantir que as atualizações sejam implementadas de forma rápida e segura (PRATAMA; KUSUMO, 2021).

4.2 Lei 2: Aumento da Complexidade

Lições Aprendidas:

- Crescimento da Complexidade: A análise das métricas de acoplamento entre objetos CBO e complexidade ciclomática indicou um aumento na complexidade dos sistemas ao longo das versões. No entanto, esforços de refatoração em K-9 Mail mostram que a complexidade pode ser gerenciada com intervenções adequadas;
- Práticas Recomendadas: Implementar práticas de refatoração contínua e utilizar ferramentas de análise de código para monitorar e reduzir a complexidade ao

longo do desenvolvimento.

A refatoração contínua é uma prática essencial para gerenciar a complexidade crescente do *software*. Estudos como os de Chidamber e Kemerer (1994) demonstram que métricas de complexidade, como CBO e complexidade ciclomática, podem ser significativamente reduzidas por meio de refatoração. Além disso, o uso de ferramentas de análise estática, como SonarQube, pode ajudar a identificar áreas de alta complexidade que necessitam de refatoração (FAKHOURY *et al.*, 2019).

4.3 Lei 3: Auto-Regulação

Lições Aprendidas:

- Adaptações Internas: Os sistemas exibiram padrões de adição e exclusão de classes que refletem ajustes internos contínuos para manter a estabilidade e a funcionalidade. A análise mostrou que as mudanças eram frequentemente direcionadas por necessidades internas e *feedback* dos usuários;
- Práticas Recomendadas: Estabelecer processos internos que permitam ajustes e otimizações contínuas no sistema, tais como refatoração de código e realocação de recursos internos para áreas críticas.

A auto-regulação nos sistemas de *software* é vital para manter a estabilidade. Conforme Lehman (1996), os sistemas devem ser ajustados continuamente para responder às mudanças internas. Implementar um ciclo de autoavaliação dentro da equipe de desenvolvimento permite identificar e corrigir problemas de forma proativa, garantindo que o sistema permaneça estável e funcional.

4.4 Lei 4: Conservação da Estabilidade Organizacional

Lições Aprendidas:

- Taxa de Atividade: A análise combinada das métricas de *commits* e mudanças incrementais sugere que a taxa de atividade de desenvolvimento varia ao longo do tempo, desafiando a estabilidade organizacional. Períodos de alta atividade foram seguidos por fases mais estáveis;
- Práticas Recomendadas: Manter um ritmo constante de desenvolvimento e estabelecer mecanismos de controle de versões para gerenciar a estabilidade

organizacional durante períodos de intensa atividade.

Manter a estabilidade organizacional é fundamental para a evolução saudável do *software*. Estudos sugerem que a implementação de metodologias ágeis pode ajudar a gerenciar melhor as flutuações na taxa de atividade de desenvolvimento. Métodos como Scrum e Kanban permitem que as equipes ajustem rapidamente suas prioridades e se adaptem a mudanças sem comprometer a estabilidade organizacional (BECK *et al.*, 2001).

4.5 Lei 5: Conservação da Familiaridade

Lições Aprendidas:

- Desafios na Familiaridade: A quantidade de *issues* abertas e fechadas nos sistemas indica que a familiaridade com o sistema pode ser comprometida devido à evolução constante e complexidade crescente;
- Práticas Recomendadas: Utilizar ferramentas modernas de documentação ágil, treinamento contínuo e práticas colaborativas para garantir que a familiaridade com o sistema seja mantida.

A conservação da familiaridade é crucial para garantir que os desenvolvedores possam manter e evoluir o sistema de maneira eficaz. No contexto atual, a documentação contínua e ágil é preferida em relação à documentação detalhada tradicional. Ferramentas como Confluence, Notion e GitHub Wikis permitem a criação e manutenção de documentação de forma colaborativa e contínua, facilitando o acesso e a atualização por todos os membros da equipe (MURPHY, 2019). Além disso, a adoção de princípios de *clean code* e o uso de comentários estruturados no código são essenciais para complementar a documentação e facilitar a compreensão do código (LAPLANTE; KASSAB, 2022).

4.6 Lei 6: Crescimento Contínuo

Lições Aprendidas:

- Expansão Funcional: O crescimento nas métricas de linhas de código, *commits* e métodos por versão confirma que os sistemas evoluem continuamente para incorporar novas funcionalidades e atender às necessidades dos usuários;
- Práticas Recomendadas: planejar ciclos de desenvolvimento que incluam fases de expansão funcional, sempre alinhadas às necessidades e *feedback* dos usuários.

O crescimento contínuo do *software* é necessário para atender às necessidades dinâmicas dos usuários. Isso envolve a adição de novas funcionalidades, melhorias de desempenho e a correção de defeitos, o que, por sua vez, aumenta a complexidade do *software*. Para gerenciar esse crescimento de maneira eficaz, é fundamental adotar uma abordagem estruturada. Uma dessas abordagens é o GQM, que pode ser usada para definir metas claras de crescimento funcional (CALDIERA; ROMBACH, 1994).

4.7 Lei 7: Declínio da Qualidade

Lições Aprendidas:

- Declínio na Coesão: a análise das métricas de coesão LCOM e acoplamento CBO sugere que a qualidade do sistema pode declinar ao longo do tempo devido ao aumento da complexidade e redução da coesão;
- Práticas Recomendadas: implementar práticas de garantia de qualidade, como revisões de código e testes automatizados, para mitigar o declínio na qualidade conforme o sistema evolui.

O declínio da qualidade do *software* é um desafio contínuo. Conforme Baxter e Sommerville (2011), a implementação de práticas rigorosas de garantia de qualidade, incluindo revisões de código regulares e testes automatizados, é essencial para mitigar esse declínio. Ferramentas de integração contínua (CI) e entrega contínua (CD) também podem ajudar a manter altos padrões de qualidade.

4.8 Lei 8: Sistema de *Feedback*

Lições Aprendidas:

- *Loop* de *Feedback*: a interação das métricas de *issues*, mudanças incrementais e proporção de comentários para linhas de código evidencia a importância do *feedback* contínuo no processo de desenvolvimento e evolução dos sistemas;
- Práticas Recomendadas: estabelecer um ciclo de *feedback* estruturado com usuários e desenvolvedores, utilizando ferramentas de gestão de *issues* e monitoramento de métricas de código para orientar as mudanças e melhorias.

A implementação de um sistema de *feedback* eficaz é crucial para a evolução do *software*. Os desenvolvedores devem receber *feedback* contínuo dos usuários finais para garantir

que o *software* evolua de acordo com suas necessidades. Ferramentas como Jira e GitHub podem ser usadas para gerenciar e rastrear *feedback*, garantindo que ele seja incorporado de maneira estruturada e eficiente.

Diretrizes Práticas para Desenvolvedores e Pesquisadores:

1. Adaptação Contínua: implementar processos regulares de revisão e atualização de código, utilizando integração contínua e entrega contínua (CI/CD) para garantir que os sistemas se adaptem às novas necessidades e tecnologias emergentes.
2. Gestão da Complexidade: implementar práticas de refatoração contínua e utilizar ferramentas de análise de código, como *SonarQube*, para monitorar e reduzir a complexidade ao longo do desenvolvimento.
3. Auto-Regulação: estabelecer processos internos robustos que permitam ajustes e otimizações contínuas no sistema, como refatoração de código e realocação de recursos internos para áreas críticas.
4. Estabilidade Organizacional: manter um ritmo constante de desenvolvimento e estabelecer mecanismos de controle de versões, utilizando metodologias ágeis como *Scrum* e *Kanban* para gerenciar a estabilidade organizacional durante períodos de intensa atividade.
5. Manutenção da Familiaridade: utilizar ferramentas modernas de documentação ágil, como *Confluence, Notion e GitHub Wikis*, além de treinamento contínuo e adoção de *clean code*, para garantir que a familiaridade com o sistema seja mantida.
6. Crescimento Funcional: planejar ciclos de desenvolvimento que incluam fases de expansão funcional, sempre alinhadas às necessidades e *feedback* dos usuários, utilizando a abordagem GQM para definir metas claras de crescimento funcional.
7. Garantia de Qualidade: implementar práticas rigorosas de garantia de qualidade, como revisões de código e testes automatizados, utilizando ferramentas de integração contínua (CI) e entrega contínua (CD) para manter altos padrões de qualidade.
8. Feedback Contínuo: estabelecer um ciclo de *feedback* estruturado com usuários e desenvolvedores, utilizando ferramentas de gestão de *issues* como Jira e GitHub para monitorar e incorporar *feedback* de maneira estruturada e eficiente.

5 CONCLUSÕES E TRABALHOS FUTUROS

A evolução contínua de sistemas de *software* é um fenômeno inevitável e necessário para garantir que os sistemas permaneçam relevantes e atendam às demandas dos usuários e organizações. As Leis de Lehman, formuladas há décadas, continuam a oferecer uma base teórica sólida para entender essa evolução. Este estudo validou empiricamente a aplicabilidade dessas leis no contexto atual de desenvolvimento de *software*, utilizando métricas e análises dos três sistemas estudados.

Os resultados confirmam a hipótese formulada na introdução do trabalho: as Leis de Lehman são aplicáveis e relevantes para os sistemas de *software* modernos, fornecendo diretrizes para a gestão da evolução do *software*. Além disso, este estudo incluiu a sumarização das lições aprendidas e a criação de diretrizes práticas, que são essenciais para o desenvolvimento e manutenção eficaz de sistemas de *software*.

5.1 Contribuições do Estudo

Este estudo contribui para a compreensão da evolução de *software* e a aplicabilidade das Leis de Lehman no contexto moderno. Suas principais contribuições podem ser categorizadas em termos de avanços acadêmicos e benefícios práticos para a indústria de *software*. As contribuições são detalhadas a seguir:

1. **Validação empírica das Leis de Lehman**: este estudo fornece uma validação empírica das Leis de Lehman no cenário atual de desenvolvimento de *software*. Através da análise dos sistemas Dropwizard, K-9 Mail e LanguageTool e da aplicação de métricas relevantes, as conclusões obtidas oferecem uma perspectiva sobre a conformidade dessas leis em um contexto moderno;
2. **Utilização do *Framework* GQM**: a aplicação do *framework* GQM como base metodológica para a análise das Leis de Lehman oferece uma abordagem estruturada e sistemática para conduzir a pesquisa. Isso não apenas garante rigor na análise, mas também oferece um modelo replicável que pode ser adaptado para futuras investigações;
3. **Diretrizes práticas para melhoria contínua**: com base nas descobertas da análise quantitativa dos três sistemas de *software*, este estudo desenvolve diretrizes para o desenvolvimento e manutenção de *software*. As lições aprendidas

são transformadas em práticas recomendadas que podem ser implementadas para melhorar a adaptabilidade, reduzir a complexidade e garantir a qualidade contínua dos sistemas.

5.2 Limitações e Trabalhos Futuros

Embora este estudo tenha alcançado bons resultados, há sempre oportunidades para expandir e aprofundar a pesquisa. Algumas áreas que podem ser exploradas em trabalhos futuros incluem a realização de estudos adicionais em uma variedade maior de sistemas de *software*, abrangendo diferentes domínios de aplicação e escalas de projeto, para verificar a generalização dos resultados obtidos. Além disso, é importante investigar como fatores externos, como mudanças organizacionais, avanços tecnológicos e práticas de desenvolvimento, influenciam a evolução do *software* e a aplicabilidade das Leis de Lehman. Outra área promissora para futuras pesquisas é o desenvolvimento de ferramentas automatizadas para monitorar e analisar a evolução do *software* em conformidade com as Leis de Lehman, facilitando a aplicação das diretrizes práticas propostas neste estudo. Essas futuras investigações podem proporcionar uma compreensão mais abrangente da evolução do *software*, contribuindo para a melhoria contínua das práticas de desenvolvimento e manutenção.

REFERÊNCIAS

BAXTER, G.; SOMMERVILLE, I. Socio-technical systems: From design methods to systems engineering. **Interacting with computers**, Oxford, v. 23, n. 1, p. 4–17, 2011. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0953543810000652>. Acesso em: 10 mai. 2023.

BECK, K.; BEEDLE, M.; BENNEKUM, A. V.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R. *et al.* Manifesto for agile software development. Snowbird, UT, 2001. Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 20 nov. 2022.

BOSCH, J.; OLSSON, H. H.; CRNKOVIC, I. Engineering ai systems: A research agenda. **Artificial Intelligence Paradigms for Smart Cyber-Physical Systems**, IGI global, Ithaca, p. 1–19, 2021. Disponível em: <https://arxiv.org/abs/2101.07353>. Acesso em: 22 nov. 2023.

CALDIERA, V. R. B. G.; ROMBACH, H. D. The goal question metric approach. **Encyclopedia of software engineering**, John Wiley & Sons, New York, p. 528–532, 1994. Disponível em: <https://www.cs.umd.edu/~basili/publications/proceedings/P78.pdf>. Acesso em: 12 ago. 2022.

CAO, J.; WANG, X.; LI, Z.; GU, Q.; CHEN, Z. The evolution of open-source blockchain systems: An empirical study. In: **Proceedings of the 11th Asia-Pacific Symposium on Internetware**. New York, NY, USA: Association for Computing Machinery, 2019. (Internetware '19). ISBN 9781450377010. Disponível em: <https://doi.org/10.1145/3361242.3361248>. Acesso em: 20 nov. 2022.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. **IEEE Transactions on software engineering**, IEEE, Los Alamitos, CA, v. 20, n. 6, p. 476–493, 1994.

COOK, S.; HARRISON, R.; LEHMAN, M. M.; WERNICK, P. Evolution in software systems: foundations of the spe classification scheme. **Journal of Software Maintenance and Evolution: Research and Practice**, Chichester, v. 18, n. 1, p. 1–35, 2006. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.314>. Acesso em: 04 fev. 2023.

COSENTINO, V.; IZQUIERDO, J. L. C.; CABOT, J. A systematic mapping study of software development with github. **Ieee access**, IEEE, New York, v. 5, p. 7173–7192, 2017.

FAKHOURY, S.; ROY, D.; HASSAN, A.; ARNAOUDOVA, V. Improving source code readability: Theory and practice. In: **2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)**. Montreal, QC, Canada: [s.n.], 2019. p. 2–12.

FENTON, B.; NORMAN, J. **Software metrics: a rigorous and practical approach**. 3. ed. Boca Raton: CRC press, 2014.

FERREIRA, F.; VALE, G.; DINIZ, J. P.; FIGUEIREDO, E. Evaluating t-wise testing strategies in a community-wide dataset of configurable software systems. **Journal of Systems and Software**, New York, v. 179, p. 110990, 2021. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016412122100087X>. Acesso em: 13 abr. 2023.

FREUND, R. J.; WILSON, W. J.; SA, P. **Regression Analysis: Statistical Modeling of a Response Variable**. 2. ed. Burlington, MA: Academic Press, 2006.

GIRAY, G. A software engineering perspective on engineering machine learning systems: State of the art and challenges. **Journal of Systems and Software**, Elsevier, New York, v. 180, p. 111031, 2021.

GODFREY, G.; MICHAEL, D. M. The past, present, and future of software evolution. In: IEEE. **2008 Frontiers of Software Maintenance**. Beijing, China, 2008. p. 129–138.

HALL, T.; BEECHAM, S.; BOWES, D.; GRAY, D.; COUNSELL, S. A systematic literature review on fault prediction performance in software engineering. **IEEE Transactions on Software Engineering**, Los Alamitos, CA, v. 38, n. 6, p. 1276–1304, 2012.

HERRAIZ, I.; RODRIGUEZ, D.; ROBLES, G.; GONZALEZ-BARAHONA, J. M. The evolution of the laws of software evolution: A discussion based on a systematic literature review. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 46, n. 2, p. 1–28, 2013.

KOUR, G.; SINGH, P. Using lehman's laws to validate the software evolution of agile projects. In: IEEE. **2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)**. New Delhi, India, 2016. p. 90–96.

LAPLANTE, P. A.; KASSAB, M. **What every engineer should know about software engineering**. Boca Raton: CRC Press, 2022.

LEE, Y.; YANG, J.; CHANG, K. H. Metrics and evolution in open source software. In: IEEE. **Seventh International Conference on Quality Software (QSIC 2007)**. Portland, OR, USA, 2007. p. 191–197.

LEHMAN, M. Lifecycles and the laws of software evolution. **Proceedings of the IEEE, Special Issue on Software Engineering**, v. 19, p. 1060–1076, 1980.

LEHMAN, M. M. On understanding laws, evolution, and conservation in the large-program life cycle. **Journal of Systems and Software**, Elsevier, New York, v. 1, p. 213–221, 1979.

LEHMAN, M. M. Laws of software evolution revisited. In: **Proceedings of the 5th European Workshop on Software Process Technology**. Berlin, Heidelberg: Springer-Verlag, 1996. (EWSPT '96), p. 108–124.

LEHMAN, M. M.; RAMIL, J. F. Rules and tools for software evolution planning and management. **Annals of software engineering**, Springer, New York, v. 11, p. 15–44, 2001.

MOLLÉRI, J. S.; PETERSEN, K.; MENDES, E. Cerse-catalog for empirical research in software engineering: A systematic mapping study. **Information and Software Technology**, Elsevier, Amsterdam, v. 105, p. 117–149, 2019.

MURPHY, G. C. Beyond integrated development environments: adding context to software development. In: IEEE. **2019 IEEE/ACM 41st international conference on software engineering: new ideas and emerging results (ICSE-NIER)**. Montreal, QC, Canada, 2019. p. 73–76.

NUMMENMAA, T.; KULTIMA, A.; ALHA, K.; MIKKONEN, T. Applying lehman's laws to game evolution. In: **Proceedings of the 2013 International Workshop on Principles of Software Evolution**. New York, NY, USA: Association for Computing Machinery, 2013. (IWPSE 2013), p. 11–17. Disponível em: <https://doi.org/10.1145/2501543.2501546>. Acesso em: 03 out. 2023.

PRATAMA, M. R.; KUSUMO, D. S. Implementation of continuous integration and continuous delivery (ci/cd) on automatic performance testing. In: IEEE. **2021 9th International Conference on Information and Communication Technology (ICoICT)**. Yogyakarta, Indonesia, 2021. p. 230–235.

ROSENBERG, L. H.; STAPKO, R.; GALLO, A. Risk-based object oriented testing. NASA Goddard Space Flight Center, Greenbelt, 2000.

RUNESON, P.; HÖST, M. Guidelines for conducting and reporting case study research in software engineering. **Empirical software engineering**, Springer, Dordrecht, v. 14, p. 131–164, 2009.

SEJFIA, A. A pilot study on architecture and vulnerabilities: Lessons learned. In: **2019 IEEE/ACM 2nd International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)**. Montreal, QC, Canada: [s.n.], 2019. p. 42–47.

STOL, K.-J.; FITZGERALD, B. The abc of software engineering research. **ACM Transactions on Software Engineering and Methodology (TOSEM)**, ACM New York, NY, USA, v. 27, n. 3, p. 1–51, 2018.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. Berlin, Heidelberg: *Springer Science & Business Media*, 2012.