MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

CAP MARCELO EDUARDO MACHADO COTA

AVALIAÇÃO QUANTITATIVA DA DETERIORAÇÃO ARQUITETURAL NA EVOLUÇÃO DO SOFTWARE: UMA ABORDAGEM CONSIDERANDO COMPONENTES NÃO CONECTADOS ESTRUTURALMENTE

Rio de Janeiro
2018

INSTITUTO MILITAR DE ENGENHARIA

CAP MARCELO EDUARDO MACHADO COTA

AVALIAÇÃO QUANTITATIVA DA DETERIORAÇÃO ARQUITETURAL NA EVOLUÇÃO DO SOFTWARE: UMA ABORDAGEM CONSIDERANDO COMPONENTES NÃO CONECTADOS ESTRUTURALMENTE

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientador: Prof. Ricardo Choren Noya - D.Sc.

Rio de Janeiro
2018

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 - Praia Vermelha
Rio de Janeiro - RJ CEP 22290-270

005 Cota, Marcelo Eduardo Machado
C843a Avaliação quantitativa da deterioração arquitetural na evolução do software: uma abordagem considerando componentes não conectados estruturalmente / Marcelo Eduardo Machado Cota; orientado por Ricardo Choren Noya - Rio de Janeiro: Instituto Militar de Engenharia, 2018.

94p.: il.

Dissertação (Mestrado) - Instituto Militar de Engenharia, Rio de Janeiro, 2018.

1. Curso de Sistemas e Computação - teses e dissertações. 2. Arquitetura de Software. 3. Métrica. 4. Acoplamento Evolutivo. I. Noya, Ricardo Choren. II . Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

CAP MARCELO EDUARDO MACHADO COTA

AVALIAÇÃO QUANTITATIVA DA DETERIORAÇÃO ARQUITETURAL NA EVOLUÇÃO DO SOFTWARE: UMA ABORDAGEM CONSIDERANDO COMPONENTES NÃO CONECTADOS ESTRUTURALMENTE

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Sistemas e Computação.

Orientador: Prof. Ricardo Choren Noya - D.Sc.

Aprovada em 17 de Maio de 2018 pela seguinte Banca Examinadora:

Prof. Ricardo Choren Noya - D.Sc. do IME - Presidente

Prof. Ronaldo Ribeiro Goldschmidt - D.Sc. do IME

Prof. Toacy Cavalcante de Oliveira - D.Sc. da COPPE/UFRJ

Rio de Janeiro
2018

Dedico esta pesquisa aos meus filhos, Ísis e Victor, fonte de toda minha motivação. Espero que este trabalho também os incentive aos estudos durante toda a vida. Ao resgatarem o texto esquecido em uma gaveta ou arquivo digital, como se resgata uma foto antiga para recordação, que seja retribuído a eles todo estímulo que sempre me ofertaram.

AGRADECIMENTOS

Primeiramente, à Marinha do Brasil e ao Exército Brasileiro.

Principalmente, à minha esposa, Vânia, pela compreensão e ajuda irrestritas que permitiram a obtenção dos resultados aqui apresentados.

Ao Centro de Análises de Sistemas Navais, em especial ao Exmo. Contra-Almirante Alfredo Martins Muradas, Ex-Diretor, pela confiança depositada e ajuda incondicional que permitiram que eu fosse designado para realização deste curso de pós-graduação.

Aos familiares e amigos, em especial aos meus pais, por entenderem o afastamento devido aos estudos. À minha afilhada, Maria Eduarda, pela revisão do inglês. Ao grande amigo, Marcelo Valentim, pela materialização impressa deste trabalho.

Aos Comandantes: Rodrigo Abrunhosa Collazo, Pier-Giovanni Taranti e Marco Eugenio Madeira Di Beneditto, por me inspirarem, sendo exemplos de Oficiais que conciliaram com maestria a carreira operativa à capacitação nas áreas de computação aplicada.

Às Comandantes: Valéria Senna Toscano Muradas e Carla Patrícia Mello Lage, pela confiança e apoio institucional durante a realização do Mestrado.

À toda Equipe do Projeto SIPLOM, em especial ao Comandante Martins, Leandro Ouriques, Elci Lugão, Ricardo Sixel e Thiago Mayerle. Ao Eduardo Araújo, pelo apoio técnico irrestrito na configuração do ambiente de pesquisa. Todos, sempre dispostos a elucidar dúvidas técnicas ou administrativas que surgiram ao longo do curso.

Aos Mestres do Instituto Militar de Engenharia, em especial ao meu Professor Orientador, Dr. Ricardo Choren Noya, pelas instruções precisas, por sua cordialidade e pela disponibilidade de sempre.

“Não me venha com problemática que eu tenho a solucionática. ”

DARIO JOSÉ DOS SANTOS, O DADÁ MARAVILHA

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

CASNAV	Centro de Análises de Sistemas Navais
C2	Comando e Controle
CBO	Coupling Between Objects
CI	Cohesive Interactions
COC-MD	Centro de Operações Conjuntas do MD
DL	Decoupling Level
DSM	Design Structure Matrix
HCP	History Coupling Probability
HCPT	History Coupling Probability - Task Based
IAEv	Impacto do Acoplamento Evolutivo
JDK	Java Development Kit
LOC	Lines Of Code
MB	Marinha do Brasil
MD	Ministério da Defesa
NDD	Número de Desenvolvedores Distintos
NRP	Número de Revisões do Par (NRP) de Elementos
OO	Orientação a Objetos
PMD	"Estamos tentando descobrir o significado das letras PMD" (PMD, 2018)
RCI	Ratio of Cohesive Interactions
SCV	Sistema de Controle de Versões
SGP	Sistema de Gestão de Projetos
SHCP	Sliding Hystory Coupling Probability
SIPLOM	Sistema de Planejamento Operacional Militar
SMC2	Sistema Militar de Comando e Controle
SVN	Apache Subversion
WMC	Weighted Methods per Class

LISTA DE ABREVIATURAS E SÍMBOLOS

ABREVIATURAS

Ca	-	Coupling Afferent ou Acoplamento Aferente
Ce	-	Coupling Efferent ou Acoplamento Eferente
ComOpA	-	Comando Operacional Ativado
EO	-	Elementos Organizacionais
Fcte	-	Forças Componentes
JDeps	-	Java Class Dependency Analyzer
OpM	-	Operações Militares
TO	-	Teatro de Operações

SÍMBOLOS

ϑ	-	Número de Versões Consecutivas do Método SHCP

RESUMO

Mudanças tendem a degradar a estrutura do software, causando deterioração arquitetural. Partes do software podem apresentar dependências ocultas, denominadas de acoplamento evolutivo, que violam seus princípios arquiteturais projetados. Revelar essas dependências ocultas é importante porque favorece a avaliação de qualidade do código e indica a necessidade de refatoração antes que o pagamento da dívida arquitetural seja impraticável. O principal objetivo deste estudo é definir uma métrica direta para medir a deterioração arquitetural de um software, usando informações sobre o acoplamento evolutivo. A ideia é determinar quanto cada elemento arquitetural avaliado está logicamente acoplado a outros elementos e comparar os resultados com instâncias reais de acoplamento evolutivo identificadas pela nova abordagem da verificação deslizante. Técnicas têm sido propostas para avaliar a dívida arquitetural usando dependências lógicas, tal como o History Coupling Probability (HCP). No entanto, detectar o acoplamento evolutivo é uma tarefa difícil, pois requer a confirmação se, de fato, as mudanças conjuntas representam uma dependência lógica adquirida. Uma solução para esse problema é usar dados obtidos de um sistema de controle de versão no qual todas as alterações de um componente podem ser visualizadas no nível do sistema. Assim, este trabalho propõe uma variação do método HCP, integrando informações obtidas do histórico de versões com informações gerenciais sobre tarefas para melhor identificar e qualificar as mudanças conjuntas. Depois de classificar o acoplamento evolutivo, é possível medir seu impacto usando a métrica proposta. A metodologia foi aplicada a um grande cenário de evolução de um Sistema Militar de Comando e Controle (SMC2), desenvolvido pela Marinha do Brasil (MB). Os resultados mostram que a nova métrica indica o nível de decaimento da arquitetura do software e que sua medida é sensível à refatoração, indicando variação de qualidade no momento de intervenções nos elementos arquiteturais envolvidos na evolução do software. Assim, os resultados indicam que o IAEv é uma métrica promissora para apontar áreas na arquitetura de software que tiveram impacto devido ao acoplamento evolutivo, ajudando equipes de desenvolvedores, arquitetos ou gerentes de projeto a procurar possíveis demandas de melhorias no software e implementá-las imediatamente.

ABSTRACT

Changes to the software tend to degrade its structure, causing architectural decay. Parts of the software may present hidden dependencies, called evolutionary coupling, that violate its designed architectural principles. Unveiling these hidden dependencies is important because it may support the quality verification and indicate the need for refactoring before architectural debt repayment is impractical. The main goal of this study is to define a direct metric to measure the architectural decay of a software, using information about evolutionary coupling. The idea is to determine how much each evaluated architectural element is logically coupled with other elements and to compare the results with real instances of evolutionary coupling identified by the new sliding verification approach. Techniques have been proposed to evaluate the architectural debt using logical dependencies, such as History Coupling Probability (HCP). However, detecting evolutionary coupling is a hard task since it is difficult to confirm if, indeed, co-changes are logical dependency acquired. A workaround to this problem is to use data obtained from a version control system in which all changes of a component can be viewed on the system level. Thus, this work proposes a variation to the HCP method, integrating information obtained from the version history with managerial information about tasks to better identify and qualify the cochanges. After classifying evolutionary coupling, it is possible to measure its impact, using the proposed metric. The metodology was applied to a large Military Command and Control System evolution scenario, developed by the Brazilian Navy. The results shown that the new metric indicates the software architectural decay level and that its measurement is sensitive to refactoring, indicating quality variation at the moment of intervention in the architectural elements involved in software evolution. Thus, the results indicate that IAEv is a promising metric to point out areas in the software architecture that have impacted by evolutionary coupling, helping teams of the developers, architects or project managers look for possible demands for improvements to the software and implement them immediately.

1 INTRODUÇÃO

A comparação de indicadores com o mundo real permite a percepção dos efeitos relacionados aos resultados observados, embora não seja condição determinante para o diagnóstico preciso em uma amostra particular. Mesmo assim, diversos setores fazem uso de métricas para segmentar um conjunto prévio que, posteriormente, pode passar por exames detalhados para confirmação da indicação inicial. O termo métrica relacionado ao desenvolvimento de sistemas foi apresentado por Swanson (1976) como uma medida de extensão ou grau em que um produto (código-fonte) possui e exibe certas características (de qualidade).

Mais especificamente, as métricas arquiteturais (ABOWD et al., 1997) foram descritas como interpretações quantitativas colocadas em medidas particulares observáveis na arquitetura do software. Arquitetura de software é, segundo Parnas (1972), um mecanismo de ocultação de informações como meio de decomposição do sistema em alto nível (divisão do sistema em conjuntos de módulos) para melhorar a flexibilidade e a compreensão do mesmo, enquanto permite o encurtamento do seu tempo de desenvolvimento. De acordo com Garlan e Shaw (1993), a arquitetura de um software é seu nível de projeto, incluindo linguagens de interconexão dos módulos, modelos e estruturas para sistemas que atendem às necessidades de domínios específicos e modelos formais de mecanismos de integração entre componentes. Desta forma, a arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, o que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles (BASS et al., 2002).

As relações entre os componentes de software, normalmente, se referem às dependências estruturais, que podem ser identificadas no código-fonte ou via projeto de software. No entanto, existe outro tipo de dependência denominada dependência lógica. As dependências lógicas (GALL et al., 1998), também denominadas de acoplamento evolutivo (ZIMMERMANN et al., 2005; XIAO et al., 2014a), são um tipo especial de acoplamento, observáveis a partir do histórico de revisões e que ocorrem durante a evolução dos sistemas de software. De acordo com Stevens et al. (1974), o acoplamento é a medida da força de associação estabelecida por uma conexão entre módulos.

O efeito da dependência lógica na arquitetura de software pode indicar uma variação de qualidade, pois uma alta incidência deste tipo oculto de acoplamento diminui a proba-

bilidade de qualquer tipo de intervenção no código ficar limitada a uma alteração pontual. Isso, segundo Mondal et al. (2017), aumenta a complexidade e o esforço da manutenção do código, degradando a estrutura inicialmente projetada.

Esse fenômeno, que se apresenta quando a arquitetura concreta (como construída) de um sistema de software se desvia de sua arquitetura conceitual (como planejada), é denominado deterioração arquitetural por Riaz et al. (2009), sendo causado por intervenções no software ao longo do seu ciclo de vida. Parnas (1972) compara a deterioração de um sistema ao envelhecimento humano e argumenta que este problema é introduzido no software quando alterações são realizadas na estrutura de um programa por pessoas que não entendem o conceito original do projeto.

A refatoração de uma arquitetura mal projetada, no sentido de melhorar a qualidade do software, também foi considerada no escopo desta pesquisa. Neste sentido, o termo deterioração indica o desvio do conceito original e, por este motivo, essa avaliação deve ser encarada como um indicador de qualidade que permite a análise tanto do decaimento quanto de melhorias na estrutura do sistema. Quanto mais acoplado um módulo está, mais difícil será entender, mudar ou corrigi-lo, devido às fortes relações estabelecidas com outros módulos. A complexidade pode ser reduzida ao projetar sistemas com o acoplamento mais fraco possível entre os módulos.

Nos últimos anos, diversos esforços vêm sendo envidados para estabelecer métodos de avaliação quantitativa da deterioração arquitetural (MEDVIDOVIC; TAYLOR, 2010; RAJLICH, 2014; LE et al., 2016; NORD et al., 2012; ERNST et al., 2015; MACCORMACK et al., 2017). Estes trabalhos combinam métricas e avaliam as propriedades de qualidade arquiteturais impactadas a partir de análises quantitativas e/ou qualitativas, contudo não levam em consideração o acoplamento evolutivo.

Trabalhos recentes (MO et al., 2015, 2016; CAI; KAZMAN, 2016; XIAO et al., 2016) utilizam o acoplamento evolutivo para determinar deterioração arquitetural, mas de forma indireta. Investigando a relação entre a estrutura de dependências e os defeitos de software, Mo et al. (2015) visam a detecção automática dos *bad smells* (FOWLER; BECK, 1999), combinando o acoplamento evolutivo com informações de dependências estruturais, de modo a identificar os arquivos que são mais propensos a apresentar certos defeitos na arquitetura de software.

Nesta mesma direção, Mo et al. (2016) apresentam a métrica *Decoupling Level* (DL), que avalia a manutenibilidade do software a partir do nível de desacoplamento da sua arquitetura, ou seja, quão bem o sistema pode ser decomposto em módulos menores e independentemente substituíveis.

Cai e Kazman (2016) monitoram a arquitetura de software para identificar e quantificar as mudanças no nível geral de manutenção de um projeto. Já Xiao et al. (2016) utilizam o acoplamento evolutivo para quantificar a dívida arquitetural, um tipo específico de dívida técnica.

Entretanto foi observada a carência de métrica direta, baseada em um modelo de elementos arquiteturais e suas características de mudanças conjuntas, que estabeleça critérios para medir a deterioração arquitetural apenas com informações acerca do acoplamento evolutivo. Por exemplo, Mo et al. (2016) atestam que uma variação não trivial da DL indica uma maior degradação da arquitetura ou uma melhoria. No entanto, não são estabelecidos limites que indiquem diretamente a deterioração, pois o objetivo primário da medida é identificar módulos da arquitetura que estão ou não desacoplados. Neste caso, é necessário correlacionar indiretamente o fato do sistema estar fracamente ou fortemente acoplado a um baixo ou alto nível de deterioração arquitetural, respectivamente.

Fenton e Bieman (2014) destacam que toda ação de medição deve ser motivada por objetivo e necessidade específica, claramente definidos e facilmente compreensíveis. Deste modo, os benefícios para diagnosticar o problema, localizar o defeito, divulgar uma solução alternativa e reparar a falha não pode superar os custos do processo de medição. Fenton e Bieman (2014) ainda destacam que a medição direta constitui a forma natural para buscar o entendimento sobre as entidades e os atributos que elas possuem. Por essas razões, a métrica direta é vantajosa, pois elimina o esforço exigido por métodos que requerem modelos adicionais ou padrões de defeitos para inferir sobre a qualidade do software.

Pelo exposto, este estudo estabelecerá uma métrica direta para medir o impacto do acoplamento evolutivo na arquitetura de software, sem que sejam necessários modelos adicionais ou padrões de defeitos para inferir sobre a deterioração arquitetural ou melhoria da qualidade da arquitetura de software. Não obstante, o modelo de medição direta não é restritivo, ou seja, podem ser aplicadas medições adicionais para tornar essa avaliação mais precisa, usando outras medidas indiretas.

1.1 OBJETIVO E CONTRIBUIÇÕES ESPERADAS

O objetivo primário deste trabalho é definir uma métrica direta para avaliação quantitativa da deterioração arquitetural, utilizando informações acerca das características das mudanças conjuntas entre pares de elementos não conectados estruturalmente para determinar o acoplamento evolutivo. Para tanto, este trabalho propõe, ainda, uma metodologia de classificação das alterações conjuntas que ocorrem no nível arquitetural de um sistema

e detalha uma abordagem para a detecção do acoplamento evolutivo, utilizando o conceito da verificação deslizante. Desta forma, as contribuições esperadas por este trabalho são:

(i) uma proposta de abordagem para melhor detectar o acoplamento evolutivo, aplicando o conceito de verificação deslizante entre versões subsequentes dos sistemas de software;

(ii) classificação das mudanças conjuntas entre elementos arquiteturais não conectados estruturalmente, considerando as características de tarefa para ampliar o escopo da detecção do acoplamento evolutivo; e

(iii) uma métrica para quantificar diretamente o impacto do acoplamento evolutivo e auxiliar na identificação da deterioração arquitetural ao longo do processo de evolução dos sistemas de software.

A análise do acoplamento evolutivo, considerando uma série contínua de versões, permite uma melhor compreensão dos problemas arquiteturais ao longo da evolução do software, pois avalia traços desse tipo de deterioração arquitetural, mesmo quando os elementos com dependência lógica estabelecida anteriormente não sofrem alterações em uma determinada versão. Com isso, também é possível minimizar falsos positivos, causados principalmente pela alta instabilidade das versões iniciais, quando novas funcionalidades são adicionadas ao sistema. Ainda é possível identificar as correções de problemas causados devido ao acoplamento evolutivo de modo a verificar se permanecem impactando a arquitetura ao longo do tempo, facilitando a análise da refatoração de código.

A classificação do acoplamento evolutivo auxilia atividades relacionadas à correção de defeitos e contextualiza melhor as mudanças ao avaliar as características que induziram as revisões de código para cada par de elementos. Mesmo que não sejam observadas mudanças simultâneas para um par de elementos em uma mesma transação, eles serão considerados conectados logicamente, se existirem características correspondentes. Com isso, informações desprezadas por métodos atuais são consideradas e, por essa razão, a metodologia amplia o escopo de detecção. Adicionalmente, são analisadas transações com um único elemento modificado, combinando essa com outras transações que atendam à mesma tarefa.

A definição de uma métrica para quantificar diretamente o acoplamento evolutivo permite o cálculo da probabilidade e do impacto (fator de risco) para o elemento arquitetural individualmente. Assim, o acoplamento evolutivo é estabelecido por elemento, em cada versão do software avaliada. Essa nova medida, ainda possibilita a seleção de

elementos arquiteturais comprometidos pelo acoplamento evolutivo e garante o monitoramento contínuo da deterioração de cada elemento ao longo da evolução do software. Dessa forma, se no momento da refatoração de código, correção de defeitos ou adição de funcionalidades, a métrica se sensibilizar ao risco derivado do acoplamento evolutivo, os desenvolvedores, arquitetos ou gerentes de projeto passarão a contar com uma ferramenta de apoio à tomada de decisão que indica a variação de qualidade no software.

1.2 ORGANIZAÇÃO DO TEXTO

O restante deste trabalho está estruturado da seguinte forma: o Capítulo 2 apresenta o referencial teórico da dissertação, que contempla assuntos relativos à arquitetura de software, manutenção arquitetural, deterioração arquitetural, avaliação da deterioração arquitetural, acoplamento evolutivo e probabilidade de acoplamento evolutivo; o Capítulo 3 apresenta o passo a passo da metodologia de classificação das mudanças conjuntas, visando a avaliação quantitativa do impacto do acoplamento evolutivo na deterioração arquitetural; o Capítulo 4 apresenta a prova de conceito, onde o passo a passo é aplicado em um sistema de estudo de caso que possibilitou a obtenção de resultados práticos usados para validação da abordagem proposta; o Capítulo 5 discute os resultados alcançados; o Capítulo 6 apresenta os trabalhos relacionados a este estudo; o Capítulo 7 conclui, listando as contribuições alcançadas e indicando trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta tópicos necessários para o entendimento dos conceitos abordados ao longo do texto, de acordo com a temática da pesquisa.

2.1 ARQUITETURA DE SOFTWARE

De acordo com Bass et al. (2002), no coração da arquitetura de software está o princípio da abstração: ocultar alguns detalhes de um sistema através do encapsulamento, permitindo, assim, melhor identificar e sustentar suas propriedades. Segundo Clements et al. (2002), a arquitetura faz os conjuntos de peças trabalharem juntos como um todo coerente e bem-sucedido. A documentação arquitetural auxilia os arquitetos a tomarem as decisões corretas, orienta os desenvolvedores em como executar tais decisões e, ainda, registra essas decisões para fornecer aos futuros responsáveis pelo sistema uma visão da solução do projeto.

Para Bass et al. (2002), a estrutura de uma arquitetura de software pode ser dividida em três grupos, dependendo da natureza ampla dos elementos que se deseja apresentar:

- Estrutura de Módulos: refere-se às unidades de implementação, baseadas em código e delimitadas por áreas de responsabilidade funcional;
- Estrutura de Componentes e Conectores: componentes são os principais elementos de computação e conectores são os veículos de comunicação entre eles;
- Estrutura de Alocação: refere-se às relações entre elementos de software em um ou mais ambientes externos nos quais o software é criado e executado.

A Figura 2.1 apresenta os elementos arquiteturais descritos por uma linguagem formal de descrição de arquitetura - *"Architecture Description Languages (ADL)"* - que fornece uma estrutura conceitual e uma sintaxe concreta para a modelagem de arquiteturas de software. Os componentes representam elementos computacionais e armazenamentos de dados de um sistema. Um componente pode ter múltiplas interfaces, cada uma das quais é denominada porta. Os conectores representam as interações entre os componentes e possuem interfaces definidas por um conjunto de regras. Cada regra de um conector define um participante da interação representada pelo conector. Os sistemas são definidos

como gráficos nos quais os losangos representam componentes e as setas representam os conectores que estabelecem as dependências formais entre esses componentes.

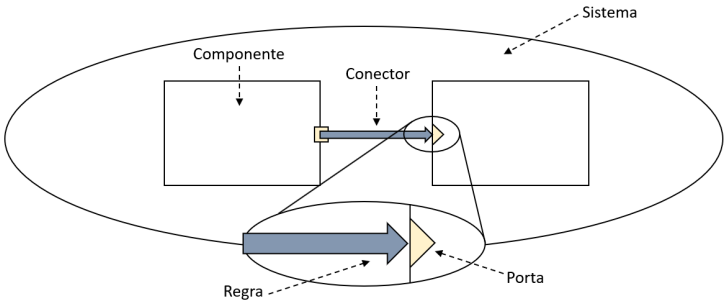

FIG. 2.1: Elementos da arquitetura descritos pela ADL ACME. (GARLAN et al., 2000).

Componentes conectados estruturalmente são aqueles que possuem conectores, representando as interações entre os elementos. Essa relação estrutural pode ser identificada na documentação ou em código-fonte, como uma representação física da estrutura conceitual observável no código-fonte ou em modelos de linguagens formais de descrição de arquitetura de software. Por outro lado, componentes não conectados estruturalmente são aqueles que não possuem dependências entre si observáveis no código fonte ou por meio de conectores na representação formal da arquitetura de software.

Clements et al. (2002) ainda acrescentam que somente criar uma arquitetura robusta não é suficiente. É necessário manter sua descrição atualizada com detalhes suficientes, sem ambiguidade, e organizada para que outras pessoas possam encontrar rapidamente as informações necessárias. Assim, manter o desenvolvimento do software adequado com o projeto arquitetural é condição indispensável para se conseguir um produto de alta qualidade, de maneira previsível e com o mínimo de retrabalho possível.

2.2 MANUTENÇÃO ARQUITETURAL

De acordo Clements et al. (2002), sempre que há múltiplas representações de um sistema há o problema de manter essas representações consistentes, sejam os modelos de projeto, a arquitetura ou o código-fonte. A representação que é mantida se torna a correta e as demais se degradam ao longo do tempo. Se não houver um forte acoplamento entre a arquitetura e o código-fonte, então surgem dois problemas: o primeiro é traduzir a especificação arquitetural para código, uma vez que o projeto deve preceder a codificação; o segundo é manter a arquitetura atualizada em face da evolução do sistema, já que o código, e não a arquitetura, geralmente se torna a representação mais atualizada.

A tipologia das atividades de manutenção de software foi introduzida por Swanson (1976), que classificou a atividade de manutenção em corretiva, adaptativa e preventiva. Outros estudos (CHAPIN et al., 2001; BUCKLEY et al., 2005; RAJLICH, 2014) expandiram essa tipologia e discutiram sua classificação no processo de desenvolvimento de sistemas. Rajlich (2014) considera necessário para o entendimento de um sistema de software a compreensão do domínio da aplicação, da sua arquitetura, dos algoritmos e das estruturas de dados. Dessa forma, a Manutenção Arquitetural compreende a avaliação das atividades que modificam a arquitetura de software e a análise das condições estruturais do sistema antes e após a realização das mudanças.

Embora os termos **evolução** e **manutenção** sejam frequentemente empregados para remeter às modificações no software, de acordo com Godfrey e German (2008), os dois tipos de intervenção possuem importantes diferenças semânticas. Evolução indica uma adaptação do sistema para uso em novos ambientes, adicionando novos recursos ou melhorando o projeto interno para atender insatisfações com a versão atual. Manutenção é direcionada para a preservação do software existente, corrigindo erros e indica a ideia de manter um sistema existente em execução sem alterar seu projeto.

Assim, mudanças são representativas da manutenção e da evolução do software e cada mudança pode introduzir um novo recurso (ou propriedade) no sistema, corrigir erros ou atualizar seu ambiente de execução. As mudanças conjuntas ou *co-changes* consistem em toda atualização de arquivo, verificável em um Sistema de Controle de Versões (SCV), que ocorram ao mesmo tempo, conforme apresentado na Figura 2.2. Para Ball et al. (1997), uma alta frequência de mudanças conjuntas serve como um indicador de problemas arquiteturais ou de modularidade. Por essa razão, a avaliação das mudanças conjuntas, quando se trata de analisar a manutenção em nível arquitetural, é importante.

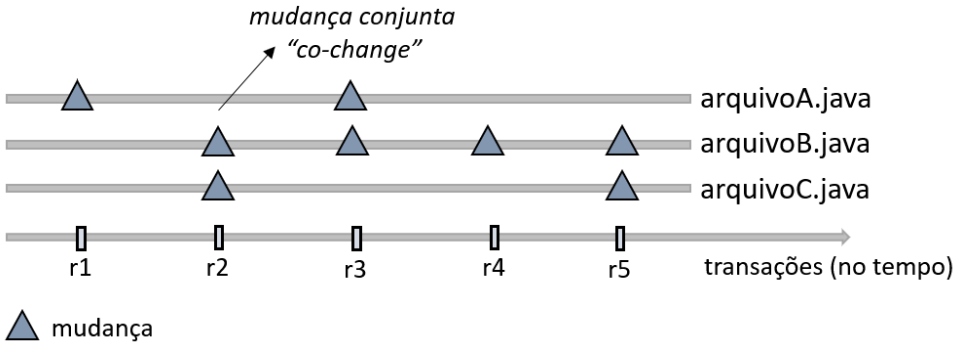

FIG. 2.2: Linha do tempo de mudanças sob a ótica de um SCV. Os triângulos representam as mudanças nos arquivos ao longo do tempo. Na transação r3, por exemplo, os arquivos A e B mudam juntos (*co-change*).

A Figura 2.2 ilustra o conceito de atomicidade nas operações que tornam permanentes na base do SCV um conjunto de mudanças experimentais em arquivos de código-fonte que pode ser recuperado para análise. Cada revisão de código pode impactar múltiplos arquivos, porém para um SCV cada revisão é originada a partir de uma única transação realizada ao longo do tempo. Ao longo deste estudo, o termo "revisão" será empregado para referir-se às mudanças em um ou em vários arquivos de código-fonte e "transação" para referir-se a cada operação atômica (*commit*) do SCV, que identifica as revisões de código executadas pelos desenvolvedores.

Kirbas et al. (2017) apresentam critérios de classificação das mudanças conjuntas para melhorar o processo de manutenção da arquitetura de software a partir do acoplamento evolutivo. Os autores empregam a teoria da medição e princípios de metrologia para o desenvolvimento de 19 (dezenove) critérios distintos, além de classificarem as mudanças conjuntas de acordo com as 7 (sete) características seguintes:

(1) **Características de Agrupamento:** A medição do acoplamento evolutivo começa com medição direta de mudanças em pares de elementos arquiteturais. Identificar como esses pares podem ser agrupados fará diferença para a métrica proposta. Dentre as abordagens de agrupamento destacadas por Kirbas et al. (2017), estão pares baseados em solicitações de mudança ou tarefas de desenvolvimento de novos requisitos.

(2) **Características de Localidade:** A distância entre os elementos que estão sendo modificados conjuntamente pode impactar uma métrica que utilize informações acerca do acoplamento evolutivo. Segundo Kirbas et al. (2017), mudanças conjuntas que envolvem arquivos de diferentes subsistemas são mais propensos a apresentar defeitos que aquelas que envolvem arquivos do mesmo subsistema.

(3) **Características da Mudança:** Segundo Kirbas et al. (2017), o motivo para a realização da mudança pode impactar uma métrica que utilize informações acerca do acoplamento evolutivo. Em particular, uma mudança representa uma correção de defeito ou o desenvolvimento de uma nova funcionalidade.

(4) **Características do Desenvolvedor:** Segundo Kirbas et al. (2017), modelos de métricas de acoplamento evolutivo que consideram o número de desenvolvedores distintos (NDC) são significativamente melhores que os modelos sem considerar essa informação. Estudos avaliados pelos autores indicam que quanto maior o NDC, maior as chances das mudanças conjuntas refletirem acoplamento evolutivo.

(5) **Características Temporais:** Segundo Kirbas et al. (2017), as características temporais podem afetar a acurácia da detecção do acoplamento evolutivo.

(6) **Características do Tipo da Mudança:** Existem três tipos básicos de mudança: adição, modificação e exclusão. Segundo Kirbas et al. (2017), cada um desses tipos pode impactar uma métrica que utilize informações acerca do acoplamento evolutivo.

(7) **Características do Tamanho da Mudança:** Segundo Kirbas et al. (2017), não é provável que grandes mudanças tenham o mesmo impacto na métrica do que pequenas mudanças. Os autores destacam que a avaliação do tamanho da mudança pode ser realizado com o auxílio de outras métricas.

2.3 DETERIORAÇÃO ARQUITETURAL

O projeto arquitetural serve como guia do desenvolvimento do software, mas durante as atividades de manutenção, mesmo que o desenvolvedor ainda mantenha o projeto em mente, esse não está em busca da solução (pois esta já foi encontrada), e sim de uma melhor solução que pode se diferir do que fora originalmente projetado (FOWLER; BECK, 1999). Porém, na maioria das vezes, as mudanças no sistema são feitas sob restrições de cronograma e orçamento. Assim, os desenvolvedores não têm tempo para entender a arquitetura e selecionar a melhor maneira de implementar as alterações (TVEDT et al., 2002). Ao longo das manutenções, a arquitetura de um sistema de software pode divergir da intenção original, tornando as mudanças mais difíceis de se implementar do que deveria ocorrer, causando deterioração arquitetural (MEDVIDOVIC; TAYLOR, 2010).

As decisões de projeto e desenvolvimento relacionadas às mudanças podem comprometer a qualidade do sistema se forem feitas sem uma análise cuidadosa sobre o impacto na arquitetura do software (EICK et al., 2001; HERRAIZ et al., 2013; LE et al., 2016; PERRY; WOLF, 1992; RAJLICH, 2014). Rajlich (2014) observa que desenvolvedores adicionam novos recursos, corrigem erros anteriores ou mal interpretados e reagem aos novos requisitos, tecnologias e à volatilidade de conhecimento ao longo do tempo, passando por manutenções e evoluções.

Deterioração arquitetural é o fenômeno que se apresenta quando a arquitetura concreta (como construída) de um sistema de software se desvia de sua arquitetura conceitual (como planejada), onde essa não mais satisfaz os principais atributos de qualidade que nortearam seu projeto ou quando a arquitetura inviabiliza novas mudanças, devido ao fato de mudanças já introduzidas no sistema ao longo do tempo tornarem sua manuten-

ção insustentável (RIAZ et al., 2009). A presença de deterioração em uma arquitetura também aumenta o tempo gasto para se manter o software (TVEDT et al., 2002).

A Figura 2.3 representa, de forma abstrata, alguns elementos arquiteturais e suas relações estabelecidas conceitualmente no projeto de software (A0) e a arquitetura resultante (A1 e A2) após implementação de duas versões subsequentes do software (V1 e V2). É possível notar que em A1 as relações entre os elementos se diferem da intenção original do arquiteto. Em A2, além das relações divergentes, um novo elemento é adicionado ao modelo, distanciando a arquitetura concreta do que fora inicialmente planejado.

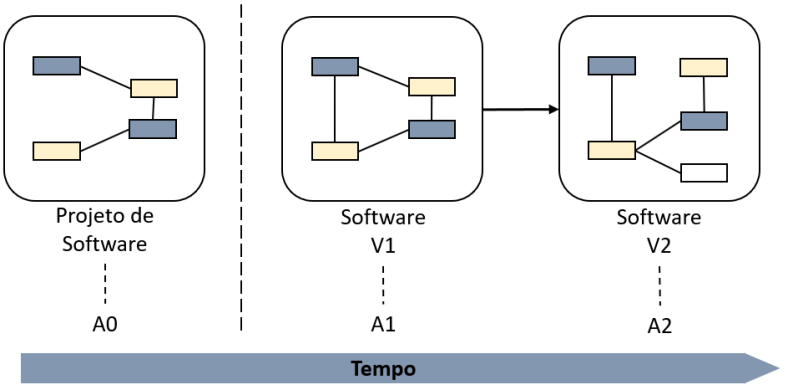

FIG. 2.3: Exemplo de deterioração arquitetural durante a evolução do software. Para cada versão do software (V1 e V2) as arquiteturas construídas (A1 e A2) se distanciam do conceito originalmente estabelecido (A0).

Estabelecer a distância estrutural entre versões da arquitetura de software é uma estratégia empregada para quantificar a deterioração arquitetural. Baseado nesse conceito, segundo o exemplo da Figura 2.3, é possível quantificar a diferença entre a arquitetura resultante (A1) da primeira versão (V1) do que fora inicialmente planejado (A0) e assim por diante. Basili e Nakamura (2005) comparam arquiteturas de software ao longo do processo de evolução e quantificam as diferenças de arquitetura para arquitetura, propondo uma métrica de distância estrutural baseada em grafos. Li et al. (2016) aplicam métricas de avaliação arquitetural existentes para estimar a qualidade em cada uma das versões do software e obter a distância entre as versões avaliadas.

Porém, no processo de deterioração arquitetural, elementos que originalmente são independentes podem passar a se relacionar logicamente com outros elementos. Contudo, essa relação lógica adquirida não é observável via documentação da arquitetura ou simplesmente analisando as dependências estruturais do código-fonte. No entanto, segundo Ball et al. (1997) existe uma maneira de descobrir essas dependências lógicas a partir de dados obtidos de um SCV.

2.4 AVALIAÇÃO DA DETERIORAÇÃO ARQUITETURAL

O processo de avaliação da arquitetura foi definido como a verificação de não conformidades entre a arquitetura descritiva e a prescritiva (CLEMENTS et al., 2002), ou seja, a análise do desvio do projeto original e do aumento da sua resistência à mudança (PERRY; WOLF, 1992).

Lindvall et al. (2002) atestam que uma avaliação da arquitetura de software pode ocorrer em diferentes momentos do ciclo de vida e com objetivos distintos. É possível distinguir a avaliação em antecipada e tardia. A avaliação antecipada é empregada para avaliar uma ou mais arquiteturas de software candidatas, que ainda não estão implementadas. A avaliação tardia pode utilizar dados medidos na implementação de software atual para quantificar a deterioração arquitetural.

Roy e Graham (2008) afirmam que no processo de avaliação de arquitetura pode se utilizar da documentação de arquitetura de software, registros de correção de defeitos e a especificação de requisitos como entradas. Eick et al. (2001) também atestam que os índices de deterioração do código podem ser quantificados e observados a partir da base de dados de gerenciamento de versões. Complementarmente, Tvedt et al. (2004) destacam que as métricas podem ser utilizadas em avaliações de arquitetura para fornecer uma noção de acoplamentos estáticos entre módulos e elementos que compõem os módulos, a partir de detecção no código.

Li et al. (2016) afirmam que, em geral, há um número grande de maneiras para avaliar uma arquitetura de software, algumas baseadas em documentação e outras em código-fonte. Diversas métricas utilizam modelos para analisar as mudanças de estrutura lógica interna e os atributos de qualidade externos para detectar sua relação no nível arquitetural. Os autores também destacam que novos processos e métricas de evolução podem ser aplicados à Arquitetura de Software, principalmente aqueles que não se concentram apenas em mudanças dos atributos de qualidade, mas que avaliam a mudança estrutural, podendo explicar melhor como a arquitetura de software deteriora.

Land (2002) afirma que não há tantas medidas propostas no nível arquitetural para medir a manutenibilidade do sistema, porém ressalta que a característica mais óbvia a ser investigada é a interdependência entre seus componentes. Para Eick et al. (2001), os índices de deterioração do código podem ser quantificados e interpretados como sintomas quantificados, fatores de risco ou prognósticos, que funcionam como preditores de respostas (custo, intervalo e qualidade). De acordo com Kirbas et al. (2017), embora tenha havido esforços para estabelecer uma base teórica para métricas de software (ZUSE,

1998; BRIAND et al., 1996; ABRAN, 2010), não é possível garantir que esses princípios de medição possam ser empregados como métricas de acoplamento evolutivo.

Uma métrica considerada por Le et al. (2016) para avaliação arquitetural foi a *Instability* (MARTIN, 1994). De acordo com Martin (1994), as categorias com maior estabilidade são as que são independentes e altamente responsáveis, ou seja, tendem a ser estáveis, embora qualquer mudança tenha um grande impacto. Dependências em categorias estáveis são consideradas "boas" dependências. Assim, o objetivo da *Instability* é evidenciar os componentes que são mais propensos a apresentar instabilidade, avaliando suas dependências estruturais em um nível de categorias de elementos (classes, componentes ou pacotes, por exemplo), auxiliando no gerenciamento das dependências entre essas categorias, de acordo com a Equação 2.1.

$$I = \frac{Ce}{(Ca + Ce)} \tag{2.1}$$

- **Acoplamento Aferente (Ca):** São somadas as classes fora da categoria que dependem de classes de dentro da categoria.

- **Acoplamento Eferente (Ce):** São somadas as classes dentro da categoria que dependem de classes fora da categoria.

O trabalho de Le et al. (2016) também considera a métrica *Ratio of Cohesive Interactions* (RCI) (BRIAND et al., 1993) para auxiliar na avaliação da deterioração arquitetural. De acordo com Briand et al. (1993), a RCI trabalha em termos de *Cohesive Interactions* (CI), que avaliam o conjunto de serviços fornecidos por um módulo e quão estreitos são os relacionamentos entre as declarações de dados dentro de uma especificação dos módulos e entre as declarações de dados e os subprogramas declarados. A RCI (Equação 2.2) é capaz de quantificar o número de dependências em um sistema (conhecidas) em comparação com o número máximo de dependências possíveis (potencial).

$$RCI = \frac{\#CIconhecidas}{\#CIpotencial} \tag{2.2}$$

Métricas diretas de um atributo ou entidade não envolvem outros atributos ou entidades. Métricas derivadas ou indiretas são frequentemente utilizadas para tornar visíveis as interações entre as medições diretas. Ou seja, às vezes é mais fácil ver o que está acontecendo em um projeto usando combinações de medidas, uma vez que muitos atributos só podem ser medidos indiretamente (FENTON; BIEMAN, 2014).

2.5 ACOPLAMENTO EVOLUTIVO

Acoplamento histórico (BALL et al., 1997), dependências lógicas (GALL et al., 1998) e acoplamento evolutivo (XIAO et al., 2014a) são sinônimos. Caracterizam-se como um tipo de dependência oculta não evidente no código-fonte e que pode ser empregada para identificar módulos candidatos a refatoração, sendo detectável a partir de padrões de mudanças conjuntas, observáveis nos relatórios de alteração de código.

A probabilidade de dois arquivos serem modificados juntos foi abordada pela primeira vez por Ball et al. (1997), que concluíram que os SCV possuem muitos dados importantes que podem ser explorados no estudo da evolução de sistemas. A pesquisa realizada por Ball et al. (1997) investigou os efeitos das decisões de desenvolvimento na evolução de sistemas de software. Gall et al. (1998) mostraram que é possível obter informações úteis sobre a arquitetura do sistema ao investigarem elementos que sempre são alterados juntos em uma revisão de código, permitindo identificar quais deles são candidatos a refatoração.

Para a identificação de padrões de mudança entre os elementos arquiteturais e a identificação das dependências ocultas entre eles, devem ser avaliadas todas as mudanças no nível do sistema, permitindo, assim, formar uma sequência de alterações que possibilitem a comparação de diferentes elementos de acordo com os dados históricos obtidos do SCV. Se os relatórios de mudança identificarem o mesmo motivo para a intervenção, o acoplamento lógico é verificado (GALL et al., 1998).

Estudos recentes correlacionam as mudanças conjuntas com informações recuperadas automaticamente de relatórios gerados por ferramentas de rastreamento de defeitos (TANTITHAMTHAVORN et al., 2013; KOUROSHFAR, 2013; MO et al., 2015; CAI; KAZMAN, 2016; MO et al., 2016).Grande parte desses estudos tem como objetivo principal a identificação de arquivos ou módulos propensos à propagação de defeitos, a partir da análise do acoplamento evolutivo. Contudo, o uso de informações provenientes de um SCV integradas às informações obtidas de um Sistema de Gestão de Projetos (SGP) pode aprofundar a a avaliação da degradação da qualidade de código.

Xiao et al. (2014a) consideram o acoplamento evolutivo como um tipo especial de relação entre elementos arquiteturais não conectados estruturalmente. Nesse estudo, os autores investigaram como correlacionar as relações lógicas aos arquivos propensos a defeitos e capturam estruturas de arquitetura e evolução, apoiados por uma ferramenta proprietária denominada Titan (XIAO et al., 2014b). Por exemplo, se dois arquivos forem alterados simultaneamente 10 vezes, conforme registrado no SCV, a ferramenta Titan considera que eles são acoplados de forma evolutiva com um peso de 10, durante o período

especificado. Dessa forma, foi possível identificar problemas ocultos na arquitetura do sistema que impactavam os custos de desenvolvimento do software durante as atividades de manutenção.

Alguns estudos (MO et al., 2015, 2016; CAI; KAZMAN, 2016; XIAO et al., 2016) utilizam o acoplamento evolutivo para determinar deterioração arquitetural de forma indireta. Entretanto, podem ser estabelecidos critérios para medir a deterioração arquitetural diretamente, apenas com parâmetros acerca do acoplamento evolutivo. Com isso, arquitetos podem dispensar modelos adicionais ou padrões de defeitos para inferir sobre a deterioração arquitetural causada pelo acoplamento evolutivo.

2.6 PROBABILIDADE DE ACOPLAMENTO EVOLUTIVO

Mudanças podem impactar em outras mudanças (efeito de propagação ou *ripple effect*) (CLEMENTS et al., 2002; MARTINI; BOSCH, 2015) e sua rastreabilidade é normalmente observável por análise baseada em código, considerando as dependências estruturais entre elementos arquiteturais. No entanto, existem dependências lógicas que não são evidentes na análise do código-fonte. Uma abordagem utilizada para descobrir tais dependências é coletar dados obtidos de um SCV, que fornece relatórios sobre alterações para cada versão do sistema.

A Probabilidade de Acoplamento Evolutivo ou *History Coupling Probability* (HCP) é uma técnica apresentada por Xiao et al. (2016), que tenta demonstrar como uma mudança em um componente influencia outros componentes quando eles mudam juntos. Essa abordagem é utilizada para apresentar as probabilidades de propagação de mudanças entre arquivos e capturar conexões problemáticas entre eles (XIAO et al., 2016).

O HCP é calculado para cada versão de um sistema e modela o acoplamento evolutivo em uma matrizes HCP, que é uma Matriz de Estrutura de Dependências ou *Design Structure Matrix* (DSM). Essas matrizes apresentam os relacionamentos hierárquicos e as interdependências lógicas entre os elementos da arquitetura de software. A DSM foi inicialmente descrita por Steward (1981), estendida e refinada por Eppinger et al. (1994) e difundida por Baldwin e Clark (2000) para a análise da estrutura do projeto de sistemas. A DSM é uma matriz quadrada, que representa conexões entre elementos do sistema - no caso de uma arquitetura de software. No processo de avaliação da arquitetura de software é preciso entender quais são os componentes e como esses se relacionam. Devido à DSM fornecer uma representação visual concisa de estruturas complexas, seu emprego pode facilitar a análise dos componentes arquiteturais do software e suas relações.

A Figura 2.4 apresenta o método HCP e como as probabilidades são calculadas.

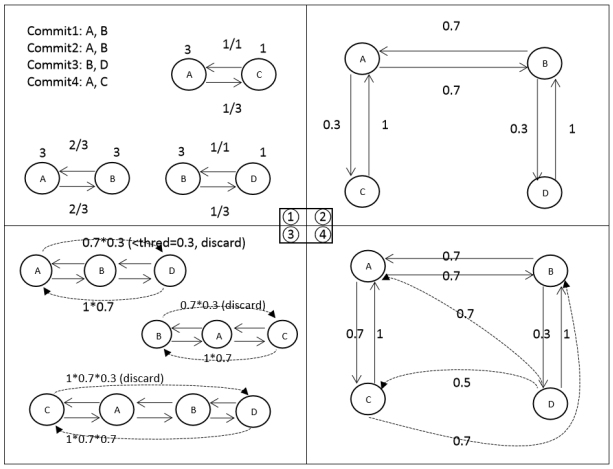

FIG. 2.4: Geração da matriz HCP (XIAO et al., 2016). Círculos representam elementos arquiteturais e as setas indicam as probabilidades de acoplamento evolutivo.

Por exemplo, a probabilidade de alterar o elemento A dado que o elemento C mudou é o número de vezes que tanto A como C mudaram na mesma transação, dividido pelo número total de mudanças de C. Nesta relação, C é dominante em relação a A, que é chamado de submisso, uma vez que a probabilidade de A mudar quando C mudou é maior do que a probabilidade de C mudar quando A mudou. É possível ver que os elementos B e C não mudam em uma mesma transação, mas ambos mudam com o elemento A. Portanto, há uma conexão histórica transitiva entre os elementos B e C calculada por Xiao et al. (2016) multiplicando a probabilidade de alterações do elemento B em conjunto com o elemento A e a probabilidade de o elemento A mudar junto com o elemento C. Dessa forma, HCP[x|y] é a probabilidade de mudar x quando y for modificado em uma versão do software. Uma probabilidade alta de mudar um arquivo quando outro for modificado em uma versão indica um maior acoplamento evolutivo e, consequentemente, um custo de manutenção maior.

A técnica inicia com o cálculo das probabilidades de alteração condicional para qualquer par de arquivos. Essas probabilidades são usadas para preencher a Matriz HCP para os pares que possuem alterações simultâneas. Para os pares que não foram cobertos no passo inicial, ou seja, arquivos que mudam em transações distintas mas potencialmente relacionadas, o HCP calcula uma probabilidade de alteração condicional usando a multiplicação das probabilidades nas relações transitivas (N-Transitive-Closure [1]). O método HCP desconsidera as probabilidades menores que 30%, por considerá-las relações fracas.

[1]As relações transitivas de um elemento do grafo são obtidas adicionando repetidamente uma dependência entre (a, c) em N para cada (a, b) ∈ N e (b, c) ∈ N.

A Figura 2.5 exemplifica o método HCP, exibindo a quantidade de mudanças e os índices de probabilidade de acoplamento evolutivo para três elementos não conectados estruturalmente (A, B e C), que foram observados durante quatro hipotéticas versões subsequentes (N, N+1, N+2 e N+3).

		versão N		
Mudanças		A	B	C
10	A		0,8	0,6
5	B	0,4		
5	C	0,3		

		versão N+1		
Mudanças		A	B	C
1	A			0,5
0	B			
2	C	1,0		

		versão N+2		
Mudanças		A	B	C
0	A			
0	B			
0	C			

		versão N+3		
Mudanças		A	B	C
1	A		0,0	0,0
5	B	0,0		
1	C	0,0		

FIG. 2.5: Matrizes de probabilidade de acoplamento evolutivo representando três elementos (A, B e C) e suas mudanças em quatro versões subsequentes (N, N+1, N+2 e N+3).

Por exemplo, na versão "N", o elemento "B" mudou 5 (cinco) vezes. Pode ser observado que o HCP do elemento "A" em relação às mudanças do elemento "B" é igual a 0,8. Isto indica que os elementos "A" e "B" mudaram juntos 4 (quatro) vezes e, portanto, a probabilidade do elemento "A" ser alterado quando o elemento "B" mudar é de 80%. Nas versões "N+1" e "N+2", o elemento "B" não é modificado, portanto não é possível observar índices de HCP com este elemento nessas versões. Na versão "N+3", o elemento "B" volta a mudar em 5 (cinco) atualizações distintas, contudo não apresenta mudanças conjuntas com o elemento "A" e, por essa razão, o índice de HCP entre esses elementos é igual a 0 (zero). Chama a atenção neste exemplo a oscilação do índice entre versões próximas e o fato das versões intermediárias não possuírem índices HCP quando o elemento não é alterado.

Também na versão "N", o HCP do elemento "A" em relação às mudanças do elemento "C" é igual a 0,6. Na versão "N+1", o índice HCP dessa relação indica 0,5. Na versão "N+2", os elementos não são modificados e, portanto, não apresentam índice HCP. Já na versão "N+3", os elementos são modificados apenas uma vez e não apresentam alterações simultâneas, portanto, o índice HCP é igual a 0 (zero). Neste exemplo, observa-se que uma relação lógica em duas versões consecutivas teve o índice decaído a 0 (zero), também causando oscilação do índice, devido aos elementos terem sido manipulados apenas uma vez, sem levar em consideração informações que qualifiquem as mudanças.

3 METODOLOGIA

A ideia central deste trabalho é realizar a classificação do acoplamento evolutivo e, a partir desta classificação, definir uma métrica que utiliza critérios para quantificar diretamente o impacto do acoplamento evolutivo em um determinado elemento da arquitetura do software. O processo metodológico por trás da definição desta métrica pode ser visto na Figura 3.1, que empregou a notação BPMN (WOHED et al., 2006).

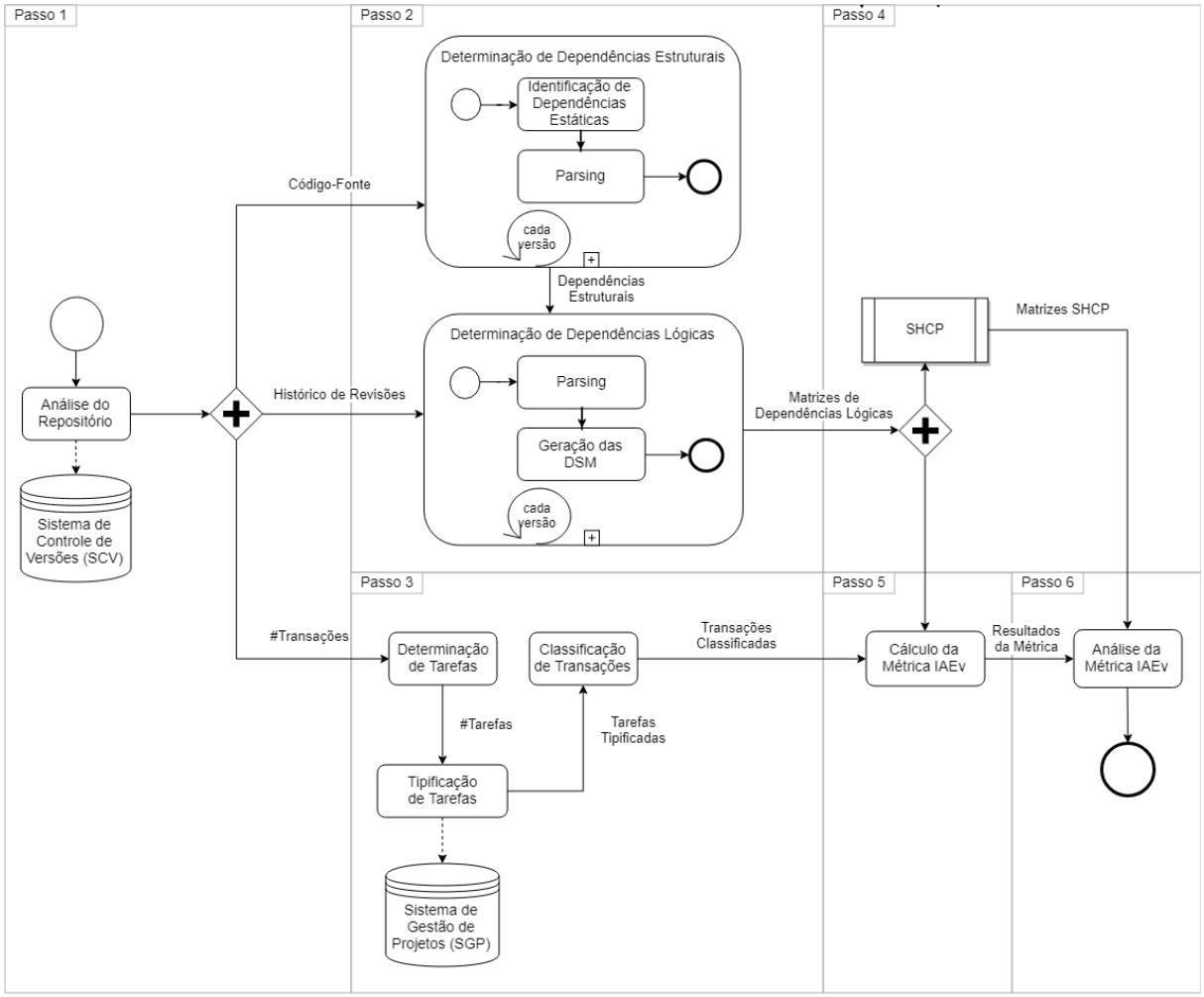

FIG. 3.1: Metodologia de validação da abordagem de classificação das mudanças conjuntas e quantificação do impacto do acoplamento evolutivo.

3.1 PASSO 1: ANÁLISE DO REPOSITÓRIO

O repositório de um SCV, normalmente, contém o versionamento de todos os diretórios de diversos projetos de software, incluindo as cópias de todos os arquivos que compõem seus códigos-fonte, bem como o histórico de adição, mudança, renomeação, cópia e exclusão de cada um desses arquivos. Assim, a partir do SCV é possível recuperar versões anteriores de um software que podem ser compiladas ou visualizar mensagens de alterações para cada uma das transações atômicas realizadas pela equipe de desenvolvimento.

Alguns SCV ainda dispõem do conceito de "linhas de desenvolvimento", separando diferentes *branchs* independentes. Quando o projeto está pronto para ser liberado como uma versão estável, o conteúdo do *branch* é combinado com *trunk* gerando uma versão. Quando os testes efetuados no *branch* estão completos, pode ser criada uma *tag* (uma variação do *branch*) que identifica uma versão enviada ao cliente.

No primeiro passo da metodologia apresentada na Figura 3.1, essas diferentes informações provenientes do repositório seguem fluxos diferentes no processo: (i) a partir do código-fonte compilado, são obtidos arquivos compactados que armazenam elementos construídos e metadados associados que constituem o programa, necessários para a determinação de dependências estruturais; (ii) a partir do relatório de registro histórico, é possível determinar todas as mudanças conjuntas entre pares de arquivos, necessárias para a geração das matrizes de dependências lógicas; (iii) para cada transação são obtidas informações sobre as tarefas executadas pela equipe de desenvolvimento, necessárias para a classificação do acoplamento evolutivo, integrando o SCV ao SGP.

A Figura 3.2 exibe um extrato de um SCV contendo as informações fornecidas a partir do histórico de revisões. São elas:

A - O número da transação que identifica a revisão de código;

B - O nome do autor que realizou as mudanças contempladas na transação;

C - A data e o horário da transação;

D - O tipo da mudança em cada arquivo: A (adição), M (modificação) ou D (exclusão);

E - O *branch* que está sendo atualizado;

F - O caminho completo dos arquivos, permitindo a identificação do pacote modificado;

G - O nome de cada um dos arquivos modificados pela transação;

H - A mensagem de texto informada pelo autor com detalhes sobre a transação;

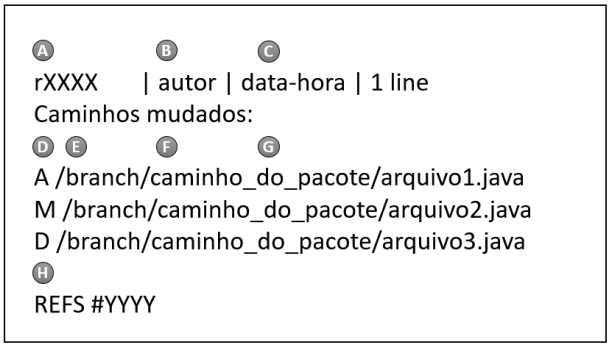

FIG. 3.2: Informações do histórico de um SCV (SVN).

3.2 PASSO 2: DETERMINAÇÃO DE DEPENDÊNCIAS ESTRUTURAIS E LÓGICAS

Os elementos arquiteturais conectados estruturalmente devem ser desconsiderados do computo do acoplamento evolutivo e, por esse motivo, esse tipo de dependência deve ser recuperada para cada versão do software em análise. Para tal, é necessário estabelecer algum mecanismo para determinar os distintos tipos de dependências estruturais em um sistema de software.

Um relacionamento de associação define o estado das instâncias dos objetos, onde instâncias de uma classe, por exemplo, de alguma forma dependem de instâncias de outra classe. A generalização corresponde à extensão de classes, onde atributos e métodos definidos na superclasse são herdados pelas subclasses. O uso de uma classe como parâmetro para um método ou como uma referência de instância local dentro de um método representa um relacionamento entre classes. A realização, que é a implementação de uma interface por uma classe, também representa um relacionamento de dependência.

Geralmente, é possível empregar alguma ferramenta automática para determinar as dependências em nível de pacote ou em nível da classe e, posteriormente, analisar sintaticamente esses dados para estabelecer as dependências estruturais em uma versão do software. Mais especificamente na linguagem Java, é possível identificar em uma classe de entrada um caminho para outro arquivo .class, um diretório, um arquivo JAR ou um nome de classe totalmente qualificado que permite a determinação de todos os arquivos dos quais a classe é dependente.

Uma vez que forem determinadas as dependências estruturais, pode-se estabelecer quais pares de elementos não estão conectados estruturalmente. Sendo assim, é possível empregar a análise de informações históricas para determinar as dependências entre estes elementos. Ball et al. (1997) ilustram algumas maneiras de utilizar informações históricas para entender melhor o processo de desenvolvimento de software. Um SCV permite

rastrear cada mudança que um desenvolvedor realiza ao longo da evolução do software e, como resultado, é possível representar as mudanças em elementos de um sistema consistente em qualquer ponto no tempo.

As informações históricas do SCV podem ser consolidadas em um relatório padronizado, permitindo analisar esses dados para estabelecer as dependências lógicas para cada versão do software. A análise sintática de relatórios provenientes do SCV permite a estruturação de dados sobre o número da revisão, o nome do desenvolvedor, a data, as mensagens e o caminho completo dos arquivos modificados em cada transação. Essa última informação é a chave para a determinação das mudanças conjuntas, pois identifica todo e qualquer par de arquivos que tenham sido modificados ao mesmo tempo a partir do histórico de versões.

Assim, utilizando as informações coletadas sobre as dependências estruturais e lógicas para os pares de elementos que compõem a arquitetura do software, o passo dois prevê a representação dessas informações em matrizes DSM para cada versão do sistema de software em análise.

3.3 PASSO 3: CLASSIFICAÇÃO DAS MUDANÇAS CONJUNTAS

O passo 3 (três) é responsável por classificar as transações do SCV e realizar a consequente classificação das mudanças conjuntas (*co-changes*). Ao avaliar as características de cada transação em que um determinado par de elementos mudou, é possível determinar as características da própria *co-change* empregada na etapa de quantificação do impacto do acoplamento evolutivo (passo 5).

A maioria das características usadas para a classificação das mudanças conjuntas podem ser estabelecidas a partir de informações extraídas diretamente do histórico do SCV. Entretanto, a classificação realizada neste estudo amplia esse escopo de características ao considerar a integração do SCV com o SGP. Essa integração possibilita a referência direta para tarefa responsável pela intervenção no código-fonte.

A motivação para se usar informações do SGP, adicionalmente àquelas já fornecidas pelo SCV, está diretamente relacionada à ampliação do conhecimento sobre as atividades de desenvolvimento de software. As transações de revisão de código podem ser correlacionadas às informações sobre tarefas e recursos humanos. Isto permite confrontar dados sobre a colaboração entre membros da equipe, produtividade, tempo de projeto, duração de atividades, custos, qualidade, riscos e funcionamento dos fluxos de trabalho, por exemplo.

O processo “determinação de tarefas” (Figura 3.1), ao receber o número de cada transação, é responsável por verificar, a partir da integração com o SGP, o identificador da tarefa responsável pela intervenção no código-fonte. A saída desse processo é o identificador da tarefa correspondente no SGP, que serve de entrada ao processo seguinte.

O processo “tipificação de tarefas” (Figura 3.1), ao receber o número da tarefa correspondente, lê no SGP o tipo de intervenção e retorna essa tipificação como entrada do processo “classificação de Transações”.

O processo “classificação de transações” (Figura 3.1) é responsável por atribuir a cada transação características da tarefa responsável pelas mudanças que motivaram a intervenção no código-fonte. Cada transação, então, pode ser agrupada conforme o tipo de mudança, de acordo com a Figura 3.3: para a perspectiva de evolução, atendimento de um novo requisito ou atendimento de uma solicitação de mudança; para a perspectiva de manutenção, correção de defeito interno ou correção de defeito externo.

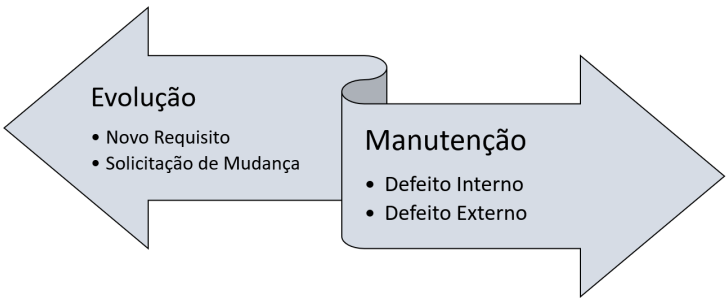

FIG. 3.3: Características de agrupamento e mudança das transações.

Além das características de agrupamento e mudança, as transações também serão caracterizadas de acordo com o tipo de tarefa.

A **classificação de tarefa** avalia as mudanças conjuntas entre pares de elementos a partir do ponto de vista do tipo de tarefa que originou a revisão de código, tendo como parâmetros informações sobre o desenvolvedor, a data da mudança e a tarefa que motivou a intervenção. Dessa forma, quando um elemento “A” mudar para atender a uma tarefa “T” e for observada mudança do elemento “B” em atendimento à mesma tarefa “T”, pelo mesmo autor e na mesma data, as mudanças conjuntas entre os elementos “A” e “B” serão consideradas, mesmo que em transações diferentes.

Assim, a classificação das mudanças conjuntas apresentada neste estudo amplia o escopo da identificação da deterioração arquitetural, pois considera mudanças em elementos arquiteturais em transações distintas, mas próximas temporalmente. Com isso, é possível determinar a dependência lógica entre esses elementos, levando em consideração o motivo da mudança e as informações acerca dos desenvolvedores.

Portanto, define-se *co-change-task* como mudanças conjuntas de um par de elementos arquiteturais, determinadas a partir da classificação de tarefa.

Para estabelecer o número de desenvolvedores distintos (NDD) que atuaram nas revisões, para cada par de elementos conectados logicamente, esta metodologia avalia o nome do autor responsável por cada transação. Para estabelecer o tamanho da mudança, esta metodologia considera o número de revisões do par de elementos (NRP) ligados logicamente. Por exemplo, o NRP entre os elementos "A" e "B", acoplados logicamente, é o número total de revisões do elemento "A" somado ao número total de revisões do elemento "B". Entende-se, portanto, que quanto mais os elementos são modificados em uma versão do software, maior é o tamanho da mudança.

A classificação temporal prevista por esta metodologia considera a informação de data e hora da transação, obtida do histórico do SCV. Esta informação tem a finalidade de identificar as transações que ocorreram no mesmo dia. Contudo, o momento do *commit* não identifica o exato momento da conclusão da tarefa de intervenção no código-fonte. Este estudo não considera a data de início e a data de conclusão da tarefa, mas reconhece que esta informação também pode ser obtida a partir da integração com o SGP.

Esta metodologia ainda utiliza a informação do tipo da mudança extraída do histórico de transações do SCV. Para minimizar o impacto que transações iniciais causam na análise das dependências lógicas, essa metodologia não contabiliza as relações entre pares de arquivos que foram adicionados (A) ao mesmo tempo no sistema. No entanto, todas modificações (M) ou exclusões (D) são consideradas no cômputo das mudanças conjuntas. Este estudo considera que, no momento da adição de arquivos ou em grandes operações de *merge* entre um *branch* e o *trunck*, a operação é executada em lotes de arquivos distintos, sem que isso caracterize a existência de dependências lógicas entre eles.

3.4 PASSO 4: DETECÇÃO DO ACOPLAMENTO EVOLUTIVO

O quarto passo da metodologia apresentada na Figura 3.1 prevê o emprego da abordagem da verificação deslizante para detectar instâncias de dependências lógicas na arquitetura do software. O objetivo desse passo é identificar potenciais elementos que apresentem o acoplamento evolutivo e comparar posteriormente com os resultados da métrica proposta.

Apresentado por Xiao et al. (2016), o método HCP é utilizado para determinar a probabilidade de acoplamento evolutivo para um par de elementos. O método é aplicado a uma versão de software por vez, de modo que as mudanças conjuntas do mesmo par de elementos em versões de software consecutivas tendem a flutuar. Dessa forma, o

HCP pode refletir informações de um par de elementos que não estão verdadeiramente acoplados historicamente, como por exemplo, devido à alta instabilidade das revisões iniciais, quando um novo recurso é adicionado ao software.

Para contornar este problema e melhorar a detecção de elementos arquiteturais com acoplamento evolutivo, foi proposta uma variação no cálculo do HCP. Esta variação baseia-se no fato de que o acoplamento evolutivo entre elementos arquiteturais não pode ser simplesmente detectado em uma versão individual de um sistema de software, mas sim em uma série contínua (janela deslizante) de versões. Esta variação, denominada *Sliding-HCP* ou, simplesmente, SHCP, busca confirmar se um acoplamento evolutivo detectado não é efêmero.

3.4.1 ABORDAGEM DA VERIFICAÇÃO DESLIZANTE

A abordagem da verificação deslizante consiste em identificar a recorrência dos índices HCP em versões consecutivas de um sistema em evolução, de tal modo que seu algoritmo analise se as probabilidades de mudança são observadas em, pelo menos, duas versões consecutivas.

Essa abordagem considera que as mudanças conjuntas entre elementos representam, de fato, instâncias de acoplamento evolutivo se, ao avaliar N versões subsequentes, identificar a recorrência dos índices HCP em ϑ versões consecutivas (onde $N \geq \vartheta > 1$). O parâmetro ϑ pode ser configurado pelo arquiteto, contudo, testes com $\vartheta > 2$ não apresentaram melhorias significativas dos resultados.

Assim, se identificados índices HCP em ϑ versões consecutivas, o acoplamento evolutivo entre os elementos é "ativado". Após ativado, são necessárias ϑ versões consecutivas em que os elementos sejam alterados sem apresentarem mudanças conjuntas para que o acoplamento evolutivo entre eles seja "desativado". Enquanto o acoplamento evolutivo permanecer ativado, a probabilidade será apresentada na DSM. Nesse caso, para as versões intermediárias em que o elemento dominante não é alterado, prevalecerá a média dos acoplamentos evolutivos observados. Se o acoplamento evolutivo for "desativado", a probabilidade indicará zero. Os valores zero indicam que os elementos permaneceram independentes entre si, como observado na análise estrutural do código. O subprocesso "SHCP" apresentado na Figura 3.1 é detalhado na Figura 3.4.

As matrizes de dependências lógicas são a entrada do subprocesso "SHCP". O primeiro passo deste subprocesso prevê a análise das versões consecutivas, de modo a gerar as matrizes HCP de acordo com a metodologia apresentada por Xiao et al. (2016). Então,

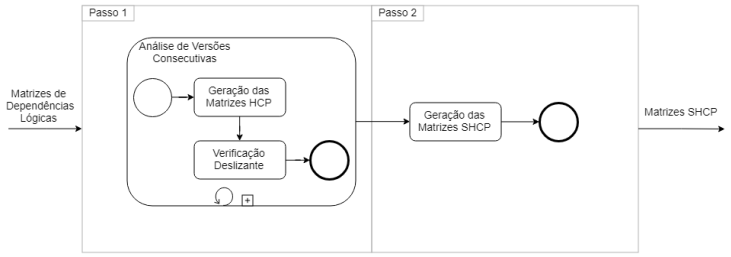

FIG. 3.4: Subprocesso SHCP previsto na metodologia ilustrada na Figura 3.1.

utilizando a série de matrizes produzidas, os dados são reprocessados para confirmar a incidência do acoplamento evolutivo a partir da abordagem da verificação deslizante. O segundo passo deste subprocesso é responsável pela geração das matrizes DSM com os índices SHCP.

A abordagem da verificação deslizante foi testada de modo a permitir a avaliação da eficácia de se considerar mais de uma versão ($\vartheta > 1$) para determinação das relações lógicas, utilizando os cálculos de probabilidade de acoplamento evolutivo. Este estudo se restringiu a apresentar a melhoria da avaliação, considerando o mínimo de versões consecutivas ($\vartheta = 2$). A partir dessa premissa, o algoritmo proposto foi definido de acordo com os seguintes passos:

a) Inicialmente são criadas DSM auxiliares vazias com as mesmas dimensões das matrizes HCP, para cada versão analisada.

b) Para a primeira versão (N), cada célula (linha x coluna) da matriz HCP é avaliada e a DSM auxiliar é preenchida na mesma posição, da seguinte forma: se o HCP for igual a 0 (zero), a célula recebe o valor “F” (falso); se o HCP for maior que 0 (zero), a célula recebe o valor “?” (interrogação). O valor “?” indica que um acoplamento evolutivo foi detectado, porém não confirmado, pois o método SHCP requer duas ocorrências em versões consecutivas (para $\vartheta = 2$). Células vazias ou com dependências estruturais são ignoradas. Após percorrer toda a matriz HCP, os resultados da DSM auxiliar são copiados para a DSM auxiliar da versão seguinte (“N+1”).

c) A partir da segunda versão, cada célula (linha x coluna) da matriz HCP é avaliada, seguindo os mesmos critérios anteriormente descritos. Contudo, os valores são combinados com os resultados já armazenados na DSM auxiliar, da seguinte forma: se na célula estiver armazenado o valor “F” e for combinado com “?”, a célula recebe o valor “?”; se na célula estiver armazenado o valor “F” e for combinado com

"F", a célula se mantém com o valor "F"; se na célula estiver armazenado o valor "?" e for combinado com "?" (duas ocorrências em versões consecutivas), a célula recebe o valor "V" (verdadeiro) e todas as mesmas células de versões anteriores que possuírem "?" também terão o acoplamento evolutivo confirmado; se na célula estiver armazenado o valor "?" e for combinado com "F", a célula recebe o valor "F" e todas as mesmas células de versões anteriores que possuírem "?" também terão o acoplamento evolutivo descartado; se na célula estiver armazenado o valor "V" e for combinado com "V", a célula se mantém com o valor "V"; se na célula estiver armazenado o valor "V" e for combinado com "F", a célula recebe o valor "?", pois são necessárias duas versões com HCP igual a zero para retornar o valor "F".

d) Em paralelo a este processo, outra matriz auxiliar deve computar as somas e quantidades de HCP para cada posição (linha x coluna), de forma a permitir o estabelecimento do HCP médio de cada relação lógica identificada para cada par de elementos avaliados.

e) Ao final do processamento das matrizes HCP de todas as versões, as matrizes DSM auxiliares são novamente analisadas para a geração das matrizes com os índices do SHCP. Dessa forma, cada célula (linha x coluna) da DSM auxiliar é avaliada para cada versão analisada. Se o valor encontrado for "V", recupera-se na matriz HCP o valor da probabilidade de acoplamento evolutivo e este valor é introduzido na matriz SHCP. Se a matriz HCP não possuir índice ou este for 0 (zero) em uma determinada versão e, ainda assim, o valor "V" for identificado na DSM auxiliar, o HCP médio é recuperado e introduzido na matriz SHCP.

3.4.2 COMPARAÇÃO ENTRE OS MÉTODOS HCP E SHCP

A Tabela 3.1 apresenta o comportamento dos métodos HCP e SHCP ($\vartheta = 2$) para as relações apresentadas na Figura 2.5, a fim de permitir a distinção do comportamento dos métodos. Por exemplo, de acordo com a Figura 2.5, na versão "N", a probabilidade do elemento "A" mudar em relação às mudanças do elemento "C" é igual a 0,6. Na versão "N+1", essa probabilidade indica 0,5. Essas duas observações consecutivas "ativam" o acoplamento evolutivo da relação para o método SHCP. Na versão "N+2", o elemento "C" não é modificado, contudo, como a relação lógica se mantém "ativada", a média dos acoplamentos evolutivos observados prevalece nessa versão. Embora na versão "N+3" o elemento "C" seja modificado, não são observadas mudanças conjuntas desse elemento com o elemento "A". Neste caso, a relação lógica continua "ativada" considerando a média

dos acoplamentos evolutivos observados. São necessárias duas versões consecutivas em que o elemento "C" seja alterado e não apresente mudanças conjuntas com o elemento "A" para que o acoplamento evolutivo entre eles seja "desativado". De acordo com o método SHCP, por exemplo, se na versão "N+4" o elemento "C" for alterado sem apresentar alteração conjunta com o elemento "A", o acoplamento evolutivo entre esses elementos seria "desativado" e o índice cai a 0 (zero).

TAB. 3.1: Comparação entre os métodos HCP e SHCP ($\vartheta = 2$) para as relações apresentadas na Figura 2.5. [A|C], por exemplo, é a probabilidade de mudar A quando C for modificado em uma determinada versão do software.

		N	**N+1**	**N+2**	**N+3**
A|B	HCP	0,8			0,0
	SHCP				
A|C	HCP	0,6	0,5		0,0
	SHCP	**0,6**	**0,5**	**0,55**	**0,55**
B|A	HCP	0,4			0,0
	SHCP				
B|C	HCP				
	SHCP				
C|A	HCP	0,3	1,0		0,0
	SHCP	**0,3**	**1,0**	**0,65**	**0,65**
C|B	HCP				
	SHCP				

A Figura 3.5 exibe a representação gráfica da relação lógica do elemento "A" em relação às mudanças do elemento "C". Enquanto o acoplamento evolutivo desse par de elementos oscila com o método HCP, nota-se essa variação mais estável ao aplicar o método SHCP. A alta instabilidade devido ao grande número de mudanças nas primeiras versões é controlada, pois são necessárias duas verificações para "ativar" o acoplamento evolutivo entre os elementos. O método HCP não apresenta índices em versões em que o elemento dominante não é modificado, contudo o método SHCP mantém a apresentação dos índices médios enquanto o acoplamento evolutivo não for efetivamente solucionado. As mudanças mínimas que causam grandes oscilações no método HCP são empregadas para confirmar o acoplamento evolutivo no método SHCP. Ao contrário do método HCP, as baixas probabilidades não são descartadas pelo método SHCP, pois considera-se que o impacto de desconsiderar tais probabilidades deveria ser melhor avaliado.

Nesse sentido, valores não-zero devem indicar um conjunto mais real de elementos que apresentam esse tipo de problema em um subconjunto contínuo de versões. Desta forma, apenas pares de elementos que apresentem mudanças conjuntas recorrentes se-

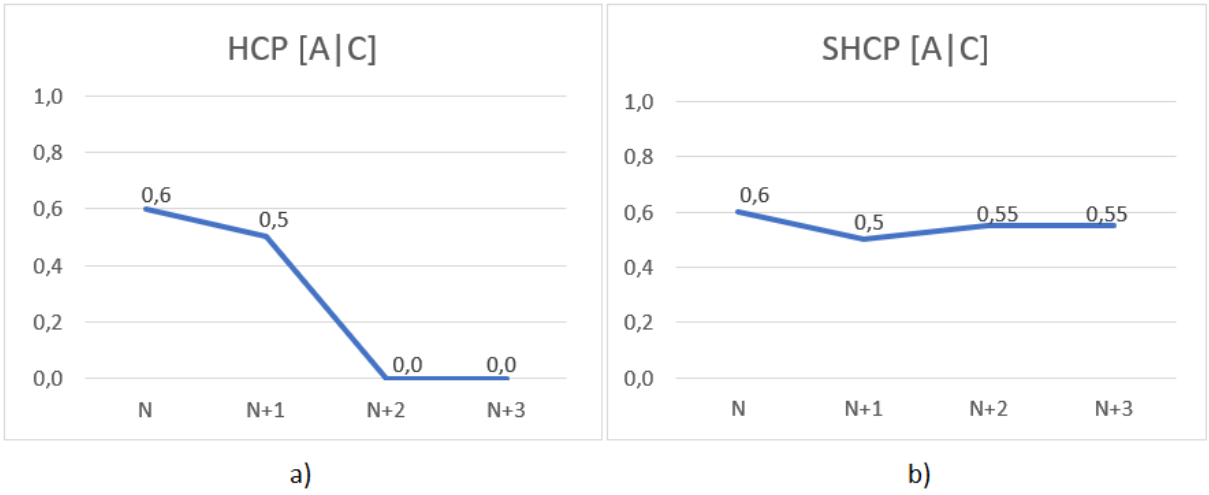

FIG. 3.5: Representação gráfica da dependência lógica [A|C] (Tabela 3.1). A variação do índice HCP é apresentada em "a". A variação do índice SHCP é apresentada em "b".

rão exibidos nos resultados, e grupos com alta propensão de estarem impactados pela deterioração arquitetural serão filtrados. Análises manuais de código podem confirmar as indicações dos dados apresentados que, devido a serem mais assertivos, diminuem o esforço do diagnóstico pelo arquiteto.

3.5 PASSO 5: CÁLCULO DA MÉTRICA IMPACTO DO ACOPLAMENTO EVOLUTIVO (IAEV)

O passo 5 (cinco) da metodologia apresentada na Figura 3.1 refere-se ao cálculo da métrica que utiliza a classificação das mudanças conjuntas realizadas pelo passo 3 (três), de modo a quantificar o impacto do acoplamento evolutivo em cada elemento avaliado.

Baseada no conceito das *co-changes-task*, a geração das matrizes HCP (XIAO et al., 2016) sofre uma variação visando a classificação das mudanças conjuntas que determinaram as probabilidades de acoplamento histórico. O HCPT (de HCP baseado em tarefas) utiliza as classificações apresentadas na Seção 3.3, de modo a fornecer a cada relação lógica de um determinado elemento arquitetural maior significado.

Para melhor ilustrar esse conceito, serão analisadas as dependências de um suposto elemento arquitetural ("A") conectado logicamente com outros seis elementos ("B até G"), de acordo com a Figura 3.6. A perspectiva da versão do exemplo está definida como Evolução. O número de vezes que cada par de elementos mudou junto, baseado na classificação de tarefa, é apresentado ao lado do rótulo *co-change-task*. O número de desenvolvedores distintos que atuaram nas revisões coincidentes, para cada par de elementos ligados logicamente, é apresentado ao lado do rótulo NDD.

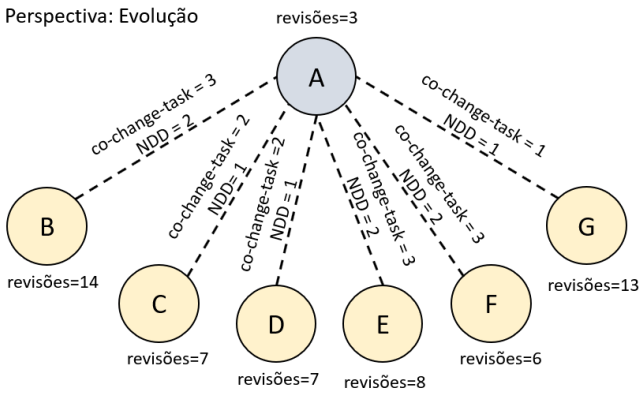

FIG. 3.6: Conexões lógicas entre elementos arquiteturais em uma versão sob a perspectiva de evolução. Cada círculo representa um elemento. As arestas indicam as conexões lógicas.

A partir da Figura 3.6, é possível constatar, por exemplo, que o elemento "A" foi alterado em 3 (três) revisões e o elemento "B" foi alterado em 14 (quatorze) revisões. Foram observadas 3 alterações conjuntas entre os elementos "A" e "B". Um total de dois desenvolvedores foram responsáveis pelas mudanças conjuntas entre esses elementos.

Além de computar o número de mudanças conjuntas de um par de elementos, o cálculo da métrica deverá considerar outros fatores que impactam diretamente na caracterização do problema estrutural causado pelo acoplamento evolutivo. As mudanças conjuntas são apenas uma evidência da existência do acoplamento evolutivo e, por esta razão, quanto mais evidências forem coletadas, maiores as chances de se determinar o impacto do acoplamento evolutivo em um elemento arquitetural.

Por exemplo, em uma dada versão dois pares de elementos apresentam 13 (treze) mudanças conjuntas cada um. Assim, se for considerado somente o número de mudanças conjuntas, esses elementos estarão impactados igualmente. Entretanto, entende-se que o par de elementos que apresenta um maior número de intervenções está mais propenso a possuir problemas estruturais que acabam por demandar as mudanças no código. Analogamente, um código mantido por apenas um desenvolvedor tende a ser mais estável se comparado com um código mantido por mais de um desenvolvedor distinto (KIRBAS et al., 2017).

Dessa forma, o método requer um fator de impacto que leve em consideração o tamanho da mudança, a partir do número de revisões do par (NRP) e o número de desenvolvedores distintos (NDD). Uma variação regular do fator de impacto, conforme aumento dessas medidas, necessita ser estabelecida. Assim, experimentos resultaram na distribuição dos valores ilustrados na Figura 3.7, respeitando a seguinte expressão matemática: $Impacto = \frac{(0{,}3 \times FaixaNRP + 0{,}1 \times FaixaNDD)}{0{,}4 \times N}$, com as faixas variando de 1 (baixa) até N (alta).

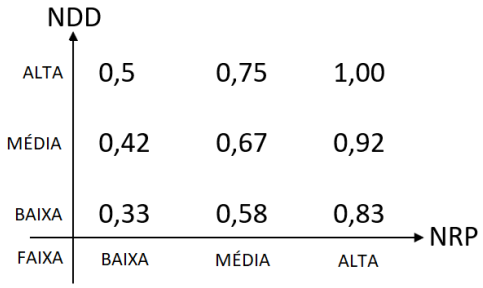

FIG. 3.7: Fator de impacto da mudança conjunta.

Os valores exibidos na Figura 3.7 levaram em consideração 3 (três) faixas. Valores absolutos para as faixas devem ser atribuídos, de acordo com as observações do sistema em análise. A Tabela 3.2 exemplifica o cálculo do fator de impacto usado para classificar as dependências lógicas significativas para a determinação do acoplamento evolutivo.

TAB. 3.2: Cálculo do fator de impacto e HCPT para os elementos logicamente dependentes do elemento "A" exibido na Figura 3.6.

Evolução Dep. "A"	**co-change task**	**revisões**	**NDD**	**NRP**	**Fator de Impacto**	**HCPT**
B	3	14	2	17	0,92	0,92
C	2	7	1	10	0,83	0,56
D	2	7	1	10	0,83	0,56
E	3	8	2	11	0,92	0,92
F	3	6	2	9	0,67	0,67
G	1	13	1	16	0,83	0,28
$\sum_{dep(A)} = 6$						$\sum_{HCPT(A)} = 3,91$

O HCPT leva em consideração as *co-change-task* e o fator de impacto da mudança. Essa variação dos índices calculados pelo método HCP, representada na Tabela 3.2, é obtida de acordo com a Equação 3.1.

$$HCPT\{x|y\} = \frac{\#cochangetask(x|y)}{\#revisoes(x)} * fatordeimpacto \tag{3.1}$$

Por exemplo, as 3 (três) *co-change-task* identificadas entre os elementos "A" e "B", representada na Figura 3.6, foram executadas por dois desenvolvedores distintos, portanto o NDD = 2 (faixa média neste exemplo). O número de revisões de "A" é igual a 3 (três) e o número de revisões de "B" é igual a 14 (quatorze), portanto o NRP = 17 (faixa alta neste exemplo). O fator de impacto da mudança, de acordo com a Figura 3.7, é 0,92 (NDD média e NRP alta). Assim, aplicando a Equação 3.1 temos:

$$HCPT\{A|B\} = \frac{3}{3} * 0,92 = 0,92$$

A métrica proposta deve indicar, para cada versão do sistema considerada na análise, qual o impacto do acoplamento evolutivo na deterioração do elemento arquitetural. Portanto, são utilizadas as informações dos pares de elementos para mensurar a propensão de cada um dos elementos estar impactado pelo acoplamento evolutivo. Essa técnica permite o monitoramento da deterioração do elemento arquitetural individualmente e, de modo mais amplo, da própria Arquitetura de Software.

Complementarmente ao cômputo das mudanças conjuntas, a métrica IAEv deve considerar, ainda, o número de dependentes lógicos aos quais os elementos se acoplam evolutivamente. Entende-se que, quanto maior o número de dependentes lógicos de um dado elemento arquitetural, mais propenso esse elemento está de apresentar acoplamento evolutivo. No entanto, é necessário realizar um ajuste das observações dependendo da perspectiva de desenvolvimento, pois ela influencia diretamente no número de dependentes lógicos de versão para versão. Essa grande oscilação, dependendo da perspectiva de desenvolvimento, causa variações bruscas na métrica que devem ser minimizadas.

Durante as fases iniciais de uma evolução, a alta instabilidade na construção de novas funcionalidades faz com que as observações de dependentes lógicos sejam mais frequentes. Porém, nas fases de manutenção verifica-se um contraponto: poucos dependentes lógicos são observados, embora sejam suficientes para confirmar o acoplamento evolutivo. Então, como as observações na perspectiva de evolução são mais frequentes, o fator de ajuste do número de dependentes lógicos sob a perspectiva de manutenção deve ser relativamente maior que o fator de ajuste sob a perspectiva de evolução.

Foram realizados experimentos com diversas faixas de valores, mantendo a regra anteriormente estabelecida. Os melhores resultados foram obtidos com a variação linear, de acordo com a distribuição apresentada na Figura 3.8.

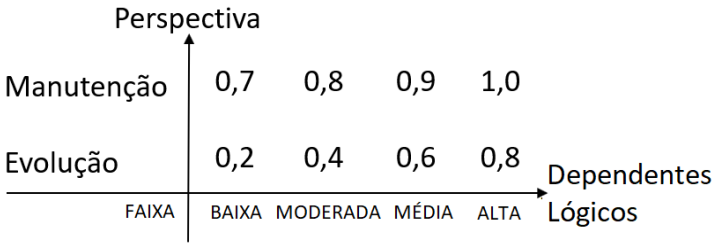

FIG. 3.8: Fator de ajuste de acordo com a perspectiva e o número de dependentes lógicos.

Para a perspectiva de manutenção foi aplicada uma variação de 0,1 em 0,1, terminando com impacto máximo de 1,0. Para a perspectiva de evolução foi aplicada uma variação de 0,2 em 0,2, terminando com impacto máximo de 0,8 (portanto sempre me-

nor que a perspectiva de manutenção). O fator de ajuste mostrou-se eficaz para mitigar a alta instabilidade verificada nas versões iniciais de uma perspectiva de evolução, pois equilibraram a variação da métrica entre versões com perspectivas diferentes. No entanto, cabe destacar que outros fatores de ajuste podem ser definidos, desde que se mantenham os valores sob a perspectiva de manutenção maiores que os valores sob a perspectiva de evolução para cada uma das faixas pré-estabelecidas.

A etapa final para o cálculo da métrica, que quantifica o impacto do acoplamento evolutivo (IAEv) em um dado elemento da arquitetura de software, é realizado considerando a soma de todos os índices HCPT desse elemento, o total de dependências lógicas desse elemento e o fator de ajuste, conforme a Equação 3.2.

$$IAEv\{x\} = \frac{\sum_{HCPT(x)}}{\sum_{dep(x)}} * fatordeajuste \tag{3.2}$$

O número de dependências lógicas de um elemento é determinado somando o número de elementos que dependem logicamente desse elemento. Para o elemento "A" detalhado na Tabela 3.2 foram somadas 6 (seis) dependências lógicas (faixa alta neste exemplo) em uma versão cuja perspectiva é de Evolução (Figura 3.6). Portanto, o fator de ajuste, de acordo com a Figura 3.8, para o elemento "A" é de 0,8. Aplicando a Equação 3.2 temos:

$$IAEv\{A\} = \frac{3,91}{6} * 0,8 = 0,52.$$

Dessa forma, o resultado final da métrica é obtido. Isso indica que, para o exemplo na versão avaliada sob a perspectiva de Evolução, o elemento "A' possui cerca de 52% de probabilidade de apresentar acoplamento evolutivo com outros elementos da arquitetura de software.

No caso de monitoramento contínuo da variação da métrica entre versões, caso o elemento não tenha sido manipulado em uma determinada versão, ou seja, não tenha revisões observáveis no histórico do SCV, a métrica pode recuperar o indicador da versão anterior. Com isso, a métrica só decai a 0% quando o elemento for efetivamente manipulado em uma versão e, mesmo assim, não apresente mudanças conjuntas com quaisquer outros elementos desconectados estruturalmente.

3.6 PASSO 6: ANÁLISE DA MÉTRICA IAEV

O último passo da metodologia apresentada na Figura 3.1 é responsável pela análise dos resultados da métrica IAEv. Esta análise é realizada confrontando os resultados da métrica com as instâncias detectadas de acoplamento evolutivo.

O processo "análise da métrica IAEv" recebe como entrada as matrizes SHCP resultantes do passo quatro, de acordo com o método de detecção apresentado na Seção 3.4. A outra entrada desse processo são os resultados do cálculo da métrica IAEv para cada versão avaliada, resultantes do passo 5 (cinco), apresentado na Seção 3.5 para os mesmos elementos contidos nas matrizes SHCP.

É importante destacar que, de acordo com o processo exibido na Figura 3.1, as DSM usadas no cálculo da métrica IAEv são as matrizes de dependência lógica antes do subprocesso SHCP. O método SHCP não tem nenhuma influência no cálculo da métrica IAEv e serve como método de detecção de instâncias reais de acoplamento evolutivo. Por essa razão, o processo de detecção pode ser substituído por qualquer outro método de acordo com a avaliação do arquiteto de software.

Aplicar o cálculo da métrica IAEv apenas nos elementos mantidos pelo método SHCP eliminaria a possibilidade de monitoramento continuado da evolução da deterioração arquitetural. A metodologia estabelecida, portanto, utiliza as matrizes de dependências lógicas como entrada do processo de cálculo da métrica IAEv e, assim, permite identificar índices contínuos (mesmo que baixos) que indicam a deterioração arquitetural em cada elemento e para cada versão, impactados pelo acoplamento evolutivo.

É possível, portanto, verificar se os elementos que tiverem deterioração arquitetural mantida pela abordagem SHCP (ou qualquer outro método definido pelo arquiteto), de fato, são aqueles que apresentam os índices mais elevados da métrica IAEv. É esperado que os elementos arquiteturais com relações lógicas mantidas pelo método SHCP nas DSM apresentem maior probabilidade de impacto e, do contrário, os elementos arquiteturais sem relações lógicas estabelecidas ou com índices SHCP baixos apresentem menor probabilidade de impacto, de acordo com a métrica proposta.

4 PROVA DE CONCEITO

Para a realização da prova de conceito foi utilizado um sistema como estudo de caso, onde um conjunto de dados foi selecionado para demonstrar a validade da metodologia apresentada no Capítulo 3. Os passos previstos na Figura 3.1 foram seguidos para obtenção dos resultados.

4.1 ESTUDO DE CASO

A quarta geração de um SMC2 denominado "Sistema de Planejamento Operacional Militar" (SIPLOM 4), desenvolvido em atendimento ao Ministério da Defesa, foi empregado ao longo desta pesquisa. Sua utilização teve como objetivo demonstrar a validade da metodologia proposta. Este não é um sistema de código aberto, entretanto, é um projeto mantido pelo Centro de Análises de Sistemas Navais (CASNAV), uma Organização Militar da MB, que autorizou a realização desta pesquisa.

O SIPLOM 4 emprega o *Java* como linguagem de desenvolvimento, o *Apache Subversion* (SVN, 2018) como SCV e o *Redmine* (REDMINE, 2018) como SGP. Os critérios de seleção deste sistema se basearam nas seguintes considerações:

a) Foi utilizado algum nível de integração entre o SCV e o SGP, que permitiu a extração automática e detalhada das informações históricas e de tarefas;

b) Foi implementado em linguagem sob o paradigma OO, usando a linguagem Java, o que permitiu a extração automática das dependências estruturais e lógicas;

c) O autor deste estudo foi membro do projeto (inclusive atuando como Gerente de Projeto) por, aproximadamente, seis anos; e

d) Um conjunto relevante de versões foi disponibilizado para esta pesquisa, totalizando 2.033 pontos de função, com aproximadamente 38.000 horas de esforço de desenvolvimento, em um período de 3 anos.

4.1.1 SELEÇÃO DOS DADOS

O repositório do SVN do SIPLOM 4, disponibilizado para essa pesquisa, possuía quatorze versões consecutivas, conforme Tabela 4.1.

TAB. 4.1: Versões e *branchs* avaliados do SIPLOM 4 e suas perspectivas de desenvolvimento. O campo ID é um identificador de cada *branch*.

Versão	**ID**	**Branch**	**Perspectiva**
Siplom 4.0	V0	4.0-ALPHA_1	Evolução
	V1	4.0-ALPHA_3	Evolução
	V2	4.0-ALPHA_4	Evolução
	V3	4.0-MANUTENCAO	Manutenção
	V4	4.0.4-MANUTENCAO	Manutenção
	V5	4.0.5-MANUTENCAO	Manutenção
	V6	4.0.6-MANUTENCAO	Manutenção
	V7	4.0.7-MANUTENCAO	Manutenção
Siplom 4.1	V8	4.1-ALPHA_1	Evolução
	V9	4.1-ALPHA_2	Evolução
	V10	4.1.1-MANUTENCAO	Manutenção
	V11	4.1.2-MANUTENCAO	Manutenção
	V12	4.1.3-MANUTENCAO	Manutenção
	V13	4.1.4-MANUTENCAO	Manutenção

A metodologia apresentada no Capítulo 3 foi aplicada a todos elementos da arquitetura do software e para todas as versões disponíveis. Entretanto, os dados selecionados para esta dissertação foram referentes aos esforços de desenvolvimento ocorridos durante e após os *branchs* V8 e V9, devido a representarem o maior passo evolutivo que resultou na substituição da versão 4.0 pela versão 4.1.

As funcionalidades atinentes aos Documentos Operacionais caracterizam a versão 4.1. A partir desta versão, o SIPLOM 4 passou a controlar a comunicação entre as Forças Componentes (FCte), o Comando Operacional Ativado (ComOpA) e o Centro de Operações Conjuntas do Ministério da Defesa (COC-MD). Essa nova versão implementou a elaboração, trâmite, controle de acesso e impressão dos seguintes documentos: Mensagem Operacional, Ordem de Coordenação e Sumário Diário de Situação.

O SIPLOM 4 possui funcionalidades comuns a sistemas integrados de gestão (SIG) e, complementarmente, incorpora as funcionalidades específicas do segmento Militar. Para evitar que os resultados sejam entendidos como válidos apenas a sistemas com finalidade Militar, foram desconsideradas as funcionalidades específicas de Comando e Controle (C2).

Dessa forma, dois fatores foram determinantes para a seleção dos dados: o primeiro refere-se à existência de elementos envolvidos tanto no processo de desenvolvimento de novas funcionalidades quanto de refatoração de funcionalidades existentes; o segundo é devido a esta funcionalidade não se restringir ao escopo Militar e, dessa forma, a generalização dos resultados ser mais abrangente.

Cabe destacar a implementação de um protótipo para automatizar parcialmente a metodologia apresentada no Capítulo 3. Esta implementação considerou o ambiente de desenvolvimento, a linguagem e as ferramentas utilizadas pelo sistema de estudo de caso. Os passos destacados nas seções a seguir consideram a utilização deste protótipo.

4.2 PASSO 1: ANÁLISE DO REPOSITÓRIO

Para a análise do repositório, este estudo dispensou o uso de ferramentas proprietárias ou de engenharia reversa para a obtenção das informações previstas, de acordo com a Figura 3.1. O protótipo implementado utilizou um algoritmo simples para gerar arquivos de texto padronizados referentes aos dados selecionados e que foram obtidos a partir das seguintes linhas de comando:

- SVN log: é uma linha de comando do *Apache Subversion* que fornece um relatório com o histórico das versões do software. Se nenhum argumento for fornecido, o comando apresentará o histórico para todos os arquivos e diretórios internos, incluindo o diretório atual da cópia de trabalho. O comando pode ser refinado com parâmetros para definição do caminho do repositório, quantidade de revisões ou qualquer combinação dos dois. O parâmetro "-v" é utilizado para imprimir, para cada transação, todos os caminhos e arquivos alterados.

- *Java Class Dependency Analyzer* (JDeps): é uma linha de comando do Java fornecida a partir da versão 8. Apresenta dependências em nível de pacote ou de classe, que são úteis para identificar as dependências declaradas estaticamente (dependências estruturais). A linha de comando JDeps lê um conjunto de arquivos de classe binária e apresenta as dependências estáticas com base nas relações de uso entre as mesmas, sendo possível armazenar os dados resultantes em arquivos de texto padronizados (.txt) ou, até mesmo, gerar arquivos com informações estruturais de representação gráfica (.dot), que podem ser reprocessados por ferramentas de visualização de grafos orientados ou redes abstratas.

4.3 PASSO 2: DETERMINAÇÃO DE DEPENDÊNCIAS ESTRUTURAIS E LÓGICAS

Para a Determinação das Dependências Estruturais e Lógicas, o protótipo empregou um algoritmo simples para analisar sintaticamente os arquivos de texto gerados no passo anterior.

Utilizando a análise sintática do arquivo resultante da linha de comando "JDeps", foram geradas listas de dependências estruturais entre pares de elementos que compõem o conjunto de dados selecionados para cada versão do sistema de estudo de caso. Utilizando a análise sintática do arquivo resultante da linha de comando "SVN log", as listas de dependências estruturais foram atualizadas com as dependências lógicas para os pares de classes que compõem o conjunto de dados selecionados para cada versão avaliada.

Embora a representação de grafos seja mais intuitiva para os engenheiros de software, à medida que os números de nós e bordas crescem, a visão pode se tornar inteligível. Por isso, neste estudo, considerou-se o layout de grafos não escalável. Por esta razão, a DSM foi empregada por ser mais eficiente e representativa na exibição de estruturas complexas de arquitetura de software. Portanto, as listas de dependências foram projetadas em matrizes DSM para cada versão do sistema de software em análise.

4.4 PASSO 3: CLASSIFICAÇÃO DAS MUDANÇAS CONJUNTAS

Para a classificação das mudanças conjuntas, os dados selecionados para o sistema de estudo de caso foram avaliados, considerando informações originadas após a execução do passo 2 (dois), detalhado na Seção 4.3. As listas de dependências entre pares de elementos foram projetadas em Matrizes de Dependências Lógicas referentes ao conjunto de dados selecionados para cada versão do sistema de estudo de caso.

O SIPLOM 4 emprega o *Redmine*, um SGP desenvolvido para ambiente web que suporta múltiplos projetos em uma organização. O *Redmine* permite o rastreamento de defeitos e de tarefas atribuídas a equipes de desenvolvimento, bem como possibilita a gestão do projeto via gráfico de *Gantt* e o acompanhamento ao longo do tempo, dentre outras funcionalidades. Este SGP pode ser integrado com o SVN e também permite a configuração de campos personalizados, de acordo com necessidades específicas de cada projeto. O campo "Tipo", que classifica as tarefas, foi personalizado para atendimento às necessidades de controle gerenciais. Essa personalização permitiu a distinção entre atividades de elaboração de documentação, correção de defeitos (internos e externos), atendimento de novos requisitos e solicitações de mudança, por exemplo.

Como mecanismo de integração entre o SCV e o SGP, o SIPLOM 4 prevê a adição de referências nas mensagens informadas pelos autores de cada transação do SVN para a tarefa correspondente do *Redmine*, conforme ilustra a Figura 4.3. O padrão estabelecido prevê a inclusão do caractere # seguido do identificador da tarefa. Neste exemplo, as transações r2400 e r2653 fazem referência às tarefas #10679 e #11105, respectivamente.

Existe outro mecanismo de integração entre o SVN e o Redmine, realizado partir do uso de *web-service*, não explorado neste estudo.

```
r2400 | ricardo.sixel | 2015-10-06 15:18:54 -0300 (ter, 06 out 2015) | 1 line
Caminhos mudados:
   A /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/DocumentoOperacionalSecaoSubdivisaoCelulaBusinessRuleDecorator.java
   D /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/ModeloDocumentoOperacionalSecaoSubdivisaoCelulaBusinessRuleDecorator.java
   A /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/DocumentoOperacionalSecaoSubdivisaoCelulaServiceImpl.java
   D /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/ModeloDocumentoOperacionalSecaoSubdivisaoCelulaServiceImpl.java
   A /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/DocumentoOperacionalSecaoSubdivisaoCelulaService.java
   D /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/ModeloDocumentoOperacionalSecaoSubdivisaoCelulaService.java
   M /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/UsuarioEmpregadoDTO.java
   M /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/ModeloDocumentoOperacionalSecao.java
   A /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/DocumentoOperacionalSecaoSubdivisaoCelula.java
   M /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/EmpregoSubdivisaoCelula.java
   ...

REFS #10679
------------------------------------------------------------------------
r2653 | thiago.leite | 2015-11-10 15:25:39 -0200 (ter, 10 nov 2015) | 1 line
Caminhos mudados:
   M /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/SumarioDiarioSituacaoDTOBusinessRuleDecorator.java
   M /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/UsuarioCelulaEmpregadaBusinessRuleImpl.java
   M /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/UsuarioEmpregadoBusinessRuleImpl.java
   M /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/UsuarioSubdivisaoEmpregadaBusinessRuleImpl.java
   M /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/SumarioDiarioSituacaoDTOServiceImpl.java
   M /branches/4.1-ALPHA_2/codigo/SiplomBusiness/...<package>/SubdivisaoEmpregadaServiceImpl.java
   M /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/SumarioDiarioSituacaoDTO.java
   A /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/SumarioDiarioSituacaoDTOConverter.java
   M /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/SubdivisaoFcEmpregada.java
   ...
REFS #11105
```

FIG. 4.1: Extrato de histórico do SVN do SIPLOM 4. Transações r2400 e r2653.

O *Redmine* permite a exportação de um relatório padronizado, conforme mostrado na Figura 4.2, sobre as tarefas de cada um dos projetos controlados via ferramenta. Neste exemplo, os detalhes das tarefas #10679 e #11105 são exibidos.

	A	B	C	D	E	F	G	H
1	#	Projeto	Tipo	Situação	Título	Delegado por	Atribuído para	Data prevista
2	10679	P5166 - SIPLOM 4	Cronograma	Aprovada	Desenvolver e testar unitariamente: back-end - Associar Células Operacionais às Subdivisões	Marcelo Machado	Ricardo Sixel	14/10/2015
3	11105	P5166 - SIPLOM 4	Defeito Interno	Aprovada	Desabilitar usuários já empregados no contexto da Operação	Leandro Carvalho	Thiago Leite	28/10/2015
4								

FIG. 4.2: Extrato de relatório do *Redmine* do SIPLOM 4. Tarefas #10679 e #11105.

Dessa forma, informação constante na coluna Tipo (Figura 4.2) permitiu a tipificação de cada transação de acordo com as perspectivas de evolução (planejadas em cronograma) e aquelas relacionadas à manutenção (correção de defeitos), por exemplo. A partir da determinação da perspectiva de desenvolvimento dos *branchs*, foi realizado o agrupamento das mudanças. Posteriormente, para cada um dos grupos, foram tipificadas as transações relacionadas aos elementos armazenados nas listas de dependências que compõe o conjunto de dados selecionados.

Para a realização da Classificação de Tarefa, conforme detalha a Seção 3.3, foi utilizado o mecanismo de integração previsto pelo SIPLOM 4 entre o SVN e o Redmine.

A Figura 4.3 exibe um extrato do histórico do SVN do SIPLOM 4, onde é possível identificar duas revisões de código seguidas. Na transação r2673, o arquivo MensagemOperacionalDTO.java foi modificado. Na transação r2674, os arquivos MensagemOperacionalDTOConverter.java e MensagemOperacionalURIs.java foram modificados. O mesmo desenvolvedor realizou ambas revisões de código em atendimento à mesma tarefa referenciada pelo identificador "REF #11151 - Documento operacional".

```
r2673 | fabio | 2015-11-12 11:10:49 -0200 (qui, 12 nov 2015) | 1 line
Caminhos mudados:
   M /branches/4.1-ALPHA_2_20150720/codigo/SiplomDomain/...<package>/MensagemOperacionalDTO.java

REF #11151 - Documento operacional
------------------------------------------------------------------------
r2674 | fabio | 2015-11-12 11:11:54 -0200 (qui, 12 nov 2015) | 1 line
Caminhos mudados:
   D /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/MensagemOperacionalDTOConverter.java
   M /branches/4.1-ALPHA_2/codigo/SiplomDomain/...<package>/MensagemOperacionalDTOURIs.java

REF #11151 - Documento operacional
```

FIG. 4.3: Extrato de histórico do SVN do SIPLOM 4. Transações r2673 e r2674.

Convencionalmente, a determinação das mudanças conjuntas passa pela correspondência exata de transações para um dado par de elementos de software, observáveis no histórico de um SCV. Em uma avaliação tradicional, para o exemplo ilustrado na Figura 4.3, apenas os elementos MensagemOperacionalDTOConverter.java e MensagemOperacionalURIs.java apresentariam *co-changes*, pois mudaram juntos na transação r2674.

Um olhar mais atento para as duas revisões de código mostradas na Figura 4.3 permite constatar que as transações foram realizadas na mesma data, com uma diferença de aproximadamente um minuto entre elas, pelo mesmo desenvolvedor e em atendimento à mesma tarefa. A data da transação é posterior ao período de intervenção no código-fonte, porém o momento da transação é considerado para a avaliação das mudanças conjuntas, assim como a maioria dos estudos correlatos. Não obstante, a razão que levou o desenvolvedor a separar esta atualização de código em duas transações não deve afastar a possibilidade de todos os arquivos mudados para atendimento da tarefa estarem conectados logicamente.

Para o exemplo ilustrado na Figura 4.3, este estudo considera que os três elementos foram modificados simultaneamente. Com isso, informações desprezadas pelos métodos atuais são consideradas e, assim, o método apresentado ampliou o escopo da detecção do acoplamento evolutivo a partir da classificação de tarefas.

A partir da classificação das transações, foi possível realizar, por inferência, a classificação das mudanças conjuntas entre os pares de elementos de acordo com os dados

selecionados para o estudo de caso. As mudanças conjuntas, após a execução deste passo, foram melhor qualificadas e serviram como entrada no processo de cálculo da métrica de impacto do acoplamento evolutivo. Adicionalmente, cada par de elementos, armazenados na listas de dependências lógicas que compõe o conjunto de dados selecionados, foram classificados considerando:

- O Número de Desenvolvedores Distintos (NDD). A partir do nome do autor responsável por cada transação, extraído do histórico de transações do SVN, foi possível executar este passo.
- O Número de Revisões do Par (NRP), somando o número de revisões de cada elemento, de acordo com dados do histórico de transações do SVN.

4.5 PASSO 4: DETECÇÃO DO ACOPLAMENTO EVOLUTIVO

Nesse passo, foram geradas as matrizes HCP, referentes aos dados contidos nas matrizes de dependências lógicas, que armazenam as informações de cada par de elementos modificados conjuntamente e que compõem o conjunto de dados definidos na Seção 4.1.1.

Em seguida, foi realizada a análise das versões consecutivas, aplicando a abordagem da verificação deslizante, cujo algorítmo está detalhado na Seção 3.4.1. Dessa forma, foram obtidas as matrizes SHCP, referentes a esse conjunto de dados para cada versão do sistema de estudo de caso. Utilizando as matrizes SHCP foram identificados manualmente grupos de elementos que possuíam alta probabilidade de estarem impactados pelo acoplamento evolutivo, detalhados nas Seções 4.5.1 e 4.5.2.

4.5.1 GERAÇÃO DE RELATÓRIOS DOS DOCUMENTOS OPERACIONAIS

Um primeiro grupo de elementos, que foi exibido nas matrizes SHCP com alta probabilidade de estarem impactados pelo acoplamento evolutivo, refere-se às classes responsáveis pela geração de relatórios dos Documentos Operacionais, conforme exibido na Figura 4.4. Essas classes foram selecionadas e analisadas manualmente, o que permitiu constatar que, de fato, possuíam dependências lógicas.

Por exemplo, a classe *ReportService* que implementa a *ReportInterface* possuía um método denominado *createReportPDF*, responsável pela geração de arquivos no formato PDF. Nesse método, foi identificada uma declaração condicional "se" que gera os relatórios para todos os três tipos distintos de Documentos Operacionais suportados pela aplicação. Isto introduziu uma dependência lógica entre a classe *ReportService* e as classes terminadas

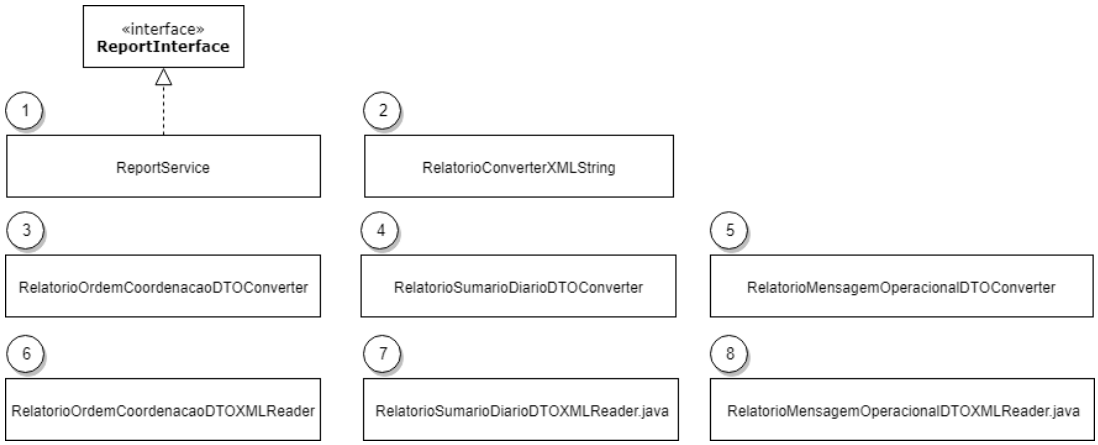

FIG. 4.4: Classes responsáveis pela geração de relatórios dos Documentos Operacionais.

em *DTOXMLReader* (uma para cada tipo de documento), que faz com que, para um novo tipo de documento adicionado, uma nova cláusula "senão" (amostra de código duplicado) tenha que ser inserida no método.

Outro exemplo de dependência lógica identificada neste grupo de elementos refere-se às classes que terminam com *DTOConverter*, responsáveis por converterem os dados da entidade que manipula cada tipo de Documento Operacional preparando-os para atender as requisições do front-end. A classe *RelatorioConverterXMLString*, única para todos os tipos de documento, realizava a conversão de dados recebidos do front-end do formato XML para uma String, antes da conversão para o formato PDF. Essa solução causou a dependência lógica entre as classes terminadas em *DTOConverter* e as classes terminadas em *DTOXMLReader*, contrariando a independência estrutural inicialmente projetada.

4.5.2 REFATORAÇÃO DO EMPREGO DE USUÁRIOS

Um segundo grupo de elementos, que foi exibido nas matrizes SHCP com alta probabilidade de estarem impactados pelo acoplamento evolutivo, refere-se às classes responsáveis pela refatoração do emprego de usuários nos Elementos Organizacionais (EO). Essas classes foram selecionadas e analisadas manualmente, o que permitiu constatar que, de fato, possuíam dependências lógicas.

Desde sua primeira versão, o SIPLOM 4 permitia a manipulação de dados referentes aos EO que representavam a organização estrutural do Comando Operacional Ativado (ComOpA), que são ativados durante a fase de planejamento das Operações Militares (OpM) acompanhadas via sistema.

A criação da estrutura organizacional na fase de planejamento foi um passo importante na estruturação do ComOpA. Isso permitiu que os Militares recebessem uma delega-

ção para atuarem em um dos setores que compõe o Teatro de Operações (TO), com regras de visibilidade específicas, de acordo com suas funções. Cada partícipe da operação deveria ser cadastrado como usuário do SIPLOM 4, de modo que pudesse ser associado a um nó de uma estrutura hierarquizada, denominada Árvore de Operações. Assim, de acordo com sua delegação de competência, receberiam autorizações de visibilidades e controle de acesso específicas.

Nos primeiros lançamentos do sistema, até a V8, o usuário era empregado nos EO que, por sua vez, eram associados ao ComOpA para cada OpM. Cabe destacar que a entidade *ComandoOperacionalAtivado*, na modelagem do sistema, também era uma especialização da entidade *ElementoOrganizacional*. Dessa forma, como vemos no diagrama da Figura 4.5, a implementação das regras de negócio do emprego de usuários dependia das regras de negócio de emprego dos EO.

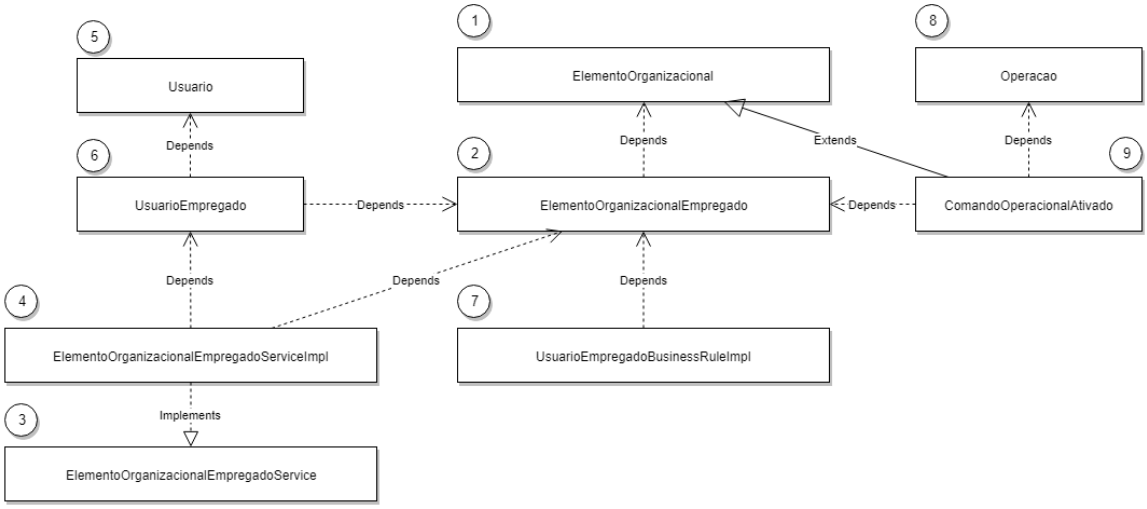

FIG. 4.5: Dependências do objeto responsável pela implementação das regras de negócio de Emprego de Usuários antes da refatoração.

Contudo, com a adição dos Documentos Operacionais, a partir da V9, o emprego de usuários no nível dos EO não foi suficiente para controlar o envio e recebimento de documentos entre as Subdivisões e Células Operacionais. Essa evolução do sistema demandou a modelagem de novas entidades independentes da estrutura do ComOpA. A partir da refatoração, usuários passaram a ser empregados nas Subdivisões e Células Operacionais (não mais nos EO). Conforme o diagrama da Figura 4.6 as regras de negócio do emprego de usuários passaram a não depender das regras que controlam a estrutura da OpM.

A principal diferença entre os diagramas da Figura 4.5 e Figura 4.6 refere-se à estruturação dos EO no contexto de uma OpM, que passou a ser desacoplado das funcionalidades de emprego de Usuários. Somente a Subdivisão possuía dependência estrutural com os

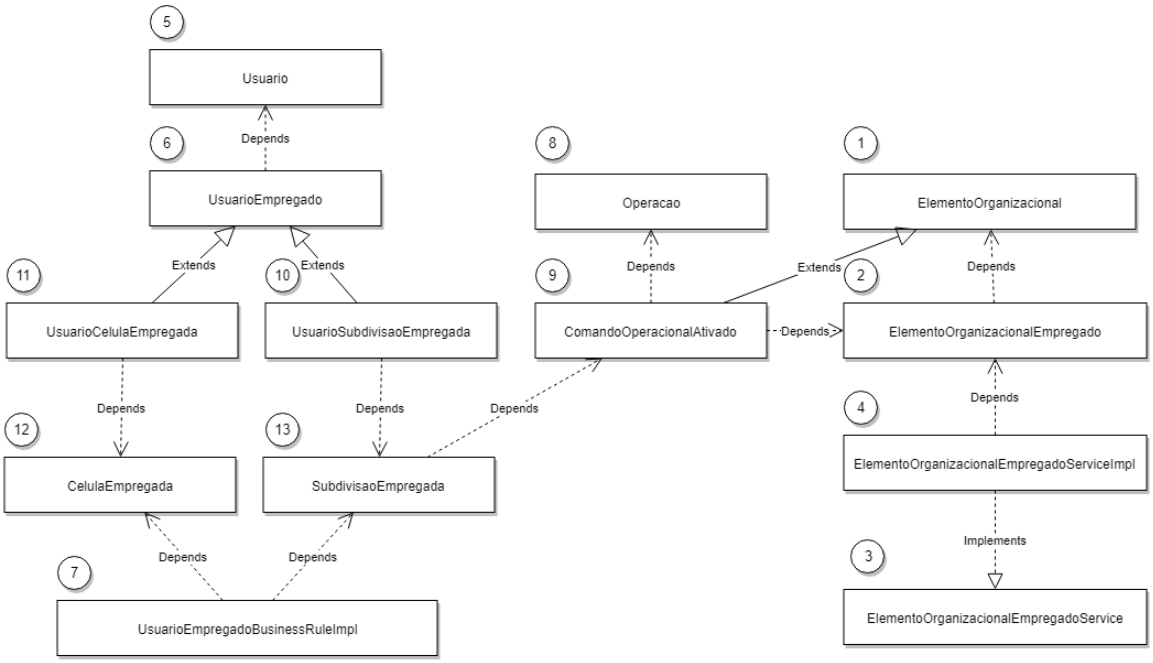

FIG. 4.6: Dependências do objeto responsável pela implementação das regras de negócio de Emprego de Usuários após a refatoração.

EO especializado em um ComOpA. Nas primeiras versões (V0 até V8), por exemplo, toda a visibilidade e permissões de acesso de Usuários eram dependentes da associação aos EO.

Após a refatoração, a visibilidade e controle de acesso foram modificados, passando a depender exclusivamente da associação dos usuários nas Subdivisões e Células Operacionais. A estruturação da OpM passou a não influenciar diretamente nos controles necessários para os usuários do sistema, o que indicou a diminuição do acoplamento evolutivo entre os elementos Usuário e Operação.

O cálculo da métrica IAEv foi aplicado, conforme estabelecido no passo 5 (cinco) (Figura 3.1) da metodologia apresentada no capitulo 3, de modo que foi possível verificar se a métrica se sensibilizou à refatoração que impactou as instâncias de acoplamento evolutivo na arquitetura do sistema de estudo de caso. Esses resultados foram analisados no Capítulo 5.

4.6 PASSO 5: CÁLCULO DA MÉTRICA IMPACTO DO ACOPLAMENTO EVOLUTIVO (IAEV)

Para testar a validade da métrica tanto na perspectiva de evolução quanto na perspectiva de manutenção, o cálculo da métrica foi aplicado aos dois grupos de dados que apresentaram alta probabilidade de estarem impactados pelo acoplamento evolutivo, detalhados na Seção 4.5. Os seguintes passos foram executados:

- Inicialmente, a métrica IAEv foi calculada para o grupo de elementos que validaram a abordagem da verificação deslizante (Sliding-HCP). Dentro da perspectiva de evolução do software, esse grupo de elementos surgiu em atendimento aos novos requisitos que demandavam a **Geração de Relatórios dos Documentos Operacionais**. A partir dos resultados, foi possível estabelecer critérios para definir faixas de valores para o impacto medido pela métrica IAEv.

- Posteriormente, a métrica IAEv foi calculada para um grupo de elementos que foram modificados. Dentro da perspectiva de manutenção, a **Refatoração do Emprego de Usuários** nos EO foi alterado devido ao desenvolvimento dos Documentos Operacionais. A partir dos resultados foi possível verificar se a métrica se sensibilizou adequadamente no processo de manutenção do software.

Complementarmente, foram realizados testes gerais, considerando um conjunto maior de elementos para o sistema de estudo de caso, a fim de determinar as faixas de valores para o Fator de Impacto e para o Fator de Ajuste, previstos na metodologia, conforme detalha a Seção 3.5 do Capítulo 3.

O gráfico exibido na Figura 4.7 mostrou que cerca de 83,5% das mudanças conjuntas do SIPLOM 4 foram realizadas por apenas um desenvolvedor. Aproximadamente 16% foram realizadas por dois desenvolvedores. Menos de 0,5% foram realizadas por três ou mais desenvolvedores. O tamanho da equipe de desenvolvimento interferiu diretamente na determinação destes valores, por isso as faixas do Fator de Impacto devem ser estabelecidas mediante análise dos dados gerais do sistema.

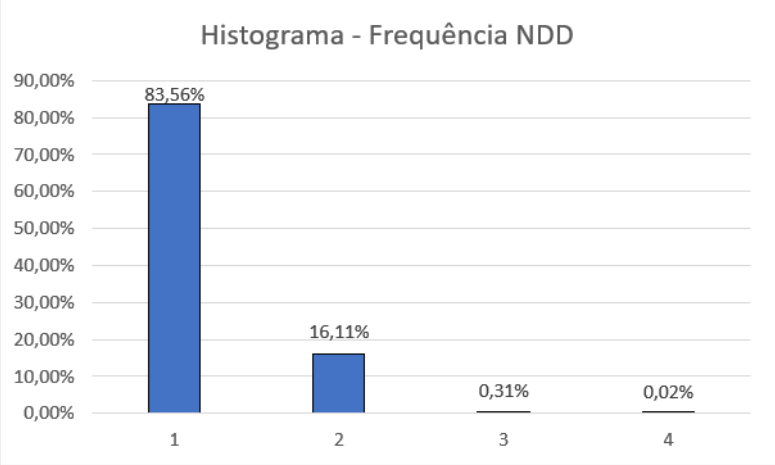

FIG. 4.7: Frequência do Número de Desenvolvedores Distintos (NDD) das mudanças conjuntas observadas no SIPLOM 4.

O gráfico exibido na Figura 4.8 mostrou a distribuição do número de revisão dos pares de elementos observados no SIPLOM 4. Mais de 47% dos pares de elementos foram alterados em uma ou duas revisões. Esses elementos puderam ser observados superando a linha de corte de 10% indicada no gráfico em questão. Cerca de 37% dos pares de elementos foram alterados em três ou mais, e menos de 10 revisões. Esses elementos puderam ser observados superando a linha de corte de 2% indicada nesse gráfico em questão. Os cerca de 16% dos pares de elementos restantes foram alterados em dez ou mais revisões e foram representados abaixo da linha de corte dos 2% indicada nesse gráfico.

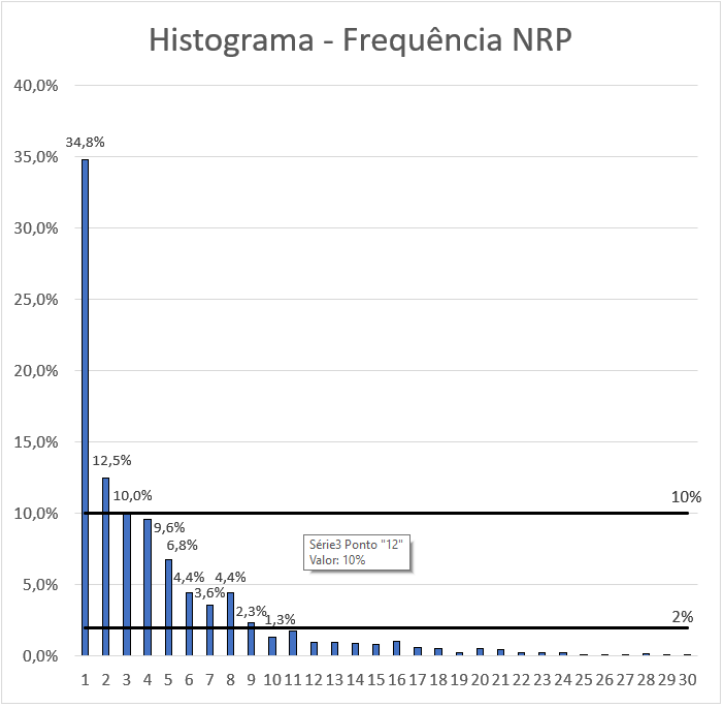

FIG. 4.8: Frequência do Número de Revisão dos Pares (NRP) observados no SIPLOM 4.

Portanto, as faixas de valores estabelecidas para a determinação do fator de impacto para o sistema de estudo de caso foram representadas de acordo com a Figura 4.9.

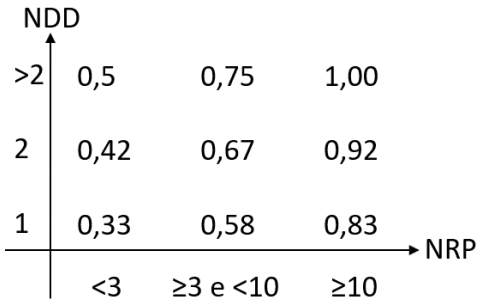

FIG. 4.9: Fator de Impacto da Mudança. Faixas definidas para o SIPLOM 4.

Referente ao número de dependentes lógicos, em versões sob a perspectiva de evolução, o SIPLOM 4 apresentou mais de 94% de elementos conectados logicamente com um número superior a 5 outros elementos. Contudo, em versões sob a perspectiva de manutenção, essa observação apresentou uma distribuição mais equânime. Com isso, para estabelecer as faixas de valores do Fator de Ajuste, os dados do histograma foram divididos em 4 (quatro) faixas, de acordo com o gráfico exibido na Figura 4.10.

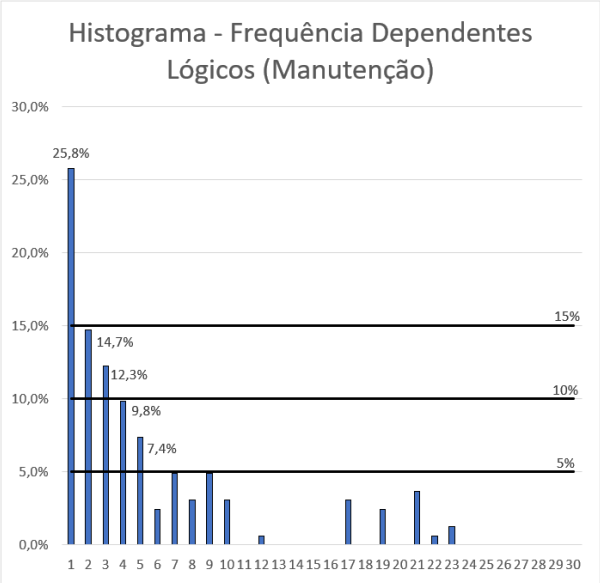

FIG. 4.10: Frequência do Número de Dependentes Lógicos (NDD) sob a perspectiva de Manutenção observados no SIPLOM 4.

Elementos com um dependente lógico foram observados em mais de 25% das amostras, superando o limite de 15%. Elementos com dois e três dependentes lógicos apresentaram cerca de 14% e 12%, respectivamente; estes elementos ficaram na faixa entre 10% e 15%. Elementos com quatro e cinco dependentes lógicos apresentaram cerca de 9% e 7%, respectivamente; estes elementos ficaram na faixa entre 5% e 10%. Elementos acima de cinco dependentes estão distribuídos abaixo do limite inferior de 5%. Portanto, as faixas de valores estabelecidas para a determinação do Fator de Ajuste para o sistema de estudo de caso foram representadas de acordo com a Figura 4.11.

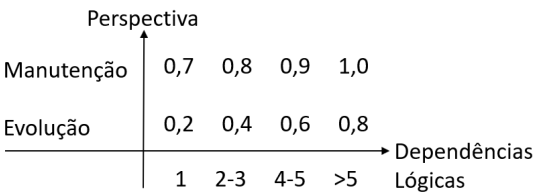

FIG. 4.11: Fator de Ajuste. Faixas definidas para o SIPLOM 4.

4.7 PASSO 6: ANÁLISE DA MÉTRICA IAEV

De posse dos resultados da métrica IAEv e das matrizes SHCP, foi possível executar manualmente o passo 6 (seis), conforme a metodologia prevista no Capítulo 3.

Os valores da métrica IAEv foram obtidos para os elementos envolvidos na geração de relatórios dos documentos operacionais que sustentaram a validação do método *Sliding HCP*. Dessa forma, comparações e análises gráficas puderam ser executadas para que fossem estabelecidas faixas de valores que indicam a alta ou baixa propensão de um dado elemento arquitetural em apresentar o acoplamento evolutivo.

A estratégia inicial traçada para execução deste passo foi a comparação da métrica IAEv com os valores da métrica *Instability*, empregada na validação da abordagem da verificação deslizante proposta por Machado e Choren (2018). A *Instability* foi considerada pelos autores como uma medida inicial para filtrar elementos arquiteturais instáveis propensos a apresentar o acoplamento evolutivo. Contudo, a partir da comparação das métricas, não foi possível verificar indícios que permitissem correlacionar a *Instability* aos elementos propensos a apresentar acoplamento evolutivo na arquitetura de software identificados a partir da Métrica IAEv.

A segunda estratégia para a execução desse passo se mostrou mais adequada: a realização de análises gráficas da variação e da linha de tendência da métrica IAEv para os elementos arquiteturais que tiveram o acoplamento evolutivo confirmado pela abordagem da verificação deslizante. Assim, a partir dessas análises gráficas, os seguintes limites foram estabelecidos:

- Valor entre 0% até 15%: Baixa propensão do elemento arquitetural em apresentar acoplamento evolutivo;
- Valor entre de 15% a 25%: Média propensão do elemento arquitetural em apresentar acoplamento evolutivo;
- Valor entre de 25% a 40%: Elevada propensão do elemento arquitetural em apresentar acoplamento evolutivo;
- Valor acima de 40%: Extrema propensão do elemento arquitetural em apresentar acoplamento evolutivo.

5 ANÁLISE DE RESULTADOS

Neste capítulo os resultados obtidos para a métrica IAEv serão apresentados e as ameaças à validade do estudo serão detalhadas.

5.1 IAEV - GERAÇÃO DE RELATÓRIOS DOS DOCUMENTOS OPERACIONAIS

A Tabela 5.1 apresenta a comparação das métricas IAEv e *Instability* (MARTIN, 1994) para o conjunto de classes envolvidas na Geração dos Relatórios dos Documentos Operacionais detalhada na Seção 4.5.1. O identificador refere-se à numeração dos elementos arquiteturais exibidos na Figura 4.4.

TAB. 5.1: Comparação das métricas IAEv e Instability (MARTIN, 1994) para os elementos avaliados na validação do método Sliding-HCP.

	V9		**V10**		**V11**		**V12**		**V13**	
Id	**IAEv**	**Inst.**	**IAEv**	**Inst.**	**IAEv**	**Inst.**	**IAEv**	**Inst.**	**IAEv**	**Inst.**
1	52%	75%	47%	75%	47%	75%	0%	77%	41%	77%
2	28%	83%	58%	88%	58%	88%	58%	88%	19%	88%
3	22%	94%	22%	94%	22%	94%	22%	94%	0%	95%
4	31%	94%	31%	94%	31%	94%	31%	94%	12%	94%
5	17%	94%	17%	94%	17%	94%	17%	94%	46%	94%
6	28%	86%	58%	89%	40%	89%	40%	89%	46%	89%
7	27%	89%	58%	91%	33%	91%	33%	91%	46%	91%
8	26%	86%	58%	89%	40%	89%	40%	89%	46%	89%

Os elementos arquiteturais que apresentam maior média de instabilidade (soma dos índices de cada versão dividido pelo número de versões) são os elementos 3, 4 e 5. Estes mesmos elementos apresentam os menores índices médios da métrica IAEv. Para os dados selecionados, o elemento 3 apresentou o maior valor absoluto de instabilidade dentre todos os elementos (95%). Contudo, o próprio método da verificação deslizante não confirmou a existência do acoplamento evolutivo nesse elemento em nenhuma das versões subsequentes analisadas. Esse mesmo elemento, portanto, apresentou o menor índice médio da métrica IAEv (18%) e seu valor decaiu a 0 (zero) na V13, o que corrobora com o fato do elemento não apresentar acoplamento evolutivo.

Adicionalmente à comparação entre as métricas IAEv e *Instability*, foram realizadas análises gráficas da variação e da linha de tendência da métrica IAEv para os elementos arquiteturais que tiveram o acoplamento evolutivo confirmado pela abordagem SHCP.

O elemento 1 (ReportService.java), cuja evolução da métrica IAEv está representada graficamente na Figura 5.1, foi adicionado ao sistema ainda na V2. Contudo, a partir da V9, esse elemento apresentou cerca de 50% de propensão ao acoplamento evolutivo. Na V12, o indicador da métrica IAEv decaiu a 0 (zero), entretanto a linha de tendência se manteve crescente e, se extrapolada em um período, indica que, na próxima versão, o elemento tenderia a apresentar aproximadamente 40% de chance de se acoplar evolutivamente com outros elementos da arquitetura. Foi confirmado que o elemento 1 apresentou dependências lógicas com os elementos 6, 7 e 8.

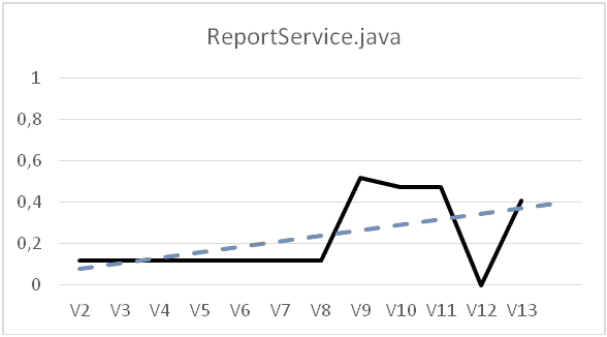

FIG. 5.1: Gráfico com a variação (linha cheia) e tendência (linha pontilhada) da IAEv para o elemento 1 (Tabela 5.1).

O elemento 2 (RelatorioConverterXMLString.java), cuja evolução da IAEv está representada graficamente na Figura 5.2, foi adicionado ao sistema na V9. Esse elemento apresentou cerca de 60% de propensão ao acoplamento evolutivo entre as versões V10 e V12. Na V13, o indicador decaiu para cerca de 20%. A linha de tendência, embora decrescente, quando extrapolada em um período, indica que na próxima versão o elemento tenderia a apresentar aproximadamente 39% de chance de se acoplar evolutivamente com outros elementos da arquitetura. Foi confirmado que o elemento 2 apresentou dependências lógicas com os elementos 6, 7 e 8.

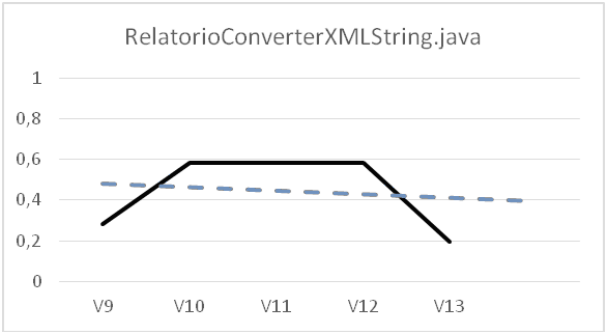

FIG. 5.2: Gráfico com a variação (linha cheia) e tendência (linha pontilhada) da IAEv para o elemento 2 (Tabela 5.1).

O elemento 3 (RelatorioOrdemCoordenacaoDTOConverter.java), cuja evolução da IAEv está representada graficamente na Figura 5.3, foi adicionado ao sistema na V9. Esse elemento apresentou cerca de 20% de propensão ao acoplamento evolutivo entre as versões V9 e V12. Na V13, o indicador decaiu a 0 (zero). A linha de tendência, decrescente, quando extrapolada em um período, indica que na próxima versão o elemento tenderia a apresentar aproximadamente 4% de chance de se acoplar evolutivamente com outros elementos da arquitetura. Foi confirmado que o elemento 3 não depende logicamente de nenhum dos outros elementos considerados nas versões avaliadas. A partir da análise deste exemplo, pode-se inferir que valores muito baixos da IAEv (até 10%) indicam que os elementos não estão, de fato, acoplados evolutivamente.

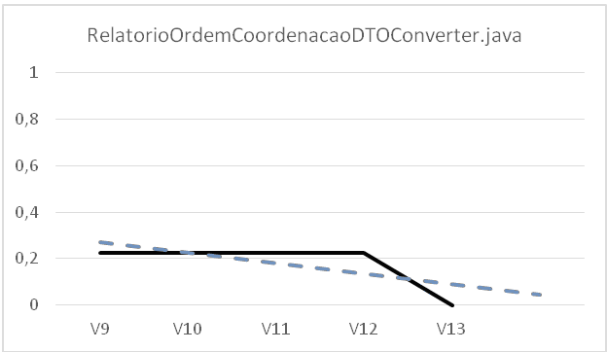

FIG. 5.3: Gráfico com a variação (linha cheia) e tendência (linha pontilhada) da IAEv para o elemento 3 (Tabela 5.1).

O elemento 4 (RelatorioSumarioDiarioDTOConverter.java), cuja evolução da IAEv está representada graficamente na Figura 5.4, foi adicionado ao sistema na V9. Esse elemento apresentou cerca de 30% de propensão ao acoplamento evolutivo entre as versões V10 e V12. A linha de tendência, embora decrescente, quando extrapolada em um período, indica que na próxima versão o elemento tenderia a apresentar aproximadamente 16% de chance de se acoplar evolutivamente com outros elementos da arquitetura. Foi confirmado que o elemento 4 apresentou dependências lógicas com os elementos 6, 7 e 8 nas versões avaliadas. A partir da análise deste exemplo, pode-se inferir que valores da métrica IAEv variando entre 10% e 15% indicam elementos que, de fato, apresentam acoplamento evolutivo.

O elemento 5 (RelatorioMensagemOperacionalDTOConverter.java), cuja evolução da IAEv está representada graficamente na Figura 5.5, foi adicionado ao sistema na V9. Esse elemento apresentou cerca de 20% de propensão ao acoplamento evolutivo entre as versões V9 e V12. Na V13, esse elemento apresentou cerca de 45% de propensão ao acoplamento evolutivo. A linha de tendência crescente, quando extrapolada em um período, indica

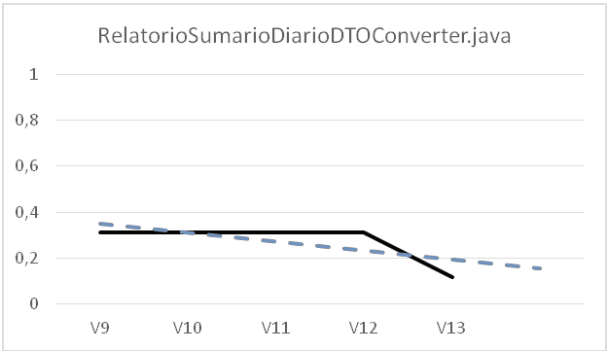

FIG. 5.4: Gráfico com a variação (linha cheia) e tendência (linha pontilhada) da IAEv para o elemento 4 (Tabela 5.1).

que na próxima versão o elemento tenderia a apresentar aproximadamente 40% de chance de se acoplar evolutivamente com outros elementos da arquitetura. Foi confirmado que o elemento 5 apresentou dependências lógicas com os elementos 6, 7 e 8.

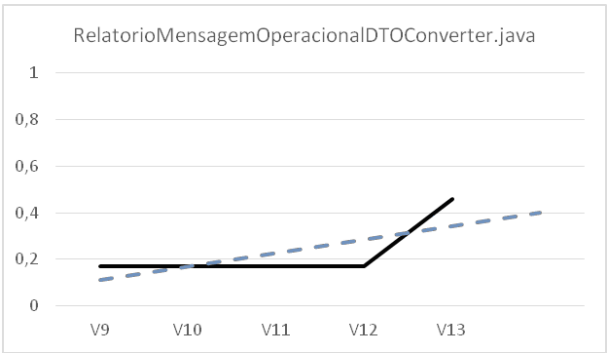

FIG. 5.5: Gráfico com a variação (linha cheia) e tendência (linha pontilhada) da IAEv para o elemento 5 (Tabela 5.1).

O elemento 6 (RelatorioOrdemCoordenacaoDTOXMLReader.java), cuja evolução da IAEv está representada graficamente na Figura 5.6, foi adicionado ao sistema na V9. Esse elemento apresentou cerca de 40% de propensão ao acoplamento evolutivo entre as versões V9 e V13. Na V10, esse elemento chegou a apresentar cerca de 60% de propensão ao acoplamento evolutivo. A linha de tendência crescente, quando extrapolada em um período, indica que na próxima versão o elemento tenderia a apresentar aproximadamente 48% de chance de se acoplar evolutivamente com outros elementos. Foi confirmado que o elemento 6 apresentou dependências lógicas com os elementos 1, 2, 7 e 8.

O elemento 7 (RelatorioSumarioDiarioDTOXMLReader.java), cuja evolução da IAEv está representada graficamente na Figura 5.7, foi adicionado ao sistema na V9. Esse elemento apresentou cerca de 40% de propensão ao acoplamento evolutivo entre as versões V9 e V13. Na V10, esse elemento chegou a apresentar cerca de 60% de propensão ao aco-

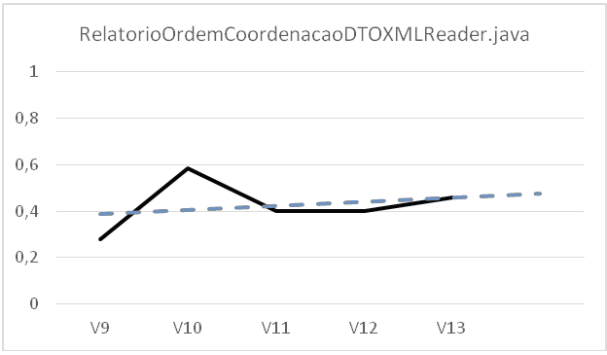

FIG. 5.6: Gráfico com a variação (linha cheia) e tendência (linha pontilhada) da IAEv para o elemento 6 (Tabela 5.1).

plamento evolutivo. A linha de tendência crescente, quando extrapolada em um período, indica que na próxima versão o elemento tenderia a apresentar aproximadamente 43% de chance de se acoplar evolutivamente com outros elementos da arquitetura. Foi confirmado que o elemento 7 apresentou dependências lógicas com os elementos 1, 2, 6 e 8.

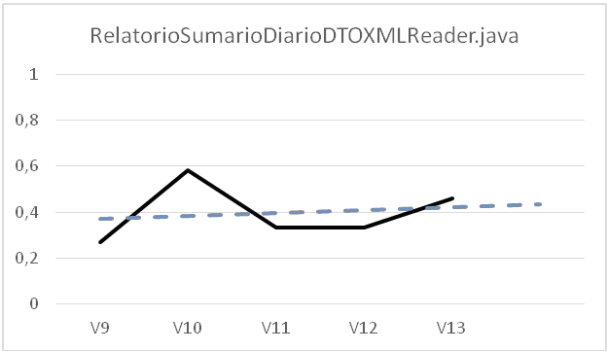

FIG. 5.7: Gráfico com a variação (linha cheia) e tendência (linha pontilhada) da IAEv para o elemento 7 (Tabela 5.1).

O elemento 8 (RelatorioMensagemOperacionalDTOXMLReader.java), cuja evolução da IAEv está representada graficamente na Figura 5.8, foi adicionado ao sistema na V9. Esse elemento apresentou cerca de 40% de propensão ao acoplamento evolutivo entre as versões V9 e V13. Na V10, esse elemento chegou a apresentar cerca de 60% de propensão ao acoplamento evolutivo. A linha de tendência crescente, quando extrapolada em um período, indica que na próxima versão o elemento tenderia a apresentar aproximadamente 49% de chance de se acoplar evolutivamente com outros elementos. Foi confirmado que o elemento 8 apresentou dependências lógicas com os elementos 1, 2, 6 e 7.

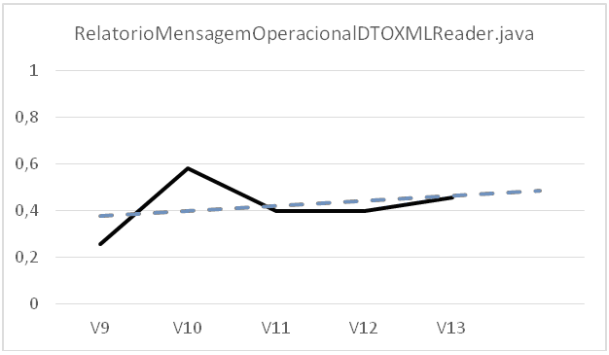

FIG. 5.8: Gráfico com a variação (linha cheia) e tendência (linha pontilhada) da IAEv para o elemento 8 (Tabela 5.1).

5.2 IAEV - REFATORAÇÃO DO EMPREGO DE USUÁRIOS

Os resultados da métrica IAEv apresentados na Tabela 5.2 referem-se aos elementos envolvidos na refatoração detalhada na Seção 4.5.2, **antes de sua execução**. O identificador refere-se à numeração das classes exibidas na Figura 4.5.

TAB. 5.2: Métrica IAEv para os elementos envolvidos na refatoração do Emprego de Usuários na estrutura da Operação Militar aplicadas às versões V0 até a V8.

ID	V0	V1	V2	V3	V4	V5	V6	V7	V8
1	21%	0%	0%	0%	0%	0%	0%	0%	0%
2		26%	26%	26%	26%	26%	26%	26%	26%
3		39%	39%	39%	39%	39%	39%	39%	39%
4		23%	23%	23%	23%	23%	23%	23%	23%
5	18%	24%	18%	18%	18%	18%	18%	18%	18%
6		26%	26%	26%	26%	26%	26%	26%	26%
7		39%	39%	39%	39%	39%	39%	39%	39%
8	21%	18%	18%	18%	18%	36%	36%	36%	36%
9	27%	39%	39%	39%	39%	39%	39%	39%	39%

Os resultados da métrica IAEv apresentado na Tabela 5.3 referem-se aos elementos envolvidos na refatoração detalhada na Seção 4.5.2, **após a sua execução**. O identificador refere-se à numeração das classes exibidas na Figura 4.6.

A partir dos resultados exibidos nas Tabelas 5.2 e 5.3, foi possível executar análises gráficas com a variação da métrica IAEv para todo o período considerado na avaliação da deterioração da arquitetura do sistema de software de estudo de caso, desde a V0 até a V13. Portanto, foi possível verificar o comportamento da métrica proposta neste estudo durante uma refatoração que, conforme discutido na Seção 4.5.2, afastou as dependências lógicas existentes entre os elementos Usuário e Operação Militar, quando foi alterada a forma de emprego dos usuários nos EO.

TAB. 5.3: Métrica IAEv para os elementos envolvidos na refatoração do Emprego de Usuários na estrutura da Operação Militar aplicadas às versões V9 até a V13.

ID	V9	V10	V11	V12	V13
1	27%	27%	27%	27%	27%
2	17%	17%	17%	17%	17%
3	9%	9%	9%	9%	27%
4	7%	40%	40%	40%	16%
5	0%	0%	0%	0%	0%
6	12%	12%	12%	12%	12%
7	20%	20%	20%	20%	20%
8	12%	12%	12%	12%	23%
9	39%	39%	39%	39%	39%
10	39%	39%	39%	39%	39%
11	36%	40%	40%	40%	40%
12	32%	32%	32%	32%	32%
13	25%	25%	25%	25%	25%

Os três gráficos ilustrados nas Figuras 5.9 , 5.10 e 5.11 apresentam o comportamento da métrica IAEv para as classes *Usuario*, *Operacao* e *ElementoOrganizacional*.

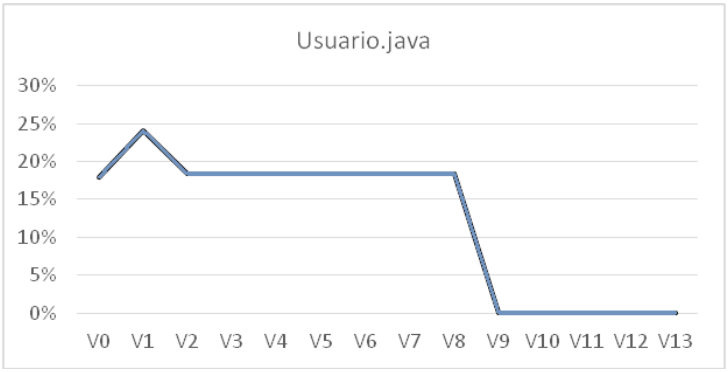

FIG. 5.9: Variação da IAEv para a classe *Usuario*, de acordo com dados exibidos nas Tabelas 5.2 e 5.3.

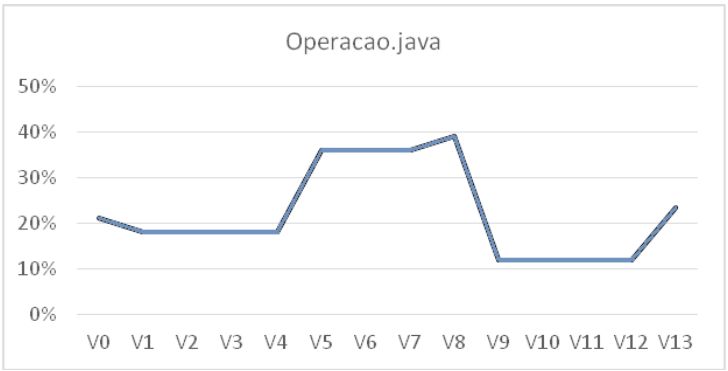

FIG. 5.10: Variação da IAEv para a classe *Operacao*, de acordo com dados exibidos nas Tabelas 5.2 e 5.3.

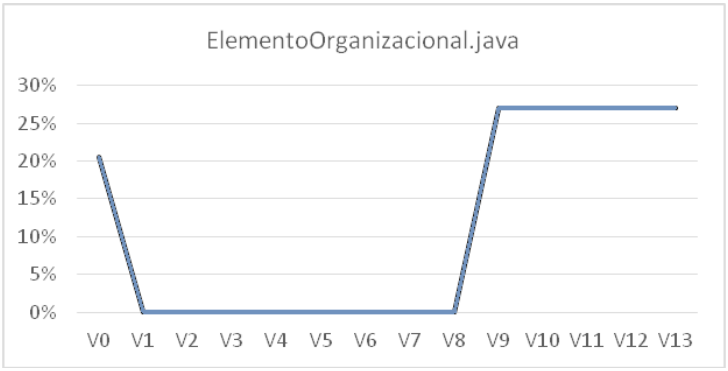

FIG. 5.11: Variação da IAEv para a classe *ElementoOrganizacional*, de acordo com dados exibidos nas Tabelas 5.2 e 5.3.

É possível constatar nestes primeiros gráficos que, no momento da refatoração, a métrica se sensibilizou e alterou bruscamente seus índices. Após a refatoração do emprego de usuário, a métrica captou a diminuição das dependências lógicas para as classes *Usuario* e *Operacao*, que tiveram seus índices de IAEv reduzidos de 18% (média) e 39% (elevada) na V8 para 0% (baixa) e 12% (baixa) a partir da V9, respectivamente.

A partir dos diagramas ilustrados nas Figuras 4.5 e 4.6, foi possível verificar as mudanças que se traduziram em melhoria do indicador para as classes *Usuario* e *Operacao*. Com o surgimento do conceito de Subdivisão e Células, a classe *SubdivisaoEmpregada* passou a depender estruturalmente da classe *ComandoOperacionalAtivado*, sendo esta uma especialização de *ElementoOrganizacional*. Até a V8, enquanto o Usuário estava ligado diretamente aos EO, havia forte acoplamento lógico entre as entidades responsáveis pela manipulação de Usuários e Operações, que não possuíam relação estrutural entre si.

Entretanto, devido à adição de dependências estruturais entre o emprego da Subdivisão e o Comando Operacional Ativado (especialização de *ElementoOrganizacional*), foram observadas novas dependências lógicas indesejadas para a entidade *ElementoOrganizacional*. A variação da IAEv, representada na Figura 5.11, é crucial para entender o impacto negativo do surgimento do acoplamento evolutivo na evolução do software. A classe *ElementoOrganizacional*, excetuando a primeira versão (V0) - quando foi adicionada ao sistema - não apresentou indicativo de acoplamento evolutivo até a V8. Na V9, a métrica IAEv subiu e se manteve em 27% (Elevada) até a última versão considerada neste estudo (V13).

As novas entidades adicionadas ao sistema após refatoração, apresentadas na Tabela 5.3 (elementos 10, 11, 12 e 13), referem-se ao emprego de Usuários em Subdivisões ou Células e ao emprego destes novos elementos no Comando Operacional. Estes elementos adicionados apresentaram indicadores da métrica IAEv variando entre 25% e 40%,

considerados elevados. Assim, embora a refatoração tenha diminuído as dependências lógicas entre Usuário e Operação, as novas entidades surgiram apresentando um efeito colateral: alta propensão de incidir acoplamento evolutivo entre si, além de comprometer a classe *ElementoOrganizacional* exibida na Figura 5.11.

Em estudo recente, Ajienka e Capiluppi (2017) abordaram a relação linear entre as dependências estruturais e lógicas e constataram que as métricas de acoplamento evolutivo podem superar as métricas estruturais na identificação das classes que podem ser afetadas por uma determinada solicitação de mudança. Seus resultados mostraram que alterações no acoplamento estrutural, na maioria dos casos, impactava as dependências lógicas. A métrica IAEv corrobora com a constatação de Ajienka e Capiluppi (2017), pois pode ser verificado que a refatoração, que envolveu a reestruturação das dependências estruturais impactou as dependências lógicas (eliminando algumas e originando outras).

Os três gráficos ilustrados nas Figuras 5.12 , 5.13 e 5.14 indicam melhorias após a refatoração, justamente nos elementos arquiteturais que cuidavam do emprego de Usuários nos EO. A refatoração ajudou a diminuir os impactos das dependências lógicas nestes elementos. A classe *ElementoOrganizacionalEmpregado* que apresentava indicador da métrica IAEv em cerca de 26% (elevada), após a refatoração passou a apresentar indicador da métrica IAEv na faixa de 17% (média). A classe *UsuarioEmpregado* que apresentava indicador da métrica IAEv em cerca de 26% (elevada), após a refatoração passou a apresentar indicador da métrica IAEv na faixa de 12% (baixa). A classe *UsuarioEmpregadoBusinessRuleImpl* que apresentava indicador da métrica IAEv em cerca de 39% (elevada), após a refatoração passou a apresentar indicador da métrica IAEv na faixa de 20% (média).

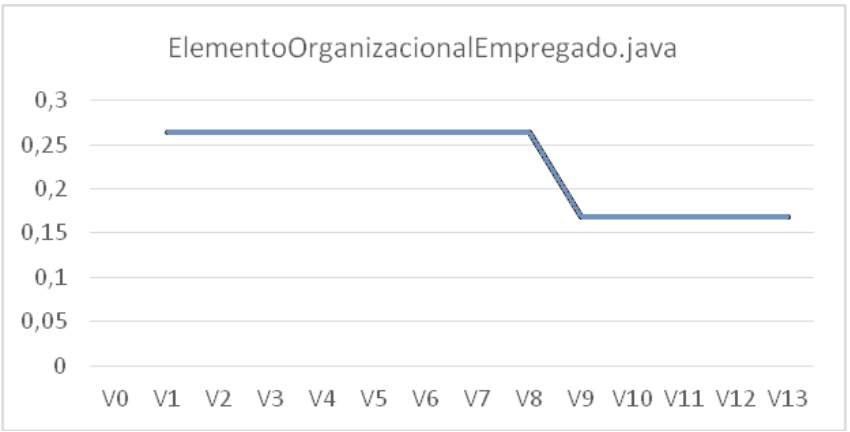

FIG. 5.12: Variação da IAEv para a classe *ElementoOrganizacionalEmpregado*, de acordo com dados exibidos nas Tabelas 5.2 e 5.3.

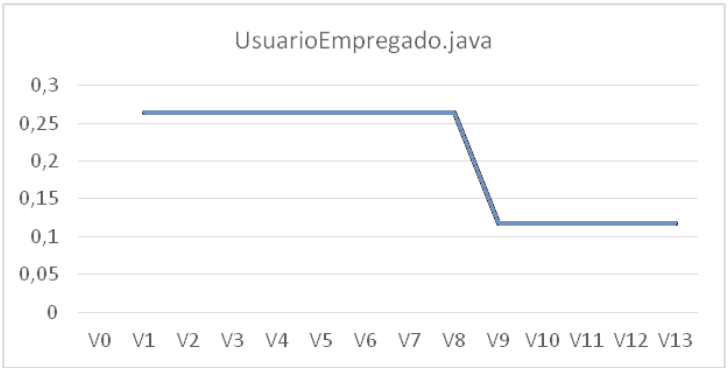

FIG. 5.13: Variação da IAEv para a classe *UsuarioEmpregado*, de acordo com dados exibidos nas Tabelas 5.2 e 5.3.

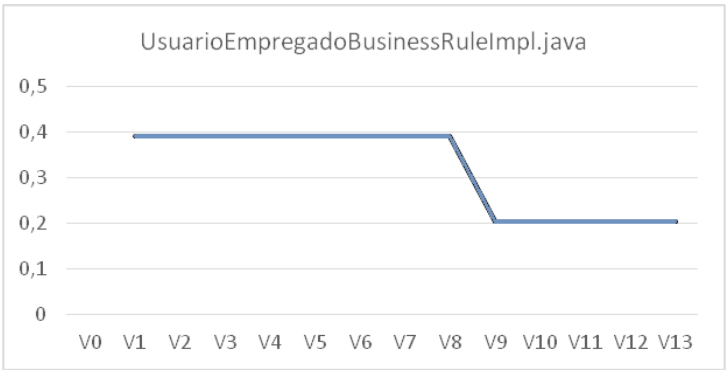

FIG. 5.14: Variação da IAEv para a classe *UsuarioEmpregadoBusinessRuleImpl*, de acordo com dados exibidos nas Tabelas 5.2 e 5.3.

Os dois gráficos ilustrados nas Figuras 5.15 e 5.16 mostram o comportamento da métrica IAEv para a interface *ElementoOrganizacionalEmpregadoService* e sua implementação: *ElementoOrganizacionalEmpregadoServiceImpl*. A interface responsável pelos serviços que cuidam do emprego dos EO apresentava indicador da métrica IAEv em 39% (elevada) até a V8. Após a refatoração, essa interface passou a apresentar indicador da ECIE em 9% (baixa). Sua implementação apresentava indicador da métrica IAEv em 23% (média) até a V8. Após a refatoração, chegou a apresentar indicador da ECIE em 7% (baixa). Nas versões seguintes, sob a perspectiva de manutenção, passou a apresentar indicador da métrica IAEv variando entre 16% (média) e 40% (elevada). Assim, embora a interface tenha melhorado o indicador da métrica IAEv (de elevada para baixa), sua implementação apresentou piora do indicador desse mesmo índice (de baixa para elevada).

Cabe destacar que indicadores baixos ou médios da métrica IAEv não confirmam, por si só, que os elementos arquiteturais deixaram de apresentar acoplamento evolutivo. De modo análogo, um indicador elevado ou extremamente elevado, por si só, não confirma a existência de acoplamento evolutivo. A métrica IAEv calcula uma probabilidade que

FIG. 5.15: Variação da IAEv para a classe *ElementoOrganizacionalEmpregadoService*, de acordo com dados exibidos nas Tabelas 5.2 e 5.3.

FIG. 5.16: Variação da IAEv para a classe *ElementoOrganizacionalEmpregadoServiceImpl*, de acordo com dados exibidos nas Tabelas 5.2 e 5.3.

indica a propensão de um determinado elemento arquitetural apresentar deterioração arquitetural, devido ao acoplamento evolutivo. Quando o indicador da métrica IAEv está elevado ou extremamente elevado para um determinado elemento arquitetural, isso significa que o arquiteto deve investigar mais o elemento para determinar se, de fato, apresenta acoplamento evolutivo.

Para ilustrar esse conceito, as constatações do gráfico ilustrado na Figura 5.16 foram melhor investigadas. Foi realizada uma análise para tentar confirmar a indicação da métrica IAEv, aplicando-se duas outras métricas aos elementos em questão. Dessa forma, os dois gráficos ilustrados nas Figuras 5.17 e 5.18 apresentam a variação das métricas LOC e WMC para a classe *ElementoOrganizacionalEmpregadoServiceImpl*, que teve indicação de ter sido acometida pelo acoplamento evolutivo após a refatoração discutida na Seção 4.5.2.

A medida de complexidade mais básica para um elemento de software é o número de linhas de código (LOC). Essa métrica simplesmente conta as linhas de código executável e declarações de dados, excetuando comentários. Embora esta medida seja extremamente simples, pode ser correlacionada com a quantidade de erros nos programas (BASILI; PER-

RICONE, 1984). A métrica *Weighted Methods per Class* (WMC) proposta por Chidamber e Kemerer (1991) é definida pelo peso das complexidades (Ci) de todos os métodos de uma classe e pode ser representada pela Equação 5.1.

$$WMC = \sum_{i=1}^{n} Ci \tag{5.1}$$

A WMC é um indicador de quanto esforço é necessário para desenvolver e manter uma classe. Embora essa métrica permita definir a noção de complexidade para cada método, isto não é previamente estabelecido, permitindo aumentar o grau de generalidade da métrica WMC. Dessa forma, com todas as complexidades mantidas com valor unitário (Ci=1), o resultado da métrica WMC é o número de métodos do elemento (n).

Conforme pode ser observado na Figura 5.17, ao ser adicionado na V1, a classe *ElementoOrganizacionalEmpregadoServiceImpl* somava 121 linhas de código, o que se manteve até a V8. A partir da V9, essa classe superou a marca de 900 linhas de código, chegando a 1222 linhas de código na V13. Já a Figura 5.18, que apresenta a variação da métrica WMC para essa mesma classe, indica que, ao ser adicionada, na V1 esta somava 5 métodos. Esse índice se manteve estável até a V8. A partir da V9, este elemento passou a contar com 19 métodos, chegando a 25 na V13.

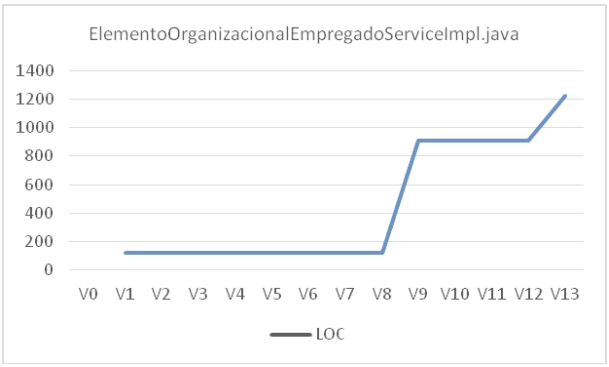

FIG. 5.17: Variação da métrica LOC para a classe *ElementoOrganizacionalEmpregadoServiceImpl* entre as versões V1 e V13.

Após execução da análise manual do código-fonte, foi confirmado que, embora as classes que tratam dos EO não tenham mantido relações estruturais com os objetos relacionados a Usuários após a refatoração detalhada na Seção 4.5.2, alguns serviços foram mantidos indevidamente na classe *ElementoOrganizacionalEmpregadoServiceImpl*.

Foi constatado que essa classe virou um "repositório de consultas para todas as relações de emprego nos EO". Por exemplo, o método *retrieveListaUsuarioDestinoDocumento*, responsável por recuperar todos os destinatários de um determinado Documento Operaci-

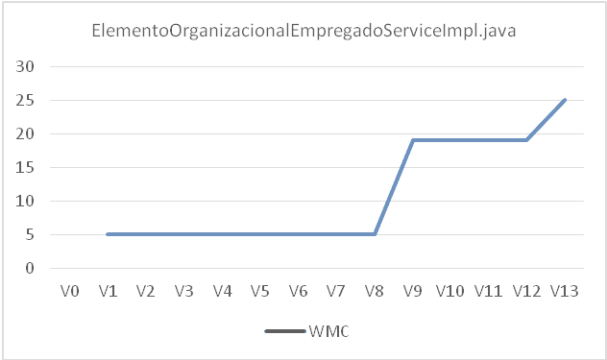

FIG. 5.18: Variação da métrica WMC para a classe *ElementoOrganizacionalEmpregadoServiceImpl* entre as versões V1 e V13.

onal, foi um exemplo que confirmou o acoplamento evolutivo detectado. Este método que tratava de recuperação de informações de usuários deveria estar implementado na classe que cuidava dos serviços de Usuário (*UsuarioServiceImpl*), e não na classe que tratava de serviços de emprego dos EO. Além disso, outros métodos foram criados nessa classe por conveniência da equipe de desenvolvimento. A consequência disso foi o surgimento do acoplamento evolutivo detectado pela métrica IAEv.

Dessa forma, a classe apresentada na Figura 5.16 destoava das outras classes analisadas. A maioria dos elementos envolvidos na refatoração apresentaram melhora do indicador da métrica IAEv. Justamente na classe que apresentou uma piora desse indicador, foi possível identificar métodos "esquecidos" pelos desenvolvedores. Isso corroborou com a validação da métrica, permitindo inferir que variações bruscas da métrica IAEv podem, efetivamente, indicar a deterioração arquitetural causada devido ao acoplamento evolutivo durante o processo de evolução do software.

5.3 AMEAÇAS À VALIDADE

Foi selecionado um projeto representativo para aplicar o método proposto e validar a nova métrica. Para a análise dos resultados, foi necessário um repositório de código-fonte integrado com um SGP, que possuísse a indicação das tarefas no SCV. Uma objeção que pode ser sugerida a este trabalho é devido aos resultados apresentados terem sido obtidos de um sistema com viés Militar e, assim, não terem representatividade em sistemas com viés distinto. No entanto, o SIPLOM 4 possui funcionalidades comuns aos SIG e, complementarmente, incorpora as funcionalidades específicas da área Militar. Os dados selecionados para a validação dos resultados referem-se às funcionalidades que se assemelham aos SIG, principalmente aqueles de controle de *workflow* de documentos.

Devido aos resultados estarem baseados apenas em um sistema de estudo de caso, pode ser sugerida a objeção quanto aos resultados não poderem ser generalizados para outros projetos e domínios. Para mitigar essa ameaça, foram utilizados vários lançamentos que compreendem um período de 3 anos de desenvolvimento na análise dos resultados. Complementarmente, o conjunto de elementos arquiteturais selecionados para validação da metodologia foi comprovadamente refatorado, sendo possível testar a validade da nova métrica no momento de uma evolução adaptativa na arquitetura do software.

Outra objeção relevante é quanto ao projeto ser de código fechado e, assim, não poder inferir se os resultados se aplicam a projetos de código aberto com a mesma eficácia. Contudo, os sistemas de código aberto disponíveis na web não são acompanhados dos SGP que detalham as tarefas que originaram as revisões de código, tal como foi estabelecido na metodologia dessa pesquisa. Contudo, essa metodologia pode facilmente ser reproduzida em sistemas de código aberto, desde que as informações provenientes do SGP sejam obtidas por meio de outro processo a ser estabelecido pelo arquiteto.

O sistema de estudo de caso foi desenvolvido em Java e, neste caso, o conceito de dependência é aquele fornecido pela linha de comando JDeps. A objeção que se apresenta é se os resultados continuariam válidos caso o processo de definição das dependências estruturais seja realizado em sistemas desenvolvidos em outras linguagens ou obtidos por meio de métodos distintos. Devido à semelhança estrutural de linguagens orientadas a objetos, possivelmente os resultados gerais apresentados neste estudo podem ser generalizados a todo o grupo de linguagens orientadas a objetos. Entretanto, outros paradigmas de desenvolvimento vão requerer investigações adicionais.

Foi observado que a entrada de informações sobre a tarefa na transação que motivou uma revisão de código, dependendo do membro da equipe de desenvolvimento, não era informada. O número de revisões sem essa informação no sistema de estudo de caso foi inferior a 3% e o problema foi contornado utilizando a perspectiva do *branch* para classificar tais ocorrências e mitigar a ameaça. Contudo, projetos que tenham um número elevado de revisões não classificadas, devido à falha no processo de integração entre o SCV e SGP, podem ter os resultados comprometidos.

6 TRABALHOS RELACIONADOS

Estudos recentes, avaliados durante atividades de pesquisa e revisão sistemática da bibliografia sobre o tema, apresentaram diferentes abordagens para quantificar o acoplamento evolutivo e avaliar as implicações desse tipo de deterioração na arquitetura de software. Esta seção apresenta os principais trabalhos relacionados a este estudo.

6.1 COMPREENDENDO A INTERAÇÃO ENTRE O ACOPLAMENTO LÓGICO E ESTRUTURAL DE CLASSES DE SOFTWARE

Ajienka e Capiluppi (2017) quantificaram a sobreposição ou interseção de dependências estruturais e lógicas entre classes de sistemas de software sob o paradigma da orientação a objetos. Complementarmente, os autores calcularam estatisticamente a correlação entre os pontos fortes das dependências lógicas e estruturais, com o objetivo de determinar a estabilidade dos sistemas.

Embora não haja evidências fortes de correlação linear entre os pontos fortes dos tipos de acoplamento, Ajienka e Capiluppi (2017) demonstraram que há evidências substanciais para concluírem que os pares de classes estruturalmente acoplados geralmente também incluem dependências lógicas. No entanto, nem todos os pares de classes alterados também estão ligados por dependências estruturais. Os resultados apontados ainda indicam a presença de uma pequena sobreposição entre dependências estruturais e lógicas, na maioria dos projetos de software avaliados.

Para avaliarem o nível de estabilidade das dependências estruturais, Ajienka e Capiluppi (2017) determinaram quais pares de classes estavam estruturalmente acoplados e quantas vezes eles foram alterados simultaneamente. Os autores dividiram em duas partes iguais o ciclo de vida (período em que o par de classes apresenta dependências estruturais). Posteriormente, contaram as *co-changes* para cada metade do ciclo de vida e classificaram da seguinte forma:

- Pares estáveis: são observadas *co-changes* apenas na primeira metade;
- Pares parcialmente estáveis: são observadas *co-changes* nas duas metades, mas com a maior parte na primeira metade;

- Pares parcialmente instáveis: são observadas *co-changes* nas duas metades, igualmente ou com a maior parte na segunda metade. Isso indica que, mesmo após estabelecida a relação estrutural, o par de classes continua apresentando necessidade de manutenção ao mesmo tempo.

- Pares instáveis: somente são observadas *co-changes* na segunda metade. A ligação lógica entre o par de classes é estabelecida somente em um momento futuro (após o surgimento da dependência estrutural).

Os resultados apresentaram uma estabilidade estrutural dos pares de classes, separadas na amostra geral de projetos avaliados pelos autores. Ajienka e Capiluppi (2017) constataram ainda que métricas de acoplamento evolutivo podem superar as métricas estruturais na identificação das classes que podem ser afetadas por uma determinada solicitação de mudança. Seus resultados mostraram que alterações no acoplamento estrutural, na maioria dos casos, impacta nas dependências lógicas.

Este trabalho diferencia-se da abordagem analisada por Ajienka e Capiluppi (2017), pois foram concentrados esforços para avaliar as relações lógicas estabelecidas exclusivamente entre pares de arquivos não conectados estruturalmente.

6.2 MELHORANDO A PRECISÃO DA DETECÇÃO DO ACOPLAMENTO EVOLUTIVO PELA MÉTRICA *CHANGE CORRESPONDENCE*

Mondal et al. (2014) propuseram uma nova métrica para melhorar a precisão da detecção do acoplamento evolutivo, denominada *Change Correspondence*. Os autores observam que a regra de associação usada para expressar o acoplamento evolutivo depende apenas do número de vezes que as entidades foram alteradas, sem considerar se realmente as entidades estão, de fato, relacionadas. A métrica *Change Correspondence* combina a ideia de localização em uma base de código para determinar se as mudanças nas entidades alteradas são correspondentes e, portanto, se elas estão realmente relacionadas.

Os resultados de investigação em quatro sistemas, escrito em duas linguagens de programação distintas, mostraram que a métrica proposta pelos autores tem o potencial de determinar com precisão se duas entidades estão relacionadas, mesmo que elas tenham apresentado poucas modificações simultâneas observadas.

A métrica proposta por Mondal et al. (2014) emprega o conceito de localização em uma base de código e utiliza palavras semanticamente semelhantes para uma determinada consulta (consistindo em uma ou mais palavras), de modo que seja possível identificar

entidades do programa (por exemplo, métodos) contendo as palavras que entraram na consulta (em termos de identificadores e comentários). As entidades resultantes provavelmente implementam o conceito subjacente da consulta e, portanto, os autores consideram estas entidades como relacionadas.

O cálculo dessa medida envolve a inspeção automática das linhas de código-fonte modificadas e dos métodos alterados para inferir se as *co-changes* são correspondentes. Distintamente, neste estudo, a detecção do acoplamento evolutivo ocorre em nível de classe, sem a necessidade de avaliar o conteúdo em nível de código/método. A avaliação semântica pode resultar em problemas de performance em sistemas de grande porte, como o caso do sistema de estudo de caso deste trabalho. O estudo de Mondal et al. (2014) não discute amplamente o porte dos sistemas em que o método foi avaliado e não apresenta resultados comparativos de performance da métrica proposta.

6.3 IDENTIFICANDO E QUANTIFICANDO A DÍVIDA ARQUITETURAL

Xiao et al. (2016) calculam o custo de manutenção a partir da análise da propagação de defeitos na arquitetura do software. O método utiliza representação matricial do acoplamento evolutivo, onde cada célula na matriz exibe o número de vezes que dois arquivos mudaram juntos. Para manifestar como uma mudança em um arquivo impacta em outros arquivos, foi proposto um modelo denominado *"history coupling probability" (HCP)*, detalhado na Seção 2.6. O conceito proposto por Xiao et al. (2016) foi aprimorado ao empregar o método da janela deslizante para melhorar a detecção do acoplamento evolutivo, conforme detalhado na Seção 3.4.1.

Os autores modelaram grupos de elementos conectados que acumulam altos custos de manutenção, indicando dívidas arquiteturais. Para quantificar a deterioração, a dívida arquitetural foi definida formalmente e foi proposto um método para identificar automaticamente os grupos de elementos comprometidos. Adicionalmente, os custos de manutenção foram quantificados, de tal modo que foi possível modelar esses custos ao longo do tempo. Foram avaliados neste estudo sete projetos de código aberto de grande escala e demonstrado que uma parcela significativa da manutenção total do projeto é consumida pelo pagamento de "juros" sobre as dívidas arquiteturais.

Xiao et al. (2016) calculam as probabilidades diretas e transitivas em cada versão na matriz, que manifesta a probabilidade de mudança em um arquivo quando outro for alterado. Assim, o cálculo do custo de manutenção se diferencia daquele apresentado neste estudo, pois entende-se que o acoplamento evolutivo não pode ser simplesmente detectado

utilizando uma única versão de um sistema de software, mas sim em uma série contínua de lançamentos. Além disso, o trabalho de Xiao et al. (2016) não realiza a classificação das mudanças conjuntas que impõem maior significado na análise do acoplamento evolutivo.

6.4 MONITOR DE SAÚDE DA ARQUITETURA DE SOFTWARE

Cai e Kazman (2016) propuseram um método para monitoramento da arquitetura de software, onde são identificados arquivos ou módulos propensos à propagação de defeitos, devido ao acoplamento evolutivo. Embora o método apresentado pelos autores torne a deterioração arquitetural visível em uma análise automática e contínua, a detecção das matrizes de dependências históricas (*History DSM*) consideram apenas mudanças conjuntas observáveis na mesma transação do SCV.

Cai e Kazman (2016) utilizaram o método proposto por Mo et al. (2015) para definirem formalmente e detectarem automaticamente os *Bad Smells* (FOWLER; BECK, 1999) identificáveis na arquitetura de software. Para identificarem as dependências lógicas, os autores consideraram mudanças conjuntas como sendo o número de vezes que um arquivo x muda junto com um arquivo y em um período de tempo, sem maiores classificações, como foi o caso deste estudo.

A metodologia apresentada neste trabalho aperfeiçoa essa detecção das mudanças conjuntas, pois utiliza características de tarefas para agrupar arquivos que mudam em transações próximas temporalmente e que estão relacionadas à mesma correção de defeitos ou implementação de nova funcionalidade.

Cai e Kazman (2016) afirmam desconhecer uma medida de dados históricos que tenha sido utilizada para comparar a arquitetura de software com uma métrica real. Isso porque, segundo os autores, a informação fornecida após a geração do histórico é obtida quando as penalidades já se acumulam no software. No entanto, os desvios no projeto arquitetural são as principais causas da deterioração arquitetural (Eick et al., 2001; Vogel et al., 2011; Li et al., 2016).

Pelos motivos expostos, este estudo considera que a determinação do acoplamento evolutivo pode ser utilizada como uma métrica que compara a arquitetura de software a partir dos dados históricos. A IAEv permite ao arquiteto de software identificar a deterioração arquitetural mesmo em estágios iniciais do desenvolvimento de software e, tão logo surjam evidências da existência do acoplamento evolutivo, estes podem selecionar os componentes que são fortes candidatos a refatoração.

Para determinar a deterioração arquitetural, Cai e Kazman (2016) utilizam a métrica

Decouplin Level (DL) proposta por Mo et al. (2016) que, embora utilize informações sobre as dependências lógicas como uma medida indireta, visa quantificar a manutenibilidade do software, a partir do nível de desacoplamento da sua arquitetura, ou seja, quão bem o sistema pode ser decomposto em módulos menores e independentemente substituíveis.

A DL avalia a manutenibilidade do software a partir do nível de desacoplamento da sua arquitetura, ou seja, quão bem o sistema pode ser decomposto em módulos menores e independentemente substituíveis. A DL considera módulos em todas as camadas de uma DSM, classificando essas camadas a partir de um algoritmo de clusterização denominado *Design Rule Hierarchy* (DRH). O DRH leva o tamanho do módulo em consideração e identifica as dependências diretas e transitivas.

Por exemplo, se um sistema tem apenas uma camada com 100 arquivos e formam 25 módulos com 4 arquivos cada um, seu DL é 100%, significando que um módulo pode melhorar, sem impactar os demais módulos e melhorar a qualidade do sistema como um todo. À medida que o tamanho de cada módulo cresce, torna-se mais difícil para eles evoluírem. Se cada módulo tem 25 arquivos, então seu DL diminui para 50%, e depois para 41% se cada módulo tem 50 arquivos. Se os 100 arquivos formam apenas um módulo, então ele tem um DL de apenas 35%.

Assim, valores mais altos da métrica DL são considerados resultados melhores, pois indicam que uma arquitetura é mais modular e, portanto, mais fácil de realizar manutenções. Além disso, resultados da DL extraídos de versões consecutivas não refatoradas são muito estáveis e a variação não trivial da métrica indica uma maior deterioração ou uma melhoria da arquitetura.

A métrica IAEv também pretende evidenciar a deterioração arquitetural, contudo, utiliza somente parâmetros relacionados ao acoplamento evolutivo para medir diretamente o impacto na degradação da arquitetura de software. Assim como observado no comportamento da DL, a métrica IAEv também mantém valores estáveis para versões consecutivas não refatoradas e a variação não trivial da métrica também indica alteração no nível de deterioração da arquitetura de software.

6.5 DISCUSSÃO

Os estudos sumarizados na Tabela 6.1 referem-se, respectivamente, aos trabalhos relacionados. Todos esses trabalhos avaliam as dependências lógicas ou acoplamento evolutivo com propósitos diversos. As principais vantagens, condições e limitações deste estudo em comparação aos trabalhos relacionados serão discutidas nesta seção.

TAB. 6.1: Comparação dos Trabalhos Relacionados.

Abordagem	**Métrica**	**Tipo**	**Objeto**	**Dep. Estrut.**	**Dep. Transit.**	**Nível**
6.1. Comparação Dep. Estruturais e Lógicas	-	-	Par	Sim	Não	Classe
6.2. Melhoria Detecção Acoplamento Evolutivo	CC	Indireta	Par	Não	Não	Método
6.3. Quantificação da Dívida Arquitetural	HCP	Indireta	Par	Não	Sim	Classe
6.4. Monitoramento Qualidade Arquitetural	DL	Indireta	Par	Não	Sim	Classe
Avaliação Deterioração Arquitetural	IAEv	Direta	Único	Não	Não	Classe

Ajienka e Capiluppi (2017) comparam as dependências estruturais às dependências lógicas para correlacionar o impacto de um tipo de dependência em relação ao outro no processo de correção de defeitos. Ainda com foco na correção de defeitos, Mondal et al. (2014) apresentam uma forma para melhorar a detecção do acoplamento evolutivo. Voltados à quantificação da dívida arquitetural, Xiao et al. (2016) apresentam a abordagem HCP. E Cai e Kazman (2016) monitoram a qualidade da arquitetura de software empregando a métrica DL, que avalia o nível de manutenibilidade da arquitetura baseado em seu nível de desacoplamento.

Com excessão do estudo de Ajienka e Capiluppi (2017), os outros estudos utilizam métricas para a avaliação da arquitetura de software usando informações acerca do acoplamento evolutivo. Porém, somente a métrica IAEv avalia diretamente o impacto da deterioração arquitetural para cada elemento da arquitetura de software.

As abordagens correlatas a este estudo e que quantificam o acoplamento evolutivo, normalmente, consideram o par de objetos para apresentação dos resultados. Assim, os valores estabelecidos são sempre referentes a um objeto em relação ao outro. Este estudo apresentou um indicador único para cada elemento arquitetural, que indica a probabilidade desse elemento estar comprometido com a deterioração causada pelo acoplamento evolutivo. Um alto índice da métrica IAEv indica que a manutenibilidade do elemento arquitetural foi prejudicada ao longo do processo de evolução do software.

Apenas o estudo de Ajienka e Capiluppi (2017) considera as dependências estruturais para obtenção dos resultados. Todos os outros trabalhos correlatos, incluindo este estudo, apenas identificam tais dependências para desconsiderar os pares de elementos do computo do acoplamento evolutivo. Uma desvantagem deste trabalho em relação aos estudos apresentados por Xiao et al. (2016) e Cai e Kazman (2016) é que as dependências

transitivas (ou indiretas) não foram consideradas.

A maioria dos trabalhos que envolvem a identificação do acoplamento evolutivo, incluindo os trabalhos correlatos, avaliam o software em nível de classe (ou entidade, ou arquivo) para estabelecer as relações lógicas entre eles. Dentre as métricas comparadas na Tabela 6.1, apenas a *Change Correspondence* avalia o código-fonte em nível de método, combinando a ideia de localização em uma base de código para determinar se as mudanças nas entidades alteradas são correspondentes. O impacto de performance de analisar sintaticamente todo o código fonte deve ser levado em consideração.

Neste estudo foram empregados os critérios de classificação das mudanças conjuntas definidos por Kirbas et al. (2017), tendo a métrica proposta se mantido aderente a outros critérios estabelecidos, conforme ilustra a Tabela 6.2.

TAB. 6.2: Critérios estabelecidos por Kirbas et al. (2017) para métricas de acoplamento evolutivo e a aderência da métrica IAEv.

Critério	**IAEv**
C1. Granularidade da Entidade	Aderente
C2. Abordagem de Agrupamento	Aderente
C3. Características de Localidade	Não aderente
C4. Características de Mudança	Aderente
C5. Tipo de Mudança	Aderente
C6. Autores Distintos	Aderente
C7. Característica Temporal - Regularidade	Não aderente
C8. Característica Temporal - Recente	Aderente
C9. Tamanho da Mudança	Aderente
C10. Normalização	Aderente
C11. Abordagem de Agrupamento - Temporal	Aderente
C12. Grandes Transações de Merge	Aderente
C13. Branchs	Aderente
C14. Agregação de Bases de Medição	Aderente
C15. Único Elemento Modificado	Aderente
C16. Refatoração de Elementos	Aderente
C17. Adequação do Histórico de Versões	Aderente
C18. Impacto Relativo	Não aderente
C19. Validação Matemática	Não aderente

De acordo com a Tabela 6.2, a métrica proposta neste estudo está aderente a 15 de 19 critérios propostos por Kirbas et al. (2017). A granularidade dos elementos (C1) avaliados pela métrica IAEv está em nível de arquivos ou classes. Os pares de elementos foram agrupados (C2) com base em transações e tarefas. As características de localidade (C3) não impactam no resultado da métrica IAEv. A métrica proposta considera as características

de mudança (C4), permitindo a configuração do escopo de avaliação (correção de defeitos e aprimoramento do software). Os tipos de mudança (C5) podem ser configurados para obtenção dos resultados da IAEv, sendo possível selecionar entre adição, modificação e exclusão.

A avaliação da IAEv considera o número de autores distintos (C6) e as características temporais (C8), avaliando mudanças recentes (que ocorrem no mesmo dia, pelo mesmo autor em atendimento à mesma tarefa). A regularidade das mudanças (C7) não foram avaliadas pela métrica proposta.

O tamanho da mudança (C9) é avaliado pela métrica IAEv considerando o número de elementos modificados durante a revisão de código. Os resultados da métrica proposta estão normalizados (C10) e consideram o agrupamento temporal (C11) de acordo com o início e o fim de cada *branch* sequencial avaliado (C13), conforme apresentado na Tabela 4.1. As grandes transações de merge (C12) entre *branchs* foram descartadas pela métrica IAEv.

Múltiplas solicitações de mudança ou múltiplas transações são avaliadas para uma mesma tarefa, desse modo a métrica IAEv permite agregação de base de medição (C14). A métrica proposta ainda permite que transações com um único elemento modificado (C15) sejam consideradas, pois podem ser combinadas com outras que atendem à mesma tarefa.

Os resultados da métrica proposta podem indicar quais elementos arquiteturais precisam ser refatorados (C16) e, ainda, indicam o impacto de uma refatoração nos elementos analisados. O histórico de versões deve possuir atividades suficientes para representar os acoplamentos (C17). Entretanto, nem todas as transações são consideradas pela métrica IAEv, por esse motivo o impacto relativo (C18) não é avaliado. Finalmente a métrica proposta não apresentou validação matemática (C19), pois os resultados foram validados a partir da aplicação em sistema de estudo de caso.

7 CONCLUSÃO

O objetivo principal deste trabalho é a definição de uma métrica direta para avaliação quantitativa da deterioração arquitetural, utilizando informações acerca das características das mudanças conjuntas entre pares de elementos para determinar o acoplamento evolutivo. Para tanto, este trabalho propõe, ainda, uma metodologia de classificação das alterações conjuntas que ocorrem no nível arquitetural de um sistema e detalha uma nova abordagem para a detecção do acoplamento evolutivo, utilizando o conceito da verificação deslizante.

Os resultados deste estudo foram obtidos durante a aplicação da metodologia proposta em um grande cenário de evolução de um Sistema Militar de Comando e Controle desenvolvido pela MB em atendimento ao Ministério da Defesa. A partir do estudo de caso que empregou o SIPLOM 4 como prova de conceito, foi possível identificar instâncias de acoplamento evolutivo a partir do conceito da verificação deslizante e da classificação de tarefas. Com isso, informações desprezadas pelos métodos atuais foram utilizadas para ampliar o escopo da detecção do acoplamento evolutivo. Quanto à métrica apresentada neste estudo - Impacto do Acoplamento Evolutivo (IAEv) - os resultados indicaram que a medida foi sensível à refatoração de código, apresentando variação indicativa sobre as melhorias nos elementos arquiteturais envolvidos na evolução do software. Da mesma forma, para um elemento que comprovadamente adquiriu acoplamento evolutivo, a métrica IAEv evidenciou a deterioração arquitetural no momento em que a conexões lógicas foram introduzidas na arquitetura.

O conceito da verificação deslizante se baseia na ideia de que as dependências lógicas adquiridas por elementos arquiteturais, ao longo do processo de evolução do software, não devem ser simplesmente avaliadas em uma única versão de um sistema, mas sim em uma série contínua de versões. A aplicação do método SHCP tornou o processo de identificação do acoplamento evolutivo estável entre versões consecutivas e minimizou a alta instabilidade nas revisões iniciais de um novo recurso. O método SHCP ainda se mostrou útil na determinação das mudanças conjuntas que, de fato, indicam dependência lógica adquirida, minimizando falsos positivos.

A classificação das mudanças conjuntas permitiu o entendimento da natureza das instâncias do acoplamento evolutivo e o seu impacto na arquitetura do software. Mo et al. (2015) observam que, em muitos casos, módulos possuem dependências lógicas nocivas e

estas dependências devem ser removidas. Foi constatado que o acoplamento evolutivo leva à deterioração arquitetural ou, em última análise, é um tipo de deterioração arquitetural. Este estudo, portanto, pode auxiliar na avaliação sobre a viabilidade de empregar a equipe de desenvolvimento para reverter a deterioração arquitetural causada pelo acoplamento evolutivo.

Foram estabelecidos critérios para medir a deterioração arquitetural diretamente, apenas com parâmetros acerca do acoplamento evolutivo, de modo a dispensar o uso de modelos adicionais ou padrões de defeitos para inferir sobre a deterioração arquitetural causada pelo acoplamento evolutivo. A partir da métrica IAEv, o peso de cada ocorrência de acoplamento evolutivo foi ajustado de modo a atualizar a maneira pela qual métricas anteriores mediam seu impacto na arquitetura de software. A substituição de medições derivadas por uma métrica direta tornou a análise da deterioração arquitetural causada pelo acoplamento evolutivo mais intuitiva.

Cai e Kazman (2016) afirmam que o processo de refatoração é caro, consome o cronograma do projeto e não produz novos recursos. Os custos de refatoração são imediatos e concretos, ao passo que os benefícios são vagos e de longo prazo. E, por conseguinte, os projetos raramente investem em refatoração.

Desse modo, mesmo se uma área do sistema apresentar baixos índices de manutenção, a partir da metodologia apresentada neste estudo, podem ser identificadas instâncias de acoplamento evolutivo que comprometem os atributos de qualidade estabelecidos para a arquitetura do software. Desse modo, o impacto dessa relação lógica pode ser quantificado para destacar elementos potencialmente comprometidos pelo acoplamento evolutivo. As dependências lógicas podem acarretar futuros esforços de desenvolvimento e, em última instância, degradarem o software a tal ponto que seja impraticável a refatoração no longo prazo. Por essa razão, detectar o acoplamento evolutivo com maior precisão e de forma contínua poderá auxiliará na redução dos custos de manutenção do software.

Por exemplo, quando um Gerente de Projetos delegar uma determinada tarefa de correção de defeito ou implementação de nova funcionalidade, será possível identificar o impacto desta decisão, imediatamente após a mudança ser adicionada ao SCV. A identificação das áreas na arquitetura do software que passaram a incidir o acoplamento evolutivo pode sugerir a refatoração, antes que outras mudanças ocorram e inviabilizem as ações de manutenção.

7.1 CONTRIBUIÇÕES

As seguintes contribuições foram alcançadas:

(i) A detecção do acoplamento evolutivo foi aprimorada, considerando uma série contínua de versões, de acordo com a abordagem da verificação deslizante. Isso permitiu uma melhor compreensão desse tipo de deterioração arquitetural ao longo da evolução do software;

(ii) O escopo de detecção do acoplamento evolutivo foi ampliado a partir da classificação das mudanças simultâneas entre elementos arquiteturais não conectados estruturalmente, considerando as características de tarefas. Assim, mesmo que não sejam observadas mudanças simultâneas para um par de elementos em uma mesma transação, eles podem ser considerados conectados logicamente se existirem características correspondentes. Com isso, informações desprezadas pelos métodos passaram a ser consideradas.

(iii) Foi detalhada uma nova métrica que quantifica diretamente o impacto do acoplamento evolutivo e auxilia na identificação da deterioração arquitetural ao longo do processo de evolução dos sistemas de software. Essa nova medida identifica elementos arquiteturais comprometidos pelo acoplamento evolutivo e garante o monitoramento contínuo da deterioração de cada elemento ao longo da evolução do software.

7.2 TRABALHOS FUTUROS

Trabalhos futuros podem aplicar a metodologia apresentada em softwares com viés distinto ao Militar, incluindo softwares de código aberto e características distintas àquelas apresentadas neste texto, garantindo assim a generalização dos resultados.

A métrica IAEv mostrou-se adequada para identificar instâncias do acoplamento evolutivo. Portanto, a substituição da métrica *Instability* no método da verificação deslizante tem potencial para ser explorada. A métrica IAEv pode determinar de maneira mais confiável quais elementos da arquitetura do software se apresentam deteriorados devido à incidência do acoplamento evolutivo.

O cálculo de transitividade das dependências lógicas não foi considerado ao longo deste estudo. Assim, trabalhos futuros poderão melhorar a determinação do acoplamento evolutivo, considerando a transitividade entre os elementos arquiteturais.

O desenvolvimento de uma ferramenta, baseada no protótipo desenvolvido para obtenção dos dados analisados ao longo deste estudo, também poderá ser trabalhada futuramente.

Outra oportunidade que pode ser explorada refere-se à combinação da métrica proposta com a técnica da verificação deslizante, onde somente as instâncias de acoplamento evolutivo mantidas pelo método SHCP seriam consideradas no cômputo da métrica.

Este estudo considerou a data da transação no SCV para avaliar as mudanças conjuntas temporalmente. Contudo, estudos futuros poderão considerar a data de início e a data de conclusão da tarefa, obtidas a partir da integração com o SGP, para melhor avaliar temporalmente as mudanças conjuntas.

Outra vertente com potencial para se beneficiar com a continuidade desta pesquisa é a utilização da metodologia apresentada no desenvolvimento de técnicas de recomendação automática para identificação de tendências de problemas estruturais na arquitetura de software. Assim, gerentes de projeto e analistas poderiam ser apoiados com informações necessárias à tomada de decisão, baseadas na análise automática do impacto da deterioração causada pelo acoplamento evolutivo ao longo do processo de evolução do software.

8 REFERÊNCIAS BIBLIOGRÁFICAS

ABOWD, G.; BASS, L.; CLEMENTS, P.; KAZMAN, R. ; NORTHROP, L. **Recommended Best Industrial Practice for Software Architecture Evaluation.**. [S.l.: s.n.], 1997. (Relatório Técnico).

ABRAN, A. **Software metrics and software metrology**. [S.l.]: John Wiley & Sons, 2010.

AJIENKA, N.; CAPILUPPI, A. Understanding the interplay between the logical and structural coupling of software classes. **Journal of Systems and Software**, v. 134, p. 120–137, 2017.

BALDWIN, C. Y.; CLARK, K. B. **Design rules: The power of modularity**. [S.l.]: MIT press, 2000.

BALL, T.; KIM, J.-M.; PORTER, A. A. ; SIY, H. P. If your version control system could talk. In: ICSE WORKSHOP ON PROCESS MODELLING AND EMPIRICAL STUDIES OF SOFTWARE ENGINEERING, 1., 1997. **Anais**... [S.l.: s.n.], 1997, p. 1–5.

BASILI, V. R.; NAKAMURA, T. Metrics of software architecture changes based on structural distance. In: SOFTWARE METRICS, 2005. 11TH IEEE INTERNATIONAL SYMPOSIUM, 1., 2005. **Anais**... [S.l.: s.n.], 2005, p. 24–24.

BASILI, V. R.; PERRICONE, B. T. Software errors and complexity: an empirical investigation0. **Communications of the ACM**, v. 27, n. 1, p. 42–52, 1984.

BASS, L.; CLEMENTS, P. ; GARLAN, D. **Software architecture in practice, 2nd edition**. [S.l.]: Pearson Education India, 2002.

BRIAND, L. C.; MORASCA, S. ; BASILI, V. R. Measuring and assessing maintainability at the end of high level design. In: SOFTWARE MAINTENANCE, 1993. CSM-93, PROCEEDINGS., CONFERENCE ON, 1., 1993. **Anais**... [S.l.: s.n.], 1993, p. 88–87.

BRIAND, L. C.; MORASCA, S. ; BASILI, V. R. Property-based software engineering measurement. **IEEE transactions on software Engineering**, v. 22, n. 1, p. 68–86, 1996.

BUCKLEY, J.; MENS, T.; ZENGER, M.; RASHID, A. ; KNIESEL, G. Towards a taxonomy of software change. **Journal of Software Maintenance and Evolution: Research and Practice**, v. 17, n. 5, p. 309–332, 2005.

CAI, Y.; KAZMAN, R. Software architecture health monitor. In: PROCEEDINGS OF THE 1ST INTERNATIONAL WORKSHOP ON BRINGING ARCHITECTURAL DESIGN THINKING INTO DEVELOPERS' DAILY ACTIVITIES, 1., 2016. **Anais**... [S.l.: s.n.], 2016, p. 18–21.

CHAPIN, N.; HALE, J. E.; KHAN, K. M.; RAMIL, J. F. ; TAN, W.-G. Types of software evolution and software maintenance. **Journal of software maintenance and evolution: Research and Practice**, v. 13, n. 1, p. 3–30, 2001.

CHIDAMBER, S. R.; KEMERER, C. F. **Towards a metrics suite for object oriented design**. [S.l.]: ACM, 1991.

CLEMENTS, P.; GARLAN, D.; BASS, L.; STAFFORD, J.; NORD, R.; IVERS, J. ; LITTLE, R. **Documenting software architectures: views and beyond**. [S.l.]: Pearson Education, 2002.

EICK, S. G.; GRAVES, T. L.; KARR, A. F.; MARRON, J. S. ; MOCKUS, A. Does code decay? assessing the evidence from change management data. **IEEE Transactions on Software Engineering**, v. 27, n. 1, p. 1–12, 2001.

EPPINGER, S. D.; WHITNEY, D. E.; SMITH, R. P. ; GEBALA, D. A. A model-based method for organizing tasks in product development. **Research in engineering design**, v. 6, n. 1, p. 1–13, 1994.

ERNST, N. A.; OZKAYA, I.; BELLOMO, S.; NORD, R. L. ; GORTON, I. Measure it? manage it? ignore it? software practitioners and technical debt. In: PROCEEDINGS OF THE 2015 10TH JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, 1., 2015. **Anais**... [S.l.: s.n.], 2015, p. 50–60.

FENTON, N.; BIEMAN, J. **Software metrics: a rigorous and practical approach**. [S.l.]: CRC Press, 2014.

FOWLER, M.; BECK, K. **Refactoring: improving the design of existing code**. [S.l.]: Addison-Wesley Professional, 1999.

GALL, H.; HAJEK, K. ; JAZAYERI, M. Detection of logical coupling based on product release history. In: SOFTWARE MAINTENANCE, 1998. PROCEEDINGS., INTERNATIONAL CONFERENCE ON, 1., 1998. **Anais**... [S.l.: s.n.], 1998, p. 190–198.

GARLAN, D.; MONROE, R. T. ; WILE, D. Acme: Architectural description of component-based systems. **Foundations of component-based systems**, v. 68, p. 47–68, 2000.

GARLAN, D.; SHAW, M. An introduction to software architecture. In: _____. **Advances in software engineering and knowledge engineering**. [S.l.]: World Scientific, 1993. p. 1–39.

GODFREY, M. W.; GERMAN, D. M. The past, present, and future of software evolution. In: FRONTIERS OF SOFTWARE MAINTENANCE, 2008. FOSM 2008., 1., 2008. **Anais**... [S.l.: s.n.], 2008, p. 129–138.

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)**, v. 46, n. 2, p. 28, 2013.

KIRBAS, S.; HALL, T. ; SEN, A. Evolutionary coupling measurement: making sense of the current chaos. **Science of Computer Programming**, v. 135, p. 4–19, 2017.

KOUROSHFAR, E. Studying the effect of co-change dispersion on software quality. In: PROCEEDINGS OF THE 2013 INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1., 2013. **Anais**... [S.l.: s.n.], 2013, p. 1450–1452.

LAND, R. Software deterioration and maintainability–a model proposal. In: PROCEEDINGS OF SECOND CONFERENCE ON SOFTWARE ENGINEERING RESEARCH AND PRACTISE IN SWEDEN (SERPS), 1., 2002. **Anais**... [S.l.: s.n.], 2002, p. 1–7.

LE, D. M.; CARRILLO, C.; CAPILLA, R. ; MEDVIDOVIC, N. Relating architectural decay and sustainability of software systems. In: SOFTWARE ARCHITECTURE (WICSA), 2016 13TH WORKING IEEE/IFIP CONFERENCE ON, 1., 2016. **Anais**... [S.l.: s.n.], 2016, p. 178–181.

LI, B.; LIAO, L. ; SI, J. A technique to evaluate software evolution based on architecture metric. In: SOFTWARE ENGINEERING RESEARCH, MANAGEMENT AND APPLICATIONS (SERA), 2016 IEEE 14TH INTERNATIONAL CONFERENCE ON, 1., 2016. **Anais**... [S.l.: s.n.], 2016, p. 1–8.

LINDVALL, M.; TESORIERO, R. ; COSTA, P. Avoiding architectural degeneration: An evaluation process for software architecture. In: SOFTWARE METRICS, 2002. PROCEEDINGS. EIGHTH IEEE SYMPOSIUM ON, 1., 2002. **Anais**... [S.l.: s.n.], 2002, p. 77–86.

MACCORMACK, A. D.; LAGERSTRÖM, R.; BALDWIN, C. Y.; STURTEVANT, D. ; DOOLAN, L. Exploring the relationship between architecture coupling and software vulnerabilities: A google chrome case. **Harvard Business School**, v. 1, n. 1, p. 1–13, 2017.

MACHADO, M.; CHOREN, R. Improving the detection of evolutionary coupling: An approach considering sliding verification. In: PROCEEDINGS OF THE 33RD ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, SAC 2018, PAU, FRANCE, 1., 2018. **Anais**... [S.l.: s.n.], 2018, p. 1410–1416.

MARTIN, R. Oo design quality metrics. **An analysis of dependencies**, v. 12, n. 1, p. 151–170, 1994.

MARTINI, A.; BOSCH, J. The danger of architectural technical debt: Contagious debt and vicious circles. In: SOFTWARE ARCHITECTURE (WICSA), 2015 12TH WORKING IEEE/IFIP CONFERENCE ON, 1., 2015. **Anais**... [S.l.: s.n.], 2015, p. 1–10.

MEDVIDOVIC, N.; TAYLOR, R. N. Software architecture: foundations, theory, and practice. In: PROCEEDINGS OF THE 32ND ACM/IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING-VOLUME 2, 1., 2010. **Anais**... [S.l.: s.n.], 2010, p. 471–472.

MO, R.; CAI, Y.; KAZMAN, R. ; XIAO, L. Hotspot patterns: The formal definition and automatic detection of architecture smells. In: SOFTWARE ARCHITECTURE (WICSA), 2015 12TH WORKING IEEE/IFIP CONFERENCE ON, 1., 2015. **Anais**... [S.l.: s.n.], 2015, p. 51–60.

MO, R.; CAI, Y.; KAZMAN, R.; XIAO, L. ; FENG, Q. Decoupling level: a new metric for architectural maintenance complexity. In: PROCEEDINGS OF THE 38TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1., 2016. **Anais**... [S.l.: s.n.], 2016, p. 499–510.

MONDAL, M.; ROY, C. K. ; SCHNEIDER, K. A. Improving the detection accuracy of evolutionary coupling by measuring change correspondence. In: SOFTWARE MAINTENANCE, REENGINEERING AND REVERSE ENGINEERING (CSMR-WCRE), 2014 SOFTWARE EVOLUTION WEEK-IEEE CONFERENCE ON, 1., 2014. **Anais**... [S.l.: s.n.], 2014, p. 358–362.

MONDAL, M.; ROY, C. K. ; SCHNEIDER, K. A. Does cloned code increase maintenance effort?. In: 2017 IEEE 11TH INTERNATIONAL WORKSHOP ON SOFTWARE CLONES (IWSC), 1., 2017. **Anais**... [S.l.: s.n.], 2017, p. 1–7.

NORD, R. L.; OZKAYA, I.; KRUCHTEN, P. ; GONZALEZ-ROJAS, M. In search of a metric for managing architectural technical debt. In: SOFTWARE ARCHITECTURE (WICSA) AND EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE (ECSA), 2012 JOINT WORKING IEEE/IFIP CONFERENCE ON, 1., 2012. **Anais**... [S.l.: s.n.], 2012, p. 91–100.

PARNAS, D. L. On the criteria to be used in decomposing systems into modules. **Communications of the ACM**, v. 15, n. 12, p. 1053–1058, 1972.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. **ACM SIGSOFT Software Engineering Notes**, v. 17, n. 4, p. 40–52, 1992.

PMD. PMD Source Code Analyzer. Disponível em: <`http://pmd.github.io/`>. Acesso em: [Online; accessed March 01, 2018].

RAJLICH, V. Software evolution and maintenance. In: PROCEEDINGS OF THE ON FUTURE OF SOFTWARE ENGINEERING, 1., 2014. **Anais**... [S.l.: s.n.], 2014, p. 133–144.

REDMINE. Overview Redmine. Disponível em: <`https://www.redmine.org/`>. Acesso em: [Online; accessed March 01, 2018].

RIAZ, M.; SULAYMAN, M. ; NAQVI, H. Architectural decay during continuous software evolution and impact of 'design for change'on software architecture. In: INTERNATIONAL CONFERENCE ON ADVANCED SOFTWARE ENGINEERING AND ITS APPLICATIONS, 1., 2009. **Anais**... [S.l.: s.n.], 2009, p. 119–126.

ROY, B.; GRAHAM, T. N. Methods for evaluating software architecture: A survey. **Computing**, v. 545, n. 2008-545, p. 82, 2008.

STEVENS, W. P.; MYERS, G. J. ; CONSTANTINE, L. L. Structured design. **IBM Systems Journal**, v. 13, n. 2, p. 115–139, 1974.

STEWARD, D. V. The design structure system: A method for managing the design of complex systems. **IEEE transactions on Engineering Management**, v. 3, p. 71–74, 1981.

SVN. Apache Subversion. Disponível em: <https://subversion.apache.org/>. Acesso em: [Online; accessed March 01, 2018].

SWANSON, E. B. The dimensions of maintenance. In: PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1., 1976. **Anais**... [S.l.: s.n.], 1976, p. 492–497.

TANTITHAMTHAVORN, C.; IHARA, A. ; MATSUMOTO, K.-I. Using co-change histories to improve bug localization performance. In: SOFTWARE ENGINEERING, ARTIFICIAL INTELLIGENCE, NETWORKING AND PARALLEL/DISTRIBUTED COMPUTING (SNPD), 2013 14TH ACIS INTERNATIONAL CONFERENCE ON, 1., 2013. **Anais**... [S.l.: s.n.], 2013, p. 543–548.

TVEDT, R. T.; COSTA, P. ; LINDVALL, M. Evaluating software architectures. **Advances in Computers**, v. 61, p. 1–43, 2004.

TVEDT, R. T.; LINDVALL, M. ; COSTA, P. A process for software architecture evaluation using metrics. In: SOFTWARE ENGINEERING WORKSHOP, 2002. PROCEEDINGS. 27TH ANNUAL NASA GODDARD/IEEE, 1., 2002. **Anais**... [S.l.: s.n.], 2002, p. 191–196.

WOHED, P.; VAN DER AALST, W. M.; DUMAS, M.; TER HOFSTEDE, A. H. ; RUSSELL, N. On the suitability of bpmn for business process modelling. In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT, 1., 2006. **Anais**... [S.l.: s.n.], 2006, p. 161–176.

XIAO, L.; CAI, Y. ; KAZMAN, R. Design rule spaces: A new form of architecture insight. In: PROCEEDINGS OF THE 36TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1., 2014. **Anais**... [S.l.: s.n.], 2014, p. 967–977.

XIAO, L.; CAI, Y. ; KAZMAN, R. Titan: A toolset that connects software architecture with quality analysis. In: PROCEEDINGS OF THE 22ND ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, 1., 2014. **Anais**... [S.l.: s.n.], 2014, p. 763–766.

XIAO, L.; CAI, Y.; KAZMAN, R.; MO, R. ; FENG, Q. Identifying and quantifying architectural debt. In: PROCEEDINGS OF THE 38TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1., 2016. **Anais**... [S.l.: s.n.], 2016, p. 488–498.

ZIMMERMANN, T.; ZELLER, A.; WEISSGERBER, P. ; DIEHL, S. Mining version histories to guide software changes. **IEEE Transactions on Software Engineering**, v. 31, n. 6, p. 429–445, 2005.

ZUSE, H. **A framework of software measurement**. [S.l.]: Walter de gruyter, 1998.