

RAFAEL DE SOUZA SANTOS

AVALIAÇÃO DO ESFORÇO DE JUNÇÃO DE RAMOS EM SISTEMAS DE
CONTROLE DE VERSÃO

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Engenharia de Software.

Orientador: Prof. Dr. Leonardo Gresta Paulino Murta

Niterói

2012

RAFAEL DE SOUZA SANTOS

AVALIAÇÃO DO ESFORÇO DE JUNÇÃO DE RAMOS EM SISTEMAS DE
CONTROLE DE VERSÃO

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Engenharia de Software.

Aprovada em Agosto de 2012.

BANCA EXAMINADORA

Prof. Dr. Leonardo Gresta Paulino Murta – Orientador

UFF

Prof. Dr. Debora Christina Muchaluat Saade

UFF

Prof. Dr. Marco Aurélio Gerosa

USP

Niterói

2012

À minha avó Maria de Lourdes Figueiredo Antão (*in memoriam*).

AGRADECIMENTOS

A Deus, que nos momentos de desânimo, não me deixou desistir e me deu forças para enfrentar todas as adversidades.

Ao meu orientador Leonardo Murta, que sempre me incentivou e viabilizou que eu concretizasse o mestrado.

Aos meus colegas de mestrado Heliomar Kann, Gleiph Ghioto e Daniel Castellani que sempre foram solícitos em ajudar no desenvolvimento do Polvo.

A todos os docentes que tive o prazer de conhecer no Instituto de Computação da UFF, desde a época da graduação, por me construírem profissionalmente e pessoalmente.

À minha esposa Lidiane, por me escutar e apoiar durante as adversidades e apoios nas revisões da dissertação.

À minha família, por acreditar que eu seria capaz, em especial minha mãe, Maria José, pois sem ela eu não teria a base para chegar até aqui.

“Quando tudo nos parece dar errado,
acontecem coisas boas que não teriam
acontecido se tudo tivesse dado certo!”

Renato Russo.

RESUMO

É comum, no desenvolvimento de software sob controle de versão, a necessidade da evolução em paralelo do código via ramos. Além disso, na maioria das vezes, a junção desses ramos se faz necessária. Desta forma, este trabalho propõe a extração de métricas que estimem o esforço para a realização dessa junção, onde é possível visualizar, dentre todos os ramos, quais são os mais críticos e acompanhar a evolução das métricas desde a criação do ramo. A avaliação deste trabalho mostrou que algumas métricas se comportaram melhor na estimativa do esforço de junção de ramos, sendo que a métrica de Quantidade de Conflitos Físicos chegou a uma correlação de até 99% com o esforço real de junção.

Palavras-chave: Ramos; Controle de Versão; Métricas; Visualização

ABSTRACT

It is common, in software development under version control, the need of parallel development of the source code via branches. Moreover, in most cases, the merge of these branches is necessary. Therefore, our work proposes the extraction of metrics that estimate the effort to perform merges, making it possible to visualize, among all branches, which are the most critical and analyze the evolution of the metrics since the establishment of the branch. The evaluation of our work showed that some metrics behave better in order to estimate the effort of integrating branches. For instance, the metric Number of Physical Conflicts reached up to 99% correlation when compared to the actual merge effort.

Keywords: Branches; Version Control; Metrics; Visualization

LISTA DE ILUSTRAÇÕES

Figura 1: Exemplo de cenário de desenvolvimento de software.	15
Figura 2: Integração das mudanças entre os desenvolvedores.	20
Figura 3: Modelo check-out–edição–check-in.	21
Figura 4: Exemplo do versionamento do arquivo prog.java.	21
Figura 5: Sistema de Controle de Versão Centralizado.	22
Figura 6: Sistema de Controle de Versão Distribuído.	23
Figura 7: Estrutura do repositório.	24
Figura 8: Delta entre versões.	24
Figura 9: Criação de ramo do arquivo prog.java.	25
Figura 10: Junção do ramo do arquivo prog.java.	27
Figura 11: Exemplo de conflito físico.	28
Figura 12: Execução do diff3.	28
Figura 13: Estratégia de Ramificação Em Série.	30
Figura 14: Estratégia de Ramificação Em Cascata.	30
Figura 15: Estratégia de Ramificação Por Componentes.	31
Figura 16: Estratégia de Ramificação Por Desenvolvedor.	31
Figura 17: Estratégia de Ramificação Por Requisições.	32
Figura 18: Estratégia de Ramificação – Por Subprojetos.	32
Figura 19: Estratégia de Ramificação – Por Personalizações.	33
Figura 20: Estratégia de Ramificação Por Terceirização.	33
Figura 21: Questões da abordagem.	37
Figura 22: Abordagem Polvo.	38
Figura 23: Abordagem Polvo não intrusiva.	39
Figura 24: Exemplo da métrica Quantidade de Artefatos Modificados.	41
Figura 25: Exemplo da métrica Quantidade de Artefatos Modificados em Comum.	42
Figura 26: Exemplo de resultados de uma busca realizada na internet de um determinado assunto.	43
Figura 27: Exemplo da métrica Cobertura/Precisão por Quantidade de Artefatos.	44
Figura 28: Exemplo da métrica Quantidade de Linhas Diferentes.	45
Figura 29: Extração de métricas com análise da junção dos ramos.	46
Figura 30: Diagrama de classe para exemplificar as métricas (MENEZES, 2011).	46

Figura 31: Conteúdo dos arquivos da versão da linha principal antes da criação do ramo. Adaptada de Menezes (2011).	47
Figura 32: Exemplo de conflito sintático (versão na linha principal).....	47
Figura 33: Exemplo de conflito sintático (versão no ramo).....	48
Figura 34: Exemplo conflito semântico (versão na linha principal).....	48
Figura 35: Exemplo conflito semântico (versão no ramo) (MENEZES, 2011).....	49
Figura 36: Estratégias de Ramificação utilizadas no Polvo.	50
Figura 37: Página inicial do Oceano.....	53
Figura 38: Diagrama das classes persistentes resumido do Oceano e do Polvo.....	54
Figura 39: Diagrama de atividades do Polvo.	56
Figura 40: Diagrama de Atividades – Visão Específica de um Ramo.....	57
Figura 41: Exemplo Visão Específica do Ramo.....	57
Figura 42: Visão Detalhada de um Ramo.	58
Figura 43: Visão Temporal do Ramo.....	59
Figura 44: Diagrama de Atividades – Visão Geral do Projeto.....	60
Figura 45: Configuração dos Ramos.....	61
Figura 46: Classificação da criticidade dos ramos.....	61
Figura 47: Visão Geral dos Ramos do projeto Apache HTTPD.	62
Figura 48: Visão Geral dos Ramos do projeto Subversion.	63
Figura 49: Versões consideradas no Esforço de Junção (EJ).....	69
Figura 50: Conjunto de ações realizadas e efetivadas (PRUDÊNCIO <i>et al.</i> , 2012).....	71

LISTA DE TABELAS

Tabela 1: Métricas utilizadas no Polvo.	40
Tabela 2: Tabela verdade da ocorrência de conflitos semânticos no código e nas classes de teste.	49
Tabela 3: Caracterização dos Projetos.	66
Tabela 4: Caracterização dos Ramos.	67
Tabela 5: Versões Utilizadas dos Ramos.	70
Tabela 6: Resultado das métricas e do Esforço de Junção (EJ) para os ramos.	72
Tabela 7: Correlação das Métricas Quantidade de Artefatos Modificados na Linha Principal (M1) e no Ramo (M2).	73
Tabela 8: Correlação das Métricas Quantidade de Linhas Diferentes na Linha Principal (M3) e no ramo (M4).	74
Tabela 9: Correlação das Métricas Cobertura por Quantidade de Artefatos (M5) e Precisão por Quantidade de Artefatos (M6).	75
Tabela 10: Correlação das Métricas Quantidade de Artefatos Modificados em Comum (M7) e de Conflitos Físicos (M8).	75
Tabela 11: Projetos analisados mas excluídos da avaliação.	88

LISTA DE ABREVIATURAS E SIGLAS

AE – Ações Efetivadas

AR – Ações Realizadas

CORREL. – Correlação

CE – Critério de Exclusão

EJ – Esforço de Junção

GCS – Gerência de Configuração de Software

GEMS – Grupo de Evolução e Manutenção de Software

IC – Item de Configuração

IDE – *Integrated Development Environment*

LI – Limite Inferior

LS – Limite Superior

JPA – *Java Persistence API*

JSF – *Java Server Faces*

REC – Resultados Recuperados

REL – Resultados Relevantes

SCV – Sistema de Controle de Versão

VB – Versão Base

VJ – Versão após a Junção

VL – Versão da Linha principal

VR – Versão do Ramo

SUMÁRIO

Capítulo 1 – Introdução.....	14
1.1 Motivação	14
1.2 Objetivo	15
1.3 Organização	16
Capítulo 2 – Ramificação em Sistemas de Controle de Versões.....	18
2.1 Introdução	18
2.2 Sistema de Controle de Versão	19
2.2.1 Conceitos principais de SCV	20
2.2.2 Topologias.....	22
2.2.3 Repositório	23
2.3 Ramos	24
2.4 Junção de ramos	26
2.5 Estratégias de ramificação	29
2.5.1 Estratégias para manutenção de software	29
2.5.2 Estratégias para isolamento de equipes	30
2.5.3 Estratégias para adaptação de software	32
2.6 Trabalhos relacionados	34
2.7 Considerações finais.....	36
Capítulo 3 – Visualização da complexidade de junção de ramos.....	37
3.1 Introdução	37
3.2 Visão geral	38
3.3 Métricas	40
3.3.1 Métricas com análise das diferenças	41
3.3.2 Métricas com análise de conflitos	45
3.4 Estratégia de Ramificação	49
3.5 Considerações finais.....	50

Capítulo 4 – Implementação e utilização do Polvo	52
4.1 Introdução	52
4.2 Visão geral do Oceano e Polvo	53
4.3 Tecnologias utilizadas na implementação	55
4.4 Utilização do Polvo	56
4.4.1 Visão Específica de um Ramo	56
4.4.2 Visão Geral do Projeto.....	60
4.5 Considerações finais.....	63
Capítulo 5 – Avaliação experimental do Polvo	64
5.1 Introdução	64
5.2 Projetos e seus ramos utilizados.....	64
5.3 Método utilizado	68
5.4 Aplicação do método.....	71
5.5 Análise dos resultados	73
5.6 Ameaças à validade	76
5.7 Considerações finais.....	77
Capítulo 6 – Conclusão	79
6.1 Contribuições	79
6.2 Limitações.....	79
6.3 Trabalhos futuros	80
Referências	83
Apêndice A – Projetos avaliados.....	88

CAPÍTULO 1 – INTRODUÇÃO

1.1 MOTIVAÇÃO

A Gerência de Configuração de Software (GCS) é uma subárea da Engenharia de Software que tem como principal objetivo possibilitar a evolução controlada do software (DART, 1991). O desenvolvimento de software, de modo geral, envolve um grupo de desenvolvedores cujo tamanho varia conforme a necessidade do projeto. Nesse cenário, os desenvolvedores trabalham em partes do código em paralelo, levando então à necessidade de controlar a evolução desse código. Por exemplo, esse controle é necessário para saber quem fez determinada alteração que apresentou erro numa funcionalidade, quando essa alteração foi feita e o porquê dela ter sido feita. Como parte da GCS que visa atender essa necessidade, surgem os Sistemas de Controle de Versão (SCV), cuja principal característica é permitir que desenvolvedores colaborem de forma controlada, armazenando o histórico do desenvolvimento do software (ESTUBLIER, 2000).

Dewan e Hegde (2007) afirmam que software complexos devem ser desenvolvidos colaborativamente. Em consequência disto, há a necessidade de criação de ramos (do inglês, *branches*), ou seja, versões que não seguem a linha principal de desenvolvimento (WALRAD; STROM, 2002). Através dos ramos é possível isolar o desenvolvimento de um ou mais desenvolvedores, desenvolver determinada funcionalidade separada das demais, personalizar um software para determinado cliente, separar a manutenção corretiva da evolutiva, dentre outras possibilidades.

O ramo evolui em paralelo à linha principal de desenvolvimento, que também é alvo de modificações. Na maioria das vezes, é realizada a junção desses ramos com a linha principal de desenvolvimento ou vice-versa, porém essa junção pode ser custosa com o passar do tempo. Isso acontece devido às edições no código tanto nos ramos quanto na linha principal de desenvolvimento, aumentando a chance de haver um maior número de conflitos durante a junção e inviabilizando a sua execução automática. Desta forma, saber um momento apropriado para efetuar a junção é de grande importância para manter a produtividade da equipe e a qualidade do produto.

Um cenário que motivaria a adoção da abordagem proposta neste trabalho consiste em uma equipe de desenvolvimento de um projeto de software dividida, onde uma parte da equipe trabalha na linha principal, desenvolvendo novas funcionalidades, e outra parte da equipe trabalha em um ramo auxiliar, corrigindo defeitos. Em algum momento, haverá a

necessidade da realização da junção desse ramo, por exemplo, para a liberação (do inglês, *release*) de novas funcionalidades para produção ou para a aplicação da correção de um defeito na linha principal de desenvolvimento. Porém, não há uma estimativa do momento apropriado para que seja feita essa junção em função do esforço que irá demandar. A realização precoce de junções pode provocar uma junção inadequada entre modificações corretivas e evolutivas. Por outro lado, a realização tardia de junções pode ser contraproducente e suscetível a erros. A Figura 1 ilustra este cenário.

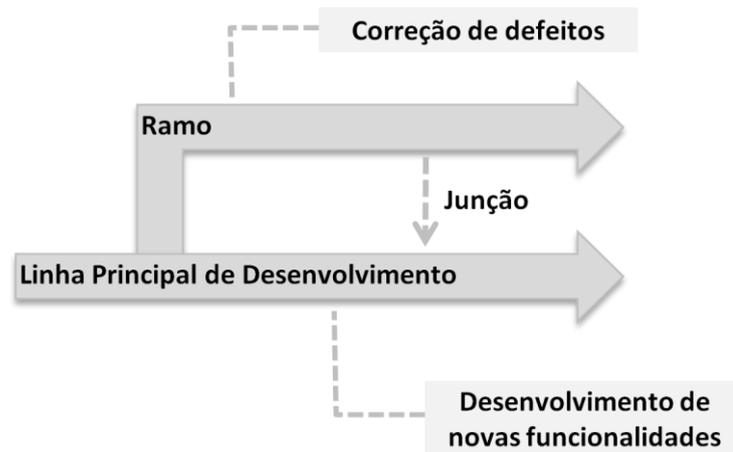


Figura 1: Exemplo de cenário de desenvolvimento de software.

Considerando agora o cenário descrito anteriormente, mas com um projeto de software de maior porte, onde estão envolvidas várias equipes de desenvolvimento trabalhando em muitos ramos diferentes, o problema fica ainda mais desafiador. Estes ramos podem estar sendo evoluídos há muito tempo, sem as equipes terem noção de o quanto estão divergindo do ramo do qual eles foram originados.

Durante o desenvolvimento deste trabalho, foi realizada uma revisão da literatura em busca de trabalhos que lidassem com a avaliação de ramos. Contudo, não foi possível identificar outros trabalhos nessa linha. Os trabalhos mais próximos tratam de comparação e percepção (do inglês, *awareness*) entre versões de software e espaços de trabalho (do inglês, *workspace*). Uns focam na identificação e significado de diferenças entre versões do software, no âmbito físico, sintático e semântico; outros focam em passar o conhecimento do impacto das alterações que estão ocorrendo em paralelo para o desenvolvedor, antes que elas sejam enviadas para o repositório.

1.2 OBJETIVO

Dado o exposto, este trabalho tem como objetivo prever o esforço para se fazer a junção de ramos, visando responder à seguinte questão de pesquisa: É viável prever com

precisão o esforço de junção de ramos em SCV a partir de análise automática sobre o repositório?

Para isso, foi elaborada uma abordagem, denominada Polvo¹, que extrai métricas que permitem avaliar continuamente o esforço de junção de ramos. Os dados extraídos pela abordagem servem de apoio gerencial para a estimativa de tempo e esforço para realização da junção. Essa informação pode ser utilizada para identificação do momento apropriado para a junção ou mesmo o *spin-off* dos ramos. A sugestão de *spin-off* ocorre quando é possível chegar à conclusão de que não vale a pena o esforço de se fazer a junção devido à complexidade. Com isso, o ramo ganha vida própria e passa a ser desenvolvido independentemente da linha principal de desenvolvimento.

Como apoio à pesquisa, foi desenvolvido um protótipo para extração dessas métricas, onde é possível visualizar todos os ramos e de onde cada um se originou. Além disso, são destacados aqueles ramos que possuem mais criticidade em relação ao esforço de junção e é possível acompanhar visualmente a evolução da métrica com o passar do tempo, desde a criação do ramo.

A abordagem Polvo foi avaliada sobre quatro projetos que possuem características distintas. Essas avaliações foram executadas em projetos reais que já tiveram junção de ramos, onde foi feito o cálculo do esforço real de junção e comparado com os resultados das métricas extraídas pela abordagem na versão do repositório anterior à realização da junção. Com a análise do resultado, foi possível identificar que algumas métricas se comportaram melhor no objetivo proposto, sendo que a métrica de Quantidade de Conflitos Físicos chegou a uma correlação de até 99% com o esforço real de junção.

1.3 ORGANIZAÇÃO

O restante deste trabalho está organizado em cinco capítulos, além do capítulo de introdução. O Capítulo 2 apresenta uma visão geral de ramificação em SCV. Neste capítulo são discutidos os principais conceitos de SCV, com ênfase em ramificação e junção, por serem mais relevantes para este trabalho. São detalhadas também algumas estratégias de ramificação utilizadas pela abordagem, além de outros trabalhos existentes com abordagens

¹ Os polvos são moluscos marinhos da classe *Cephalopoda* e da ordem *Octopoda*, que significa "oito pés". Possuem oito braços com fortes ventosas dispostas à volta da boca. Como meios de defesa, o polvo possui a capacidade de largar tinta, camuflagem (conseguida através dos cromatóforos) e autonomia de seus braços (WIKIPÉDIA, 2011). O nome dado à abordagem proposta neste trabalho foi Polvo pela semelhança dos seus braços com os ramos de um projeto.

similares que focam na comparação, combinação e na percepção entre diferentes versões do software.

No Capítulo 3 a abordagem Polvo é descrita, ressaltando as opções que o desenvolvedor tem no uso da abordagem e no que ela auxilia o desenvolvedor a visualizar o esforço da junção entre os ramos. Além disso, é apresentada a visão geral da abordagem, assim como as métricas utilizadas e como as estratégias de ramificação influenciam no seu uso.

No Capítulo 4 são descritas as características de implementação do Polvo e da infraestrutura que o suporta. Neste capítulo estão detalhadas as formas de cálculo do esforço da junção, os módulos que foram implementados, as interfaces disponibilizadas para interação do desenvolvedor com a ferramenta, juntamente com exemplos de sua utilização.

No Capítulo 5 são mostrados os resultados da avaliação do Polvo e o processo utilizado para a obtenção de tais resultados. Esse capítulo descreve os projetos utilizados na avaliação e quais foram os resultados da extração de indicadores que avaliam a precisão do Polvo para prever o esforço da junção de ramos, comparando-os com o esforço real de junção. Também são discutidas as ameaças à validade da avaliação.

Finalmente, o Capítulo 6 conclui esta dissertação, resumindo as contribuições da abordagem Polvo, discutindo algumas limitações deste trabalho e apresentando possíveis trabalhos futuros.

CAPÍTULO 2 – RAMIFICAÇÃO EM SISTEMAS DE CONTROLE DE VERSÕES

2.1 INTRODUÇÃO

De acordo com Estublier (2000), a GCS surgiu como uma disciplina logo após a “Crise do Software”, ocorrida no final dos anos 60 e início dos anos 70, em que foi entendido que a programação não cobria tudo da Engenharia de Software. Depois, o seu foco variou no decorrer do tempo, passando por: desenvolvimento de software de grande porte no início dos anos 80, tratando versionamento e construção (do inglês, *build*); desenvolvimento com equipes maiores na década de 90, tratando o controle de processos e a concorrência; e acesso remoto via web no final dos anos 90, para software global. Atualmente, um sistema de GCS típico oferece serviços na gestão do repositório, onde as versões dos artefatos são armazenadas, nas atividades habituais dos desenvolvedores e no apoio e controle do processo. Os sistemas de GCS usualmente evitam ser dependentes de linguagens de programação específicas e de aspectos semânticos do software sob GCS.

Olhando sob o foco gerencial, a GCS é dividida em cinco funções, que são (IEEE, 2005): identificação da configuração; controle da configuração; contabilização da situação da configuração; avaliação e revisão da configuração; e gerenciamento de liberação e entrega. A identificação de configuração foca na seleção, definição do esquema de nomes e números e descrição dos itens de configurações (IC's). O IC representa a agregação de hardware, software ou ambos, tratada pela GCS como um elemento único (IEEE, 1990). O controle da configuração faz o acompanhamento da evolução dos IC's selecionados e descritos pela função de identificação. A contabilização da situação da configuração armazena as informações geradas pelas outras funções e disponibiliza essas informações para o uso de medições para a melhoria do processo e para geração de relatórios gerenciais. A avaliação e revisão da configuração ocorrem quando há a liberação para o cliente da *baseline*, conjunto de IC's formalmente aprovados que serve de base para etapas seguintes (IEEE, 1990). Por fim, há o gerenciamento de liberação e entrega, que descreve o processo formal da construção, liberação e entrega do software (MURTA, 2006).

De acordo com MURTA (2006), olhando pelo foco de desenvolvimento, a GCS é dividida em três sistemas, que são: controle de modificações, controle de versões e gerenciamento de construção. Sistemas de controle de modificações executam a função de controle da configuração de forma sistemática e disponibilizam informações aos indicados

pela função de contabilização da situação da configuração. SCV lidam com a evolução dos IC's de forma distribuída e concorrente. Sistemas de gerenciamento de construção automatizam o processo de produção e disponibilização dos artefatos executáveis do software, conforme definido na função de gerenciamento de liberação e entrega, e estruturam as *baselines* envolvidas na liberação, conforme função de avaliação e revisão da configuração.

Frequentemente, durante o desenvolvimento ou manutenção de software sob controle de versão, há a necessidade de criação de ramos, ou seja, versões que não seguem a linha principal de desenvolvimento, possibilitando o desenvolvimento em paralelo (BERCZUK; APPLETON, 2002; WALRAD; STROM, 2002).

As próximas seções detalham conceitos de SCV e ramificação, devido à necessidade do entendimento dos conceitos aplicados à abordagem proposta. Deste modo, este capítulo está organizado nas seguintes seções: Na Seção 2.2 são discutidos conceitos de SCV; Na Seção 2.3 são discutidos conceitos de ramos; Na Seção 2.4 é discutida a junção dos ramos; Na Seção 2.5 são apresentadas estratégias de ramificação; Na Seção 2.6 são apresentados outros trabalhos da literatura; e, finalmente, na Seção 2.7 são feitas as considerações finais.

2.2 SISTEMA DE CONTROLE DE VERSÃO

Uma das principais características dos SCV é permitir que desenvolvedores colaborem de forma controlada. Os SCV possibilitam o armazenamento do histórico do desenvolvimento do software e a recuperação e comparação de versões de artefatos, permitindo assim o rastreamento das alterações feitas durante o desenvolvimento do software (DART, 1991; ESTUBLIER, 2000).

Segundo BERCZUK e APPLETON (2002), melhorias importantes na produtividade da equipe de desenvolvimento e na qualidade do software são obtidas com a escolha apropriada de práticas de controle de versão. Os SCV colaboram com a interação da equipe, conforme é visto na Figura 2, com a integração das mudanças realizadas pelos desenvolvedores em sua área de trabalho. As principais características dos SCV são (BERCZUK; APPLETON, 2002):

- Possibilitar o trabalho conjunto dos desenvolvedores, compartilhando código comum;
- Dividir o esforço para desenvolvimento de um módulo, onde mesmo com um dos envolvidos ausentes, outro desenvolvedor possa continuar o desenvolvimento que seria de responsabilidade da pessoa ausente;

- Permitir o desenvolvedor testar suas alterações com a versão estável do software, antes de efetivamente executar a integração;
- Possibilitar o rastreamento de problemas atuais no software, voltar com uma versão estável antiga do software e então testar o código comparando com os resultados da versão atual; e,
- Possibilitar o teste da versão atual do trabalho com alterações liberadas.

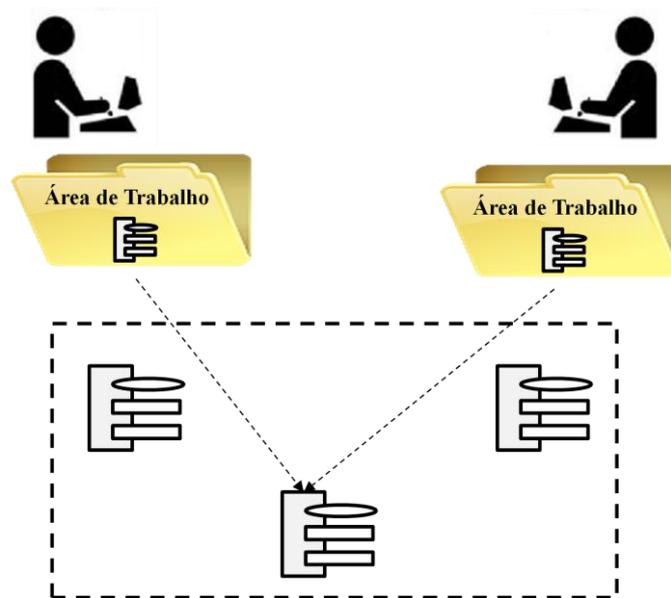


Figura 2: Integração das mudanças entre os desenvolvedores.
Adaptada de BERCZUK e APPLETON (2002).

Os conceitos principais de SCV, topologias e o repositório são discutidos no restante deste capítulo.

2.2.1 CONCEITOS PRINCIPAIS DE SCV

SCV possibilitam, de acordo com a necessidade, o desenvolvimento através de diferentes políticas de controle de concorrência (PRUDÊNCIO *et al.*, 2012). Dentre essas políticas, é possível citar as políticas pessimista e otimista. A política pessimista foca no bloqueio do artefato quando este estiver sendo alterado, inibindo o paralelismo no desenvolvimento. Por outro lado, a política otimista permite que o artefato seja modificado por vários desenvolvedores ao mesmo tempo. Contudo, se ocorrer concorrência sobre os mesmos artefatos, um mecanismo conhecido como junção (do inglês, *merge*) é adotado para combinar os trabalhos realizados em paralelo, gerando uma nova versão que integra esses trabalhos. A política otimista é mais amplamente utilizada atualmente, por atender mais

satisfatoriamente à maioria das situações referentes ao desenvolvimento de software (ESTUBLIER, 2001).

Um conceito muito difundido, utilizado pela maioria dos SCV, é o modelo *check-out–edição–check-in* para gerenciar a evolução dos artefatos armazenados no repositório. A Figura 3 ilustra este modelo. Não é permitida a modificação de nenhum artefato sem antes fazer o *check-out*, ou seja, fazer uma cópia do artefato para o espaço de trabalho do desenvolvedor. Após executar o *check-out*, o artefato poderá ser modificado livremente no espaço de trabalho do desenvolvedor. Ao término das alterações, é realizado o *check-in*, atualização do repositório com o conteúdo do artefato do espaço de trabalho (APPLETON *et al.*, 1998).

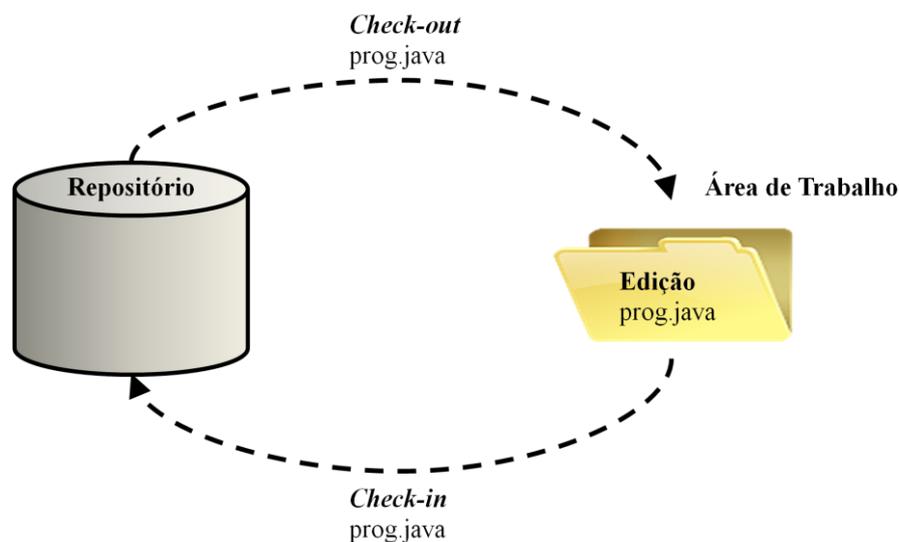


Figura 3: Modelo check-out–edição–check-in.
Adaptada de APPLETON *et al.* (1998).

Suponha que este artefato de software seja mantido por um desenvolvedor de um projeto da empresa, onde somente esse projeto faz uso dele. A cada *check-in* realizado com modificações no artefato, uma nova versão é criada no repositório. A Figura 4 mostra um exemplo do versionamento do arquivo *prog.java* iniciando a partir da versão 1.

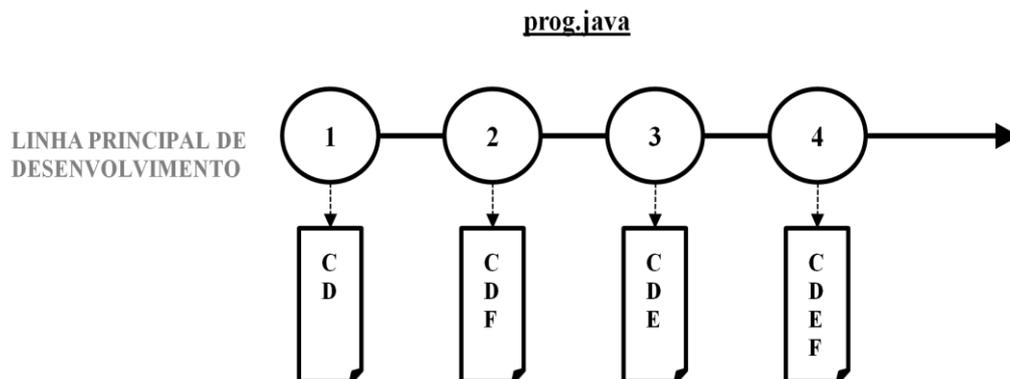


Figura 4: Exemplo do versionamento do arquivo prog.java.

Cada letra representa uma linha hipotética do arquivo. O arquivo *prog.java* na versão 1 possui duas linhas: “C” e “D”. Na versão 2, foi realizada uma alteração na versão inicial do arquivo, com a inclusão da linha “F”. Na versão 3, foi incluída a linha “E” e removida a linha “F”, onde esta é novamente inserida na versão 4. A cada modificação no artefato, e posterior execução do *check-in*, será criada uma versão armazenando o artefato com as alterações realizadas.

2.2.2 TOPOLOGIAS

Em relação à forma de armazenamento, os SCV podem ser classificados como centralizados ou distribuídos (O’SULLIVAN, 2009a). Os sistemas centralizados possuem repositório único compartilhado por todos os desenvolvedores, conforme ilustrado na Figura 5. Dentre as soluções livres e comerciais mais utilizadas, é possível citar: CVS (FOGEL; BAR, 2001), Subversion (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2008), ClearCase (WHITE, 2000) e Visual SourceSafe (SERBAN, 2007). Por outro lado, os sistemas distribuídos possuem repositório local para cada desenvolvedor, e um repositório remoto integrador das alterações de cada repositório local, conforme representado na Figura 6. Da mesma forma que as centralizadas, dentre as soluções livres e comerciais mais utilizadas, estão: Mercurial (O’SULLIVAN, 2009b), Git (CHACON, 2009), Bazaar (CANONICAL, 2011) e BitKeeper (BITMOVER, 2011).

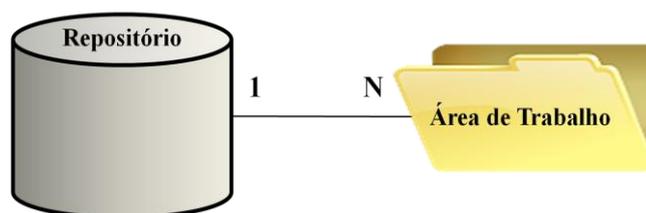


Figura 5: Sistema de Controle de Versão Centralizado.

Os sistemas centralizados são o modelo padrão para SCV (CHACON, 2009). Por terem um repositório único, pode haver sobrecarga para acessá-lo devido à concorrência. Um cuidado maior deve haver no controle de *backup*, pois uma falha no repositório coloca em risco todo o histórico dos artefatos armazenados.

Os sistemas distribuídos permitem a configuração de repositórios descentralizados e minimizam problemas referentes ao desempenho. Por terem os repositórios distribuídos, sua implementação encoraja o uso frequente de ramos e consequentemente leva a mais junções (BRUN *et al.*, 2010).

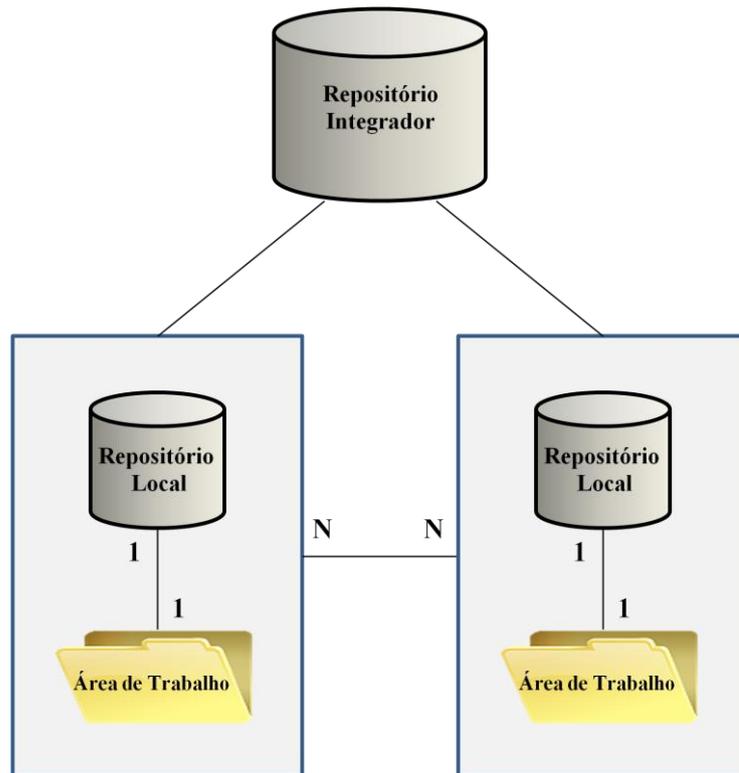


Figura 6: Sistema de Controle de Versão Distribuído.

2.2.3 REPOSITÓRIO

O repositório é o elemento principal do SCV, local onde ficam armazenados os artefatos de software (BERCZUK, 2003). Todas as versões são armazenadas nele e o acesso é feito de forma controlada (LEON, 2000).

Normalmente, os repositórios de SCV são divididos em três compartimentos para armazenamento dos artefatos de software, onde cada um possui um significado lógico para diferentes objetivos. O primeiro compartimento é a linha principal de desenvolvimento (do inglês, *mainline*), onde, se não houver uma necessidade extra de usar os outros compartimentos, concentra todos os artefatos. O segundo compartimento é constituído por ramos, que surgem da necessidade de haver uma linha de desenvolvimento isolada da linha principal de desenvolvimento. O terceiro compartimento é formado por etiquetas (do inglês, *tags*), que armazenam configurações específicas que tenham relevância para os desenvolvedores. A Figura 7 mostra a estrutura do repositório.

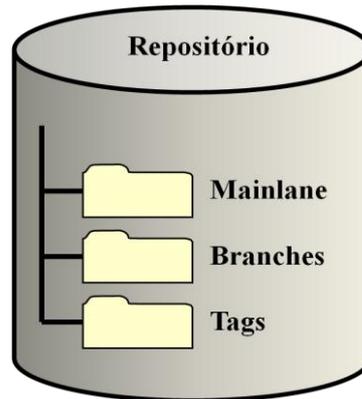


Figura 7: Estrutura do repositório.

Quando um artefato é colocado sob controle de versão, é esperado que todas as suas versões sejam armazenadas no repositório. Contudo, isso requer uma grande quantidade de espaço em disco. Para resolver essa questão, cada modificação realizada sobre o artefato, comumente chamado de *delta* (Figura 8), é armazenada pelo SCV (HUNT; MCILROY, 1976). Desta forma, os algoritmos de delta são capazes de reduzir drasticamente o espaço necessário para armazenar o conteúdo de todas as versões do artefato. Havendo a necessidade de consultar uma determinada versão, os deltas são aplicados para fazer a sua recomposição. Eles também são utilizados para realizar comparações entre as versões (WHITE, 2000).

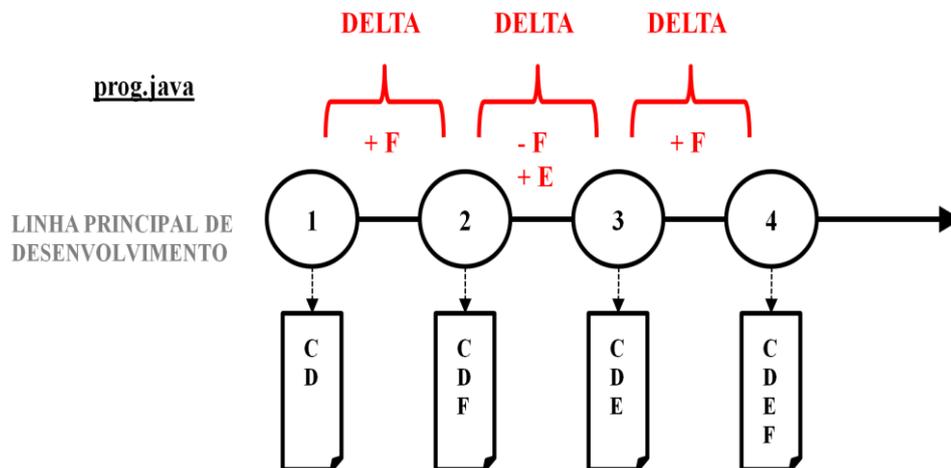


Figura 8: Delta entre versões.

2.3 RAMOS

Seguindo com o exemplo da Figura 4, que mostra o versionamento do arquivo `prog.java` utilizado no desenvolvimento de um projeto, poderia ser percebida a necessidade de outro projeto de usar este mesmo artefato, mas com alguns pequenos ajustes. O que fazer

nesta situação? Uma alternativa seria fazer uma cópia deste artefato e dar manutenção nas duas cópias separadamente. Contudo, frequentemente seriam necessárias as mesmas alterações em ambas as cópias, por exemplo, em função da descoberta de erros comuns que precisam ser corrigidos. Este cenário levaria a retrabalho e possivelmente queda da qualidade dos artefatos devido a manutenções incompletas.

O conceito de ramos surge dessa necessidade, consistindo em uma linha de desenvolvimento que existe independente de outra, mas que compartilham um histórico comum no passado. Um ramo sempre inicia a partir de algo existente, e evolui, gerando seu próprio histórico (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2008). A Figura 9 mostra um ramo criado a partir da versão 2 do versionamento do arquivo *prog.java*, em que será possível realizar alterações do arquivo no ramo (versões 2.1.1, 2.1.2, ...) independentes das que estão ocorrendo na linha principal de desenvolvimento (versões 3, 4, ...). Um ramo também pode ser criado a partir de outro ramo (LEON, 2000).

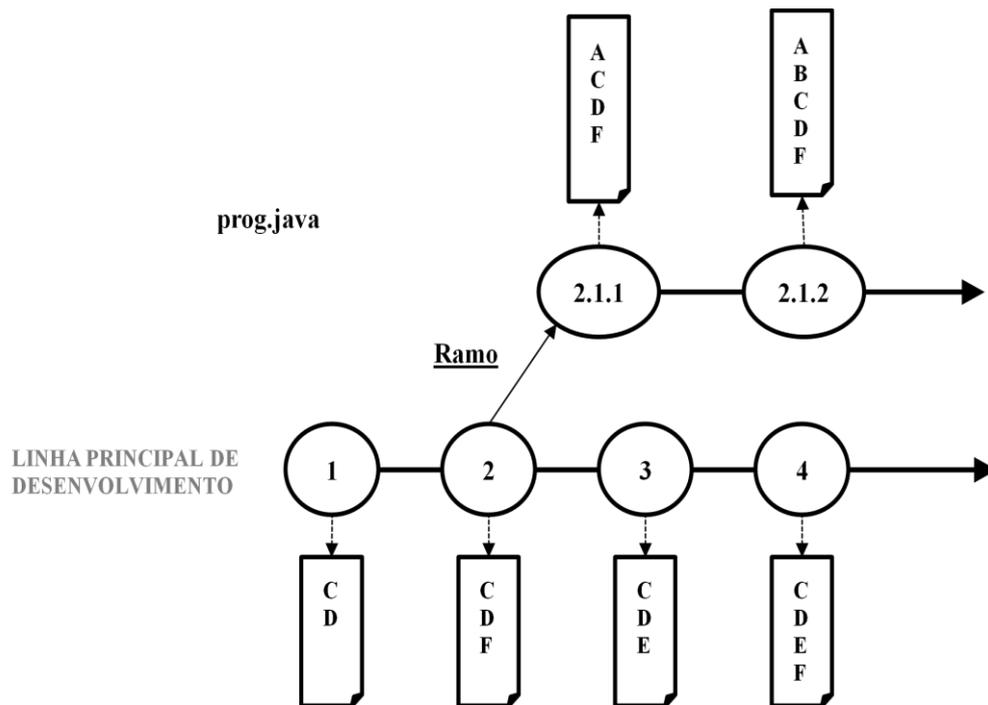


Figura 9: Criação de ramo do arquivo *prog.java*.

A ramificação é parte integrante do gerenciamento de versões, construções e liberações. Assim sendo, ramos proporcionam desenvolvimento em paralelo, permitindo o isolamento entre as linhas de desenvolvimento, a personalização do software para diferentes clientes ou plataformas, o isolamento entre desenvolvedores e equipes, o suporte à manutenção concorrente de múltiplas liberações, além de estabelecer um mapeamento entre a

identificação específica de uma liberação e as versões dos seus diferentes módulos, dentre outras funcionalidades (WALRAD; STROM, 2002).

2.4 JUNÇÃO DE RAMOS

Na maioria das vezes, os ramos sofrem junção com a linha principal de desenvolvimento em algum momento, porém esse processo pode ser custoso, em especial quando o ramo fica isolado por um longo período. Isso acontece devido a edições no código tanto nos ramos quanto na linha principal de desenvolvimento, aumentando a chance de haver um maior número de conflitos durante a junção e inviabilizando a sua execução automática (WALRAD; STROM, 2002). Desta forma, saber um momento apropriado para efetuar a junção é de grande importância para manter a produtividade da equipe e a qualidade do produto.

A junção, quando aplicada sobre dois artefatos sem um ancestral em comum, é denominada *two-way merging*. Já na abordagem *three-way merging*, é considerada na realização da junção a existência de um ancestral comum entre os dois artefatos, facilitando decisões de quais partes do artefato deverão compor o resultado da junção (MENS, 2002). No caso de ramos, que derivam de um ponto comum, a abordagem *three-way merging* sempre pode ser aplicada (CONRADI; WESTFECHTEL, 1998).

Como exemplo de execução automática da junção, a versão mais recente do arquivo *prog.java* do ramo é combinada com a versão mais recente do arquivo na linha principal de desenvolvimento, ambos apresentados na Figura 9. O resultado da junção pode ser visto na Figura 10, onde as alterações ocorridas até a versão 2.1.2 no ramo (inclusão das linhas “A” e “B”) são combinadas com a versão 4 da linha principal de desenvolvimento, gerando a versão 5.

Contudo, a abordagem *three-way merging* adotada nos SCV atuais leva em consideração somente quais linhas foram adicionadas ou removidas, e combina essas ações durante a junção. Desta forma, podem ocorrer conflitos, ou seja, o algoritmo pode não conseguir avaliar qual deve ser o resultado da junção entre partes dos artefatos. Essa avaliação envolve situações mais simples, onde o conflito físico é meramente textual (e.g., a mesma linha de um artefato ter sido editada por diferentes desenvolvedores), e situações mais complexas, que entram no âmbito sintático e semântico. Conflitos sintáticos ocorrem quando o código deixa de seguir a gramática da linguagem de programação após a junção, culminando em falhas na compilação. Já conflitos semânticos estão relacionados com erros na lógica do código e, conseqüentemente, falhas na execução dos casos de teste. Os conflitos

físicos, sintáticos e semânticos fazem com que o processo de realização da junção seja potencialmente lento, complicado e suscetível a erros (MENS, 2002; BERZINS, 1994; HORWITZ; PRINS; REPS, 1989).

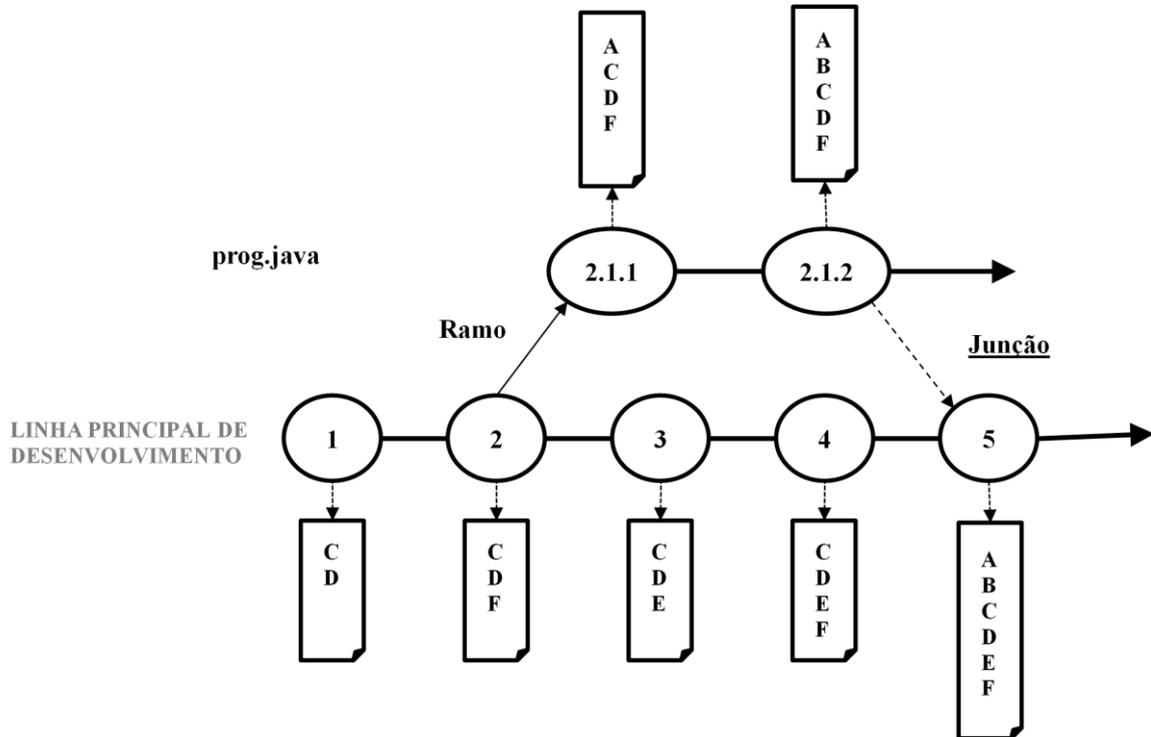


Figura 10: Junção do ramo do arquivo `prog.java`.

Um exemplo de conflito físico pode ser visto na Figura 11. Neste exemplo, as alterações ocorridas até a versão 2.1.3 do arquivo `prog.java` no ramo consistem na inclusão das linhas “A” e “B” no início do arquivo e “X” entre as linhas “C” e “D”. Entretanto, a versão 5 da linha principal de desenvolvimento também sofreu a inclusão de uma linha “Y” entre as linhas “C” e “D”. Conseqüentemente, na versão 6, resultante da junção, qual deverá ser a linha colocada entre as linhas “C” e “D”? O desenvolvedor terá que interceder para a escolha do resultado, que dependendo da situação, poderá ser uma das linhas, as duas juntas, uma nova linha ou nenhuma delas.

O programa `diff3` identifica as modificações realizadas entre dois artefatos que possuem um ancestral comum, viabilizando assim uma análise mais facilitada por parte do desenvolvedor (CONRADI; WESTFECHTEL, 1998). Na Figura 12 é mostrado um exemplo de sua aplicação, usando como base o arquivo `prog.java` armazenado no repositório, conforme Figura 11. O resultado apresenta a aplicação do programa `diff3` entre a versão 5 da linha principal de desenvolvimento e a versão 2.1.3 do ramo.

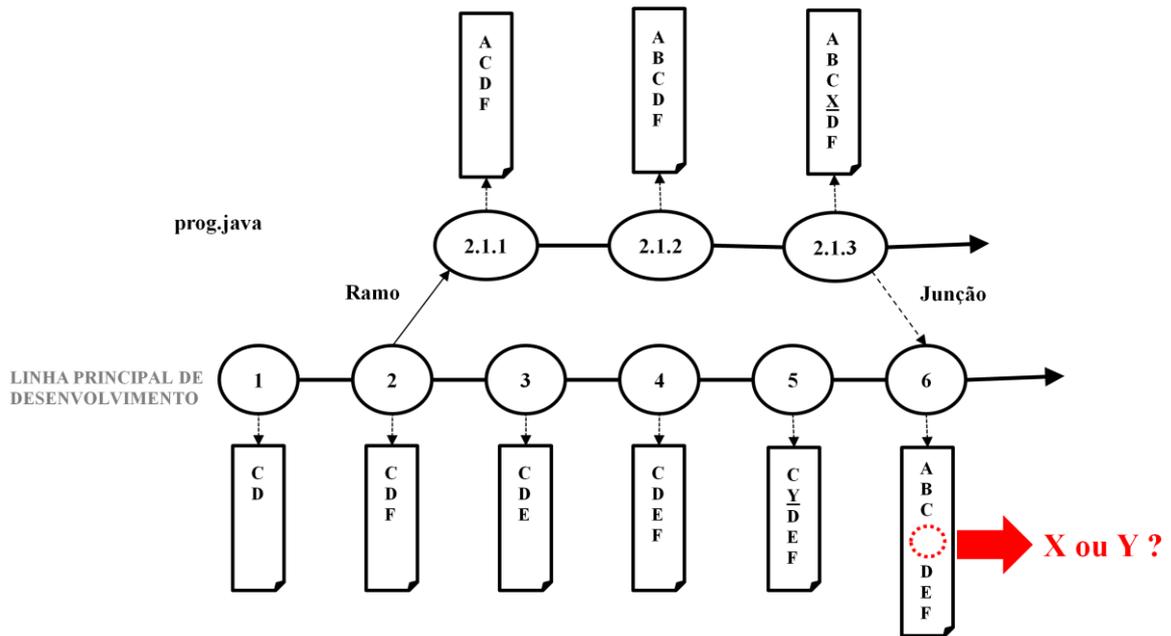


Figura 11: Exemplo de conflito físico.

```

--- prog.java    (.../trunk)
+++ prog.java    (.../branches/branch1)
@@ -1,5 +1,6 @@
+ A
+ B
  C
+ X
- Y
  D
- E
  F

```

Figura 12: Execução do diff3.

O resultado do programa *diff3* entre essas duas versões do arquivo é apresentado usando o formato unificado (GNU, 2011). Nas duas primeiras linhas do resultado, é exibido o local onde cada versão está armazenada e indicada qual legenda será utilizada para cada versão (no caso, “-” para o a linha principal e “+” para o ramo). Posteriormente, a informação entre os sinais “@@” representa a região da versão de cada arquivo que está sendo apresentada no delta gerado. No exemplo, numerando as linhas a partir de 1, o delta contempla as linhas de 1 a 5 do arquivo na linha principal de desenvolvimento, e as linhas de

1 a 6 do arquivo no ramo. Após, são mostradas as linhas pertencentes a cada versão, onde as que são precedidas do sinal “-” pertencem à versão da linha principal de desenvolvimento, as que são precedidas do sinal “+” pertencem à versão do ramo, e as linhas sem sinal são comuns às duas versões.

2.5 ESTRATÉGIAS DE RAMIFICAÇÃO

De acordo com WALRAD e STROM (2002), o conjunto de decisões sobre quando e por que criar um ramo para atingir um objetivo definem as estratégias de ramificação. Sua escolha correta torna mais fácil a coordenação do desenvolvimento e as alterações no software. Essa escolha pode contribuir evitando erros na obtenção das liberações antigas, apoiando na escolha dos artefatos para construir um produto ou permitindo o restabelecimento de um estado anterior do software. Desta forma, as estratégias de ramificação são voltadas para propósitos específicos (BERCZUK; APPLETON, 2002): manter uma base estável para desenvolvimentos novos; realizar liberação emergencial da versão em produção com a correção de defeitos e sem a incorporação de outras mudanças evolutivas; reduzir o impacto de correções emergenciais nos novos desenvolvimentos em curso; permitir a restauração de versões e liberações anteriores; e suportar múltiplas versões, concorrentes ou sequenciais, como, por exemplo, versões alternativas para sistemas operacionais ou bancos de dados diferentes.

As próximas seções deste capítulo apresentam algumas das principais estratégias de ramificação agrupadas por similaridades.

2.5.1 ESTRATÉGIAS PARA MANUTENÇÃO DE SOFTWARE

A estratégia de ramificação **Em Série** (Figura 13) é utilizada para manter em paralelo a evolução e correção do software. A linha principal de desenvolvimento contém a evolução do software e as correções são desenvolvidas nos ramos auxiliares. Essa estratégia é apropriada para situações onde não há a necessidade de mais de uma liberação concomitante em produção, pois dificulta a manutenção de várias liberações em paralelo. A junção é sempre realizada no sentido do ramo para linha principal de desenvolvimento (MURTA, 2011).

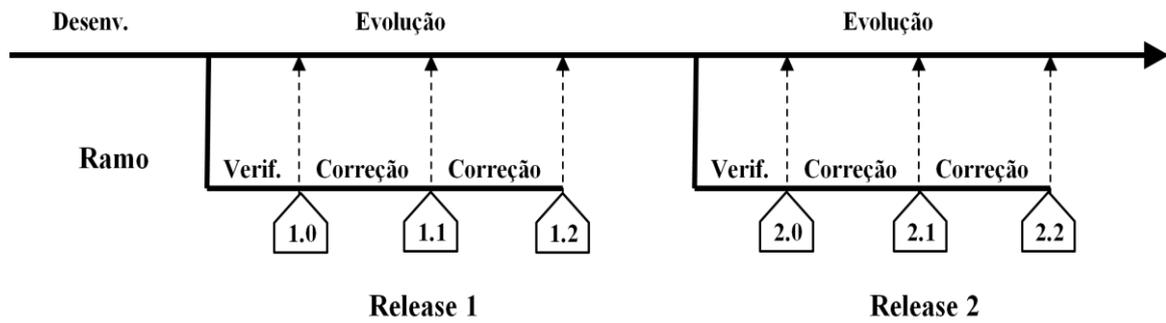


Figura 13: Estratégia de Ramificação Em Série.

A estratégia de ramificação **Em Cascata** (Figura 14) possui o mesmo objetivo da *Em Série*, que é manter em paralelo manutenções corretivas e evolutivas, porém, a linha principal de desenvolvimento contém as correções e as evoluções do código são desenvolvidas nos ramos auxiliares. Em decorrência disso, possibilita a manutenção em paralelo de múltiplas liberações em produção, porém torna mais onerosa a correção, que dependendo do caso, deverá ser aplicada em um ramo e propagada para os demais. A junção é sempre realizada no sentido da linha principal de desenvolvimento para o ramo (MURTA, 2011).

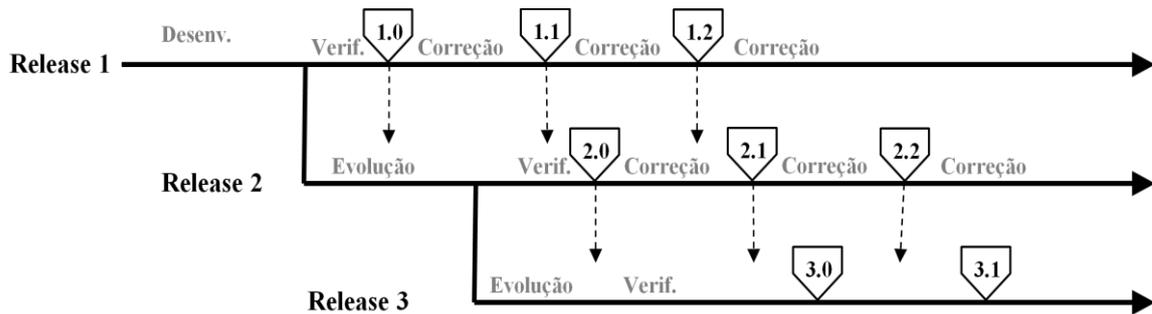


Figura 14: Estratégia de Ramificação Em Cascata.

2.5.2 ESTRATÉGIAS PARA ISOLAMENTO DE EQUIPES

A estratégia de ramificação **Por Componentes** (Figura 15) é utilizada para dar manutenção em uma parte específica do software, que pode ser um módulo ou componente. Ramos auxiliares são criados para cada componente, de onde são feitas junções para a linha principal de desenvolvimento. A junção é sempre realizada no sentido do ramo auxiliar para a linha principal de desenvolvimento (APPLETON *et al.*, 1998).

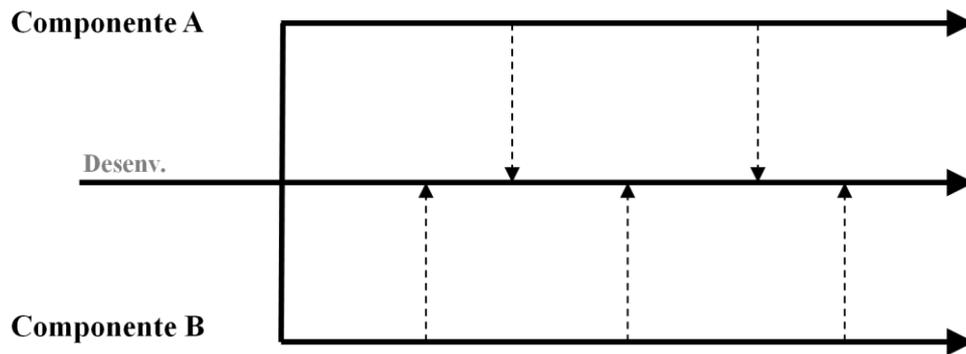


Figura 15: Estratégia de Ramificação Por Componentes.

A estratégia de ramificação **Por Desenvolvedor** (Figura 16) é utilizada para isolar o trabalho de um desenvolvedor do restante da equipe. Isso permite o desenvolvedor fazer *check-ins* intermediários nos ramos auxiliares sem afetar outros desenvolvedores que estão trabalhando na linha principal de desenvolvimento. A junção ocorre em ambos os sentidos. Quando a equipe deseja disponibilizar alterações para que sejam utilizadas pelo desenvolvedor, é realizada a junção sentido da linha principal de desenvolvimento para ramo. Por outro lado, quando a propagação do trabalho do desenvolvedor para a linha principal de desenvolvimento for segura, é realizada a junção no sentido do ramo para linha principal de desenvolvimento (MURTA, 2011).

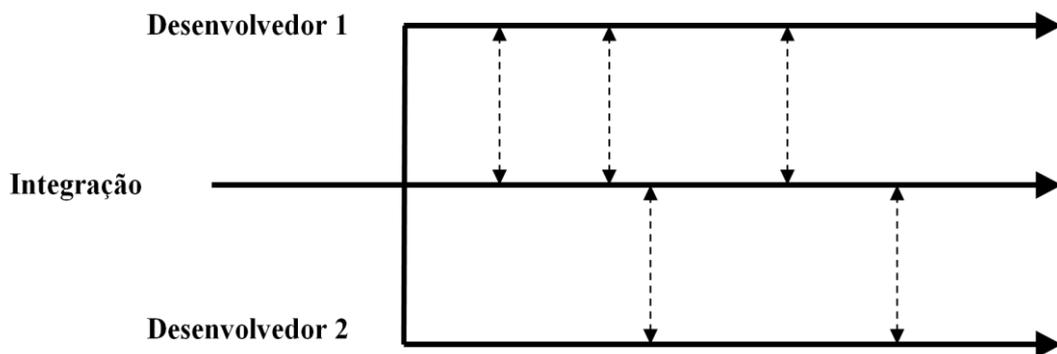


Figura 16: Estratégia de Ramificação Por Desenvolvedor.

A estratégia de ramificação **Por Requisições** (Figura 17) é utilizada para o desenvolvimento de funcionalidades, facilitando assim a sua remoção de uma determinada versão do software ou migração entre versões. Ramos auxiliares são utilizados para o desenvolvimento de requisições, não comprometendo a integridade da linha principal de desenvolvimento. A junção é sempre realizada no sentido do ramo para a linha principal de desenvolvimento (MURTA, 2011).

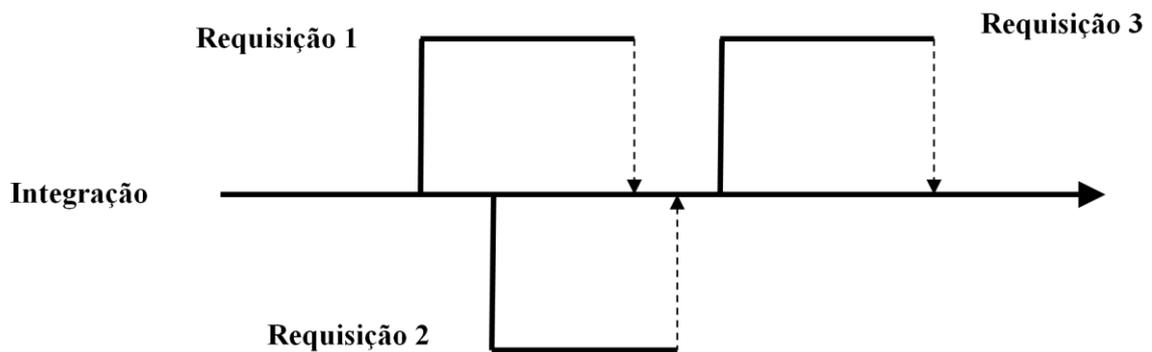


Figura 17: Estratégia de Ramificação Por Requisições.

A estratégia de ramificação **Por Subprojetos** (Figura 18) é utilizada para o desenvolvimento de subprojetos, que consistem em mais de uma tarefa. Ramos auxiliares são utilizados para isolar membros de um mesmo subprojeto. A junção ocorre em ambos os sentidos. Quando a equipe da linha principal de desenvolvimento deseja disponibilizar alterações para os membros do subprojeto, a junção é realizada no sentido da linha principal de desenvolvimento para ramo auxiliar. Por outro lado, quando alterações do subprojeto devem ser integradas à linha principal de desenvolvimento, a junção é realizada no sentido do ramo para linha principal de desenvolvimento (APPLETON *et al.*, 1998; MURTA, 2011).

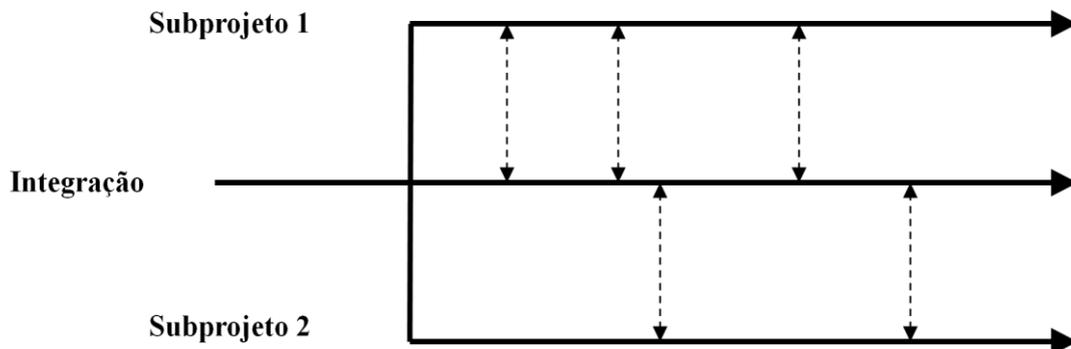


Figura 18: Estratégia de Ramificação – Por Subprojetos.

2.5.3 ESTRATÉGIAS PARA ADAPTAÇÃO DE SOFTWARE

A estratégia de ramificação **Por Personalizações** (Figura 19) é utilizada para apoiar múltiplos ambientes ou plataformas. Ramos auxiliares são utilizados para cada personalização necessária, permitindo que o software seja compatível com diferentes ambientes ou plataformas. O código comum entre eles é desenvolvido na linha principal de desenvolvimento e replicado para todos os ramos. Um exemplo seria a linha principal de desenvolvimento contendo o código base de um software e cada ramo auxiliar contendo o

código específico para a utilização do software em diferentes sistemas operacionais. Um ramo contendo personalização para o Windows, outro contendo personalização para o Mac, outro com personalização para o Linux, e assim por diante. A junção é sempre realizada no sentido da linha principal de desenvolvimento para o ramo auxiliar (APPLETON *et al.*, 1998; MURTA, 2011).

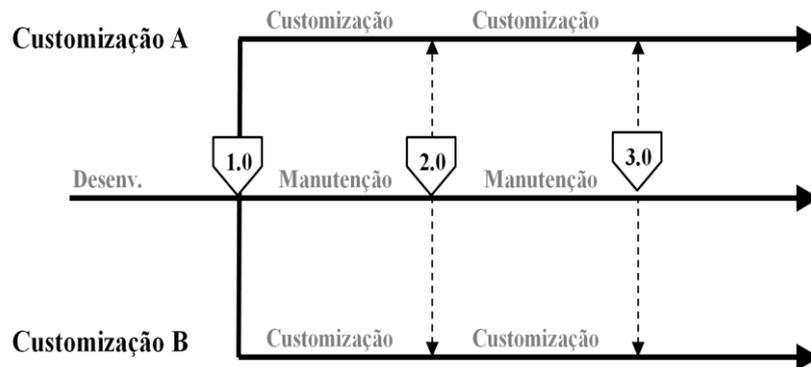


Figura 19: Estratégia de Ramificação – Por Personalizações.

A estratégia de ramificação **Por Terceirização** (Figura 20), também conhecida como *Vendor Branching* (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2008; FOGEL; BAR, 2001), permite armazenar o código de uma parte do software mantido por um grupo ou empresa externa à equipe de desenvolvimento. Um ramo auxiliar mantém esse código, que é atualizado através das liberações do fornecedor externo. Isso possibilita selecionar qual versão do código de terceiros utilizar, e, em caso de erro em alguma liberação, trocar para uma versão estável. A junção é sempre realizada no sentido do ramo (código externo) para a linha principal de desenvolvimento (APPLETON *et al.*, 1998). É importante notar que apesar da primeira versão da linha principal de desenvolvimento ser originada do código de terceiros, tradicionalmente esse código é chamado de ramo (*vendor branch*) e o código de desenvolvimento é considerado a linha principal de desenvolvimento.

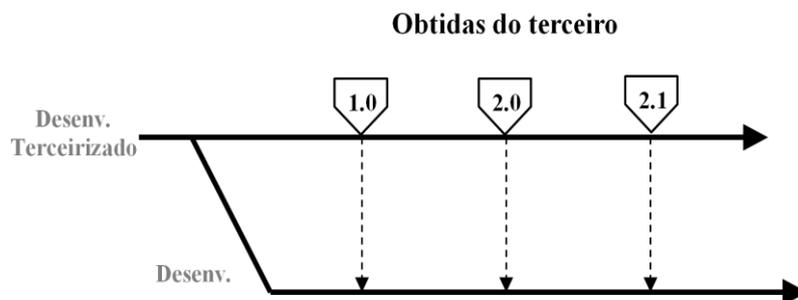


Figura 20: Estratégia de Ramificação Por Terceirização.

2.6 TRABALHOS RELACIONADOS

Esta seção apresenta trabalhos que focam na comparação, junção e percepção das alterações entre diferentes versões do software. Eles dão ênfase em um ou mais desses aspectos. Para identificar tais trabalhos, foram realizadas buscas de palavras chave nas principais bibliotecas digitais da área (*ACM*, *IEEE*, *ScienceDirect* e *SpringerLink*). No início das buscas foram utilizadas algumas combinações de palavras chave que envolviam termos como: “*branch*”, “*merge*”, “*metrics*”, “*visualization*”. Como os resultados dessas buscas não foram relevantes para a pesquisa, eles foram descartados. Em virtude disso, foram sendo tentados outros termos de sentido mais amplos que tinham relação com objetivo da pesquisa, onde no final foi utilizada uma combinação de palavras chave como: “*resemblance*”, “*likeness*”, “*comparison*”, “*awareness*”, com palavras chave como: “*application*”, “*system*”, “*program*”, “*source code*”. Dessa pesquisa, foram identificados resultados que ainda passaram por uma filtragem para descartar trabalhos que não possuem relação com o tema deste trabalho. A partir dos resultados não descartados, foram realizadas buscas de artigos que eram referenciados ou referenciavam tais trabalhos, recursivamente. Naturalmente, muitos trabalhos são descartados por não terem interseção aos assuntos básicos e a busca se esgota.

Na análise dos resultados, foram encontrados alguns trabalhos relativos à detecção de plágio: Roxas *et al.* (2006), Lesner *et al.* (2010), Ji *et al.* (2007), Li *et al.* (2010) e Ohno e Murao (2009). Outro grupo de trabalho encontrado foi relativo à similaridade de código: Chilowicz *et al.* (2009), Lakkaraju *et al.* (2008), Ilyas e Kung (2009) e Sager *et al.* (2006). E, ao final da análise, foram identificados os seguintes trabalhos mais aderentes à proposta da busca: Horwitz (1990), Berzins (1994), Sarma *et al.* (2003), Dewan e Hegde (2007), Wloka *et al.* (2009) e Brun *et al.* (2010).

Os trabalhos de Berzins (1994) e Horwitz (1990) focam na identificação e significado de diferenças entre versões de software. Berzins (1994) apresenta um modelo semântico independente de linguagem para combinar mudanças de duas versões do software. O modelo é utilizado para definir o significado semântico das mudanças nas operações de junção e estabelecer propriedades comuns do software. Sua principal contribuição é a construção de um domínio extensível, independente da linguagem, que pode apoiar a definição e formalização da noção de um conflito semântico entre as mudanças. Condições são determinadas para mudanças de subprogramas do software, e o modelo ilustra casos onde isso não é possível. Por outro lado, Horwitz (1990) propõe um conjunto de métodos para identificar diferenças textuais e semânticas entre duas versões de um software. Esses métodos

requerem um passo de pré-processamento que pode determinar trechos comuns entre as versões através da construção de uma representação gráfica do software. O algoritmo de particionamento é limitado a uma linguagem com variáveis escalares, declarações de atribuição, estruturas de decisão, estruturas de repetição e declarações de saída. Baseados na representação gráfica, três diferentes algoritmos são aplicados na busca de equivalência entre as versões. Cada trecho passa por critérios de otimização heurísticos, que maximizam o número de pares que possuem equivalência e determinam se houve uma alteração textual ou semântica naquele ponto.

Os trabalhos de Brun et al. (2010) e Wloka et al. (2009) focam em passar o conhecimento do impacto de alterações que estão ocorrendo em paralelo para o desenvolvedor, antes que elas efetivamente aconteçam no momento da junção com o conteúdo do repositório. Brun et al. (2010) avaliam se o desenvolvedor se beneficiaria em ter o conhecimento de possíveis conflitos com outros espaços de trabalho antes da execução do *check-in*. Com isso, a abordagem especula continuamente se há oportunidades seguras para a realização da junção entre as equipes de desenvolvimento. De forma equivalente, o *Safe-Commit* (Wloka et al. (2009)) foca em identificar modificações que podem ser enviadas para o repositório sem torná-lo inconsistente. Essa abordagem faz uso de políticas que diferenciam quais artefatos podem ser atualizados sem necessidade de teste (política permissiva), de outras que exigem a execução de testes quando os artefatos são atualizados (políticas moderada e restritiva). Neste último caso, o desenvolvedor terá mais confiança, pois, após as modificações serem aplicadas no repositório, os artefatos são verificados semanticamente através da execução de testes.

Sarma et al. (2003) apresentam a ferramenta Palantír, um espaço de trabalho que complementa os SCV existentes, provendo aos desenvolvedores percepção do que acontece nos espaços de trabalho de outros desenvolvedores. A ferramenta identifica conflitos físicos durante o desenvolvimento e calcula métricas do impacto dessas mudanças. Essas métricas podem ser mais simples, indicando somente se haverá ou não algum impacto dessas mudanças, ou podem ser mais sofisticadas, com o cálculo do número de linhas que tem sido somado, removido ou alterado dividido pelo número total de linhas do artefato. O resultado dessas medidas é exibido graficamente sem interferir no trabalho realizado pelo desenvolvedor. Dewan e Hegde (2007) apresentam uma abordagem que, enquanto o desenvolvedor está trabalhando em seu espaço de trabalho, identifica possíveis conflitos que venham a ocorrer com o trabalho de outros desenvolvedores. Essa abordagem proposta é baseada num modelo que provê um gerenciamento de conflito semi-sincronizado entre

espaços de trabalho. Com isso, permite que o desenvolvedor os resolva antes de finalizar sua tarefa e, conseqüentemente, antes que os conflitos ocorram de fato. De forma equivalente às abordagens do parágrafo anterior, em função das características propositais de isolamento, esses trabalhos são úteis para manter os espaços de trabalho internamente consistentes, com a diferença que esses focam na comparação com outros espaços de trabalho.

2.7 CONSIDERAÇÕES FINAIS

GCS pode ser utilizada tanto em projetos pequenos, com poucas pessoas, bem como em projetos grandes e complexos, com centenas ou milhares de pessoas envolvidas (LEON, 2000). Ela é uma disciplina necessária ao desenvolvimento profissional de software, que provê técnicas e ferramentas para controlar a evolução do software. Ela é apoiada tanto por funções de foco gerencial, quanto por sistemas com foco no desenvolvimento. Dentre esses sistemas, se destaca os SCV, que tiveram grande evolução nas últimas décadas (ESTUBLIER *et al.*, 2005).

Conforme discutido, os SCV colaboram com o trabalho em equipe, controlando a evolução dos artefatos, armazenando o histórico do desenvolvimento e permitindo o compartilhamento deles de forma coordenada. Além disso, eles proporcionam o desenvolvimento de um ou mais ramos em paralelo à linha principal de desenvolvimento, adotando diferentes estratégias de ramificação, como mostrado na Seção 2.5. Entretanto, não há indicação de qual será o esforço para a realização da junção desses ramos. Os trabalhos relacionados na Seção 2.6 focam em comparar versões de software, mas dentre esses trabalhos, não foi encontrada uma solução que pudesse mensurar o quanto um ramo será mais custoso que outro ramo para execução da junção. Apesar de alguns trabalhos lidarem com identificação de conflitos físicos (DEWAN; HEGDE, 2007; SARMA; NOROOZI; VAN DER HOEK, 2003), não foi encontrada uma solução que comparasse e avaliasse o ramo com os outros ramos do projeto e nem que viabilizasse o acompanhamento do histórico de evolução do esforço de junção do ramo.

CAPÍTULO 3 – VISUALIZAÇÃO DO ESFORÇO DE JUNÇÃO DE RAMOS

3.1 INTRODUÇÃO

Analisando repositórios de SCV que fazem uso de ramos, não há informações de o quanto será custosa a realização da junção. Tal fato motiva o surgimento de abordagens que desempenhem este papel. Contudo, conforme visto no Capítulo 2, as abordagens analisadas não focam na identificação do esforço de junção dos ramos com a linha principal de desenvolvimento. O seu foco é evitar que existam conflitos entre desenvolvedores que atuam no mesmo ramo, não dando garantias de que a junção do ramo com a linha principal de desenvolvimento seja sem conflitos e nem auxílio na identificação do momento adequado de junção. A abordagem Polvo (SANTOS, R.; MURTA, 2012) surgiu dessa necessidade e, deste modo, auxilia os desenvolvedores na análise da junção dos ramos.

Esta abordagem consiste em responder às questões da Figura 21, referentes ao quão difícil será juntar os ramos do projeto. Estas questões podem ser analisadas individualmente ou de forma geral para todo o projeto, dependendo das necessidades do desenvolvedor. A abordagem propicia caminho para as duas necessidades.

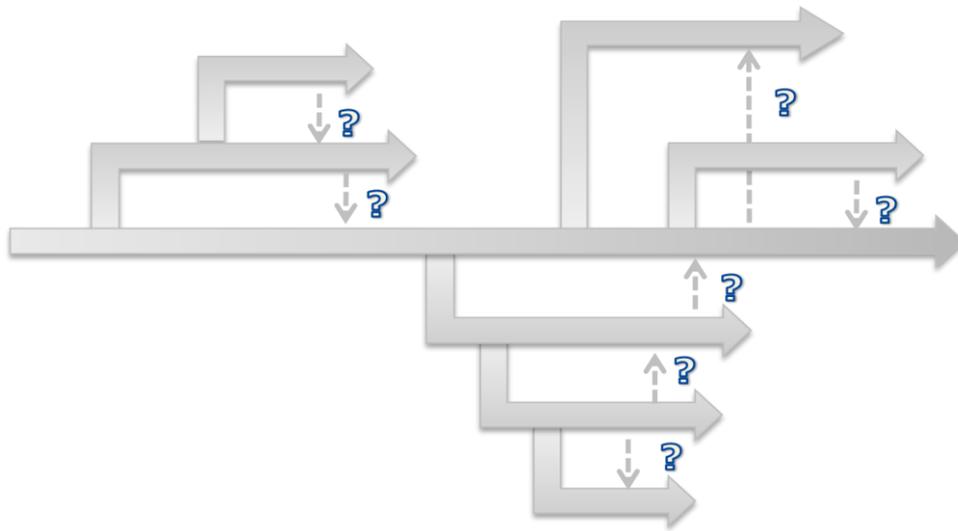


Figura 21: Questões da abordagem.

É importante notar na Figura 21 que existem ramos derivados de ramos, então sob a perspectiva do ramo derivado de outro ramo, o ramo fonte passa a ser considerado a linha principal de desenvolvimento. Além disso, as junções são direcionais e podem ocorrer em sentidos diferentes, dependendo da estratégia de ramificação que foi adotada. Outro ponto

relevante é a possível existência de vários ramos com origem a partir da linha principal de desenvolvimento. Contudo, mesmo que outro ramo mais recente tenha sido criado antes do momento da avaliação do esforço de junção, este não interferirá no cálculo.

Para visualizar o esforço da junção entre os ramos, a abordagem Polvo é proposta e definida neste capítulo. Este capítulo apresenta a seguinte divisão: na Seção 3.2 é dada uma visão geral do Polvo, ressaltando as opções que o desenvolvedor tem no uso da abordagem; na Seção 3.3 são descritas as métricas utilizadas; a Seção 3.4 aborda o uso da estratégia de ramificação; e, finalmente, na Seção 3.5 são feitas as considerações finais da abordagem.

3.2 VISÃO GERAL

A Figura 22 ilustra a visão geral da abordagem Polvo, cuja implementação será detalhada na Seção 4.4. Na abordagem é possível observar que o desenvolvedor tem dois caminhos a seguir: (1) fazer a análise diretamente de um determinado ramo que tenha interesse ou (2) ter uma visão geral do estado de todos os ramos. A partir destes dois caminhos é possível (3) acessar a visão temporal da evolução de um ramo específico.

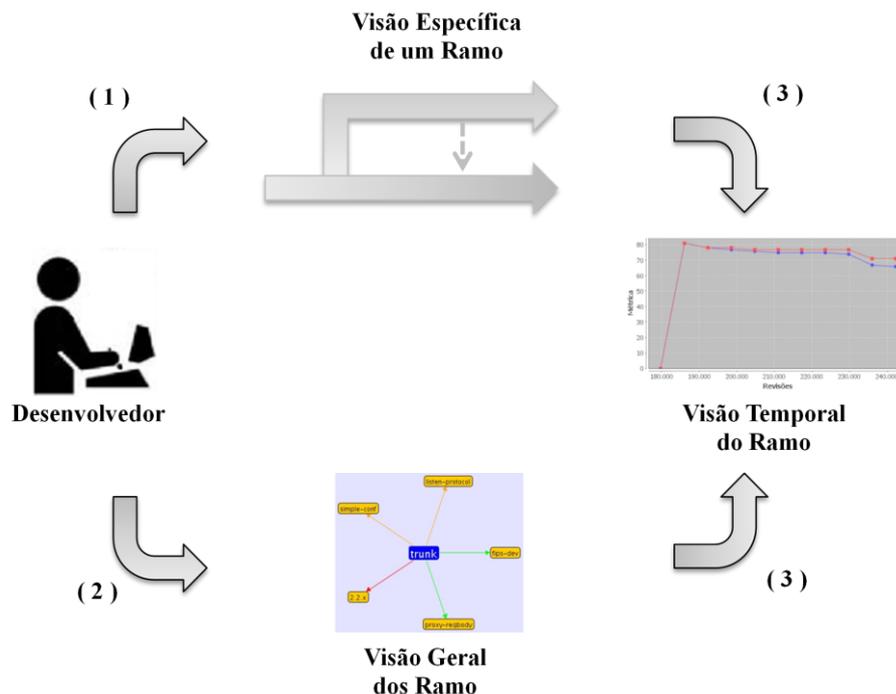


Figura 22: Abordagem Polvo.

Na *Visão Específica de um Ramo*, o desenvolvedor escolhe uma métrica para comparação e informa a estratégia de ramificação adotada para o ramo em questão. A abordagem, de posse dessas informações, realiza o cálculo do esforço de junção para aquele

ramo. Assim, o esforço de junção pode ser calculado de formas diferentes em função das métricas disponíveis, sendo estas um ponto de extensão da abordagem.

Nas outras visões da abordagem são adotadas metáforas visuais. Elas fornecem ao desenvolvedor representações mais perceptíveis para o entendimento da situação do repositório, tornando visíveis elementos que estão escondidos nos dados (DIEHL, 2007).

Para a exibição de todos os ramos na *Visão Geral dos Ramos*, foi utilizado um grafo com uma visão hiperbólica das arestas e nós. Essa visão permite a visualização de mais nós do que a visão tradicional do diagrama de árvore de duas dimensões (CHEN, 2006). Os nós centrais possuem uma resolução maior que os nós nas extremidades, onde, quando selecionados, passam a ser o foco principal. Esse efeito também é conseguido através de translação realizada no grafo, ao arrastar o nó desejado para o centro.

Na *Visão Temporal do Ramo* foi utilizado um gráfico de linha. No eixo X são apresentadas as versões do repositório analisado. No eixo Y são apresentadas as medições realizadas referentes ao esforço da junção do ramo. O cálculo da métrica selecionada é feito em intervalos regulares das versões desde a criação do ramo, mostrando sua evolução no tempo.

A abordagem Polvo não interfere no estado do repositório (Figura 23), os desenvolvedores poderão continuar trabalhando normalmente, realizando *check-outs* e *check-ins*, e, em paralelo, o Polvo acessa o histórico das versões enviadas para o repositório e extrai as informações utilizadas na abordagem.

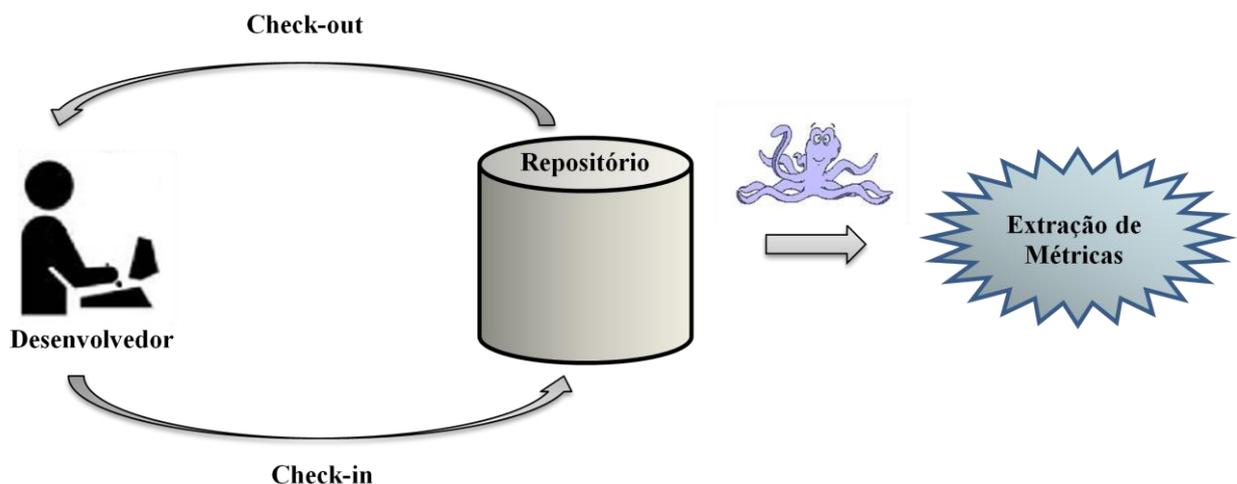


Figura 23: Abordagem Polvo não intrusiva.

No restante deste capítulo, estes conceitos são descritos com maiores detalhes, a fim de dar um melhor entendimento de como é feita a análise do esforço da junção dos ramos. As próximas seções apresentam as métricas utilizadas para comparar os ramos e as estratégias de

ramificação consideradas pela abordagem. Em função da estratégia de ramificação adotada, o sentido do cálculo das métricas pode variar.

3.3 MÉTRICAS

Dez métricas foram preconcebidas como forma de quantificar o esforço da junção dos ramos. Contudo, novas métricas podem ser adicionadas à abordagem. Uma descrição resumida dessas métricas é apresentada na Tabela 1. Para facilitar a nomenclatura, a referência para criação de um ramo é denominado *linha principal* e o ramo criado a partir da linha principal é denominado *ramo*. É importante notar que a linha principal pode já ser um ramo, mas sob a perspectiva da abordagem, naquele contexto, ele estaria exercendo o papel de linha principal.

Tabela 1: Métricas utilizadas no Polvo.

Métrica	Descrição
Quantidade de Artefatos Modificados na Linha Principal	Contabiliza o total de artefatos que foram modificados na linha principal.
Quantidade de Artefatos Modificados no Ramo	Contabiliza o total de artefatos que foram modificados no ramo.
Quantidade de Artefatos Modificados em Comum	Contabiliza o total de artefatos em comum que foram modificados tanto na linha principal como no ramo.
Cobertura por Quantidade de Artefatos (%)	Calcula a relação entre o número de artefatos iguais sobre o total de artefatos da linha principal.
Precisão por Quantidade de Artefatos (%)	Calcula a relação entre o número de artefatos iguais sobre o total de artefatos do ramo.
Quantidade de Linhas Diferentes na Linha Principal	Contabiliza o total de linhas de código que foram modificadas na linha principal.
Quantidade de Linhas Diferentes no Ramo	Contabiliza o total de linhas de código que foram modificadas no ramo.
Quantidade de Conflitos Físicos	Contabiliza o total de conflitos físicos que seriam reportados no processo de junção.
Quantidade de Conflitos Sintáticos	Não havendo conflitos físicos, contabiliza o total de conflitos sintáticos que seriam reportados no processo de junção.
Quantidade de Conflitos Semânticos	Não havendo conflitos físicos nem sintáticos, contabiliza o total de conflitos semânticos que seriam reportados no processo de junção.

Nas seções a seguir, essas métricas serão detalhadas, sendo agrupadas por formas de análise.

3.3.1 MÉTRICAS COM ANÁLISE DAS DIFERENÇAS

As primeiras sete métricas da Tabela 1 são extraídas através da análise do resultado do programa *diff3* executado entre a linha principal e o ramo.

A métrica **Quantidade de Artefatos Modificados na Linha Principal** apresenta uma visão geral, no nível de artefatos, de tudo o que foi modificado (inclusões, alterações e exclusões) na linha principal desde a versão em que foi realizada a criação. Sua medição se dá através da contabilização de todos os artefatos da linha principal que constem no *diff3* realizado entre a linha principal e o ramo. No exemplo apresentado na Figura 24, o resultado da aplicação dessa métrica é 50. Para chegar a esse resultado, somam-se os artefatos novos na linha principal (20), mais os que existiam no momento da criação do ramo e foram alterados ou excluídos (30).

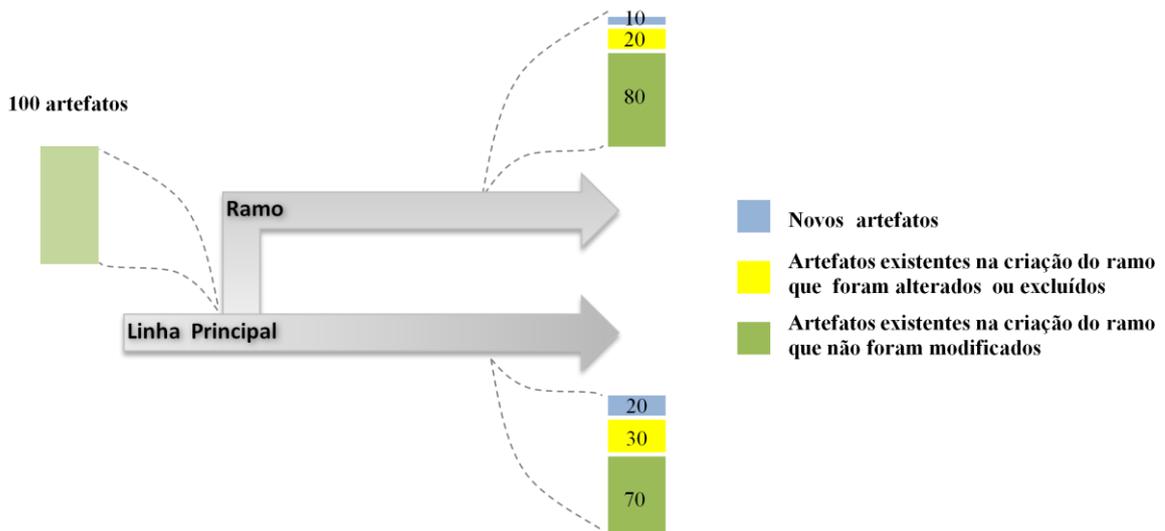


Figura 24: Exemplo da métrica Quantidade de Artefatos Modificados.

A métrica **Quantidade de Artefatos Modificados no Ramo** é similar à métrica Quantidade de Artefatos Modificados na Linha Principal, porém aplicada ao ramo. Sua medição se dá através da contabilização de todos os artefatos do ramo que constem no *diff3* realizado entre a linha principal e o ramo. No exemplo apresentado na Figura 24, o resultado da aplicação dessa métrica é 30. Para chegar a esse resultado, somam-se os artefatos novos no ramo (10), mais os que existiam no momento da criação do ramo e foram alterados ou excluídos (20).

A métrica **Quantidade de Artefatos Modificados em Comum** totaliza os artefatos existentes na criação do ramo que foram modificados tanto na linha principal quanto no ramo. Sua medição permite ao desenvolvedor ter a percepção de quais artefatos em comum foram modificados em ambos, dentre o total de artefatos modificados. Ou seja, artefatos que são

possíveis candidatos a ter algum tipo de conflito. Por exemplo, se a edição for na mesma área em ambos os artefatos, ocorrerá necessariamente um conflito físico. Sua medição se dá através da contabilização dos artefatos que constem no *diff3*, os quais possuam modificações na linha principal e no ramo em qualquer parte de seu conteúdo. No exemplo exibido na Figura 25, o resultado da aplicação dessa métrica é 10, ou seja, dos 30 artefatos modificados na linha principal e dos 20 modificados no ramo, 10 são os mesmos artefatos que existiam na criação do ramo e foram modificados em ambos.

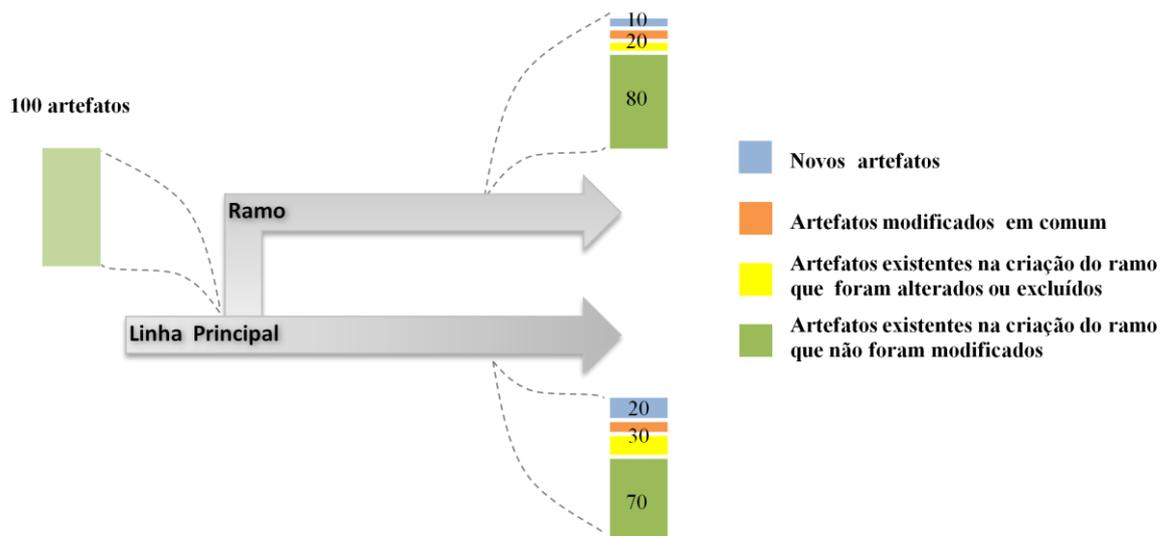


Figura 25: Exemplo da métrica Quantidade de Artefatos Modificados em Comum.

Para entendimento das duas próximas métricas são introduzidos os conceitos de precisão (do inglês, *precision*) e cobertura (do inglês, *recall*) (BAEZA-YATES; RIBEIRO-NETO, 1999; VAN RIJSBERGEN, 1979), amplamente utilizado na área de recuperação de informação. Com o propósito de simplificar a assimilação desses conceitos, a Figura 26 apresenta um exemplo de uma busca realizada na Internet de um determinado assunto, onde cada ponto representa um resultado. O círculo do lado esquerdo (com borda cinza claro) contém todos os resultados recuperados (REC) pela busca. O círculo do lado direito (com borda cinza escuro) contém todos os resultados relevantes (REL) do assunto pesquisado existentes na Internet. A interseção entre os dois círculos contém os resultados relevantes recuperados (colorido de vermelho). É visto que, de todos os resultados recuperados, nem todos atendem ao critério de busca (coloridos de cinza claro), e nem todos os resultados corretos são encontrados pela busca (coloridos de cinza escuro).

A precisão consiste na análise da corretude dos dados recuperados por uma busca, ou seja, o quão eficaz foi a busca em recuperar somente resultados relevantes. Ela é composta

pela relação entre os resultados relevantes recuperados sobre todos os resultados recuperados, cuja fórmula é:

$$\text{Precisão} = (\text{REC} \cap \text{REL}) / \text{REC} \quad (1)$$

Por outro lado, a cobertura consiste na análise da completude dos dados recuperados pela busca, ou seja, o quão eficaz foi a busca em recuperar todos os resultados relevantes. Ela é composta pela relação entre os resultados relevantes recuperados sobre todos os resultados relevantes, cuja fórmula é:

$$\text{Cobertura} = (\text{REC} \cap \text{REL}) / \text{REL} \quad (2)$$

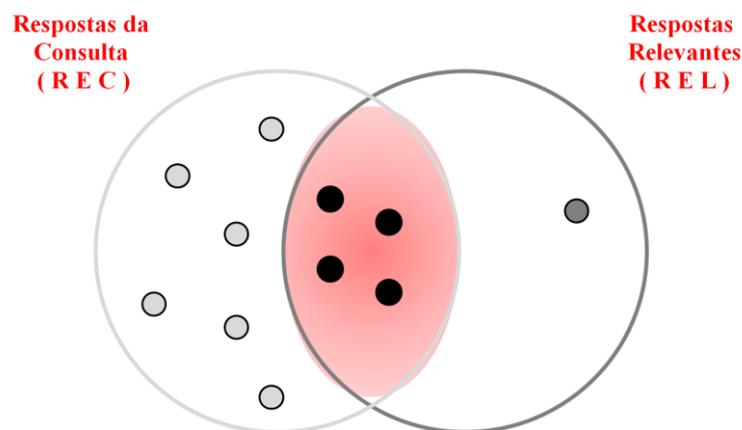


Figura 26: Exemplo de resultados de uma busca realizada na internet de um determinado assunto.

Ao aplicar a equação 1 no exemplo da Figura 26, a precisão é de 40%, que corresponde ao número de resultados da interseção (4), dividido pelo total de resultados retornados pela consulta (10). Ao aplicar a equação 2 no mesmo exemplo, a cobertura é de 80%, que corresponde ao número de resultados da interseção (4), dividido pelo total de resultados existentes relevantes (5).

Realizando uma transposição desse conceito para a abordagem, duas métricas foram criadas: **Precisão por Quantidade de Artefatos** e **Cobertura por Quantidade de Artefatos**. Essas métricas visam somente perceber qual o percentual de similaridade entre a linha principal e o ramo, realizando uma equiparação com os resultados relevantes e recuperados usados no conceito de precisão e cobertura. Elas consistem em considerar os artefatos da linha principal como sendo todos os resultados relevantes da busca, e os artefatos do ramo como

sendo todos os resultados recuperados da busca. Os artefatos não alterados tanto na linha principal quanto no ramo são correspondentes aos dados recuperados corretamente pela busca. Desta forma, é possível quantificar a aderência entre a linha principal e o ramo considerando ambas as métricas. Para exemplificá-las é mostrado na Figura 27, que 50 artefatos existentes na criação do ramo não foram alterados nem na linha principal nem no ramo, correspondendo ao conjunto interseção ($REC \cap REL$) da Figura 26. O total de artefatos da linha principal corresponde ao conjunto de resultados relevantes (REL), e o total de artefatos do ramo corresponde ao conjunto de resultados consultados (REC). Portanto, o resultado da métrica Precisão por Quantidade de Artefatos é igual a $(50/110) \approx 45\%$, ou seja, tem-se a certeza que 45% da linha principal não interferirá na junção. Por outro lado, o resultado da métrica Cobertura por Quantidade de Artefatos é igual a $(50/120) \approx 42\%$, ou seja, tem-se a certeza que 42% do ramo não interferirá na junção.

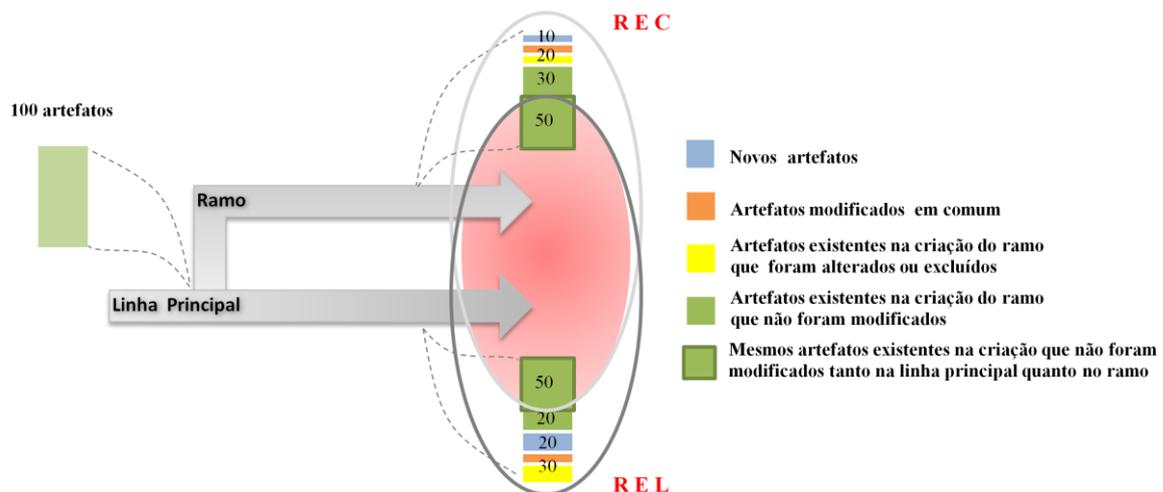


Figura 27: Exemplo da métrica Cobertura/Precisão por Quantidade de Artefatos.

A métrica **Quantidade de Linhas Diferentes na Linha Principal** apresenta uma visão geral, no nível de linhas de código, de tudo o que foi modificado nos artefatos da linha principal desde a versão em que foi realizada a criação do ramo. Sua medição se dá através da contabilização, para cada artefato, do número de linhas de código que há diferentes (inclusões, alterações e exclusões) em relação à versão do artefato na criação do ramo. Como exemplo de medição, analisando a Figura 28, a linha de código “Z” foi incluída e a linha de código “D” foi excluída na versão do arquivo *prog.java* na linha principal, portanto o valor da métrica é 2, somente considerando este arquivo. Para contabilização da métrica, esse cálculo é realizado para todos os artefatos e realizado o somatório dos resultados.

A métrica **Quantidade de Linhas Diferentes no Ramo** é similar à métrica Quantidade de Linhas Diferentes na Linha Principal, porém aplicada ao ramo. Considerando

novamente o exemplo da Figura 28, as linhas de código “X” e “Y” foram incluídas e a linha de código “E” foi excluída na versão do arquivo *prog.java* no ramo, portanto o valor da métrica é 3. Essas duas métricas reduzem a granularidade da comparação entre a linha principal e o ramo em relação às métricas anteriores, proporcionando maior detalhamento para o desenvolvedor da diferença entre eles.

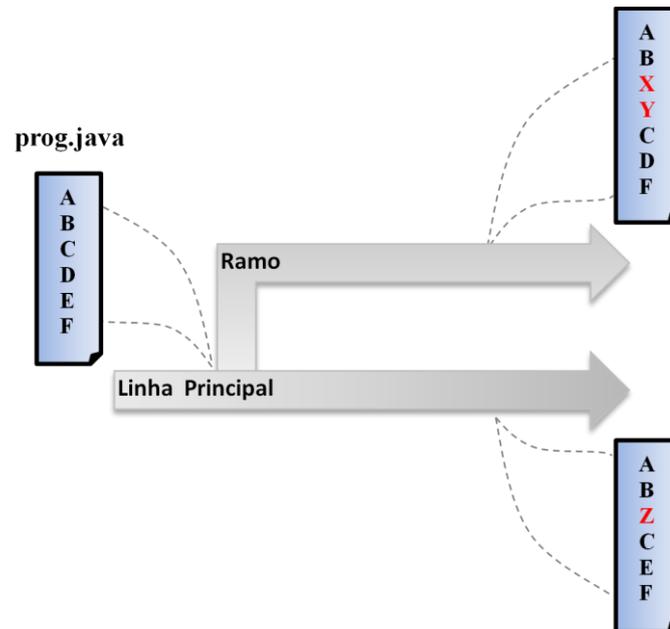


Figura 28: Exemplo da métrica Quantidade de Linhas Diferentes.

3.3.2 MÉTRICAS COM ANÁLISE DE CONFLITOS

As métricas Quantidade de Conflitos Físicos, Sintáticos e Semânticos são extraídas analisando o resultado da simulação da junção dos ramos. Uma visão geral de como é realizada essa análise pode ser visualizada através da Figura 29. Como a junção não poderia ser efetivamente feita no repositório que está sendo analisado, é feito um *check-out* do ramo que receberá a junção para uma pasta temporária (passo 1), e é realizada a junção do ramo desejado nesta pasta temporária (passo 2), de onde serão extraídas as métricas (passo 3).

No exemplo mostrado na Figura 29, foi realizado o *check-out* da linha principal de desenvolvimento (*Mainline*) para a pasta temporária. Posteriormente, foi feita a junção do ramo 1.1, contido na pasta dos ramos (*Branches*), com o conteúdo da pasta temporária, ou seja, simulando a junção do ramo 1.1 com a linha principal de desenvolvimento. Com isso, analisando a pasta, são extraídas as métricas descritas nesta seção. Como a abordagem proposta neste trabalho não interfere na rotina de desenvolvimento, esta pasta temporária é removida logo após o cálculo das métricas.

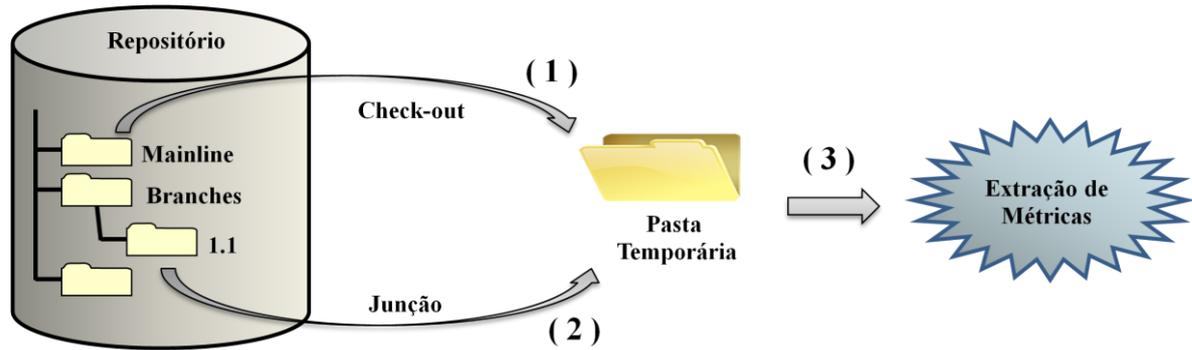


Figura 29: Extração de métricas com análise da junção dos ramos.

A métrica **Quantidade de Conflitos Físicos** contabiliza os locais dos artefatos em que não foi possível realizar a junção de forma automática. Com isso, o desenvolvedor tem a percepção da quantidade de vezes que terá que interceder para resolver os conflitos físicos para finalizar a execução da junção.

Para exemplificar as métricas Quantidade de Conflitos Sintáticos e Semânticos, foi construído um diagrama de classes (Figura 30). Neste diagrama é possível visualizar as classes: *Transformacoes*, responsável por realizar transformações ou conversões de temperaturas, onde $tCelsius$ é a temperatura em Celsius e K é uma constante utilizada para transformar Celsius em Kelvin; *LeiDoGasIdeal*, uma classe responsável por calcular o volume segundo a Lei do Gás Ideal, onde R é a constante dos gases perfeitos, p é a pressão, n é a quantidade de mols do gás e $tCelsius$ é a temperatura em Celsius; e *Testes*, que é um caso de teste que verifica o cálculo do volume conforme a Lei do Gás Ideal (MENEZES, 2011). O conteúdo dos arquivos *Transformacoes* e *LeiDoGasIdeal* da versão antes da criação do ramo pode ser visto na Figura 31.

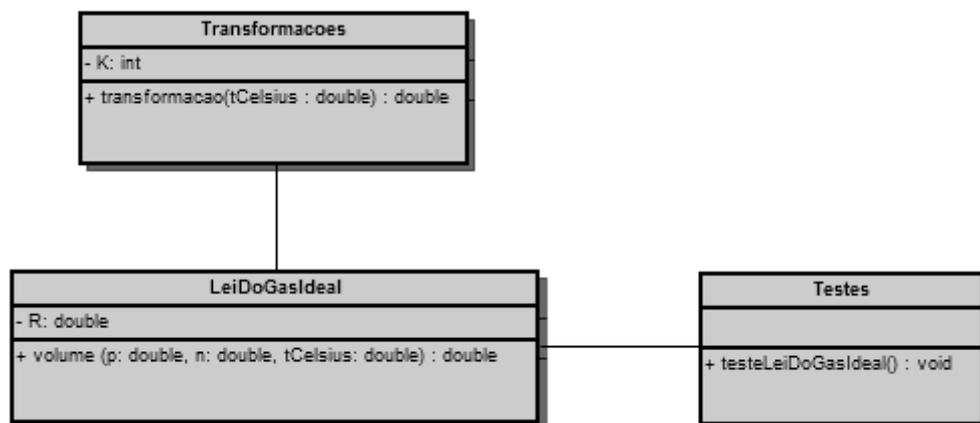


Figura 30: Diagrama de classe para exemplificar as métricas (MENEZES, 2011).

```

1 public class Transformacoes {
2     public static final int K = 273;
3     public static double transformacao(double tCelsius){
4         return tCelsius + K;
5     }
6 }

```

```

1 public class LeiDoGasIdeal {
2     public final double R = 8.314472;
3     public double volume(double p, double n, double tCelsius){
4         double tKelvin = Transformacoes.transformacao(tCelsius);
5         return n * R * tKelvin / p;
6     }
7 }

```

Figura 31: Conteúdo dos arquivos da versão da linha principal antes da criação do ramo. Adaptada de Menezes (2011).

A métrica **Quantidade de Conflitos Sintáticos** contabiliza a quantidade de erros de compilação existente. Para que essa métrica possa ser extraída, a métrica Quantidade de Conflitos Físicos tem que ser igual a zero.

Como exemplo de ocorrência do conflito sintático, os arquivos do exemplo da Figura 31 são evoluídos. Um ramo é criado a partir desta versão desses arquivos, então o desenvolvedor na linha principal altera o método *transformacao* para *transformar* na classe *Transformacoes*, e ajusta a referência para esse método na classe *LeiDoGasIdeal*. A versão no repositório após a realização do *check-in* na linha principal é a mostrada na Figura 32.

```

1 public class Transformacoes {
2     public static final int K = 273;
3     public static double transformar(double tCelsius){
4         return tCelsius + K;
5     }
6 }

```

```

1 public class LeiDoGasIdeal {
2     public final double R = 8.314472;
3     public double volume(double p, double n, double tCelsius){
4         double tKelvin = Transformacoes.transformar(tCelsius);
5         return n * R * tKelvin / p;
6     }
7 }

```

Figura 32: Exemplo de conflito sintático (versão na linha principal). Adaptada de Menezes (2011).

No ramo, outro desenvolvedor não está ciente dessas modificações ocorridas na linha principal e cria em paralelo uma classe *Testes* (Figura 33), que utiliza o método alterado. Ao

ser feita a junção do ramo para a linha principal, a classe *Testes* será incorporada a linha principal. Conseqüentemente, nenhum conflito físico ocorrerá, mas para realizar o cálculo da métrica Quantidade de Conflitos Sintáticos, o código será compilado após a junção e será contabilizado um conflito sintático, devido à classe *Testes* usar um método não mais existente na linha principal.

```

1 public class Testes {
2     @Test
3     public void testeLeiDoGasIdeal () {
4         double oraculo = 296;
5         double delta = 0.000001;
6         assertEquals(oraculo, Transformacoes.transformacao(23), delta);
7     }
8 }

```

Figura 33: Exemplo de conflito sintático (versão no ramo).
Adaptada de Menezes (2011).

Por fim, quando as métricas Quantidade de Conflitos Físicos e Sintáticos têm valor zero, é possível realizar a análise da métrica **Quantidade de Conflitos Semânticos**. De forma análoga, ela contabiliza a quantidade de testes que falham na sua execução.

Como exemplo de ocorrência do conflito semântico, é considerada a evolução dos mesmos arquivos da Figura 31 utilizada como exemplo do conflito sintático, onde não há a classe de teste criada. Um ramo é criado a partir da versão desses arquivos, então o desenvolvedor na linha principal altera na classe *Transformacoes* a ação do método *transformacao*, fazendo o *check-in* da versão ilustrada na Figura 34. Essa alteração muda o comportamento do método *transformação*, que passou a transformar uma temperatura de graus Celsius para Fahrenheit, ao invés de transformar de Celsius para Kelvin.

```

1 public class Transformacoes {
2     public static double transformacao(double tCelsius){
3         return 5 * (tCelsius - 32) / 9;
4     }
5 }

```

Figura 34: Exemplo conflito semântico (versão na linha principal).
Adaptada de Menezes (2011).

Da mesma forma da ocorrência do conflito sintático, outro desenvolvedor, trabalhando no ramo, não está ciente das modificações ocorridas na linha principal e cria um caso de teste com a classe *Testes* (Figura 35). Esta classe verifica o comportamento do método *volume* esperando a transformação de Celsius para Kelvin. Ao ser feita a junção do ramo para a linha principal, a classe *Testes* será incorporada a este ramo. Conseqüentemente, nenhum conflito

físico ou sintático ocorrerá, mas para realizar o cálculo da métrica Quantidade de Conflitos Semânticos, todos os casos de teste são executados após a junção e é contabilizado um conflito semântico.

```

1 public class Testes {
2     @Test
3     public void testeLeiDoGasIdeal () {
4         LeiDoGasIdeal leiGasIdeal = new LeiDoGasIdeal ();
5         double oraculo = 9744.561184;
6         double delta = 0.000001;
7         assertEquals(oraculo, leiGasIdeal.volume(25, 100, 20), delta);
8     }
9 }

```

Figura 35: Exemplo conflito semântico (versão no ramo) (MENEZES, 2011).

Uma situação importante a considerar é a ocorrência de conflitos semânticos nas próprias classes de teste, após a junção. Se o código que esta classe estiver testando não contiver conflitos, ocorrerá um falso positivo, ou seja, o teste falhará mesmo com o código estando correto. Porém, caso o código testado também contenha um conflito semântico e por coincidência um conflito anular o outro, nada será reportado e ocorrerá um falso negativo, onde o código testado contém um conflito que não foi computado, pois a classe que está testando esse código também está com problema. A Tabela 2 detalha as possíveis situações.

Tabela 2: Tabela verdade da ocorrência de conflitos semânticos no código e nas classes de teste.

Conflito semântico no código	Conflito semântico nas classes de teste	Comentário sobre o resultado
Falso	Falso	Não há problema com o resultado (verdadeiro negativo)
Falso	Verdadeiro	Ocorrência de falso positivo
Verdadeiro	Falso	Não há problema com o resultado (verdadeiro positivo)
Verdadeiro	Verdadeiro	Possível ocorrência de falso negativo

3.4 ESTRATÉGIA DE RAMIFICAÇÃO

Como visto na Seção 2.5, as estratégias de ramificação diferenciam entre si de acordo com o objetivo, possuindo comportamentos díspares para momentos e condições de criação e junção dos ramos. Uma característica que influencia diretamente a abordagem é a direção em que será realizada a junção. A direção pode ser da linha principal para o ramo, do ramo para a linha principal, ou em ambos os sentidos. Apenas essa informação sobre a direção da junção bastaria para que fosse aplicada a abordagem. Contudo, registrar o conhecimento do propósito

do ramo traz benefícios para padronização da utilização deste ramo. Com isso, a extração das métricas é influenciada diretamente pela escolha da estratégia de ramificação aplicada ao ramo. Como a extração das métricas Quantidade de Artefatos Modificados em Comum, Quantidade de Conflitos Físicos, Sintáticos e Semânticos independem do sentido em que são analisadas, a estratégia de ramificação não tem influência na análise destas métricas.

As estratégias contempladas pela abordagem Polvo são apresentadas na Figura 36, destacando os possíveis sentidos esperados de junção.

O Polvo calcula o esforço da junção considerando os possíveis sentidos de junção para cada estratégia de ramificação definida para o ramo. No sentido linha principal para o ramo, quando a estratégia de ramificação escolhida for *Em Cascata* ou *Por Personalizações*; no sentido ramo para a linha principal, quando for escolhida a estratégia *Em Série*, *Por Componentes*, *Por Requisições* ou *Por Terceirização*; e em ambos os sentidos, quando a estratégia for *Por Desenvolvedor* ou *Por Subprojetos*.

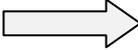
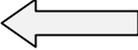
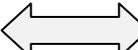
ESTRATÉGIA DE RAMIFICAÇÃO		SENTIDO DA JUNÇÃO	
Em Cascata Por Personalizações	L I N H A P R I N C I P A L		R A M O
Em Série Por Componentes Por Requisições Por Terceirização			
Por Desenvolvedor Por Subprojetos			

Figura 36: Estratégias de Ramificação utilizadas no Polvo.

3.5 CONSIDERAÇÕES FINAIS

Nos projetos que fazem uso de SCV, quando há criação de ramos e estes, por algum motivo, precisam sofrer junção, o desenvolvedor não tem condição de avaliar quão custoso será essa tarefa. Esse ramo pode estar há muito tempo sendo evoluído em paralelo à linha principal, e sua junção ser bastante complexa.

Em auxílio ao desenvolvedor no sentido de apoiar a análise do estado dos ramos, foi proposta a abordagem Polvo. Conforme descrito neste capítulo, a abordagem proposta

consiste em extrair métricas dos ramos, considerando sua estratégia de ramificação, de modo a dar subsídios para tomada de decisões. A extração das métricas realizada pela abordagem não interfere no trabalho realizado pelo desenvolvedor no dia-a-dia, podendo este acessar o repositório normalmente para realização de *check-ins* e *check-outs*.

A abordagem flexibiliza a forma de estudar os ramos, onde pode ser analisado diretamente um ramo específico ou ter uma visão geral da análise de todos os ramos. Metáforas visuais são utilizadas no auxílio para entendimento dos dados extraídos. Além disso, um ramo pode ter sua evolução no tempo analisada desde a sua criação até o momento atual. A implementação e a utilização do Polvo são detalhadas no próximo capítulo.

CAPÍTULO 4 – IMPLEMENTAÇÃO E UTILIZAÇÃO DO POLVO

4.1 INTRODUÇÃO

A abordagem Polvo foi implementada em um ambiente, denominado Oceano, que reúne módulos desenvolvidos pelo GEMS (Grupo de Evolução e Manutenção de Software). O Oceano é uma aplicação web que possui uma infraestrutura para apoiar soluções de interesses relacionados à GCS, como cadastro de projetos de software e cadastro de usuários para acesso a projetos restritos, dentre outras funcionalidades.

Atualmente, esse ambiente possui três módulos, além do Polvo. O primeiro módulo, denominado Peixe-Espada, utiliza o tempo ocioso das máquinas para refatorar código fonte visando à melhoria dos atributos de qualidade do software em questão (SANTOS, H. K., 2011). O segundo módulo, denominado Ostra, apresenta um apoio para que gerentes de projeto e equipes de desenvolvimento possam tomar decisões mais precisas, baseada em mineração de indicadores e métricas. O terceiro módulo, denominado Ouriço, amplifica o ciclo tradicional de GCS, verificando se os artefatos de software enviados para o repositório possuem conflitos físicos, sintáticos ou semânticos, antes que cheguem de fato no repositório (MENEZES, 2011).

Neste capítulo é detalhada a implementação do Polvo, que é o resultado deste trabalho de mestrado. Para viabilizar sua concretização foram tomadas algumas decisões de projeto. A versão atual do protótipo implementado faz uso do Subversion (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2008)(ROONEY, 2004) tanto para identificar os ramos quanto para extrair as métricas relacionadas a artefatos, linhas de código e conflitos físicos. Além disso, o Maven (O'BRIEN *et al.*, 2008) é necessário para a extração das métricas Quantidade de Conflitos Sintáticos e Quantidade de Conflitos Semânticos. O Maven é uma ferramenta para gerenciamento e automação da construção de projetos. Assim, ele apoia na execução de tarefas pré-definidas como parte de processos de compilação, empacotamento de código e testes, por exemplo. A utilização de Subversion e Maven foi escolhida por ser uma combinação bastante utilizada por empresas de desenvolvimento de software e também por projetos *open source*. Tal escolha permitiu que as avaliações, discutidas no Capítulo 5, fossem realizados devido à disponibilidade de projetos que utilizam esses sistemas e, conseqüentemente, puderam ser utilizados.

O restante deste capítulo está organizado com as seguintes seções: na Seção 4.2 é dada uma visão geral do Polvo e do Oceano; na Seção 4.3, são discutidos alguns pontos do Oceano

que são importantes para entendimento da abordagem; na Seção 4.4, é apresentado o Polvo com o detalhamento das formas de cálculo do esforço da junção. Além disso, é apresentado o detalhamento das funcionalidades do Polvo com exemplos de telas da abordagem; e, finalmente, na Seção 4.5, são feitas as considerações finais relativas à implementação.

4.2 VISÃO GERAL DO OCEANO E POLVO

Antes de discutir a abordagem Polvo, é necessário o entendimento de alguns aspectos do Oceano. Primeiramente, para acessar as funcionalidades do Polvo, assim como as das outras abordagens, o desenvolvedor deverá ser cadastrado e autenticar-se. Se a autenticação ocorrer com sucesso, o desenvolvedor terá acesso à tela inicial do Oceano, apresentada na Figura 37, onde aparecem em destaque as funcionalidades do Polvo.

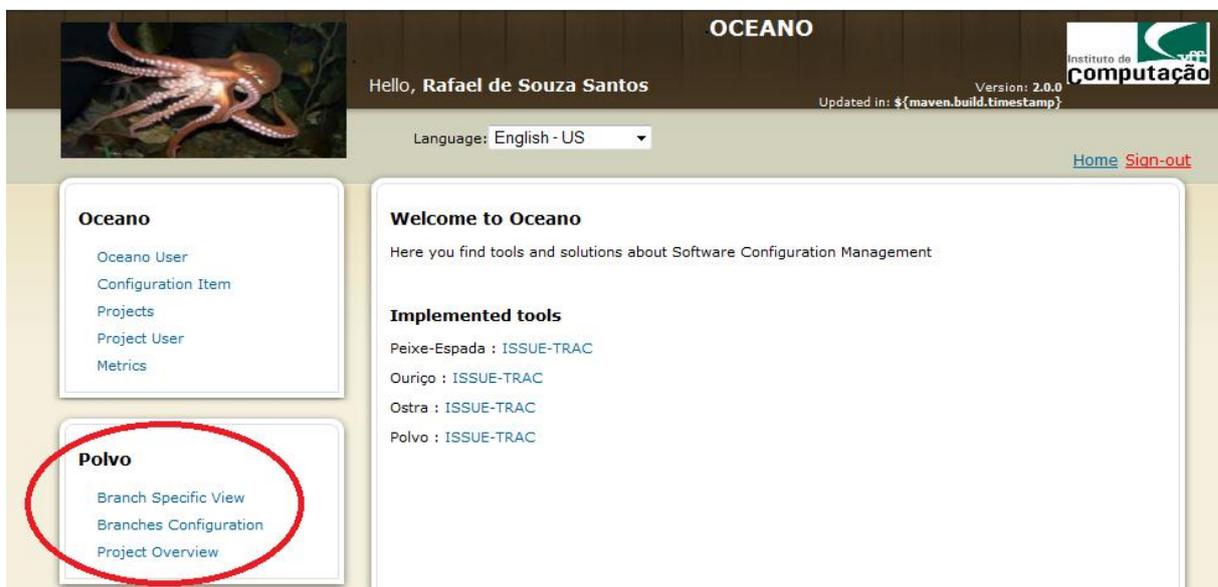


Figura 37: Página inicial do Oceano.

O diagrama de classes persistentes resumido do Oceano e do Polvo possui a modelagem representada na Figura 38. O lado esquerdo do diagrama representa as classes do Oceano utilizadas pela abordagem, e o lado direito representa as classes utilizadas propriamente pelo Polvo. Estão representadas as principais classes do Oceano necessárias para a abordagem e que dão suporte às funcionalidades do Oceano citadas neste capítulo. Mais detalhes podem ser consultados em Menezes (2011) e Santos (2011).

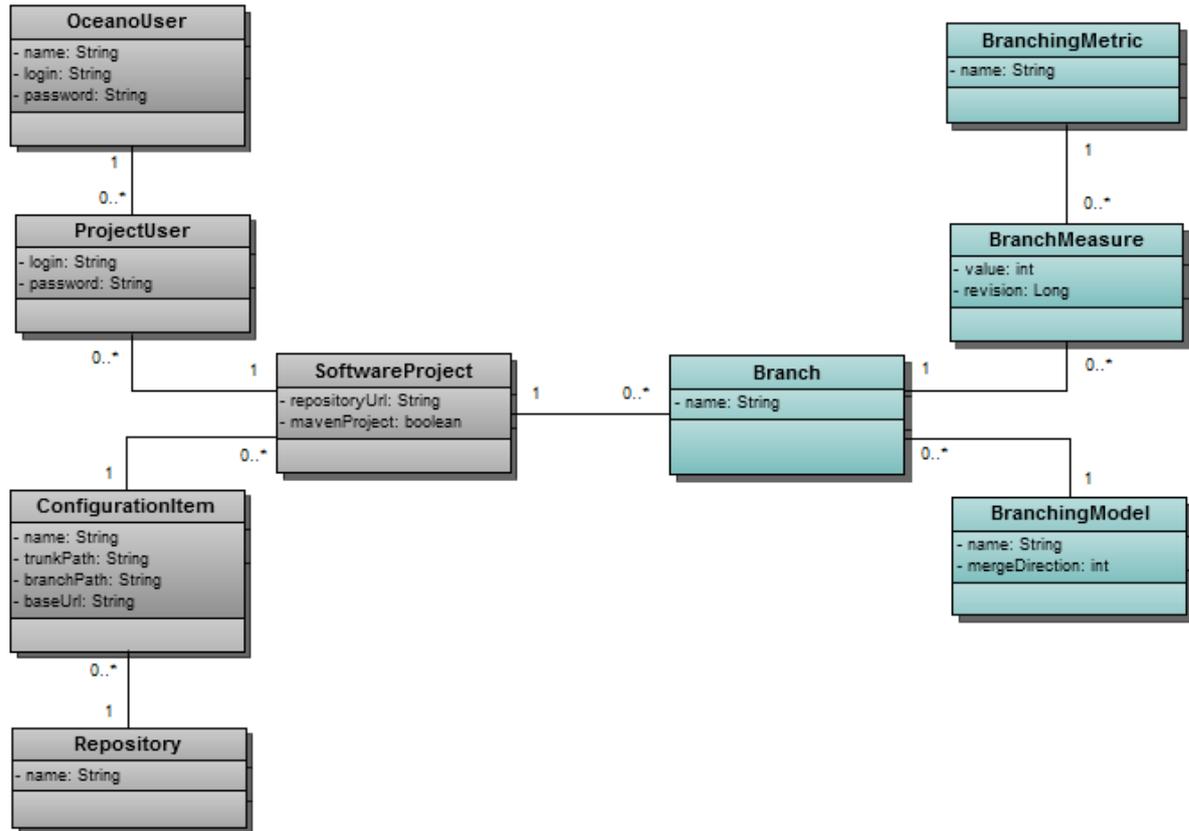


Figura 38: Diagrama das classes persistentes resumido do Oceano e do Polvo.

No Oceano é possível realizar toda a parte administrativa de organização dos projetos e suas dependências. Os projetos analisados pelo Polvo e pelas outras abordagens são cadastrados através de funcionalidades do Oceano, que atualmente suportam apenas projetos armazenados no SCV Subversion. As informações dos projetos são persistidas nas classes *SoftwareProject*, *ConfigurationItem* e *Repository*. As principais informações cadastradas utilizadas pelo Polvo são: a url onde está armazenado o projeto (exemplo: <https://sel.ic.uff.br/svn/oceano/polvo>); o local dentro do projeto onde está localizada a linha principal de desenvolvimento, normalmente denominado *trunk*; e o local onde estão os ramos, normalmente denominado *branches*. É informado também se o projeto obedece à estrutura do Maven. O Maven utiliza uma estrutura conhecida como *Project Object Model* (POM), que descreve as dependências do projeto e é representada fisicamente por um arquivo denominado *pom.xml*.

Outra funcionalidade do Oceano consiste em permitir que projetos com acesso restrito possam ser acessados com o cadastro de *login* e senha para acesso ao repositório, persistidos na classe *ProjectUser*. O *login* e a senha são associados ao usuário do Oceano, persistidos na classe *OceanoUser*, onde somente este terá acesso ao repositório do projeto para realizar análise de acordo com o propósito da abordagem utilizada. Portanto, para que um

desenvolvedor possa usar as funcionalidades do Polvo, é fundamental o cadastro dessas informações no Oceano. O Oceano possui outras funcionalidades (SANTOS, H. K., 2011), entretanto, o foco destas está fora do escopo deste trabalho.

Em relação às classes do Polvo, a classe *Branch* é responsável por persistir os ramos de um projeto de software. Ela possui relacionamento com as classes *BranchingModel*, *BranchMetric* e *SoftwareProject*. A classe *SoftwareProject* é responsável por persistir dados de um projeto de software no Oceano. A classe *BranchingModel* é responsável por persistir as estratégias de ramificação de cada ramo. Ela possui o atributo *mergeDirection*, que define o sentido da junção para a estratégia de ramificação, com valor “MAINLINE_TO_BRANCH” definindo sentido linha principal para ramo, “BRANCH_TO_MAINLINE” definindo sentido ramo para linha principal e “MAINLINE_BRANCH” para ambos os sentidos. A classe *BranchMetric* contém os valores para as métricas aplicadas ao ramo de acordo com o relacionamento que possui com a classe *BranchingMetric*, responsável por persistir as métricas de ramificação utilizadas no Polvo para avaliação do esforço da junção.

4.3 TECNOLOGIAS UTILIZADAS NA IMPLEMENTAÇÃO

No que tange a implementação desta abordagem, foram utilizadas algumas tecnologias que são discutidas nesta seção. A implementação da abordagem possui interface web, onde foi utilizada a linguagem de programação Java, rodando sobre o servidor de aplicação Glassfish (HEFFELFINGER, 2007). Foram utilizados os *frameworks* JSF (*JavaServer Faces*) (GEARY; HORSTMANN, 2004), Facelets (ARANDA; WADIA, 2008) e Rich Faces (KATZ, 2008) para interface com o usuário e JPA (*Java Persistence API*) (YANG, 2010) no tratamento dos dados persistentes, utilizando o padrão de projetos DAO (*Data Access Object*) (NOCK, 2003) na camada de acesso aos dados. O SVNKit (TMATE SOFTWARE, 2011), ferramenta Java que implementa uma API de acesso ao Subversion, foi utilizado para viabilizar o acesso programático ao Subversion. As bibliotecas Treebolic (BOU, 2011) e JFreeChart (GILBERT; MORGNER, 2011) foram utilizadas, respectivamente, para apoiar no desenho dos grafos e gráficos.

A implementação do Polvo encontra-se inserida nos módulos do Oceano. Estes módulos são: o *oceano-core*, onde se encontram as classes de negócio e as classes responsáveis pela persistência dos dados; e o *oceano-web*, módulo responsável pela interface com o usuário.

4.4 UTILIZAÇÃO DO POLVO

Conforme discutido no Capítulo 3, a intenção do Polvo é proporcionar ao desenvolvedor a visualização do esforço de junção de ramos. Para alcançar tal objetivo, duas formas são disponibilizadas pela abordagem para extração das métricas desejadas: uma a partir da escolha de um ramo específico e outra a partir da visão geral do projeto como um todo. Na Figura 39 é exibido o diagrama de atividades com o detalhamento das duas formas propostas pela abordagem.

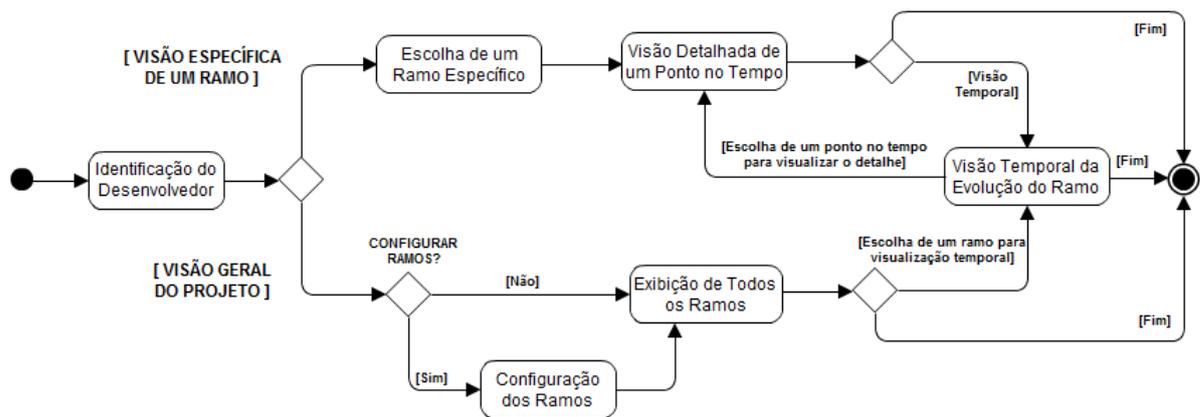


Figura 39: Diagrama de atividades do Polvo.

Após o desenvolvedor se identificar no Oceano para usar o Polvo, as duas formas disponibilizadas pela abordagem para extração das métricas estão disponíveis para análise dos ramos do repositório. Elas são independentes uma da outra, podendo o desenvolvedor fazer uso delas individualmente para determinado propósito.

Nas seções a seguir são detalhadas as duas formas disponibilizadas pela abordagem para extração das métricas.

4.4.1 VISÃO ESPECÍFICA DE UM RAMO

No exemplo do fluxo seguindo pelo caminho da **Visão Específica de um Ramo**, destacado de vermelho na Figura 40, após o desenvolvedor se identificar, há a atividade *Escolha de um Ramo Específico*. Nesta atividade o desenvolvedor escolhe a linha principal do projeto a ser analisado para comparar com um ramo. Deve ser informada também a métrica desejada, a estratégia de ramificação adotada para este ramo e, opcionalmente, a versão do repositório que deseja consultar. Caso a versão não seja informada, é mostrada a versão atual.

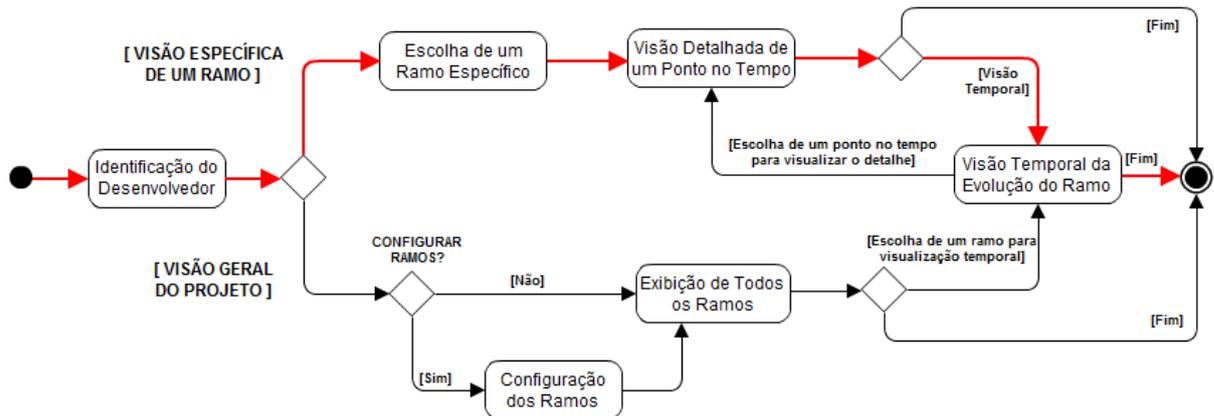


Figura 40: Diagrama de Atividades – Visão Específica de um Ramo.

A tela em execução da atividade *Escolha de um Ramo Específico* é mostrada na Figura 41. Na seleção do projeto, são disponibilizados os que foram cadastrados na funcionalidade de Cadastro de Projeto do Oceano. Após o projeto ser selecionado, é disponibilizada a linha principal de desenvolvimento (*trunk*) e todos os ramos deste projeto para serem escolhidos como linha principal e ramo. São disponibilizadas para escolha as estratégias de ramificação (como detalhado na Seção 3.4) e as métricas (como detalhado na Seção 3.3). Todos os campos são obrigatórios para que seja solicitada a comparação da linha principal com o ramo.

Branch Specific View

Project:
forrest - file:///C:/repositorio/asf/forrest/trunk

Mainline:
trunk

Branch:
locationmap_branch

Branching Strategy:
By Subprojects

Metric:
Recall by Amount of Artifacts(%)

Revision:

Action

Figura 41: Exemplo Visão Específica do Ramo.

Como exemplo, foi escolhido um ramo do projeto *Apache Forrest*, que é um *framework* que transforma entradas de várias origens em uma apresentação unificada com um ou mais formatos de saída. Para a linha principal foi escolhido o *trunk* e o *locationmap_branch* foi escolhido como o ramo para comparação utilizando a estratégia de

ramificação *Por Subprojetos*. Além disso, foi selecionada a métrica *Cobertura por Quantidade de Artefatos*.

Após a solicitação da comparação, o Polvo, de posse dessas informações, aplica a métrica escolhida sobre a versão atual do repositório, considerando a estratégia de ramificação para definir a direção da análise da junção, ou seja, em ambos os sentidos. O resultado do cálculo da métrica é apresentado na atividade *Visão Detalhada de um Ponto no Tempo*.

A tela resultante é apresentada na Figura 42, onde, além das informações selecionadas para o cálculo do esforço da junção, é exibida a versão dos artefatos no repositório e a quantidade total de artefatos da linha principal e do ramo. Como foi selecionada a estratégia de ramificação *Por Subprojetos*, utilizada para isolar membros de um mesmo projeto, onde é possível a realização da junção nos dois sentidos, a métrica *Cobertura por Quantidade de Artefatos (%)* selecionada é calculada no sentido linha principal para o ramo, cujo valor é 61%, e no sentido ramo para linha principal, cujo valor é 85%. O cálculo dessa métrica em um sentido é correspondente ao cálculo da métrica *Precisão por Quantidade de Artefatos (%)* no sentido oposto.

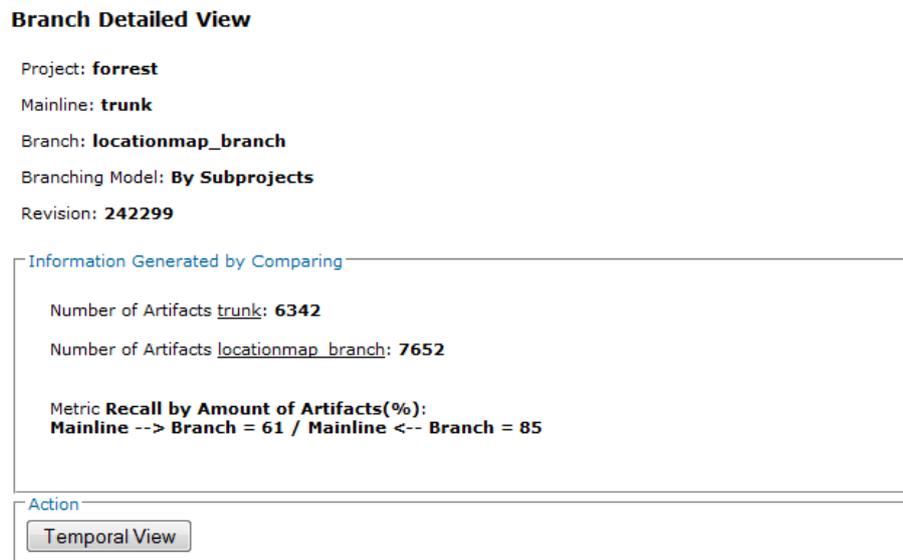


Figura 42: Visão Detalhada de um Ramo.

Após a consulta do resultado, existe a opção de visualizar a evolução do esforço da junção da linha principal com o ramo no decorrer do tempo. Na atividade *Visão Temporal da Evolução do Ramo* é mostrado um gráfico com a evolução do cálculo da métrica em dez intervalos regulares (valor parametrizado) de mesma distância entre as versões desde a criação do ramo. Caso a estratégia de ramificação deste ramo possibilite a junção nos dois sentidos, o gráfico mostrará duas linhas distintas com a evolução para cada sentido da junção. A funcionalidade *Visão Temporal do Ramo* tanto pode ser acionada pelo botão *Visão*

Temporal presente na funcionalidade *Visão Detalhada do Ramo* quanto na *Visão Geral do Ramo* (como é apresentado na Seção 4.4.2) com a escolha de um ramo do grafo.

Um exemplo da tela em execução dessa atividade é mostrado na Figura 43. É apresentada a evolução no tempo do esforço da junção entre a linha principal (*trunk*) e o ramo *locationmap_branch* do projeto *Apache Forrest* desde a sua criação (aproximadamente na versão 180.000) usando a métrica *Cobertura por Quantidade de Artefatos (%)*. Como a estratégia de ramificação propicia a junção nos dois sentidos, a evolução da junção no sentido linha principal para o ramo está representada na cor vermelha e a evolução da junção no sentido ramo para linha principal está representada na cor azul.

Branch Temporal View

Project: **forrest**

Mainline: **trunk**

Branch: **locationmap_branch**

Branching Model: **By Subprojects**

Metric: **Recall by Amount of Artifacts(%)**

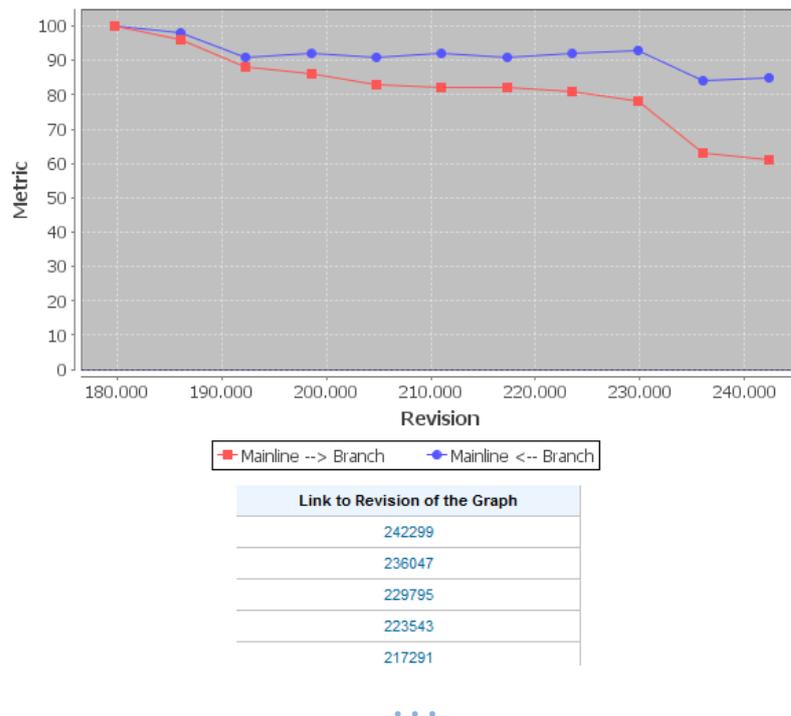


Figura 43: Visão Temporal do Ramo.

É possível observar no gráfico da Figura 43 que a junção no sentido linha principal para o ramo é mais complexa de acordo com a métrica em questão. Na versão 242.299, último ponto medido do gráfico, o valor para a métrica da linha principal para o ramo é 61%, e do ramo para a linha principal é 85%, conforme apresentado na visão detalhada do ramo (Figura 42). Nos pontos destacados do gráfico da Figura 43, podem ser consultadas as informações

detalhadas das outras versões, conforme *links* disponibilizados abaixo do gráfico, que redirecionam para as suas respectivas visões detalhadas.

4.4.2 VISÃO GERAL DO PROJETO

Outro fluxo é via **Visão Geral do Projeto**, destacado na Figura 44. Nesta, primeiramente há a atividade *Configuração dos Ramos*, em que o desenvolvedor opta por informar a estratégia de ramificação de cada ramo de um projeto antes de analisá-los na atividade de *Exibição de Todos os Ramos*. Essa atividade não é obrigatória, e caso não sejam informadas, é considerada como *default* para cada ramo a estratégia de ramificação *Por Requisições*, por ser a mais comumente utilizada.

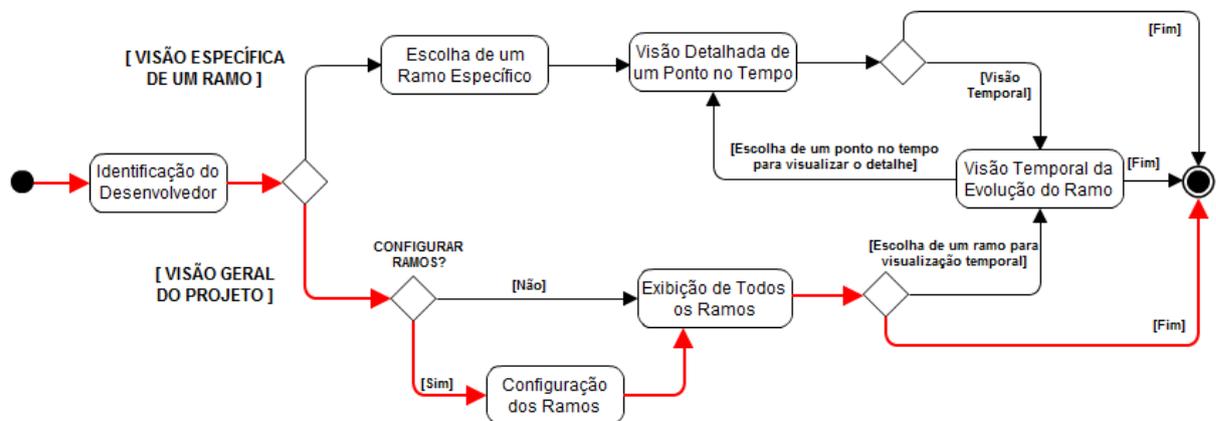


Figura 44: Diagrama de Atividades – Visão Geral do Projeto.

Na Figura 45 é mostrada a tela em execução dessa atividade. Após a escolha do projeto pelo desenvolvedor, os ramos são carregados e, para cada ramo o desenvolvedor deve informar a estratégia de ramificação adotada em sua criação. No exemplo da Figura 45, após a seleção do projeto *Apache HTTP Server (httpd)*, que consiste no servidor web Apache, seus ramos são listados. Após o desenvolvedor informar a estratégia para cada um dos ramos, essas informações são utilizadas no decorrer da funcionalidade *Exibição de Todos os Ramos*, como detalhado a seguir.

Branches Configuration

Project:
 httpd/httpd - file:///C:/repositorio/asf/httpd/httpd/trunk

Branches

Names	Branching Strategy
2.2.x	In Series
proxy-reqbody	By Requests
fips-dev	By Requests
listen-protocol	By Requests
simple-conf	By Requests

Action

Figura 45: Configuração dos Ramos.

Com a conclusão da configuração dos ramos do projeto, na atividade *Exibição de Todos os Ramos*, o desenvolvedor seleciona o projeto e a métrica desejada e o Polvo realiza o cálculo do esforço para cada ramo deste projeto em relação à linha principal que o originou. Finalizado o cálculo para todos os ramos, eles são apresentados em forma de grafo. Os ramos são coloridos baseado na escala de cores *Traffic Light* (DIEHL, 2007), que usa a cor verde para representar um estado seguro, cor amarela para representar um estado de possível perigo e cor vermelha para representar um estado de perigo efetivo. Para isso, os ramos são separados em quatro grupos de igual tamanho, conforme exibido na Figura 46, onde os limites inferior e superior são respectivamente o menor e maior valor calculado de acordo com a métrica escolhida. Após essa divisão, os ramos são classificados como: criticidade muito baixa (ramos coloridos em verde); criticidade baixa (ramos coloridos em amarelo); criticidade média (ramos coloridos em laranja); e criticidade alta (ramos coloridos em vermelho). É importante notar que esta classificação é em relação aos ramos do projeto sob análise, não significando necessariamente que tenham o esforço de junção elevado, mas alertando ao desenvolvedor que os ramos com maior criticidade são os que têm maior potencial de dificuldade de junção em relação aos demais ramos do projeto.

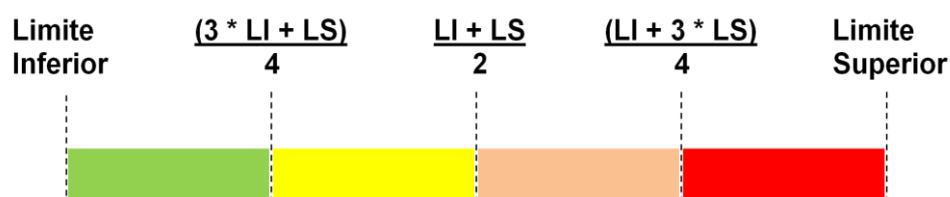


Figura 46: Classificação da criticidade dos ramos.

Na Figura 47 é mostrado um exemplo da tela em execução dessa atividade. O preenchimento da métrica e do projeto são obrigatórios. Após a solicitação de geração do gráfico, a abordagem realiza o cálculo do esforço da junção para cada ramo. No exemplo da Figura 47, há a escolha da métrica *Precisão por Quantidade de Artefatos (%)* e do projeto `httpd`, onde foi gerado o grafo com cada nó representando a linha principal e os ramos. O nó central representa a linha principal de desenvolvimento (no exemplo, foi escolhido o `trunk`) e como os demais ramos foram originados da linha principal de desenvolvimento, eles são representados por nós ligados a este nó central, ou seja, ramos originados do `trunk`. Como pode ser observado, as arestas entre o nó central e os nós que representam os ramos são coloridas de acordo com a notação de cores previamente detalhada.

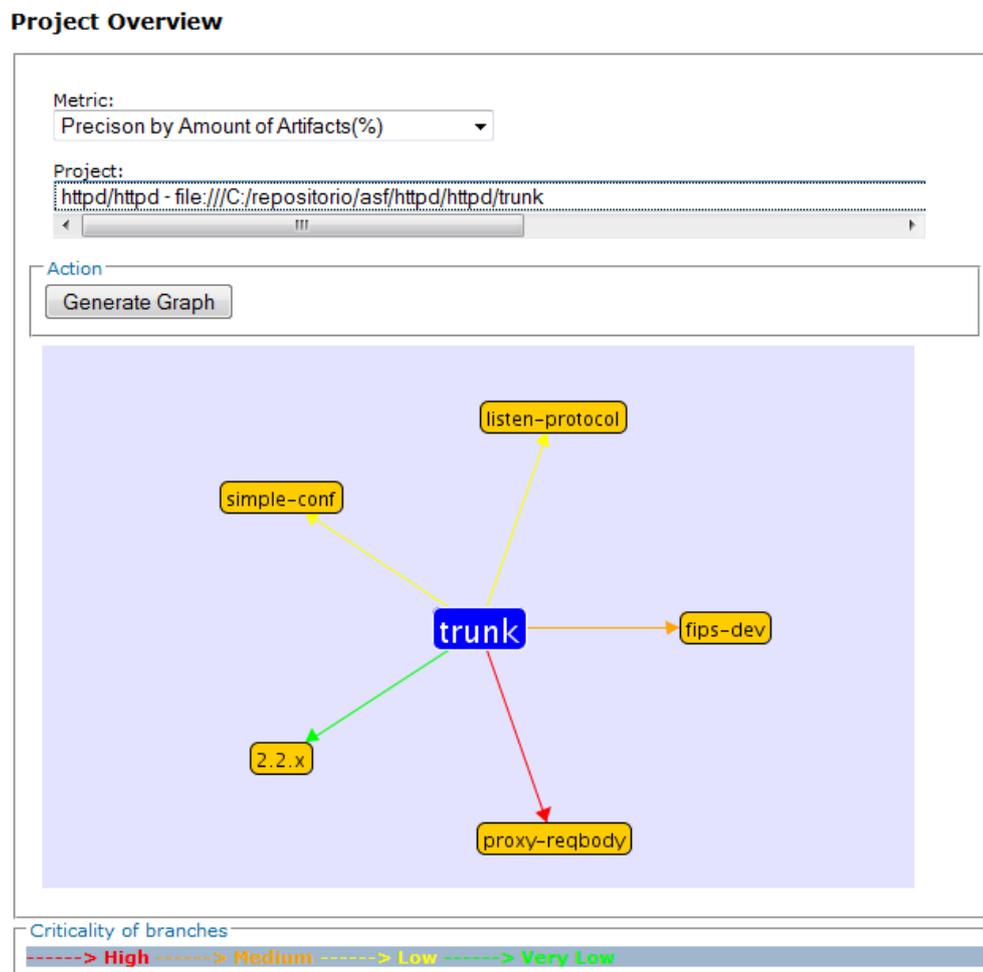


Figura 47: Visão Geral dos Ramos do projeto Apache HTTPD.

O ramo *proxy-reqbody* é o que possui menor valor para esta métrica (65%), sendo, no caso, o de maior criticidade, pois quanto menor a precisão do ramo em relação à linha principal, maior será a dificuldade para a realização da junção. O ramo *fips-dev*, com valor

78%, foi classificado como criticidade média. Os ramos *simple-cont* e *listen-protocol*, com valores intermediários, de 82% e 85%, respectivamente, foram classificados com criticidade baixa e o ramo 2.2.x, com valor de 95%, foi classificado como criticidade muito baixa.

Caso algum ramo tivesse sido originado de outro ramo, o ramo origem seria considerado linha principal, e a seta indicaria o sentido de criação do ramo. Na Figura 48 é apresentado um pedaço da visão geral do repositório do projeto *Apache Subversion*, com o uso da métrica *Quantidade de Artefatos Modificados em Comum*, com o ramo 1.6.x possuindo 7 ramos originados a partir dele.

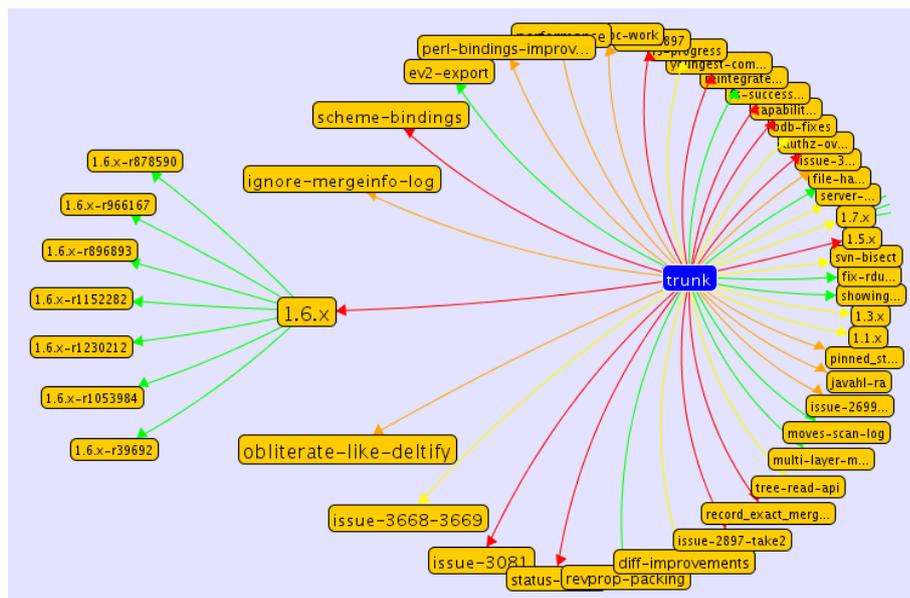


Figura 48: Visão Geral dos Ramos do projeto Subversion.

A partir da visualização de todos os ramos, o desenvolvedor tem a opção de escolher um ramo específico para visualizar informações da evolução da métrica no tempo (atividade *Visão Temporal da Evolução do Ramo*), comum ao fluxo detalhado na Seção 4.4.1.

4.5 CONSIDERAÇÕES FINAIS

O Polvo apresenta uma implementação que auxilia o desenvolvedor, em qualquer momento do projeto, a ter uma visão específica de um ramo ou a ter uma visão geral de todos os ramos em relação ao esforço para realização da junção. Métricas e estratégias de ramificação dão apoio a essa avaliação, permitindo ao desenvolvedor ter uma visão dos ramos mais críticos.

No próximo capítulo é apresentado um estudo experimental que visa avaliar o Polvo em ação, sendo utilizado sobre projetos reais.

CAPÍTULO 5 – AVALIAÇÃO EXPERIMENTAL DO POLVO

5.1 INTRODUÇÃO

Como visto anteriormente, o Polvo tem o objetivo de auxiliar os desenvolvedores na análise do esforço de junção de ramos em SCV. Para isso, disponibiliza métricas como forma de quantificar esse esforço. Neste capítulo são descritas as avaliações realizadas com o intuito de responder à questão de pesquisa introduzida no Capítulo 1 deste trabalho: “É viável prever com precisão o esforço de junção de ramos em SCV a partir de análise automática sobre o repositório?”. Com o objetivo de avaliar essa questão de pesquisa, foi realizada uma comparação das medidas obtidas pelo Polvo com o esforço real de junção dos ramos. .

Para responder a essa questão, foi conduzida uma avaliação em ambiente controlado, como *in-vitro* (TRAVASSOS; BARROS, 2003). Essa decisão está ligada à conveniência para obtenção de repositórios de projetos *open-source*, comparado à dificuldade de inserção desta abordagem em uma organização de desenvolvimento de software antes de se ter indícios sobre a sua viabilidade. Contudo, as restrições atuais do protótipo implementado sobre os projetos alvo (uso de Subversion e Maven) restringiu o espaço de opções dos projetos a serem analisados. Além disso, os projetos deveriam possuir ramos que tivessem sido efetivamente utilizados e posteriormente juntados à linha principal ou vice-versa.

O restante deste capítulo está organizado com as seguintes seções: na Seção 5.2 são descritos os projetos e seus ramos utilizados para a avaliação com detalhamento de suas características; na Seção 5.3, é discutido o método utilizado para realizar a avaliação; na Seção 5.4, é discutida a aplicação do método; na Seção 5.5, são apresentados os resultados e as suas análises; na Seção 5.6, são discutidas as ameaças à validade da avaliação; e, finalmente, na Seção 5.7, são feitas as considerações finais relativas à avaliação.

5.2 PROJETOS E SEUS RAMOS UTILIZADOS

Alguns critérios foram seguidos para a escolha dos projetos e seus ramos para utilização na avaliação. O universo inicial de projetos somente contemplava os que estavam armazenados em repositório Subversion. O primeiro critério utilizado foi considerar somente projetos que possuíam ramos. Ao final desta etapa de verificação, foram encontrados alguns repositórios contendo projetos diversos e constatado que o repositório ASF da Apache possuía uma série de projetos passíveis de serem utilizados pela abordagem.

Numa segunda etapa de verificação, considerando somente os projetos com ramos previamente selecionados pela primeira etapa, foi possível constatar que alguns deles não possuíam uma estrutura de diretório organizada segundo as boas práticas do Subversion (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2008). Isso dificulta a aplicação da abordagem, fazendo com que esses projetos fossem descartados da avaliação.

De posse destes projetos, foi realizada uma terceira etapa de verificação, que visava identificar quais projetos utilizavam o Maven, caracterizados pela presença do seu arquivo descritor denominado *pom.xml*. Durante a execução dessa busca, foram identificados alguns projetos que atendiam a essas restrições, mas seus ramos ainda precisariam passar por outros critérios para serem utilizados como objetos desta avaliação.

Dos projetos resultantes da etapa anterior, uma última etapa consistiu em identificar se existia algum ramo que atendesse às seguintes condições: ser possível identificar a versão que o ramo sofreu alguma junção, para que fosse possível realizar o cálculo do valor do esforço real de junção; se utilizava o Maven nesse momento da junção; ser possível identificar a versão base de criação do ramo, onde cabe ressaltar a grande dificuldade de extrair essa informação do histórico do repositório, devido a alguns terem sido criados há muito tempo e armazenados, na época, com versões antigas do Subversion, que possuíam esquema de metadados diferente (o uso de metadados para rastreamento de junção dos ramos foi introduzido no Subversion a partir da versão 1.5); e, ter um número de *check-ins* significativo, pois há casos de ramos com muito pouca ou nenhuma modificação em relação à sua origem. A listagem completa dos projetos descartados seguindo os critérios de exclusão é mostrada no Apêndice A. Finalmente, no final dessa etapa, foram obtidos dos projetos da Tabela 3 alguns ramos de alguns projetos satisfatórios para a avaliação. Esses projetos possuem perfis variados quanto ao número de desenvolvedores, linhas de código e ramos. Deste modo, chegou-se a um conjunto de projetos composto por sistemas acadêmicos e *frameworks* utilizados no mercado. Eles estão caracterizados na Tabela 3, que foi montada com base em estatísticas obtidas através da ferramenta StatSVN².

²<http://www.statsvn.org>

Tabela 3: Caracterização dos Projetos.

Projeto	Desenvolvedores	Linhas de Código	Número de Ramos	Data Inicial (mês/ano)	Última Modificação (mês/ano) ³
Process Broker	2	17.627	3	11/2009	05/2010
Oceano-Core	6	42.258	3	08/2010	12/2011
Jakarta Cactus	11	106.651	13	04/2001	07/2007
Apache Cocoon	56	327.633	7	02/2003	05/2006

O primeiro dos projetos é o *Process Broker*. Ele consiste em uma infraestrutura para intercâmbio de componentes de processo entre instituições implementadoras de processos de software, e foi desenvolvido pelo IC/UFF em parceria com a COPPE/UFRJ. Esse projeto é de acesso restrito e foi retirado do repositório armazenado no seguinte endereço: <https://sel.ic.uff.br/svn/processbroker>.

Como visto no Capítulo 4, o projeto *Oceano-Core* é o módulo principal do Oceano, que serviu de base para a implementação do Polvo. Ele consiste em uma biblioteca de apoio a pesquisas em manutenção e evolução de software em desenvolvimento pelo IC/UFF. Esse projeto também é de acesso restrito e foi retirado do repositório armazenado no seguinte endereço: <https://sel.ic.uff.br/svn/oceano/oceano-core>.

O projeto *Jakarta Cactus* é um *framework* de código livre para realização de testes unitários em código Java no lado servidor (*Servlets, EJBs, Tag Libs, Filters, etc.*). Seu objetivo é baixar o custo de escrever testes para o desenvolvimento no lado servidor. Esse projeto foi retirado do repositório armazenado no seguinte endereço: <http://svn.apache.org/repos/asf/jakarta/cactus>.

O projeto *Apache Cocoon* é um *framework* de código livre para desenvolvimento *web* construído em torno dos conceitos de separação de interesses e desenvolvimento baseado em componentes *web*. Esse projeto foi retirado do repositório armazenado no seguinte endereço: <http://svn.apache.org/repos/asf/cocoon>.

Os ramos utilizados na avaliação desses projetos são caracterizados na Tabela 4, onde são mostradas informações do número de desenvolvedores que realizaram algum *check-in* neste ramo, número de *check-ins* realizados no ramo, *data* de criação e junção do ramo e a estratégia de ramificação empregada.

³A coluna “Última Modificação” da Tabela 3 representa a última modificação feita na linha principal até a coleta do repositório.

Tabela 4: Caracterização dos Ramos.

Projetos / Ramos	Desenvolvedores	Número de Check-ins	Data de Criação (mês/ano)	Data da Junção (mês/ano)	Estratégia de Ramificação
Oceano-Core					
(R1) Oceano-Core-Peixe-Espada	1	9	02/2011	12/2011	Por Subprojetos
Process Broker					
(R2) r967	2	4	02/2010	05/2010	Por Terceirização
Apache Cocoon					
(R3) BRANCH_2_1_X	42	2.673	05/2004	05/2006	Em Série
(R4) BRANCH_2_1_X-dojo1_1	2	36	08/2008	11/2008	Por Requisições
Jakarta Cactus					
(R5) CACTUS_17_BRANCH	1	17	07/2005	01/2006	Por Requisições
(R6) CACTUS_15_BRANCH	1	61	07/2003	10/2003	Por Requisições
(R7) CACTUS_14_ANT_BRANCH	2	113	04/2003	05/2003	Por Requisições
(R8) CACTUS_13_BRANCH	1	13	04/2002	06/2002	Por Requisições
(R9) CACTUS_TRUNK_MAMOUTH	1	132	06/2007	02/2008	Por Requisições

Analisando a Tabela 4 é possível observar que o ramo R3, pertencente ao projeto Apache Cocoon, possuiu um grande número de desenvolvedores atuando sobre ele e teve o maior tempo de duração entre a criação até a realização da junção: dois anos. Além disso, o ramo R4, pertencente ao mesmo projeto, foi originado a partir do R3 e é o único dos ramos analisados que tem origem de outro ramo, diferentemente dos demais que tiveram origem da linha principal. Por fim, os ramos utilizados dos projetos do repositório ASF da Apache (Cocoon e Cactus) foram considerados com estratégia de ramificação *Em Série* (ramo R3) e *Por Requisições* (ramos R4 a R9) baseado na nomenclatura dos ramos e análise dos comentários das versões.

Para facilitar a realização da avaliação, os projetos foram replicados para um repositório local através do programa *svnsync* (ROONEY, 2004). Este programa permite copiar o histórico completo de desenvolvimento de um repositório remoto para um repositório local. A realização da cópia é feita versão a versão, com leitura do repositório remoto e escrita no repositório local, onde, após sua conclusão, o repositório local passa a ser idêntico ao remoto no momento da execução do programa. Para que novas alterações realizadas no repositório remoto possam ser replicadas para o repositório local, o programa *svnsync* precisa ser executado novamente.

A replicação para um repositório local através do programa *svnsync* permitiu maior agilidade para execução das avaliações. Entretanto, nem sempre foi possível utilizá-lo devido às limitações impostas pelos servidores que hospedam tais repositórios. Um exemplo dessas limitações acontece no repositório ASF da Apache, onde alguns projetos são muito grandes, com um número elevado de versões, e ao tentar realizar a replicação, que dura um longo período de tempo, o servidor Apache identifica que é a mesma máquina fazendo o acesso, e bloqueia a conexão.

Não há empecilhos de executar a avaliação diretamente no repositório remoto, porém, com a utilização da cópia local, há um aumento da confiabilidade, já que a dependência da rede é neutralizada, não ficando sujeito a alguma instabilidade para acessar o repositório. Também foi alcançada uma significativa redução do tempo de execução das avaliações. Isso aconteceu devido à redução do tempo de trafegar os dados pela rede. Outra opção seria, havendo a possibilidade de acessar a máquina que hospeda o repositório remoto, configurar a abordagem Polvo para rodar nessa máquina, ou numa outra máquina na mesma rede local. Isso também permitiria maior agilidade para execução das avaliações.

5.3 MÉTODO UTILIZADO

O desenvolvedor que deseja utilizar o Polvo, na maioria das vezes, tem interesse em avaliar os ramos atualmente em desenvolvimento para obter informações do esforço da junção. Porém, para avaliar o auxílio provido pela abordagem proposta, foi necessário obter o valor real do esforço de junção da linha principal com o ramo ou vice-versa, e comparar com os valores preditos pelas métricas.

A obtenção do valor real do esforço de junção (EJ), conforme proposto em trabalhos anteriores (PRUDÊNCIO *et al.*, 2012), considera que um artefato foi modificado em um ramo concomitantemente com outras modificações na linha principal. Considerando a junção no sentido ramo para a linha principal, o EJ para esse artefato consiste no número de ações extras (adições, remoções ou modificações sobre os artefatos) que o desenvolvedor precisou fazer durante a junção para compatibilizar as modificações feitas no ramo com as modificações feitas em paralelo na linha principal. Essas ações podem ser decorrentes de quaisquer demandas que surgiram durante a efetuação da junção, como, por exemplo, unificação de estilos de programação, resolução de conflitos, refatorações, etc. Com isso, é possível obter o EJ ao analisar o histórico do repositório e combinar os esforços individuais de junção dos artefatos do projeto. A Figura 49 ilustra este cenário, onde a identificação dos pontos destacados é necessária para realização do cálculo.

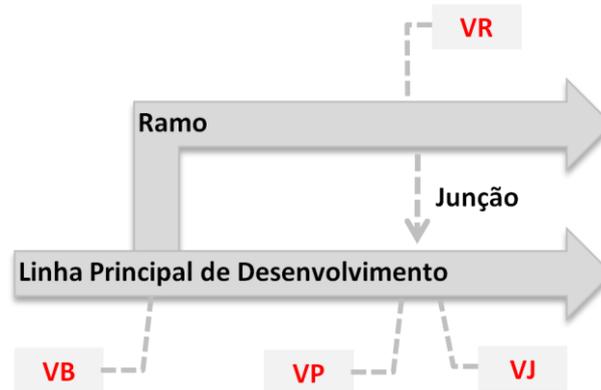


Figura 49: Versões consideradas no Esforço de Junção (EJ).

Considerando a junção realizada no sentido ramo para a linha principal, a versão a partir da qual foi criado o ramo é a versão base (VB), a versão do ramo antes da realização da junção é denominada versão do ramo (VR), e as versões da linha principal antes e depois da realização da junção são denominadas versão da linha principal (VP) e versão após a junção (VJ), respectivamente.

Como forma de tornar a avaliação passível de ser repetida, a Tabela 5 apresenta as versões utilizadas dos ramos durante a execução da avaliação. Ao analisar as versões do ramo R1, como exemplo, a versão base é a 901, ou seja, a versão onde ocorreu a criação do ramo. Para todos os ramos, a versão anterior à realização da junção, tanto para a linha principal, quanto para o ramo foi considerada como a versão da linha principal e do ramo, respectivamente. No caso do ramo R1, essa versão é a 1.116 do repositório, e a versão 1.117 é a versão gerada após a realização da junção.

Com essas quatro versões mapeadas para os ramos, é possível a identificação do conjunto de ações realizadas e do conjunto de ações efetivadas, conforme definidos a seguir.

O conjunto de ações realizadas (AR) abrange todo o desenvolvimento realizado na linha principal e no ramo desde sua criação até o último momento antes da junção. O levantamento dessas informações é feita através do uso do algoritmo de *diff3* considerando VB, VR e VP:

$$\mathbf{AR = diff3(VB, VR, VP)} \quad (3)$$

Tabela 5: Versões Utilizadas dos Ramos.

Projetos / Ramos	VB	VR	VP	VJ
Oceano-Core				
(R1) Oceano-Core-Peixe-Espada	901	1.116	1.116	1.117
Process Broker				
(R2) r967	9	23	23	24
Apache Cocoon				
(R3) BRANCH_2_1_X	398.882	410.024	410.024	410.025
(R4) BRANCH_2_1_X-dojol_1	687.721	711.583	711.583	711.584
Jakarta Cactus				
(R5) CACTUS_17_BRANCH	239.183	369.214	369.214	369.215
(R6) CACTUS_15_BRANCH	238.419	238.601	238.601	238.602
(R7) CACTUS_14_ANT_BRANCH	237.658	237.860	237.860	237.861
(R8) CACTUS_13_BRANCH	236.380	236.438	236.438	236.439
(R9) CACTUS_TRUNK_MAMOUTH	549.456	630.706	630.706	630.707

O conjunto de ações efetivadas abrange todo o desenvolvimento realizado desde a criação do ramo até a finalização da junção, ou seja, tudo que foi efetivamente aproveitado do desenvolvimento realizado em paralelo da linha principal e do ramo. O levantamento dessas informações é feito através do uso do algoritmo de *diff* (HUNT; MCILROY, 1976) considerando VB e VJ:

$$AE = \text{diff}(VB, VJ) \quad (4)$$

Na Figura 50 são mostrados os dois conjuntos de ações. A interseção dos conjuntos representa as ações realizadas que foram efetivadas na VJ e que não demandaram esforço durante a junção. As ações que foram realizadas e não efetivadas na VJ são ações descartadas, ou seja, esforço desperdiçado. Por outro lado, as ações que foram efetivadas e que não tinham sido realizadas, são ações que foram feitas para resolver conflitos na realização da junção, significando esforço extra para conciliar as alterações de VR e VP (colorido de cinza escuro). Neste caso, o valor de EJ é maior devido à necessidade desse esforço extra. Utilizando os resultados das equações 3 e 4, o EJ é dado pela seguinte equação:

$$EJ = |AE \setminus AR| \div |AE| \quad (5)$$

Onde $|X|$ representa o número de elementos do conjunto X, AR representa o conjunto de ações realizadas, AE representa o conjunto de ações efetivadas e o símbolo “\” representa a operação de diferença de conjuntos. O resultado é um valor que varia entre 0 e 1, em que quanto mais alto for o valor, maior foi o esforço necessário para a realização da junção. Em outras palavras, mais ações extras tiveram que ser feitas para compatibilizar as ações realizadas.

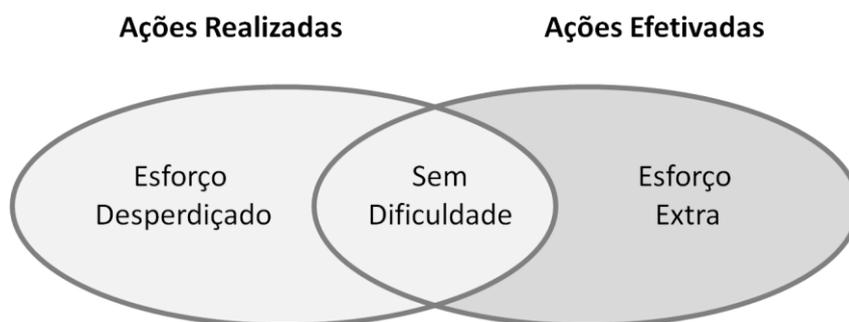


Figura 50: Conjunto de ações realizadas e efetivadas (PRUDÊNCIO *et al.*, 2012).

Portanto, seguindo o exemplo da junção do ramo sentido linha principal, o EJ deste ramo serve para mensurar o esforço a mais que foi necessário para compatibilizar as alterações realizadas no ramo com as da linha principal. O método de execução da avaliação consiste em comparar o EJ com cada um dos valores das 10 métricas propostas, obtidos no momento anterior a junção de cada ramo. Essa comparação é feita através da função de correlação de Pearson (SPIEGEL, 1994), também conhecida como coeficiente de correlação (CORREL.). Esta função indica a força e a direção do relacionamento linear entre duas variáveis aleatórias. Os resultados obtidos variam entre -1 e 1, onde 0 representa a ausência de correlação, -1 representa a perfeita anticorrelação e 1 representa a perfeita correlação. Consequentemente, a função de correlação permite identificar quais medidas indicam de forma mais precisa o esforço de junção.

5.4 APLICAÇÃO DO MÉTODO

Para cada um dos ramos listados na Tabela 4, 8 métricas da abordagem, detalhadas no Capítulo 3, foram calculadas no momento anterior à realização das junções. Desta forma, para cada uma das métricas, foram coletadas 8 medidas (uma por ramo), mostradas na Tabela 6. As métricas de Quantidade de conflitos Sintáticos e Semânticos não foram consideradas na avaliação, pois as suas medições só seriam possíveis para o ramo R4, que não possuiu conflitos físicos. O ramo R4 também não possui conflitos sintáticos, nem semânticos. Com isso, a análise isolada dessas métricas em um único ramo não permitiria o cálculo de

correlação, limitando severamente as conclusões sobre os resultados obtidos. Além do resultado das métricas, foi realizado o cálculo do EJ para os ramos e versões listados na Tabela 5, aplicando o método descrito na Seção 5.3.

Tabela 6: Resultado das métricas e do Esforço de Junção (EJ) para os ramos.

	R1	R2	R3	R4	R5	R6	R7	R8	R9
Quantidade de Artefatos Modificados na Linha Principal	139	65	5	89	36	35	129	69	18
Quantidade de Artefatos Modificados no Ramo	136	142	4.709	129	34	35	293	60	794
Quantidade de Linhas Diferentes na Linha Principal	6.232	4.317	520	1.529	1.829	1.288	11.179	3.369	2.203
Quantidade de Linhas Diferentes no Ramo	3.796	15.042	718.106	7.558	1.607	1.667	32.880	2.009	108.180
Cobertura por Quantidade de Artefatos (%)	48	74	35	99	98	98	40	88	26
Precisão por Quantidade de Artefatos (%)	84	78	45	99	98	98	59	93	33
Quantidade de Artefatos Modificados em Comum	124	50	1	39	29	28	97	54	0
Quantidade de Conflitos Físicos	23	5	1	0	7	9	1	2	0
Esforço de Junção (EJ) (x 100)	0,22	0,06	0,003	0	0,07	0,06	0,006	0,03	42,8

Na análise dos dados da Tabela 6, o ramo R4 do projeto Apache Cocoon obteve EJ com valor zero, ou seja, não demandou ação extra para a concretização da junção. Outra informação que se destaca é a discrepância do valor do EJ para o ramo R9 comparado com os demais ramos. Com uma análise mais detalhada sobre o que aconteceu no cálculo do EJ deste ramo, foi percebido que houve uma movimentação de diretório no momento da junção, que foi interpretada como a remoção e adição de diversos artefatos. Com isso, na realização do cálculo do EJ, essas adições e remoções de artefatos foram consideradas individualmente como ações extras, ocasionando o alto valor do EJ. Como o método de cálculo do EJ não é capaz de identificar movimentações da forma que foi implementado, esse ramo foi considerado como um *outlier*⁴. Em virtude disso, este ramo não foi considerado na análise dos resultados da avaliação.

⁴ Em Estatística, um *outlier* é um dado numericamente distante quando comparado aos demais dados de um conjunto observado (SPIEGEL, 1994).

5.5 ANÁLISE DOS RESULTADOS

Na Tabela 7 são apresentados os resultados das métricas de Quantidade de Artefatos Modificados na Linha Principal e no Ramo. Os resultados foram anotados com coloração baseada na escala de cores *Traffic Light*, a mesma utilizada para colorir a classificação da criticidade dos ramos (Figura 47). As cores auxiliam na visualização da ordenação dos valores, onde a cor vermelha indica os piores valores dentro de sua escala, e a cor verde os melhores valores também dentro de sua escala. As demais cores são de valores intermediários, variando do vermelho até o verde. Vale notar que a escala é referente a cada métrica individualmente, representada nas linhas da tabela. Por exemplo, o valor 139 é colorido com a cor vermelha para a métrica M1, porém o valor 142, superior ao anterior, é colorido com a cor amarela para a métrica M2. Pela ausência de normalização entre as métricas e o EJ, a percepção relativa de variação dos valores, fornecida pelas cores, é mais relevante que o valor absoluto em si.

Tabela 7: Correlação das Métricas Quantidade de Artefatos Modificados na Linha Principal (M1) e no Ramo (M2).

	R1	R2	R3	R4	R5	R6	R7	R8	CORREL.
M1	139	65	5	89	36	35	129	69	→ 0,43
M2	136	142	4709	129	34	35	293	60	→ -0,31
EJ	0,22	0,06	0,003	0	0,07	0,06	0,006	0,03	

Como pode ser observado, o resultado da métrica M1 para o ramo R1 tem alta correlação com o resultado do EJ (ambos são vermelhos), ou seja, estão entre os piores valores medidos nas suas escalas. De forma análoga, o resultado da métrica M1 para o ramo R3 também tem alta correlação em relação ao resultado do EJ (ambos são verdes), pois estão entre os melhores valores nas suas escalas. Por outro lado, a métrica M2 obtém a sua pior medida no ramo R3, que, por sua vez, tem um dos melhores valores de EJ, fazendo com que diminua a correlação entre eles.

Na análise do resultado total da correlação para essas duas métricas, conclui-se que a Quantidade de Artefatos Modificados na Linha Principal está mais correlacionada com o EJ do que a medida feita para o ramo. Porém, observando-as individualmente, é possível constatar que não são boas métricas para predizer o EJ, o que é esperado, pois elas estão medindo o que está divergindo na linha principal ou no ramo individualmente, sem considerar o que é comum entre eles. Contudo, essas métricas são importantes para quando se tem

necessidade de realizar uma avaliação da evolução da linha principal ou do ramo separadamente.

É possível chegar à mesma conclusão analisando os resultados das métricas da Quantidade de Linhas Modificadas na Linha Principal e no Ramo, detalhados na Tabela 8. Neste caso, o resultado total da correlação dessas duas métricas, que possuem granularidade fina (linha de código), é pior do que o das métricas anteriores, com granularidade grossa (artefato).

Tabela 8: Correlação das Métricas Quantidade de Linhas Diferentes na Linha Principal (M3) e no ramo (M4).

	R1	R2	R3	R4	R5	R6	R7	R8	CORREL.
M3	6232	4317	520	1529	1829	1288	11179	3369	→ 0,17
M4	3796	15042	718106	7558	1607	1667	32880	2009	→ -0,31
EJ	0,22	0,06	0,003	0	0,07	0,06	0,006	0,03	

É possível observar na Tabela 7 e na Tabela 8 que os ramos com estratégia de ramificação *Por Subprojetos*, *Por Terceirização* e *Em Série* (ramos R1, R2 e R3) se correlacionam bem com o EJ nas métricas que envolvem a linha principal (M1 e M3). Diferentemente do que acontece com os ramos com estratégia de ramificação *Por Requisições* (ramos R4 a R8), que possuem correlação ruim tanto para as métricas que envolvem a linha principal (M1 e M3) quanto para as métricas que envolvem o ramo (M2 e M4).

Na Tabela 9 são apresentados os resultados das métricas Cobertura e Precisão por Quantidade de Artefatos. No caso específico destas métricas, a coloração dos valores está invertida em relação às demais métricas, fazendo com que a cor vermelha continue indicando os piores (menores) valores dentro de sua escala, e a cor verde os melhores (maiores) valores dentro de sua escala. Isso é devido ao fato de que, quanto maior for o valor para cobertura e precisão, melhor é a aderência entre a linha principal e o ramo. Conseqüentemente, o EJ é avaliado em relação à anticorrelação, ou seja, quanto mais a correlação se aproxima de -1, menor é a dificuldade de junção. Porém, apesar delas já considerarem a interseção das modificações ocorridas na linha principal e no ramo nos seus cálculos, a correlação com o EJ está longe do ideal.

Tabela 9: Correlação das Métricas Cobertura por Quantidade de Artefatos (M5) e Precisão por Quantidade de Artefatos (M6).

	R1	R2	R3	R4	R5	R6	R7	R8	CORREL.
M5	48	74	35	99	98	98	40	88	→ -0,13
M6	84	78	45	99	98	98	59	93	→ 0,24
EJ	0,22	0,06	0,003	0	0,07	0,06	0,006	0,03	

Outra análise extraída observando os dados da Tabela 9 é que, com exceção do ramo R4 (o único ramo que tem origem a partir de outro ramo), as métricas tiveram correlação ruim com os ramos R3 e R7, aqueles que possuem os valores menores de EJ. É importante notar que o ramo R3, que teve a pior correlação com o EJ para ambas as métricas, é o único dos ramos analisados com um grande número de desenvolvedores atuando sobre ele.

Na Tabela 10 são apresentados os últimos resultados medidos da avaliação, referentes às métricas Quantidade de Artefatos Modificados em Comum e Quantidade de Conflitos Físicos. Estas métricas são as que apresentam os melhores resultados para a correlação, justamente por medirem o trabalho em comum realizado na linha principal e no ramo, tanto no nível de artefato quanto no nível de linha de código.

Tabela 10: Correlação das Métricas Quantidade de Artefatos Modificados em Comum (M7) e de Conflitos Físicos (M8).

	R1	R2	R3	R4	R5	R6	R7	R8	CORREL.
M7	124	50	1	39	29	28	97	54	→ 0,62
M8	23	5	1	0	7	9	1	2	→ 0,99
EJ	0,22	0,06	0,003	0	0,07	0,06	0,006	0,03	

Como pode ser observada, a métrica M7 possui uma boa correlação com o EJ, onde a maior discrepância acontece nas medições dos ramos R4 e R7, cujo EJ é nulo ou quase nulo. Como essa métrica foca nos artefatos modificados em comum, esses ramos tiveram alterações nestes artefatos que resultaram em nenhuma ou pouca necessidade de esforço extra para a junção. Porém, a métrica de conflito físico (M8), chega quase ao valor ideal de 100% de correlação. Isso indica que esta métrica foi totalmente alinhada com o EJ, e, com base na avaliação, ela seria a mais indicada para o desenvolvedor se basear para estimar o esforço de junções futuras.

Com uma análise geral de todos os dados extraídos, é possível notar que todas as medições do ramo R2 sempre se correlacionaram bem com o EJ. Este ramo utiliza a estratégia

de ramificação *Por Terceirização*, mas como só foi analisado este ramo para esta estratégia, é prematuro afirmar que as métricas se comportam melhor para determinada estratégia, necessitando de mais avaliações de ramos com estratégias diferentes para chegar a uma conclusão.

5.6 AMEAÇAS À VALIDADE

Com as avaliações descritas neste capítulo foi possível responder à questão proposta, em que as métricas de Quantidade de Artefatos Modificados em Comum e Quantidade de Conflitos Físicos tiveram resultados muito promissores, onde a de Conflitos Físicos alcançou quase o valor ideal de 100% de correlação com o EJ. Entretanto não é possível generalizar os resultados devido a algumas limitações que podem ameaçar a validade dos estudos realizados. Em função do pequeno número de projetos, ramos e estratégias de ramificação analisados, o suporte estatístico não viabiliza a generalização dos resultados obtidos. Independentemente disso, os resultados iniciais são indícios promissores da abordagem e motivam a execução de novas avaliações, inclusive em projetos reais na indústria.

Dos projetos utilizados, dois foram desenvolvidos em meio acadêmico, que normalmente não simulam as condições de um ambiente de desenvolvimento de software por completo (GOMES *et al.*, 2009). Com relação aos *frameworks*, há a falta de informações a respeito do motivo real de criação dos ramos, tendo gerado a necessidade de inferir a informação do tipo de estratégia utilizada. Essa inferência, apesar de baseada no nome e informações do ramo, é suscetível a erros.

Por fim, a avaliação foi realizada utilizando dados históricos (repositórios de SCV) e comparado com o EJ (calculado). Por isso, é desejável que esses resultados sejam validados pelos membros das equipes de desenvolvimento em termos de aderência à realidade. No caso dos projetos *Process Broker* e *Oceano-Core* foi possível um contato por email com os responsáveis pela junção dos ramos, com o intuito de obter uma percepção inicial nesse sentido.

O responsável pela junção do ramo *r967* do projeto *Process Broker* (R2) não participou diretamente do desenvolvimento da linha principal ou do ramo, porém possui bastante experiência com realização de junções em diferentes projetos. Para a realização dessa junção gastou pouco tempo e considerou que foi relativamente fácil, classificando com nível de dificuldade 2 numa escala de 1 a 10. O responsável mencionou também que não sabe se o tempo melhoraria caso tivesse a participação dos desenvolvedores, mas a certeza que a solução estaria correta seria maior.

No caso da junção do ramo *Oceano-Core-Peixe-Espada* do projeto *Oceano-Core* (R1), o desenvolvedor que trabalhou no ramo (Desenvolvedor 1) e um dos desenvolvedores que trabalhou na linha principal (Desenvolvedor 2) foram os responsáveis pela realização da junção. Os dois desenvolvedores possuem também experiência com realizações de junções, porém tiveram percepções diferentes em relação à dificuldade da realização desta junção. A conclusão da junção levou cerca de 3 horas, porém, o Desenvolvedor 1 classificou como dificuldade 9, numa escala de 1 a 10, devido ao fato de terem que fazer a junção sem o apoio da ferramenta de junção do Subversion, pois esta não estava reconhecendo o ancestral comum de alguns artefatos. Já o Desenvolvedor 2 classificou como dificuldade 2, isso provavelmente se deve ao fato deste desenvolvedor ter mais conhecimento sobre o que estava sendo desenvolvido na linha principal, e saber exatamente o que deveria ser migrado do ramo. O Desenvolvedor 2 relatou que fez uso da ferramenta de comparação da própria IDE (*Integrated Development Environment*) para auxiliar na identificação do que deveria ser juntado. Como os que trabalharam diretamente no desenvolvimento eram os responsáveis pela junção, os dois desenvolvedores concordam que mais pessoas envolvidas provavelmente não influenciariam na melhora do tempo de conclusão da junção.

No restante dos projetos utilizados na avaliação, esse contato com os responsáveis pela junção não foi possível. Esse tipo de validação poderia contribuir inclusive para a obtenção de dados qualitativos sobre a abordagem.

5.7 CONSIDERAÇÕES FINAIS

A avaliação do Polvo teve como objetivo identificar se a abordagem auxilia os desenvolvedores na análise do esforço de junção de ramos em SCV. Os estudos realizados mostraram que, das métricas utilizadas pela abordagem, as métricas de Quantidade de Artefatos Modificados em Comum e de Conflitos Físicos foram as que melhor tiveram correlação com o esforço necessário para a realização da junção dos ramos dos casos estudados, sendo que a métrica de Quantidade de Conflitos Físicos obteve quase 100% de correlação. Além disso, outras observações importantes foram extraídas para as demais métricas e para determinados tipos de estratégias de ramificação ou características do ramo.

A utilização de projetos reais para a realização da avaliação foi considerada um fator fundamental para a avaliação da abordagem. Durante a seleção dos projetos houve alguns empecilhos que inviabilizaram uma análise de um maior número de ramos. Além da dificuldade de encontrar repositórios com projetos que atendiam a todas as restrições da

abordagem, a maior dificuldade foi extrair do histórico dos repositórios as informações necessárias para execução da avaliação pela ausência dos metadados necessários.

O próximo capítulo apresenta as considerações finais deste trabalho, descrevendo algumas contribuições, limitações e possíveis trabalhos futuros.

CAPÍTULO 6 – CONCLUSÃO

6.1 CONTRIBUIÇÕES

Uma equipe de desenvolvimento, quando questionada sobre quais dos ramos estão divergindo acentuadamente e qual o esforço que demandará as suas junções, dará respostas possivelmente baseadas em experiências anteriores ou em palpites. A abordagem proposta neste trabalho dá subsídios para a tomada de decisões gerenciais e provê conhecimento sobre a evolução dos ramos do projeto por meio do cálculo de métricas e posterior visualização.

Este trabalho forneceu indícios sobre a viabilidade de avaliar ramos e estimar o esforço de junção por meio de métricas. De acordo com os resultados apresentados na avaliação, algumas métricas se comportaram melhor para atingir o objetivo, sendo que a métrica de Quantidade de Conflitos Físicos chegou a uma correlação de até 99% com o esforço real de junção.

Finalmente, conclui-se que este trabalho possui as seguintes contribuições objetivas:

- Abordagem para avaliação do esforço de junção de ramos em SCV;
- Uso de métricas (com 10 implementações) e estratégias de ramificação (com 8 implementações) como apoio à abordagem;
- Protótipo implementado com o ciclo completo da abordagem Polvo; e,
- Identificação, por meio de uma avaliação (com utilização de 2 projetos acadêmicos e 2 *open-source*), de métricas que representam com precisão o esforço de junção;

6.2 LIMITAÇÕES

Apesar de o Polvo atingir os seus objetivos, esta abordagem possui algumas limitações. A primeira das limitações, mencionada no Capítulo 4, é devido à implementação da abordagem estar atualmente restrita a projetos versionados pelo Subversion e compatíveis com Maven. Contudo, para a avaliação, o Maven não foi necessário devido à impossibilidade da extração das métricas Quantidade de Conflitos Sintáticos e Semânticos para os ramos utilizados. Outras limitações são discutidas a seguir.

A identificação de conflitos semânticos depende dos testes criados pelo desenvolvedor – Conforme discutido no Capítulo 3, o Polvo possui algumas métricas implementadas para estimar o esforço da junção dos ramos. Dentre as métricas, uma é

baseada na Quantidade de Conflitos Semânticos resultante da junção do ramo, onde só é permitida sua execução se não há ocorrência de conflitos físicos nem sintáticos.

A contabilização dos conflitos semânticos é obtida em verificações realizadas por um conjunto de testes desenvolvidos durante o ciclo de desenvolvimento do projeto. Com isso, a qualidade da identificação de quebras semânticas depende dos testes criados pelo desenvolvedor. Porém, isso depende de como o projeto está sendo desenvolvido, pois em alguns casos a equipe de desenvolvimento possui outras tarefas com prioridade mais alta do que a escrita dos casos de teste. Pode-se citar, como exemplo, a correção de defeitos sem a manutenção dos testes existentes ou a criação de novos testes.

Complementar a isso, uma situação importante a ser considerada é a ocorrência de conflitos semânticos nas próprias classes de teste, após a simulação da junção. Se o código que esta classe estiver testando não contiver conflitos, ocorrerá um falso positivo, ou seja, o teste falhará mesmo com o código estando correto. Porém, caso o código testado também contenha um conflito semântico e, por coincidência, um conflito anular o outro, nada será reportado e ocorrerá um falso negativo, onde o código testado contém um conflito que não foi computado, pois a classe que está testando esse código também está com problema. A versão atual da abordagem não é capaz de identificar os casos de conflitos semânticos dentro das próprias classes de testes, que possam ocorrer após a simulação da junção.

A qualidade da visualização depende do tamanho do projeto – O software alvo do Polvo pode ser de pequeno porte, envolvendo poucos desenvolvedores, ou de grande porte, sendo desenvolvidos por grandes equipes ou várias equipes de desenvolvimento, podendo haver necessidade de criação de diversos ramos. Em virtude disso, é possível citar um aspecto que pode comprometer a adoção da abordagem em sistemas de grande porte: escalabilidade da visualização. Como não há restrição da quantidade de ramos que se pode criar, o número muito elevado de ramos pode comprometer a compreensão das visualizações adotadas pela abordagem. A adoção de mecanismos de filtragem pode atenuar essa limitação.

6.3 TRABALHOS FUTUROS

O desenvolvimento da abordagem Polvo gerou algumas possibilidades para trabalhos futuros. Um trabalho futuro natural é a implementação de novas métricas para dar mais subsídios para a tomada de decisão. Uma alternativa seria propor uma métrica que levasse em consideração vários aspectos que hoje são considerados separadamente pelas métricas existentes. Outra melhoria seria a exibição do resultado do cálculo de todas as métricas numa única tela. Outro trabalho é a implementação de suporte a outras estratégias de ramificação

pela abordagem. Mais um trabalho natural seria a realização de um estudo detalhado com as equipes responsáveis pelos projetos utilizados nas avaliações, a fim de verificar se o resultado alcançado com a avaliação condiz qualitativamente com a realidade. Além disso, uma análise mais social, de quem comunicou com quem, poderia ser útil para avaliar o impacto no esforço de junção. Contudo, devido à existência da abordagem e sua implementação, é possível enumerar outros trabalhos futuros, como detalhados a seguir.

Ordem da junção e influência nas métricas – Quando o Polvo realiza a avaliação do esforço de junção entre a linha principal e o ramo, somente são considerados esses elementos nessa avaliação. Todavia, pode haver um cenário onde, por exemplo, a linha principal possua 3 ramos (R1, R2 e R3), e que a ordem de junção deles possa influenciar no cálculo do esforço da junção. Ou seja, o resultado da junção do ramo R3, pode variar se os ramos R1, R2 ou ambos são juntados à linha principal antes do ramo R3. O mesmo vale para o cálculo do esforço de junção de R1 e R2. Com isso, há uma possibilidade de trabalho futuro que alteraria significativamente o propósito da abordagem: em vez de considerar somente junções individuais de ramos, essa avaliação poderia considerar as ordens possíveis de junção, visando sugerir uma ordem de junção que minimize o esforço total.

Classificação dos ramos independentemente do projeto – Conforme visto no Capítulo 3, a classificação da criticidade dos ramos (Figura 46) é realizada em relação aos valores medidos dos ramos do projeto. Uma opção de trabalho futuro seria identificar valores de referência para cada métrica, que indicassem a criticidade independentemente do projeto.

Mecanismo proativo de monitoramento – A abordagem Polvo, para ser executada atualmente, depende da ação do desenvolvedor para extrair as informações de interesse sobre a junção de ramos. No entanto, um mecanismo proativo de monitoramento automático dos ramos poderia ser introduzido. Esse mecanismo poderia extrair as métricas e notificar a equipe sempre que algum ramo apresente valores superiores aos indicados para determinadas métricas, podendo utilizar a classificação dos ramos citada no trabalho futuro discutido anteriormente. Além disso, poderia haver, dentro de uma empresa, vários projetos sendo monitorados, com uma base de dados com informações dessas medições. Com o passar do tempo e com a análise destes dados, seria possível um estudo que possibilitaria prever o comportamento futuro dos ramos que estão em desenvolvimento em relação ao esforço da junção e avaliar padrões comportamentais das estratégias de ramificação.

Aplicação em SCV distribuídos – Conforme visto no Capítulo 2, em SCV distribuídos, os desenvolvedores trabalham em repositórios locais (clones), onde posteriormente suas alterações são integradas ao repositório remoto, levando ao surgimento

de um número elevado de ramos implícitos. Um trabalho futuro seria a aplicação da abordagem para avaliação do esforço da junção dos clones criados em relação ao repositório remoto, fazendo uma avaliação similar à que é feita nos ramos em relação à linha principal.

REFERÊNCIAS

- APPLETON, B.; BERCZUK, S.; CABRERA, R.; ORENSTEIN, R. Streamed lines: Branching patterns for parallel software development. In: IN PROCEEDINGS OF THE 1998 PATTERN LANGUAGES OF PROGRAMS CONFERENCE, 1998, Monticello, Illinois, USA: ACM, 1998.
- ARANDA, B.; WADIA, Z. *Facelets Essentials: Guide to JavaServer(TM) Faces View Definition Framework*. 1. ed. New York, NY: Apress, 2008.
- BAEZA-YATES, R.; RIBEIRO-NETO, B. *Modern Information Retrieval*. New York, NY: ACM Press, 1999.
- BERCZUK, S. Pragmatic Software Configuration Management. *IEEE Software*, v. 20, n. 2, p. 15-17, 2003.
- BERCZUK, S.; APPLETON, B. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. Boston, MA, USA: Addison-Wesley Professional, 2002. v. 1st.
- BERZINS, V. Software Changes Merge: Semantics of Combining to Programs. New York, NY: ACM Transactions on Programming Languages and Systems, 1994. v. 16. p. 1875-1903.
- BITMOVER. *BitKeeper*. Disponível em: <<http://www.bitkeeper.com>>. Acesso em: 10 out. 2011.
- BOU, B. *Treebolic a java applet for hyperbolic hendering of hierarchical data*. Disponível em: <<http://treebolic.sortilege.net/en/index.html>>. Acesso em: 10 out. 2011.
- BRUN, Y.; HOLMES, R.; ERNST M. D.; NOTKIN D. *Speculative Identification of Merge Conflicts and Non-Conflicts*. 2010. Technical Report UW-CSE-10-03-01 – University of Washington, Washington,USA, 2010.
- CANONICAL. *Bazaar*. Disponível em: <<http://bazaar-vcs.org/>>. Acesso em: 10 out. 2011.
- CHACON, S. *Pro Git*. 1. ed. Berkeley, Calif.: Apress, 2009.
- CHEN, C. *Information Visualization: Beyond the Horizon*. 2. ed. London: Springer, 2006.
- CHILOWICZ, M.; DURIS, E.; ROUSSEL, G. Finding Similarities in Source Code Through Factorization. *Electronic Notes in Theoretical Computer Science*, p. 47-62, 2009.
- COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.; PILATO, C. M. *Version Control with Subversion*. Sebastpol, CA, USA: O'Reilly Media, 2008. v. 2.
- CONRADI, R.; WESTFECHTEL, B. Version Models for Software Configuration Management. *ACM Computing Surveys*, v. 30, n. 2, p. 232-282, 1998.
- DART, S. Concepts in Configuration Management Systems. In: INTERNATIONAL WORKSHOP ON SOFTWARE CONFIGURATION MANAGEMENT (SCM), 1991, Trondheim, Norway: ACM Press, 1991. p. 1-18.

DEWAN, P.; HEGDE, R. Semi-synchronous conflict detection and resolution in asynchronous software development. *European Conference on Computer-Supported Cooperative Work (ECSCW)*, p. 159-178, 2007.

DIEHL, S. *Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software*. London: Springer, 2007.

ESTUBLIER, J. Objects Control for Software Configuration Management. In: CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING (CAISE), 2001, Interlaken, Switzerland: Springer, 2001. p. 359-373.

ESTUBLIER, J. Software Configuration Management: a Roadmap. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 2000, Limerick, Ireland: ACM, 2000. p. 279-289.

ESTUBLIER, J.; LEBLANG, D.; HOEK, A. VAN DER; CONRADI, R.; CLEMM, G.; TICHY, W.; WIBORG-WEBER, D. Impact of software engineering research on the practice of software configuration management. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, v. 14, n. 4, p. 383-430, 1 out. 2005.

FOGEL, K.; BAR, M. *Open Source Development with CVS*. Scottsdale, Arizona, USA: The Coriolis Group, 2001.

GEARY, D.; HORSTMANN, C. S. *Core JavaServer Faces*. Boston, MA: Prentice Hall, 2004.

GILBERT, D.; MORGNER, T. *JFreeChart, a free Java class library for generating charts*. Disponível em: <<http://www.jfree.org/jfreechart>>. Acesso em: 10 out. 2011.

GNU. *Comparing and Merging Files*. Disponível em: <<http://www.gnu.org/s/hello/manual/diff/Unified-Format.html#Unified-Format>>. Acesso em: 10 out. 2011.

GOMES, M.; VIANA, D.; CHAVES, L.; CASTRO, A.; VAZ, V. T.; SOARES, A.; TRAVASSOS, G. H.; CONTE, T. WDP-RT: Uma técnica de leitura para inspeção de usabilidade de aplicações Web. In: PROCEEDINGS OF THE 6TH EXPERIMENTAL SOFTWARE ENGINEERING LATIN AMERICAN WORKSHOP (ESELAW 2009), 2009, São Carlos, Brasil: v. 1, 2009. p. 124-133.

HEFFELFINGER, D. *Java EE 5 Development using GlassFish Application Server: The complete guide to installing and configuring the GlassFish Application Server and ... 5 applications to be deployed to this server*. 1. ed. Birmingham, UK: Packt Publishing, 2007.

HORWITZ, S. Identifying the Semantic and Textual Differences Between Two Versions of a Program. *Proc. SIGPLAN '90 Conf. Programming Language Design and Implementation*, p. 234-244, 1990.

HORWITZ, S.; PRINS, J.; REPS, T. Integrating non-interfering versions of programs. In: ACM TRANSACTIONS ON PROGRAMMING LANGUAGES AND SYSTEMS, 1989, New York, NY, USA: ACM, 1989. p. 133-145.

HUNT, J. W.; MCILROY, M. D. *An Algorithm for Differential File Comparison*. Computing Science Technical Report. Murray Hill, New Jersey: Bell Laboratories, 1976.

IEEE. *Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology*. . New York, NY, USA: Institute of Electrical and Electronics Engineers, 1990.

IEEE. *Std 828 - IEEE Standard for Software Configuration Management Plans*. . New York, NY, USA: Institute of Electrical and Electronics Engineers, 2005.

ILYAS, M.; KÜNG, J. A similarity measurement framework for requirements engineering. *4th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2009)*, p. 31-34, 2009.

JI, J. H.; PARK, S. H.; WOO, G.; CHO, H. G. Evolution analysis of homogenous source code and its application to plagiarism detection. *Proceedings of the Frontiers in the Convergence of Bioscience and Information Technologies (FBIT 2007)*, p. 813-818, 2007.

KATZ, M. *Practical Richfaces*. New York, NY: Apress, 2008.

LAKKARAJU, P.; GAUCH, S.; SPERETTA, M. Document similarity based on concept tree distance. *Proceedings of the 19th ACM Conference on Hypertext and Hypermedia, HT'08 with Creating'08 and WebScience'08*, p. 127-131, 2008.

LEON, A. *A Guide to Software Configuration Management*. Norwood, MA: Artech House Publishers, 2000.

LESNER, B.; BRIXTEL, R.; BAZIN, C.; BAGAN, G. A novel framework to detect source code plagiarism: Now, students have to work for real! *Proceedings of the ACM Symposium on Applied Computing*, p. 55-57, 2010.

LI, H.; LIU, C.; LIU, N.; LI, X. Method and its system of Java source and byte code plagiarism detection. *Beijing Hangkong Hangtian Daxue Xuebao/Journal of Beijing University of Aeronautics and Astronautics*, v. 36, p. 424-428, 2010.

MENEZES, G. *Ouriço: Uma Abordagem para Manutenção da Consistência em Repositórios de Gerência de Configuração*. 2011. Dissertação de Mestrado – Universidade Federal Fluminense - UFF, Niterói, RJ, 2011.

MENS, T. A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering*, v. 28, n. 5, p. 449-462, 2002.

MURTA, L. G. P. *Gerência de Configuração no Desenvolvimento Baseado em Componentes*. 2006. Tese de Doutorado – UFRJ, COPPE, Rio de Janeiro, Brasil, 2006.

MURTA, L. G. P. *Gerência de Configuração: Ramificação*. Disponível em: <<http://www.ic.uff.br/~leomurta/courses/2009.1/gc/aula4.pdf>>. Acesso em: 10 out. 2011.

NOCK, C. *Data Access Patterns: Database Interactions in Object-Oriented Applications*. 1. ed. Boston, MA, USA: Addison-Wesley Professional, 2003.

O'BRIEN, T.; CASEY, J.; FOX, B.; SNYDER, B.; ZYL, J. V.; REDMOND, E. *Maven: The Definitive Guide*. 1st. ed. Sebastopol, CA, USA: O'Reilly, 2008.

- O'SULLIVAN, B. Making sense of revision-control systems. *Communications of the ACM*, v. 52, n. 9, p. 56, 1 set. 2009a.
- O'SULLIVAN, B. *Mercurial: the definitive guide*. 1. ed. Sebastopol CA: O'Reilly Media, 2009b.
- OHNO, A.; MURAO, H. A new similarity measure for in-class source code plagiarism detection. *International Journal of Innovative Computing, Information and Control*, v. 5, p. 4237-4247, 2009.
- PRUDÊNCIO, J. G.; MURTA, L.; WERNER, C.; CEPÊDA, R. To lock, or not to lock: That is the question. *Journal of Systems and Software*, v. 85, n. 2, p. 277-289, fev. 2012.
- ROONEY, G. *Practical Subversion*. 1. ed. ;: Apress, 2004.
- ROXAS, R. E.; LIM, N. R.; BAUTISTA, N. Automatic generation of plagiarism detection among student programs. *7th International Conference on Information Technology Based Higher Education and Training (ITHET)*, p. 226-235, 2006.
- SAGER, T.; BERNSTEIN, A.; PINZGER, M.; KIEFER, C. Detecting similar Java classes using tree algorithms. *International Conference on Software Engineering*, p. 65-71, 2006.
- SANTOS, H. K. *Rumo ao rejuvenescimento automático de software guiado por atributos de qualidade*. 2011. Dissertação de Mestrado – UFF, Niterói, RJ - Brasil, 2011.
- SANTOS, R.; MURTA, L. G. P. Monitoramento da Complexidade de Junção de Ramos em Sistemas de Gerência de Configuração. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (ACEITO PARA PUBLICAÇÃO), 2012, Natal: IEEE, 2012.
- SARMA, A.; NOROOZI, Z.; VAN DER HOEK, A. Palantír: Raising Awareness among Configuration Management Workspaces. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 2003, Portland, Oregon: IEEE, 2003. p. 444-454.
- SERBAN, A. *Visual Sourcesafe 2005 Software Configuration Management in Practice*. Birmingham, UK: Packt Publishing, 2007.
- SPIEGEL, M. *Estatística*. 3. ed. São Paulo, SP: McGraw-Hill, 1994.
- TMATE SOFTWARE. *SVNKit*. Disponível em: <<http://svnkit.com/>>. Acesso em: 10 out. 2011.
- TRAVASSOS, G. H.; BARROS, M. O. Contributions of In Vitro and In Silico Experiments for the Future of Empirical Studies in Software Engineering. *Workshop on Empirical Studies in Software Engineering (WSESE)*, p. 117-130, 2003.
- VAN RIJSBERGEN, C. J. *Information Retrieval*. 2. ed. London: Butterworth, 1979.
- WALRAD, C.; STROM, D. The Importance of Branching Models in SCM. *IEEE Computer*, v. 35, n. 9, p. 31-38, 2002.
- WHITE, B. A. *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*. Boston, MA, USA: Addison-Wesley, 2000.

WIKIPÉDIA. *Polvo*. Disponível em: <<http://http://pt.wikipedia.org/wiki/Polvo>>. Acesso em: 10 out. 2011.

WLOKA, J.; RYDER, B.; TIP, F.; XIAOXIA REN. Safe-commit analysis to facilitate team software development. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 2009, Vancouver, BC: IEEE, 2009. p. 507-517.

YANG, D. *Java (TM) Persistence with JPA*. Parker, CO: Outskirts Press, 2010.

APÊNDICE A – PROJETOS AVALIADOS

Para a execução da avaliação descrita no Capítulo 5, foi necessário que os projetos e os ramos desses projetos atendessem a alguns critérios. Desta forma, diversos projetos foram analisados com o intuito de selecionar os adequados à avaliação. Quatro critérios de exclusão foram considerados:

- **CE1** – Projeto não possui ramos (não foi identificado seu uso).
- **CE2** – Projeto não possui uma estrutura de diretório organizada segundo as boas práticas do Subversion (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2008).
- **CE3** – Projeto não utiliza o Maven.
- **CE4** – Projeto não possui algum ramo que atenda às seguintes condições: já ter sofrido junção; utilização do Maven no momento da junção; ser possível identificar a versão base de criação do ramo; e, ter um número de *check-ins* significativos.

Na Tabela 11 são listados os projetos analisados, o repositório onde estão armazenados e o critério de exclusão (CE) que fez com que não fossem adotados.

Tabela 11: Projetos analisados mas excluídos da avaliação.

Projeto / Módulo	Repositório	Critério de Exclusão
Ant		
Principal	http://svn.apache.org/repos/asf/ant/core/	CE3
Ivy (principal)	http://svn.apache.org/repos/asf/ant/ivy/core/	CE3
Ivy (plugin Eclipse)	http://svn.apache.org/repos/asf/ant/ivy/ivyde/	CE3
AntUnit	http://svn.apache.org/repos/asf/ant/antlibs/antunit/	CE1
Dotnet Antlib	http://svn.apache.org/repos/asf/ant/antlibs/dotnet/	CE3
SVN Antilib	http://svn.apache.org/repos/asf/ant/antlibs/svn/	CE1
Bean Scripting Framework (BSF)		
Principal	http://svn.apache.org/repos/asf/commons/proper/bsf/	CE4
Checkstyle		
Principal	http://checkstyle.svn.sourceforge.net/svnroot/checkstyle/	CE4
Commons BCEL		
Principal	http://svn.apache.org/repos/asf/commons/proper/bcel/	CE4
DB		
DdlUtils	http://svn.apache.org/repos/asf/db/ddlutils/	CE1
Derby (principal)	http://svn.apache.org/repos/asf/db/derby/code/	CE3
Derby (site)	http://svn.apache.org/repos/asf/db/derby/site/	CE1
Jdo	http://svn.apache.org/repos/asf/db/jdo/	CE4
Ojb	http://svn.apache.org/repos/asf/db/ojb/	CE3

Projeto / Módulo	Repositório	Critério de Exclusão
Torque	http://svn.apache.org/repos/asf/db/torque/	CE2
Forrest		
Principal	http://svn.apache.org/repos/asf/forrest/	CE3
Geronimo		
Bundles	https://svn.apache.org/repos/asf/geronimo/bundles/	CE1
Components	https://svn.apache.org/repos/asf/geronimo/components/	CE1
Daytrader	https://svn.apache.org/repos/asf/geronimo/daytrader/	CE4
External	https://svn.apache.org/repos/asf/geronimo/external/	CE2
Genesis	https://svn.apache.org/repos/asf/geronimo/genesis/	CE4
Gshell	https://svn.apache.org/repos/asf/geronimo/gshell/	CE4
Javamail	https://svn.apache.org/repos/asf/geronimo/javamail/	CE1
Server (principal)	https://svn.apache.org/repos/asf/geronimo/server/	CE4
Specs	https://svn.apache.org/repos/asf/geronimo/specs/	CE2
Yoko	https://svn.apache.org/repos/asf/geronimo/yoko/	CE1
Gump		
Principal	http://svn.apache.org/repos/asf/gump/	CE3
HTTP Server		
Principal	http://svn.apache.org/repos/asf/httpd/httpd/	CE3
Java Caching System (JCS)		
Principal	http://svn.apache.org/viewvc/commons/proper/jcs/	CE4
JMeter		
Principal	https://svn.apache.org/repos/asf/jakarta/jmeter/	CE3
Logging Services		
Chainsaw	http://svn.apache.org/viewvc/logging/chainsaw/	CE4
Log4cxx	http://svn.apache.org/viewvc/logging/log4cxx/	CE4
Log4j	http://svn.apache.org/viewvc/logging/log4j/	CE4
Log4net	http://svn.apache.org/viewvc/logging/log4net/	CE1
Log4php	http://svn.apache.org/viewvc/logging/log4php/	CE4
Site	http://svn.apache.org/viewvc/logging/site/	CE3
Lucene		
Principal	http://svn.apache.org/repos/asf/lucene/dev/	CE3
Maven		
Principal (versão 1)	http://svn.apache.org/repos/asf/maven/maven-1/	CE2
Principal (versão 2)	http://svn.apache.org/repos/asf/maven/maven-2/	CE2
Principal (versão 3)	http://svn.apache.org/repos/asf/maven/maven-3/	CE4
Mojo		
Principal	http://svn.codehaus.org/mojo/	CE2
POI		

Projeto / Módulo	Repositório	Critério de Exclusão
Principal	http://svn.apache.org/repos/asf/poi/	CE3
Struts		
Principal (versão 1)	http://svn.apache.org/viewvc/struts/struts1/	CE4
Principal (versão 2)	http://svn.apache.org/viewvc/struts/struts2/	CE4
Subversion		
Principal	http://svn.apache.org/repos/asf/subversion/	CE3
Tapestry		
Principal (versão 5)	http://svn.apache.org/repos/asf/tapestry/tapestry5/	CE3
Tomcat		
Principal (desenvolvimento inicial)	http://svn.apache.org/repos/asf/tomcat/	CE2
JK (conectores Tomcat)	http://svn.apache.org/repos/asf/tomcat/jk/	CE3
Plugin Maven	http://svn.apache.org/repos/asf/tomcat/maven-plugin/	CE1
Native (conector Native/APR)	http://svn.apache.org/repos/asf/tomcat/native/	CE3
Release 5.5.x	http://svn.apache.org/repos/asf/tomcat/tc5.5.x/	CE1
Release 6.0.x	http://svn.apache.org/repos/asf/tomcat/tc6.0.x/	CE1
Release 7.0.x	http://svn.apache.org/repos/asf/tomcat/tc7.0.x/	CE1
Toolchains		
Principal	http://elftoolchain.svn.sourceforge.net/	CE3
Velocity		
Anakia	http://svn.apache.org/repos/asf/velocity/anakia/	CE4
Dvsl	http://svn.apache.org/repos/asf/velocity/dvsl/	CE2
Engine	http://svn.apache.org/repos/asf/velocity/engine/	CE2
Site	http://svn.apache.org/repos/asf/velocity/site/	CE2
Site-tools	http://svn.apache.org/repos/asf/velocity/site-tools/	CE1
Texen	http://svn.apache.org/repos/asf/velocity/texen/	CE4
Tools	http://svn.apache.org/repos/asf/velocity/tools/	CE4
XMLBeans		
Principal	http://svn.apache.org/repos/asf/xmlbeans/	CE3