COPPE/UFRJ

ORION: UMA ABORDAGEM PARA SELEÇÃO DE POLÍTICAS DE CONTROLE DE CONCORRÊNCIA

João Gustavo Gomes Prudêncio

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Cláudia Maria Lima Werner
Leonardo Gresta Paulino Murta

Rio de Janeiro
Dezembro de 2008

ORION: UMA ABORDAGEM PARA
SELEÇÃO DE POLÍTICAS DE CONTROLE DE CONCORRÊNCIA

João Gustavo Gomes Prudêncio

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

_____

Prof.ª Cláudia Maria Lima Werner, D.Sc.

_____

Prof. Leonardo Gresta Paulino Murta, D.Sc.

_____

Prof.ª Marta Lima de Queirós Mattoso, D.Sc.

_____

Prof.ª Renata Pontin de Mattos Fortes, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2008

Prudêncio, João Gustavo Gomes

Orion: Uma Abordagem para Seleção de Políticas de Controle de Concorrência / João Gustavo Gomes Prudêncio. – Rio de Janeiro: UFRJ/COPPE, 2008.

XIV, 122 p.: il.; 29,7 cm.

Orientadores: Cláudia Maria Lima Werner

Leonardo Gresta Paulino Murta

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2008.

Referencias Bibliográficas: p. 99-104.

1. Controle de Concorrência no Desenvolvimento de Software. 2. Gerência de Configuração de Software. I. Werner, Cláudia Maria Lima *et al*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Titulo.

À minha mãe Vânia e
ao meu pai Prudêncio (*in memorian*).

Agradecimentos

Aos meus pais, pelo investimento em minha educação. Agradeço, em especial, ao meu pai, por ter despertado em mim a curiosidade pela computação e à minha mãe, por todo apoio e carinho. Ao meu irmão, Pedro Henrique, por sempre ter sido um grande companheiro.

À Paula Cibele, por, desde a graduação, estar ao meu lado, me apoiando nos momentos bons e nos momentos difíceis.

Aos meus orientadores, professora Cláudia Werner e professor Leonardo Murta, pela confiança e por terem participado de forma efetiva na minha formação acadêmica e profissional.

Ao professor Márcio Barros, por ter participado com contribuições na proposta desta dissertação.

Às professoras Marta Mattoso e Renata Pontin, por terem aceitado participar desta banca.

Aos integrantes do grupo de Reutilização de Software da COPPE/UFRJ, por terem acompanhado e contribuído para o desenvolvimento deste trabalho.

Aos meus amigos de Fortaleza, que me apoiaram na decisão de estudar no Rio de Janeiro e, apesar da distância, sempre estiveram presentes.

À CAPES, pelo apoio financeiro.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ORION: UMA ABORDAGEM PARA SELEÇÃO DE POLÍTICAS DE CONTROLE DE CONCORRÊNCIA

João Gustavo Gomes Prudêncio

Dezembro/2008

Orientadores: Cláudia Maria Lima Werner
Leonardo Gresta Paulino Murta

Programa: Engenharia de Sistemas e Computação

Atualmente, o número de desenvolvedores envolvidos em um projeto de desenvolvimento de software vem aumentando cada vez mais, devido à necessidade de entregar aos clientes sistemas mais complexos, com maior qualidade e em um tempo mais curto. Dessa forma, para que o processo de desenvolvimento de software ocorra de forma organizada, é preciso prover mecanismos de controle de concorrência sobre os artefatos do projeto. Esses mecanismos são implementados por meio de políticas de controle de concorrência adotadas nos sistemas de controle de versão, que podem permitir (política otimista) ou inibir (política pessimista) o paralelismo no desenvolvimento.

Este trabalho de pesquisa apresenta a abordagem Orion, que faz uma análise do histórico de modificações do projeto e seleciona a política de controle de concorrência mais indicada para cada elemento, além de identificar elementos críticos que devam sofrer algum tipo de reestruturação. Essa seleção tem o intuito de minimizar as situações de conflitos, e, conseqüentemente, aumentar a produtividade da equipe de desenvolvimento. Um protótipo foi desenvolvido para viabilizar a aplicação da abordagem proposta. Além disso, foram realizados dois estudos preliminares para avaliar a abordagem Orion.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ORION: AN APPROACH FOR
CONCURRENCY CONTROL POLICIES SELECTION

João Gustavo Gomes Prudêncio

December/2008

Advisors: Cláudia Maria Lima Werner
Leonardo Gresta Paulino Murta

Department: Computer and Systems Engineering

Currently, the number of developers involved in a software development project is increasing because of the need to deliver to customers systems with higher complexity and quality and to reduce time-to-market. In order to have the software development process taking place in an organized way, we must provide mechanisms to control competition over the artifacts of the project. These mechanisms are implemented by concurrency control policies used in version control systems, which may allow (optimistic policy) or inhibit (pessimistic policy) parallel development.

This work presents the Orion approach, which analyzes the historical changes of a project and selects the most appropriate concurrency control policy for each element. In addition, it identifies critical elements that must receive some kind of refactoring. This selection aims to minimize conflict situations, and thus improve the productivity of the development team. A prototype was built to enable the application of the proposed approach and two studies were performed as a preliminary evaluation.

Índice

Capítulo 1 – Introdução 1
1.1 Preâmbulo 1
1.2 Motivação 1
1.3 Problema 2
1.4 Objetivo 3
1.5 Organização 4
Capítulo 2 – Controle de Concorrência 5
2.1 Introdução 5
2.2 Controle de concorrência em Banco de Dados 5
2.3 Controle de concorrência em Engenharia de Software 7
2.3.1 Introdução 8
2.3.2 Processos de GCS 8
2.3.3 Perspectivas da GCS 10
2.3.4 Sistemas de GCS 12
2.3.5 Conceitos de GCS 13
2.4 Políticas de controle de concorrência 17
2.4.1 Política pessimista 18
2.4.2 Política otimista 20
2.4.3 Comparação entre as políticas pessimista e otimista 21
2.4.4 Demais políticas 22
2.5 Trabalhos relacionados 23
2.5.1 Abordagens para o desenvolvimento de software 24
2.5.2 Abordagens na área de banco de dados 26
2.6 Considerações finais 26
Capítulo 3 – A Abordagem Orion 27
3.1 Introdução 27
3.2 Abordagem proposta 29
3.2.1 Informações históricas 30
3.2.2 Métricas para seleção de políticas de controle de concorrência 32
3.2.3 Elementos críticos 43
3.2.4 Seleção das políticas de controle de concorrência 43

3.2.5 Visualização 45
3.2.6 Aplicação da abordagem 46
3.3 Exemplo 47
3.4 Considerações finais 51
Capítulo 4 – Avaliação 52
4.1 Introdução 52
4.2 Objetivos 52
4.2.1 Estudo de viabilidade 52
4.2.2 Estudo de observação 53
4.3 Definição dos estudos 53
4.4 Estudo de viabilidade 56
4.4.1 Descrição 56
4.4.2 Procedimento 56
4.4.3 Resultados 58
4.4.4 Avaliação dos participantes 63
4.5 Estudo de observação 64
4.5.1 Descrição 64
4.5.2 Procedimento 65
4.5.3 Resultados 66
4.5.4 Avaliação dos participantes 71
4.6 Validade 72
4.7 Considerações finais 73
Capítulo 5 – O Protótipo Orion 75
5.1 Introdução 75
5.2 Odyssey-VCS 76
5.2.1 Versão 1 76
5.2.2 Versão 2 79
5.3 Protótipo Orion 84
5.3.1 Métrica de concorrência 85
5.3.2 Métrica de dificuldade de junção 86
5.3.3 Resultados 88
5.3.4 Visualização dos resultados 89
5.4 Implementação da abordagem Orion utilizando outros sistemas de controle de versão 91

5.5 Considerações finais 93
Capítulo 6 – Conclusão 75
6.1 Epílogo 94
6.2 Contribuições 95
6.3 Limitações 96
6.4 Trabalhos Futuros 97
Referências Bibliográficas 99
Apêndice I 105
Apêndice II 106
Apêndice III 107
Apêndice IV 109
Apêndice V 111
Apêndice VI 113
Apêndice VII 115
Apêndice VIII 116
Apêndice IX 117
Apêndice X 119
Apêndice XI 120
Apêndice XII 121

Índice de Figuras

Figura 2.1. Categorias de funcionalidades dos sistemas de GCS (DART, 1991). 11
Figura 2.2. *Check-out* e *check-in*. 15
Figura 2.3. Ramos de desenvolvimento. 16
Figura 2.4. Política pessimista [Adaptado de COLLINS-SUSSMAN *et al.*, (2004)]. ... 18
Figura 2.5. Política otimista [Adaptado de COLLINS-SUSSMAN *et al.*, (2004)]. 20
Figura 3.1. Visão geral da abordagem proposta. 28
Figura 3.2. Visão do uso da abordagem Orion. 29
Figura 3.3. Visualização do histórico de um repositório Subversion
pela ferramenta TortoiseSVN. 31
Figura 3.4. Situação para análise do nível de concorrência. 35
Figura 3.5. Unidades de versionamento compostas por unidade de comparação. 37
Figura 3.6. Versões de um IC no momento posterior do *check-in*. 39
Figura 3.7. Representação da intersecção dos conjuntos ações aplicadas
e ações efetivadas. 40
Figura 3.8. Dificuldade de junção igual a 0. 41
Figura 3.9. Dificuldade de junção igual a 1. 41
Figura 3.10. Exemplo do cálculo da métrica de concorrência. 47
Figura 3.11. Versões necessárias para o cálculo da métrica de dificuldade de junção. . 48
Figura 3.12. Conjuntos *ações aplicadas* e *ações efetivadas*. 49
Figura 3.13. Intersecção dos conjuntos ações aplicadas e ações efetivadas. 49
Figura 3.14. Determinação da política selecionada. 50
Figura 4.1. Histórico de um IC provido pelo sistema de controle de versão. 54
Figura 4.2. Gráfico representando informações históricas adicionais
de um determinado IC. 55
Figura 5.1. Versão 1 do Odyssey-VCS. 78
Figura 5.2. Criação implícita de ramos. 81
Figura 5.3. Ramos e ponteiros na versão 2 do Odyssey-VCS. 82
Figura 5.4. Política pessimista no Odyssey-VCS. 83
Figura 5.5. Protótipo Orion. 84
Figura 5.6. Algoritmo simplificado do cálculo da métrica de concorrência. 86

Figura 5.7. Algoritmo simplificado do cálculo da métrica de dificuldade de junção..... 87
Figura 5.8. Apresentação dos resultados no modo individual. 88
Figura 5.9. Resultados no modo completo. 89
Figura 5.10. Integração EvolTrack e protótipo Orion
[Adaptado de CEPEDA *et al.*, (2008)]. 90
Figura 5.11. Resultados da abordagem Orion visualizados no EvolTrack. 91

Índice de Tabelas

Tabela 3.1. Relação entre as métricas definidas e a gerência de riscos. 33
Tabela 3.2. Indicação das políticas de concorrência. 44
Tabela 3.3. Resultados da métrica de dificuldade de junção do exemplo. 50
Tabela 4.1. Definição do objetivo do estudo de viabilidade. 53
Tabela 4.2. Definição do objetivo do estudo de observação. 53
Tabela 4.3. Resumo do Questionário de Caracterização dos participantes do estudo de viabilidade. 59
Tabela 4.4. Resumo dos tempos da tarefa dos participantes. 59
Tabela 4.5. Resumo das tarefas executadas pelo participante P1. 60
Tabela 4.6. Resumo das tarefas executadas pelo participante P2. 60
Tabela 4.7. Resumo das tarefas executadas pelo participante P3. 61
Tabela 4.8. Resumo do Questionário de Caracterização dos participantes do estudo de observação. 67
Tabela 4.9. Resumo dos tempos da tarefa dos participantes. 67
Tabela 4.10. Resumo das tarefas executadas pelo participante P4. 68
Tabela 4.11. Resumo das tarefas executadas pelo participante P5. 68
Tabela 4.12. Resumo das tarefas executadas pelo participante P6. 69
Tabela 4.13. Estimativa de tempo para seleção de políticas para o projeto Subversion. 74
Tabela 5.1. Comparação entre sistemas de controle de versão. 92

Índice de Abreviaturas

ANSI	*American National Standards Institute*
API	*Application Programming Interface*
CASE	*Computer-Aided Software Engineering*
COPPE	Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia
CMM	*Capability Maturity Model*
CMMI	*Capability Maturity Model Integration*
CVS	*Concurrent Version System*
EMF	*Eclipse Modeling Framework*
GCS	Gerência de Configuração de Software
IC	Item de Configuração
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*International Organization for Standardization*
JMI	*Java Metadata Inferface*
MDR	*Metadata Repository*
MOF	*Meta Object Facility*
MPS.BR	Melhoria de Processos do Software Brasileiro
OMG	*Open Management Group*
RCS	*Revision Control System*
SGBD	Sistema de Gerenciamento de Banco de Dados
UFRJ	Universidade Federal do Rio de Janeiro
UML	*Unified Modeling Language*
VCS	*Version Control System*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*

Capítulo 1 – Introdução

1.1 Preâmbulo

O desenvolvimento de sistemas de software é uma atividade que, na grande maioria das vezes, envolve mais de uma pessoa. Pessoas estas que formam uma equipe e que colaboram para juntas atingirem o mesmo objetivo: construir software. A atividade de desenvolvimento de software pode ainda envolver várias equipes ao mesmo tempo, que podem estar distribuídas geograficamente (ESTUBLIER, 2000).

Além disso, cada vez mais a quantidade de pessoas que compõem essas equipes de desenvolvimento de software vem aumentando. Alguns fatores têm contribuído para este crescimento, entre eles: a competição entre as empresas que constroem sistemas de software e a necessidade de oferecer aos clientes soluções mais complexas, com mais qualidade e em menor tempo (GRINTER, 1995). Razões como essas, além do aumento da demanda por inovação, contribuem também para o crescimento da indústria de desenvolvimento de sistemas de software como um todo.

Diante desses desafios, as empresas empregam mais pessoas com o objetivo de construir sistemas de forma mais ágil e com maior qualidade (GRINTER, 1995; ESTUBLIER e GARCIA, 2006), o que resulta em uma maior pressão por trabalho concorrente (ESTUBLIER, 2000).

1.2 Motivação

Os sistemas de Gerência de Configuração de Software (DART, 1991; ESTUBLIER, 2000; CONRADI e WESTFECHTEL, 1998), em especial os sistemas de controle de versão, oferecem infra-estruturas que auxiliam na coordenação dos esforços dos membros da equipe de desenvolvimento (GRINTER, 1995). Esses sistemas oferecem, entre outras funcionalidades: um repositório central para o armazenamento dos artefatos do projeto de desenvolvimento; o controle de acesso e de modificação desses artefatos pelos desenvolvedores; e a comparação e combinação de diferentes versões de um mesmo artefato. Dessa forma, utilizando alguma ferramenta de controle de versão, é possível que os desenvolvedores trabalhem em paralelo, usando cópias locais dos artefatos.

Os mecanismos de controle dos artefatos presentes no repositório oferecidos pelos sistemas de controle de versão são implementados por meio das políticas de controle de concorrência. Existem basicamente duas categorias de políticas: pessimista e otimista (SARMA *et al.*, 2003). A política pessimista requer que o desenvolvedor faça um bloqueio no artefato em que pretende trabalhar, evitando assim que outros desenvolvedores modifiquem esse mesmo artefato concomitantemente. Por outro lado, a política otimista permite o paralelismo entre os desenvolvedores, no entanto, modificações sobrepostas podem resultar em situações de conflitos. Adotando a política otimista, assume-se que a quantidade de conflitos é naturalmente baixa e que é mais produtivo tratar cada conflito individualmente, caso eles venham a ocorrer, por meio da combinação dos esforços dos desenvolvedores.

Em geral, esses conflitos são resolvidos pelos desenvolvedores envolvidos, comparando as versões de cada um e adicionando ou removendo partes do artefato a fim de "montar" a versão final do elemento em questão. Alguns desses conflitos são fáceis e rápidos de serem resolvidos. No entanto, dependendo do tamanho das modificações efetuadas, da complexidade do elemento modificado e do conhecimento do desenvolvedor que faz a combinação, esses conflitos podem exigir muito tempo e um grande esforço para serem resolvidos, geralmente gastos em algum tipo de trabalho adicional ou retrabalho, que não seriam necessários caso não tivesse ocorrido o conflito (SOUZA *et al.*, 2003; SARMA e VAN DER HOEK, 2004). Dessa forma, os conflitos podem impactar negativamente a produtividade da equipe de desenvolvimento.

1.3 Problema

Alguns sistemas de controle de versão, como, por exemplo, o Visual SourceSafe (ROCHE e WHIPPLE, 2001), adotam preferencialmente a estratégia pessimista, por outro lado, soluções como o CVS (FOGEL e BAR, 2001), por exemplo, adotam basicamente a política otimista. No entanto, sistemas de controle de versão mais atuais, como o Subversion (COLLINS-SUSSMAN *et al.*, 2004) e o IBM Rational ClearCase (WHITE, 2000), oferecem suporte às duas categorias de políticas e ainda possibilitam que sejam escolhidas diferentes políticas para diferentes artefatos do projeto.

Contudo, apesar de os sistemas de controle de versão mais atuais oferecerem uma maior liberdade na adoção de políticas de controle de concorrência, eles não apóiam os usuários na escolha de qual política é mais adequada para cada elemento do projeto e não fornecem informações suficientes para que os usuários tomem esse tipo de

decisão. Além disso, não foi encontrada na literatura uma abordagem ou ferramenta que apóie o desenvolvedor ou gerente de configuração de software neste tipo de atividade.

Uma escolha adequada da política de controle de concorrência pode diminuir a quantidade de conflitos que ocorrem no processo de desenvolvimento. Essa diminuição implica em menos esforço e tempo dos desenvolvedores desperdiçados na resolução de conflitos, o que pode resultar em um ganho de produtividade para a equipe de desenvolvimento (SARMA e VAN DER HOEK, 2004).

1.4 Objetivo

Diante do problema exposto na Seção 1.3, o objetivo deste trabalho consiste em prover uma abordagem que auxilie os engenheiros de software a analisar as informações históricas de um projeto de desenvolvimento de software, armazenadas nos sistemas de controle de versão, e selecionar as políticas de controle de concorrência para cada um dos elementos do projeto, visando uma maior produtividade no processo de desenvolvimento. Além disso, a abordagem deve identificar os elementos que merecem uma maior atenção em relação ao controle de concorrência.

Para atingir esses objetivos, foram detectadas algumas metas que devem ser alcançadas: (1) identificar quais informações são disponibilizadas pelos sistemas de controle de versão e que podem ser utilizadas na seleção das políticas de controle de concorrência; (2) identificar informações históricas adicionais que podem ser importantes na tarefa de seleção de políticas; (3) analisar como as informações históricas identificadas como relevantes contribuem para seleção de políticas; e (4) representar os resultados da abordagem de forma clara para os envolvidos no projeto, apontando a política de controle de concorrência mais indicada para cada elemento e identificando os elementos que merecem uma maior atenção em relação ao controle de concorrência, os chamados elementos críticos.

Para atingir os objetivos listados acima, foram definidas duas métricas a respeito de controle de concorrência, com base nas informações históricas identificadas. Essas métricas são utilizadas na escolha da política de controle de concorrência para cada elemento do projeto de desenvolvimento e na identificação dos elementos críticos.

Além disso, foram realizados dois estudos preliminares para avaliar a abordagem proposta neste trabalho de pesquisa e foi desenvolvido um protótipo para viabilizar a aplicação da abordagem.

1.5 Organização

O restante desta dissertação está organizado em outros cinco capítulos, além deste capítulo de introdução.

O Capítulo 2 apresenta uma introdução e uma revisão da literatura sobre o tema controle de concorrência, principalmente relacionado ao processo de desenvolvimento de software. Além disso, são discutidos alguns conceitos que são amplamente utilizados nesta dissertação, como, por exemplo, conceitos de Gerência de Configuração de Software. São apresentadas ainda algumas abordagens existentes que oferecem apoio ao controle de concorrência.

O Capítulo 3 apresenta a abordagem proposta por este trabalho de pesquisa, com base nos problemas identificados na literatura e nas abordagens existentes atualmente. Além disso, são discutidas as soluções escolhidas para atingir os objetivos descritos na Seção 1.4. Esse capítulo apresenta, também, uma visão geral da abordagem e suas principais características e funcionalidades. Para facilitar a compreensão, ao fim do capítulo, é apresentado um exemplo de aplicação da abordagem proposta.

O Capítulo 4 apresenta dois estudos efetuados para avaliar a abordagem proposta. Esses estudos de avaliação permitem identificar limitações e possibilidades de melhoria da abordagem proposta.

O Capítulo 5 discute os detalhes de implementação da abordagem proposta por esse trabalho de pesquisa por meio de um protótipo para seleção de políticas de controle de concorrência.

O Capítulo 6 apresenta algumas conclusões desta dissertação, destacando as suas principais contribuições, relatando as limitações detectadas e listando possíveis trabalhos futuros.

Finalmente, os Apêndices apresentam os documentos utilizados para a aplicação dos estudos descritos no Capítulo 4.

Capítulo 2 – Controle de Concorrência

2.1 Introdução

O controle de concorrência é a atividade de coordenar as ações de processos que operam em paralelo, acessam dados compartilhados, e, portanto, ações que potencialmente interferem umas nas outras (BERNSTEIN *et al.*, 1987). No contexto de processo de desenvolvimento de sistemas de software, o controle de concorrência representa a atividade de coordenar os esforços dos desenvolvedores na construção de um software.

Como foi discutido no Capítulo 1, cada vez mais as empresas empregam mais pessoas com o objetivo de construir sistemas de forma mais ágil e com maior qualidade (GRINTER, 1995; ESTUBLIER e GARCIA, 2006), o que resulta em uma maior pressão por trabalho concorrente (ESTUBLIER, 2000).

Entretanto, para que a inclusão de mais pessoas no processo de desenvolvimento não tenha um efeito negativo, acabando por trazer problemas para a coordenação dos esforços de desenvolvimento, há a necessidade de definir estratégias de cooperação, possibilitando que vários engenheiros de software modifiquem e gerenciem o mesmo conjunto de artefatos[1] de forma eficiente.

Na Seção 2.2, é apresentado como o controle de concorrência é tratada na área de Banco de Dados. Na Seção2.3, é discutido como o controle de concorrência é tratado na área de Engenharia de Software e são apresentados conceitos importantes de Gerência de Configuração de Software. A Seção 2.4 discute sobre políticas de controle de concorrência. Na Seção 2.5, são apresentados alguns trabalhos relacionados. Finalmente, a Seção 2.6 conclui o capítulo com algumas considerações finais.

2.2 Controle de concorrência em Banco de Dados

Em Computação, diversas áreas já se depararam com a questão de controle de concorrência. A primeira delas foi a área de sistemas operacionais. Em seguidas, outras

[1] O termo artefato refere-se a artefato de software, que representa qualquer informação gerada no processo de desenvolvimento, incluindo especificações do sistema, código-fonte, planos de teste, projeto, entre outros (CONRADI e WESTFECHTEL, 1998).

áreas como computação paralela, banco de dados (CONRADI e WESTFECHTEL, 1998) adaptaram a estratégia utilizada em sistemas operacionais. A seguir, é apresentado como esse tópico é abordado na área de banco de dados, onde esse tema já foi amplamente discutido.

A consistência de um banco de dados é mantida se todo item de dado no banco satisfaz as restrições de consistência específicas da aplicação. Por exemplo, em um sistema de venda de passagens de uma empresa de aviação, uma restrição de consistência pode ser a exclusividade de cada assento da aeronave para apenas um passageiro. Para efetuar ações como essas, os Sistemas de Gerenciamento de Banco de Dados (SGBDs) transformam cada uma delas em operações de leitura e escrita, com o objetivo de garantir que o banco de dados esteja sempre em um estado consistente.

Para resolver esse problema, as operações de leitura e escrita executadas por um programa que acessa um banco de dados são agrupadas em seqüências chamadas de transações. Os usuários interagem com os SGBDs por meio da execução de transações (BARGHOUTI e KAISER, 1991).

Para garantir que as transações sejam processadas de forma confiável, os SGBDs utilizam um conjunto de quatro propriedades: atomicidade, consistência, isolamento e durabilidade (ACID). A primeira delas considera uma transação como unidade atômica de processamento que deve ser totalmente executada, ou totalmente desfeita. A propriedade de consistência exige que uma transação sempre leve o banco de dados de um estado consistente a outro estado consistente ao fim de sua execução. A terceira propriedade, isolamento, garante que a execução de uma transação não seja afetada por outra transação que esteja sendo executada concomitantemente. Por último, a propriedade de durabilidade refere-se à garantia de que, uma vez que a transação seja executada, esta irá persistir e não será desfeita, mesmo em casos de uma eventual falha do sistema (GRAY, 1981; KUNG e ROBINSON, 1981).

Em um sistema multiusuário, transações são executadas concorrentemente por diferentes usuários e, devido a isso, a integridade e consistência de um banco de dados pode ser violada. O SGBD resolve esse problema por meio de mecanismos de controle de concorrência, que permitem que sejam executadas somente transações concorrentes que preservem a integridade do SGBDs (BARGHOUTI e KAISER, 1991). Esses mecanismos são divididos basicamente em duas categorias. A primeira delas, categoria pessimista, utiliza bloqueios sobre as transações para evitar conflitos, como, por

exemplo, o mecanismo de bloqueio em duas fases[2]. Por outro lado, a categoria otimista, que não faz uso de bloqueios, inclui, por exemplo, o esquema de ordenação dos *timestamps*, que registra a ordem de execução das transações antes que elas comecem a ser executadas (BERNSTEIN *et al.*, 1987).

No contexto de desenvolvimento de software, foco deste trabalho, o controle de concorrência é utilizado para gerenciar os engenheiros de software que trabalham ao mesmo tempo sobre os mesmos artefatos. No cenário de SGBDs, a questão de controle de concorrência foi tratada e resolvida por meio das transações. No entanto, em bancos de dados, uma típica transação dura uma fração de segundo (ESTUBLIER, 2001) e, normalmente, envolve poucos elementos, enquanto que, no contexto de desenvolvimento de software, a duração de um conjunto de modificações pode ser na ordem de horas, dias ou mais, dependendo, entre outros fatores, da complexidade do projeto e do estilo de trabalho da equipe de desenvolvimento. Além disso, o escopo da modificação pode ser extenso, incluindo vários elementos. Como conseqüência, a abordagem de serialização utilizada em banco de dados, onde as modificações são executadas em seqüência, nem sempre pode ser utilizada no âmbito do desenvolvimento de software (ESTUBLIER, 2001). Em razão dessas particularidades, é inevitável que o controle de concorrência seja tratado na área de desenvolvimento de software de forma diversa ao tratamento dado em outras áreas da computação, como a de banco de dados, discutida nesta seção.

2.3 Controle de concorrência em Engenharia de Software

O controle de concorrência no processo de desenvolvimento de software representa a atividade de coordenar os esforços dos desenvolvedores na construção de um software. Essa coordenação é feita por meio da adoção de sistemas de Gerência de Configuração de Software (GCS).

Esta seção está organizada da seguinte forma: na seção 2.3.1, é introduzido o tópico de GCS; na seção 2.3.2, são apresentadas algumas normas que abordam GCS; na seção 2.3.3, são discutidas as perspectivas atendidas pela GCS; na seção 2.3.4, são apresentados os sistemas que compõem a GCS; e, finalmente, a seção 2.3.5 discute conceitos importantes da GCS.

[2] Do termo inglês: *two phase locking*.

2.3.1 Introdução

A GCS é uma disciplina que visa identificar, controlar, acompanhar e verificar os artefatos produzidos e modificados durante o ciclo de vida do software (MURTA, 2006). Dessa forma, a GCS possibilita que os sistemas de software evoluam de forma controlada (DART, 1991) e, portanto, contribuam para o atendimento às restrições de qualidade e de tempo (ESTUBLIER, 2000). Além de assegurar um processo de desenvolvimento de software sistemático e rastreável, no qual todas as modificações são precisamente gerenciadas, a GCS busca fornecer suporte ao trabalho concorrente no processo de desenvolvimento de software (ESTUBLIER *et al.*, 2005).

A disciplina de Gerência de Configuração surgiu inicialmente na indústria aeroespacial nos anos 50, quando a produção de aeronaves experimentou dificuldades causadas pela falta de documentação eficiente das mudanças no processo de construção (LEON, 2000). Anos mais tarde, os sistemas de software começaram a apresentar alguns dos mesmos desafios em termos de gerenciamento de modificações. Logo, ficou claro que eram necessários métodos similares aos existentes na Gerência de Configuração a serem aplicados sobre os artefatos de software. Com isso, ao final dos anos 70, a GCS surgiu como uma disciplina própria na área de Engenharia de Software (ESTUBLIER *et al.*, 2005).

De acordo com LEON (2000), no cenário de desenvolvimento de software, é possível detectar aumentos substanciais de qualidade e produtividade devido à adoção de GCS. Além disso, o autor apresenta outros benefícios: menores custos da manutenção do software; redução dos defeitos; rápida identificação de problemas e conserto de defeitos; e desenvolvimento dependente de processo, ao invés de desenvolvimento dependente das pessoas.

Os sistemas de GCS foram projetados para dar suporte ao desenvolvimento de software concorrente realizado por múltiplos engenheiros de software e prover mecanismos para desfazer alterações indesejáveis (STOREY *et al.*, 2005). Além disso, esses sistemas possibilitam o gerenciamento detalhado das modificações, extensões e adaptações que foram aplicadas ao software durante todo o seu ciclo de vida (ESTUBLIER *et al.*, 2005).

2.3.2 Modelos de GCS

Existem diversas normas produzidas por órgãos internacionais como, por exemplo, IEEE (*Institute of Electrical and Electronics Engineers*), ANSI (*American*

National Standards Institute) e ISO (*International Organization for Standardization*), que refletem a importância da GCS (MURTA, 2006; CONRADI e WESTFECHTEL, 1998).

A norma IEEE Std 1042 (IEEE, 1987) é considerada a norma internacional mais completa sobre GCS (MURTA, 2006) e serve como um guia para a aplicação da IEEE Std 828. O objetivo principal desta norma é descrever a aplicação da disciplina de Gerência de Configuração no desenvolvimento de software. A norma ressalta a importância da GCS como processo de apoio ao processo de desenvolvimento e afirma que as atividades de GCS devem ser incorporadas ao dia-a-dia do projeto para garantir sua efetividade.

A norma IEEE Std 828 (IEEE, 2005) aborda a atividade de planejamento da GCS. Além disso, afirma que a GCS, de alguma forma, sempre é utilizada nos projetos de software, seja planejada ou de forma *ad-hoc*. No entanto, quando há um planejamento das atividades de GCS, ocorre um aumento na sua eficiência. Esta norma discute as informações importantes e a estrutura que os planos de GCS devem apresentar. Além disso, trata também da evolução desses documentos durante o ciclo de vida do software.

A norma ISO 10007 (ISO, 1995) é bastante abrangente e discute a aplicação de Gerência de Configuração em qualquer produto. Ela fornece diretrizes para a utilização de GCS na indústria e define a interface da GCS com as demais áreas de gerência. Além disso, a norma aborda a necessidade de auditoria sobre os planos e procedimentos da GCS.

Os modelos de maturidade CMM (*Capability Maturity Model*) (JALOTE, 1999) e sua evolução CMMI (*Capability Maturity Model Integration*) (CMMI Product Team, 2006) fornecem modelos multi-níveis para a classificação de maturidade de empresas de desenvolvimento de software. São definidos cinco níveis de maturidade: inicial, repetível, definido, gerenciável e otimizado. Para cada um desses níveis, são definidas áreas chave de processo que devem ser atendidas, possibilitando que esse nível seja alcançado. A GCS representa uma área chave para evoluir do nível Inicial, onde não existe um processo definido, para o Repetível, onde, além da GCS, outras áreas chave devem estar presentes como, por exemplo, gerência de projeto e garantia de qualidade. A GCS ainda contribui indiretamente para o atendimento de outras áreas chave, como, por exemplo, prevenção de defeitos do nível gerenciável e gerenciamento quantitativo do processo, presente no nível otimizado (MURTA, 2006).

O modelo CMMI surgiu em 2002 tendo como base a versão 2.0 do CMM. Assim como no CMM, o CMMI apresenta a GCS como uma área chave do grupo de processos de suporte. O CMMI é composto por seis níveis de capacidade: incompleto, executável, gerenciável, definido, quantitativamente gerenciável e otimizado. A GCS é um requisito para que qualquer área chave de processo possa atingir o nível gerenciável. Da mesma forma que no CMM, a GCS está fortemente relacionada com o atendimento a outras áreas chave, como, por exemplo, planejamento de projetos e análise de decisão e resolução (MURTA, 2006).

Por fim, o MPS.BR (Melhoria de Processos do Software Brasileiro) (SOFTEX, 2007a) consiste em um programa para melhoria de processos de software, voltado para a realidade das micro, pequenas e médias empresas de software brasileiras. O MPS.BR apresenta sete níveis de maturidade: parcialmente gerenciado (nível G), gerenciado (nível F), parcialmente definido (nível E), largamente definido (nível D), definido (nível C), gerenciado quantitativamente (nível B) e em otimização (nível A). A GCS representa um dos processos do nível F.

2.3.3 Perspectivas da GCS

De acordo com CONRADI e WESTFECHTEL (1998), a GCS atende, basicamente, a duas diferentes frentes no processo de desenvolvimento. Na primeira delas, a GCS representa uma disciplina de suporte à gerência de projeto, já que, com os sistemas de GCS, é possível, por exemplo, identificar os componentes do produto, estabelecer procedimentos e obter acompanhamentos das modificações. Na outra frente, a GCS representa uma disciplina para suporte ao desenvolvimento, uma vez que os sistemas de GCS provêem aos engenheiros de software funcionalidades que os assistem na coordenação de modificações sobre o produto, como, por exemplo, manutenção das versões dos artefatos do produto.

Para atender às duas frentes apresentadas, os sistemas modernos de GCS possuem muitas funcionalidades de alto nível. DART (1991) as classificou em oito categorias, levando em consideração os diferentes tipos de usuários dos sistemas de GCS. Cada usuário tem um papel específico e, portanto, possui diferentes requisitos para os sistemas de GCS. Na Figura 2.1, cada uma dessas categorias é representada por uma caixa. Uma caixa sobre a outra indica que a categoria de funcionalidades da caixa superior conta com características providas pela categoria representada pela caixa inferior. Cada categoria relaciona-se com diferentes necessidades dos usuários:

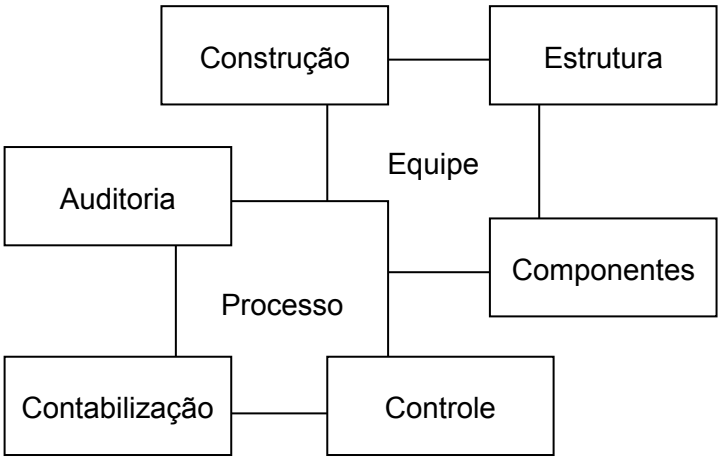


Figura 2.1. Categorias de funcionalidades dos sistemas de GCS (DART, 1991).

- *Equipe*: categoria que possibilita a colaboração dos engenheiros de software. Envolve o gerenciamento dos seus espaços de trabalho e a identificação e resolução de conflitos.
- *Componentes*: categoria responsável por identificar, classificar, armazenar e acessar os itens que compõem o produto. Envolve o gerenciamento das versões e configurações[3] do sistema.
- *Estrutura*: os sistemas de GCS devem suportar a representação da estrutura do produto, identificando as partes do sistema que se relacionam umas com as outras.
- *Construção*: relacionada com a construção do programa executável de forma eficiente, a partir dos códigos fonte.
- *Processo*: os sistemas de GCS devem suportar a seleção de tarefas a serem executadas durante todo o processo de ciclo de vida do software, permitindo controlar como o produto evolui.
- *Controle*: categoria de funcionalidades importantes no entendimento, por parte dos usuários, do impacto das modificações. Controla como e quando as modificações são feitas, permitindo que os artefatos passíveis de GCS possam evoluir de forma controlada.
- *Auditoria*: categoria de funcionalidades que mantém um rastro auditável do produto e do processo de desenvolvimento do produto. Informações sobre

[3] O termo configuração refere-se a um conjunto de artefatos que juntos constituem um sistema de software válido (ESTUBLIER, 2000).

quem, quando e por que as modificações ocorreram, por exemplo, são relevantes para essa questão.

- Contabilização: os sistemas devem dar apoio aos usuários em coletas de estatísticas sobre o sistema que está sendo construído e seu processo de desenvolvimento.

Essas funcionalidades contribuem para a evolução controlada do software e oferecem meios para dar suporte ao trabalho concorrente no processo de desenvolvimento. O controle de concorrência está fortemente relacionado com a funcionalidade Equipe, por envolver atividades de resolução de conflitos no processo de junção de esforços de desenvolvedores; e com a funcionalidade Controle, por incluir tarefas relacionadas ao controle de acesso aos elementos do sistema e requisição e propagação de modificações no processo de desenvolvimento.

2.3.4 Sistemas de GCS

Na perspectiva de desenvolvimento de software, as funcionalidades de GCS apresentadas na seção anterior são implementadas por meio de, basicamente, três classes de sistemas: sistema de controle de modificação, sistema de controle de versão e sistema de gerenciamento de construção (MURTA, 2006).

O sistema de controle de modificação tem como função fazer o controle da configuração. Ele é responsável por armazenar todas as informações geradas durante o andamento das solicitações de modificação e relatar essas informações aos participantes interessados e autorizados. O software Bugzilla (BARNSON *et al.*, 2003) é um exemplo dessa classe de sistemas.

O sistema de controle de versão permite que os itens do produto sejam identificados e que evoluam de forma distribuída e concorrente, porém disciplinada. Isso permite que diversas solicitações de modificação possam ser atendidas em paralelo pela equipe de desenvolvimento, sem desorganizar o sistema de GCS como um todo. As soluções CVS (FOGEL e BAR, 2001) e Subversion (COLLINS-SUSSMAN *et al.*, 2004) são exemplos desse tipo de sistema.

O sistema de gerenciamento de construção é responsável pela automatização do complexo processo de transformação dos diversos artefatos de software que compõem um projeto no sistema executável. Além disso, auxilia na função de liberação e entrega do produto. Como exemplo desse sistema podemos citar o Ant (HATCHER e LOUGHRAN, 2004).

Dentre os sistemas apresentados anteriormente, a categoria dos sistemas de controle de versão foi a que recebeu um maior número de contribuições, tanto do meio acadêmico quanto da indústria (MURTA, 2006). Várias soluções comerciais[4] foram desenvolvidas, entre elas, IBM Rational ClearCase (WHITE, 2000), Microsoft Visual SourceSafe (ROCHE e WHIPPLE, 2001) e BitKeeper (BITMOVER, 2008). Outras tantas soluções livres[5] estão disponíveis, como, por exemplo, RCS (TICHY, 1985), CVS e Subversion. Além disso, soluções acadêmicas também estão disponíveis na literatura, como, por exemplo, Odyssey-VCS (MURTA *et al.*, 2008) e Phoca (JUNQUEIRA, 2008).

Dentre as soluções livres, o CVS, por muito tempo, foi considerado a escolha padrão de sistemas de controle de versão, sendo amplamente utilizado até os dias atuais. No entanto, com o passar do tempo, foram identificados alguns pontos que poderiam ser melhorados no CVS, como por exemplo, versionamento de diretórios, *check-ins* atômicos, versionamento de meta-dados e manipulação mais eficiente dos dados (COLLINS-SUSSMAN *et al.*, 2004). Em conseqüência disso, surgiu o Subversion, com a finalidade de substituir o CVS. Alguns dos idealizadores do CVS participaram da criação do Subversion e desenvolveram um sistema de controle de versão semelhante ao primeiro, com o intuito de não modificar efetivamente a forma de trabalho dos usuários do CVS. Entretanto, buscou-se agregar características que permitissem uma implementação melhor, mais eficiente e que resolvessem os pontos fracos identificados (COLLINS-SUSSMAN *et al.*, 2004). Atualmente, muitas empresas estão migrando para o Subversion e este está se tornando um padrão de fato entre as soluções de sistema de controle de versão.

2.3.5 Conceitos de GCS

As soluções de sistema de controle de versão existentes provêem uma infra-estrutura para a definição de quais artefatos serão passíveis de GCS, os chamados itens de configuração (ICs). IC representa a agregação de hardware, software ou ambos, tratada pela GCS como um elemento único (IEEE, 1990). Além da identificação, as soluções de sistema de controle de versão também provêem várias operações sobre esses ICs. Uma dessas operações, chamada de *check-out*, representa o processo de

[4] Soluções proprietárias, que necessitam de licença paga para serem utilizadas.

[5] Soluções abertas (*opensources*), que podem ser usadas livremente.

solicitação, aprovação e cópia de ICs do repositório, construindo uma cópia de trabalho[6] no espaço de trabalho do engenheiro de software (LEON, 2000). O repositório representa o local de acesso controlado, provido pelos sistemas de controle de versão, onde as versões dos ICs são armazenadas. O termo versão representa o estado de um IC em um determinado momento do desenvolvimento de software (LEON, 2000). O espaço de trabalho representa o local destinado à criação, alteração e teste dos ICs na estação local de trabalho dos engenheiros de software. Cada um destes possui seu espaço de trabalho completamente isolado, o que significa que nada de novo deve entrar na área de trabalho sem a autorização explícita do desenvolvedor. O processo de atualização do espaço de trabalho permite que novidades disponíveis no repositório sejam incorporadas à versão da área de trabalho.

Depois de efetuar modificações no seu espaço de trabalho, o engenheiro de software salva suas modificações no repositório, mediante um processo denominado de *check-in*, que representa o processo de revisão, aprovação e cópia de ICs do espaço de trabalho do engenheiro de software para o repositório (LEON, 2000).

O ciclo de modificações se resume, então, em fazer *check-out* de um ou mais ICs, fazer as alterações necessárias, e fazer *check-in* da(s) nova(s) versão(ões) gerada(s) do(s) item(s). A Figura 2.2 ilustra alguns conceitos vistos até aqui. No exemplo, um desenvolvedor faz o *check-out* de um arquivo texto na versão 4, modifica este item no seu espaço de trabalho e, em seguida, salva suas alterações no repositório, por meio do *check-in*, gerando uma nova versão do IC.

A freqüência com que esse ciclo de modificações é efetuado varia de acordo com o tamanho de cada tarefa, com o processo adotado pela equipe, ou com a experiência dos desenvolvedores do projeto (SOUZA *et al.*, 2003).

Quando um engenheiro de software tenta fazer um *check-in* e o IC em questão sofreu modificações de outros engenheiros desde o seu último *check-out*, pode ocorrer uma situação de conflito. Os conflitos podem ser divididos em dois tipos: conflitos diretos e conflitos indiretos (SARMA *et al.*, 2007). Conflitos diretos são causados por modificações em paralelo no mesmo IC, por exemplo, quando dois ou mais engenheiros de software alteram um mesmo arquivo ao mesmo tempo em seus respectivos espaços de trabalho. Conflitos indiretos são causados por modificações em um IC que afeta modificações concorrentes em outro IC. Isso pode ocorrer quando um engenheiro de

[6] Do termo em inglês: *working copy*.

software, por exemplo, trabalhando em seu espaço de trabalho privado, altera uma interface Java e outro engenheiro de software está desenvolvendo uma classe que implementa essa interface em seu respectivo espaço de trabalho. Apenas os conflitos diretos são identificados pelos sistemas de controle de versão, cabendo à equipe de desenvolvimento o controle manual dos conflitos indiretos. Nesta dissertação, são tratados apenas os conflitos diretos, portanto quando conflitos forem mencionados, referem-se a este tipo.

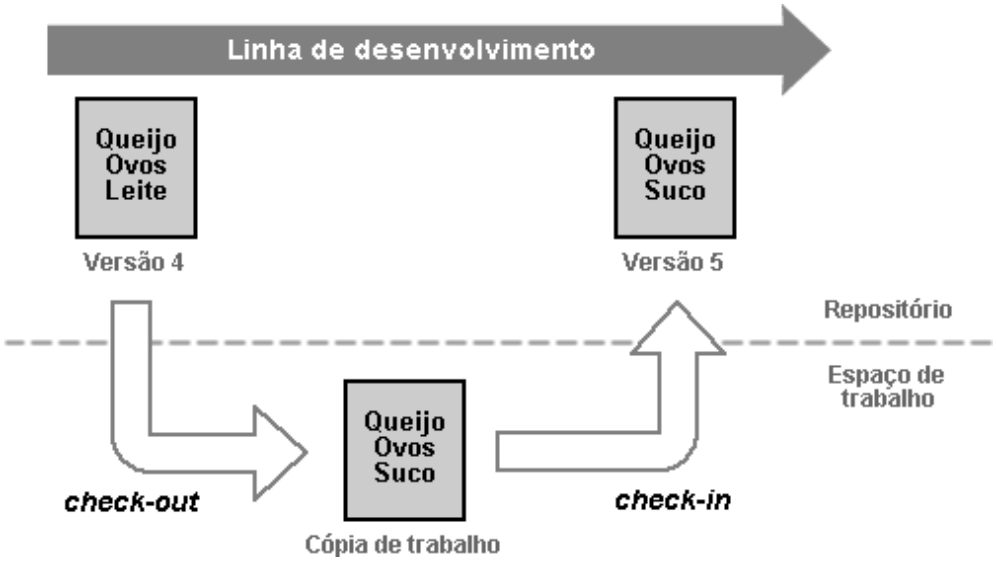


Figura 2.2. *Check-out* e *check-in*.

Dependendo do sistema de controle de versão, um conflito pode ocorrer quando há modificações no mesmo IC, ainda que em locais diferentes, como, por exemplo, alterações em métodos diferentes de uma mesma classe; ou pode ocorrer somente quando há modificações que se sobrepõem, por exemplo, alterações em uma mesma linha de um mesmo arquivo.

Outro recurso muito útil nos sistemas de controle de versão é a comparação de versões de um determinado IC. O usuário pode, por exemplo, comparar uma versão atual com uma versão do mês anterior para saber o que foi modificado nesse período de tempo. Para executar essa operação, a maioria dos sistemas utiliza o algoritmo *diff* (CONRADI e WESTFECHTEL, 1998). Muitas vezes, é possível utilizar uma terceira versão nessa comparação, uma versão anterior comum às duas versões comparadas, chamada de pivô. Isso é muito utilizado no momento do *check-in*, para verificação de conflitos. A versão que o desenvolvedor faz *check-in* e a versão atual do repositório têm uma versão em comum, que é a versão que o desenvolvedor fez *check-out*. Utilizando esta versão como pivô, a comparação fornece resultados mais ricos. É possível saber,

por exemplo, se determinada parte do IC foi removida ou adicionada, comparando com a versão original (pivô). Nestes casos, onde três versões são levadas em consideração, é utilizado o algoritmo *diff3* (CONRADI e WESTFECHTEL, 1998).

Além de permitir o controle das alterações dos arquivos e garantir que todos os desenvolvedores trabalhem com a mesma cópia atualizada, alguns sistemas de controle de versão apresentam ainda um mecanismo interessante: possibilitam a criação e o gerenciamento de várias linhas de desenvolvimento, os chamados ramos[7] (WALRAD e STROM, 2002). Este mecanismo permite um grau ainda maior de concorrência entre os desenvolvedores do projeto (Figura 2.3). Muitas vezes, é necessário criar versões diferentes de um software, mas que contenham a mesma base. Um exemplo de aplicação desta funcionalidade é a confecção de duas versões de um software para duas plataformas diferentes, como, por exemplo, Windows e Linux. Apesar das duas versões serem bem parecidas, ajustes particulares, pertinentes à plataforma alvo, precisam ser feitos. Além disso, os ramos permitem ainda que diferentes equipes de engenheiros de software trabalhem de forma paralela com objetivos diferentes. Por exemplo, uma equipe trabalhando na correção de defeitos identificados pelos clientes em uma versão liberada e outra equipe desenvolvendo paralelamente novas funcionalidades para as futuras versões do software.

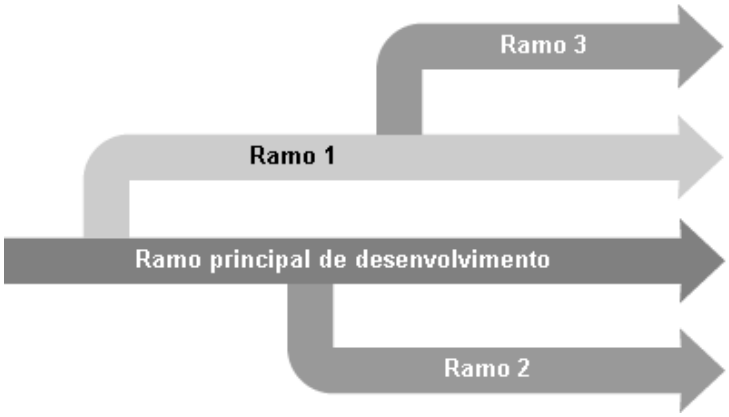


Figura 2.3. Ramos de desenvolvimento.

Além dessas operações, as soluções de sistema de controle de versão normalmente suportam a adoção de diferentes políticas de controle de concorrência que podem ser aplicadas a cada IC de um projeto, permitindo ou não o paralelismo no desenvolvimento. A descrição dessas políticas é apresentada na seção seguinte.

[7] Do termo em inglês *branches*.

2.4 Políticas de controle de concorrência

LEON (2000) apresenta uma série de problemas que têm afetado o processo de desenvolvimento de software e que impulsionaram a criação e utilização dos sistemas de GCS. Primeiramente, em projetos com apenas um desenvolvedor, não havia problemas de concorrência e de comunicação. No entanto, à medida que o número de engenheiros de software foi crescendo, esses problemas começaram a aparecer. No início, os artefatos de software eram compartilhados por diversos engenheiros, porém modificações efetuadas por um deles geravam efeitos colaterais nos artefatos dos outros. Considere, por exemplo, dois engenheiros de software, José e Maria, que compartilham um artefato. A aplicação que José estava compilando normalmente pode deixar de compilar devido a alterações feitas por Maria no artefato compartilhado. José, sem estar ciente da modificação, pode gastar horas tentando encontrar o problema.

Uma solução inicial para essa questão, chamada de compartilhamento de dados, foi a criação de espaços de trabalho separados e independentes, onde cada engenheiro de software tivesse sua própria cópia dos artefatos. Nesta solução, quando uma modificação era efetuada, os outros engenheiros não eram afetados. Por outro lado, muito espaço físico era utilizado e faltava controle sobre as cópias existentes. Além disso, era necessária a replicação das modificações nas diferentes cópias para implementar os mesmos requisitos e corrigir os mesmos defeitos. Isso ficou conhecido como o problema de manutenção múltipla (LEON, 2000).

Adotou-se, então, a utilização de repositórios centralizados que armazenavam versões comuns dos artefatos. Neste cenário, se uma modificação era efetuada, a nova versão do artefato era copiada para o repositório e disponibilizada a todos os engenheiros de software. Contudo, surgiu outra questão: não havia controle sobre as modificações feitas nos artefatos compartilhados, o que gerou o chamado problema da atualização simultânea. Para exemplificar, consideremos outra situação, onde os engenheiros de software José e Maria precisam adicionar novas funcionalidades ao mesmo artefato. Ambos fazem a cópia da versão do repositório e as modificam nos seus respectivos espaços de trabalho, gerando duas versões diferentes. Em seguida, Maria copia sua versão para o repositório. Momentos depois, João resolve fazer o mesmo e acaba sobrescrevendo a versão de Maria, desperdiçando todo o trabalho dela.

Para resolver essa questão, os sistemas de GCS impuseram uma disciplina sobre os repositórios centralizados, criando mecanismos de controle. Esses mecanismos são

implementados por meio da utilização de políticas de controle de concorrência pelos sistemas de controle de versão. Essas políticas podem ser divididas basicamente em duas classes: pessimista e otimista (SARMA *et al.*, 2003).

2.4.1 Política pessimista

A política pessimista enfatiza o uso de *check-out* reservado, efetuando um bloqueio e inibindo o paralelismo do desenvolvimento sobre um mesmo IC. Utilizando esta política, as situações de conflito são evitadas (MURTA, 2006).

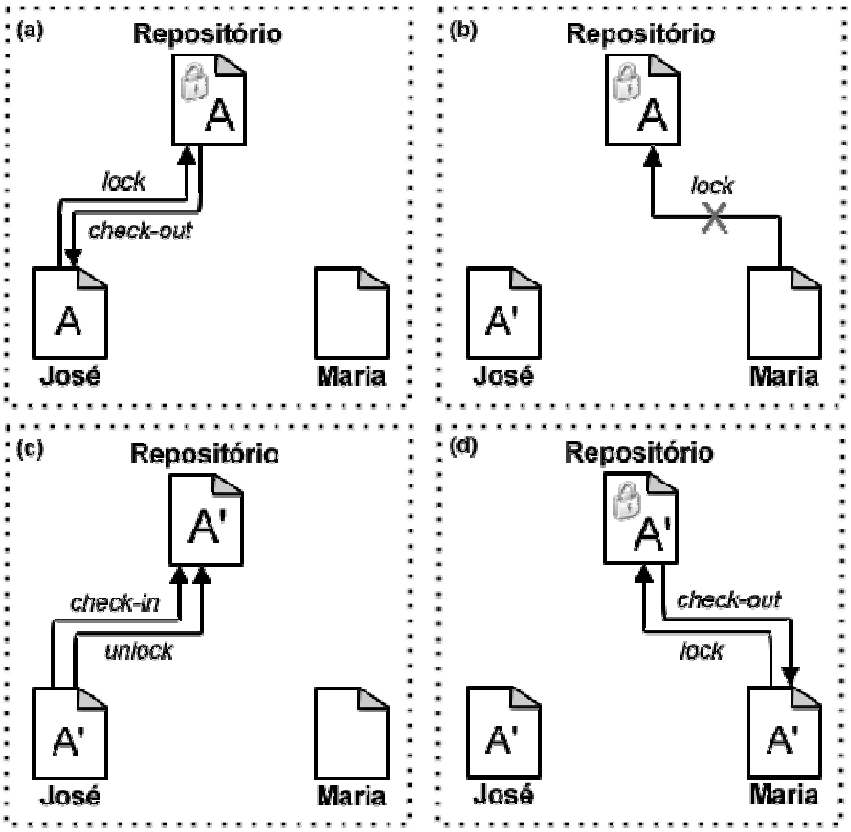


Figura 2.4. Política pessimista [Adaptado de COLLINS-SUSSMAN *et al.*, (2004)].

A Figura 2.4 ilustra a utilização da política pessimista. O engenheiro de software José precisa fazer modificações no IC A. Porém, além de fazer o *check-out*, é necessário que José faça um bloqueio (*lock*) no IC antes de modificá-lo (Figura 2.4a). Em sistemas que adotam como padrão a política pessimista, usualmente o *check-out* já engloba o bloqueio do artefato, o que é chamado de *check-out* reservado. Mais tarde, Maria também precisa alterar o mesmo IC A, no entanto, ele está bloqueado por José. Dessa forma, Maria terá que esperar pelo desbloqueio para que possa realizar suas modificações (Figura 2.4b). Depois de algum tempo, José salva suas modificações no

repositório e desbloqueia o IC (Figura 2.4c). Somente após isso, Maria pode fazer o *check-out*, em seguida ao bloqueio, e iniciar suas alterações no IC A (Figura 2.4d).

A política pessimista apresenta vantagens como: (1) não ocorrem conflitos diretos, pois não existe paralelismo, e, portanto, os engenheiros de software não vão gastar tempo e esforço na resolução de conflitos; e (2) o bloqueio pode servir como uma forma de comunicação entre os engenheiros de software, alertando-os sobre a serialização do IC e instigando-os a estabelecer uma comunicação direta entre os interessados em alterar o mesmo IC naquele instante.

Por outro lado, a política pessimista possui algumas desvantagens. Em primeiro lugar, o esquema de bloqueio pode trazer problemas administrativos. Se um engenheiro de software esquecer algum IC bloqueado, pode impedir que outros engenheiros de software modifiquem o mesmo IC, dando a falsa impressão de que o IC está sofrendo alterações. Se o engenheiro que esqueceu o IC bloqueado viajar ou tiver que se ausentar por motivos de doença, por exemplo, os outros engenheiros de software terão que entrar em contato com o administrador do sistema de controle de versão para poder liberar o IC, caso contrário, terão seus trabalhos adiados, o que pode provocar atrasos no projeto. No entanto, nos sistemas de controle de versão mais modernos, como, por exemplo, Subversion, situações de impasses (*deadlocks*) são evitadas, por meio de um mecanismo de "roubo" de bloqueios, que deve ser utilizado somente em situações onde o IC não pode ser desbloqueado normalmente por quem o bloqueou.

Em segundo lugar, a política pessimista pode causar uma serialização desnecessária. Dois engenheiros de software não terão como modificar o mesmo arquivo (e.g. classe), mesmo sendo modificações em locais distintos e independentes dentro do arquivo (e.g. métodos diferentes).

E por último, os bloqueios podem criar uma falsa sensação de segurança (COLLINS-SUSSMAN *et al.*, 2004). Um engenheiro de software, que está alterando um determinado IC, pode achar que, como está com o IC bloqueado, os outros engenheiros não terão como influenciar no seu desenvolvimento. Porém, esse IC que está sendo modificado pode depender de outro elemento (e.g. classe, método), que está em outro IC e que pode estar sendo modificado por um outro engenheiro de software. Este é um exemplo típico do já discutido conflito indireto, um problema que não é prevenido pela utilização da política pessimista.

2.4.2 Política otimista

A política otimista, já utilizada em sistemas de controle de versão baseados em arquivo, desde os anos 80 (CONRADI e WESTFECHTEL, 1998), assume que a quantidade de conflitos é naturalmente baixa e que é mais produtivo tratar cada conflito individualmente, caso eles venham a ocorrer. O mecanismo de resolução de conflitos utilizado pela política otimista é a junção, que une os trabalhos efetuados em paralelo sobre um mesmo IC e produz uma nova versão desse IC, que contém a combinação desses trabalhos (MURTA, 2006).

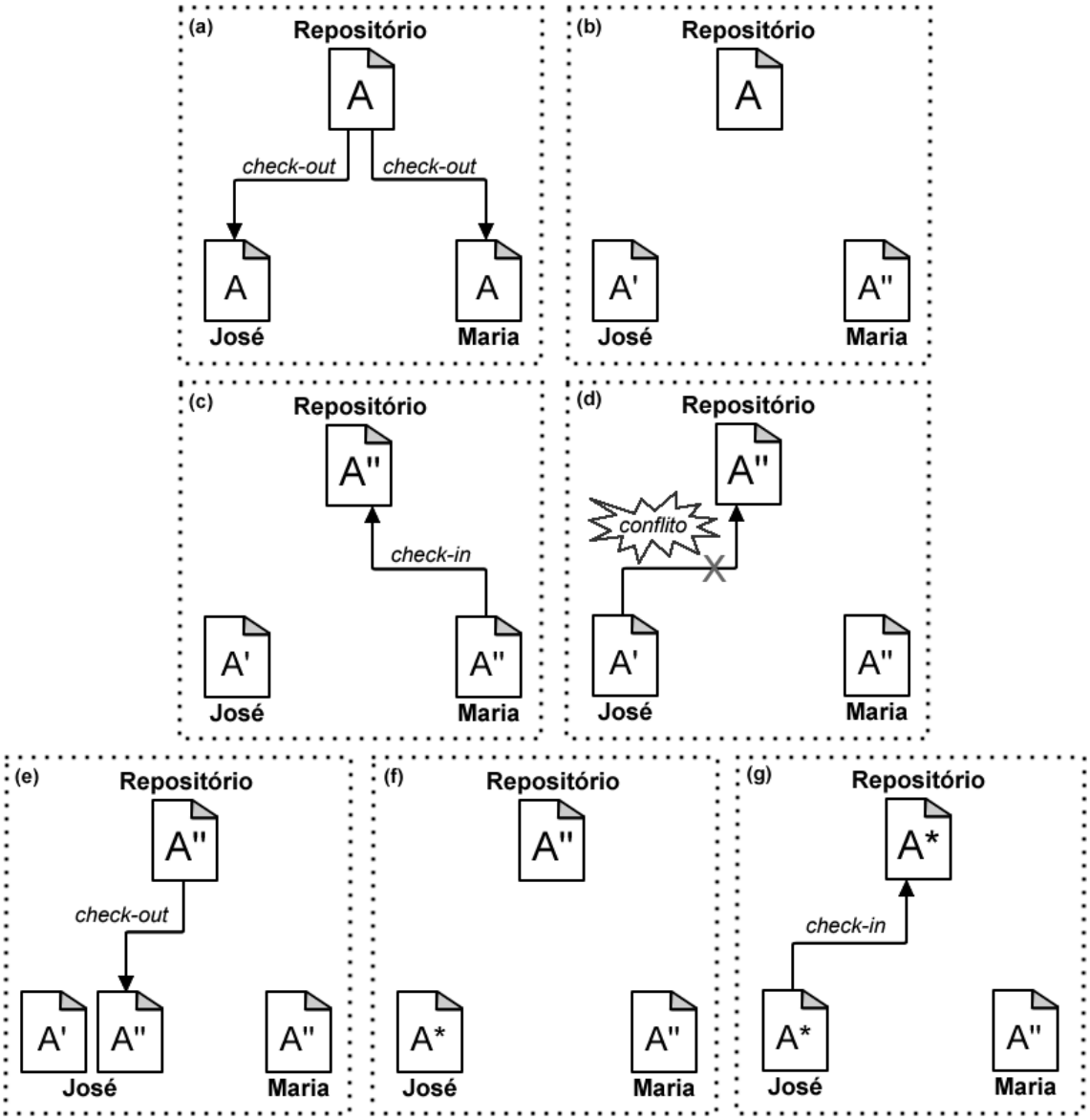


Figura 2.5. Política otimista [Adaptado de COLLINS-SUSSMAN *et al.*, (2004)].

A Figura 2.5 ilustra a utilização da política otimista. Nesse exemplo, os engenheiros de software José e Maria fazem *check-out* do mesmo IC A que está no repositório (Figura 2.5a). Os dois engenheiros modificam suas cópias em seus

respectivos espaços de trabalho e geram novas versões do IC A. José gera a versão IC A' e Maria gera a versão IC A'' (Figura 2.5b). Maria publica sua versão primeiro, ou seja, faz *check-in* do IC A. Nesse momento, o repositório é atualizado para a versão de Maria, A'' (Figura 2.5c). Mais tarde, José tenta salvar suas modificações no repositório, porém, como o IC A foi modificado desde o último *check-out* de José, as suas modificações podem não ser efetuadas, ocorrendo, neste caso, um conflito (Figura 2.5d). Para resolvê-lo, José precisa fazer o *check-out* da versão mais atual do IC A (Figura 2.5e) e, manualmente, gerar uma terceira versão do IC, A*, que representa a junção das versões modificada por José (A') e da versão modificada por Maria, que se encontra no repositório (A'') (Figura 2.5f). Somente depois disso, é possível fazer *check-in* e salvar as alterações feitas por José no repositório (Figura 2.5g).

A política otimista possui a vantagem de permitir o paralelismo entre os engenheiros de software. No entanto, tem a desvantagem de, quando ocorrem conflitos, o engenheiro precisar empregar tempo e esforço adicionais para resolvê-los, o que na maioria das vezes, precisa ser feito manualmente. Além disso, em geral, os engenheiros de software precisam entrar em contato uns com os outros para chegarem a um consenso e decidirem como montar a nova versão do IC, sem desfazer nenhuma modificação efetuada. Para que isso ocorra, os engenheiros de software precisam estar disponíveis no momento da resolução dos conflitos.

2.4.3 Comparação entre as políticas pessimista e otimista

De acordo com REDMILES *et al.* (2007), quanto mais ocorre trabalho paralelo, menor é a qualidade do código que é desenvolvido. Além disso, os engenheiros de software consideram os conflitos como algo ruim, pois, para resolvê-los, é necessário um esforço adicional ao desenvolvimento. Em conseqüência disso, os engenheiros podem entrar numa corrida para serem os primeiros a efetuar suas modificações no repositório, tentando evitar as tarefas de resolução dos conflitos e, conseqüentemente, tarefas de teste, aprovação e revisão dos ICs após o processo de junção.

Por outro lado, foram apresentadas no início desse capítulo, algumas razões para a adoção do trabalho concorrente, como, por exemplo, permitir uma construção mais ágil de um sistema, possibilitando que vários desenvolvedores trabalhem sobre o mesmo conjunto de ICs. Portanto, é importante analisar com atenção, quando é proveitosa ou não a permissão do paralelismo no processo de desenvolvimento por meio da escolha correta da política de controle de concorrência.

Existem situações onde uma determinada política é mais indicada do que a outra. Em casos de ICs em que a junção dos trabalhos tende a ser complexa, quando, por exemplo, os ICs não são textuais e a ferramenta que reconhece a representação binária dos ICs não oferece apoio automatizado para junções, pode ser mais aconselhado trabalhar usando a política pessimista. Por outro lado, nos casos onde os ICs são textuais e a junção não é complexa, a utilização da política otimista é mais indicada (ESTUBLIER, 2001).

A freqüência de conflitos depende da quantidade de engenheiros de software que trabalham concorrentemente sobre um mesmo IC. Em um trabalho prático, ESTUBLIER (2001) apresenta um cenário real de desenvolvimento de um grande projeto que utiliza a política otimista. O exemplo abordado pelo pesquisador consiste em um software de quatro milhões de linhas de código desenvolvido por uma equipe de 800 engenheiros de software. Neste contexto, em média três engenheiros de software atuam sobre o mesmo IC em paralelo, podendo chegar a até trinta engenheiros nos horários de pico, que ocorrem no início do dia e no final do dia. Em situações como esta, certamente a quantidade de ocorrência de conflitos aumenta e, conseqüentemente, existe uma demanda maior de tempo e esforço por parte dos engenheiros de software na resolução de conflitos.

O suporte à adoção de políticas de controle de concorrência pessimista e otimista varia nas diferentes soluções de sistema de controle de versão. O Visual SourceSafe e o RCS, por exemplo, adotam preferencialmente a utilização da política pessimista, por meio da utilização de bloqueios. O CVS, por outro lado, trabalha basicamente com a política otimista, oferecendo suporte limitado à adoção da estratégia pessimista. O Subversion, por sua vez, trabalha por padrão com a política otimista, mas permite também a utilização de política pessimista. Em particular, o Subversion apresenta um suporte mais maduro na adoção de políticas de controle de concorrência, pois com ele é possível fazer uma combinação de ambas as estratégias, selecionando para cada IC uma política diferenciada.

2.4.4 Demais políticas

Como foi mencionado anteriormente, as duas principais políticas de controle de concorrência são as pessimista e otimista (CONRADI e WESTFECHTEL, 1998). Contudo, existe na literatura a definição de outros tipos de políticas de controle de concorrência. Normalmente, elas adicionam outras funcionalidades às políticas

otimistas e pessimistas ou envolvem novos conceitos introduzidos por sistemas específicos, deixando as novas políticas dependentes de características específicas desses sistemas.

O sistema de controle de versão CVS provê um mecanismo que adiciona percepção[8] às políticas de controle de concorrência (BERLINER, 1990). O mecanismo é chamado de *watch* e possibilita ao engenheiro de software escolher um determinado conjunto de artefatos que ele deseja monitorar. Antes de modificar um artefato, os engenheiros de software devem anunciar sua intenção por meio de um comando de edição do CVS. Quando isso ocorre, um gatilho é acionado e todos os desenvolvedores que estão monitorando este artefato recebem, normalmente via email, uma notificação avisando sobre a intenção de modificações. Dessa forma, por meio deste mecanismo de percepção, o CVS incrementa a política otimista, propiciando aos engenheiros de software um maior conhecimento sobre as atividades que ocorrem em paralelo.

ESTUBLIER *et al.* (2001) propõem uma nova arquitetura para os sistemas de GCS. Alguns novos conceitos são definidos como, por exemplo, grupo de trabalho, que representa um grupo de engenheiros de software com o mesmo interesse. Além disso, são definidas novas políticas de controle de concorrência baseadas nesses novos conceitos da abordagem proposta pelos pesquisadores. As políticas apresentam diferentes regras, caso os desenvolvedores concorrentes estejam alocados ou não no mesmo grupo de trabalho. Assim, essas novas políticas somente podem ser utilizadas em conjunto com a abordagem para sistemas de GCS proposta pelos autores.

2.5 Trabalhos relacionados

Como foi apresentado neste capítulo, o controle de concorrência desempenha um papel importante no processo de desenvolvimento de software atual, principalmente em relação ao desenvolvimento distribuído.

A seguir são apresentados alguns trabalhos relacionados a esta dissertação que tratam, de alguma forma, sobre o trabalho concorrente. Esses trabalhos estão divididos em duas áreas: abordagens para o desenvolvimento de software e abordagens na área de banco de dados.

[8] Do termo em inglês *awareness*.

2.5.1 Abordagens para o desenvolvimento de software

ESTUBLIER (2001) afirma que as infra-estruturas de GCS não atentam às restrições de escalabilidade e eficiência necessárias ao controle de concorrência e, por isso, propõe uma nova arquitetura para os sistemas de GCS. Essa nova arquitetura vai de encontro às que existiam à época em que o trabalho foi desenvolvido. Entretanto, algumas das deficiências apontadas em sua pesquisa já são atendidas pelos atuais sistemas de GCS, como, por exemplo: arquitetura distribuída e possibilidade de definir diferentes políticas para diferentes artefatos de um projeto.

Para contornar o problema de escalabilidade, é proposta a criação de grupos onde um ambiente de trabalho serve como integrador para os demais ambientes de trabalho do grupo. Para estabelecer a mecânica da colaboração entre os ambientes de trabalho dentro de um grupo, foram definidas seis ações atômicas: modificar, propor, integrar, sincronizar, reservar e liberar. Além disso, foram definidas três políticas de controle de concorrência: exclusão, reserva tardia e reserva tardia com integração. Cada política é descrita por ações atômicas. Por exemplo, a política de exclusão é descrita por: sincronizar, reservar, modificar, propor, integrar e liberar.

Em outro trabalho (ESTUBLIER *et al.*, 2001), os autores apresentam uma complementação do trabalho anterior, e propõem uma linguagem para a definição das políticas utilizando o novo modelo de trabalho baseado em grupos. As ações que foram definidas permitem que o engenheiro de software crie novas políticas de controle de concorrência que podem variar desde políticas mais restritivas (uso de bloqueios) até outras mais flexíveis (sem bloqueios). Apesar de tratar de definição de novas políticas de controle de concorrência, esse trabalho não apóia a escolha da política mais indicada para cada IC do sistema.

MCCM (VAN DER LIGEN e VAN DER HOEK, 2004) é uma infra-estrutura de composição modular de políticas de GCS proposta para diminuir o esforço de criação de sistemas de GCS. Nessa abordagem, uma política de GCS é formada por diferentes módulos, entre eles, módulo de armazenamento, de composição, de hierarquia e de concorrência. Este último possibilita a utilização ou não de bloqueios nos artefatos. Uma das vantagens do MCCM é permitir que esses módulos sejam reutilizados na construção de novas políticas de GCS. Em relação às políticas de controle de concorrência, MCCM possibilita a definição e a reutilização, mas não se preocupa com a seleção de uma determinada política.

Pelo fato de nenhum desses trabalhos tratarem sobre a seleção e sugestão de políticas de controle de concorrência para cada artefato do sistema, a abordagem proposta nesta dissertação pode servir como complemento para as abordagens apresentadas.

Alguns outros trabalhos abordam o paradigma de Coordenação Contínua (REDMILES *et al.*, 2007) e buscam oferecer soluções para apoiar a coordenação do trabalho em paralelo no desenvolvimento de software distribuído. Essas abordagens buscam permitir a detecção antecipada dos conflitos e mitigar seus impactos. São exemplos desses trabalhos: Palantír (SARMA *et al.*, 2003) e Lighthouse (DA SILVA *et al.*, 2006).

Palantír é uma ferramenta de percepção que atua no espaço de trabalho dos engenheiros de software e provê informações sobre as modificações que estão em progresso nos espaços de trabalho remotos. Além disso, a ferramenta calcula uma medida de magnitude e impacto dessas modificações. Dessa forma, Palantír provê aos desenvolvedores o contexto de modificações e os ajuda na coordenação dos esforços de desenvolvimento, tentando diminuir o impacto que pode ser gerado por meio do trabalho concorrente.

Com objetivos semelhantes aos do Palantír, Lighthouse é uma plataforma de coordenação que busca detectar as situações de conflito assim que forem surgindo. A ferramenta utiliza o conceito de *design* emergente que possibilita uma representação atualizada do *design* do projeto. Para utilizar a abordagem, os autores indicam fortemente a utilização de mais de um monitor, a fim de permitir uma visualização adequada das informações oferecidas.

Nesses trabalhos de coordenação contínua, a perspectiva de percepção recebe importantes contribuições, no entanto, adicionam novas atividades para os engenheiros de software relacionadas à observação do que os outros engenheiros estão fazendo. Dessa forma, a efetividade da abordagem depende de como as informações geradas serão apresentadas, percebidas e compreendidas por cada engenheiro de software. Por outro lado, com o controle do trabalho concorrente por meio da seleção de diferentes políticas de controle de concorrência, proposto nesta dissertação, os desenvolvedores ficam “amarrados” à estratégia escolhida previamente e se concentram exclusivamente nas suas tarefas de desenvolvimento.

2.5.2 Abordagens na área de banco de dados

Na área de banco de dados, onde o controle de concorrência também exerce um papel fundamental, um trabalho se destaca na comparação com a abordagem proposta nesta dissertação. ESHEL *et al.* (2004) apresentam uma solução para trocar dinamicamente os diferentes tipos de mecanismos de controle de concorrência, incluindo aqueles baseados em bloqueio e os não baseados em bloqueios. Essas trocas fundamentam-se nos padrões de acesso e/ou nos requisitos da aplicação para cada arquivo e têm como objetivo o aumento do desempenho no compartilhamento dos dados, sendo, então, feitas em tempo real. Isso é possível porque, no âmbito de banco de dados, as transações são usualmente curtas. Por outro lado, no contexto de desenvolvimento de software, as transações são longas, podendo durar horas ou dias, o que torna inviável essa troca dinâmica. O trabalho de ESHEL *et al.* mostra que a questão de seleção de mecanismos para o controle de concorrência é importante na área de banco de dados, e, fornece indícios de que, assim como na área de banco de dados, essa questão também se apresenta relevante para a área de desenvolvimento de software.

2.6 Considerações finais

Neste capítulo, foi apresentada a importância do trabalho concorrente no cenário atual de desenvolvimento de software, discutindo-se ainda como a GCS pode apoiar esse processo. Foram discutidos também mecanismos para prover gerenciamento sobre repositórios de controle de versões por meio de políticas de controle de concorrência e quais as características desses mecanismos.

Alguns trabalhos propõem soluções para a definição de políticas de controle de concorrência, alguns incrementam essas políticas com preocupações adicionais, como percepção, por exemplo, mas nenhum deles trata da questão de qual política deve ser utilizada para cada artefato do sistema, visando um aumento na produtividade.

Foi apresentado, também, que essa preocupação com a seleção de mecanismos para controle de concorrência foi abordada na área de banco de dados e que, no entanto, devido a peculiaridades das áreas, o tratamento dessa questão no âmbito de desenvolvimento de software precisa ser diferenciado.

No próximo capítulo, é proposta uma abordagem para seleção de políticas de controle de concorrência no contexto de desenvolvimento de software, chamada Orion.

Capítulo 3 – A Abordagem Orion

3.1 Introdução

No Capítulo 2, foi apresentada a revisão da literatura sobre controle de concorrência e foram discutidas abordagens que, de alguma forma, tratam sobre políticas de controle de concorrência. A partir desse estudo, foi possível identificar as principais deficiências no apoio à seleção de políticas de controle de concorrência no processo de desenvolvimento de software. Foi visto que essas abordagens, ou apóiam a definição de novas políticas de controle de concorrência, levando em consideração novos conceitos, ou buscam evitar situações de conflito, por meio de percepção, assumindo que a política otimista é sempre a estratégia utilizada, não levando em consideração a adoção da estratégia pessimista.

A análise das deficiências identificadas motivou a identificação dos principais requisitos para a abordagem proposta, que são:

1. Identificação de informações históricas, disponibilizadas pelos sistemas de controle de versão mais utilizados, que são úteis para fazer a seleção de políticas de controle de concorrência;
2. Identificação de informações históricas adicionais, que normalmente não são disponibilizadas pelos sistemas de controle de versão, mas que também seriam úteis para fazer a seleção de políticas de controle de concorrência;
3. Mecanismo de seleção de políticas de controle de concorrência para cada IC do projeto de desenvolvimento;
4. Mecanismo de identificação dos ICs que merecem uma maior atenção em relação ao controle de concorrência, os chamados elementos críticos;
5. Mecanismo de visualização dos elementos críticos e das políticas de controle de concorrência selecionadas para cada IC.

Neste capítulo, é apresentada a abordagem proposta, intitulada Orion (PRUDENCIO *et al.*, 2007a; PRUDENCIO *et al.*, 2007b), que visa fornecer mecanismos e ferramental de apoio para a seleção de políticas de controle de concorrência no processo de desenvolvimento de software, atendendo aos requisitos supracitados.

A Figura 3.1 apresenta uma visão geral da abordagem Orion. Os repositórios dos sistemas de controle de versão são utilizados como fontes de dados. A partir daí, são extraídos os dados históricos referentes ao projeto de software que está sendo desenvolvido e, em seguida, são filtradas as informações pertinentes à abordagem. Informações estas que incluem todas as versões dos ICs do projeto e dados sobre estas versões, como, por exemplo, o momento em que o desenvolvedor fez o *check-in* desta versão. Após a coleta e análise dessas informações, algumas métricas são calculadas e armazenadas em um repositório local, diferente do repositório da fonte de dados. Os resultados dessas métricas são utilizados para fazer a seleção de políticas de controle de concorrência e identificar os elementos críticos. Por fim, as informações geradas pela abordagem podem ser visualizadas pelos usuários por meio de ferramentas de visualização.

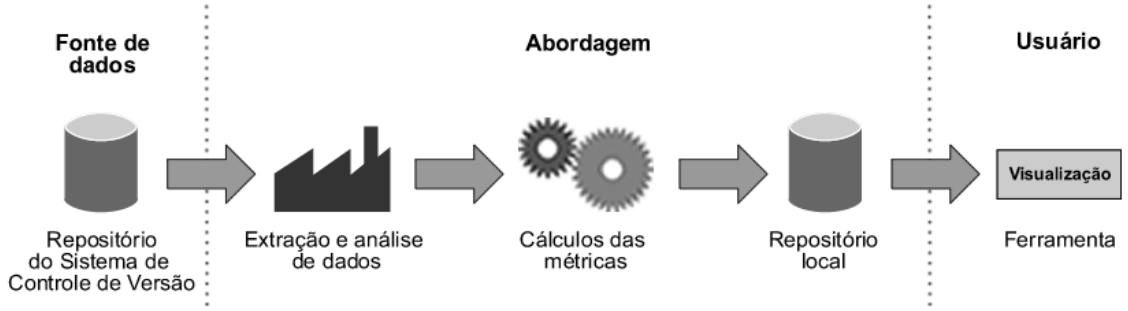


Figura 3.1. Visão geral da abordagem proposta.

A utilização da abordagem consiste em, inicialmente, uma atividade de configuração inicial, onde o usuário define alguns pontos de corte, discutidos na Seção 3.2.4, que são utilizados para chegar aos resultados finais: a seleção das políticas de controle de concorrência e a identificação dos elementos críticos. Em seguida, é necessário identificar o período histórico que a abordagem Orion deve analisar. Neste passo, o usuário pode querer analisar todo o histórico de modificações, desde o início do projeto, ou escolher um intervalo mais curto. Após a segunda atividade, o usuário deve escolher um dos dois modos de operação da abordagem. No primeiro deles, o usuário escolhe apenas um IC do projeto para obter os resultados. No segundo modo de operação, um projeto é escolhido e todos os ICs desse projeto são analisados. Por fim, o usuário visualiza os resultados da aplicação da abordagem (Figura 3.2).

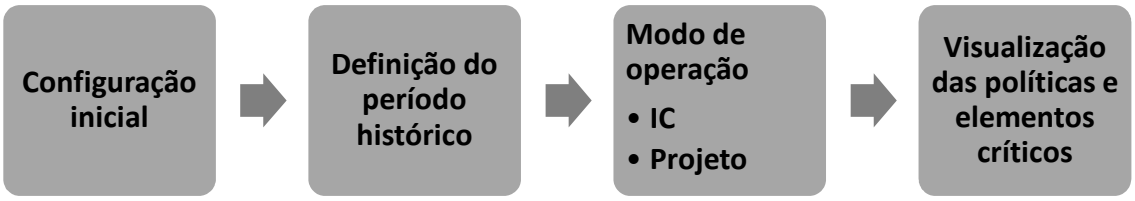


Figura 3.2. Visão do uso da abordagem Orion.

Dessa forma, a abordagem Orion permite que o gerente de configuração de software, por exemplo, possa impor que os engenheiros de software adotem determinada estratégia escolhida, por meio dos mecanismos oferecidos pelos sistemas de controle de versão. Permitindo, assim, que cada um deles se foque no trabalho e se preocupe menos com a questão do paralelismo no desenvolvimento de software e suas conseqüências.

Este capítulo está dividido da seguinte forma: a Seção 3.2 detalha a abordagem Orion; a Seção 3.3 apresenta um exemplo de aplicação dessa abordagem; e, por último, a Seção 3.4 apresenta as considerações finais e discute algumas limitações da abordagem proposta.

3.2 Abordagem proposta

Como discutido no Capítulo 2, a política otimista é importante, pois possibilita que vários desenvolvedores trabalhem sobre o mesmo conjunto de artefatos. No entanto, a adoção dessa estratégia pode acarretar situações de conflito entre as modificações realizadas pelos engenheiros de software. Conflitos que, na maioria das vezes, demandam tempo e esforço adicionais para serem resolvidos. Dependendo do tamanho desses conflitos, do conhecimento dos engenheiros e da comunicação entre eles, a tarefa de resolução de conflitos pode representar um impacto significante na produtividade da equipe de desenvolvimento. Dessa forma, os conflitos podem ser considerados riscos potenciais ao processo de desenvolvimento.

Por outro lado, a adoção da estratégia pessimista, apesar de impor a serialização dos esforços dos engenheiros de software sobre o mesmo IC, é importante, pois evita a ocorrência dos conflitos.

Em um sistema que está em desenvolvimento, alguns ICs são mais acessados e modificados pelos engenheiros de software. Podem ser ICs que representam elementos essenciais do sistema, ou ainda, simplesmente ICs que, por algum motivo precisam ser atualizados rotineiramente pelos desenvolvedores. Existem também nos projetos, ICs

que comumente sofrem maior quantidade de conflitos. Conflitos que podem ser fáceis de serem resolvidos ou podem ser mais difíceis, o que traz um maior impacto no desempenho dos desenvolvedores. Dessa forma, os ICs que possuem um histórico relevante de conflitos e que demandaram grande esforço e tempo dos engenheiros de software são tratados nesta abordagem como *elementos críticos*.

Esta seção está organizada da seguinte forma: na Seção 3.2.1, são apresentadas as informações históricas normalmente disponibilizadas pelos sistemas de controle de versão, além disso, são identificadas informações históricas adicionais que também são úteis na seleção de políticas de controle de concorrência; na Seção 3.2.2 são definidas e descritas duas métricas de controle de concorrência; na Seção 3.2.3, os elementos críticos são apresentados e é discutido como eles são identificados; na Seção 3.2.4, é apresentado o mecanismo de seleção de políticas de controle de concorrência; na Seção 3.2.5 é detalhada a questão da visualização dos resultados gerados pela abordagem; finalmente, na Seção 3.2.6, são discutidos o público-alvo e a aplicação da abordagem Orion no processo de desenvolvimento de software.

3.2.1 Informações históricas

Como foi mostrado no Capítulo 2, uma das funções principais dos sistemas de controle de versão é prover um histórico de modificações de todos os ICs do projeto. A maioria das soluções existentes dessa classe de sistemas, como, por exemplo, CVS, Subversion, IBM Rational ClearCase e Microsoft Visual SourceSafe, fornecem informações sobre os *check-ins* feitos no repositório. Essas informações incluem, para cada *check-in*, por exemplo, o identificador (número da versão do IC ou do repositório, após a operação de *check-in*); o autor; a data; os ICs modificados; a ação feita em cada IC; e a mensagem do autor das modificações.

A Figura 3.3 representa o histórico de modificações de um determinado projeto armazenado em um repositório do Subversion, utilizando a ferramenta cliente TortoiseSVN (Tigris, 2008). Na Figura 3.3, são mostradas as informações históricas providas por esse sistema de controle de versão, discutidas no parágrafo anterior.

Basicamente, essas são as únicas informações históricas armazenadas e disponibilizadas aos usuários nos principais sistemas de controle de versão e, portanto, as únicas informações que podem ser utilizadas para a seleção de políticas de controle de concorrência. Essas são informações úteis para esse fim, no entanto, não são suficientes.

Foram realizadas entrevistas informais com especialistas e percebeu-se que as informações histórias armazenadas e disponibilizadas nos sistemas de controle de versão em geral, não eram suficientes para tomar decisões relacionadas à seleção de políticas de controle de concorrência. Esses especialistas afirmaram que para executar essa tarefa, eles levam em consideração conhecimentos adquiridos ao longo do projeto de desenvolvimento a respeito dos conflitos ocorridos.

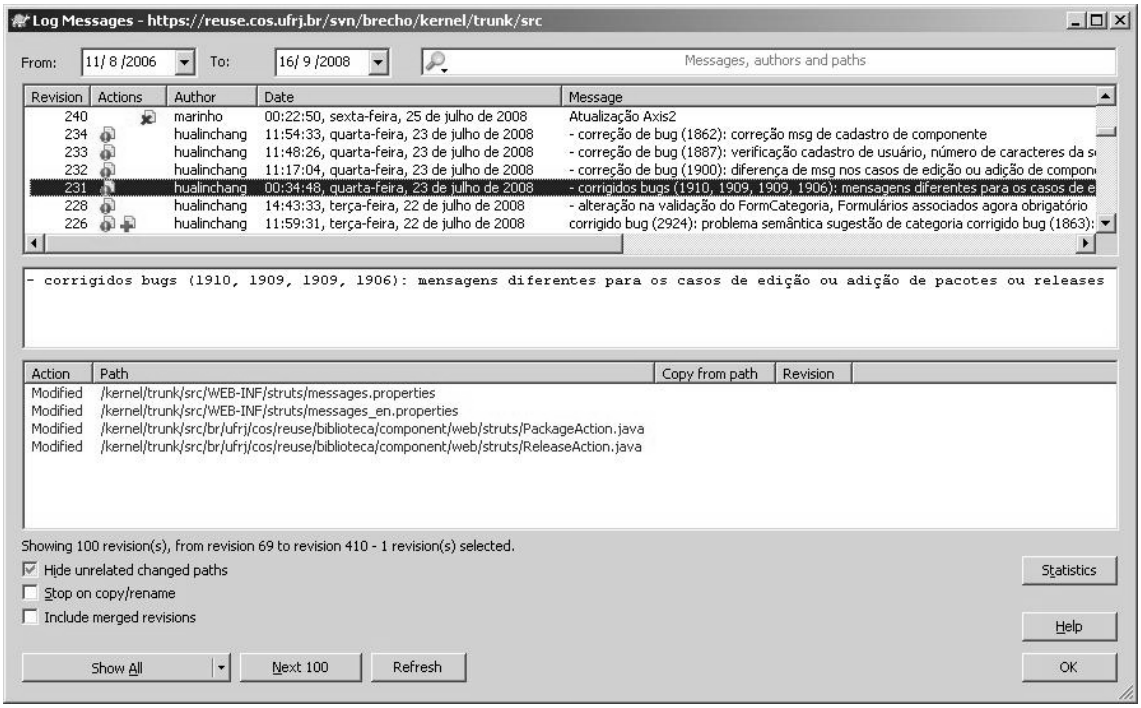


Figura 3.3. Visualização do histórico de um repositório Subversion pela ferramenta TortoiseSVN.

Em posse somente dessas informações históricas apresentadas na Figura 3.3, não é possível saber para cada IC, por exemplo:

- Se ocorreram conflitos;
- Quando ocorreram conflitos;
- Quais foram os conflitos que ocorreram;
- Em que momentos houve concorrência no desenvolvimento;
- Quantos desenvolvedores concorreram; e
- Qual foi a duração de uma determinada modificação.

Certamente, com os dados gerados pelos sistemas de controle de versão, esses pontos poderiam ser conhecidos, porém esses dados não são armazenados por esses sistemas. Um dos motivos possíveis pelo qual essas informações não são persistidas é não aumentar o tamanho dos repositórios e otimizar o uso do disco rígido nos

servidores. Outro motivo, menos provável, é não ter sido identificada nenhuma aplicação que faça uso de informações como essas.

Conhecer essas informações levantadas, no entanto, poderia ser muito útil para determinar qual política de controle de concorrência devemos adotar para cada IC do projeto, sem depender da experiência e do conhecimento prévio do engenheiro de software a respeito do projeto de desenvolvimento. Para que os usuários pudessem ter conhecimento dessas informações, foram identificados os seguintes dados que deveriam ser coletados, armazenados e disponibilizados pelos sistemas de controle de versão:

- *Data do check-out ou atualização dos ICs*. Com esse dado, seria possível determinar o período que um determinado IC ficou em posse dos desenvolvedores e, portanto, o intervalo de tempo em que foi efetuada uma modificação. Além disso, é possível saber em que momentos houve concorrência e quantos desenvolvedores concorreram sobre um IC;
- *Tentativas de check-ins sem sucesso*. Por meio desse dado, poderíamos saber se ocorreram conflitos e quando eles ocorreram;
- *Versões dos ICs que os desenvolvedores tentaram fazer check-in, sem sucesso*. Em posse dessas versões, podemos conhecer a "intenção" do desenvolvedor na sua tentativa de *check-in* e, com isso, saber exatamente quais foram os conflitos que ocorreram.

No decorrer deste capítulo, esses três dados são levados em consideração e também são utilizados como insumos para o cálculo das métricas propostas.

3.2.2 Métricas para seleção de políticas de controle de concorrência

Nesta seção, são descritas duas métricas, que são utilizadas para fazer a seleção de políticas de controle de concorrência e identificar os elementos críticos do sistema.

3.2.2.1 Definição das métricas

A IEEE Std 1540 (IEEE, 2001) define risco como a probabilidade de um evento, perigo, ameaça ou situação ocorrer, associado às conseqüências indesejáveis, ou seja, um problema potencial. Neste contexto, o risco tratado por esse trabalho é a perda de produtividade, ocasionado pelos conflitos e pelos elementos críticos. De acordo com a disciplina de gerência de riscos (BOEHM, 1991), os riscos que devem receber maior atenção em um projeto de software são os que têm maior probabilidade de acontecer e que têm potencial de causar um maior impacto no projeto. Assim, são estipulados

valores referentes à probabilidade e ao impacto desse risco. Em seguida, esses valores são multiplicados e um terceiro valor, chamado de exposição, é obtido (Fórmula 1). A exposição indica o quão preocupante é o risco (SOFTEX, 2007b).

$$\boldsymbol{exposição = probabilidade \times impacto} \qquad \textbf{(Fórmula 1)}$$

Utilizando como base os conhecimentos de gerência de riscos, a abordagem Orion define duas métricas e as utiliza para calcular um valor que representa o quão crítico é cada IC do sistema. Este valor leva em consideração a combinação das duas métricas, que são: nível de *concorrência* de um IC e nível de *dificuldade de junção* do IC.

A Tabela 3.1 apresenta uma relação entre as métricas que são definidas nesta abordagem e as métricas utilizadas na disciplina de gerência de riscos para calcular a exposição de um risco. A métrica de concorrência mede o nível de concorrência dos engenheiros de software sobre um determinado IC. Assim como a probabilidade do risco na gerência de riscos, quanto maior o nível de concorrência, maior a probabilidade de ocorrer conflitos (PERRY *et al.*, 2001), e, conseqüentemente, haver perda de produtividade da equipe de desenvolvimento. A métrica de dificuldade de junção mede o nível de esforço necessário para resolver conflitos em um determinado IC. Da mesma forma que o impacto do risco na gerência de riscos, quanto mais difícil for a junção, maior será o impacto na produtividade da equipe. De forma resumida, o risco, na gerência de riscos, corresponde à perda de produtividade, na abordagem Orion. Então, a probabilidade e o impacto estão para a gerência de riscos, assim como as métricas concorrência e dificuldade de junção estão para a abordagem Orion, respectivamente.

Tabela 3.1. Relação entre as métricas definidas e a gerência de riscos.

Gerência de Riscos	Abordagem Orion
Risco	Perda de produtividade
Probabilidade	Concorrência
Impacto	Dificuldade de junção

Ambas as métricas são calculadas com base nos dados históricos dos ICs armazenados pelos sistemas de controle de versão do projeto. A seguir, as métricas são detalhadas.

3.2.2.2 Métrica de concorrência

Como foi discutido no Capítulo 2, para fazer uma modificação em um determinado IC do projeto, o engenheiro de software deve proceder da seguinte forma: (1) executar a operação de *check-out* dos elementos que serão modificados ou de uma parte do software que inclua os elementos a serem modificados, e, dessa forma, trazê-los para seu espaço de trabalho; (2) em seguida, executar a modificação propriamente dita, na maioria das vezes, utilizando um ambiente de desenvolvimento, como o Eclipse (ECLIPSE FOUNDATION, 2008a), por exemplo; (3) depois de finalizadas as alterações, executar a operação de *check-in*. Somente os elementos que de fato foram modificados são salvos no repositório. Para permitir a execução dessa operação, o sistema de controle de versão faz uma comparação das versões obtidas no *check-out* de todos os elementos com as novas versões geradas após a intervenção do desenvolvedor.

Para calcular o nível de concorrência de um determinado IC, é analisado todo o histórico das operações de *check-in* e suas respectivas operações de *check-out*. Podem existir operações de *check-out* que não resultaram em uma alteração efetiva do IC, ou seja, não resultaram na realização de uma operação de *check-in*. Portanto, estas operações não são consideradas.

No contexto desta abordagem, leva-se em consideração que a duração da atividade de execução de um conjunto de modificações feitas por um desenvolvedor em um IC é determinada pelo período entre o momento do *check-out* de um IC e o momento do *check-in*. Dessa forma, utilizando os dados históricos dos sistemas de controle de versão, é possível identificar quanto tempo um determinado IC ficou em posse de um desenvolvedor para modificações.

A Figura 3.4 apresenta uma visualização das informações discutidas no parágrafo anterior. Este é um exemplo da representação das informações de um histórico de modificações de um sistema de controle de versão. A figura representa as modificações sobre um determinado IC do período de 0 a 10 unidades de tempo. Neste intervalo, o IC foi modificado por três desenvolvedores (A, B e C). As linhas pretas iniciam na operação de *check-out* (círculo preto) e terminam na operação de *check-in* (círculo branco). Conseqüentemente, essas linhas indicam o período de posse do IC por um desenvolvedor para modificações. O desenvolvedor A, por exemplo, em dois momentos executou alterações sobre este IC: a primeira do tempo 1 ao 3 e a segunda do tempo 4 ao 9.

Utilizando a figura, é possível identificar três momentos distintos do tempo de vida do IC: (1) momentos em que nenhum desenvolvedor estava em posse deste IC para modificação, como, por exemplo, os períodos de 0 a 1 e 9 a 10; (2) momentos em que apenas um desenvolvedor estava em posse deste IC, ou seja, momentos em que não houve concorrência sobre o IC, por exemplo, os períodos de 1 a 2 e 3 a 4; e, por último, (3) momentos onde mais de um desenvolvedor estava com a posse deste IC, ou seja, momentos em que houve concorrência, como, por exemplo, os períodos de 2 a 3 e 4 a 9.

Figura 3.4. Situação para análise do nível de concorrência.

Para fazer o cálculo do nível de concorrência, são levados em consideração o tempo em que mais de um desenvolvedor teve posse do item para modificação, chamado *período de concorrência*, e o tempo em que ao menos um desenvolvedor manteve o item em sua área de trabalho para modificação, chamado *tempo de posse*.

Na Figura 3.4, o período de concorrência é realçado com uma linha preta grossa na linha do tempo de vida do IC. Já o período de posse é sinalizado por uma linha preta na linha do tempo de vida do IC, que pode ser grossa ou fina, pois o período de concorrência obrigatoriamente está contido no período de posse. Os períodos em que o IC não ficou em posse de nenhum desenvolvedor são visualizados por meio de linhas na cor cinza.

Com isso, para um dado IC, a métrica de concorrência é calculada, levando em consideração um período determinado de tempo (Fórmula 2):

$$\boldsymbol{concorrência(IC)} = \frac{\boldsymbol{TC(IC)}}{\boldsymbol{TP(IC)}} \qquad \textbf{(Fórmula 2)}$$

onde TC(IC) significa o tempo que houve concorrência sobre o IC e TP(IC) representa o tempo em que o IC ficou em posse de pelo menos um desenvolvedor no período analisado. Dessa forma, o nível de concorrência é determinado por um valor normalizado (i.e., entre 0 e 1, inclusive), já que o tempo de concorrência é sempre

menor ou igual ao tempo de posse. Quanto mais próximo de 1, maior o nível de concorrência desse item. A aplicação dessa fórmula é exemplificada na Seção 3.3.

3.2.2.2.1 Análise crítica sobre a métrica de concorrência

A métrica de concorrência foi apresentada para alguns especialistas e foram apontados alguns pontos discrepantes no cálculo da métrica. Nesta seção, é feita uma discussão crítica a respeito da métrica de concorrência.

Como foi visto no cálculo da métrica de concorrência, o intervalo de tempo de um conjunto de modificações é determinado pelo intervalo entre o *check-out* (ou atualização) e o *check-in* dos ICs. Esse intervalo de tempo significa o tempo em que os ICs ficaram em posse do desenvolvedor para modificações. No entanto, não significa que esse foi o tempo real das modificações.

Em uma situação onde um desenvolvedor faz a atualização do seu espaço de trabalho, mas só inicia suas modificações algumas horas depois, o cálculo do tempo de modificação será maior do que a realidade. Por outro lado, essa situação pode ser evitada se a equipe de desenvolvimento tiver o hábito de sempre iniciar alterações em versões atualizadas dos ICs. Esse simples cuidado também pode evitar conflitos no momento do *check-in*. Outra saída para esse problema é utilizar um mecanismo de aviso de início de modificação, presente em alguns sistemas de controle de versão, como, por exemplo, o mecanismo *watch* do CVS (O'REILLY *et al.*, 2003). Esse mecanismo envia uma notificação quando o desenvolvedor inicia realmente sua modificação no IC.

Porém, se o desenvolvedor faz uma pausa entre o início e o fim da modificação, esse intervalo não é detectado pelo cálculo da métrica, e também não seria detectado com os mecanismos discutidos no parágrafo anterior.

Outro aspecto levado em consideração no cálculo da métrica de concorrência é o fato de que todos ICs modificados em um *check-in* possuem o mesmo tempo de posse (ou modificação). Em alguns casos, esse aspecto pode não ser verdadeiro. Um *check-in* pode incluir modificações em um ou vários ICs. Em casos onde mais de um IC foi alterado, não necessariamente o tempo deles foi o mesmo. Para atender a essas situações, a utilização de um mecanismo de aviso de início de modificação seria útil.

Outro ponto importante a ser discutido é o fato de que a quantidade de desenvolvedores concorrentes não está sendo levada em consideração no cálculo da métrica. O que está sendo considerado é se houve concorrência ou não.

As métricas definidas neste trabalho visam ser abrangentes e não focar em um determinado sistema de controle de versão. Supondo que a equipe de desenvolvimento sempre faça uma atualização dos ICs antes das modificações, acreditamos que a métrica de concorrência forneça resultados não muito distantes dos resultados reais.

A questão da quantidade de desenvolvedores concorrentes foi considerada em um determinado momento da pesquisa, mas deu origem a uma fórmula bem mais complexa do que a apresentada. A decisão foi adotar uma fórmula mais simples que venha a ser aperfeiçoada, assim que forem realizados testes mais profundos com a métrica de concorrência.

3.2.2.3 Métrica de dificuldade de junção

Antes de detalhar a métrica de dificuldade de junção, é necessário definir alguns novos conceitos. Nesta abordagem, cada IC é tratado como uma unidade de versionamento (UV), definida como um elemento atômico associado ao controle de versão. Cada UV, por sua vez, é composto por sub-ICs, chamados de unidades de comparação (UCs), que representam elementos atômicos usados para comparação de UVs e para computação de conflitos (MURTA *et al.*, 2007). Dessa forma, a seleção dos ICs permite ao engenheiro de software definir o nível de granularidade dos elementos e analisar o sistema em diferentes níveis de abstração.

A Figura 3.5 apresenta um exemplo de UVs e UCs. Caso documentos texto sejam considerados UVs, parágrafos ou frases desse documento podem ser considerados UCs. Em outro exemplo, considerando o paradigma de programação orientada a objetos, quando uma classe Java é considerada UV, os métodos desta classe podem ser definidos como UCs.

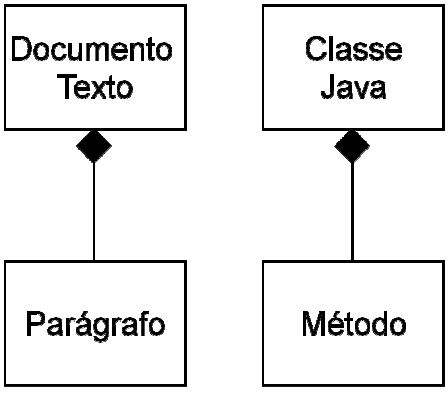


Figura 3.5. Unidades de versionamento compostas por unidade de comparação.

Definidos esses conceitos, voltemos ao cálculo da métrica de dificuldade de junção. Esta métrica quantifica a dificuldade de se fazer a junção de duas versões de um

IC. Como vimos no Capítulo 2, a junção é a forma utilizada para resolver conflitos que possam ocorrer. Sempre a junção é feita entre duas versões: a versão que o engenheiro de software tentou salvar no repositório, por meio da operação de *check-in*, e a versão do IC no repositório, no momento do *check-in*.

Considerando que houve uma situação de conflito, a dificuldade de resolução desse conflito é quantificada, com um valor que também varia de 0 a 1. Quanto mais próximo de 1, maior a dificuldade de se fazer a junção.

O cálculo desta métrica leva em consideração a quantidade de UCs do IC que sofreram conflitos. Considere uma classe Java com 10 métodos, por exemplo. Depois de fazer alterações, o engenheiro de software tentou fazer o *check-in* desta classe e foi impedido devido à existência de conflitos neste IC. Independente do número de conflitos que ocorram no IC, a operação de *check-in* é abortada. Nesta situação, para o cálculo da métrica de dificuldade de junção, serão analisados quantos dos 10 métodos (UCs do IC) sofreram algum conflito.

Para poder contabilizar e analisar quais as UCs que sofreram conflito e, conseqüentemente, permitir o cálculo da métrica de dificuldade de junção, diferentes versões do IC devem ser analisadas. No momento imediatamente posterior ao desenvolvedor fazer o *check-in* de um determinado IC, podem ser destacadas quatro versões deste item (Figura 3.6):

- *Versão original*, versão do IC que o desenvolvedor fez *check-out*;
- *Versão do usuário*, versão do IC resultante depois de aplicadas as modificações feitas pelo desenvolvedor na sua área de trabalho e representa a versão que foi feito *check-in*;
- *Versão atual*, versão do IC que está no repositório no momento imediatamente anterior ao *check-in* e que pode ser diferente da original, visto que podem ter havido *check-ins* de outros desenvolvedores;
- *Versão final*, versão do IC que foi salva no repositório após uma operação de *check-in* bem sucedida. Caso tenha ocorrido algum conflito e este tenha sido solucionado, esta versão representa a junção da versão do usuário com a versão atual.

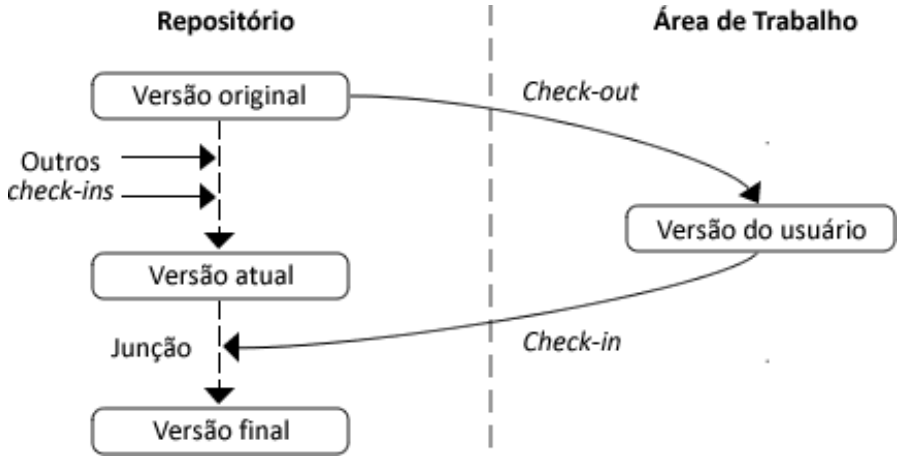


Figura 3.6. Versões de um IC no momento posterior do *check-in*.

Para medir a dificuldade de junção de uma determinada operação de *check-in*, é preciso identificar as ações necessárias por parte do desenvolvedor para resolver os conflitos. Essas ações adicionais foram aplicadas na versão final do IC, já que esta representa a versão após a resolução dos conflitos. Uma ação pode consistir em uma modificação, adição ou remoção das UCs do IC.

Dessa forma, após a identificação dessas quatro versões para cada *check-in*, é possível destacar dois conjuntos de ações:

- *Ações aplicadas*, ações executadas pelos desenvolvedores nas UCs do IC desde a sua versão original. Para o cálculo dessa primeira informação, são identificadas as ações efetuadas na versão do usuário (VU) e na versão atual (VA), tendo como base a versão original (VO) do item. Para isso, é utilizado o algoritmo *diff3*, apresentado no Capítulo 2 (Fórmula 3).

$$\boldsymbol{ações\ aplicadas}(\boldsymbol{IC}) = \boldsymbol{diff3}(\boldsymbol{VU}, \boldsymbol{VA}, \boldsymbol{VO}) \qquad \textbf{(Fórmula 3)}$$

- *Ações efetivadas*, ações executadas pelos desenvolvedores e que realmente foram efetivadas na versão final. Para encontrar esse conjunto são comparadas a versão final (VF) e a versão original (VO) das UCs, utilizando o algoritmo *diff*, apresentado no Capítulo 2 (Fórmula 4).

$$\boldsymbol{ações\ efetivadas}(\boldsymbol{IC}) = \boldsymbol{diff}(\boldsymbol{VO}, \boldsymbol{VF}) \qquad \textbf{(Fórmula 4)}$$

Fazendo a intersecção desses dois conjuntos de ações (Figura 3.7), três subconjuntos podem ser destacados:

- *Esforço desperdiçado*, subconjunto que inclui as ações que foram feitas pelos desenvolvedores, mas devido a alguma razão, na tarefa de resolução de conflitos, foram descartadas e não estão presentes na versão final do IC.

Estas ações representam esforços que foram desperdiçados pelo desenvolvedor que resolveu os conflitos e não foram efetivadas na versão final. Este subconjunto é encontrado fazendo a diferença entre o conjunto *ações aplicadas* e o conjunto *ações efetivadas*;

- *Nenhuma dificuldade*, subconjunto representado pelas ações que foram executadas pelos desenvolvedores e que estão presentes na versão atual do IC e na versão após a resolução de conflitos. Estas ações foram aceitas pelo desenvolvedor que fez o *check-in* e não trouxe a ele dificuldade adicional na resolução dos conflitos. Este subconjunto é encontrado fazendo a interseção entre o conjunto *ações aplicadas* e o conjunto *ações efetivadas*;
- *Esforço adicional*, subconjunto que engloba as ações que foram feitas pelo desenvolvedor que fez o *check-in* com o objetivo de resolver os conflitos que por ventura surgiram. Estas ações representam esforços adicionais necessários para a resolução dos conflitos. Este subconjunto é encontrado fazendo a diferença entre o conjunto *ações efetivadas* e o conjunto *ações aplicadas* e simboliza o efetivo esforço necessário para fazer a junção das versões do IC.

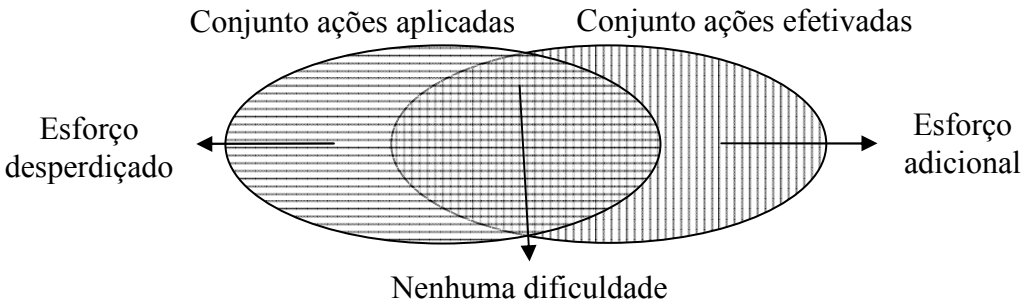


Figura 3.7. Representação da intersecção dos conjuntos ações aplicadas e ações efetivadas.

Dessa forma, a métrica de dificuldade de junção é calculada levando em consideração a proporção entre a quantidade de ações destinadas à resolução de conflitos (*esforço adicional*) e a quantidade de ações efetivadas na versão final do IC (Fórmula 5):

$$\boldsymbol{dificuldade\ de\ junção(check_in, IC) = \frac{|(AE - A)|}{|AE|}} \qquad \textbf{(Fórmula 5)}$$

onde |X| representa o número de elementos do conjunto X, AE representa o conjunto *ações efetivadas*, A representa o conjunto *ações aplicadas* e "–" representa a operação de diferença de conjuntos.

Em casos onde não houve conflitos ou a resolução destes não demandou nenhum esforço adicional (AE – A = Ø), que representa casos onde o desenvolvedor aceita a versão atual do IC ou impõe sua versão, a métrica de dificuldade de junção terá resultado igual a 0. A Figura 3.8 ilustra uma situação onde a métrica de dificuldade de junção é nula. Neste exemplo, todas as ações efetivadas foram ações aplicadas pelos desenvolvedores (conjunto AE está contido no conjunto A), conseqüentemente, não ocorreram ações adicionais necessárias para resolução de conflitos.

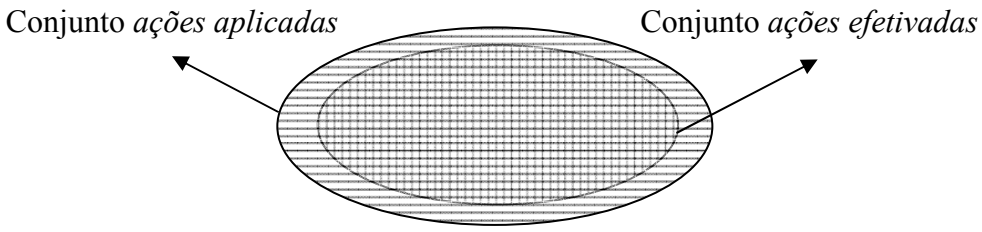


Figura 3.8. Dificuldade de junção igual a 0.

No extremo oposto, quando todas as ações que resultaram na versão final do IC representarem esforços adicionais (AE – A = AE), a métrica de dificuldade de junção terá resultado igual a 1, o valor máximo. A Figura 3.9 mostra uma situação onde a métrica de dificuldade de junção tem valor máximo. Neste exemplo, todas as ações dos desenvolvedores foram descartadas e um conjunto distinto de ações foi executado para solucionar os conflitos ocorridos (conjuntos A e AE disjuntos).

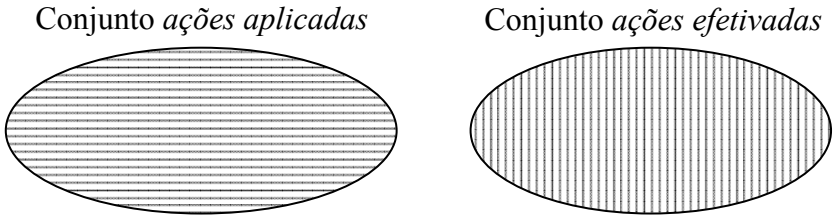


Figura 3.9. Dificuldade de junção igual a 1.

É importante enfatizar que a métrica de dificuldade de junção é calculada para cada operação de *check-in*. Então, para calcular o valor desta métrica de um IC para um determinado período de tempo, calcula-se a métrica para cada operação de *check-in* desse IC e faz-se então a média aritmética desses valores (Fórmula 6):

$$\boldsymbol{dificuldade\ de\ jun}\text{ção}(\boldsymbol{IC}) = \frac{\sum \boldsymbol{dificuldade\ de\ jun}\text{ção}(\boldsymbol{check_in,\ IC})}{\boldsymbol{n}\text{úmero de check_ins}} \qquad \textbf{(Fórmula 6)}$$

Considere, por exemplo, uma classe que possui 10 *check-ins* em um determinado período. Para cada um desses *check-ins*, são identificadas as quatro versões (versão original, do usuário, atual e final) discutidas acima e, em seguida, os conjuntos de *ações aplicadas* e *ações efetivadas*. Com essas informações, é calculada a métrica de dificuldade de junção para cada *check-in*. Esses dez valores são somados e divididos por dez e, finalmente, chega-se ao resultado da métrica de dificuldade de junção para esta classe neste período de tempo.

3.2.2.3.1 Análise crítica sobre a métrica de dificuldade de junção

Assim como foi feito para a métrica de concorrência, a métrica de dificuldade de junção foi apresentada a especialistas que indicaram alguns pontos discrepantes no cálculo da métrica. Nesta seção, é feita uma discussão crítica a respeito da métrica de dificuldade de junção.

O cálculo da métrica de dificuldade de junção é um pouco mais complexo do que o cálculo da métrica de concorrência. Poderíamos ter optado por uma forma mais simples para chegar ao resultado, levando em consideração apenas o tempo de resolução dos conflitos. Esse tempo poderia ser medido, determinando o intervalo desde a tentativa de *check-in* sem sucesso até o *check-in*. Essa forma de calcular a métrica de dificuldade de junção, no entanto, apresentaria alguns dos problemas discutidos na análise crítica sobre a métrica de concorrência (Seção 3.2.2.2.1). O intervalo de tempo desses dois acontecimentos não necessariamente representaria o tempo real que o desenvolvedor levou para resolver os conflitos. Pode ter havido pausas nesse intervalo que não seriam detectadas. Optamos por fazer o cálculo da forma que foi apresentada na seção anterior, pois assim conseguimos identificar exatamente quais e onde ocorreram os conflitos.

Além disso, a forma escolhida para o cálculo da métrica de dificuldade de junção possibilita a identificação de outros indícios, como, por exemplo: ações que não representaram dificuldade na resolução de conflitos, ações que representaram esforço adicional e ações que representaram esforço desperdiçado. Grande parte desses indícios foi utilizada no cálculo da métrica. O esforço desperdiçado, por exemplo, não foi utilizado, mas pode ser utilizado para outras conclusões futuras. Este indicador representa quanto esforço poderia ter sido empregado na efetiva evolução dos ICs, mas, ao contrário disso, foi totalmente desperdiçado na resolução de conflitos. Um IC que

apresente muito esforço desperdiçado, talvez deva ser analisado mais de perto e isso pode ser um indício de que a política de controle de concorrência deve ser modificada.

3.2.3 Elementos críticos

Uma vez calculadas as duas métricas, o valor que representa o nível crítico do item, que também varia entre 0 e 1, é calculado por meio da multiplicação do nível de concorrência com o nível de dificuldade de junção do IC em um determinado período de tempo (Fórmula 7). O nível crítico representa o risco de perda de produtividade, discutido na Seção 3.2.2.1.

$$\boldsymbol{nível\ crítico(IC) = concorrência(IC) \times dificuldade\ de\ junção(IC)} \quad \textbf{(Fórmula 7)}$$

Como definido na Seção 3.1, os elementos críticos representam os ICs do sistema que merecem uma maior atenção em relação ao controle de concorrência. Dessa forma, os ICs com alto nível crítico, ou seja, com valores próximos de 1, são considerados elementos críticos. O valor exato que define o limite a partir do qual um elemento é considerado crítico pode ser ajustado a cada situação, permitindo que o engenheiro de software adapte a abordagem Orion para cada projeto de desenvolvimento. Esse ajuste é feito na etapa inicial de configuração, discutida no início do capítulo.

A abordagem proposta sugere que os elementos críticos devam sofrer algum tipo de reestruturação. Certamente, esses elementos são responsáveis por diversas funcionalidades, devido ao alto nível de concorrência e alto nível de dificuldade de junção. Possivelmente, com a divisão de um elemento crítico em mais de um elemento, os valores referentes às métricas dos elementos menores serão mais baixos e estes deixarão de ser considerados elementos críticos. Isto diminui o risco de perda de produtividade da equipe de desenvolvimento.

3.2.4 Seleção das políticas de controle de concorrência

Além da identificação dos elementos críticos do sistema, é sugerida a política de controle de concorrência mais indicada para cada IC em questão, baseada nas duas métricas propostas neste capítulo. Pontos de corte definem se o valor obtido para cada métrica é alto ou baixo e podem também ser definidos pelo engenheiro de software, na etapa inicial de configuração, visando um melhor ajuste da abordagem.

A partir da análise dos valores das métricas de concorrência e dificuldade de junção, quatro possibilidades podem ser destacadas para cada IC (Tabela 3.2):

- *Baixa concorrência e baixa dificuldade de junção*. Neste caso, qualquer uma das duas políticas pode ser utilizada para o IC. A política pessimista vai obrigar os engenheiros de software a fazer o bloqueio do IC, o que não traz grandes problemas, visto que o nível de concorrência é baixo. Por outro lado, a estratégia otimista também pode ser adotada, uma vez que, caso haja conflitos, a dificuldade de resolvê-los normalmente é baixa. No entanto, como uma delas precisa ser selecionada, a política otimista deveria ser escolhida, já que ela é a política padrão dos sistemas de controle de versão que oferecem suporte a ambas as estratégias;
- *Baixa concorrência e alta dificuldade de junção*. Para ICs nesta situação, a abordagem sugere a adoção da política pessimista, pois um bloqueio no item não trará muitos problemas, já que o IC possui historicamente um baixo nível de concorrência, no entanto evitará conflitos que demandam um maior esforço do desenvolvedor;
- *Alta concorrência e baixa dificuldade de junção*. Neste caso, a probabilidade de ocorrer um conflito é alta, no entanto não é difícil resolvê-lo, logo a política otimista é indicada;
- *Alta concorrência e alta dificuldade de junção*. Classe que engloba os elementos críticos. Neste caso, tanto a adoção da estratégia pessimista como a adoção da estratégia otimista pode trazer problemas. Dessa forma, é sugerido que estes elementos sofram algum tipo de reestruturação, como discutido na Seção 3.2.3.

Tabela 3.2. Indicação das políticas de concorrência.

		Concorrência	
		Baixo	Alto
Dificuldade de Junção	Baixo	Indiferente	Política otimista
	Alto	Política pessimista	Reestruturação

Vale ressaltar que a abordagem não age de forma intrusiva no trabalho dos engenheiros de software. São feitas sugestões a respeito das políticas de controle de concorrência de cada IC do sistema e cabe ao desenvolvedor ou gerente de configuração selecionar qual a estratégia que deve ser adotada junto ao sistema de controle de versão utilizado.

3.2.5 Visualização

A abordagem Orion prevê a utilização de técnicas de visualização de software na exibição dos resultados obtidos para o usuário, que incluem técnicas de desenho, coloração, animação e agrupamento de informações. A visualização de software tem como objetivo facilitar o entendimento das informações relacionadas ao desenvolvimento de software (CEMIN, 2001) e ainda atuar como uma forma de comunicação entre as pessoas que estão explorando ou manipulando a mesma informação (MACKINLAY e SHNEIDERMAN, 2000).

Com a visualização, é possível explorar o potencial da cognição humana e habilidades de percepção que não são possíveis apenas com representações textuais (LINTERN *et al.*, 2003). Representações visuais tendem a proporcionar uma forma mais simples e intuitiva de entender melhor e mais rapidamente o significado dos dados representados. Essa necessidade se torna ainda mais importante à medida que o volume de informação aumenta.

Neste trabalho, são propostos a representação e o agrupamento dos ICs analisados, assim como a coloração desses ICs de acordo com algum indicador fornecido pela abordagem, que podem ser: o nível crítico, a métrica de concorrência ou a métrica de dificuldade de junção. Como qualquer um desses indicadores varia de 0 a 1, é utilizada a cor verde, quando o valor é 0, e vermelho, quando o valor é 1. Os ICs com valores entre 0 e 1 são coloridos com as variações de cores entre verde e vermelho. Além disso, são utilizados ícones para a rápida identificação das políticas de controle de concorrência selecionadas para cada IC.

Dessa forma, é possibilitada aos usuários uma forma de visualização de alto nível do projeto, permitindo a rápida visualização dos resultados fornecidos pela abordagem Orion. Quando o usuário analisa o nível crítico dos ICs, por exemplo, rapidamente conseguirá identificar os elementos críticos, que estarão representados com uma coloração mais avermelhada.

Outro ponto importante é a possibilidade de ter acesso aos indicadores dos ICs nas versões anteriores do projeto. Assim, o usuário tem a possibilidade de visualizar a evolução desses indicadores ao longo da vida do software, podendo analisar, por exemplo, como o nível de concorrência dos ICs mudou ao longo do projeto.

No Capítulo 5, é apresentado como essa visualização foi implementada neste trabalho.

3.2.6 Aplicação da abordagem

Uma vez utilizada a abordagem, os resultados gerados podem apoiar quatro diferentes papéis no processo de desenvolvimento de software[9] em diferentes propósitos:

- *Desenvolvedor*, que pode fazer alterações na política de controle de concorrência dos ICs a serem modificados. Por exemplo, com o objetivo de evitar esforço adicional para resolver conflitos, que historicamente são difíceis, o desenvolvedor pode optar por fazer um bloqueio em uma determinada classe antes de fazer alterações, baseado na sugestão de políticas da abordagem;
- *Gerente de configuração*, que pode instituir aos desenvolvedores a utilização das políticas sugeridas pela abordagem por meio de mecanismos oferecidos pelos sistemas de controle de versão. Por exemplo, o gerente de configuração pode exigir que qualquer desenvolvedor que for fazer modificações em um determinado IC com alta dificuldade de junção e baixo nível de concorrência, deva adotar a estratégia pessimista;
- *Gerente de projeto*, que pode alocar melhor seus recursos, tentando evitar que muitos desenvolvedores trabalhem em um elemento crítico ou elemento com um alto nível de dificuldade de junção ao mesmo tempo, procurando minimizar o impacto de possíveis conflitos;
- *Arquiteto de software,* que pode decidir fazer reestruturações nos elementos críticos do sistema para diminuir os níveis de concorrência e de dificuldade de junção.

Como foi discutido anteriormente, a abordagem proposta necessita de dados históricos do projeto de desenvolvimento, assim, ela só pode ser aplicada após o início do mesmo em momentos escolhidos pelo engenheiro de software. Ou ainda, para evitar a ação direta de algum usuário, dependendo da integração com o sistema de controle de versão, também é possível programar a abordagem em um modo automático, onde sempre que um novo *check-in* é feito no repositório do sistema de controle de versão, as métricas da abordagem são recalculadas para os ICs afetados e os resultados são atualizados, assim como a visualização desses resultados. Desta forma, os valores das

[9] Esses papéis representam especializações do papel engenheiro de software.

métricas, as sugestões de políticas e as indicações de elementos críticos vão sendo modificados à medida que o projeto vai evoluindo.

3.3 Exemplo

Nesta seção, é apresentado um exemplo do cálculo do nível crítico e da indicação da política de controle de concorrência para um determinado IC. Na aplicação real da abordagem, esse cálculo é feito para todos os ICs do sistema. Este exemplo é propositalmente simples, com o objetivo de facilitar o entendimento da abordagem Orion.

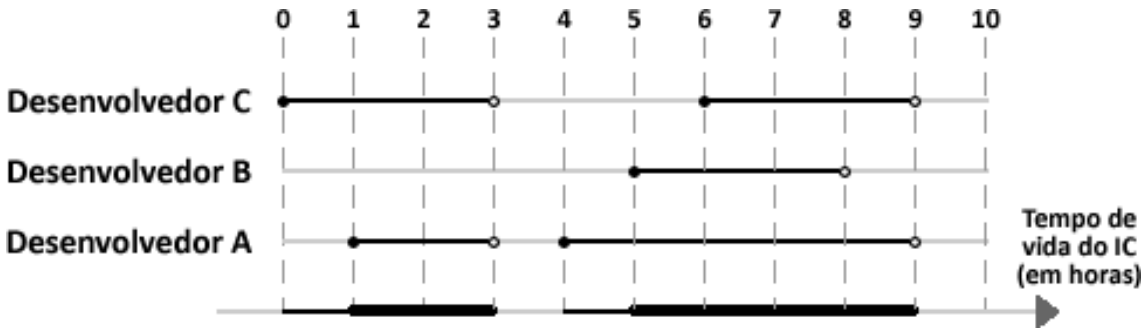


Figura 3.10. Exemplo do cálculo da métrica de concorrência.

A Figura 3.10 ilustra o histórico de uma classe Java que será utilizada como exemplo para o cálculo das métricas. Nela são exibidos os períodos de tempo que o IC ficou em posse dos desenvolvedores para modificação. É possível perceber que a classe tem um histórico de 10 horas (unidade de tempo utilizado neste exemplo), e que nesse período três desenvolvedores (A, B e C) fizeram modificações nessa classe (operações de *check-in* e *check-out*). Em seguida, devemos calcular o tempo que a classe ficou em posse dos três desenvolvedores, tempo de posse (TP) e o tempo em que a classe ficou em posse de dois ou mais desenvolvedores, tempo de concorrência (TC). Segundo o exemplo, o período de posse da classe foi da hora 0 a 3 e da hora 4 a 9, totalizando 8 horas. O tempo de concorrência inclui o período de 1 a 3 horas e de 5 a 9 horas, somando o total de 6 horas. Após essa análise, é possível calcular a métrica de concorrência dessa classe no período de 0 a 10 horas utilizando a Fórmula 2:

$$concorrência(IC) = \frac{TC(IC)}{TP(IC)} = \frac{6}{8} = 0{,}75 = 75\%$$

Portanto, a classe analisada apresentou um nível de 75% de concorrência neste período.

Para calcular a segunda métrica, devem ser analisadas as cinco operações de *check-in* ocorridas durante o período do exemplo (ver Figura 3.10). Em seguida, é necessário calcular a dificuldade de junção em cada um dos cinco *check-ins* e fazer a média aritmética desses valores. Para simplificar, neste exemplo será calculado o valor referente apenas a um *check-in* e serão apresentados os outros quatro valores, que são calculados de forma análoga.

Para calcular a métrica para uma determinada operação de *check-in* deve-se, inicialmente, destacar as quatro versões do IC analisado discutidas na Seção 3.2.2.3. A Figura 3.11 apresenta essas quatro versões da classe Java utilizada no exemplo. Os números circulados representam as unidades de comparação dessa classe, neste exemplo, representam métodos. Círculos com o mesmo número, mas com cores diferentes representam diferentes versões de uma mesma unidade de comparação. Círculos cortados representam uma ação de exclusão, círculos com o sinal de + representam uma ação de adição e círculos cujos números vêm acompanhados de uma aspa simples (´) representam uma ação de modificação da unidade de comparação.

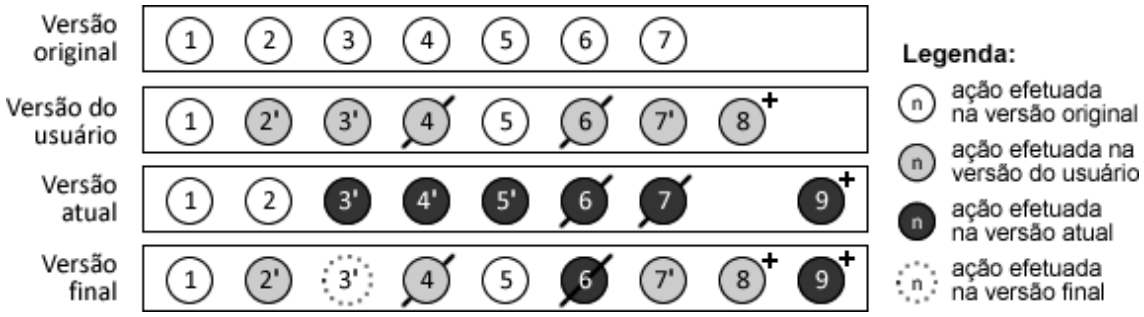


Figura 3.11. Versões necessárias para o cálculo da métrica de dificuldade de junção.

Na versão original do IC existiam sete métodos (1 a 7), representados pelos círculos na cor branca. Na versão do usuário foram removidos os métodos 4 e 6, adicionado o método 8 e alterados os métodos 2, 3 e 7. Todas as unidades de comparação que sofreram algum tipo de ação na versão do usuário foram coloridas com cinza. Na versão atual, foram removidos os métodos 6 e 7, adicionado o método 9 e modificados os métodos 3, 4 e 5. Todas as unidades de comparação que sofreram algum tipo de ação na versão do usuário foram coloridas com preto. Após o *check-in*, o IC (versão final) se apresentava da seguinte forma: o método 2 permaneceu com a versão do usuário; o 3, devido a um possível conflito, teve uma nova versão gerada (círculo pontilhado), baseada nas versões do usuário e atual; o método 4 foi removido, como feito na versão do usuário; o método 5 teve as modificações dos outros desenvolvedores

descartadas e permaneceu com a versão original; o método 6 foi removido; o 7, apesar de ter sido excluído na versão atual, foi mantido com as alterações do usuário; e, finalmente, os métodos adicionados 8 e 9 permaneceram.

Após a identificação das quatro versões necessárias, são aplicadas as Fórmulas 3 e 4 e chega-se aos conjuntos *ações aplicadas* e *ações efetivadas* (Figura 3.12).

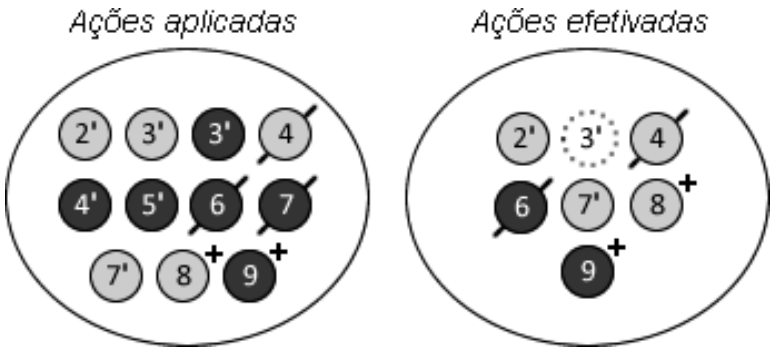


Figura 3.12. Conjuntos *ações aplicadas* e *ações efetivadas.*

No próximo passo, é possível identificar os subconjuntos de ações que representaram esforço desperdiçado, esforço adicional e que não representaram nenhuma dificuldade para resolução dos conflitos. Para isso, faz-se a intersecção entre os conjuntos *ações aplicadas* (A) e *ações efetivadas* (AE) (Figura 3.13).

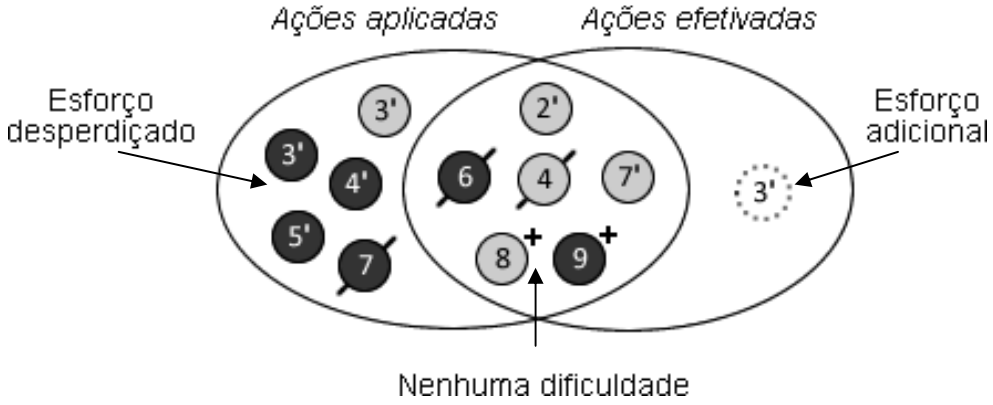


Figura 3.13. Intersecção dos conjuntos ações aplicadas e ações efetivadas.

Finalmente, a métrica pode ser calculada utilizando a Fórmula 5, contando o número de elementos do subconjunto *esforço adicional* (AE – A) e do conjunto *ações efetivadas*:

$$dificuldade\ de\ junção(check_in, IC) = \frac{|(AE - A)|}{|AE|} = \frac{1}{7} \cong 0{,}143 = 14{,}3\%$$

Como foi discutido anteriormente, esse valor representa o resultado para um *check-in*. Como ocorreram cinco no período analisado, os outros quatro valores estão apresentados na Tabela 3.3. É importante lembrar que os *check-ins* que não

apresentaram conflitos terão valor igual a 0, assim como o primeiro *check-in* de cada IC, que nunca apresentará conflitos.

Tabela 3.3. Resultados da métrica de dificuldade de junção do exemplo.

Check-in	Dificuldade de junção
1	0
2	14,3
3	34,9
4	0
5	10,8

Após realizado o cálculo da métrica para todos os *check-ins*, é calculada a média aritmética desses valores utilizando a Fórmula 6:

$$\boldsymbol{dificuldade\ de\ junção(IC) = \frac{0 + 14,3 + 34,9 + 0 + 10,8}{5} = 12\%}$$

Analisando os resultados das métricas de concorrência e dificuldade de junção e com base na Tabela 3.2, a política de controle de concorrência indicada para a classe Java do exemplo seria a política otimista (Figura 3.14).

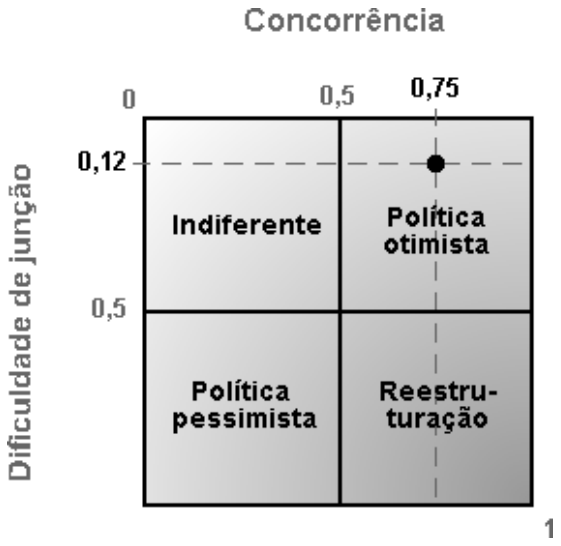


Figura 3.14. Determinação da política selecionada.

Além disso, podemos quantificar o nível crítico do IC durante o período de tempo analisado por meio da Fórmula 7:

$$nível\ crítico(IC) = concorrência(IC) \times dificuldade\ de\ junção(IC) =$$
$$= 0,75 \times 0,12 = 9\%$$

Com o valor de 9% para o nível crítico, o IC não é considerado um elemento crítico e não demanda reestruturação.

Neste exemplo, os valores das métricas acima de 0,5 foram considerados altos. Este ponto de corte é necessário para sugestão da política de controle de concorrência, utilizado na Figura 3.14.

3.4 Considerações finais

Atualmente, a escolha das políticas de controle de concorrência é feita com base no conhecimento dos desenvolvedores e do gerente de configuração sobre o histórico de projeto. Neste capítulo, foi apresentada a abordagem Orion, que busca dar suporte a diferentes papéis envolvidos no processo de desenvolvimento de software relacionados a tomadas de decisão referentes ao controle de concorrência, seja por meio da sugestão da política de controle de concorrência dos ICs, seja por meio da identificação dos elementos críticos. Além disso, para atingir o objetivo proposto pela abordagem, foram identificadas as informações históricas que são úteis para atingir esse objetivo e foram definidas duas métricas: concorrência e dificuldade de junção.

As informações históricas adicionais identificadas neste trabalho de pesquisa podem ainda ser utilizadas para outros propósitos, como, por exemplo, permitir a visualização do modo que cada desenvolvedor trabalha com o controle de versão, incluindo a freqüência de *check-ins* e atualizações dos ICs. Essas informações podem ajudar alguns desenvolvedores a diminuir problemas relacionados ao controle de concorrência.

Outro ponto importante é que a abordagem Orion pode ainda motivar a troca do sistema de controle de versão utilizado pela equipe de desenvolvimento. Caso esse sistema não forneça suporte às estratégias otimista e pessimista, ou ainda, não permita a seleção de diferentes estratégias para diferentes ICs, a equipe pode decidir buscar outras soluções que permitam a aplicação dos resultados propostos pela abordagem.

Algumas limitações das métricas propostas foram identificadas e discutidas nas seções de análise crítica das métricas. Outra limitação que pode ser apontada é o fato de que os dados utilizados pelas fórmulas que calculam as duas métricas podem não ser fornecidos, dependendo do sistema de controle de versão utilizado. No entanto, existem técnicas auxiliares, como a utilização de ramos ou *hooks*, que possibilitam a obtenção desses dados e a aplicação da abordagem.

Capítulo 4 – Avaliação

4.1 Introdução

A abordagem Orion, descrita no Capítulo 3, foi alvo de dois estudos preliminares de avaliação. Os resultados desses estudos, apesar de serem restritos, puderam ser utilizados na identificação de limitações e melhorias da abordagem proposta, determinando alguns passos necessários para a aplicação da abordagem Orion no contexto do desenvolvimento de um projeto de software real.

No total, participaram desses estudos seis alunos de pós-graduação da COPPE/UFRJ. O primeiro consiste em um estudo de viabilidade da abordagem Orion, do qual participaram três alunos. O segundo representa um estudo de observação onde os outros três alunos tiveram contato com a abordagem Orion e analisaram sua aplicação.

Este capítulo está organizado da seguinte forma: inicialmente, na Seção4.2, são apresentados os objetivos gerais dos dois estudos. Na Seção 4.3, ambos os estudos são definidos. Em seguida, são apresentados os dois estudos separadamente. Na Seção 4.4, é apresentado o estudo de viabilidade e, na Seção 4.5, o estudo de observação. Na Seção 4.6, é discutida a validade dos estudos. Finalmente, a Seção 4.7 conclui o capítulo com algumas considerações finais.

4.2 Objetivos

Nesta seção são apresentados os objetivos gerais do estudo de viabilidade e do estudo de observação.

4.2.1 Estudo de viabilidade

Com o intuito de caracterizar a viabilidade do processo de seleção de políticas de controle de concorrência no suporte à coordenação de equipes de desenvolvimento de software, foi realizado um estudo de viabilidade. Este tipo de estudo visa coletar dados para avaliar se a metodologia proposta é viável (SHULL *et al.*, 2001).

O objetivo deste estudo pode ser apresentado de acordo com o modelo descrito por WOHLIN *et al.* (2000), apresentado na Tabela 4.1.

Tabela 4.1. Definição do objetivo do estudo de viabilidade.

Analisar o processo de seleção de políticas de controle de concorrência da abordagem Orion
Com o propósito de caracterizar a viabilidade
Em relação ao controle de concorrência no desenvolvimento de software
Do ponto de vista do gerente de configuração de software
No contexto de equipes de desenvolvimento de software

4.2.2 Estudo de observação

Visando analisar a aplicação do mecanismo de seleção de políticas de controle de concorrência proposto pela abordagem Orion no suporte à coordenação de equipes de desenvolvimento de software, foi realizado um estudo de observação. Em estudos deste tipo, o participante realiza alguma tarefa enquanto é observado por um experimentador. Este tipo de estudo tem a finalidade de coletar dados sobre como determinada tarefa é realizada (SHULL *et al.*, 2001). Por meio dessas informações, pode-se obter uma compreensão de como um novo processo é utilizado.

Assim como no estudo de viabilidade, o objetivo deste estudo de observação pode ser apresentado de acordo com o modelo descrito por WOHLIN *et al.*, (2000), apresentado na Tabela 4.2.

Tabela 4.2. Definição do objetivo do estudo de observação.

Analisar o mecanismo de seleção de políticas de controle de concorrência da abordagem Orion
Com o propósito de caracterizar a aplicação
Em relação ao controle de concorrência no desenvolvimento de software
Do ponto de vista do gerente de configuração de software
No contexto de equipes de desenvolvimento de software

4.3 Definição dos estudos

Estes estudos foram realizados em um ambiente acadêmico, na COPPE/UFRJ. Para participar dos estudos, os voluntários deveriam basicamente possuir experiência em algum sistema de controle de versão. Seis alunos de pós-graduação da linha de pesquisa em Engenharia de Software da COPPE/UFRJ foram voluntários na participação desses estudos. Não houve nenhum tipo de compensação para os participantes.

Na situação proposta aos participantes de ambos os estudos, eles deveriam exercer o papel de gerente de configuração de software no contexto de um projeto de desenvolvimento. Eles tinham como atividade principal fazer a seleção das políticas de controle de concorrência para cada elemento do sistema de software utilizado no estudo, com o intuito de otimizar a produtividade da equipe de desenvolvimento. Para facilitar a imersão do participante no contexto do estudo, foi informado que ele havia sido contratado como gerente de configuração por uma empresa de desenvolvimento de software que decidiu impor um processo rígido de controle de versões em todos os seus projetos.

O projeto de desenvolvimento que o participante atuou no estudo foi um projeto fictício, chamado *HotelSystem*, que desenvolve um sistema para gerenciamento de hotéis e que se encontra em etapa de elaboração. Uma descrição mais detalhada do projeto pode ser encontrada no Apêndice I e o diagrama de classes do projeto está representado no Apêndice II.

A equipe do projeto é formada por três integrantes que atuam como desenvolvedores e utilizam um sistema de controle de versão baseados em modelos UML.

Como fontes de informação, estavam disponíveis aos participantes, além da descrição do projeto e o diagrama de classes da versão atual do projeto, o histórico de modificações de cada IC abordado no estudo provido pelo sistema de controle de versão utilizado pela equipe de desenvolvimento. Nesses históricos, eram apresentadas informações como o identificador, o autor, a data e a hora de cada *check-in*. Um exemplo desse histórico é mostrado na Figura 4.1. As informações contidas nesse histórico são basicamente as mesmas encontradas na maioria dos sistemas de controle de versão, tais como, CVS e Subversion. O histórico completo utilizado no estudo pode ser encontrado no Apêndice III.

Check-in	Autor	Data
25	Bruno	21/07/2008 - 15:36
33	Pedro	22/07/2008 - 17:49
37	Marta	23/07/2008 - 15:11
44	Bruno	23/07/2008 - 17:25

Figura 4.1. Histórico de um IC provido pelo sistema de controle de versão.

Além das informações citadas no parágrafo anterior, estava disponível também para o participante, dependendo da etapa do estudo, uma imagem que representa

graficamente algumas informações históricas adicionais, como, por exemplo, momento de *check-out* ou atualização do IC; momentos em que houve concorrência; tentativas de *check-ins* sem sucesso, devido a conflitos; e tempo necessário aos desenvolvedores para resolução dos conflitos (Figura 4.2). Estas informações históricas adicionais foram identificadas neste trabalho de pesquisa e são utilizadas na abordagem Orion. Elas estão disponibilizadas no Apêndice IV.

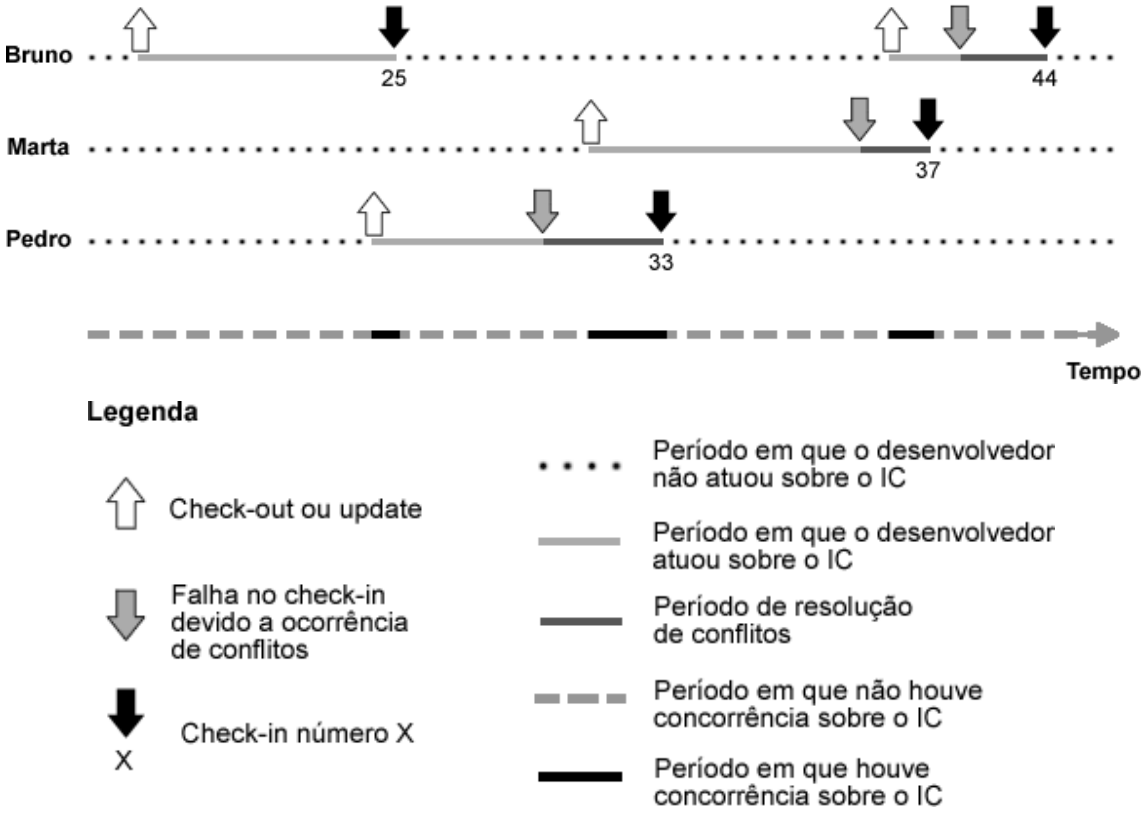


Figura 4.2. Gráfico representando informações históricas adicionais de um determinado IC.

Por meio do número de identificação dos *check-ins*, o participante conseguia fazer uma correspondência entre o gráfico com as informações adicionais (Figura 4.2) e o histórico provido pelo sistema de controle de versão (Figura 4.1).

A tarefa do participante consistia em analisar as informações disponibilizadas, referentes ao estado atual e passado do projeto, que podiam mudar dependendo da etapa realizada, e fazer a seleção de políticas de controle de concorrência para cada IC analisado no estudo. Essa seleção foi feita em um formulário impresso entregue ao participante, que também continha algumas questões a serem respondidas. Esses formulários são apresentados nas seções seguintes.

4.4 Estudo de viabilidade

Nesta seção, o estudo de viabilidade da abordagem Orion é descrito detalhadamente (Seção 4.4.1) e são apresentados: os procedimentos adotados (Seção 4.4.2) e os resultados atingidos (Seção 4.4.3). Por fim, são discutidas as avaliações dos participantes (Seção 4.4.4).

4.4.1 Descrição

Este primeiro estudo foi dividido em duas etapas, a primeira delas tem como finalidade entender como a atividade de seleção de políticas de controle de concorrência é executada pelo gerente de configuração de software, tendo como insumos as informações do projeto de desenvolvimento e as informações históricas armazenadas pela maioria dos sistemas de controle de versão existentes.

Na segunda etapa, busca-se analisar a execução da mesma tarefa de seleção de políticas de controle de concorrência, desta vez provendo ao gerente de configuração de software algumas informações históricas adicionais, identificadas neste trabalho de pesquisa (representadas anteriormente na Figura 4.2).

Os principais pontos a serem observados neste estudo são:

1. Quais informações os participantes utilizam na atividade de seleção de políticas de controle de concorrência;
2. Se as informações históricas adicionais são utilizadas na atividade de seleção de políticas de controle de concorrência;
3. Se a disponibilização das informações históricas adicionais impactou na atividade de seleção de políticas de controle de concorrência; e
4. Se a disponibilização das informações históricas adicionais possibilitou uma seleção mais confiante das políticas de controle de concorrência.

4.4.2 Procedimento

Ao todo, três alunos de pós-graduação participaram deste estudo. Foram executadas três sessões e em cada uma delas foi utilizado apenas um único participante. Inicialmente, o participante foi informado sobre o estudo por meio do Formulário de Consentimento (Apêndice V). Caso concordasse em participar, ele preenchia o Questionário de Caracterização (Apêndice VI), que avalia o nível de conhecimento e experiência do participante em diferentes temas relacionados ao estudo, como, por

exemplo, desenvolvimento de software, modelagem de sistemas de informação e sistemas de controle de versão. Estas informações foram utilizadas para garantir que os participantes estavam aptos a executar o estudo de avaliação e, além disso, as informações ajudaram na interpretação dos resultados obtidos por cada um dos participantes.

Após o preenchimento do Questionário de Caracterização, foi feita ao participante uma apresentação de aproximadamente 10 minutos sobre Gerência de Configuração de Software (GCS), que englobou uma discussão sobre sistemas de controle de versão e políticas de controle de concorrência, além da definição de alguns conceitos que seriam utilizados posteriormente nos documentos fornecidos. Em seguida, foi entregue ao participante o documento de Descrição Geral da Tarefa (Apêndice I), que descreve o contexto em que ele estará inserido e a especificação de sua tarefa no estudo de avaliação. Após conhecer a tarefa a ser executada, o participante recebia o documento de Descrição do Projeto (Apêndice II), que possui o diagrama de classes da versão atual do projeto adotado no estudo e identifica os 6 ICs que seriam alvos de sua tarefa. Vale lembrar que como o projeto estava em sua etapa inicial, ainda não havia sido iniciada a fase de implementação.

Em adição ao documento de descrição, o participante recebia também o documento de Histórico do Projeto (Apêndice III), como exemplificado na Seção 4.3, que contém informações a respeito do histórico de cada um dos 6 ICs adotados no estudo, armazenados no sistema de controle de versão.

Dessa forma, em posse dos documentos: Descrição do Projeto e Histórico do Projeto, o participante deveria executar sua tarefa de seleção de políticas de controle de concorrência, preenchendo o Formulário de Seleção de Políticas – 1 (Apêndice VII). Neste formulário, o participante preenchia uma tabela, onde selecionava para cada um dos 6 ICs: política otimista; política pessimista; política indiferente (significando que tanto a política otimista ou pessimista poderia ser aplicada) e nenhuma das duas políticas (caso o participante entendesse que nem a política otimista nem a pessimista era aplicável por alguma razão, que deveria ser explicitada). Nessa mesma tabela, o participante deveria ainda assinalar a sua confiança na seleção feita. Além disso, eram propostas aos participantes duas questões relacionadas com sua tarefa.

Essa é considerada a primeira etapa do estudo. Ela engloba a análise das informações oferecidas ao participante para o preenchimento do Formulário de Seleção de Políticas – 1, além do preenchimento do mesmo.

Depois de terminada essa primeira etapa, era disponibilizado ao participante o documento de Informações Históricas Adicionais (Apêndice IV), que inclui 6 figuras que representam graficamente as informações históricas adicionais exemplificadas na Seção 4.3. Após a explicação dos gráficos, era solicitado ao participante o preenchimento do Formulário de Seleção de Políticas – 2 (Apêndice VIII), bastante semelhante ao Formulário de Seleção de Políticas – 1, com diferença apenas nas questões abertas. O participante deveria, então, executar a mesma tarefa efetuada na primeira etapa, só que agora levando em consideração as novas informações históricas dos ICs do projeto. Em uma das questões do formulário, o participante deveria fazer uma comparação com a sua seleção de políticas feita na primeira etapa. Por isso, o Formulário de Seleção de Políticas – 1 era disponibilizado neste momento apenas para critério de consulta. Essa é considerada a segunda etapa. Ela inclui a análise das informações oferecidas ao participante para o preenchimento do Formulário de Seleção de Políticas – 2, além do seu preenchimento.

Concluída essa segunda etapa, o participante preenchia o Questionário de Avaliação (Apêndice IX), por meio do qual deveria expressar sua opinião a respeito: das tarefas executadas no estudo, das informações históricas adicionais representadas nos gráficos e do controle de concorrência no processo de desenvolvimento de software.

Em média, o tempo total deste primeiro estudo foi de 1 hora.

4.4.3 Resultados

Participaram deste estudo, três alunos de mestrado com experiência em desenvolvimento de software e sistemas de controle de versão. A Tabela 4.3 apresenta um resumo da caracterização desses participantes. Os níveis de experiência variam de 0 a 4, onde 0 representa o nível mais baixo e 4 representa o nível mais alto de experiência.

A Tabela 4.4 mostra o tempo gasto pelos três participantes na execução de suas tarefas de seleção de políticas. Esse tempo leva em consideração apenas o tempo de análise das informações oferecidas ao participante e o preenchimento dos Formulários de Seleção de Políticas fornecido na etapa. Esses números não incluem o tempo gasto com as fases iniciais do estudo como apresentação de GCS e preenchimento do Formulário de Consentimento e do Questionário de Caracterização do Participante. Além disso, não foi incluído o tempo com a fase final, de preenchimento do Questionário de Avaliação.

Tabela 4.3. Resumo do Questionário de Caracterização dos participantes do estudo de viabilidade.

Critério				P1	P2	P3
1			Formação acadêmica	Mestrando	Mestrando	Mestrando
2	**2.1**		Experiência em desenvolvimento	Curso	Indústria	Curso
	2.2		Tempo de experiência (anos)	4	6	4
	2.3		Tamanho máximo da equipe (pessoas)	5	6	5
	2.4	**2.4.1**	Experiência em UML (0-4)	2	4	4
		2.4.2	Experiência em modelagem (0-4)	4	4	2
		2.4.3	Experiência em sistemas de controle de versão (0-4)	4	4	4
	2.5		Sistemas de controle de versão que já utilizou	CVS, Subversion, VSS	CVS, Subversion e ClearCase	CVS, Subversion
3			Experiência no domínio de hotéis (0-4)	0	2	2

A seguir são detalhados os resultados obtidos nas três sessões e, por fim, é feita uma análise geral dos resultados.

Tabela 4.4. Resumo dos tempos da tarefa dos participantes.

Participante	Duração da tarefa (em min.) Etapa 1	Duração da tarefa (em min.) Etapa 2
P1	15	15
P2	20	12
P3	25	22

4.4.3.1 Participante P1

Na primeira etapa do estudo, o participante selecionou a política otimista para os 6 ICs e afirmou que o critério utilizado para essa escolha foi a quantidade de pessoas que fizeram modificações nos ICs. Já na segunda etapa, o participante fez modificações em 3 ICs, trocando sua seleção de otimista para indiferente (Tabela 4.5).

Analisando os resultados, fica claro que o participante P1 fez uso das informações históricas adicionais, porém não as utilizou para analisar a dificuldade e o tempo de resolução dos conflitos. Por essa razão, em nenhum momento o participante selecionou a política pessimista. As três modificações ocorreram nos ICs com um maior período de concorrência. Das três, a modificação da política do IC 6 é a única

considerada negativa, já que o IC 6 não apresenta histórico de conflitos, o que caracterizaria a adoção de uma estratégia otimista. Nos outros dois casos, existem momentos de concorrência e conflitos, o participante decidiu modificar a política para indiferente, não engessando nenhuma política.

Ele afirmou que considera que as informações históricas adicionais providas foram úteis para a realização da sua tarefa, pois, dessa forma, foi possível identificar o real período de concorrência sobre um determinado IC, além de observar a quantidade de desenvolvedores que concorrem sobre os ICs. Segundo o participante, na segunda etapa ele fez uma seleção mais confiante das políticas de controle de concorrência.

Tabela 4.5. Resumo das tarefas executadas pelo participante P1.

IC	Seleção Etapa 1	Seleção Etapa 2
1	Otimista	Otimista
2	Otimista	Otimista
3	Otimista	Indiferente
4	Otimista	Otimista
5	Otimista	Indiferente
6	Otimista	Indiferente

4.4.3.2 Participante P2

Na etapa inicial do estudo, o participante P2 selecionou a política otimista para os ICs 1, 2 e 3 e política pessimista para os ICs 4, 5 e 6. Nesta etapa, o participante afirmou que levou em consideração dois critérios: quantidade de desenvolvedores e grau de acoplamento entre as classes. Na etapa 2, o participante fez apenas uma modificação na sua seleção inicial, trocou a política pessimista do IC 6 para otimista (Tabela 4.6). De acordo com o participante, os novos critérios levados em consideração foram: período de concorrência e dificuldade de junção.

O IC 6 é um exemplo de que uma quantidade relativamente alta de desenvolvedores e um intervalo curto entre *check-ins* não significa que deve ser adotada a política pessimista. Na segunda etapa, o participante pôde perceber que não houve nenhum conflito no IC 6 e modificou a sua política para otimista. Portanto, essa modificação foi considerada positiva.

De acordo com o participante, as informações históricas adicionais foram úteis para a realização de sua tarefa, pois "quanto maior o grau de dificuldade na junção, mais

tempo será gasto resolvendo os conflitos". Os resultados também mostraram que, na segunda etapa do estudo, o participante realizou sua tarefa com mais confiança.

Tabela 4.6. Resumo das tarefas executadas pelo participante P2.

IC	Seleção Etapa 1	Seleção Etapa 2
1	Otimista	Otimista
2	Otimista	Otimista
3	Otimista	Otimista
4	Pessimista	Pessimista
5	Pessimista	Pessimista
6	Pessimista	Otimista

4.4.3.3 Participante P3

Na etapa 1, o participante P3 assinalou: política otimista para os ICs 1, 3 e 5; política pessimista para o IC 6; e política indiferente para os ICs 2 e 4. Para chegar a esse resultado, o participante levou em consideração: quantidade de desenvolvedores e intervalo de tempo entre *check-ins*. Na etapa posterior, o participante fez 4 modificações na sua seleção inicial. Para os ICs 1, 4 e 5 foi escolhida a política pessimista e para o IC 6 foi escolhida a política otimista (Tabela 4.7). Os novos critérios considerados para a realização da tarefa foram: período de concorrência e dificuldade de junção.

Tabela 4.7. Resumo das tarefas executadas pelo participante P3.

IC	Seleção Etapa 1	Seleção Etapa 2
1	Otimista	Pessimista
2	Indiferente	Indiferente
3	Otimista	Otimista
4	Indiferente	Pessimista
5	Otimista	Pessimista
6	Pessimista	Otimista

Os ICs 1 e 4 se caracterizam por terem um baixo nível de concorrência e algumas situações de conflitos. Nesses casos, o participante modificou a política para pessimista. O IC 5 se caracteriza por ter alto nível de concorrência e grande quantidade de conflito. Nesse caso, o participante optou por evitar o paralelismo e não perder tempo para resolver conflitos. O IC 6, como discutido na seção anterior, apresenta bastante

concorrência e nenhum conflito, por isso a mudança para a política otimista. Todas essas 4 modificações são consideradas positivas.

Além disso, o participante afirmou que as informações históricas adicionais foram úteis para a seleção de políticas, pois, segundo ele, estas informações "adicionam novos importantes indicadores para seleção de políticas". O participante ainda confirmou executar sua tarefa com maior confiança comparando com a primeira etapa.

4.4.3.4 Análise geral

Em geral, os resultados deste estudo foram positivos. A única ressalva aparece no resultado da escolha de uma política do participante P1.

Analisando os resultados obtidos e os pontos a serem observados levantados na Seção 4.4.1, podemos afirmar que para este estudo de viabilidade realizado:

1. Na etapa inicial, para executar suas tarefas de seleção de políticas de controle de concorrência, os participantes utilizaram as seguintes informações:
    - Quantidade de desenvolvedores que modificaram o IC (citado pelos três participantes);
    - Intervalo de tempo entre os *check-ins* (citado por um participante); e
    - Grau de acoplamento entre os ICs (citado por um participante).
2. As informações históricas adicionais foram utilizadas na atividade de seleção de políticas de controle de concorrência, fornecendo os seguintes tipos de informação adicionais:
    - Período em que houve concorrência (citado pelos três participantes);
    - Dificuldade de resolução de conflitos (citado por dois participantes); e
    - Quantidade de desenvolvedores que concorreram sobre um determinado IC (citado por um participante).
3. A disponibilização das informações históricas adicionais impactou na atividade de seleção de políticas de controle de concorrência em todos os três casos.
4. A disponibilização das informações históricas adicionais possibilitou uma seleção mais confiante das políticas de controle de concorrência em todos os três casos.

4.4.4 Avaliação dos participantes

Após a conclusão de suas tarefas, os participantes de ambos os estudos preencheram um questionário de avaliação. Em geral, a avaliação do estudo de viabilidade foi positiva. Sobre o estudo em si, todos os três participantes afirmaram que ficaram satisfeitos com o resultado final das tarefas e que não sentiram dificuldades na realização das mesmas.

Quando questionados sobre as informações históricas adicionais fornecidas, todos os participantes afirmaram que utilizaram essas informações para a realização de suas tarefas e consideram importante a disponibilização dessas informações adicionais pelos sistemas de controle de versão. Além disso, os participantes afirmaram que enxergam outros benefícios na utilização dessas informações adicionais, além da seleção de políticas de controle de concorrência, como, por exemplo:

- "Indicação ao desenvolvedor do modo que ele deve trabalhar com versionamento. Por exemplo, fazer *check-ins* mais freqüentes";
- "Acompanhamento da ocorrência de conflitos";
- "Indicação do nível de complexidade de um determinado IC"; e
- "Contribuição para futuras decisões de projeto".

Os participantes responderam ainda algumas perguntas sobre o controle de concorrência no desenvolvimento de software. Todos afirmaram que tinham conhecimento das políticas de controle de concorrência e consideram que é importante que os sistemas de controle de versão possibilitem a utilização de diferentes políticas para diferentes ICs. Por fim, foi questionado se eles acreditam que a escolha adequada das políticas para cada IC influencia de forma positiva na produtividade da equipe de desenvolvimento. As respostas foram:

- "Sim. Com a melhor política selecionada, culmina em um menor tempo de desenvolvimento, já que o grau de paralelismo pode aumentar e possíveis *merges* complicados podem ser evitados";
- "Sim, porque desta forma reduz-se o tempo gasto resolvendo conflitos e aumenta a produtividade do desenvolvimento, paralelizando itens que podem ser desenvolvidos desta forma"; e
- "Sim, pois paralelismo no desenvolvimento e esforço no *merge* têm implicações diretas na produtividade de uma equipe de desenvolvimento".

4.5 Estudo de observação

Nesta seção, o estudo de observação da abordagem Orion é detalhado. São apresentados: a descrição do estudo (Seção 4.5.1), os procedimentos de execução (Seção 4.5.2), os resultados alcançados (Seção 4.5.3) e, finalmente, as avaliações dos participantes (Seção 4.5.4).

4.5.1 Descrição

Este segundo estudo também foi dividido em duas etapas, a primeira delas tem como finalidade observar como a atividade de seleção de políticas de controle de concorrência é executada pelo gerente de configuração de software, tendo como insumos as informações do projeto de desenvolvimento, as informações históricas armazenadas pela maioria dos sistemas de controle de versão existentes e as informações históricas adicionais identificadas neste trabalho. Esta etapa se assemelha bastante com a segunda etapa do estudo de viabilidade, pois o participante tem a mesma tarefa e possui disponíveis os mesmos tipos de informações a respeito do projeto de desenvolvimento[10].

Na segunda etapa, é apresentada ao participante a abordagem Orion e como ela trabalha para executar a seleção de políticas de controle de concorrência para cada IC analisado. Em seguida, o participante tem acesso ao conjunto de políticas selecionado pela abordagem Orion para o mesmo projeto que ele utilizou na primeira etapa. O voluntário deve, então, analisar esses resultados e expressar sua concordância e aceitação ou não a respeito das políticas e sugestões propostas pela abordagem.

Os principais pontos a serem observados neste estudo são:

1. Se as informações históricas adicionais são úteis na atividade de seleção de políticas de controle de concorrência;
2. Se o mecanismo de seleção de políticas de controle de concorrência da abordagem Orion está coerente com o pensamento do participante; e
3. Se o participante adotaria as seleções de políticas de controle de concorrência e sugestões feitas pela abordagem Orion.

[10] É importante enfatizar que essas etapas semelhantes pertencem a estudos diferentes, portanto, foram realizadas com grupos de participantes diferentes. Além disso, no estudo de observação, essa etapa é necessária para aproximar o participante da metodologia adotada pela abordagem Orion, que será apresentada a ele em seguida, na segunda etapa.

4.5.2 Procedimento

Ao todo, participaram deste estudo três alunos de pós-graduação. Foram executadas três sessões e em cada uma delas foi utilizado apenas um único participante. Os procedimentos iniciais de execução deste estudo são idênticos ao do estudo de viabilidade.

Inicialmente, o participante preenche o Formulário de Consentimento (Apêndice V) e, caso aceite os termos de participação, ele deve preencher o Questionário de Caracterização (Apêndice VI). Em seguida, é feita ao participante a mesma apresentação sobre Gerência de Configuração de Software feita para os participantes do estudo de viabilidade. Depois disso, são fornecidos os documentos: Descrição Geral da Tarefa (Apêndice I), Descrição do Projeto (Apêndice II) e Histórico do Projeto (Apêndice III).

A partir deste ponto, este estudo se diferencia do estudo de viabilidade. Ao contrário do estudo anterior, na primeira etapa deste estudo já é disponibilizado ao participante o documento de Informações Históricas Adicionais (Apêndice IV).

Em posse desses três documentos, Descrição Geral da Tarefa, Histórico do Projeto e Informações Históricas Adicionais, o participante deve executar sua tarefa preenchendo o Formulário de Seleção de Políticas (Apêndice X), semelhante aos formulários do mesmo tipo preenchidos pelo grupo de participantes do estudo de viabilidade.

A etapa 1 deste estudo incluiu a análise das informações oferecidas ao participante para o preenchimento do Formulário de Seleção de Políticas, além do seu preenchimento.

Finalizada essa primeira etapa, foi feita mais uma apresentação ao participante. Desta vez, foi apresentada a abordagem Orion, suas características e como ela executa o mecanismo de seleção de políticas de controle de concorrência para cada IC do projeto. Em seguida, foi fornecido ao participante o Formulário de Seleção de Políticas – Orion (Apêndice XI), onde são apresentadas as políticas de controle de concorrência e sugestões da abordagem Orion para cada um dos 6 ICs do mesmo projeto que o participante analisou na primeira etapa, além das razões para essa escolha. No mesmo formulário, o participante deveria considerar o conjunto de políticas proposto, comparar com a sua seleção feita na primeira etapa e decidir se ele adotaria as sugestões propostas pela abordagem Orion.

Essa segunda etapa englobou a análise das políticas de controle de concorrência sugeridas pela abordagem Orion no Formulário de Seleção de Políticas – Orion e o seu preenchimento.

Concluída essa etapa, o participante preenchia o Questionário de Avaliação (Apêndice XII), por meio do qual deveria expressar sua opinião a respeito: das tarefas executadas no estudo, da abordagem Orion e do controle de concorrência no processo de desenvolvimento de software.

Em média, o tempo total deste segundo estudo foi em torno de 1 hora e 15 minutos.

Na execução de ambos os estudos, os participantes não tiveram a necessidade de fazer utilização de computadores, todas as informações foram fornecidas oralmente ou por meio de documentos impressos, assim como os formulários e questionários preenchidos pelos participantes.

4.5.3 Resultados

Participaram deste segundo estudo outros três alunos de mestrado, com experiência em desenvolvimento de software e sistemas de controle de versão. A Tabela 4.8 apresenta um resumo da caracterização desses participantes. Os níveis de experiência variam de 0 a 4, onde 0 representa o nível mais baixo e 4 representa o nível mais alto de experiência.

A Tabela 4.9 mostra o tempo gasto pelos três participantes para executar suas tarefas de seleção de políticas da etapa 1. Na segunda etapa, não houve seleção de políticas feita pelos participantes, por isso só é apresentado o tempo da tarefa na etapa 1. Esse tempo leva em consideração o tempo de análise das informações oferecidas ao participante e o preenchimento do Formulário de Seleção de Políticas. Esses números não incluem o tempo gasto com as fases iniciais do estudo como apresentação de GCS e preenchimento do Formulário de Consentimento e do Questionário de Caracterização do Participante. Além disso, não foi incluído o tempo com a fase final, de preenchimento do Questionário de Avaliação.

Tabela 4.8. Resumo do Questionário de Caracterização dos participantes do estudo de observação.

Pergunta				P4	P5	P6
1			Formação acadêmica	Mestrando	Mestrando	Mestrando
2	**2.1**		Experiência em desenvolvimento	Curso	Indústria	Indústria
	2.2		Tempo de experiência (anos)	4	13	4
	2.3		Tamanho máximo da equipe	5	7	6
	2.4	**2.4.1**	Experiência em UML (0-4)	4	4	3
		2.4.2	Experiência em modelagem (0-4)	4	4	2
		2.4.3	Experiência em sistemas de controle de versão (0-4)	3	4	4
	2.5		Sistemas de controle de versão que já utilizou	CVS, Subversion	Subversion	CVS, Subversion
3			Experiência no domínio de hotéis (0-4)	2	2	0

A seguir são detalhados os resultados de cada um dos três participantes e, por fim, é feita uma análise geral dos resultados obtidos.

Tabela 4.9. Resumo dos tempos da tarefa dos participantes.

Participante	Tempo da tarefa (em min.) Etapa 1
P4	40
P5	20
P6	25

4.5.3.1 Participante P4

O participante, na primeira etapa do estudo, escolheu a política pessimista para os ICs 1, 4 e 5; política otimista para os ICs 3 e 6; e política indiferente para o IC 2. Para executar sua tarefa, o participante levou em consideração: a quantidade de *check-ins*, período em que houve concorrência e dificuldade de junção. Além disso, ele afirmou que as informações históricas adicionais são úteis e facilitaram sua tomada de decisão em relação à seleção das políticas de controle de concorrência.

Na etapa seguinte, o participante P4 comparou sua seleção executada na etapa 1 com a seleção feita pela abordagem Orion (Tabela 4.10). Dois casos (ICs 3 e 5) apresentarem divergências, em ambos, a abordagem sugeriu uma reestruturação para os ICs. O participante declarou que faria sim as reestruturações sugeridas pela abordagem

Orion, pois, segundo ele, “haveria provavelmente ganhos significantes de produtividade”.

Tabela 4.10. Resumo das tarefas executadas pelo participante P4.

IC	Seleção Etapa 1	Seleção Orion	Concordância
1	Pessimista	Pessimista	Sim
2	Indiferente	Indiferente	Sim
3	Otimista	Reestruturação	Sim
4	Pessimista	Pessimista	Sim
5	Pessimista	Reestruturação	Sim
6	Otimista	Otimista	Sim

4.5.3.2 Participante P5

Na etapa inicial do estudo, o participante selecionou política otimista para os ICs 1, 2, 3, 6 e política pessimista para os ICs 4 e 5. Para isso, os critérios utilizados foram: quantidade de conflitos e períodos de concorrência. O participante ainda afirmou que as informações históricas adicionais foram úteis na realização da sua tarefa, pois, segundo ele, “permitem uma melhor compreensão da quantidade de conflitos que estão ocorrendo ao longo do tempo e com elas é possível perceber o quanto o trabalho de um desenvolvedor está interferindo no trabalho do outro”.

Tabela 4.11. Resumo das tarefas executadas pelo participante P5.

IC	Seleção Etapa 1	Seleção Orion	Concordância
1	Otimista	Pessimista	Sim
2	Otimista	Indiferente	Sim
3	Otimista	Reestruturação	Sim
4	Pessimista	Pessimista	Sim
5	Pessimista	Reestruturação	Sim
6	Otimista	Otimista	Sim

Na etapa posterior, o participante comparou sua seleção da primeira etapa com a seleção proposta pela abordagem Orion e detectou divergências em 3 ICs (Tabela 4.11). Em dois deles (ICs 3 e 5), a abordagem Orion sugeriu uma reestruturação. Nestes casos, o participante P5 afirmou que “optaria pela reestruturação sugerida, uma vez que o trabalho de um programador realmente está conflitando com o trabalho do outro”. No

terceiro caso que apresentou diferenças (IC 1), o participante declarou que concorda, mas "faria uma nova análise para verificar se os conflitos são difíceis ou não de serem resolvidos". Para o IC 2, a abordagem sugeriu política indiferente e o participante sugeriu política otimista, no entanto, isto não foi considerado uma divergência na análise do participante.

4.5.3.3 Participante P6

Com relação ao participante P6, a seleção feita na etapa 1 foi: política otimista para os ICs 3 e 6; política pessimista para os ICs 1 e 4; e política indiferente para os ICs 2 e 5. Para essa seleção, o participante utilizou os seguintes critérios: período em que houve concorrência, quantidade de conflitos e dificuldade de junção. Além disso, o participante afirmou que as informações históricas adicionais são úteis, pois, segundo ele, "sabe-se exatamente o tempo que se ganhou com paralelismo e o tempo que se perdeu resolvendo conflitos".

Tabela 4.12. Resumo das tarefas executadas pelo participante P6.

IC	Seleção Etapa 1	Seleção Orion	Concordância
1	Pessimista	Pessimista	Sim
2	Indiferente	Indiferente	Sim
3	Otimista	Reestruturação	Não
4	Pessimista	Pessimista	Sim
5	Indiferente	Reestruturação	Sim
6	Otimista	Otimista	Sim

Na etapa seguinte, em comparação com as sugestões da abordagem Orion, o participante identificou duas divergências nos ICs 3 e 5 (Tabela 4.12). Nestes casos, a abordagem Orion sugeriu reestruturação. Em relação ao IC 5, o participante afirmou que atenderia sim a sugestão de reestruturação, no entanto, para o IC 3 ele afirmou que "manteria uma política otimista, visto que o tempo ganho com o paralelismo superou o tempo utilizado na resolução de conflitos".

4.5.3.4 Análise geral

Em geral, os resultados foram positivos, pois na grande maioria dos casos, ou a seleção da abordagem Orion foi igual a do participante ou o participante concordou em

modificar sua seleção diante do resultado exposto da abordagem. Somente em um caso, com o participante P6, a sugestão não foi acatada.

Analisando os resultados obtidos e os pontos a serem observados levantados na Seção 4.5.1, podemos afirmar que para este estudo de observação realizado:

1. As informações históricas adicionais são úteis na atividade de seleção de políticas de controle de concorrência, fornecendo as seguintes informações:
   - Período em que houve concorrência (citado pelos três participantes);
   - Dificuldade de junção (citado pelos dois participantes);
   - Quantidade de conflitos (citado por dois participantes); e
   - Quantidade de *check-ins* (citado por um participante).
2. O mecanismo de seleção de políticas de controle de concorrência da abordagem Orion está coerente com o pensamento do participante. Considerando que "reestruturação" não era, claramente, uma opção fornecida para os participantes, a análise dos resultados do estudo permite afirmar que as escolhas de políticas feitas pelo participante estão muito próximas da seleção proposta pela abordagem Orion. Dos 18 casos analisados neste estudo (6 por participante), em 6 (2 por participante) a abordagem sugeriu reestruturação. Em 10 dos 12 casos restantes, a abordagem indicou exatamente a mesma opção que o participante. Nos 2 casos que diferiram, o participante concordou em adotar a seleção da abordagem Orion.
3. Na maioria dos casos, o participante adotou as seleções de políticas de controle de concorrência e sugestões feitas pela abordagem Orion ou analisaria novamente sua escolha. Somente em um caso, o participante afirmou que não adotaria a sugestão, pois, segundo ele, o tempo utilizado na resolução de conflitos foi inferior ao tempo em que houve concorrência. Este participante utilizou a dimensão tempo para tomar sua decisão, no entanto, o tempo entre a tentativa frustrada de *check-in* e o *check-in* propriamente dito não representa o real período de tempo que o desenvolvedor necessitou para a resolução dos conflitos. A abordagem Orion, por sua vez, leva em consideração a quantidade de operações necessárias para a resolução dos conflitos, independente do tempo utilizado para a realização dessa tarefa.

4.5.4 Avaliação dos participantes

Após a conclusão de suas tarefas, os participantes de ambos os estudos deveriam preencher um questionário de avaliação. Em geral, a avaliação deste segundo estudo também foi positiva. Sobre o estudo em si, todos os três participantes afirmaram que ficaram satisfeitos com o resultado final das tarefas. O participante P5 afirmou que não sentiu dificuldades na realização das tarefas, no entanto, os outros dois alegaram que sentiram dificuldades parciais, pois não tinham muita experiência no assunto abordado.

A respeito da abordagem Orion, todos os participantes declararam que utilizariam uma ferramenta que implementasse o mecanismo de seleção de políticas da abordagem proposta. Os motivos listados foram:

- "Pois os resultados foram próximos dos meus e os que divergiram me fizeram repensar o problema";
- "Principalmente por sugerir a reestruturação nos casos em que o trabalho de um desenvolvedor está conflitando com o trabalho do outro"; e
- "Pois os resultados obtidos com a ferramenta estavam de acordo com as minhas idéias".

Além disso, apontaram os seguintes aspectos positivos da abordagem Orion:

- "Considera métricas válidas e coletadas automaticamente e contribui fortemente para o planejamento da política de concorrência de ICs em um projeto";
- "Fornece sugestões sobre as políticas de controle de concorrência em função do nível de concorrência e dificuldade de junção. A abordagem também pode servir para o acompanhamento da ocorrência de conflitos após a troca das políticas ou da reestruturação"; e
- "Visa reduzir riscos durante o desenvolvimento de software; pode ser "rodada" durante várias etapas do desenvolvimento para melhor adequar as políticas de acesso, e aponta falhas na estrutura do software".

Nenhum aspecto negativo da abordagem foi listado pelos participantes, no entanto, algumas sugestões para melhoria foram fornecidas:

- "Futuramente, se fosse possível e a ferramenta já não o faça, seria interessante em ICs críticos a exibição dos sub-ICs que são mais críticos também";

- "Tentar sugerir algo mais específico para a reestruturação, como, por exemplo, tais métodos devem mudar para outra classe, etc."; e
- Possibilitar a visualização das informações históricas adicionais da mesma forma que foi utilizado no estudo, por meio de gráficos.

Em relação às perguntas sobre o controle de concorrência no desenvolvimento de software, dois participantes afirmaram que tinham conhecimento das políticas e apenas o participante P6 declarou que seu conhecimento era parcial. Além disso, os três consideram que é importante que os sistemas de controle de versão possibilitem a utilização de diferentes políticas para diferentes ICs, pois, segundo eles:

- "Como as tarefas demonstraram, uma única política de concorrência poderia impactar negativamente na produtividade da equipe trabalhando sobre determinados ICs";
- "Permite maior controle sobre a evolução dos ICs"; e
- "Como mostrado na tarefa realizada, nem todos os ICs se comportam da mesma forma, sendo necessária uma política de concorrência para cada um".

Por fim, sobre a influência de uma escolha adequada das políticas de controle de concorrência na produtividade da equipe de desenvolvimento, todos os participantes afirmaram ser positiva, pois, segundo eles:

- "Permite aproveitar melhor as potencialidades de desenvolvimento de cada membro da equipe, reduzindo o tempo gasto para resolução de conflitos";
- "Os problemas relacionados com conflitos podem ser resolvidos de acordo com as particularidades de desenvolvimento de cada item"; e
- "Não será perdido tempo na resolução de conflitos, quando estes puderem ser evitados".

4.6 Validade

Estes estudos preliminares foram executados a fim de analisar a viabilidade e a aplicação da abordagem proposta neste trabalho de pesquisa. A leitura dos resultados obtidos e da avaliação dos participantes nos ajudou a entender alguns pontos que precisam ser melhorados na abordagem Orion, assim como outros benefícios que podem ser alcançados com a utilização da abordagem. No entanto, devido às restrições deste estudo, estes resultados não podem ser generalizados.

A seleção dos participantes foi feita por meio da solicitação de voluntários dentro de um grupo de alunos que compartilham um mesmo laboratório de pesquisa do qual também fazem parte o experimentador e os pesquisadores responsáveis pelo estudo. Isto foi necessário devido a restrições de tempo e de pessoal disponível. Como conseqüência, o grupo escolhido pode não ser representativo para a população que se deseja testar e pode ser influenciado pela sua relação com o experimentador. Além disso, houve somente um número reduzido de participantes neste estudo. É possível que os resultados sejam influenciados pelo tamanho e pelas características específicas do grupo. O uso de alunos de pós-graduação, não tão experientes quanto profissionais da indústria, também restringe a generalização das observações obtidas.

A tarefa de seleção de políticas de controle de concorrência utilizada durante o estudo era pequena e simples em comparação a projetos reais. Isto foi necessário devido a limitações de tempo para cada sessão do estudo. A simplicidade desses ICs também pode ter influenciado os resultados obtidos. Acreditamos, porém, que a abordagem Orion tem o potencial de ser ainda mais útil em projetos maiores.

Outra limitação deste estudo é o fato de que as tarefas foram executadas logo após as apresentações dos conceitos que seriam abordados no estudo, como, por exemplo, as políticas de controle de concorrência, sem que os participantes, que não tinham total conhecimento, tivessem tempo para amadurecer e assimilar todos os assuntos discutidos. O resultado do estudo pode ser influenciado pela falta de tempo de maturação do conhecimento necessário para realizar as tarefas propostas corretamente. Acreditamos ainda que a abordagem Orion possa ser melhor aplicada por indivíduos com um maior entendimento sobre a abordagem e a aplicação das políticas de controle de concorrência.

4.7 Considerações finais

Neste capítulo, foram apresentados dois estudos de avaliação da abordagem proposta nesta dissertação. O estudo de viabilidade buscou analisar como a tarefa de seleção de políticas de controle de concorrência é executada diante dos dados fornecidos pela grande maioria dos atuais sistemas de controle de versão. Além disso, foi analisado se o uso de algumas informações históricas adicionais, identificadas neste trabalho de pesquisa, poderia fornecer algum tipo de apoio na execução desta mesma tarefa.

No estudo de observação, buscamos analisar a abordagem Orion em termos de sua aplicação em um projeto de desenvolvimento de software. Além disso, foi analisado

se o mecanismo de seleção de políticas de controle de concorrência estava de acordo com o pensamento dos participantes.

Um ponto importante a ser discutido é o tempo empregado para analisar o histórico dos ICs e fazer a seleção de políticas de controle de concorrência para cada IC do projeto. Como foi visto na Seção 4.4.3, os participantes levaram em média 30 minutos pra analisar as 27 operações de *check-in*, de apenas três desenvolvedores, que constam no histórico de somente 6 ICs do projeto utilizados no estudo. Um projeto da indústria certamente apresenta um número bem superior de *check-ins* e ICs. Analisando o repositório do sistema de controle de versão do projeto Subversion (COLLINS-SUSSMAN *et al.*, 2004), por exemplo, é possível observar que desde o início de desenvolvimento do projeto, em agosto de 2001 até hoje, cerca de 140 desenvolvedores executaram mais de 33.000 *check-ins* e atualmente existem mais de 5.000 ICs no projeto, considerando arquivos como ICs. Comparando com o tempo que os participantes levaram para executar suas tarefas, a Tabela 4.13 apresenta uma estimativa de tempo que seria necessário para selecionar as políticas de controle de concorrência para o projeto Subversion, levando em consideração o número de ICs e quantidade de *check-ins* e assumindo um crescimento linear.

Tabela 4.13. Estimativa de tempo para seleção de políticas para o projeto Subversion.

Projeto	ICs	*Check-ins*	Tempo (horas)
Estudo de avaliação	6	27	0,5
Projeto Subversion	5000	33000	509259

Diante da grandeza desses números, seria inviável que alguém fizesse manualmente a análise do histórico desses ICs e sugerisse as políticas de controle de concorrência que deveriam ser adotadas. Por outro lado, com uma ferramenta para automatizar esse processo, certamente essa tarefa seria mais fácil e rápida de ser executada. Isso motivou o desenvolvimento de um protótipo, que é apresentado no próximo capítulo.

Por fim, esses estudos representam um primeiro passo para a avaliação da abordagem Orion, proposta neste trabalho de pesquisa. Como foi discutido na Seção 4.6, os estudos são limitados em diferentes aspectos, restringindo a generalização dos resultados alcançados. Certamente, são necessários estudos adicionais para avaliar de forma mais completa o que foi proposto.

Capítulo 5 – O Protótipo Orion

5.1 Introdução

No capítulo 4, foram descritos dois estudos de avaliação da abordagem Orion, que motivaram a construção de um protótipo que implementa a abordagem proposta. Neste capítulo, são discutidas as decisões tomadas com relação ao desenvolvimento desse protótipo.

Para atingir seus objetivos, a abordagem Orion utiliza informações históricas de sistemas de controle de versão. Porém, dependendo do sistema de controle de versão escolhido, algumas dessas informações não são coletadas ou não estão disponíveis. Devido a isso, o protótipo Orion foi implementado sobre o sistema de controle de versão Odyssey-VCS (OLIVEIRA et al., 2005; MURTA *et al.*, 2007; MURTA *et al.*, 2008), desenvolvido pelo Grupo de Reutilização de Software da COPPE/UFRJ. A adoção do Odyssey-VCS possibilita que modificações sejam feitas no sistema, permitindo assim que todas as informações necessárias à aplicação da abordagem Orion sejam coletadas, armazenadas e acessadas.

No entanto, nos sistemas de controle de versão em que os dados necessários para a aplicação da abordagem Orion não estão disponíveis, essas informações históricas podem ser obtidas por meio de ferramentas adicionais que as capturam ou por meio de técnicas que podem ser adotadas pelos desenvolvedores, fazendo com que elas sejam armazenadas por esses sistemas. Essas ferramentas e técnicas são apresentadas e discutidas neste capítulo.

Mesmo não implementando o protótipo para outros sistemas de controle de versão, neste capítulo, é discutido como a abordagem Orion poderia ser utilizada com sistemas comerciais e mais conhecidos, como o Subversion, por exemplo.

Este capítulo está organizado da seguinte forma: a Seção 5.2 apresenta o Odyssey-VCS e suas evoluções; a Seção 5.3 descreve o protótipo Orion desenvolvido para trabalhar com o Odyssey-VCS; a Seção 5.4 trata como a abordagem Orion pode ser implementada sobre outros sistemas de controle de versão; e, por fim, a Seção 5.5 apresenta algumas considerações finais.

5.2 Odyssey-VCS

O Odyssey-VCS (OLIVEIRA *et al.*, 2005; MURTA *et al.*, 2007; MURTA *et al.*, 2008) é um sistema de controle de versão para elementos da UML (*Unified Modeling Language*) (OMG, 2005; OMG, 2006). No contexto dessa dissertação, um elemento da UML é uma instância de qualquer conceito descrito no meta-modelo da UML, como, por exemplo, pacote, classe, operação, atributo, entre outros.

Esse sistema tem como objetivo apoiar o desenvolvimento de software orientado a modelos, que surgiu como uma importante técnica para o desenvolvimento de software. As abordagens orientadas a modelos focam na definição de modelos de alto nível e aplicam transformações para obter artefatos de implementação (BEYDEDA *et al.*, 2005). Uma das mais conhecidas organizações por trás do desenvolvimento orientado a modelo é a OMG (*Open Management Group*) (OMG, 2008), que incentiva o uso da UML, entre outros padrões. Devido a este cenário, a UML está se tornando mais do que uma notação para documentação de software.

No passado, grande parte da infra-estrutura construída de Gerência de Configuração de Software (GCS) oferecia suporte somente ao desenvolvimento e manutenção de código-fonte, que não trabalha de forma correta com modelos. Um dos elementos chave dessa infra-estrutura é o sistema de controle de versão, que, como discutido no Capítulo 2, é responsável por manter as diversas versões e configurações do sistema de software de forma organizada. A maioria dos sistemas de controle de versão atuais é baseada na estrutura de sistemas de arquivo, enquanto linguagens de modelagem são baseadas em estruturas de alto nível (ESTUBLIER, 2000; ESTUBLIER *et al.*, 2005).

Recentemente, o Odyssey-VCS recebeu importantes evoluções, o que deu origem à versão 2 do sistema. A seguir, a Seção 5.2.1 apresenta a versão inicial do Odyssey-VCS e a Seção 5.2.2 apresenta a nova versão e suas novas características importantes para essa dissertação.

5.2.1 Versão 1

A primeira versão do Odyssey-VCS (OLIVEIRA *et al.*, 2005; MURTA *et al.*, 2007) tinha como principal objetivo prover um sistema de controle de versão que oferecesse apoio aos arquitetos na modelagem concorrente de sistemas de software utilizando diferentes ferramentas CASE baseadas na UML.

Além disso, durante a construção do Odyssey-VCS, foram abordados alguns desafios descritos na literatura de GCS (ESTUBLIER, 2000), como: modelo de dados que trata objetos complexos; versionamento homogêneo para diferentes tipos de objetos; espaços de trabalho distribuídos e heterogêneos; e trabalho concorrente com modelos de alto nível.

Para atingir esses objetivos, o Odyssey-VCS seguia especificações difundidas, como MOF (*Meta Object Facility*) (OMG, 2002), UML, formato XMI (*XML Metadata Interchange*) (OMG, 2007) e JMI (*Java Metadata Inferface*) (DIRCKZE, 2002).

Para se comunicar com as ferramentas CASE baseadas em UML, o Odyssey-VCS utiliza serviços Web (BOOTH *et al.*, 2005). As ferramentas CASE externalizam modelos UML como arquivos XMI e enviam esses arquivos para o Odyssey-VCS por meio de chamadas de serviços Web. No lado do servidor, o Odyssey-VCS, por sua vez, carregava esses arquivos XMI em repositórios MOF chamados MDR (*Metadata Repository*) (MATULA, 2008) e manipulava os modelos UML via API JMI. Cada arquivo XMI, que representa uma versão específica do modelo UML, tornava-se uma extensão no repositório MDR.

No lado cliente, com o intuito de não alterar de forma relevante o modo de trabalho dos desenvolvedores, o Odyssey-VCS trabalha de maneira semelhante à grande parte dos sistemas de controle de versão baseados em arquivos. Os modelos são armazenados em repositórios centrais e podem ser obtidos pelos desenvolvedores por meio da operação de *check-out*. Os desenvolvedores podem trabalhar sobre os modelos de forma paralela e salvam suas alterações no repositório por meio da operação de *check-in*.

A Figura 5.1 apresenta uma captura da tela da versão 1 do Odyssey-VCS. Nela são apresentados as funcionalidades principais de interesse do desenvolvedor e dois espaços: à esquerda, o repositório (*Repository*) e à direita, o espaço de trabalho do desenvolvedor (*Workspace*). Além das operações de *check-in* e *check-out*, existem as operações para fazer a autenticação do usuário e conexão em um repositório (*Login*); carregar uma versão do modelo no espaço de trabalho a partir de um arquivo (*Load*); exportar para um arquivo o modelo que foi feito *check-out* e está presente no espaço de trabalho (*Save*); fazer a inserção inicial da primeira versão do modelo (*Insert*) e listar as versões do modelo e seus ICs que estão no repositório (*List*).

A interface cliente oferece operações importantes para a interação com os desenvolvedores, no entanto, o Odyssey-VCS atua principalmente no lado do servidor,

provendo o controle de concorrência e o versionamento de modelos. Devido a isso, a representação visual das diferenças dos modelos e a edição dos modelos no lado do cliente são feitas em ferramentas externas, como as ferramentas CASE. É importante ressaltar que o Odyssey-VCS não é destinado a um diagrama UML específico. Ele trabalha sobre os modelos UML e pode versionar qualquer tipo de informação contida nesses modelos.

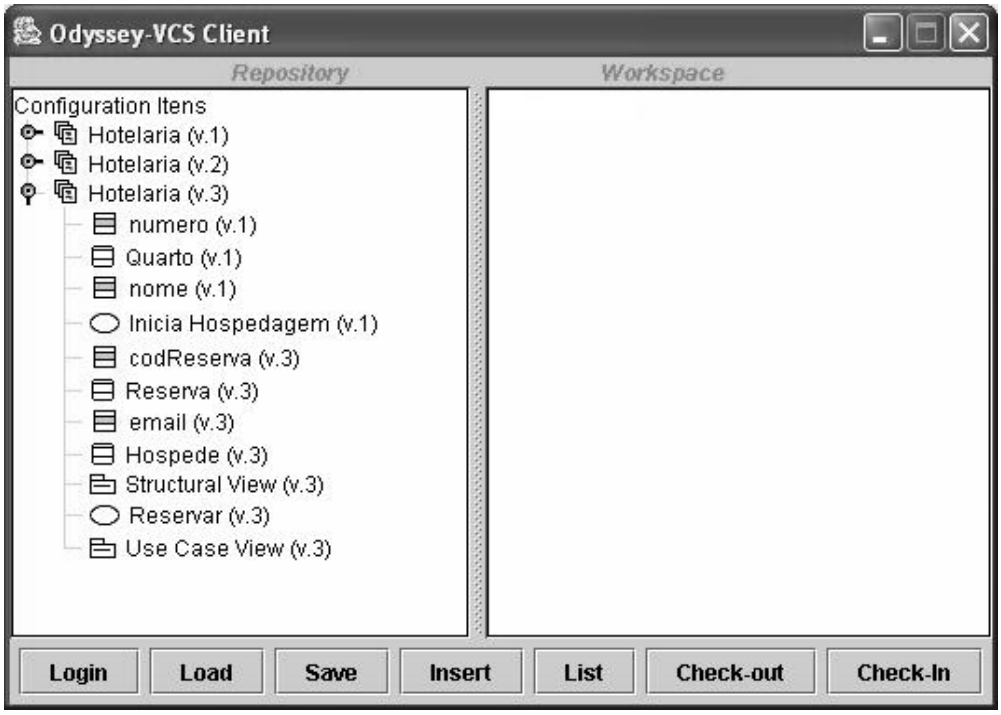


Figura 5.1. Versão 1 do Odyssey-VCS.

O Odyssey-VCS possui seu próprio modelo de versionamento. Esse modelo é responsável por armazenar as informações de versionamento e manter a ligação dessas informações de versionamento com o modelo de dados. No caso do Odyssey-VCS, o modelo de dados é uma instância do meta-modelo da UML.

Por fim, o Odyssey-VCS mantém um descritor de comportamento por projeto, que informa como cada tipo de elemento do modelo UML deve ser manipulado. O descritor de comportamento determina se a informação de evolução é necessária ou não para um elemento específico do modelo UML. Esta informação de evolução inclui uma identificação de versão única e uma informação contextual auxiliar, como quem modificou o elemento, quando ele foi modificado e porque ele foi modificado. Além disso, o descritor de comportamento indica quais elementos são considerados atômicos, com o intuito de detectar conflitos. O Odyssey-VCS sinaliza um conflito toda vez que dois ou mais desenvolvedores modificam concorrentemente um elemento que é considerado atômico.

5.2.2 Versão 2

A segunda versão do Odyssey-VCS (MURTA *et al.*, 2008) engloba diversas evoluções sobre a primeira. Algumas dessas evoluções demandaram mudanças no modelo de versionamento do Odyssey-VCS. As novas características são:

- *Suporte a UML 2*. A versão 1 do Odyssey-VCS apresentava alguns problemas de desempenho (MURTA *et al.*, 2007), principalmente devido a sobrecargas impostas pelo MDR. Além disso, o projeto MDR segue a especificação MOF 1.4, que não oferece suporte a UML 2. Devido a isso, na versão 2 do Odyssey-VCS, o MDR foi substituído por uma nova infra-estrutura de meta-modelagem, chamada *Eclipse Modeling Framework* (EMF) (ECLIPSE FOUNDATION, 2008c), que possui um módulo específico que provê uma implementação para o meta-modelo da UML 2 (ECLIPSE FOUNDATION, 2008b).
- *Processamento reflexivo*. Na sua primeira versão, o Odyssey-VCS utilizava manipuladores (*handlers*) para tratar tipos específicos de elementos de modelos. Dessa forma, existia uma classe de manipulação para cada tipo de elemento de modelo do meta-modelo UML. Isso causava um suporte incompleto ao meta-modelo UML e dificuldades na sua manutenção. Na versão 2 do Odyssey-VCS, foi utilizada uma API reflexiva provida pela EMF que possibilitou a criação de algoritmos genéricos para o tratamento dos elementos de modelo.
- *Algoritmo genérico de junção*. As vantagens trazidas pelo processamento reflexivo permitiram uma melhoria no algoritmo de junção, possibilitando, por exemplo, maior facilidade de manutenção perante a evolução do meta-modelo. O novo algoritmo substituiu o antigo algoritmo específico, aplicando uma estratégia genérica a todos os tipos de elemento de modelo, devido à utilização da API reflexiva provida pelo EMF.
- *Suporte a hooks*. Uma deficiência da primeira versão do Odyssey-VCS dizia respeito aos mecanismos de extensão. Na nova versão são providos mecanismos para ativar ferramentas externas em resposta a eventos específicos. Geralmente, este tipo de suporte é necessário para executar tarefas externas antes ou após um determinado evento.

Além dessas quatro novas características da versão 2 do Odyssey-VCS discutidas até aqui, existem mais três características que representam parte do esforço deste trabalho de pesquisa, relacionado à coleta e ao armazenamento das informações históricas necessárias à aplicação da abordagem Orion, identificadas no Capítulo 3. Essas outras características são descritas nas próximas seções.

5.2.2.1 Suporte a ramos

Na primeira versão do Odyssey-VCS, quando uma operação de *check-in* falhava, devido a conflitos, as versões dos ICs não eram armazenadas no repositório. Como foi discutido no Capítulo 3, essas informações são importantes para identificarmos quando e quais conflitos ocorreram ao longo da existência do projeto.

Para permitir o armazenamento dessas informações, a versão 2 do Odyssey-VCS oferece suporte à criação de ramos. Além disso, essa nova funcionalidade possibilita que o trabalho dos desenvolvedores possa ocorrer em diferentes linhas de desenvolvimento (como visto no Capítulo 2), o que não era possível na versão 1 do sistema de controle de versão.

A versão 2 do Odyssey-VCS possui infra-estrutura para a criação de dois tipos de ramos: explícito e implícito. Um ramo explícito ocorre quando o usuário explicitamente cria um ramo de desenvolvimento a partir de outro ramo. Já a criação implícita ocorre sem a intervenção direta do usuário. É uma criação automática de ramos, utilizada para guardar a "intenção do usuário" no momento do *check-in*.

A Figura 5.2 demonstra como funciona a criação implícita de ramos. Sempre que um desenvolvedor faz *check-in* de algum IC, a versão que ele está levando para o repositório fica guardada em um ramo implícito. Isso acontece em todas as operações de *check-in*. Portanto, com a introdução do mecanismo de criação implícita de ramos, foi necessário modificar o algoritmo de *check-in* do Odyssey-VCS. Na versão 2, o *check-in* consiste basicamente em criar o ramo implícito, aplicar as modificações do desenvolvedor no ramo e tentar fazer a junção do ramo criado com o ramo principal de desenvolvimento. Se a junção falhar, devido a conflitos, a "intenção" do desenvolvedor ficará armazenada no repositório. Ele então precisará resolver os conflitos e depois voltar a fazer o *check-in*. O novo *check-in*, com os conflitos resolvidos, irá diretamente para a linha principal de desenvolvimento, finalizando o ciclo de vida do ramo implícito criado.

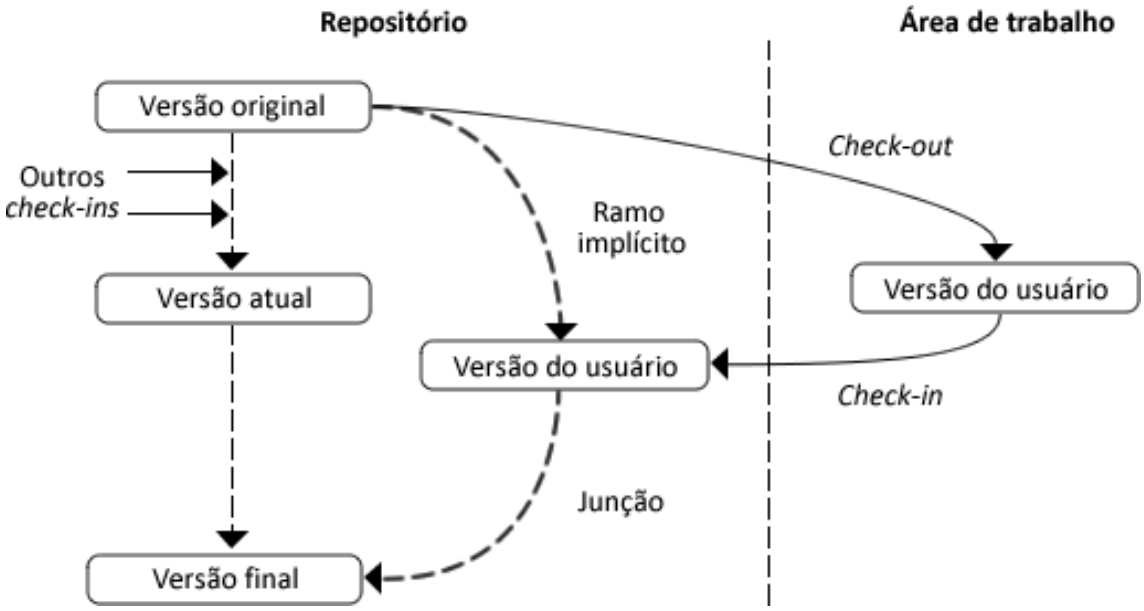


Figura 5.2. Criação implícita de ramos.

Antes de possuir suporte a ramos, o Odyssey-VCS contava com dois ponteiros em cada versão, que permitiam a navegação entre as diversas versões de um IC. Esses ponteiros indicavam a versão anterior (*previous*) e a versão posterior (*next*) de uma determinada versão. Apenas esses dois ponteiros eram suficientes, pois somente poderia existir uma linha de desenvolvimento.

Na versão 2 do Odyssey-VCS, entretanto, mais alguns ponteiros foram criados para possibilitar uma completa navegação entre as versões de um IC nas diversas linhas de desenvolvimento. Além dos ponteiros *previous* e *next*, foram adicionados: o ponteiro *branched from*, que aponta para a versão de onde foi criado o ramo; os ponteiros *branched into*, que apontam para os ramos criados a partir de uma determinada versão; o ponteiro *merged from*, que indica de que ramo foi feita a junção que deu origem a versão analisada; e os ponteiros *merged into*, que apontam para as versões de outros ramos originados a partir da versão analisada. Vários ramos diferentes podem ser criados a partir de uma mesma versão, por isso, podem existir vários ponteiros dos tipos *branched into* e *merged into*.

A Figura 5.3 apresenta uma tela da versão 2 do Odyssey-VCS. Em comparação com a versão 1, continuam presentes as duas áreas que representam o repositório e o espaço de trabalho do desenvolvedor, mas os comandos de *Login*, *Check-in*, *Check-out*, entre outros, estão organizados no menu *Commands*. Na parte inferior da janela, foi inserido um painel com algumas informações referentes à versão selecionada. Entre essas informações estão: o nome do IC (*name*), versão selecionada (*version*), o

desenvolvedor responsável pelo *check-in* (*author*), a data do *check-in* (*date*), o comentário do *check-in* que deu origem a versão selecionada (*comment*), se a versão pertence a um ramo implícito (*auto-branch*) e os ponteiros discutidos nesta seção (*branched from*, *branched into*, *merged from* e *merged into*). No exemplo da figura, temos que a versão 1.2.1 da classe *Client* pertence a um ramo implícito criado a partir da versão 1 e que deu origem a versão 3 deste IC. Vale ressaltar que os valores dos ponteiros *previous* e *next* não estão definidos, pois essa é a única versão do ramo.

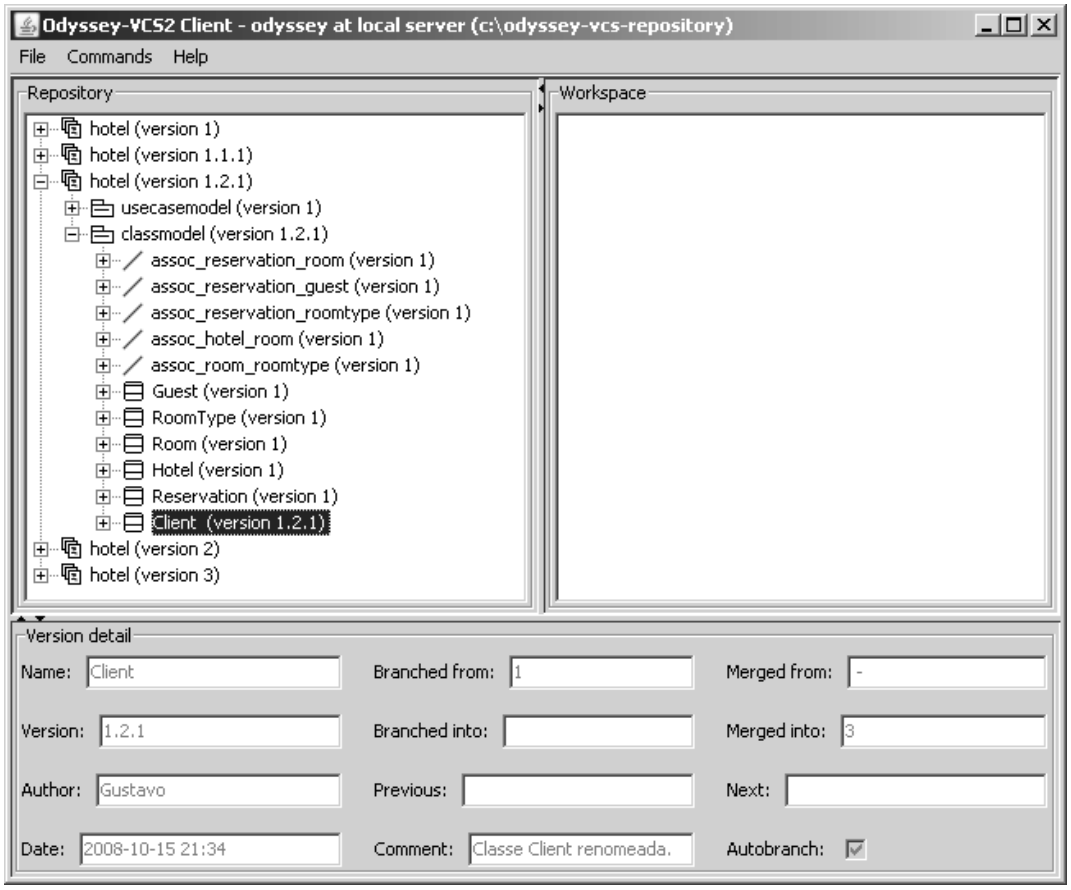


Figura 5.3. Ramos e ponteiros na versão 2 do Odyssey-VCS.

A Figura 5.3, também, destaca a forma de numeração das versões do Odyssey-VCS. As versões que pertencem à linha principal de desenvolvimento são numeradas com apenas um número inteiro, como, por exemplo, as versões 1, 2 e 3 do modelo *hotel* representado na figura. Os ramos são numerados utilizando três números inteiros. O primeiro deles indica de que versão do modelo o ramo foi criado. O segundo deles representa um seqüencial dos ramos criados a partir de uma mesma versão, já que podem ser criados vários ramos a partir de uma mesma versão do modelo. Por fim, o terceiro número indica a versão do ramo. Um ramo que possui versão 1.2.1, como o

exemplo da figura, indica que essa é a primeira versão do segundo ramo criado a partir da versão 1 do modelo.

Somente os ICs que foram modificados e os ICs que os contém recursivamente recebem uma nova versão. No exemplo representado pela Figura 5.3, somente o IC *Client* foi modificado. Dessa forma, o IC *Client e* os ICs *classmodel* e *hotel* (ICs que englobam *Client*) receberam uma nova versão (1.2.1). Os ICs que não foram modificados permanecem com a versão que o desenvolvedor fez *check-out* (versão 1).

5.2.2.2 Suporte a política pessimista

A versão anterior do Odyssey-VCS somente permitia a adoção da política otimista. Atualmente, na versão 2 do Odyssey-VCS, a estratégia padrão é a otimista, no entanto, o desenvolvedor pode explicitamente selecionar a política pessimista para qualquer IC do projeto, por meio do comando de bloqueio (*lock*) (Figura 5.4).

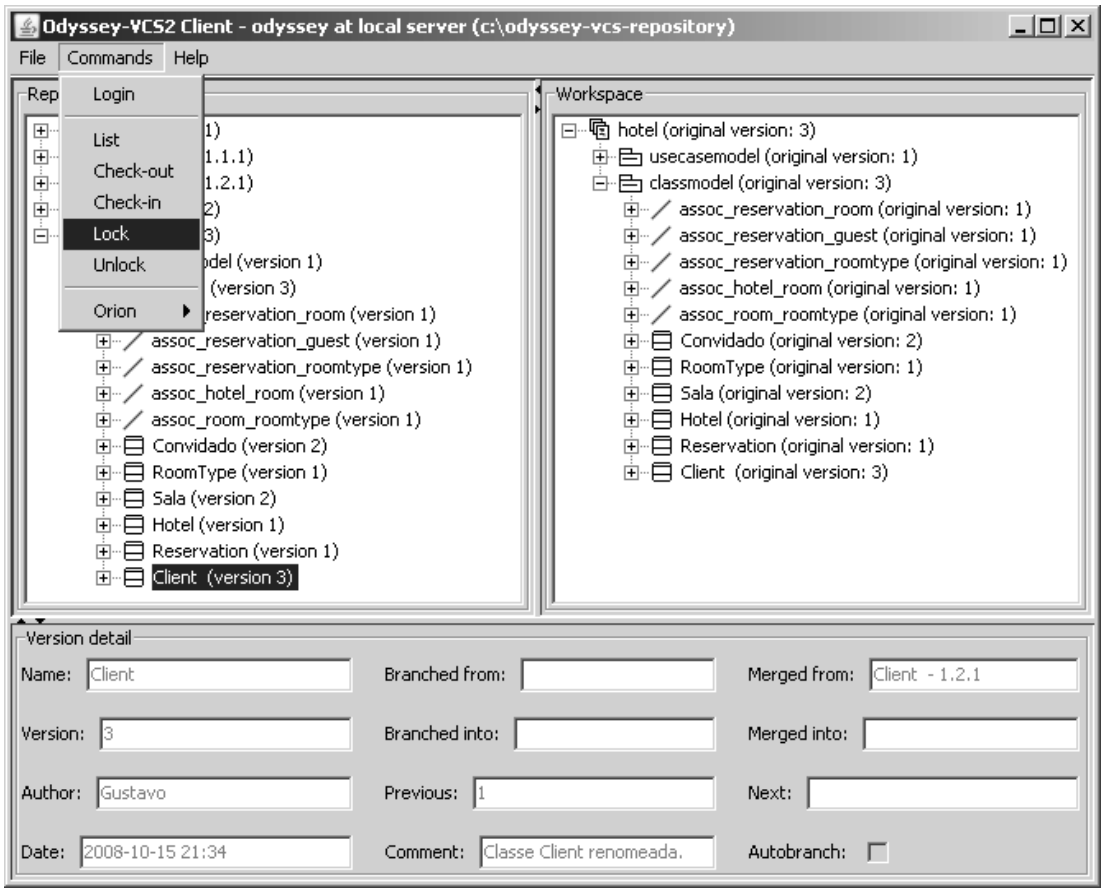


Figura 5.4. Política pessimista no Odyssey-VCS.

Quando um IC é bloqueado, cria-se uma transação de bloqueio que associa o IC ao desenvolvedor que executou o comando. Além disso, essa transação armazena um comentário do desenvolvedor, relacionado ao bloqueio.

Durante o período em que um IC está bloqueado, nenhum *check-in* é aceito, exceto o *check-in* executado pelo responsável pelo bloqueio. Quando isso ocorre, automaticamente o bloqueio é removido. Outra forma de remover um bloqueio é utilizar o comando de desbloqueio (*unlock*).

5.2.2.3 Momento do *check-out*

Outra modificação implementada na versão 2 do Odyssey-VCS para permitir a aplicação da abordagem Orion foi o armazenamento do dia e hora que foram executadas as operações de *check-out*.

Sempre que é executada uma operação de *check-out*, é criada uma transação de *check-out* com o momento em que a operação foi efetuada. O identificador dessa transação é anexado às versões dos ICs que o desenvolvedor recebe por meio do *check-out*. Dessa forma, quando o desenvolvedor executar a operação de *check-ins* desses ICs, é possível saber de que versão ele fez *check-out* e qual o momento que isso ocorreu.

5.3 Protótipo Orion

O protótipo Orion implementa a abordagem descrita no Capítulo 3 e foi desenvolvido como um *plug-in* para o Odyssey-VCS. As funcionalidades do protótipo são acessadas por meio de um menu do sistema de controle de versão (Figura 5.5).

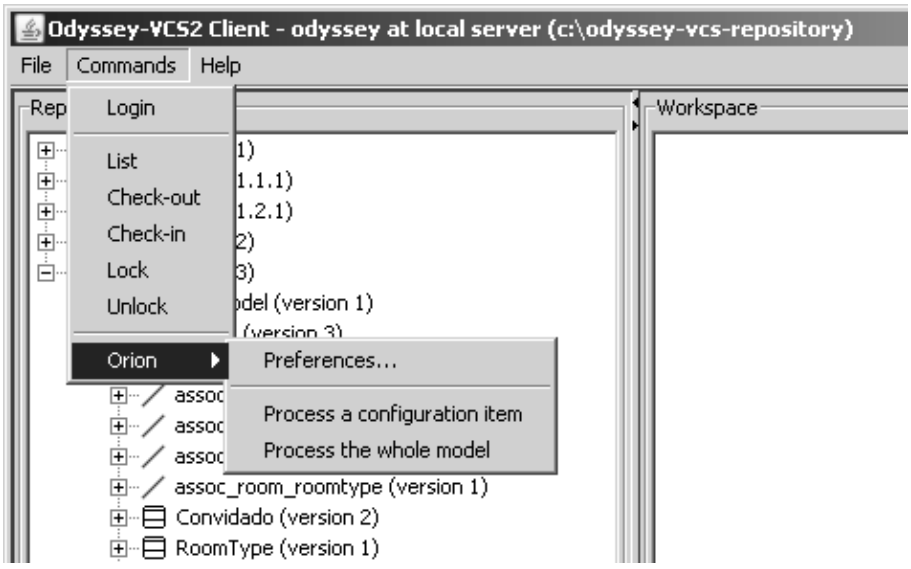


Figura 5.5. Protótipo Orion.

O primeiro passo para executar o protótipo é ajustar os pontos de corte das métricas discutidos no Capítulo 3. Esses valores são utilizados para a seleção das políticas de controle de concorrência e identificação de elementos críticos. Essa

configuração é feita por meio da opção *Preferences* do menu (Figura 5.5). Em seguida, o usuário deve escolher um dos dois modos de operação:

- Individual (opção *Process a configuration item*), onde a abordagem Orion é executada para um determinado IC escolhido; ou
- Completo (opção *Process the whole model*), onde a abordagem Orion é executada para todos os ICs pertencentes ao modelo.

A seguir, nas Seções 5.3.1 e 5.3.2, é discutido como são calculadas as duas métricas utilizadas na abordagem. Mais adiante, para dar continuidade ao exemplo de utilização do protótipo, a Seção 5.3.3 apresenta como os resultados são exibidos para os usuários e, na Seção 5.3.4, a questão da visualização dos resultados é abordada.

5.3.1 Métrica de concorrência

Como foi discutida no Capítulo 3, a métrica de concorrência mede em quanto do tempo que os desenvolvedores levaram para modificar um IC houve concorrência.

Para calcular essa métrica no protótipo Orion para um determinado IC, o primeiro passo consiste em recuperar as informações e analisar cada uma das operações de *check-in* que envolveram esse IC, exceto a primeira, pois não existe uma operação de *check-out* associada a ela.

Para cada um desses *check-ins* são recuperados: o nome do desenvolvedor que executou o *check-in*, o momento que o desenvolvedor executou essa operação e o momento em que foi executada a operação de *check-out* relacionada a esse *check-in*. A forma como essa última informação é recuperada foi discutida na Seção 5.2.2.3.

A Figura 5.6 apresenta um algoritmo simplificado do cálculo da métrica de concorrência para um IC. Inicialmente, o usuário escolhe uma versão de um determinado IC. Cada versão está relacionada a uma operação de *check-in*, que por sua vez está relacionada a uma operação de *check-out*, ambas efetuadas por um determinado usuário. A partir da versão selecionada, as versões anteriores do IC são percorridas, utilizando o ponteiro *previous*, discutido na Seção 5.2.2.1. Para cada uma dessas versões, exceto a primeira, são identificados os momentos que foram efetuados o *check-in* e o *check-out* desse IC e o usuário responsável por essas operações. Com essas informações, é instanciado um objeto da classe `Checkin`. Este objeto é armazenado em um vetor (`vetorCheckins`). Após serem percorridas todas as versões, o vetor está construído e, a partir da análise desse vetor, são identificados o período de tempo em que esse IC ficou em posse dos desenvolvedores para modificação e o período de tempo

em que houve concorrência. Por fim, é feita a divisão desses valores e chega-se ao resultado da métrica de concorrência para o IC selecionado.

```
versão := versãoEscolhida;

enquanto (versão.getPrevious() ≠ nulo)
      dataCheckin := getCheckinTime(versão);
      dataCheckout := getCheckoutTime(versão);
      usuário := getUser(versão);
      Checkin checkin :=
            new Checkin(dataCheckin, dataCheckout, usuário);
      vetorCheckins.insere(checkin);
      versão := versão.getPrevious();

tempoPosse := getPossessionTime(vetorCheckins);
tempoConcorrência := getConcurrencyTime(vetorCheckins);
métricaConcorrência := tempoConcorrência / tempoPosse;
```

Figura 5.6. Algoritmo simplificado do cálculo da métrica de concorrência.

No modo completo de execução do protótipo Orion, esse procedimento é feito para cada IC que compõe o modelo analisado.

5.3.2 Métrica de dificuldade de junção

Como foi discutida no Capítulo 3, a métrica de dificuldade de junção representa a porcentagem das ações executadas em um determinado IC que foram efetuadas especialmente para resolução de conflitos.

A Figura 5.7 apresenta uma versão simplificada do algoritmo utilizado para o cálculo dessa métrica para um IC. Inicialmente, a métrica deve ser calculada separadamente para cada operação de *check-in* que está associada a uma determinada versão do IC. Essas versões são percorridas por meio da navegação utilizando o ponteiro *previous* (discutido na Seção 5.2.2.1), a partir da versão selecionada pelo usuário. Para cada uma dessas versões, são identificadas as seguintes versões: *versão original*, *versão do usuário*, *versão atual* e *versão final*, apresentadas no Capítulo 3.

Se for identificado que para uma determinada operação de *check-in* não houve trabalho concorrente, ou seja, a versão atual do IC é igual a versão original, o valor da métrica para esse *check-in* é 0. Nos *check-ins* em que houve concorrência, os sub-ICs que sofreram conflito são analisados. Para isso, devem ser comparadas as versões dos

sub-ICs de cada uma das quatro versões do IC identificadas previamente. Por meio da comparação das versões dos sub-ICs das versões original e final do IC, é identificado o conjunto de *ações efetivadas* no IC. Comparando as versões dos sub-ICs das versões original, atual e do usuário do IC, é identificado o conjunto de *ações aplicadas* no IC.

```
versão = versãoEscolhida;

enquanto (versão.getPrevious() ≠ nulo)
    vUsuário := versão.getMergedFrom();
    vOriginal := vUsuário.getBranchedFrom();
    vAtual := versão.getPrevious();
    vFinal := versão;

    se (vAtual = vOriginal)     // Não houve concorrência
        dificuldade := 0;
    senão                       // Houve concorrência
        cjAcoesAplicadas := diff3(vUsuário, vAtual, vOriginal);
        cjAcoesEfetivadas := diff(vFinal, vOriginal);
        cjEsforcoAdicional :=
            DiferencaCj(cjAcoesEfetivadas, cjAcoesAplicadas);
        dificuldade := qtdeElementos(cjEsforcoAdicional) /
            qtdeElementos (cjAcoesEfetivadas);

    vetorDificuldades.insere(dificuldade);
    versão = versão.getPrevious();

métricaDificuldade := médiaAritmética(vetorDifuculdades);
```

Figura 5.7. Algoritmo simplificado do cálculo da métrica de dificuldade de junção.

Por fim, é identificado o conjunto de ações que representaram *esforço adicional*, por meio da diferença entre os conjuntos *ações efetivadas* e *ações aplicadas*. Para chegar ao resultado da métrica de dificuldade de junção de um *check-in*, o número de elementos do conjunto *esforço adicional* é dividido pelo número de elementos do conjunto *ações efetivadas*.

Os resultados da métrica para cada *check-in* é armazenado em um vetor e depois de todos os *check-ins* processados, é feita a média aritmética desses valores. Esta média representa o resultado final da métrica de dificuldade de junção para o IC selecionado.

Da mesma forma que a métrica de concorrência, no modo completo de execução do protótipo, esse procedimento é feito para cada IC que compõe o modelo analisado.

5.3.3 Resultados

No modo individual, os resultados das métricas (*concurrency* e *merge effort*) são simplesmente apresentados para o usuário por meio de uma janela *pop-up*. Além disso, é exibido o nível crítico do IC (*critical level*) e é indicada a política de controle de concorrência (*Orion suggestion*) a ser adotada para o IC analisado (Figura 5.8). Dependendo dos valores das métricas, o IC analisado pode ser identificado como um elemento crítico.

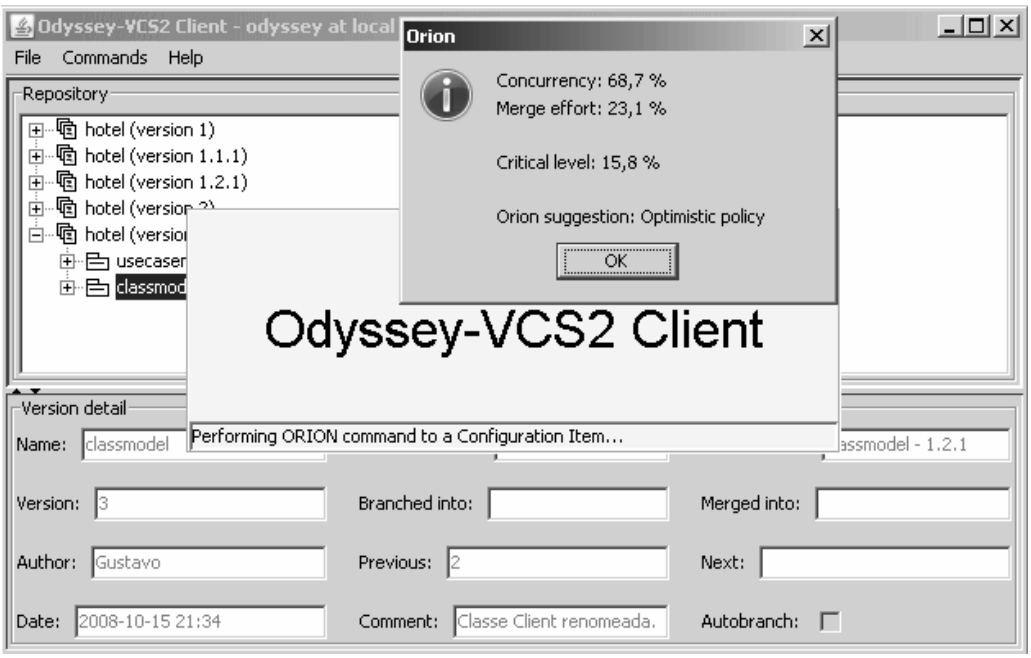


Figura 5.8. Apresentação dos resultados no modo individual.

No modo completo, o usuário escolhe uma versão do modelo e processa todos os seus ICs. O protótipo gera um modelo UML com os valores das métricas, as indicações de políticas e as indicações de elementos críticos anexados ao modelo gerado por meio de estereótipos e valores etiquetados (*tagged values*), que seguem um perfil (*profile*) determinado[11]. Esse modelo UML é, então, exportado para um arquivo XMI marcado com os resultados da abordagem Orion (Figura 5.9). O desenvolvedor pode assim ter acesso aos resultados utilizando a ferramenta CASE que ele utiliza para modificação dos modelos.

[11] Perfis, estereótipos e valores etiquetados constituem mecanismos padrão de extensão da UML (OMG, 2006).

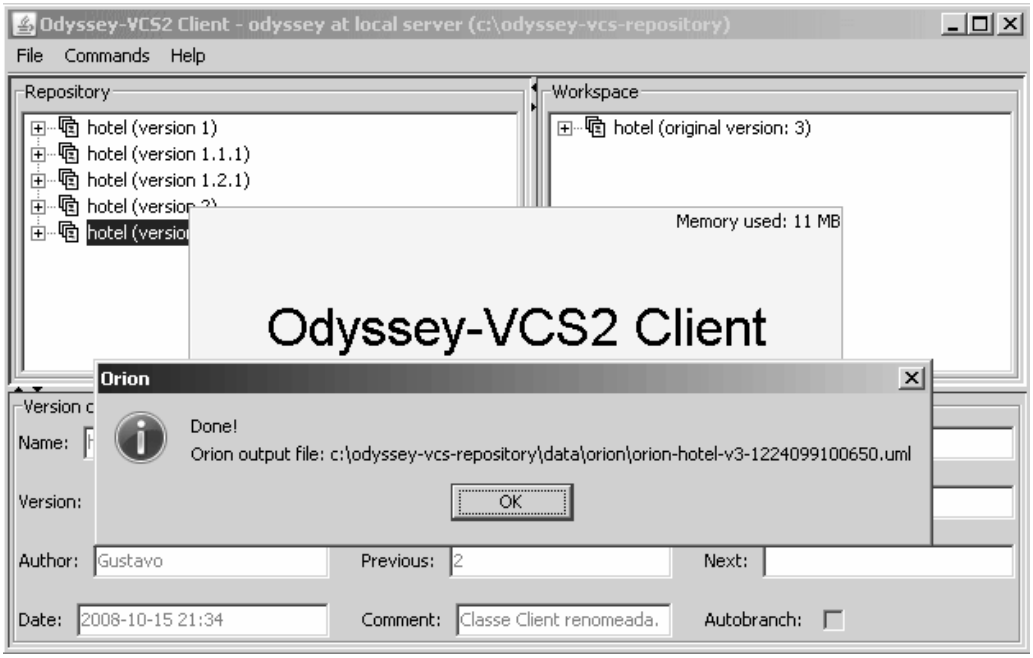


Figura 5.9. Resultados no modo completo.

Além disso, nos dois modos de operação, são armazenados no repositório, arquivos com o histórico de utilização do protótipo.

5.3.4 Visualização dos resultados

No Capítulo 3, foram discutidos aspectos relacionados à aplicação de técnicas de visualização de software para a representação dos resultados gerados pela abordagem Orion. Para atingir esse objetivo, foi utilizado o mecanismo de visualização EvolTrack (CEPEDA *et al.*, 2008).

O mecanismo EvolTrack provê uma infra-estrutura de visualização da evolução do software, capaz de capturar as várias versões existentes de um projeto de software, armazenadas em um sistema de controle de versão, e representá-las. Esse mecanismo foi implementado como um *plug-in* do ambiente de desenvolvimento Eclipse (ECLIPSE FOUNDATION, 2008a).

As informações capturadas a partir de uma fonte de dados servem como base para a criação da representação do ciclo de evoluções do projeto. O Odyssey-VCS é uma das fontes de dados que podem ser utilizadas com o EvolTrack. Neste caso, o Odyssey-VCS fornece o modelo UML de cada uma das versões armazenadas no seu repositório e o EvolTrack transforma cada modelo em um formato específico de saída, como, por exemplo, diagramas de classes UML. Quando o protótipo Orion é utilizado, os modelos UML de cada versão são enviados para o EvolTrack marcados com os resultados da abordagem Orion (Figura 5.10).

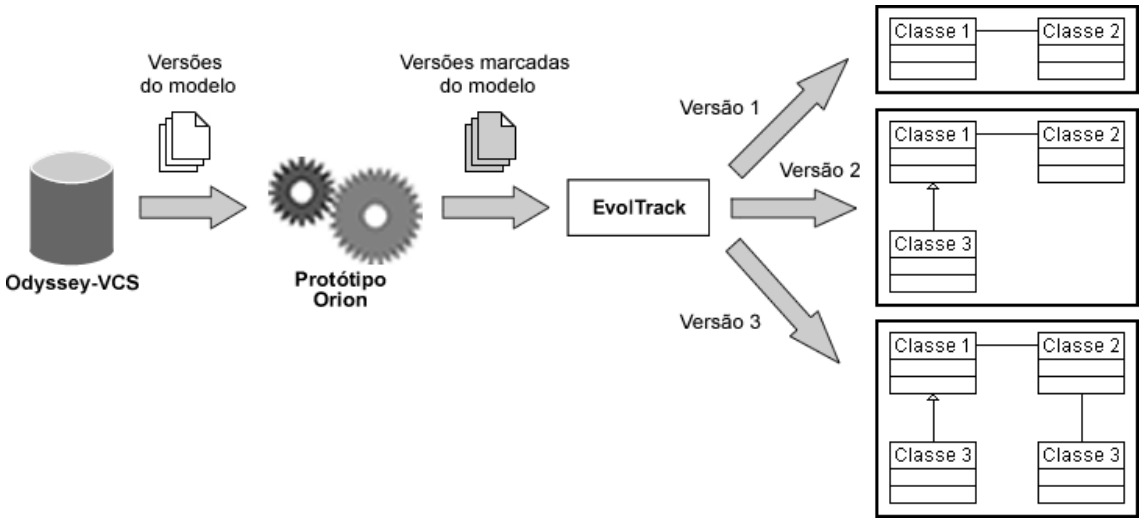


Figura 5.10. Integração EvolTrack e protótipo Orion [Adaptado de CEPEDA *et al.*, (2008)].

Além de representar a estrutura dos modelos UML, o EvolTrack também é capaz de interpretar e representar métricas anexadas a esses modelos. Recentemente, o EvolTrack foi integrado com o protótipo Orion (CEPEDA *et al.*, 2008) e passou a representar as métricas de concorrência, dificuldade de junção e nível crítico dos ICs analisados pela abordagem Orion. A Figura 5.11 apresenta uma tela do EvolTrack com a visualização de um modelo UML marcado com os resultados da abordagem. Os ICs (classes, neste exemplo) estão coloridos com base no seu nível crítico. Os ICs mais avermelhados representam elementos críticos. Além disso, é possível visualizar a política de controle de concorrência mais indicada para cada IC.

Outra vantagem da utilização do protótipo Orion e do EvolTrack é a possibilidade de analisar a evolução das métricas propostas. Como foi visto na Figura 5.10, o EvolTrack gera uma visualização para cada versão do modelo do projeto, que são coloridas de acordo com o valor de alguma das métricas. Dessa forma, o EvolTrack, utilizando as várias visualizações geradas, produz uma animação, permitindo a análise da variação dos valores das métricas (por meio, da variação da cor do IC) ao longo do tempo de vida do projeto.

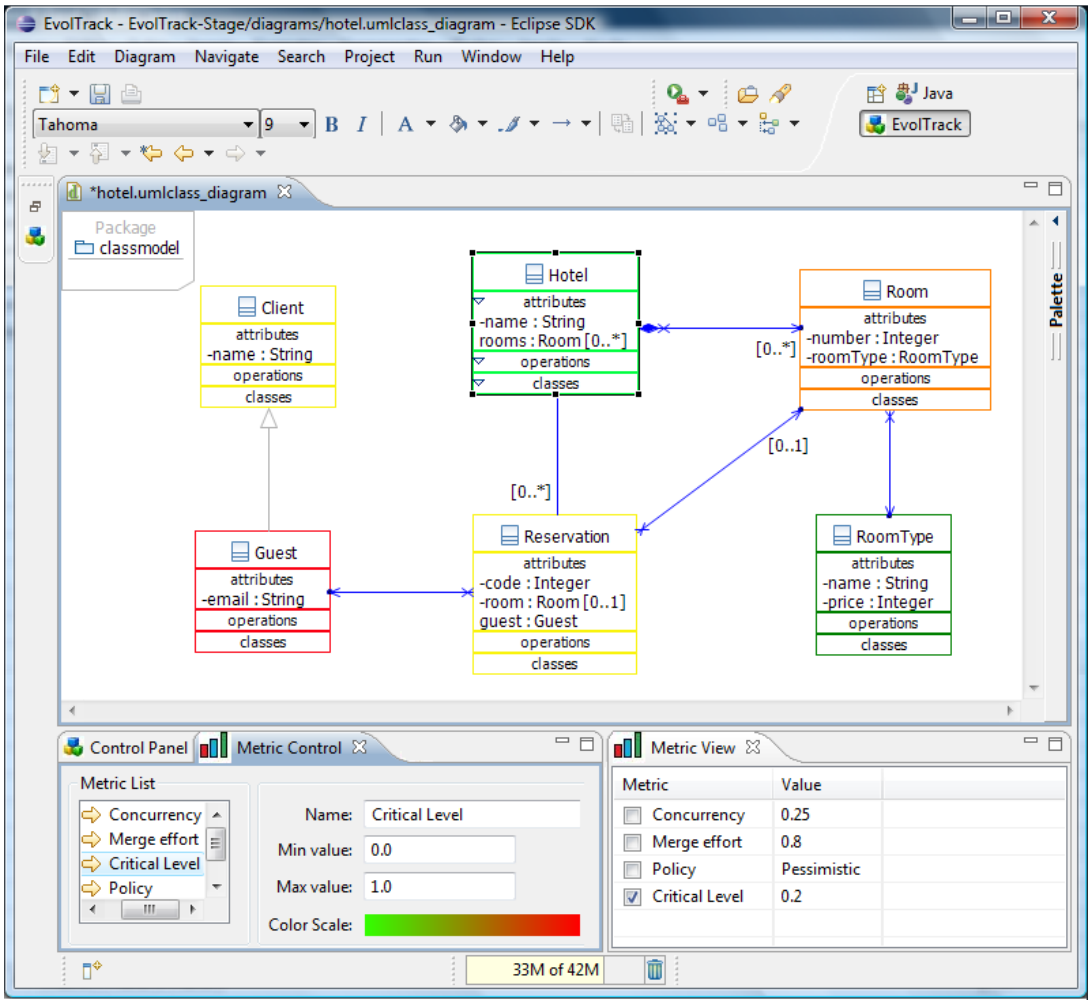


Figura 5.11. Resultados da abordagem Orion visualizados no EvolTrack.

5.4 Implementação da abordagem Orion utilizando outros sistemas de controle de versão

A abordagem Orion foi desenvolvida de forma genérica, sem se prender a nenhum sistema de controle de versão em particular, nem a sistemas voltados para elementos UML. Neste capítulo, foi apresentada uma implementação da abordagem Orion que utiliza como fonte de dados o Odyssey-VCS. Como discutido na Seção 5.1, essa decisão foi tomada devido aos sistemas de controle de versão mais utilizados atualmente não fornecem algumas das informações necessárias à aplicação da abordagem Orion.

Nesta seção, é discutido como a abordagem proposta poderia ser implementada utilizando outros sistemas de controle de versão mais difundidos, focando no Subversion, um dos sistemas livres mais utilizados.

Basicamente, esses são os três aspectos que possibilitam a aplicação da abordagem Orion com esses outros sistemas:

1. Suporte a política pessimista e otimista e possibilidade de selecionar diferentes políticas para diferentes ICs;
2. Registro do momento das operações de *check-out* e atualização dos ICs; e
3. Acesso às versões dos ICs em tentativas não efetivadas de *check-in*, devido a conflitos ("intenção" do usuário).

A Tabela 5.1 apresenta como esses três aspectos são tratados em alguns sistemas de controle de versão.

Tabela 5.1. Comparação entre sistemas de controle de versão.

	Sub-version	**CVS**	**IBM ClearCase**	**Visual SourceSafe**	**Bit-Keeper**	**Odyssey-VCS Versão 2**
Suporte à políticas	Sim	Não	Sim	Não	Não	Sim
Momento *check-out*	Não	Parcial-mente	Não	Parcial-mente	Não	Sim
Intenção do usuário	Não	Não	Não	Não	Sim	Sim

Alguns sistemas não suportam a estratégia pessimista de forma satisfatória, como, por exemplo, o CVS e o BitKeeper. Outros sistemas, como o Visual SourceSafe, por exemplo, suportam as duas estratégias, mas não permitem a seleção de diferentes políticas para diferentes ICs, sendo necessário escolher uma única estratégia para todos ICs do projeto. Nesses casos, a utilização da abordagem Orion fica comprometida, já que seu principal objetivo é selecionar as políticas de controle de concorrência para cada IC. Por outro lado, o Subversion é um exemplo de sistema de controle de versão que oferece suporte a ambas as estratégias e permite que sejam selecionadas diferentes políticas para diferentes ICs do projeto.

Com relação ao momento do *check-out*, em geral, os sistemas de controle de versão não armazenam esse tipo de informação. O Visual SourceSafe, por ser um sistema que adota basicamente a política pessimista, bloqueia os ICs sempre que o desenvolvedor faz um *check-out* (*check-out* reservado). Nesse caso, o momento do *check-out* é armazenado como o momento em que os ICs foram bloqueados. Já o CVS apresenta o mecanismo *watch,* que percebe quando o desenvolvedor inicia modificações nos ICs e envia uma notificação por email, que pode ser usada na abordagem Orion, substituindo o momento do *check-out*. Nos outros sistemas de controle de versão, essa informação pode ser coletada utilizando ferramentas externas que monitorem os acessos

ao repositório e identifiquem essas operações. O Subversion, por exemplo, normalmente é instalado juntamente com o servidor Web Apache (APACHE SOFTWARE FOUNDATION, 2008). Nesse caso, é possível criar uma aplicação no Apache que monitora o repositório do Subversion e permite a execução de ferramentas externas quando um *check-out* ou uma atualização é executado (RYAN DESIGN, 2008).

Sobre o terceiro aspecto levantado, alguns sistemas de controle de versão como, por exemplo, o BitKeeper, implementam o mecanismo de criação implícita de ramos, semelhante à solução implementada no Odyssey-VCS e discutida na Seção 5.2.2.1. Nos sistemas que isso não acontece, como, Subversion, por exemplo, uma possível solução é desenvolver uma ferramenta externa que sempre que um desenvolvedor fizer um *check-in*, crie um ramo e em seguida faça a junção do ramo criado com o ramo que o desenvolvedor está trabalhando. Dessa forma, a "intenção" do usuário seria armazenada e poderiam ser identificadas as situações de conflitos.

5.5 Considerações finais

Neste capítulo, foi apresentado um protótipo que implementa a abordagem proposta neste trabalho de pesquisa. O protótipo funciona com o sistema de controle de versão Odyssey-VCS, um sistema acadêmico. No entanto, foi feita uma discussão de alto nível sobre como a abordagem pode ser implementada em sistemas de controle de versão mais utilizados. Isso é importante para a difusão da abordagem proposta.

Inicialmente, foram discutidas as modificações feitas no Odyssey-VCS que possibilitaram a coleta e o armazenamento das informações necessárias para a aplicação da abordagem Orion e, em seguida, foi apresentado um *plug-in* do Odyssey-VCS que implementa a abordagem. Além disso, foi discutido como os resultados são gerados e como eles podem ser representados para os usuários, utilizando uma ferramenta que aplica técnicas de visualização de software para facilitar a compreensão e assimilação dos resultados gerados pela abordagem.

O protótipo desenvolvido, no entanto, possui algumas limitações. Pelo fato da implementação ter sido voltada para o Odyssey-VCS, somente projetos que utilizam esse sistema de controle de versão podem ser analisados pelo protótipo. Além disso, o protótipo ainda não foi utilizado para projetos grandes e reais. Somente foram feitos testes e demonstrações simples, com modelos pequenos e históricos de desenvolvimento curtos. Portanto, questões relacionadas à escalabilidade, desempenho e interação humano computador ainda precisam ser analisadas.

Capítulo 6 – Conclusão

6.1 Epílogo

O desenvolvimento de software é um processo em que, normalmente, várias pessoas colaboram. Atualmente, o número de desenvolvedores em um projeto vem aumentando cada vez mais, devido à necessidade de entregar aos clientes sistemas de software mais complexos, com maior qualidade e em um tempo mais curto. Além disso, a competição entre empresas que constroem esses tipos de sistemas contribui ainda mais para esse aumento. Portanto, para que o processo de desenvolvimento de software ocorra de forma organizada, é preciso prover mecanismos de controle de concorrência.

Os sistemas de controle de versão representam uma classe de sistemas que permite a aplicação desses mecanismos. Eles fornecem um repositório central, onde são armazenados os ICs do projeto, e provêem políticas de controle de acesso a esses ICs. Para permitir que os desenvolvedores possam trabalhar ao mesmo tempo sobre os mesmos artefatos, a política otimista é bastante utilizada. No entanto, a adoção dessa estratégia pode trazer impactos à produtividade da equipe, devido à necessidade de tempo e esforço extra para resolução de possíveis conflitos que, em alguns casos, poderiam ser evitados se bloqueios (política pessimista) fossem aplicados.

Visando minimizar esse problema, este trabalho de pesquisa apresentou a abordagem Orion, que analisa todo o histórico de modificações dos ICs de um projeto de desenvolvimento, armazenado em algum sistema de controle de versão, e seleciona a política de controle de concorrência mais indicada para cada IC, com o intuito de minimizar as situações de conflitos e, conseqüentemente, aumentar a produtividade da equipe. Além disso, a abordagem Orion identifica os ICs que possuem um histórico relevante de concorrência e dificuldade de resolução de conflitos, apontando-os como elementos críticos e sugerindo que esses elementos sofram algum tipo de reestruturação.

Um protótipo da abordagem proposta foi desenvolvido utilizando como fonte de dados o sistema de controle de versão Odyssey-VCS. Além disso, foram feitos dois estudos preliminares para avaliar a abordagem Orion. Apesar de restritos, os estudos indicaram algumas oportunidades de melhoria da abordagem. Em geral, a avaliação dos participantes foi positiva.

Neste último capítulo, as principais contribuições são destacadas na Seção 6.2. Na Seção 6.3, são apresentadas as limitações deste trabalho. Finalmente, na Seção 6.4, são discutidos os potenciais trabalhos futuros.

6.2 Contribuições

Esta dissertação apresentou os resultados de um trabalho de pesquisa que teve como objetivo propor uma abordagem para seleção de políticas de controle de concorrência no processo de desenvolvimento de software. Entre as suas principais contribuições, podemos destacar:

- Definição de uma abordagem genérica, denominada Orion, para oferecer: (1) um mecanismo de seleção de políticas de controle de concorrência para os diferentes ICs do projeto; (2) um mecanismo para identificação dos elementos críticos; e (3) um mecanismo de visualização dos resultados da abordagem. Esses resultados podem ser utilizados por diferentes papéis no processo de desenvolvimento, como, por exemplo, gerente de projeto, gerente de configuração e desenvolvedores;
- Definição de duas métricas relacionadas ao controle de concorrência: *concorrência* e *dificuldade de junção*, utilizadas na abordagem proposta, que se baseiam nas informações históricas de modificações dos ICs armazenadas nos sistemas de controle de versão;
- Desenvolvimento de um protótipo que implementa a abordagem proposta, utilizando os dados históricos do sistema de controle de versão Odyssey-VCS. Além disso, foi desenvolvida a integração do protótipo com a infra-estrutura EvolTrack, que possibilitou a implementação do mecanismo de visualização proposto na abordagem Orion;
- Avaliação preliminar da abordagem proposta, por meio de dois estudos realizados com alunos de pós-graduação. Os resultados obtidos nos estudos poderão ser utilizados para melhorar a abordagem Orion e servirão como base para uma avaliação futura mais completa da abordagem.

Podemos ainda ressaltar as seguintes contribuições secundárias:

- Estudo das áreas de Controle de Concorrência no Desenvolvimento de Software e Gerência de Configuração de Software, visando encontrar as limitações existentes;

- Identificação das informações importantes para a tomada de decisão relacionada à seleção das políticas de controle de concorrência. Essas informações são utilizadas para calcular as duas métricas definidas pela abordagem Orion. Além disso, foi efetuado um estudo dos principais sistemas de controle de versão, visando verificar a disponibilização dessas informações por esses sistemas;
- Discussão inicial de como a abordagem Orion poderia ser implementada utilizando como fonte de dados sistemas de controle de versão comerciais, mais difundidos e utilizados.

6.3 Limitações

A partir de uma análise crítica da abordagem proposta e do protótipo desenvolvido, foi possível identificar algumas limitações deste trabalho. As principais são listadas a seguir:

- A abordagem proposta faz uso de algumas informações históricas que precisam ser coletadas e armazenadas pelos sistemas de controle de versão. No entanto, essa coleta e armazenamento não são realizados nos sistemas mais utilizados. Para atenuar essa limitação foram discutidos mecanismos que podem ser adotados para contornar a falta de disponibilização desse tipo de informação nesses sistemas de controle de versão;
- A coleta e o armazenamento das informações históricas necessárias para a aplicação da abordagem Orion podem resultar em uma maior utilização do espaço em disco nos repositórios. Isso acontece devido à necessidade de armazenar as versões que os usuários fazem *check-in* ("intenção" do usuário) em ramos separados;
- Algumas limitações foram identificadas em relação às métricas propostas, apresentadas nas seções de análise crítica das métricas no Capítulo 3. Entre elas podemos destacar: a não consideração da quantidade de desenvolvedores concorrentes e a não identificação da duração real das modificações;
- Apesar da abordagem ser genérica, independente do sistema de controle de versão utilizado, o protótipo desenvolvido somente permite a aplicação da

abordagem Orion em projetos que utilizam o Odyssey-VCS, um sistema de controle de versão acadêmico;

- O protótipo desenvolvido não foi testado com projetos reais e de tamanho significativo. Dessa forma, questões relacionadas à escalabilidade e desempenho não foram analisadas.

6.4 Trabalhos Futuros

A partir do que foi proposto e implementado ao longo deste trabalho, é possível identificar possibilidades de melhoria e expansão da abordagem e do protótipo aqui apresentados. Além disso, foram abertas novas perspectivas de pesquisa. A seguir, possíveis trabalhos futuros são descritos:

- Planejamento e execução de estudos de avaliação mais completos que incluam, por exemplo, a avaliação das métricas propostas e que englobem, além da abordagem Orion, o protótipo desenvolvido. Além disso, um simulador de modificações dos desenvolvedores poderia ser desenvolvido para aprimorar os estudos de avaliação a serem efetuados;
- Aplicação da abordagem e do protótipo implementado em projetos maiores e reais. Isso fornecerá indícios mais fortes a respeito de melhorias na abordagem, como, por exemplo, ajuste e evolução das métricas definidas. Ou ainda, a definição de outras métricas que possam ser utilizadas pela abordagem;
- Análise mais profunda de como a abordagem Orion pode ser implementada em outros sistemas de controle de versão. Em seguida, desenvolver protótipos que implementem a abordagem, utilizando, como fonte de dados, informações históricas armazenadas em sistemas mais conhecidos, como o Subversion, por exemplo;
- Na abordagem proposta, muitas informações são geradas nos cálculos das métricas, mas algumas delas não são utilizadas, como, por exemplo, o *esforço desperdiçado*. Um estudo mais aprofundado pode demonstrar que essas informações podem fortalecer ainda mais alguns indicadores da abordagem, ou podem ser utilizadas para outros fins relacionados ao controle de concorrência;

- As informações históricas adicionais identificadas neste trabalho de pesquisa poderiam também ser utilizadas para outros propósitos, como, por exemplo, permitir a visualização do modo com que cada desenvolvedor trabalha com o controle de versão, incluindo a freqüência de *check-ins* e atualizações dos ICs;
- Estender o mecanismo de visualização, possibilitando a representação das informações históricas adicionais identificadas na forma de gráficos, como os utilizados nos estudos de avaliação da abordagem Orion;
- Para selecionar a política de controle de concorrência e identificar os elementos críticos, a abordagem Orion se baseia na definição de alguns pontos de corte, que podem ser configurados pelos usuários. Estudos mais aprofundados poderiam sugerir aos usuários valores para esses pontos de corte dependendo do projeto analisado;
- Mecanismo de acompanhamento da evolução das seleções de políticas de controle de concorrência e identificação de elementos críticos e notificação de usuários interessados a respeito de possíveis mudanças nas sugestões.
- Em relação aos elementos críticos identificados pela abordagem, a sugestão aos usuários é que sejam aplicadas reestruturações. No entanto, poderiam ser fornecidos maiores detalhes a respeito dessas reestruturações, o que apoiaria os engenheiros de software nessa tarefa, como, por exemplo, sugerir a divisão do IC, separando determinados sub-elementos do IC original.

Referências Bibliográficas

APACHE SOFTWARE FOUNDATION, 2008, "Apache HTTP Server Project". In: http://httpd.apache.org/, acessado em 02 de Outubro.

BARGHOUTI, N., KAISER, G., 1991, "Concurrency control in advanced database applications", *ACM Computing Surveys*, v. 23, n. 3 (September), pp. 269-317.

BARNSON, M., STEENHAGEN, J., WEISSMAN, T., 2003, *The Bugzilla Guide - 2.17.5 Development Release*, The Bugzilla Team.

BERLINER, B., 1990, "CVS II: Parallelizing Software Development". In: *USENIX Winter 1990 Technical Conference*, pp. 341-352, Washingon DC, USA, June.

BERNSTEIN, P., HADZILACOS, V., GOODMAN, N., 1987, *Concurrency control and recovery in database systems*, Addison-Wesley Longman Publishing Co.

BEYDEDA, S., BOOK, M., GRUHN, V., 2005, *Model-Driven Software Development*, Springer.

BITMOVER, 2008, "BitKeeper". In: http://www.bitkeeper.com, acessado em 02 de Outubro.

BOEHM, B., 1991, "Software Risk Management: Principles and Practices", *IEEE Software*, v. 8, n. 1 (1), pp. 32-41.

BOOTH, D., HAAS, H., MCCABE, F., ET AL., 2005, "Web Services Architecture - W3C Working Group Note". In: http://www.w3.org/TR/ws-arch, acessado em 26 de Setembro.

CEMIN, C., 2001, *Visualização de Informações Aplicada à Gerência de Software*, Tese de D.Sc., Instituto de Informática, UFRGS, Porto Alegre, Rio Grande do Sul, Brasil.

CEPEDA, R., MURTA, L.G., WERNER, C., 2008, "Visualizando a Evolução de Software no Desenvolvimento Distribuído", *INFOCOMP*, Aceito para publicação.

CMMI PRODUCT TEAM, 2006, *CMMI for Development, Version 1.2, CMMI-DEV v1.2*, Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute.

COLLINS-SUSSMAN, B., FITZPATRICK, B., PILATO, C., 2004, *Version Control with Subversion*, O'Reilly.

CONRADI, R., WESTFECHTEL, B., 1998, "Version Models for Software Configuration Management", *ACM Computing Surveys*, v. 30, n. 2 (June), pp. 232-282.

DA SILVA, I., CHEN, P., VAN DER WESTHUIZEN, C., et al., 2006, "Lighthouse: Coordination through Emerging Design". In: *Eclipse '06: OOPSLA Workshop on eclipse technology eXchange*, pp. 11-15, Portland, Oregon, October.

DART, S., 1991, "Concepts in Configuration Management Systems". In: *International Workshop on Software Configuration Management (SCM)*, pp. 1-18, Trondheim, Norway, June.

DIRCKZE, R., 2002, *Java Metadata Interface (JMI) Specification, Version 1.0*, Unisys Corporation and Sun Microsystems.

ECLIPSE FOUNDATION, 2008a, "Eclipse". In: http://www.eclipse.org, acessado em 28 de Maio.

ECLIPSE FOUNDATION, 2008b, "EMF-based UML 2.x Metamodel Implementation". In: http://www.eclipse.org/uml2, acessado em 26 de setembro.

ECLIPSE FOUNDATION, 2008c, "Eclipse Modeling Framework (EMF)". In: http://www.eclipse.org/emf, acessado em 26 de Setembro.

ESHEL, M., HARTMAN, C., SCHMUCK, F., et al., 2004, "Dynamically switching between different types of concurrency control techniques to provide an adaptive access strategy for a parallel file system", Patent number: 6826570, November, International Business Machines Corporation.

ESTUBLIER, J., 2000, "Software Configuration Management: a Roadmap". In: *International Conference on Software Engineering (ICSE)*, pp. 279-289, Limerick, Ireland, June.

ESTUBLIER, J., 2001, "Objects Control for Software Configuration Management". In: *Conference on Advanced Information Systems Engineering (CAiSE)*, pp. 359-373, Interlaken, Switzerland, June.

ESTUBLIER, J., GARCIA, S., 2006, "Concurrent Engineering support in Software Engineering". In: *IEEE International Conference on Automated Software Engineering*, pp. 209-220, Tokyo, Japan, September.

ESTUBLIER, J., GARCIA, S., VEGA, G., 2001, "Defining and Supporting Concurrent Engineering policies in SCM". In: *International Workshop on Software Configuration Management*, pp. 1-15, Portland, Oregon, USA, May.

ESTUBLIER, J., LEBLANG, D., VAN DER HOEK, A., et al., 2005, "Impact of Software Engineering Research on the Practice of Software Configuration Management", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, v. 14, n. 4 (October), pp. 1-48.

FOGEL, K., BAR, M., 2001, *Open Source Development with CVS*, The Coriolis Group.

GRAY, J., 1981, "The transaction concept: Virtues and limitations". In: *International Conference on Very Large Data Bases*, p. 144–154, Vallco Parkway, Cupertino CA, .

GRINTER, R., 1995, "Using a Configuration Management Tool to Coordinate Software Development". In: *Conference on Organizational Computing Systems*, pp. 168 - 177, Milpitas, California, August.

HATCHER, E., LOUGHRAN, S., 2004, *Java Development with Ant*, Manning Publications Company.

IEEE, 1987, *Std 1042 - IEEE Guide to Software Configuration Management*, Institute of Electrical and Electronics Engineers.

IEEE, 1990, *Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology*, Institute of Electrical and Electronics Engineers.

IEEE, 2001, *Std 1540 - IEEE Standard for Software Life Cycle Processes - Risk Management*, Institute of Electrical and Electronics Engineers.

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

ISO, 1995, *ISO 10007, Quality Management - Guidelines for Configuration Management*, International Organization for Standardization.

JALOTE, P., 1999, *CMM in Practice: Processes for Executing Software Projects at Infosys*, Addison-Wesley.

JUNQUEIRA, D., 2008, *Um controle de versões refinado e flexível para artefatos de software*, Dissertação de M.Sc., ICMC-USP, São Carlos, SP, Brasil.

KUNG, H.T., ROBINSON, J.T., 1981, "On optimistic methods for concurrency control". In: *ACM Transactions on Database Systems (TODS)*, pp. 213-226, New York, NY, USA, June.

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

LINTERN, R., MICHAUD, J., STOREY, M., et al., 2003, "Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse". In: *ACM*

Symposium on Software Visualization (SoftVis), p. 47-ff, San Diego, California, June.

MACKINLAY, J., SHNEIDERMAN, B., 2000, *Information Visualization, Using Vision to Think*, Morgan Kaufmann.

MATULA, M., 2008, "NetBeans Metadata Repository". In: http://mdr.netbeans.org, acessado em 26 de Setembro.

MURTA, L., 2006, *Gerência de Configuração no Desenvolvimento Baseado em Componentes*, Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

MURTA, L., CORREA, C., PRUDENCIO, J., et al., 2008, "Towards Odyssey-VCS 2: Improvements over a UML-based Version Control System". In: *International Workshop on Comparison and Versioning of Software Models*, pp. 25-30, Leipzig, Germany, May.

MURTA, L., OLIVEIRA, H., DANTAS, C., et al., 2007, "Odyssey-SCM: An Integrated Software Configuration Management Infrastructure for UML models", *Science of Computer Programming*, v. 65, n. 3 (1), pp. 249-274.

OLIVEIRA, H., MURTA, L., WERNER, C., 2005, "Odyssey-VCS: a Flexible Version Control System for UML Model Elements". In: *International Workshop on Software Configuration Management (SCM)*, pp. 1-16, Lisbon, Portugal, September.

OMG, 2002, *Meta Object Facility (MOF) Specification, version 1.4*, Object Management Group.

OMG, 2005, *Unified Modeling Language (UML) Superstructure Specification, version 2.0*, Object Management Group.

OMG, 2006, *Unified Modeling Language (UML) Infrastructure Specification, version 2.0*, Object Management Group.

OMG, 2007, *MOF 2.0/XMI Mapping, Version 2.1.1*, Object Management Group.

OMG, 2008, "Object Management Group". In: http://www.omg.org, acessado em 26 de Setembro.

O'REILLY, C., MORROW, P., BUSTARD, D., 2003, "Improving Conflict Detection in Optimistic Concurrency Control Models". In: *International Workshop on Software Configuration Management*, pp. 191-205, Portland, Oregon, May.

PERRY, D., SIY, H., VOTTA, L., 2001, "Parallel changes in large-scale software development: an observational case study", *ACM Trans. Softw. Eng. Methodol.*, v. 10, n. 3 (1), pp. 308-337.

PRUDENCIO, J., MURTA, L., WERNER, C., 2007a, "Políticas de Controle de Concorrência no Desenvolvimento Distribuído de Software". In: *Workshop de Desenvolvimento Distribuído de Software (WDDS)*, pp. 57-64, João Pessoa, Outubro.

PRUDENCIO, J., MURTA, L., WERNER, C., 2007b, "Aplicando Técnicas de Visualização de Software para Apoiar a Escolha de Políticas de Controle de Concorrência". In: *Simpósio Brasileiro de Engenharia de Software (SBES), Workshop de Teses e Dissertações em Engenharia de Software (WTES)*, pp. 61-66, João Pessoa, Outubro.

REDMILES, D., VAN DER HOEK, A., AL-ANI, B., et al., 2007, "Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects", *Withschaftsinformatik*, v. 49, n. 1 (1), p. 28–38.

ROCHE, T., WHIPPLE, L., 2001, *Essential SourceSafe*, Hentzenwerke Publishing.

RYAN DESIGN, 2008, "Subversion hook dispatcher". In: http://www.ryandesign.com/svnhookdispatcher/, acessado em 02 de Outubro.

SARMA, A., BORTIS, G., VAN DER HOEK, A., 2007, "Towards Supporting Awareness of Indirect Conflicts across Software Configuration Management Workspaces". In: *International Conference on Automated Software Engineering (ASE'07)*, p. 94–103, Atlanta, Georgia, USA, November.

SARMA, A., NOROOZI, Z., VAN DER HOEK, A., 2003, "Palantír: Raising Awareness among Configuration Management Workspaces". In: *International Conference on Software Engineering (ICSE)*, pp. 444-454, Portland, Oregon, May.

SARMA, A., VAN DER HOEK, A., 2004, "A conflict detected earlier is a conflict resolved easier". In: *Collaboration, Conflict, and Control: Workshop on Open Source Software Engineering*, pp. 82-86, Edinburgh, United Kingdom, May.

SHULL, F., CARVER, J., TRAVASSOS, G.H., 2001, "An empirical methodology for introducing software processes", *ACM SIGSOFT Software Engineering Notes*, v. 26, n. 5 (September), pp. 288-296.

SOFTEX, 2007a, *MPS.BR - Melhoria de Processo do Software Brasileiro - Guia Geral (Versão 1.2)*, Associação para Promoção da Excelência do Software Brasileiro.

SOFTEX, 2007b, *MPS.BR - Melhoria de Processo do Software Brasileiro - Guia de Implementação - Parte 5: Nível C (Versão 1.1)*, Associação para Promoção da Excelência do Software Brasileiro.

SOUZA, C.R.B., REDMILES, D., MARK, G., et al., 2003, "Management of interdependencies in collaborative software development". In: *International Symposium on Empirical Software Engineering*, p. 294, Rome, Italy, September.

STOREY, M., CUBRANIC, D., GERMAN, D., 2005, "On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and a Framework ". In: *ACM Symposium on Software Visualization (SoftVis)*, pp. 193-202, St. Louis, Missouri, May.

TICHY, W., 1985, "RCS: a system for version control", *Software - Practice and Experience*, v. 15, n. 7 (July), pp. 637-654.

TIGRIS, 2008, "TortoiseSVN". In: http://tortoisesvn.tigris.org, acessado em 30 de Setembro.

VAN DER LIGEN, R., VAN DER HOEK, A., 2004, "An Experimental, Pluggable Infrastructure for Modular Configuration Management Policy Composition". In: *International Conference on Software Engineering (ICSE'04)*, pp. 573-582, Scotland, UK, May.

WALRAD, C., STROM, D., 2002, "The Importance of Branching Models in SCM", *IEEE Computer*, v. 35, n. (9), pp. 31-38.

WHITE, B., 2000, *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*, Addison-Wesley.

WOHLIN, C., RUNESON, P., HÖST, M., OHLSSON, M., REGNELL, B., WESSLEN, A., 2000, *Experimentation in software engineering: an introduction*, Kluwer Academic Publishers.

Apêndice I

Descrição Geral de Tarefa

Instruções

Este é um estudo de observação, por isso, sempre que possível, verbalize seus pensamentos, para que o experimentador possa melhor avaliar os resultados obtidos. Pergunte e comente tudo que achar necessário.

Contextualização

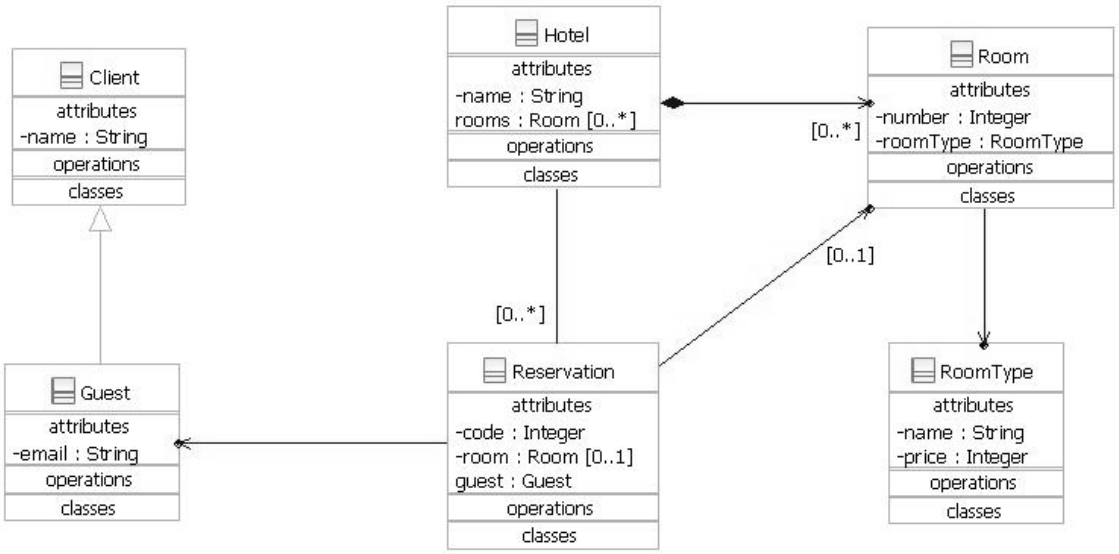
Você acaba de ser contratado pela a empresa SCM para atuar como gerente de configuração dos projetos de desenvolvimento de software da empresa. Recentemente, a empresa decidiu implantar um processo de controle de versões em todos seus projetos. O primeiro projeto em que você irá atuar será o projeto *HotelSystem*, um sistema para gerenciamento de hotéis, que se encontra em etapa de elaboração. A equipe do projeto possui três integrantes que atuam como desenvolvedores e está utilizando um sistema de controle de versão baseados em modelos UML. A sua primeira tarefa é selecionar as políticas de controle de concorrência para cada item de configuração do projeto *HotelSystem*, com o intuito de otimizar a produtividade da equipe de desenvolvimento.

Apêndice II

Descrição do Projeto

Diagrama

Diagrama de classes do projeto HotelSystem.



Itens de configuração adotados no estudo

A seguir estão listados os ICs que serão utilizados no estudo:

1. Client
2. RoomType
3. Guest
4. Hotel
5. Room
6. Reservation

Apêndice III

Histórico do Projeto

IC 1

Check-in	Autor	Data
20	Bruno	21/07/2008 - 11:37
26	Pedro	21/07/2008 - 16:03
34	Pedro	23/07/2008 - 10:57
39	Bruno	23/07/2008 - 16:44

IC 2

Check-in	Autor	Data
24	Marta	21/07/2008 - 15:25
32	Pedro	22/07/2008 - 17:22
38	Pedro	23/07/2008 - 15:38
41	Bruno	23/07/2008 - 17:18

IC 3

Check-in	Autor	Data
22	Marta	21/07/2008 - 14:25
28	Bruno	21/07/2008 - 17:38
36	Marta	23/07/2008 - 14:50
43	Bruno	23/07/2008 - 17:22

IC 4

Check-in	Autor	Data
25	Bruno	21/07/2008 - 15:36
33	Pedro	22/07/2008 - 17:49
37	Marta	23/07/2008 - 15:11
44	Bruno	23/07/2008 - 17:25

IC 5

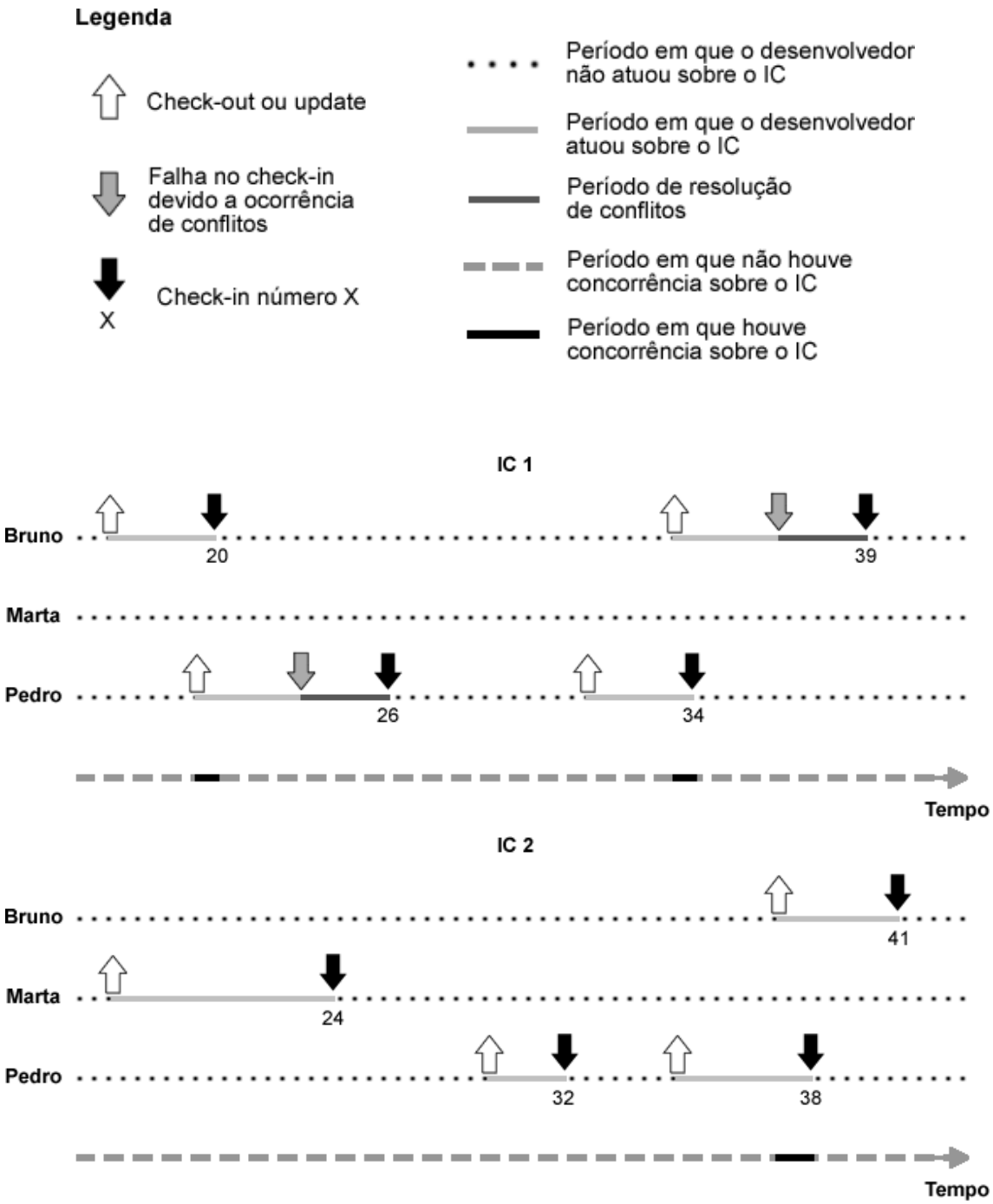
Check-in	Autor	Data
21	Pedro	21/07/2008 - 14:19
29	Marta	21/07/2008 - 18:01
31	Bruno	22/07/2008 - 14:18
35	Pedro	23/07/2008 - 12:20
46	Marta	23/07/2008 - 17:43

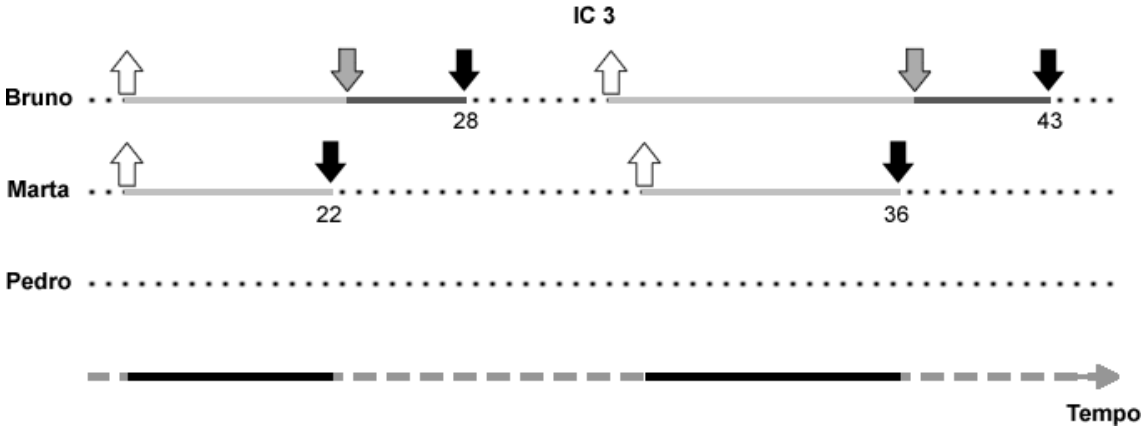
IC 6

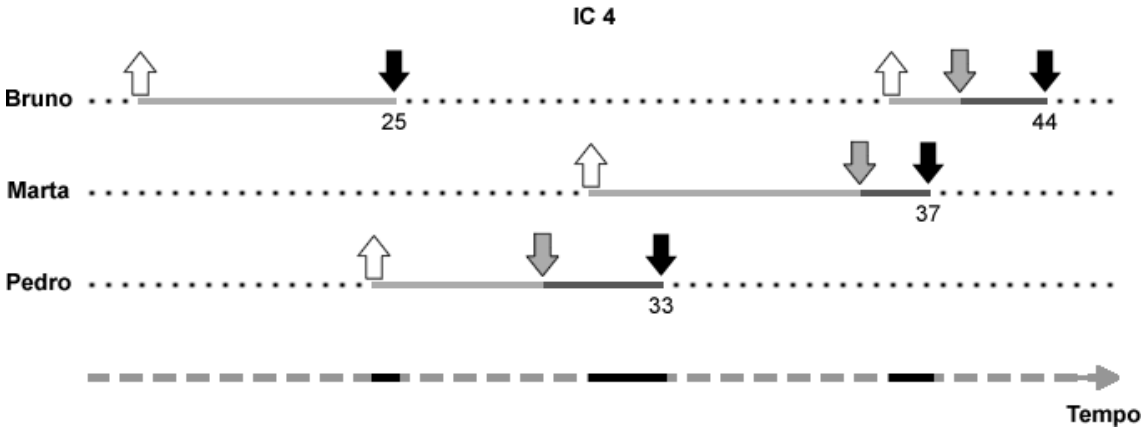
Check-in	Autor	Data
23	Pedro	21/07/2008 - 15:20
27	Marta	21/07/2008 - 17:32
30	Bruno	22/07/2008 - 10:07
40	Marta	23/07/2008 - 17:02
42	Bruno	23/07/2008 - 17:20
45	Pedro	23/07/2008 - 17:32

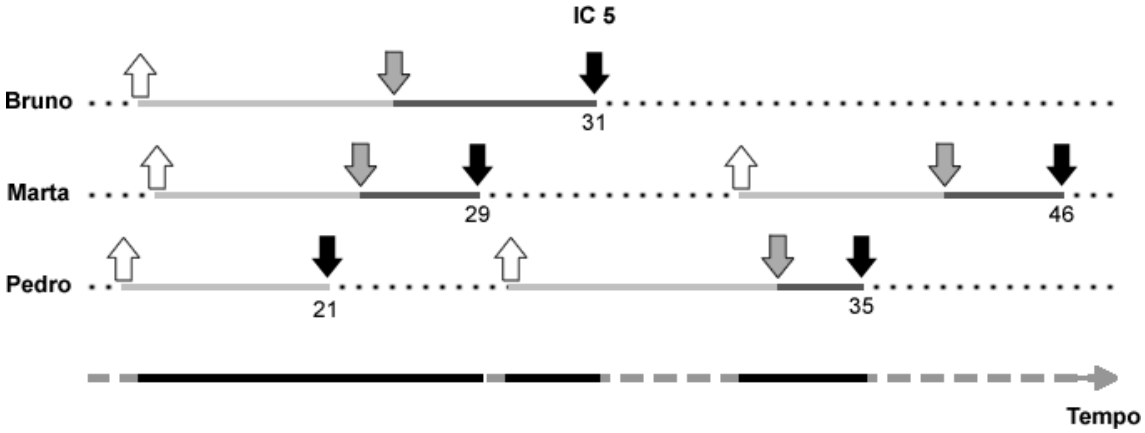
Apêndice IV

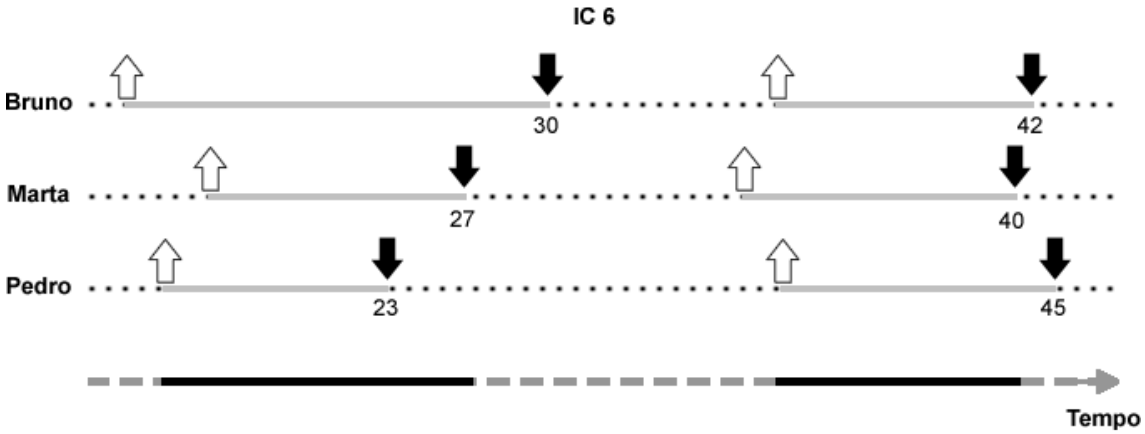
Informações Históricas Adicionais




IC 3
Bruno
28
43
Marta
22
36
Pedro
Tempo


IC 4
Bruno
25
44
Marta
37
Pedro
33
Tempo


IC 5
Bruno
31
Marta
29
46
Pedro
21
35
Tempo


IC 6
Bruno
30
42
Marta
27
40
Pedro
23
45
Tempo

Apêndice V

Formulário de Consentimento

Estudo

Este estudo visa caracterizar a aplicação e viabilidade do mecanismo de seleção de políticas de controle de concorrência da abordagem Orion no apoio ao controle de concorrência no desenvolvimento de software.

Idade

Eu declaro ter mais de 18 anos de idade e concordar em participar de um estudo conduzido por João Gustavo Gomes Prudêncio na Universidade Federal do Rio de Janeiro.

Procedimento

Este estudo acontecerá em uma única sessão, que incluirá análise de alguns elementos UML que fazem parte de um projeto maior, análise de seus históricos armazenados em sistemas de controle de versão e seleção de políticas de controle de concorrência para esses elementos. Eu entendo que, uma vez o experimento tenha terminado, os trabalhos que desenvolvi serão estudados visando entender a eficiência dos procedimentos e as técnicas propostas.

Confidencialidade

Toda informação coletada neste estudo é confidencial, e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

Benefícios e liberdade de desistência

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e apresentado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

Pesquisador responsável

João Gustavo Gomes Prudêncio
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Professores responsáveis

Profª. Cláudia Maria Lima Werner
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Prof. Leonardo Paulino Gresta Murta
Instituto de Computação - Universidade Federal Fluminense (UFF)

Nome (em letra de forma):_____

Assinatura:_____ D**ata**:_____

Apêndice VI

Questionário de Caracterização

Este formulário contém algumas perguntas sobre sua experiência acadêmica e profissional.

1) Formação acadêmica

( ) Doutorado
( ) Doutorando
( ) Mestrado
( ) Mestrando
( ) Graduação

Ano de ingresso: _____ Ano de conclusão (ou previsão de conclusão): _____

2) Formação geral

2.1) Qual é sua experiência com desenvolvimento de software? (marque aqueles itens que melhor se aplicam)

( ) já li material sobre desenvolvimento de software.
( ) já participei de um curso sobre desenvolvimento de software.
( ) nunca desenvolvi software.
( ) tenho desenvolvido software para uso próprio.
( ) tenho desenvolvido software como parte de uma equipe, relacionado a um curso.
( ) tenho desenvolvido software como parte de uma equipe, na indústria.

2.2) Por favor, explique sua resposta. Inclua o número de semestres ou número de anos de experiência relevante em desenvolvimento de software. (E.g. "Eu trabalhei por 2 anos como programador de software na indústria")

_____

_____

_____

2.3) Qual é sua experiência com desenvolvimento de software em equipes? Qual a maior equipe de que você participou?

_____

_____

2.4) Por favor, indique o grau de sua experiência nesta seção seguindo a escala de 5 pontos abaixo:

0 = nenhum
1 = estudei em aula ou em livro
2 = pratiquei em projetos em sala de aula
3 = usei em projetos pessoais
4 = usei em projetos na indústria

2.4.1) UML (*Unified Modeling Language*)	0	1	2	3	4
2.4.2) Modelagem de sistemas de informação	0	1	2	3	4
2.4.3) Sistemas de Controle de Versão (CVS, SVN etc.)	0	1	2	3	4

2.5) Qual(is) sistema(s) de controle de versão você já utilizou?

_____

_____

3) Experiência em contextos diferentes

Esta seção será utilizada para compreender quão familiar você está com o domínio que será utilizado para as atividades durante o experimento. Por favor, indique o grau de experiência nesta seção seguindo a escala de 3 pontos abaixo:

0 = Eu não tenho familiaridade com este domínio.
1 = Eu utilizo isto algumas vezes, mas não sou um especialista.
2 = Eu sou muito familiar com este domínio.

Domínio de Hotéis	0	1	2

Obrigado por sua colaboração!

Apêndice VII

Formulário de Seleção de Políticas - 1

Seleção de políticas de controle de concorrência

Tendo como base as informações fornecidas do projeto e o histórico dos ICs, faça a seleção de políticas de controle de concorrência que deve ser adotada por cada um desses ICs, preenchendo a tabela abaixo. Em seguida, responda as perguntas.

Obs.: Caso você considere que a escolha de política é indiferente, marque "X" nas duas colunas. Caso você considere que nenhuma das opções se aplica, não marque nenhuma das duas colunas e justifique sua resposta no campo de observações. Se necessário, utilize o campo de observações para comentários. O grau de confiança representa a sua confiança na sua escolha.

Grau de confiança: 0 = muito baixa, 1 = baixa, 2 = alta, 3 = muito alta

IC	Política pessimista	Política otimista	Grau de confiança				Observações
1			0	1	2	3	
2			0	1	2	3	
3			0	1	2	3	
4			0	1	2	3	
5			0	1	2	3	
6			0	1	2	3	

1) Quais critérios que você utilizou para chegar neste conjunto de políticas?

_____

_____

_____

_____

2) Você considera o histórico dos ICs uma informação importante para seleção de políticas de controle de concorrência? Quais outras informações você considera útil para fazer essa seleção?

_____

_____

_____

Apêndice VIII

Formulário de Seleção de Políticas - 2

Seleção de políticas de controle de concorrência

Utilizando as informações adicionais sobre o histórico dos ICs do projeto *HotelSystem*, além das informações utilizadas na etapa anterior, faça novamente a seleção de políticas de controle de concorrência que deve ser adotada por cada um desses ICs, preenchendo a tabela abaixo. Em seguida, responda as perguntas, levando em consideração o conjunto de políticas selecionado na etapa anterior.

Obs.: Caso você considere que a escolha de política é indiferente, marque "X" nas duas colunas. Caso você considere que nenhuma das opções se aplica, não marque nenhuma das duas colunas e justifique sua resposta no campo de observações. Se necessário, utilize o campo de observações para comentários. O grau de confiança representa a sua confiança na sua escolha.

Grau de confiança: 0 = muito baixa, 1 = baixa, 2 = alta, 3 = muito alta

IC	Política pessimista	Política otimista	Grau de confiança				Observações
1			0	1	2	3	
2			0	1	2	3	
3			0	1	2	3	
4			0	1	2	3	
5			0	1	2	3	
6			0	1	2	3	

1) Sua seleção de políticas apresentou modificações comparada à seleção efetuada na etapa anterior? Em caso afirmativo, quais novos critérios você utilizou que resultou nessa mudança?

_____

_____

2) Você considera que as informações históricas adicionais apresentadas nos gráficos são úteis para seleção de políticas? Por quê?

_____

_____

_____

Apêndice IX

Questionário de Avaliação

Realização das tarefas

1) Você ficou satisfeito com o resultado final das tarefas? Especifique, se necessário.	□ Sim □ Não □ Parcialmente

2) Você sentiu dificuldades na realização das tarefas? Especifique.	□ Sim □ Não □ Parcialmente

Informações históricas adicionais representadas nos gráficos

3) Você fez uso dessas informações adicionais para a sua seleção de políticas? De que forma?	□ Sim □ Não □ Parcialmente

4) O uso dessas informações adicionais aumentou sua confiabilidade na seleção de políticas?	□ Sim □ Não □ Parcialmente

5) Você considera importante a disponibilização dessas informações adicionais pelos sistema de controle de versão? Por quê?	□ Sim □ Não □ Parcialmente

6) Além de apoiar a escolha das políticas, você enxerga outros benefícios na utilização dessas informações adicionais? Especifique?	□ Sim □ Não □ Parcialmente

Controle de concorrência no desenvolvimento de software

7) Você tinha conhecimento das políticas de concorrência?	□ Sim □ Não □ Parcialmente

8) Você considera importante que os sistemas de controle de versão possibilitem a utilização de diferentes políticas para diferentes ICs? Por quê?	□ Sim □ Não □ Parcialmente

9) Você acredita que a escolha adequada das políticas para cada IC influencia de forma positiva a produtividade da equipe de desenvolvimento? Por quê?	□ Sim □ Não □ Parcialmente

Obrigado por sua colaboração!

Apêndice X

Formulário de Seleção de Políticas

Seleção de políticas de controle de concorrência

Tendo como referência as informações fornecidas do projeto, o histórico dos ICs e as informações históricas adicionais, faça a seleção de políticas de controle de concorrência que deve ser adotada por cada um desses ICs, preenchendo a tabela abaixo. Em seguida, responda as perguntas.

Obs.: Caso você considere que a escolha de política é indiferente, marque "X" nas duas colunas. Caso você considere que nenhuma das opções se aplica, não marque nenhuma das duas colunas e justifique sua resposta no campo de observações. Se necessário, utilize o campo de observações para comentários. O grau de confiança representa a sua confiança na sua escolha.

Grau de confiança: 0 = muito baixa, 1 = baixa, 2 = alta, 3 = muito alta

IC	Política pessimista	Política otimista	Grau de confiança				Observações
1			0	1	2	3	
2			0	1	2	3	
3			0	1	2	3	
4			0	1	2	3	
5			0	1	2	3	
6			0	1	2	3	

1) Quais critérios que você utilizou para chegar neste conjunto de políticas?

_____

_____

2) Você considera o histórico dos ICs uma informação importante para seleção de políticas de controle de concorrência?

_____

3) Você considera que as informações históricas adicionais dos ICs apresentadas nos gráficos são úteis para seleção de políticas? Por quê?

_____

_____

Apêndice XI

Formulário de Seleção de Políticas - Orion

Resultado da abordagem Orion

IC	Nível de concorrência	Dificuldade de junção	Sugestão Orion
1	Baixa	Alta	Política pessimista
2	Baixa	Baixa	Política indiferente
3	Alta	Alta	Reestruturação
4	Baixa	Alta	Política pessimista
5	Alta	Alta	Reestruturação
6	Alta	Baixa	Política otimista

Seleção de políticas de controle de concorrência

Responda as perguntas abaixo levando em consideração as seleções de políticas feitas por você na etapa anterior e a seleção feita pela abordagem Orion.

1) A seleção de políticas proposta pela Orion apresentou divergências comparada com a seleção feita por você na etapa anterior? Em caso afirmativo, diante do resultado da abordagem Orion, você faria alguma modificação na sua seleção de políticas? Quais e por quê?

_____

_____

_____

_____

_____

Apêndice XII

Questionário de Avaliação

Realização das tarefas

1) Você ficou satisfeito com o resultado final das tarefas? Especifique, se necessário.	□ Sim □ Não □ Parcialmente

2) Você sentiu dificuldades na realização das tarefas? Especifique.	□ Sim □ Não □ Parcialmente

Abordagem Orion

3) Você utilizaria uma ferramenta que implementasse o mecanismo de seleção de políticas da abordagem Orion? Por quê?	□ Sim □ Não □ Parcialmente

4) Liste aspectos positivos da abordagem.

5) Liste aspectos negativos da abordagem.

6) Você tem alguma sugestão para melhorar a abordagem? Especifique?	□ Sim □ Não □ Parcialmente

Controle de concorrência no desenvolvimento de software

7) Você tinha conhecimento das políticas de concorrência?	□ Sim □ Não □ Parcialmente

8) Você considera importante que os sistemas de controle de versão possibilitem a utilização de diferentes políticas para diferentes ICs? Por quê?	□ Sim □ Não □ Parcialmente

9) Você acredita que a escolha adequada das políticas para cada IC influencia de forma positiva a produtividade da equipe de desenvolvimento? Por quê?	□ Sim □ Não □ Parcialmente

Obrigado por sua colaboração!