UMLCollab: A Hybrid Approach for Collaborative Modeling of UML Models

UMLCollab: Uma abordagem híbrida para modelagem colaborativa de modelos UML

McLyndon S. de L. Xavier
Universidade do Vale do Rio Sinos
São Leopoldo, RS
mlimaxavier2@edu.unisinos.br

Kleinner Farias
Universidade do Vale do Rio Sinos
São Leopoldo, RS
kleinnerfarias@unisinos.br

Jorge Barbosa
Universidade do Vale do Rio Sinos
São Leopoldo, RS
jbarbosa@unisinos.br

Lucian Gonçales
Universidade do Vale do Rio Sinos
São Leopoldo, RS
lucianj@edu.unisinos.br

Vinicius Bishoff
Universidade do Vale do Rio Sinos
São Leopoldo, RS
viniciusbischof@edu.unisinos.br

ABSTRACT

In collaborative software modeling the two main types of collaboration still present problems, such as the constant interruptions that hinder the cognitive process in synchronous collaboration, and the complicated and costly stages of conflict resolution in asynchronous collaboration. For this, this paper proposes a technique called "UMLCollab". This technique combines aspects from synchronous and asynchronous collaboration. Through experiments, developers applied the proposed solution and they achieved to an intermediate productivity in relation to traditional collaboration methods. The results showed that the "UMLCollab" improved the correctness of the changed models, the notion of developer regarding to the resolution of conflicts, and enabled the parallel changes occurring while other collaborators are working on without degrade the software diagrams being modelled locally.

CCS CONCEPTS

• **Information systems** → *Synchronous editors*; *Asynchronous editors*; • **Software and its engineering** → *Software design engineering*; Programming teams.

KEYWORDS

Software Engineering Oriented by Collaborative Models , Collaborative Software Modeling , Collaborative Software Engineering

ACM Reference Format:
McLyndon S. de L. Xavier, Kleinner Farias, Jorge Barbosa, Lucian Gonçales, and Vinicius Bishoff. 2019. UMLCollab: A Hybrid Approach for Collaborative Modeling of UML Models. In *XV Brazilian Symposium on Information Systems (SBSI'19), May 20–24, 2019, Aracaju, Brazil.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3330204.3330239

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SBSI'19, May 20–24, 2019, Aracaju, Brazil
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7237-4/19/05…$15.00
https://doi.org/10.1145/3330204.3330239

1 INTRODUÇÃO

A colaboração é um fator crucial para a manutenibilidade e produtividade no desenvolvimento de sistemas de informação. O desenvolvimento de software colaborativo é uma tendência crescente na indústria de software [1]. Nesse ambiente, engenheiros trabalham de forma independente e paralela nos mesmos artefatos de software [8]. A alteração de componentes de software de forma simultânea permite aos desenvolvedores focarem nas partes mais relevantes dos modelos de software [10]. Portillo-Rodríguez et al. [18] ressaltaram que apesar das vantagens, vários desafios precisam ser resolvidos em relação ao desenvolvimento de software colaborativo para tornar-se uma solução definitiva nos ambientes de desenvolvimento de software. Os autores ressaltaram também que a distância dos times de desenvolvimento prejudica a comunicação entre membros da equipe e também dificulta o gerenciamento das equipes. Além disso, as diferenças culturais causam mal-entendidos, e prejudica a relação de confiança entre os membros.

Polančič e Jošt [17] também ressaltaram que as organizações precisam de ferramentas eficientes para colaboração síncronas e assíncronas em ambientes distribuídos. Além disso, futuras técnicas de colaboração devem viabilizar que cada membro da equipe esteja ciente das mudanças realizadas em paralelo com informações atualizadas em relação as alterações. Costa e Murta [7] identificaram que existe a necessidade de ferramentas para detecção de conflitos de código fonte em tempo real [11]. Apesar dos esforços para evoluir e melhorar técnicas e metodologias para a resolução de conflitos de código, poucos avanços foram alcançados na resolução de conflitos nos modelos de software [8]. O principal desafio é a sincronização das mudanças em tempo real durante modelagem colaborativa de diagramas de software [15]. Atualmente, essas sincronizações ocorrem em modo síncrono ou assíncrono.

Na colaboração síncrona, cada desenvolvedor envia e recebe as alterações dos demais colaboradores de forma instantânea. Esse tipo de colaboração evita conflitos nos modelos, porém o recebimento constante de alterações vindas dos demais colaboradores, especialmente em cenários com mais de dois colaboradores, podem causar várias interferências no trabalho cognitivo dos desenvolvedores devido ao excesso de interrupções.

A Figura 1 apresenta um exemplo do problema da colaboração síncrona em dois cenários. Em ambos os cenários há um servidor de

RIGHTSLINK

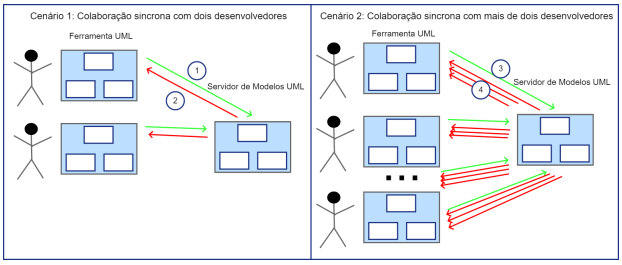

Figura 1: Comparação de colaboração com vários desenvolvedores

modelos UML responsável por receber as alterações de cada desenvolvedor, e enviá-las aos desenvolvedores que estão colaborando no mesmo modelo. No primeiro cenário, com somente 2 desenvolvedores, ocorre o envio síncrono das alterações locais (passo 1), e o recebimento síncrono das alterações remotas do outro desenvolvedor (passo 2). Como somente existe outro desenvolvedor colaborando com o modelo simultaneamente, a interferência no trabalho de cada usuário pode ser considerada como razoável. Porém, no cenário 2 há uma maior quantidade de desenvolvedores alterando os diagramas simultaneamente. Nesse caso, semelhante ao cenário 1, o desenvolvedor envia suas alterações locais de forma síncrona (passo 3), mas o recebimento das alterações remotas (passo 4) é proporcional ao número de usuários trabalhando simultaneamente no mesmo modelo. Isto causa na maioria das vezes uma interferência em maior escala no trabalho de cada usuário.

Na colaboração assíncrona o envio e recebimento de alterações é feito manualmente pelo desenvolvedor, isto é, a sincronização ocorre assim que o desenvolvedor atualiza o repositório aonde estão os arquivos dos modelos, normalmente através de um sistema de controle de versão, tais como o Subversion (SVN) ou o Git. Dessa forma, o desenvolvedor trabalha nas suas alterações e apenas recebe as alterações dos demais colaboradores quando requisitar ao repositório de desenvolvimento. As principais desvantagens desse tipo de colaboração, é a alta complexidade e a exigência de mais esforço dos desenvolvedores para a resolução de conflitos. Além disso, os conflitos tendem a se intensificar se houver demora para sincronizar os arquivos dos modelos locais com o repositório.

Dado a existência dessas desvantagens em ambos os tipos de colaboração, esse trabalho propõe uma técnica híbrida que combine as vantagens da sincronização síncrona e assíncrona, buscando minimizar as deficiências de cada uma delas. Dessa forma, a seguinte questão de pesquisa foi investigada nesse artigo: "Como combinar as vantagens da colaboração síncrona com a colaboração assíncrona de forma a permitir a modelagem colaborativa com menor esforço e maior produtividade para o desenvolvedor?"

O restante desse artigo é apresentado nas seis seções seguintes. Seção 2 apresenta a fundamentação teórica dessa pesquisa. Seção 3 descreve os trabalhos relacionados a modelagem colaborativa de software. Seção 4 apresenta a técnica proposta para a resolução de conflitos de modo colaborativo em modelos de software. Seção 5 descreve a avaliação da técnica proposta. Seção 6 analisa e apresenta os resultados obtidos. Finalmente, a seção 7 apresenta as considerações finais e descreve os trabalhos futuros dessa pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção apresenta a fundamentação teórica relacionada a modelagem colaborativa de modelos de software. Na seção 2.1 são apresentados os conceitos relacionados a modelagem colaborativa de software. Na seção 2.2 falamos sobre resolução de conflitos.

2.1 Modelagem Colaborativa de Software

A modelagem de software colaborativa é uma crescente tendência tanto nas pesquisas acadêmicas, e na indústria de software [13][4]. Os modelos no contexto do desenvolvimento de software são usados para representar, de forma simplificada, elementos de nossa realidade, com o intuito de permitir o entendimento dos conceitos e dinâmicas relacionados ao sistema sendo desenvolvido [16]. Além disso, o desenvolvimento de software é atualmente uma atividade colaborativa entre vários desenvolvedores que cada vez mais valorizam a colaboração para os projetos de software [16].

Caso os desenvolvedores de modelagem de software atuem no mesmo local, eles podem interagir informalmente conversando sobre os modelos sendo elaborados, e assim discutir, e resolver conflitos para convergir nas decisões em equipe [16]. Porém, essa interação pode ser problemática se os desenvolvedores estão distribuídos globalmente [19]. As ferramentas de modelagem distribuídas ainda dependem da comunicação por chat, videoconferência, e mecanismos de compartilhamento como Dropbox ou anexos de e-mail, que geralmente afetam negativamente a produtividade nesse ambiente [16].

2.2 Resolução de conflitos

Bang et al. [21] e Altmanninger et al. [2] sugeriram dois métodos para os sistemas de controle de versão: o pessimista, que mantém o artefato de software bloqueado para que seja alterado apenas por um membro da equipe em um determinado momento; e o otimista, onde cada membro tem sua cópia local dos artefatos para integrar as alterações posteriormente.

O método pessimista não evita conflitos de dependência entre artefatos, e impede que mais de um membro trabalhe na evolução do modelo ao mesmo tempo. A abordagem otimista não informa as alterações dos outros desenvolvedores. Consequentemente, mais conflitos ocorrem. Além disso, como as notificações sobre conflitos são adiadas até o próximo merge dos modelos, as resoluções de conflitos se postergam e consequentemente ocorre o aumento da lista de conflitos. Apesar de geralmente ser possível fazer merge automático de alterações em diferentes partes dos documentos, é mais complicado tratar conflitos de documentos não textuais [9].

Bang, Popescu e Medvidovic [21] ressaltaram ainda que as ferramentas de modelagem devem tratar de dois tipos de conflitos: conflitos de sincronização, e conflitos de alta ordem. Os conflitos de sincronização são aqueles que ocorrem na alteração de um mesmo artefato ou outro diretamente relacionado. Conflitos de alta ordem ocorrem por violações da sintaxe de modelagem ou semântica dos modelos, e por fim são mais difíceis e custosos para detecta-los.

Para Dam e Ghose [8] esses conflitos podem ser denominados de conflitos diretos e indiretos, respectivamente. Além desses problemas, o ambiente de modelagem colaborativo não é estável pois as mudanças realizadas por um desenvolvedor podem falhar devido as alterações em paralelo, tais como um método que foi renomeado ou

RIGHTSLINK

excluído por outro desenvolvedor sem o conhecimento do primeiro [8]. Dam e Ghose frisaram que, com relação as várias visões de um modelo UML, como diagramas de classes e diagramas de sequência, é importante manter a consistência sintática (conformidade com o metamodelo) e semântica (coerência entre as diferentes visões de um modelo).

3 TRABALHOS RELACIONADOS

Essa seção apresenta os trabalhos sobre técnicas híbridas de colaboração síncrona e colaboração assíncrona, e demais técnicas que são relevantes para modelagem colaborativa de software. Para selecionar os trabalhos nesse tema de pesquisa foi realizada uma revisão sistemática da literatura. Por isso, critérios de seleção de estudos relacionados foram definidos visando encontrar estudos sobre o desenvolvimento colaborativo de modelos de software. Para iniciar o processo de revisão desta pesquisa e encontrar os estudos relacionados foram selecionadas as bases de dados. Os artigos pesquisados em cada base de dados foram catalogados com o auxílio do aplicativo Mendeley [22]. A string de busca seguinte foi utilizada e adaptada nos motores de busca selecionados: "*(Collaborative OR Cooperative OR Team OR Contribute OR Distributed) AND (Software OR Application OR System OR Diagram) AND (Modeling OR Development)*". Em seguida foram definidos critérios para inclusão e exclusão dos artigos no processo de seleção dos artigos.

As buscas resultaram inicialmente em 2.272 artigos. Após a remoção de impurezas, e filtragem por título dos trabalhos encontrados, restaram 388 artigos. Destes, apenas 52 restaram após serem filtrados pelo resumo ou abstract. Os textos completos destes 52 trabalhos foram analisados, e partir desse critério restaram 18 trabalhos que tinham relação com os objetivos dessa pesquisa. No entanto, 4 estudos foram removidos pois eram artigos não eram estudos empíricos resultando em 14 artigos. Dessa amostra, 4 artigos de controle foram utilizados nessa pesquisa. Cada etapa do processo de seleção dos estudos foi validada por pares. Como possível fragilidade em relação a validade dessa revisão sistemática, há uma restrição das palavras chaves utilizadas na string de busca. Os quatro artigos de controle são descritos e analisados em seguida.

Brosch et al. [5] propuseram uma técnica de merge colaborativo para evitar que esta tarefa seja realizada por um único usuário, o que normalmente é uma atividade demorada e sujeita a erros. A alteração dos modelos usa colaboração assíncrona pois cada desenvolvedor trabalha na sua cópia dos modelos de forma isolada, as altera e as submete a um Sistema de Controle de Versão (SCV) otimizado. Para resolver conflitos, esta abordagem adota uma colaboração síncrona, notificando os desenvolvedores envolvidos no conflito, e permitindo que os mesmos trabalhem em paralelo no modelo que foi gerado a partir das alterações.

Cicchetti et al. [6] apresentaram um modelo com colaboração síncrona para compartilhamento do espaço de modelagem, modelos, documentações, configurações, mas assíncrona para merge de modelos que foram alterados isoladamente por cada desenvolvedor de software.

Krusche e Bruegge [14] sugeriram uma abordagem baseada na colaboração síncrona ponto a ponto (P2P) para sincronizar alterações em tempo real e evitar graves conflitos em cópias desatualizadas dos modelos.

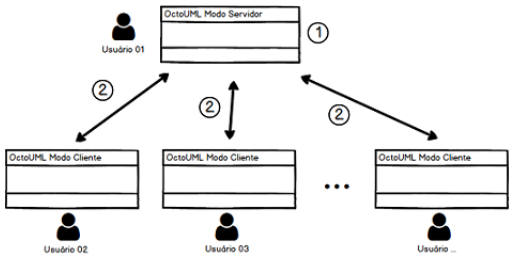

Figura 2: Ambiente da técnica UMLCollab.

Zhu et al. [23] propuseram a ferramenta Pounamu com o objetivo de prestar suporte ao desenvolvimento de DSLs (linguagem de domínio específico). Essa ferramenta suporta múltiplas visões, vários usuários, alterações dos meta-modelos, e arquitetura aberta para integração com outras ferramentas.

Em geral, a maioria dos trabalhos nessa revisão sistemática aplicaram abordagens de resolução de conflitos otimista, síncrona, ou tempo real com recursos de notificações das alterações. Permaneceram como problemas e desafios de pesquisa, a falta de escalabilidade dessas ferramentas, a resolução de conflitos indiretos, e a resolução de conflitos semânticos.

4 TÉCNICA PROPOSTA

Nesta seção é apresentada a UMLCollab, uma técnica que propõe aliar as vantagens da colaboração síncrona com a colaboração assíncrona buscando reduzir o tempo de resolução de conflitos. A estrutura desta seção foi organizada da seguinte forma: A seção 4.1 apresenta uma visão geral da técnica proposta. A seção 4.2 realiza uma análise comparativa do UMLCollab em relação aos tipos tradicionais de colaboração síncrona e assíncrona. A seção 4.3 detalha o merge automático, e o mecanismo de resolução de conflitos da técnica proposta. A seção 4.4 sumariza as principais características do UMLCollab.

4.1 Visão geral da técnica proposta

A técnica proposta UMLCollab utiliza como base a técnica OctoUML, a qual que é extendida com a finalidade de corresponder com os objetivos da técnica proposta. Os motivos da escolha dessa ferramenta são o suporte para colaboração síncrona, código aberto, suporte a notações formais (diagrama de classes e sequência da UML), e informais pois não têm regras restritivas de tipos, atributos e assinaturas de métodos, o que foi considerado como "ideal" segundo a literatura [16].

A Figura 2 apresenta uma visão geral do ambiente de utilização da técnica UMLCollab. O foco da UMLCollab é diminuir o tempo para resolução de conflitos. Futuramente, pretende-se integrar esta técnica com SCVs. No item 1 da Figura 2, o arquivo XML dos modelos é carregado pelo sistema OctoUML. Nesse momento o usuário inicia o modo servidor do OctoUML, viabilizando que outros usuários colaborem na elaboração do modelo carregado na memória do sistema. No item 2, ocorre a troca das versões do modelo entre um usuário rodando o OctoUML no modo servidor, e os demais usuários rodando no modo cliente. As alterações realizadas pelo usuário no modo servidor são enviadas pelo OctoUML a todos

RIGHTSLINK

os usuários a ele conectados. Cada usuário no modo cliente envia suas alterações para o servidor do OctoUML, e este envia aos demais usuários. Após isso, realiza o tratamento local das alterações recebidas. O detalhamento de como ocorre a sincronização nesse envio e recebimento de alterações é descrito na próxima seção.

4.2 UMLCollab e a colaboração síncrona e assíncrona

A colaboração síncrona apesar de evitar conflitos, não permite ao desenvolvedor alterar o modelo em paralelo sem interferir no trabalho de outros desenvolvedores. De fato, apesar desse tipo de colaboração notificar usuários sobre alterações em tempo real, entra em conflito com o requisito de buscar identificar se outro desenvolvedor deseja ser interrompido durante o processo de modelagem. Em um cenário com mais de 2 ou 3 usuários é provável essa característica prejudique uma interação produtiva para elaboração dos modelos.

Por outro lado, a colaboração assíncrona permite que o desenvolvedor trabalhe de forma isolada, porém perde-se os benefícios da colaboração em tempo real já que os outros desenvolvedores saberão das sobre as alterações nos modelos apenas no momento que o desenvolvedor atualiza os modelos através de um *update* ou *commit* para o sistema de controle de versão. Este tipo de colaboração garante ao usuário um ambiente de trabalho sem interferências, pois é permitido a escolha de quando se enviar as modificações realizadas ou receber alterações de outros usuários. Porém, como não há a recepção de notificações em tempo real, há maior probabilidade de ocorrência de conflitos, e utilização de mais esforço na solução dos mesmos devido a possibilidade na demora para realizar um *update* ou *commit*.

A colaboração síncrona e a colaboração assíncrona possuem juntas os requisitos ideais para modelagem colaborativa. Porém esses requisitos são atendidos de forma isolada. Por isso, o ideal é conciliar em uma mesma ferramenta as funcionalidades para notificar usuários sobre alterações em tempo real, e identificar se o desenvolvedor pode ser interrompido durante a modelagem.

A Tabela 1 apresenta uma análise comparativa dos tipos de colaboração demostrando como ocorre o envio e o recebimento das alterações entre os dois métodos.

4.3 Merge automático e Resolução de conflitos

Ao receber o modelo alterado por outros usuários a técnica proposta compara cada elemento (classe, atributos, operações e relacionamentos) com o modelo atual para identificar quais e que tipos de alterações foram realizadas. Para alterações que não conflitem, estas são imediatamente aplicadas e destacadas na cor verde para que o usuário se beneficie imediatamente do trabalho colaborativo com as alterações completas de outros usuários. Para as alterações que gerem conflitos, é feito destaque na cor vermelha e realizado o tratamento conforme Figura 3. A Figura 3 apresenta 19 passos utilizados pela técnica para realizar o tratamento de conflitos. Esses passos são descritos a seguir.

Passo 1: O sistema local recebe as alterações realizadas pelos usuários remotos a partir de suas respectivas versões do modelo base. Cada sistema local contém uma versão distinta do modelo com as alterações locais, alterações remotas feitas por merge automático, e alterações remotas em conflito com as alterações locais,

Figura 3: Merge e Resolução de Conflitos

já aprovadas, rejeitadas ou ainda aguardando avaliação. O sistema inicia a análise de cada elemento remoto em comparação ao elemento local e depois o sistema continua no passo seguinte (passo 2).

Passos 2 e 3: Se o elemento remoto seja igual ao elemento local, o elemento remoto é ignorado (passo 3) e nenhuma ação é realizada. Caso seja uma nova versão, o sistema continua no passo seguinte (passo 4).

Passo 4: O sistema verifica se o elemento é novo, isto é, não existe localmente. Caso sim, o sistema vai para o passo 5. Caso contrário, pula para o passo 15.

Passos 5, 6 e 7: Mesmo um elemento remoto novo, pode ter o mesmo nome de um elemento local existente. Nesse contexto, ambos terão identificadores distintos. Caso os nomes sejam idênticos, o sistema reconhece que se trata do mesmo elemento e iguala os identificadores (passo 7) e segue para o passo seguinte (passo 8). Caso contrário, trata-se de fato de um elemento totalmente novo. Elementos novos são considerados não conflitantes, logo é realizado um merge automático com integração do elemento ao modelo (passo 6). O sistema destaca o elemento com a cor verde e ainda registra o nome do autor, data e hora de recebimento com a finalidade de indicar ao usuário sobre o merge que foi realizado automaticamente.

Passos 8 e 9. Se o elemento local não sofreu alterações, o elemento remoto não é considerado conflitante com o elemento local. Desse modo, é realizado um merge automático com sobrescrita do elemento local com os dados do elemento remoto (passo 9). Caso haja alguma alteração local, então há algum conflito que é tratado a partir do passo 10.

Passos 10 e 11. Ainda que o usuário local não tenha aprovado ou rejeitado uma alteração remota em conflito com alteração local, é perfeitamente possível que o usuário remoto, autor da primeira alteração em outro momento, altere novamente o mesmo elemento e envie essa nova alteração. Nesse caso, o sistema deve considerar somente a última alteração, pois reflete a decisão mais recente do usuário remoto que a enviou. Nesse contexto, o conflito anteriormente registrado localmente é descartado e substituído pelo registro do novo conflito (passo 11). Devido a característica dinâmica do trabalho colaborativo, descartar conflitos antigos automaticamente é um comportamento interessante para diminuir a necessidade de

RIGHTSLINK

Tabela 1: Comparativo dos tipos de colaboração

Características	Síncrona	Assíncrona	Híbrido (UMLCollab)
Envio de alterações	Em tempo real	Manual	Manual
Recebimento de alterações	Em tempo real	Manual	Em tempo real
Observações	Sem necessidade de intervenção do usuário.	Envio manual das alterações locais para outros usuários, ocasião na qual recebe as alterações remotas.	No cenário híbrido o sistema tem comportamento assíncrono no envio de alterações, e síncrono no recebimento de alterações.

intervenção do usuário para resolver conflitos. Caso o autor da alteração remota não tenha enviado anteriormente alterações que geraram um conflito, o sistema continua a partir do passo 12.

Passos 12, 13 e 14: Enquanto um determinado elemento recebido continua pendente de avaliação, pode ocorrer que outro usuário remoto altere o mesmo elemento e envie suas alterações. Nesse contexto, ocorre o conflito da alteração local com *n* alterações recebidas de usuários remotos. Nesse caso o usuário que as recebeu deve ter a possibilidade de decidir qual alteração vai efetivar em detrimento das outras. Portanto, se para o mesmo elemento local há conflitos provenientes de usuários remotos diferentes, o sistema adiciona a nova alteração como alternativa de resolução do conflito na fila de alterações pendentes de aplicação do elemento (passo 13), caso contrário adiciona a alteração na fila de alterações pendentes de aplicação do elemento (passo 14).

Passos 15 e 16: Se o elemento remoto não é novo e o elemento local for excluído, o sistema sinaliza esse caso especial de conflito restaurando o elemento local com dados do elemento remoto (passo 16). No caso de aprovação da alteração remota, o elemento local restaurado com dados do elemento remoto é mantido. Se este elemento for rejeitado, o elemento local é novamente excluído. Se o elemento local não foi excluído, o sistema continua no passo 17.

Passo 17: Se o elemento local não foi alterado, e não possui conflitos, então é realizado o merge automático previsto no passo 9. Caso contrário, segue o passo 18.

Passos 18 e 19: Se o elemento remoto foi excluído, há outro caso especial de conflito. Se houvesse a remoção imediata do elemento local, o usuário não poderia validar se a exclusão é devida, logo nesse caso o sistema indica o conflito (passo 19). Se a alteração remota for aprovada, o elemento local é excluído, mas o elemento é mantido caso essa alteração for rejeitada. Se o elemento remoto não tenha sido excluído, o sistema continua a partir do passo 10 descrito anteriormente.

4.4 Principais características do UMLCollab

A técnica UMLCollab se concentrou nos três requisitos que são relevantes para a produtividade na modelagem colaborativa: rastrear alterações realizadas pelos desenvolvedores; notificar usuários sobre alterações; e filtrar atualizações de informações do usuário. A seguir é apresentado como esses requisitos foram abordados pelo UMLCollab.

4.4.1 Filtro de atualizações de informações para o usuário. O filtro de atualizações de informações para o usuário pode ser avaliado como o principal benefício da UMLCollab, pois como o usuário só recebe alterações consideradas completas ou suficientemente prontas de outros usuários, possibilita um menor fluxo de alterações a serem recebidas, independentemente da quantidade de usuários simultâneos atuando no modelo. Desse fluxo reduzido, aquelas alterações que não conflitam são imediatamente aplicadas e consequentemente libera o usuário da obrigação de avaliá-las, porém permitindo que as revise se desejar.

4.4.2 Notificação dos usuários de alterações. Na UMLCollab o usuário é notificado sobre todas as alterações recebidas sem que elas atrapalhem a atenção do usuário na elaboração dos modelos. Isso é realizado de forma discreta através de um esquema de cores. A cor verde é utilizada para indicar merge automático quando não ocorre conflitos, e a cor vermelha é usada para indicar conflitos devido a alterações recebidas que necessitam de aprovação ou rejeição do usuário.

4.4.3 Rastreamento de alterações realizadas pelos usuários. Todas as alterações recebidas são rastreáveis com o nome do autor, data e hora de recebimento. Todo esse histórico fica disponível tanto para alterações pendentes de avaliação com aqueles já aprovadas ou rejeitadas. A UMLCollab não suporta a integração com sistemas de controle versão, pois o foco é delimitado para aumentar a produtividade da modelagem colaborativa. A recuperação e o armazenamento do arquivo contento os modelos é realizado manualmente pelo usuário responsável pela execução do OctoUML em modo servidor.

4.4.4 Merge automático. O merge automático, conforme descrito anteriormente, visa minimizar a necessidade na interação do usuário nas mudanças recebidas. As alterações que não geram conflitos são imediatamente aplicadas. Com isso o foco se concentra nas alterações que estão sendo elaboradas, ao invés daquelas alterações enviadas pelos usuários que conflitam com suas alterações.

4.4.5 Aprovação e rejeição de alterações. A técnica UMLCollab procura maximizar a colaboração ao passo que evita o recebimento de alterações incompletas no modelo através do recebimento de alterações remotas de forma síncrona. Porém, o envio das alterações locais ocorre de forma assíncrona. Nesse cenário, o usuário acaba por acumular diversas alterações recebidas que deve aprovar ou rejeitar. Independente de existir alterações recebidas pendentes de avaliação, o usuário pode a qualquer momento enviar suas alterações juntamente com as dos demais usuários já aprovados. Esse sistema de merge automático, com resolução manual de conflitos, e

RIGHTSLINK

com aprovação e rejeição de alterações pode ser mais eficiente que as abordagens clássicas de colaboração síncrona e assíncrona.

5 AVALIAÇÃO

A avaliação da técnica proposta visa comparar a UMLCollab com a colaboração síncrona, e a colaboração assíncrona. Além da UMLCollab, que representa a técnica de colaboração híbrida, a colaboração síncrona foi representada por uma implementação original do software OctoUML [20]. Para a colaboração assíncrona foi adotado o software IBM Rational Software Architect Designer (IBM RSAD) [12]. O experimento consiste em um conjunto de atividades que os participantes devem realizar para alterar um modelo base (M_B) composto de um Diagrama de Classes da UML, em um modelo desejado (M_D). Dessa forma, seguindo o modelo GQM (Goal, Questions, Metrics) [3] o objetivo do experimento foi organizado como a seguir:

Analisar *as técnicas de modelagem colaborativa e resolução de conflitos de modelos de software*
com o propósito de *investigar seus efeitos*
em respeito ao *esforço*
a partir da *perspectiva de desenvolvedores*
no contexto de *evolução de modelos de software.*

Para esse objetivo foram elaboradas atividades para evidenciar situações de conflito que podem ocorrer com a concorrência na alteração dos modelos. Nesse experimento, busca-se medir a produtividade dos desenvolvedores usando os diferentes tipos de colaboração em comparação a técnica proposta UMLCollab. Com isso, esse experimento busca responder a seguinte questão de pesquisa: "Qual o tipo de técnica de sincronização e resolução de conflitos exige menor esforço do desenvolvedor?"

Partindo do pressuposto que a colaboração síncrona prejudica o processo cognitivo com o aumento dos desenvolvedores atuando no mesmo modelo simultaneamente, e a assíncrona sofre com o problema dos conflitos, esperamos que a colaboração híbrida da técnica UMLCollab traga mais produtividade, ou reduza o esforço na resolução de conflitos. Dessa forma as seguintes hipóteses foram elaboradas:

- **Hipótese nula 1** (H_{1-0}): O esforço relativo na colaboração híbrida com o aumento dos usuários é igual ou maior que os esforços da concorrência síncrona, e concorrência assíncrona.
- **Hipótese alternativa 1** (H_{1-1}): O esforço relativo na colaboração híbrida com o aumento dos usuários é menor que os esforços da concorrência síncrona, e concorrência assíncrona.

As variáveis independentes são o tamanho dos modelos, lista de atividades, o tempo em segundos para execução das mudanças em cada modelo, detecção e resolução de conflitos, e as notas a partir do questionário da pesquisa. As variáveis dependentes que serão medidas no experimento são as técnicas de modelagem colaborativas síncrona, assíncrona, e a técnica híbrida (UMLCollab).

Neste experimento foram realizados pré-testes e testes com a participação dos alunos de graduação em informática e da pós-graduação stricto sensu (mestrados e doutorados) do curso de computação aplicada da Universidade do Vale do Rio dos Sinos (UNISINOS).

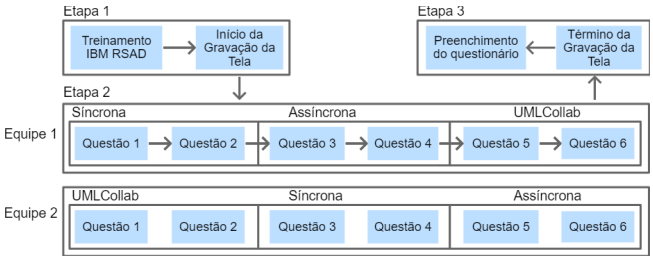

Figura 4: Desenho do experimento

Para isso, foram criados 6 cenários, onde cada participante executou 3 atividades. Para esse experimento inicial, participaram 6 estudantes de graduação e pós-graduação em informática, divididos em 2 equipes de 3 participantes cada, que foram convidados dentro dos ambientes laboratoriais de pesquisa de informática. Para melhorar os resultados do experimento futuramente, pretende-se convidar mais participantes para o experimento.

5.1 Desenho do Experimento

Conforme a Figura 4, o experimento foi organizado em três etapas: a primeira etapa consiste no treinamento sobre as funcionalidades das ferramentas utilizadas no experimento, e o início da gravação do experimento para documentar e validar o experimento. Na etapa seguinte é iniciado o experimento das técnicas de colaboração, com a realização de atividades para evidenciar conflitos distribuídas em 6 questões. Na etapa 3, o experimento é finalizado logo após os usuários responderam um questionário de informações acadêmicas e profissionais.

O questionário do experimento foi elaborado para registrar o número sequencial do experimento, data e hora de realização, como também dados referentes ao perfil acadêmico e profissional dos participantes. A cada 2 questões a técnica era trocada, iniciando pela modelagem colaborativa síncrona, depois a modelagem colaborativa assíncrona, e por último a técnica de modelagem colaborativa híbrida UMLCollab era utilizada.

Além dos dados básicos dos participantes, para cada etapa do experimento usando a colaboração síncrona, assíncrona, e híbrida foi solicitado uma nota de 1 a 5 para a produtividade percebida pelos participantes, sendo a nota 1 a de menor produtividade e 5 a maior produtividade. Em busca das impressões dos usuários com cada ferramenta, incluímos também para cada tipo de colaboração, um campo de texto livre para que o participante justifique sua nota. Também foram perguntados aos participantes questões em relação a aceitabilidade de tecnologia. Eles responderam questões sobre percepção de facilidade de uso, percepção de utilidade, atitude e intenção de comportamento em relação a tecnologia proposta.

6 RESULTADOS

Para cada método (colaboração síncrona, assíncrona, e UMLCollab) foram elaborados 2 questões, cada uma com um modelo de classes específico, e 3 atividades que geraram conflitos. Para cada questão, cada participante tinha um atributo ou método que era seu objetivo manter na versão final do modelo após a resolução de conflitos.

RIGHTSLINK

Tabela 2: Tempo médio de resolução das questões do experimento por tipo de colaboração

Tipo de colaboração	Média em minutos
Colaboração assíncrona	09:28
Colaboração síncrona	02:28
Técnica UMLCollab	05:17

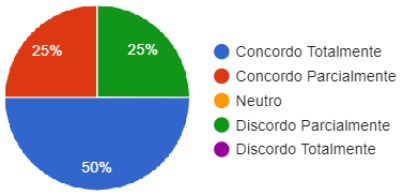

Figura 5: A técnica UMLCollab ajudaria aumentar a produtividade

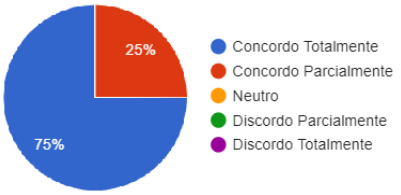

Figura 6: A técnica UMLCollab proporcionaria ao desenvolvedor estar ciente das alterações de outros colaboradores sem atrapalhar seu trabalho de modelagem

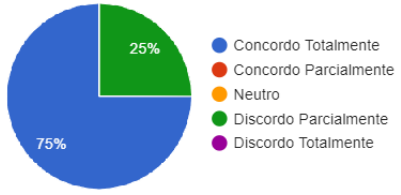

Figura 7: A técnica UMLCollab facilitaria a resolução de conflitos.

Para avaliar a produtividade, foram coletados 3 métricas distintas: (1) o tempo em que cada equipe levou para terminar cada questão, o qual foi medido do início de cada questão até que cada participante atingisse seu objetivo; (2) a percepção da produtividade, e (3) uma nota para a produtividade de cada tipo de colaboração em relação as demais.

Conforme a Tabela 2, em uma amostra de duas equipes, foi possível verificar o tempo médio de cada questão de 09:28 (nove minutos e vinte e oito segundos) para colaboração assíncrona. Na colaboração síncrona o tempo médio foi reduzido para 02:28 (dois minutos e 28 segundos), isto é, uma redução de 73,93%. Essa redução era esperada, pois, na colaboração síncrona o desenvolvedor não perde tempo enviando e recebendo alterações manualmente, e também não houveram conflitos significantes na elaboração dos modelos. A técnica UMLCollab, com envio manual e recebimento automático, apresentou uma média de 5:17 (cinco minutos e dezessete segundos), ou seja, uma redução significativa de 44,29% em relação a colaboração assíncrona, porém ainda 53,21% maior que a colaboração síncrona.

Tabela 3: Corretude

Tipo de colaboração	Corretude
Colaboração assíncrona	0,94
Colaboração síncrona	0,91
Técnica UMLCollab	0,96

Apesar de sua produtividade intermediária, a técnica UMLCollab tem a vantagem em relação a colaboração síncrona de não interromper o desenvolvedor quando há alguém alterando o mesmo elemento, como um atributo. Pois na técnica síncrona, dois colaboradores não conseguem efetivamente alterar o mesmo elemento sem atrapalhar o que o primeiro está tentando elaborar. Complementando essa avaliação, o questionário na Figura 5 apresenta que 50% dos participantes concordaram totalmente que a técnica UMLCollab ajudaria aumentar a produtividade, isto é, 25% concordaram totalmente e 25% concordaram parcialmente. Quando requisitados a fornecer uma nota de 1 a 5 para cada tipo de colaboração, a colaboração assíncrona obteve a média 2,67, e a colaboração síncrona uma nota média de 4,33. Nesse contexto, era de se esperar uma nota intermediária para a técnica UMLCollab, porém a média obtida pela UMLCollab foi de 4,50, superando as técnicas tradicionais. Essa sensação de maior produtividade se deve ao menor nível de interrupção oferecido pela técnica UMLCollab.

Para corroborar esse resultado, conforme Figura 7, 75% dos participantes concordaram totalmente que a UMLCollab facilitaria a resolução de conflitos contra 25% que discordaram parcialmente. A Figura 6 apresenta que 75% dos participantes concordaram totalmente, e 25% concordaram parcialmente no que diz respeito a técnica UMLCollab proporcionar ao desenvolvedor uma notificação apropriada das alterações de outros colaboradores sem atrapalhar a modelagem.

Para medir a corretude do modelo alterado pelos participantes em relação ao modelo esperado foi elaborado uma pontuação máxima para cada questão, sendo 4 pontos para operações (visibilidade, nome, argumentos e tipo de retorno) e 3 pontos para atributos (visibilidade, nome e tipo). Com isso por exemplo, um modelo com classes contendo dois atributos e 8 operações havia no máximo 38 pontos. Cada subelemento (visibilidade, nome, etc) errado implicou na subtração de um ponto da nota máxima. Cada atributo duplicado ou removido implicou na penalização de 3 pontos, e cada operação duplicada ou removida implicou na subtração de 4 pontos. Usando essa metodologia, conforme Tabela 3, constatou-se que a colaboração assíncrona obteve uma média de 94% de corretude no modelo alterado. Apesar de apresentar uma maior produtividade, o modelo síncrono conseguiu uma média menor de 91% de corretude. A técnica UMLCollab atingiu a média de 96% de corretude no modelo alterado.

RIGHTSLINK

7 CONCLUSÃO

A modelagem de software colaborativa teoricamente permite aumento da produtividade e qualidade dos modelos através do trabalho simultâneo de dois ou mais desenvolvedores em um mesmo modelo. As técnicas colaborativas atuais apresentam problemas crescentes em complexidade à medida que o número de desenvolvedores aumenta. A colaboração síncrona, na qual as alterações locais são imediatamente enviadas e as alterações remotas imediatamente recebidas, evita conflitos, porém, podem atrapalhar o processo cognitivo de elaboração dos modelos, pois quanto maior o número de usuários simultâneos maior quantidade de alterações remotas. Em contrapartida, a colaboração assíncrona evita esses problemas, porém os conflitos são mais complexos e demorados para serem resolvidos.

Para tratar dessa problemática, foi proposta a técnica UMLCollab. Nesta técnica as alterações locais são enviadas manualmente pelo usuário e as alterações remotas imediatamente recebidas, permitindo um sistema combinado de merge automático e manual visando minimizar o tempo de resolução de conflitos. A avaliação foi realizada através de um experimento, para medir o ganho de produtividade da técnica UMLCollab em comparação as técnicas tradicionais de colaboração síncrona e assíncrona, e ainda a percepção dos participantes em relação ao uso da técnica proposta.

Os resultados demostraram uma produtividade intermediária da técnica proposta, porém a corretude dos modelos gerados foi superior as técnicas tradicionais. Além disso, constatou-se uma percepção positiva da técnica proposta em relação a resolução dos conflitos. A UMLCollab também foi avaliada positivamente por não proporcionar o usuário estar ciente das alterações de outros usuários sem atrapalhar seu trabalho no desenvolvimento de modelos de software.

Os trabalhos futuros de pesquisa são a integração com os atuais sistemas de controle de versão para viabilizar um gerenciamento melhor das versões de modelos de software, e a integração com sistemas de comunicação por chat e videoconferência, para viabilizar a interação remota entre colaboradores.

ACKNOWLEDGMENTS

Os autores agradecem à Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e à Universidade do Vale do Rio dos Sinos (Unisinos) pelo apoio ao desenvolvimento desse trabalho. Os autores reconhecem especialmente o apoio do Programa de Pós-Graduação em Computação Aplicada (PPGCA) da Unisinos.

REFERENCES

[1] Kari Alho and Reijo Sulonen. 1998. Supporting virtual software projects on the Web. In *Seventh IEEE International Workshop on Enabling Technologies: Infrastucture for Collaborative Enterprises (WET ICE'98)*. 10–14. https://doi.org/10.1109/ENABL.1998.725660

[2] Kerstin Altmanninger, Martina Seidl, and Manuel Wimmer. 2009. A survey on model versioning approaches. *International Journal of Web Information Systems* 5, 3 (2009), 271–304.

[3] V Basili, G Caldiera, and DH Rombach. [n. d.]. The Goal Question Metric Paradigm: Encyclopedia of software engineering. 1994.

[4] M Brambilla, J Cabot, and M Wimmer. 2012. Model-driven software engineering in practice. *Synthesis Lectures on* (2012).

[5] P Brosch, M Seidl, K Wieland, M Wimmer, and P Langer. 2009. We can work it out: Collaborative conflict resolution in model versioning. *ECSCW 2009* (2009).

[6] Antonio Cicchetti, Henry Muccini, Patrizio Pelliccione, and Alfonso Pierantonio. 2009. Towards a framework for distributed and collaborative modeling. In *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*. IEEE, 149–154.

[7] Catarina Costa and Leonardo Murta. 2013. Version Control in Distributed Software Development: A Systematic Mapping Study. *Proceedings of IEEE 8th International Conference on Global Software Engineering - ICGSE'13* (2013), 90–99. https://doi.org/10.1109/ICGSE.2013.19

[8] Hoa Khanh Dam and Aditya Ghose. 2011. An agent-based framework for distributed collaborative model evolution. (2011), 121. https://doi.org/10.1145/2024445.2024468

[9] A. De Lucia, F. Fasano, G. Scanniello, and G. Tortora. 2007. Enhancing collaborative synchronous UML modelling with fine-grained versioning of software artefacts. *Journal of Visual Languages and Computing* 18, 5 (2007), 492–503. https://doi.org/10.1016/j.jvlc.2007.08.005

[10] Guilherme Ermel, Kleinner Farias, Lucian José Gonçales, and Vinicius Bischoff. 2018. Supporting the Composition of UML Component Diagrams. In *Proceedings of the XIV Brazilian Symposium on Information Systems*. ACM, 56.

[11] Vinícius Soares Fonseca, Monalessa Perini Barcellos, and Ricardo de Almeida Falbo. 2016. Tools integration for supporting software measurement: a systematic literature review. *iSys-Revista Brasileira de Sistemas de Informação* 8, 4 (2016), 80–108.

[12] IBM. 2019. IBM Rational Software Architect Designer. https://www.ibm.com/developerworks/downloads/r/architect/index.html, acesso em 13 de Maio 2019.

[13] Dimitrios S Kolovos, Louis M Rose, Nicholas Matragkas, Richard F Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, Massimo Tisi, and Others. 2013. A research roadmap towards achieving scalability in model driven engineering. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*. ACM, 2.

[14] Stephan Krusche and Bernd Bruegge. 2014. Model-based real-time synchronization. In *International Workshop on Comparison and Versioning of Software Models (CVSM14)*.

[15] Dilshodbek Kuryazov and Andreas Winter. 2015. Collaborative Modeling Empowered By Modeling Deltas. In *3rd International Workshop on (Document) Changes: modeling, detection, storage and visualization - DChanges 2015 (DChanges 2015)*. 1–6. https://doi.org/10.1145/2881631.2881633

[16] Haidar Osman. 2013. Web-Based Collaborative Software Modeling. (2013).

[17] Gregor Polančič and Gregor Jošt. 2016. The impact of the representatives of three types of process modeling tools on modeler's perceptions and performance. *Journal of Software: Evolution and Process* 28, 1 (2016), 27–56. https://doi.org/10.1002/smr.1764

[18] Javier Portillo-Rodríguez, Aurora Vizcaíno, Mario Piattini, and Sarah Beecham. 2012. Tools used in Global Software Engineering: A systematic mapping review. *Information and Software Technology* 54, 7 (2012), 663–685.

[19] Anita Sarma and Andre Van Der Hoek. 2006. Towards awareness in the large. In *IEEE International Conference on Global Software Engineering (ICGSE'06)*. IEEE, 127–131.

[20] Boban Vesin, Rodi Jolak, and Michel R V Chaudron. 2017. OctoUML: an environment for exploratory and collaborative software design. In *39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE Press, 7–10.

[21] J young Bang and D Popescu. 2012. Enabling Workspace Awareness for Collaborative Software Modeling. (2012).

[22] Holt Zaugg, Richard E West, Isaku Tateishi, and Daniel L Randall. 2011. Mendeley: Creating communities of scholarly inquiry through research collaboration. *TechTrends* 55, 1 (2011), 32–36.

[23] Nianping Zhu, John Grundy, John Hosking, Na Liu, Shuping Cao, and Akhil Mehra. 2007. Pounamu: A meta-tool for exploratory domain-specific visual language tool development. *Journal of Systems and Software* 80, 8 (2007), 1390–1407.

RIGHTSLINK