Informações contextuais do desenvolvimento de software na predição de propagação de mudanças

Igor Scaliante Wiese, Reginaldo Ré, Igor Steinmacher
Departamento de Computação
Universidade Tecnológica Federal do Paraná (UTFPR) - Brasil
{igor, reginaldo.re, igorfs}@utfpr.edu.br

Rodrigo Takashi Kuroda
PPGI / PPGI - UTFPR/CP
Universidade Tecnológica Federal do Paraná (UTFPR) - Brasil
rodrigokuroda@gmail.com

Gustavo Ansaldi Oliva, Marco Aurélio Gerosa
Departamento de Ciência da Computação
Universidade de São Paulo (USP)
São Paulo, SP, Brasil
{goliva, gerosa}@ime.usp.br

Resumo — Propagação de mudança ocorre quando uma mudança em um artefato leva a alterações em outros artefatos. Pesquisas anteriores têm utilizado análise de frequência de mudanças entre os artefatos, análise estática, dinâmica e textual como base para técnicas de análise da propagação de mudanças. No entanto, ainda é necessário melhorar a acurácia desses modelos. Nossa estratégia consiste em combinar diferentes informações contextuais relacionadas às características das tarefas, das comunicações dos desenvolvedores e das modificações dos artefatos. Este trabalho, portanto, propõe o uso de informações contextuais para prever a propagação de mudanças, baseado na ideia de que essas informações capturam aspectos particulares de quando mudanças conjuntas ocorrem. Conduzimos um estudo empírico em 4 projetos de software livre, Cassandra, Camel, Hadoop e Lucene. Foram construídos classificadores para cada par de artefatos que mudam frequentemente. Os modelos obtidos têm valores de Área sob a curva (AUC) de 0.849 em média. Além disso, a sensibilidade (*recall*) obtida é quase 4 vezes maior (57.06% vs. 15.70%) quando comparamos nossos modelos com um modelo de regras de associação. Com a melhoria na sensibilidade, os modelos poderiam ser utilizados na prática para ajudar desenvolvedores em tarefas de manutenção e evolução do software.

Palavras chave — propagação de mudança, acoplamento de mudança, mudanças conjuntas, modelos de predição, informações contextuais

I. INTRODUÇÃO

Em um projeto de software, uma mudança em um artefato muitas vezes resulta em mudanças em outros artefatos por diferentes razões (acoplamento estrutural, lógico, semântico, etc.) [1]. Nesse contexto, abordagens de propagação mudança são utilizadas para predizer quais artefatos mudam conjuntamente.

A questão central das abordagens de propagação de mudança consiste em responder à pergunta: "quando uma mudança em um artefato propaga mudanças para outros artefatos?". Suponha que um desenvolvedor(a) esteja realizando uma tarefa, por exemplo, corrigindo um defeito no software. Esse desenvolvedor(a) modificou um artefato mas não sabe quais outros artefatos deveriam ser mudados conjuntamente. Nesse cenário, uma mudança incompleta pode levar o desenvolvedor a introduzir um defeito ou até mesmo reabrir a tarefa no futuro, adicionando custos de manutenção [1], [2].

Em projetos de software livre, percebemos esse cenário acontecendo frequentemente. Desenvolvedores modificam artefatos cotidianamente durante a evolução de software [3]. Além disso, sabe-se que novatos têm dificuldade de encontrar quais artefatos devem modificar durante a realização de uma tarefa [4]. Nesse sentido, a análise de propagação de mudanças pode auxiliar os desenvolvedores durante a realização de suas tarefas [5], [6].

Pesquisas anteriores propuseram diferentes abordagens para prever a propagação de mudanças. Dentre elas, destacam-se as abordagens baseadas em mineração de repositórios de software aplicando abordagens para identificação de propagação de mudanças e análise conceitual de artefatos [7]–[10], análise estática e dinâmica do código fonte [11], [12] e abordagens híbridas [6], [13]. O objetivo dessas abordagens consiste em determinar a propagação de mudança identificando artefatos que são mais propensos a mudar conjuntamente utilizando regras de associação [14]–[16].

Apesar do esforço dos pesquisadores, ainda existe muito trabalho a ser feito para melhorar a acurácia das abordagens de propagação de mudança. Uma das alternativas para melhor a acurácia apoia-se no uso de novas fontes de informações e métricas para capturar aspectos que não são considerados nas abordagens atuais [17]. Assim, o objetivo desse trabalho consiste em investigar o uso de informações contextuais para a predição de propagação de mudanças. Nossa motivação para o uso de informações contextuais diz respeito a sua capacidade de descrever o contexto que cada mudança acontece, uma vez que mudanças podem acontecer por diferentes motivos ao longo da evolução do software.

Nesse trabalho, consideramos três tipos de informações contextuais: a solicitação da mudança, que diz respeito ao relato da tarefa (quem reportou a tarefa, quem corrigiu a tarefa e o tipo da tarefa); a comunicação dos desenvolvedores nas tarefas em que os artefatos foram modificados (número de comentários, quantidade de palavras, número de desenvolvedores que comentaram a tarefa); e os meta-dados da própria modificação dos artefatos (linhas adicionadas/ modificadas, linhas excluídas, autor do *commit*).

Construímos modelos de predição para prever a propagação de mudanças nos acoplamentos de mudanças fortes. Queremos prever se uma mudança conjunta de um acoplamento forte volta a acontecer nas versões subsequentes e se nossos modelos conseguem prever quando uma mudança conjunta acontece.

Nossa opção por estudar os acoplamentos de mudanças fortes é baseada no impacto que esse fenômeno tem na evolução e manutenção do software e por envolverem artefatos que mudam juntos mais frequentemente. Pesquisadores têm estudado os efeitos da ocorrência de acoplamento fortes, como por exemplo, a associação com o número de defeitos e a importância no projeto [17]–[19].

Desta forma, investigamos a seguinte questão central: *"Qual o impacto das informações contextuais das solicitações de mudança, da comunicação dos desenvolvedores e das mudanças dos artefatos na acurácia da propagação de mudança nos artefatos que mudam juntos frequentemente?"*

Para esse propósito, construímos classificadores e avaliamos o impacto do uso das informações contextuais em termos de precisão e sensibilidade. Para essa tarefa, quatro projetos da *Apache Software Foundation* (Cassandra, Camel, Lucene e Hadoop) foram utilizados para investigar as seguintes questões de pesquisa (QP):

(QP1) É possível prever a propagação de mudanças entre dois artefatos que mudam frequentemente usando informações contextuais das solicitações de mudança, da comunicação dos desenvolvedores e das modificações dos artefatos? Sim, nossos classificadores predizem a propagação de mudança de acoplamentos de mudança fortes com a média de 0.849 de AUC. Trabalhos anteriores [10] reportam valores de precisão entre 70% e 90%, enquanto a sensibilidade (*recall*) frequentemente encontra-se entre 25% e 10%. Nosso modelo, na média, obteve valor de sensibilidade de 57.06% e precisão de 72.07%. Comparando os resultados com o modelo de regras de associação, os resultados indicam melhoria de quase 4 vezes nos valores de sensibilidade já que as regras de associação obtiveram 15.70% de sensibilidade.

(QP2) Qual é a dimensão contextual mais relevante para a predição de propagação de mudanças? A comparação dos modelos de cada dimensão mostrou que a dimensão contextual da solicitação de mudança apresentou diferença estatística em alguns projetos. Entretanto, os melhores desempenhos dos modelos foram obtidos combinando duas dimensões quaisquer. A adição de uma terceira dimensão não apresentou diferença estatística no desempenho dos modelos de predição da propagação de mudança nos quatro projetos avaliados.

Nesse estudo, descobrimos que com o uso de informações contextuais obtidas das solicitações de mudança, comunicação dos desenvolvedores e modificações dos artefatos é possível melhorar a predição da propagação de mudança. Estes resultados sugerem que é possível reduzir o número de mensagens e alertas de propagação de mudanças errados para os desenvolvedores, tornando a nossa abordagem mais propícia para adoção em um cenário real.

O artigo é organizado da seguinte forma. Na Seção II apresentamos o conceito de acoplamento de mudança, foco deste estudo. A Seção III descreve o método de pesquisa, enquanto a Seção IV apresenta os resultados com respeito às duas questões de pesquisa. A Seção V discute ameaças à validade do nosso estudo. A Seção VI apresenta os trabalhos relacionados. Finalmente, a Seção VII apresenta a conclusão e trabalhos futuros.

II. ACOPLAMENTO DE MUDANÇA

Alguns artefatos tendem a serem alterados conjuntamente ao longo do desenvolvimento do software. O conceito de acoplamento de mudança captura o relacionamento implícito entre esses artefatos. Alguns benefícios da análise de acoplamento de mudanças foram discutidos por D'Ambros et al. [20]. Por exemplo, acoplamentos de mudança podem revelar relacionamentos que não estão presentes no código ou documentação. Outros pesquisadores mostraram que acoplamentos de mudança afetam a qualidade de software [17]–[19].

Para esse trabalho, o conceito de acoplamento de mudança pode ser aplicado a partir do seguinte cenário. Suponha que um desenvolvedor de software fez uma mudança em uma determinada parte de um sistema, uma classe, por exemplo. O que mais esse desenvolvedor tem que mudar para evitar uma mudança incompleta? Com base na ideia de que os artefatos que mudaram juntos no passado são propensos a mudarem juntos no futuro, pesquisadores utilizaram acoplamentos de mudança para desenvolverem mecanismos de predição de propagação de mudanças [7], [9], [21]. Com base nesse cenário, queremos utilizar as informações contextuais para predizer a propagação de mudança em versões consecutivas entre dois artefatos que possuem acoplamento forte de mudança.

III. MÉTODO

Nessa seção, apresentamos nossa justificativa para selecionar essas questões de pesquisa. Também descrevemos a extração de dados, o procedimento para identificar acoplamentos de mudança fortes e a abordagem de criação dos modelos de predição e análise dos resultados.

A. Justificativa das Questões de Pesquisa

(QP1) É possível prever a propagação de mudanças entre dois artefatos que mudam frequentemente usando informações contextuais das solicitações de mudança, da comunicação dos desenvolvedores e das modificações dos artefatos? Pesquisas anteriores mostraram que os classificadores podem prever a propagação mudança. No entanto, nossa hipótese é que o uso de informações contextuais melhoram o desempenho desses modelos dado que as mudanças acontecem em diferentes circunstâncias. Se obtivermos modelos com maior sensibilidade, podemos melhorar a acurácia evitando alertas enganosos para os desenvolvedores, especialmente porque conseguimos identificar quando a propagação de mudança acontece ou não.

(QP2) Qual é a dimensão contextual mais importante para a predição de propagação de mudanças? Saber quais informações contextuais são indicadores influentes de propagação de mudança pode ajudar gerentes de projetos e desenvolvedores durante a realização das solicitações de mudanças. Além disso, saber sobre a quantidade de dimensões necessárias para a construção dos modelos é importante no dimensionamento do esforço para a construção dos modelos de predição de propagações de mudança.

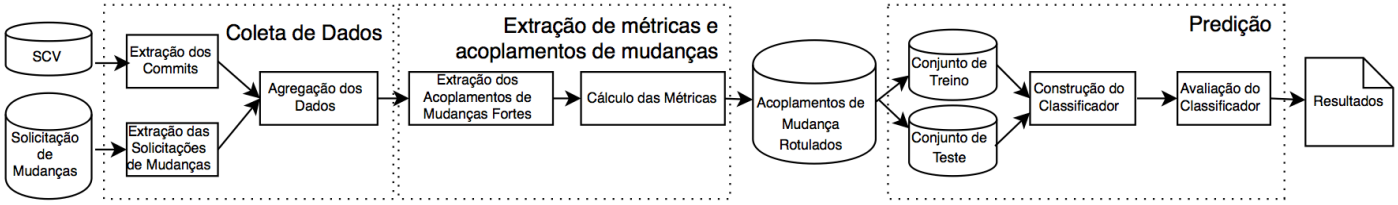

Fig. 1 Visão geral do método utilizado para Coleta de dados; extração de métricas e acoplamentos de mudança; e predição

B. *Projetos Estudados*

A fim de responder as nossas questões de pesquisa, estudamos 4 projetos hospedados na *Apache Software Foundation*. Escolheu-se projetos de domínios diferentes, para aumentar a generalização do estudo. A Tabela I apresenta os projetos, versões, domínio e número de acoplamento de mudanças fortes em cada versão.

TABELA I. PROJETOS ANALISADOS

Projetos Apache	**Versões**	**Domínio**	**Número de acoplamentos de mudanças fortes**
Cassandra	1.0, 1.1, 1.2	Banco de Dados	1.0 (11) , 1.1 (22), 1.2 (20)
Camel	2.1	Framework de integração	6
Hadoop	2.0	Framework para processamento Distribuído	19
Lucene	4.0	Framework para Indexação e Consulta	9

Para determinar os acoplamentos de mudança forte, nós calculamos os valores de suporte e confiança, conforme descrição na Seção III.D. Uma das tarefas centrais no paper é prever se um acoplamento forte identificado em uma versão resultará ou não em uma mudança conjunta nas versões subsequentes. No total foram estudados 87 acoplamentos de mudanças fortes, identificados em 6 diferentes versões.

É importante mencionar que alguns desses acoplamentos propagaram mudanças por mais de uma versão. Por exemplo, o arquivo *"StorageService.java"* foi frequentemente acoplado com o arquivo *"Gossiper.java"* na versão 1.0 do projeto Cassandra. Foram preditas as mudanças conjuntas desses dois artefatos durante as versões 1.0, 1.1 e 1.2. Não necessariamente o acoplamento entre *"StorageService.java" e "Gossiper.java"* apareceu na lista de acoplamentos de mudança fortes nas versões consecutivas. Nesse sentido, a quantidade de modelos construídos é muito maior do que o número de acoplamentos de mudanças fortes listados na Tabela I. Por exemplo, na versão 1.0 do projeto Cassandra encontramos 11 acoplamentos fortes, mas foram construídos 29 classificadores.

A Fig. 1 apresenta a visão geral do método utilizado nesse trabalho. Nas próximas seções descrevemos os passos de coleta de dados, extração dos acoplamentos de mudança fortes e das métricas de contexto e construção dos modelos de predição.

C. *Coleta de Dados*

Duas fontes de dados foram utilizadas para a coleta de dados: Sistema de Controle de Versão (SCV) e Gerenciadores de Tarefas (*issue tracker system - ITS*).

Uma solicitação de mudanças (*issue*) é uma tarefa que compreende a correção de um defeito ou a implementação de um novo requisito. Entretanto, muitas transações (*commits*) podem ser necessárias para concluir uma solicitação de mudança. As informações de contexto coletadas sobre as mudanças dos artefatos foram coletadas dos *commits* armazenados no SCV. Nos projetos hospedados na *Apache Software Foundation*, o ITS JIRA é utilizado para armazenar as solicitações de mudanças e permitir a comunicação dos desenvolvedores em uma solicitação de mudança. Desse ambiente, portanto, foram coletadas informações do contexto da solicitação de mudança e comunicação entre os desenvolvedores.

Como as duas fontes de dados não são relacionadas, precisamos usar o identificador único de cada solicitação de mudança para encontrar transações realizadas para conclusão da solicitação de mudança. Abaixo, descrevemos brevemente as etapas da coleta de dados e como esse procedimento é realizado.

Extração dos *Commits* e Solicitações de Mudanças. Depois de obter dados do SCV de cada projeto armazenado, coletamos informações dos metadados das modificações realizadas em cada mudança, tais como: o autor do *commit*, quando ela aconteceu, a mensagem dos *commits* e informações de quantas linhas foram adicionas e removidas de cada artefato. Do Jira, coletamos informações sobre cada solicitação de mudança, tais como: o tipo da solicitação (defeito, melhoria, novo requisito, etc.), estado da solicitação (aberta, fechada, inválida, etc.), quem é o responsável, quem relatou a solicitação e a sua descrição. Também coletamos informações sobre a comunicação dos desenvolvedores, tais como: comentários e os respectivos autores. Para realizar a coleta do SCV usamos uma ferramenta chamada CVSAnalY[1]. Para coletar dados das solicitações de mudança e a comunicação dos desenvolvedores usamos a ferramenta Bicho[2].

Agregação dos Dados. Uma solicitação de mudança pode ser resolvida a partir de muitos *commits*. Nesses casos, é necessário realizar a agregação dos *commits* em uma única transação. Para agrupar as transações analisamos as mensagens de cada *commit* procurando por indicativos de um identificador único de uma solicitação de mudança. Normalmente, nos repositórios Apache, esse indicativo é feito a partir da consulta pelo símbolo "#"+"número da solicitação".

Assim, cada solicitação concluída deve possuir pelo menos um *commit* agregado. Solicitações sem nenhum *commit* agregado foram descartadas, assim como modificações feitas em artefatos e que não foram associados a uma solicitação de modificação. Verificamos também se o *commit* foi realizado dentro do período em que a solicitação de mudança foi aberta e a sua data de conclusão. Transações que modificaram mais de

[1] https://github.com/MetricsGrimoire/CVSAnalY
[2] https://github.com/MetricsGrimoire/Bicho

20 artefatos foram inspecionados para garantir que não se tratavam de operações de mudança de ramo no controle de versão, ou alguma refatoração que não era associada a tarefa.

D. Extração de Métricas e Acoplamentos de Mudanças

Extração dos acoplamentos de mudanças. Para extrair e identificar os acoplamentos de mudança, utilizamos o histórico de modificações dos artefatos de cada versão. Foram consideradas apenas as modificações (*commits*) feitas em tarefas que foram indicadas como concluídas e que o código fonte foi integrado no projeto. Após esta etapa de pré-processamento, detectamos os acoplamentos de mudanças fortes formalizando-os como regras de associação e escolhendo as 25% das regras mais interessantes. A seguir, descrevemos o que são as regras de associação e como escolhemos essas regras.

Uma regra de associação é uma implicação na forma $\mathtt{I} \Rightarrow \mathtt{J}$, onde $\mathtt{I}$ e $\mathtt{J}$ são dois conjuntos disjuntos de itens (*itemset*). Uma regra interessante $\mathtt{I} \Rightarrow \mathtt{J}$ significa que quando $\mathtt{I}$ acontece, $\mathtt{J}$ é provável de mudar conjuntamente. Em outras palavras, implica que mudanças que contêm $\mathtt{I}$ são prováveis de conter $\mathtt{J}$. No escopo deste estudo, a regra $\mathtt{I} \Rightarrow \mathtt{J}$ significa que J mudou conjuntamente com I. Também consideramos que $\mathtt{I}$ e $\mathtt{J}$ são conjuntos unitários de arquivos (a), onde $\mathtt{I} = \{\mathtt{a}_i\}$ e $\mathtt{J} = \{\mathtt{a}_j\}$ e $\mathtt{a}_i \neq \mathtt{a}_j$.

Regras de associação podem ser medidas por diferentes métricas. Nesse estudo, aplicamos as métricas de *suporte* e *confiança* que tem sido frequentemente utilizada em estudos anteriores de Engenharia de Software [7], [9]. O *suporte* da regra $\mathtt{r} = \mathtt{I} \Rightarrow \mathtt{J}$ correponde ao número de mudanças conjuntas de $\mathtt{I}$ e $\mathtt{J}$. Entretanto, *suporte* determina quão evidente a regra é. Por sua vez, a *confiança* é muitas vezes interpretada como a força de uma regra. Mais especificamente, dada uma regra $\mathtt{r}$, a *confiança* se refere para a fração de mudanças contendo $\mathtt{I}$ onde $\mathtt{J}$ também aparece. Então, uma regra interessante tem altos valores de *suporte* e *confiança*. Assim, podemos calcular *suporte* e *confiança* da seguinte forma:

$$\boldsymbol{suporte}(\boldsymbol{r}) = suporte(I \Rightarrow \mathrm{J}) = supporte(I \cup J)$$
$$= suporte(\{\mathrm{a}_i\} \cup \{\mathrm{a}_j\})$$
$$= \textit{o número de transações que contêm } a_i \text{ e } a_j$$

$$\boldsymbol{confiança}(\boldsymbol{r}) = confiança(I \Rightarrow \mathrm{J}) = \frac{suporte(I \Rightarrow J)}{suporte(I)}$$
$$= \frac{suporte(I \cup J)}{suporte(I)} = \frac{suporte(\{\mathrm{a}_i\} \cup \{\mathrm{a}_j\})}{suporte(\{a_i\})}$$
$$= \text{fração de transações contendo } a_i \text{ onde } a_j \text{ também aparece.}$$

Nesse estudo, as transações equivalem as mudanças que são feitas nos artefatos e são realizadas no escopo de uma solicitação de mudança (*issue*). Dessa forma, para cada versão, calculamos todas as possíveis regras envolvendo os pares de arquivos. Depois, para cada par de regra $<\mathtt{r}_1 = \mathtt{I} \Rightarrow \mathtt{J}, \ \mathtt{r}_2 = \mathtt{J} \Rightarrow \mathtt{I}>$, descartamos a regra com menor confiança (o suporte é sempre o mesmo).

Com o conjunto de regras restantes, ordenamos todas as regras com base no seu valor de suporte, utilizando a confiança como um desempate. Nós então selecionamos as 25 regras com maiores valores de suporte e confiança, que consideramos como a lista de acoplamento de mudanças fortes de cada versão. São para esses pares de arquivos que realizamos predição da propagação de mudança.

Cálculo das Métricas. Uma vez que os acoplamentos de mudança fortes são identificados e a lista com acoplamentos de mudança mais fortes é criada, podemos calcular as métricas de cada artefato. Para cada artefato do lado esquerdo do nosso acoplamento de mudança forte calculamos as métricas contextuais listadas na Tabela III.

A Tabela II apresenta um exemplo de como as métricas são calculadas. A coluna "AM forte", indica o par de arquivos que forma o acoplamento de mudança identificado no passo anterior. Suponha que o arquivo I e J formam um acoplamento de mudança forte em uma versão qualquer analisada. Nesse exemplo, o arquivo I é o arquivo do lado esquerdo do acoplamento de mudança, logo, ele é o arquivo que mudou mais individualmente nas solicitações de mudança quando comparado ao J. Nesse caso, o arquivo J é o artefato do lado direito da regra. Todas as métricas são calculadas somente para os artefatos do lado esquerdo. A razão para fazer isso é que acreditamos ser possível predizer as mudanças conjuntas a partir do artefato que mais muda. Cada linha na Tabela II indica um *commit* feito para o arquivo I. A coluna "*Commit*" indica o identificar único do *commit* e a coluna "SM" indica o identificador único de uma solicitação de mudança onde aquele *commit* foi agregado.

Rotulando Acoplamento de Mudanças. Para avaliar a propagação de mudança para cada acoplamento de mudança, precisamos construir arquivos de treinamento e teste para realizar a predição. Assim, com cada conjunto de treino é possível executar um algoritmo de classificação para prever quando o artefato I mudou conjuntamente com J ou não. No entanto, para essa predição precisamos rotular cada acoplamento de mudança forte.

Para criar um rótulo, adicionamos a coluna "Classe" na Tabela II. Para cada *commit* feito para o artefato I (arquivo da esquerda) rotulamos como "1" se o *commit* propaga mudanças para J (arquivo da direita). Do contrário, rotulamos a coluna "Classe" com valor 0 (sem mudança conjunta). Assim, valores de 1 indicam que o arquivo da esquerda (I) mudou conjuntamente com o arquivo da direita (J).

TABELA II. EXEMPLO DO CONJUNTO DE TREINO E TESTE.

AM forte	*Commit*	SM	Métricas Contextuais	Classe
$I \Rightarrow \mathrm{J}$	# 2	# 100	Métricas do *commit* do arquivo I, na solicitação de mudança #100	0
$I \Rightarrow \mathrm{J}$	# 4	# 100	Métricas do *commit* do arquivo I, na solicitação de mudança #100	1
$I \Rightarrow \mathrm{J}$	# 235	# 135	Métricas do *commit* do arquivo I, na solicitação de mudança #135	0
$I \Rightarrow \mathrm{J}$	# 321	# 200	Métricas do *commit* do arquivo I, na solicitação de mudança #200	0
$I \Rightarrow \mathrm{J}$	…	…	...	1

Um sumário das métricas extraídas das informações contextuais das solicitações de mudanças, comunicação dos desenvolvedores e modificação dos artefatos e a justificativa de uso de cada conjunto de métricas é apresentado na Tabela III.

TABELA III. SUMARIZAÇÃO DAS MÉTRICAS OBTIDAS DAS INFORMAÇÕES CONTEXTUAIS

Dimensão	Métrica	Tipo	Definição	Justificativa
Contexto das Solicitações de Mudanças (SM)	Tipo da Tarefa	String	O Tipo da solicitação de mudança indicada quando a tarefa foi descrita. Os valores podem ser (*Bug*, *Improvement*, *New Task*, *Documentation*, *Infrastructure*, etc.)	Mudanças conjuntas podem acontecer para resolver diferentes tipos de solicitação de mudanças. Por exemplo, algumas modificações podem ser mais propensas a acontecer para corrigir defeitos e outras para implementar novos requisitos.
	Responsável	String	Nome do Desenvolvedor indicado como Responsável pela resolução da solicitação de mudança	Um desenvolvedor pode trabalhar em solicitações de mudança de partes similares do software. Nesse sentido, eles podem ser propensos a modificar artefatos e submeter contribuições para um mesmo conjunto de artefatos.
	Relator	String	Nome do usuário ou desenvolvedor que relatou a solicitação de mudança	Usuários podem ser propensos a relatar problemas para um mesmo componente ou que afetem a mesma parte do software, uma vez que seus interesses podem ser para requisitos específicos do software. Assim os relatos de um usuário específico podem ser relacionados a um mesmo conjunto de artefatos
Contexto da Comunicação (CC)	Número de comentários	Numérico	Número total de comentários feitos na solicitação de mudança	A comunicação envolve aspectos de discussão em torno das solicitações de mudanças. O Número de comentários, de comentadores, de palavras e de desenvolvedores que comentaram em uma tarefa podem indicar o conjunto de artefatos que mudam conjuntamente. Por exemplo, algumas mudanças conjuntas podem acontecer e ter uma determinada quantidade de palavras ou comentários porque a solicitação de mudança é mais difícil de entender, assim modificar artefatos que são mais propensos de ser modificados nessas condições.
	Número de comentadores	Numérico	O Número de usuários e desenvolvedores distintos que comentaram na solicitação de mudança	
	Número de palavras	Numérico	Número total de palavras contidas na descrição e comentários realizados na solicitação de mudança.	
	Número de Desenvolvedores que comentaram	Numérico	Número total de desenvolvedores distintos que previamente modificaram o artefato e também escreveram comentários na solicitação de mudança	
Contexto da modificação (CM)	Autor do *commit*	String	Nome do desenvolvedor que modificou/ realizou o *commit* do artefato	Desenvolvedores podem contribuir para partes específicas do software. Nesse sentido, eles podem modificar sempre os mesmos artefatos. Esse indicador pode ser útil para identificar quando dois artefatos são modificados conjuntamente se o mesmo conjunto de desenvolvedores modificam os mesmos artefatos.
	Número de linhas de código adic./modificad.	Numérico	Soma do número de linhas adicionadas/modificadas em um *commit*	O número de linhas modificadas, excluídas podem indicar padrões de propensão de mudança conjunta. Alguns artefatos podem mudar mais linhas ou excluir mais linhas de código quando são mudados conjuntamente com alguns artefatos e apresentar um padrão diferentes quando a quantidade de mudanças conjuntas é menor.
	Número de linhas de código excluídas	Numérico	Soma do número de linhas excluídas em um *commit*	
	Code Churn	Numérico	Soma do número de linhas adicionadas e excluídas em um *commit*	

E. Predição (análise dos dados)

A Fig. 1 apresenta os passos da abordagem de Predição. Abaixo descrevemos os passos realizados para a predição de propagação de mudanças.

Construção dos Classificadores. Para construir os classificadores precisamos usar os conjuntos de treinamento e teste. Para cada versão criamos um arquivo ".csv" similar à Tabela II. Executamos classificadores baseado no algoritmo de *random forest* para predizer a propagação de mudanças. Esse algoritmo constrói um grande número de árvores de decisão para arquivos de treinamento usando um subconjunto aleatório de todas as métricas calculadas. Esses subconjunto aleatórios são divididos garantindo que exista baixa correlação entre as métricas evitando problemas de superajuste (*overfitting*) nos modelos [22].

Usando um método de votação a partir de todas as árvores criadas, o algoritmo de *random forest* decide se a pontuação final é maior do que o limiar escolhido para determinar se a mudanças conjunta será considera como "mudou conjuntamente" ou "não mudou conjuntamente". *Random forest* já foi utilizado em outros estudos de engenharia de software para entender a necessidade da construção (*build*) do software [23]. Para construir os classificadores implementamos *scripts* usando um pacote R chamado *Caret* [24].

Avaliação da Classificação. Para avaliar o desempenho de cada classificador primeiro construímos um classificador para cada par de arquivos usando o conjunto de treinamento. Dessa forma, para cada acoplamento de mudança forte, usamos os dados obtidos na versão "N" como conjunto de treinamento e usamos o conjunto de teste obtido na versão consecutiva "N+1". Com o conjunto de teste é possível avaliar o desempenho de cada classificador para cada acoplamento de mudança.

Por exemplo, suponha que o acoplamento de mudança $I \Rightarrow J$ foi obtido na posição TOP 3 da nossa lista de acoplamento de mudanças fortes na versão 1.0. Olhamos para as versões consecutivas para construir os conjuntos de testes para esse par de arquivo. Vamos supor, que encontramos *commits* feitos em solicitações de mudança para as versões 3.1, 3.2 e 3.4. Assim, construímos um conjunto de teste para a versão 3.1, similar a Tabela II e usamos para avaliar o nosso modelo. Depois disso, construímos um novo conjunto de treino usando os dados da versão 3.1, e um novo conjunto de teste para a versão 3.2 e avaliamos novamente se o classificador foi capaz de prever a propagação de mudança.

Descartamos a versão 3.4, porque não temos dados da versão 3.3 para construir o conjunto de treino. Nesse caso, o arquivo $I \Rightarrow J$ propagou mudanças em três versões consecutivas e avaliamos a capacidade do nosso classificador predizer corretamente essa propagação versão por versão.

Para avaliar o desempenho dos classificadores, métricas de avaliação são utilizadas. Em um problema de classificação binária, como é o nosso caso, uma matriz de confusão pode ser utilizada. A Tabela IV mostra um exemplo de matriz de

confusão utilizado nesse trabalho. O valor de verdadeiro positivo indica quando o valor observado na mudança como "1", ou seja, houve mudança conjunta, foi predita pelo nosso classificador como "1". Caso o classificador responda que não houve mudança, um valor de falso positivo é gerado. O contrário acontece com os valores de verdadeiro negativo, ou seja, não foi observada a mudança conjunta e o classificador corretamente predizem como "0". O Falso negativo ocorre quando o classificador aponta que houve uma mudança, quando de fato essa mudança conjunta não existiu.

TABELA IV. UM EXEMPLO DE MATRIZ DE CONFUSÃO

	Observado como:	
Predito como:	Mudou	Não mudou
Mudou	VP (verdadeiro positivo)	FP (falso positivo)
Não mudou	FN (falso negativo)	VN (verdadeiro negativo)

O desempenho de um classificador é medido em termos de Área sob a curva (*Area Under the Curve*, AUC). Também reportamos os valores da matriz de confusão em termos de número de VP, FP, FN e VN. Usando os valores da matriz de confusão é possível obter outras métricas de avaliação da predição, tais como a sensibilidade (recall), precisão (*precision*) e a Área sob a curva (*Area Under the Curve* – AUC).

Sensibilidade: Pode-se calcular a sensibilidade para identificar a proporção das instâncias que o modelo pode recuperar com sucesso que foram indicadas para mudar junto e de fato mudaram. Para se obter o valor de sensibilidade usamos a seguinte fórmula VP/VP+FN.

Precisão: A precisão pode medir a porcentagem correta de identificação das propagações de mudanças que mudaram conjuntamente. Para se obter o valor de precisão utilizamos a seguinte fórmula VP/VP+FP.

Área sob a curva: A área sob a curva representa graficamente a taxa de verdadeiro positivo (VP/VP+FP) no eixo y e a taxa de falso positivos (FP/FP+VN) no eixo x, para vários valores de um limiar usado para determinar se o arquivo é predito como parte de um conjunto de propagação de mudança ou não. Os valores de AUC estão no intervalo de 0 (pior classificador) a 1 (melhor classificador).

Um classificador randômico tem o valor de 0.5. Esse classificador randômico determina que o classificador não é capaz de determinar se existe ou não propagação de mudança. Em geral, modelos com valores superiores a 0.75 são considerados bons modelos. Valores acima de 0.80 são desejáveis. Essa medida é a principal métrica utilizada quando se deseja comparar o desempenho de modelos. No nosso caso, pretendemos comparar o desempenho de modelos construídos com o contexto das Solicitações de Mudanças (SM), contexto da Comunicação (CC) e o contexto das modificações (CM), bem como, as combinações desses contextos.

IV. RESULTADOS

O objetivo desse trabalho é explorar o impacto das informações contextuais obtidas das solicitações de mudanças, comunicação entre os desenvolvedores e das informações de mudança para prever a propagação de mudança entre os acoplamentos de mudança fortes. Nas seções seguintes, apresentamos e discutimos os resultados de cada questão uma das questões de pesquisa.

A. *É possível prever a propagação de mudanças entre dois artefatos que mudam frequentemente usando informações contextuais das solicitações de mudança, da comunicação dos desenvolvedores e das modificações dos artefatos?*

Abordagem. Construímos um classificador para cada acoplamento de mudança forte identificado em cada versão estudada dos quatro projetos. Cada modelo é avaliado a partir dos valores de AUC obtidos durante a predição realizada no arquivo de teste. Comparamos nossos classificadores com um classificador randômico que tem valor de 0.5 e também com o modelo baseado nas regras de associação. Com o objetivo de avaliar o impacto do uso prático da nossa abordagem, usamos a matriz de confusão para apresentar o desempenho do modelo. É importante mencionar que para responder QP1, os modelos utilizam todas as métricas indicadas na Tabela III.

Resultados. Nossos classificadores superam os classificadores randômicos nos quatros projetos e versões estudadas. Comparando nossos modelos com regras de associação, observamos uma melhora nos valores de sensibilidade de quase 4 vezes.

A métrica AUC é projetada de tal forma que um classificador randômico, ou seja, que tenderia a escolher uma classe aleatoriamente deveria alcançar um valor de AUC de 0.5 [25], [26]. A Tabela V apresenta a matriz de confusão dos classificadores e o valor médio do AUC obtidos em cada versão estudada. Reportamos os valores de VP, FP, FN e VN.

TABELA V. MATRIZ DE CONFUSÃO PARA OS ACOPLAMENTOS DE MUDANÇA FORTES AVALIADOS

Projeto	Versão	AUC (média)	Matriz de Confusão			
			VP	FP	FN	VN
Cassandra	1.0	0.851	44	13	41	540
	1.1	0.869	55	23	40	402
	1.2	0.928	69	13	38	311
Camel	2.1	0.897	3	17	7	25
Hadoop	2.0	0.753	15	13	23	111
Lucene	4.0	0.797	36	7	18	88
		0.849	**222**	**86**	**167**	**1477**

Os resultados em termos absolutos mostram que os classificadores construídos, no pior caso (projeto Hadoop) são 0.253 melhores que o classificador randômico (0.753 *vs* 0.500). No melhor caso, na versão 1.2 do projeto Cassandra, nossos classificadores tem o desempenho 0.428 melhor que o classificador randômico (0.928 *vs* 0.500). Comparados com os valores sugeridos de AUC, na média, todos os projetos obtiveram desempenho superior ao valor de 0.75.

Observando os valores dos modelos para cada projeto, verificamos que os modelos construídos para o projeto Hadoop obtiveram o pior desempenho. Cinco pares de arquivos tiveram modelos superiores a 0.9. Entretanto, outros cinco modelos tiveram o desempenho próximo de um classificador randômico. Esses modelos apresentam AUC entre 0.46 e 0.58. Para esses 5 pares nosso classificador não teria segurança em predizer se existiria propagação de mudança para a próxima solicitação de mudança ou não.

Para o projeto Cassandra, verificamos que 45 modelos tiveram AUC com valor superior a 0.90. Em 23 modelos, os valores de AUC ficaram entre 0.75 e 0.89. Somente 5 modelos

ficaram com valores entre 0.5 e 0.65. Para o Lucene, 3 modelos tiveram desempenho entre 0.52 e 0.72. Os outros 6 modelos construídos para esse projeto obtiveram AUC com valor superior à 0.82. Por fim, para o projeto Camel, 4 modelos apresentaram valor de AUC acima de 0.92. Um modelo obteve AUC com valor 0.66.

Uma vez que os modelos são projetados para predizer mudanças conjuntas para acoplamentos de mudança fortes também precisamos identificar quando um dos artefatos muda sozinho. Para esse fim, avaliamos a quantidade de falso positivos e falso negativos gerados por nossa abordagem.

Considerando todas as mudanças conjuntas entre os pares de arquivos avaliados, investigamos um total de 308 *commits* com mudanças conjuntas. Em 1644 *commits*, somente o arquivo do lado esquerdo da mudança conjunta foi modificado, e a propagação de mudança não ocorreu. Verificamos que nossos classificadores geraram 38.73% de falso positivos e 11.30% de falso negativos. Esses resultados são particularmente importantes, uma vez que o falso negativo e positivo podem causar sobrecarga de trabalho por conta de alerta desnecessário podendo causar um defeito ou futura manutenção.

Além disso, podemos observar uma melhoria nos valores de sensibilidade da nossa abordagem comparado com a literatura [10]. Enquanto os valores de sensibilidade reportados na literatura alcançam no máximo 25%, nossos modelos alcançaram 57.06%. Em relação a precisão os valores encontrados na literatura sugerem valores entre 70% e 90%. Nossos modelos apresentaram na média 72% de precisão.

Apesar da melhoria da sensibilidade, ressaltamos que uma comparação direta com os trabalhos indicados na literatura é difícil. Cada trabalho, utilizou um conjunto de dados diferente o que dificulta a comparação sem a replicação completa de todas as abordagens. Para minimizar os efeitos dessa comparação indireta, comparamos os resultados dos modelos criados com informações contextuais com modelos obtidos por regras de associação usadas para identificar os acoplamentos de mudanças fortes. Uma vez que cada acoplamento de mudança forte tinha uma regra indicando que dois artefatos eram propensos a mudar conjuntamente na próxima versão, assumimos que todas as solicitações de mudanças onde o arquivo da esquerda da regra era modificado, o arquivo da direita também mudou. Dessa forma, o valor de precisão para esse modelo é de 100% uma vez que ele recomendaria todas as mudanças conjuntas desses dois artefatos na próxima versão porque a regra de associação para aquele par de artefato existe.

Entretanto, para o valor de sensiblidade, é importante que a abordagem detecte quando dois artefatos não mudam conjuntamente. Assim, para as regras de associação os valores de FN necessários para o cálculo da sensibilidade podem ser dados pela soma dos valores de FN e VN existentes (vide Tabela V). Logo o valor de sensibilidade calculada para as regras de associação é dada pela expressão (VP+FP)/ (VP+FP) + (FN+VN). Assim, os valores de sensibilidade obtidos por nossos modelos (57.06%) são quase quatro vezes maiores quando comparados com os valores de sensibilidade dos modelos usando regra de associação (15.72%).

Essa comparação é importante porque mostra que os valores de falso negativos do nosso modelo são inferiores quando comparados às regras de associação. Em um cenário real, por exemplo, podemos reduzir a quantidade de alertas falsos gerados uma vez que nosso valor de sensibilidade é maior. Assim, os modelos usando informações contextuais são úteis para prever a propagação de mudança, mas também para perceber quando não ocorreu uma propagação.

Nossos classificadores podem ser avaliados como bons preditores de propagação de mudança. Verificamos que 83% dos modelos obtiveram AUC superior a 0.75. Isso corresponde a um total de 84 modelos. Os modelos apresentaram taxa de falso positivo de 38.73% e falso negativo de 11.30%. A melhora da acurácia foi percebida especialmente em termos de sensibilidade.

B. (QP2) Qual é a dimensão contextual mais importante para a predição de propagação de mudanças?

Abordagem. Para estudar qual ou quais dimensões são mais importantes para a predição de propagação de mudanças comparamos o valor de AUC dos modelos contendo somente cada dimensão individualmente, combinadas aos pares e com todas as dimensões juntas.

Para realizar a comparação foi utilizado o teste estatístico não paramétrico *Mann-Whitney-Wilcoxon Test*. Uma vez que não é possível garantir que os grupos testaram os mesmos pares de arquivos, já que algumas métricas são categóricas, e em algumas situações elas podem ser removidas dos classificadores, optou-se por usar um teste não pareado e para comparar dois grupos que são independentes. No nosso estudo, os valores comparados são os valores de AUC dos modelos. A hipótese nula desse teste indica que os valores de um classificador contendo uma dimensão, ou um par de dimensão são maiores que outro classificador contendo uma outra dimensão individual ou outro par de dimensões. Todos os testes foram realizados com o valor de $p < 0.05$, ou seja, um nível de significância de 95%.

Resultados. A solicitação de mudança é a principal dimensão para propagação de mudança. A combinação dessa dimensão com o contexto da comunicação ou o contexto da modificação dos artefatos geram modelos melhores. A combinação das três dimensões juntas não resulta em modelos melhores.

A Fig. 2 mostra um boxplot comparando as dimensões estudadas e as suas combinações por projeto e versão estudada. Observando o gráfico é possível observar diferenças de desempenho entre as informações contextuais em diferentes projetos. Acreditamos que essas diferenças são relativas a forma que os projetos são organizados. Alguns projetos podem, por exemplo, apresentar diferenças de colaboração entre os desenvolvedores em relação a comunicação. Essa intuição é baseada em estudos de comunidades de projetos de software livre que indicam essas diferenças de organização [4], [27].

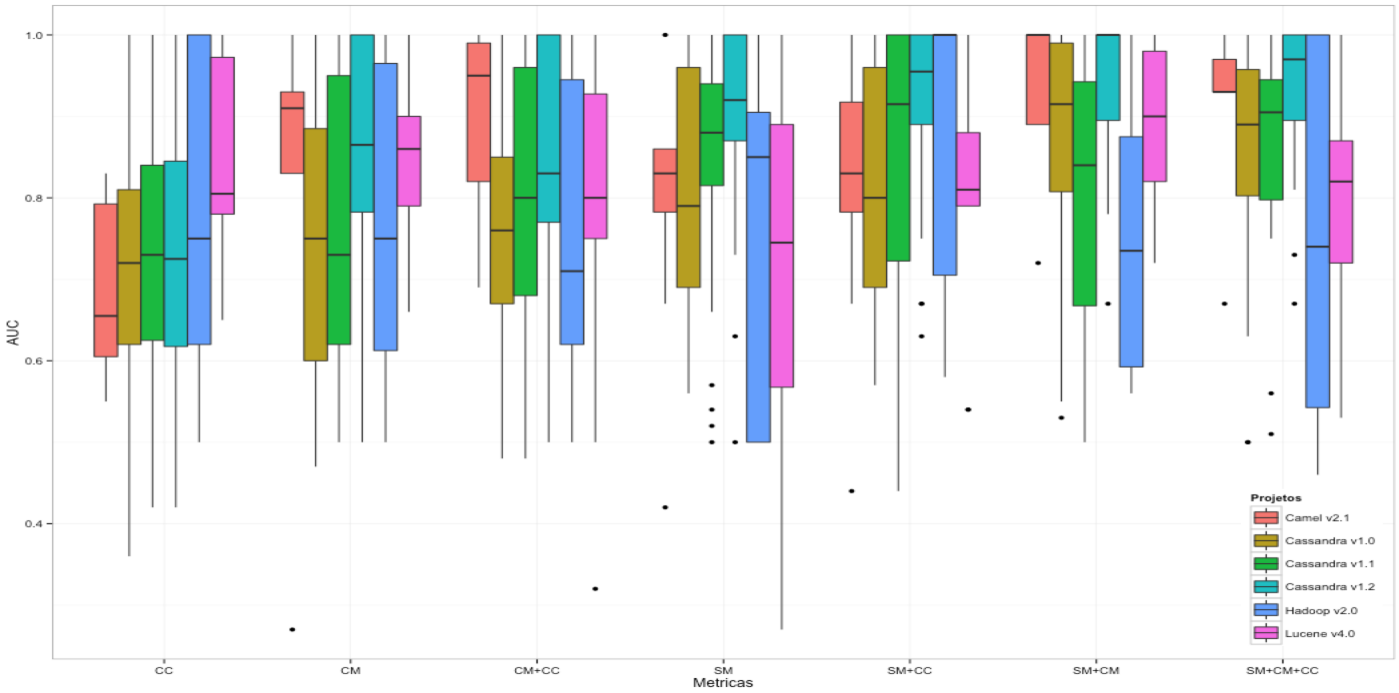

Fig. 2 Boxplot de comparação das informações contextuais capturadas para predição da propagação de mudança.

No que diz respeito a dimensão contextual de comunicação (CC) podemos observar que boa parte dos modelos apresentam mediana do valor de AUC próxima de 0.7. O melhor desempenho dessa dimensão é observado no projeto Lucene, com mediana do valor de AUC próxima de 0.8. Em relação a dimensão contextual de modificação dos artefatos (CM), observamos que três projetos apresentaram AUC superior a 0.8, casos dos projetos Camel v2.1, Cassandra v1.2 e Lucene v4.0. No caso específico do projeto Camel, os modelos apresentaram valor de AUC superior a 0.9.

A dimensão de solicitação de modificação (SM) apresentou 4 projetos com valores de AUC acima de 0.8. A versão 1.0 do projeto Cassandra apresentou mediana do valor de AUC próxima de 0.8. Entretanto, também observamos que essa dimensão também obteve os piores modelos considerando os valores mais baixos de AUC. Isso pode ser claramente observado pela maior quantidade de modelos considerados fora do padrão de distribuição (*outliers*) indicados pelos pontos no gráfico. Especialmente nos projetos Lucene e Cassandra, é possível observar esse comportamento dos modelos quando agrupamos a dimensão de comunicação com as modificações dos artefatos (CM+CC), percebemos uma melhoria nos modelos. Nessa dimensão, obtivemos quatro projetos com valores de AUC acima de 0.8.

A combinação da dimensão de solicitação de mudanças com a comunicação (SM+CC) ou com a modificação dos artefatos (SM+CM) apresentam os melhores resultados obtidos. No primeiro caso, SM+CC tem todas as medianas para todos os projetos e versões estudadas com valores de AUC acima de 0.8. Para a combinação de SM+CM, o projeto Hadoop apresentou mediana dos valores de AUC entre 0.8 e 0.7, enquanto os demais projetos apresentaram medianas acima de 0.8. Em valores absolutos essa dimensão apresentou 4 das seis versões com medianas dos valores de AUC acima de 0.9.

É interessante observar que quando comparamos essas duas últimas combinações com todas as dimensões usadas conjuntamente (SM+CM+CC), não é possível observar no gráfico uma melhoria de desempenho, o que nos permite concluir que a união de duas dimensões é suficiente para obter modelos consistentes para a propagação de mudança.

Uma vez realizada a análise visual, realizamos uma série de comparações estatísticas comparando as dimensões dentro de cada versão e projeto individualmente e comparando todos os projetos e versões conjuntamente. Observamos algumas diferenças estatísticas quando comparamos as dimensões contextuais. Por exemplo, no projeto Cassandra v1.0, a dimensão SM tem melhor desempenho quando comparado com a CC ($p < 0.027$), e CM ($p < 0.018$). Entretanto, quando comparamos as dimensões agrupadas não é possível observar diferença estatística entre elas e até mesmo quando comparamos um par de dimensão (por exemplo SM+CM) agrupada contra todas as dimensões juntas (SM+CM+CC).

Para o projeto Cassandra v1.1, encontramos as mesmas diferenças estatísticas da versão anterior. Quando comparamos SM contra CC ($p < 0.028$) e CM ($P < 0.090$). Quando comparadas as dimensões agrupadas em pares e essas mesmas dimensões contra o agrupamento de todas as informações contextuais, não houve diferença estatística. Para a terceira versão do projeto Cassandra, uma nova diferença estatística foi encontrada quando comparamos CM contra CC ($p< 0.039$). Além disso também encontramos diferença estatística entre SM+CC contra CM+CC ($p < 0.039$). As demais comparações não apresentaram diferença

estatística. Para o projeto Camel, encontramos diferenças entre SM e CC ($p < 0.028$). Para o Hadoop e Lucene, não encontramos nenhuma diferença estatística entre as informações contextuais.

Nossos classificadores derivam muito da dimensão de solicitação de mudança (SM). Entretanto, quando combinamos a solicitação de mudança com outra dimensão a quantidade de falso positivos e negativos diminui gerando modelos melhores. Não existe diferenças estatísticas quando duas dimensões são comparadas com todas as dimensões conjuntamente. Essa característica é particularmente interessante pois diminui o esforço de construção dos modelos, uma vez que a quantidade de métricas necessárias diminuiu.

V. AMEAÇAS À VALIDADE

Potenciais ameaças a validade podem afetar os resultados do nosso estudo.

Generalização: Neste estudo, analisamos 4 projetos, sendo que em um deles utilizamos 3 versões. Entretanto, nossos resultados podem não valer para outros projetos. Para reduzir essas ameaças, procuramos usar projetos de diferentes domínios, e em pelo menos um projeto, estudar mais versões consecutivas. A replicação desse trabalho em um grande conjunto de dados é necessária para se obter conclusões mais gerais.

Acoplamentos de mudança forte: Uma das ameaças de validade em torno dos acoplamentos de mudança é relacionada ao conceito de *tangled changes* [28], que se refere à interação dos desenvolvedores com o SCV. Desenvolvedores frequentemente realizam *commits* não relacionados ou de código que muitas vezes não dizem respeito a mesma tarefa. Nesse estudo, essa ameaça é limitada, uma vez que agrupamos e consideramos os *commits* feitos somente em solicitações de mudança. Além disso, executamos uma cuidadosa seleção de solicitações de mudança, considerando somente aquelas cujo as alterações foram integradas no projeto. Os acoplamentos de mudança identificados e usados para a predição de propagação de mudança compreendem arquivos que variam de suporte igual a 1 e confiança de 20% até suporte igual a 20 e confiança de 100%. Esse intervalo mostra que nossos modelos podem ser utilizados com pouco histórico de mudanças na versão anterior e, portanto, mudaram pouco entre si.

Métricas: O conjunto de informações contextuais capturados nesse trabalho pode não ser completo. Tratamos essa ameaça aplicando pelo menos três dimensões de informações contextuais: Métricas obtidas das solicitações de mudança, da comunicação e das modificações realizadas nos artefatos. Para selecionar essas métricas nós nos inspiramos em pesquisas anteriores realizadas em modelos de predição e revisões sistemáticas feitas por nós e por outros autores [23], [29], [30].

VI. TRABALHOS RELACIONADOS

Zimmermann *et al.* [7] construiu um *plug-in* para o ambiente de desenvolvimento Eclipse que coleta informação sobre as mudanças dos códigos fontes do repositório e alerta os desenvolvedores sobre prováveis mudanças conjuntas. Os autores usaram regras de associação para sugerir acoplamentos de mudança no nível de arquivos e métodos baseado nos valores de suporte, ou seja, na frequência histórica que eles ocorreram. Os autores relataram valores de precisão em torno de 30% e recomendam que para projetos com evolução contínua a precisão pode ser melhor se feita no nível de arquivo.

Canfora *et al.* [10] mostrou que os a abordagem proposta por Zimmermman pode ser melhorada se combinada com o teste de causalidade de Granger. Segundo Canfora et al., a combinação de abordagens permite melhorar a acurácia e precisão das regras utilizadas para prever as mudanças conjuntas entre os artefatos.

Zhou *et al.* [31] propôs um modelo para predizer mudanças conjuntas usando redes bayesianas. Eles extraíram informações sobre a dependência estática do código, a frequência passada das mudanças conjuntas, o período da mudança e quem realizou a mudança. Eles executaram os modelos em dois projetos de software livre: Azureus e ArgoUML reportando precisão e sensibilidade em torno de 60% e 40% respectivamente.

Nosso artigo difere das pesquisas anteriores uma vez que nenhum deles considerou o aspecto contextual associado as mudanças. A ideia de usar essas informações é nova e apresentou resultados promissores uma vez que os modelos construídos apresentaram valores de AUC acima do recomendado (superiores a 0.75), melhoria nas taxas de sensibilidade e redução da quantidade de falso negativos.

VII. CONCLUSÕES E TRABALHOS FUTUROS

Este artigo investigou como questão central se as informações contextuais obtidas das solicitações de mudança, comunicação entre os desenvolvedores e as modificações nos artefatos são úteis para prever a propagação de mudanças entre artefatos que mudam frequentemente. Do ponto de vista prático, nossa abordagem pode ser útil para alertar os desenvolvedores, durante a realização de uma tarefa, de uma possível propagação de mudança conjunta. Além disso, pode ser possível utilizar a nossa abordagem para auxiliar no fechamento de solicitações de mudanças que foram reabertas. Para Shihab *et al.* [2], um dos principais motivos associados a reabertura de solicitações de mudanças está associado à quantidade de artefatos e mudanças necessárias para concluir determinadas solicitações.

Por meio dos estudos realizados em quatro projetos hospedados na Apache Foundation concluímos que:

QP1. Os classificadores construídos com informações contextuais foram capazes de predizer a propagação de mudança alcançando 0.849 de AUC na média dos projetos e versões estudadas. Esse valor é superior a um classificador randômico (AUC 0.5). Comparado com as regras de associação (baseados em propensão de mudança), identificamos que nossos modelos melhoraram os valores de sensibilidade (57.06% vs. 15.70%), especialmente porque podem evitar um alerta falso quando não existe uma mudança propagada. Esses resultados podem reduzir possíveis efeitos negativos de uma possível adoção da nossa abordagem na prática.

QP2. A propagação de mudança pode ser predita usando diferentes informações contextuais. Entretanto, o uso das três dimensões conjuntamente não melhorou os resultados de predição em comparação a quando duas dimensões foram combinadas. Com a quantidade de projetos estudados não foram verificadas diferenças estatísticas significantes, entretanto, quando cada dimensão individualmente foi testada, as informações contextuais das solicitações de mudança obtiveram os melhores resultados.

Como trabalho futuro, pretendemos combinar nossa abordagem com análise estrutural. Além disso, temos planejado estender a análise para mais artefatos, versões e projetos, além de construir uma ferramenta para dar suporte aos desenvolvedores em novos projetos, por exemplo, para auxiliar no fechamento de solicitações de mudança que foram reabertas.

AGRADECIMENTOS

Gostaríamos de agradecer a Fundação Araucária, NAPSOL, NAWEB, FAPESP e CNPQ (461101/2014-9) pelo suporte financeiro. Marco Aurélio Gerosa recebe apoio do CNPQ. Igor recebe suporte da CAPES (BEX 2039-13-3).

REFERÊNCIAS

[1] A. E. Hassan and R. C. Holt, "Predicting change propagation in software systems," in *IEEE International Conference on Software Maintenance, ICSM*, 2004, pp. 284–293.

[2] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. I. Matsumoto, "Predicting re-opened bugs: A case study on the Eclipse project," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2010, pp. 249–258.

[3] N. Bettenburg, A. E. Hassan, B. Adams, and D. M. Germán, "Management of community contributions," *Empir. Softw. Eng.*, vol. 20, no. 1, pp. 252–289, 2015.

[4] I. Steinmacher, A. P. Chaves, T. U. Conte, and M. A. Gerosa, "Preliminary empirical identification of barriers faced by newcomers to Open Source Software projects," in *Software Engineering (SBES), 2014 Brazilian Symposium on*, 2014, pp. 51–60.

[5] S. A. Bohner and R. S. Arnold, *Software Change Impact Analysis*. 1996.

[6] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated impact analysis for managing software changes," in *Proceedings - International Conference on Software Engineering*, 2012, pp. 430–440.

[7] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," *IEEE Trans. Softw. Eng.*, vol. 31, pp. 429–445, 2005.

[8] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," *Proceedings. Int. Conf. Softw. Maint. (Cat. No. 98CB36272)*, 1998.

[9] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Trans. Softw. Eng.*, vol. 30, pp. 574–586, 2004.

[10] G. Canfora, M. Ceccarelli, L. Cerulo, and M. Di Penta, "Using multivariate time series and association rules to detect logical change coupling: An empirical study," in *IEEE International Conference on Software Maintenance, ICSM*, 2010.

[11] A. Orso, T. Apiwattanapong, J. Law, G. Rothermel, and M. J. Harrold, "An empirical comparison of dynamic impact analysis algorithms," *Proceedings. 26th Int. Conf. Softw. Eng.*, 2004.

[12] L. C. Briand, J. Wust, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," *Proc. IEEE Int. Conf. Softw. Maint. - 1999 (ICSM'99). 'Software Maint. Bus. Chang. (Cat. No.99CB36360)*, 1999.

[13] H. Kagdi, M. Gethers, and D. Poshyvanyk, "Integrating conceptual and logical couplings for change impact analysis in software," *Empir. Softw. Eng.*, vol. 18, pp. 933–969, 2013.

[14] M. Gethers, H. Kagdi, B. Dit, and D. Poshyvanyk, "An adaptive approach to impact analysis from change requests to source code," in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'2011)*, 2011, pp. 540–543.

[15] E. Hill, L. Pollock, and K. Vijay-Shanker, "Exploring the neighborhood with dora to expedite software maintenance," in *ASE'07 - 2007 ACM/IEEE International Conference on Automated Software Engineering*, 2007, pp. 14–23.

[16] M. Gethers and D. Poshyvanyk, "Using Relational Topic Models to capture coupling among classes in object-oriented software systems," in *IEEE International Conference on Software Maintenance, ICSM*, 2010.

[17] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Trans. Softw. Eng.*, vol. 35, pp. 864–878, 2009.

[18] M. D'Ambros, M. Lanza, and R. Robbes, "On the Relationship Between Change Coupling and Software Defects," in *Reverse Engineering, 2009. WCRE '09. 16th Working Conference on*, 2009, pp. 135–144.

[19] S. Kirbas, A. Sen, B. Caglayan, A. Bener, and R. Mahmutogullari, "The Effect of Evolutionary Coupling on Software Defects: An Industrial Case Study on a Legacy System," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 6:1–6:7.

[20] M. D'Ambros, M. Lanza, and M. Lungu, "Visualizing co-change information with the evolution radar," *IEEE Trans. Softw. Eng.*, vol. 35, pp. 720–735, 2009.

[21] T. Ball, J. Kim, and H. P. Siy, "If your version control system could talk," *ICSE Work. Process Model. Empir. Stud. Softw. Eng.*, 1997.

[22] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, 2001.

[23] S. McIntosh, B. Adams, M. Nagappan, and A. E. Hassan, "Mining Co-Change Information to Understand when Build Changes are Necessary," in *Proc. of the 30th Int'l Conf. on Software Maintenance and Evolution (ICSME)*, 2014, pp. 241–250.

[24] M. Kuhn, "Building Predictive Models in R Using the caret Package," *J. Stat. Softw.*, vol. 28, no. 5, pp. 1–26, 2008.

[25] D. M. W. Powers, "The problem of area under the curve," in *Proceedings of 2012 IEEE International Conference on Information Science and Technology, ICIST 2012*, 2012, pp. 567–573.

[26] D. M. Powers, "Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation," *J. Mach. Learn. Technol.*, vol. 2, pp. 37–63, 2011.

[27] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 2008.

[28] K. Herzig and A. Zeller, "The impact of tangled code changes," in *IEEE International Working Conference on Mining Software Repositories*, 2013, pp. 121–130.

[29] I. S. Wiese, F. R. Côgo, R. Ré, I. Steinmacher, and M. A. Gerosa, "Social metrics included in prediction models on software engineering: a mapping study," in *The 10th International Conference on Predictive Models in Software Engineering, {PROMISE} '14, Torino, Italy, September 17, 2014*, 2014, pp. 72–81.

[30] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6. pp. 1276–1304, 2012.

[31] Y. Zhou, M. Wursch, E. Giger, H. Gall, and J. Lu, "A Bayesian Network Based Approach for Change Coupling Prediction," *Fifteenth Work. Conf. Reverse Eng. Proc.*, pp. 27–36\r348, 2008.