

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



**ALINHAMENTO DE RASTROS DE EXECUÇÃO DE
PROGRAMAS PARA COMPREENSÃO DE PONTOS DE
VARIAÇÃO EM CÓDIGO-FONTE**

KLÉRISSE VINÍCIUS RIBEIRO DA PAIXÃO

Uberlândia - Minas Gerais

2009

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



KLÉRISSEON VINÍCIUS RIBEIRO DA PAIXÃO

ALINHAMENTO DE RASTROS DE EXECUÇÃO DE PROGRAMAS PARA COMPREENSÃO DE PONTOS DE VARIAÇÃO EM CÓDIGO-FONTE

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software.

Orientador:

Prof. Dr. Marcelo de Almeida Maia

Uberlândia, Minas Gerais

2009

Dados Internacionais de Catalogação na Publicação (CIP)

- P149a Paixão, Klérisson Vinícius Ribeiro da, 1983-
Alinhamento de rastros de execução de programas para compreensão
de pontos de variação em código-fonte / Klérisson Vinícius Ribeiro da
Paixão. - 2009.
125 f. : il.
- Orientador: Marcelo de Almeida Maia.
Dissertação (mestrado) – Universidade Federal de Uberlândia, Pro-
grama de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.
1. Engenharia de software - Teses. I. Maia, Marcelo de Almeida. II.
Universidade Federal de Uberlândia. Programa de Pós-Graduação em
Ciência da Computação. III. Título.

CDU: 681.3.06

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada “**Alinhamento de rastros de execução de programas para compreensão de pontos de variação em código-fonte**” por **Klérison Vinícius Ribeiro da Paixão** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 13 de Fevereiro de 2009

Orientador:

Prof. Dr. Marcelo de Almeida Maia
Universidade Federal de Uberlândia

Banca Examinadora:

Prof. Dr. Stéphane Julia
Universidade Federal de Uberlândia

Prof. Dr. Marco Túlio de Oliveira Valente
Pontifícia Universidade Católica de Minas Gerais

Dedicatória

Aos meus pais, obrigado!

Agradecimentos

A Deus e a Nossa Senhora por terem me acompanhado durante mais essa jornada.

Ao meu orientador, Marcelo Maia, com quem eu tanto aprendi sobre Engenharia de Software e pesquisa. Minha admiração pela sua seriedade e dedicação ao seu trabalho. Agradeço em especial pela paciência em vários momentos difíceis, palavras de incentivo e de crítica.

Aos meus pais, Adilson e Vanja, irmãos Andrey e Marcelo, pela amizade, palavras de sabedoria, amor e apoio constantes.

À minha namorada, Crícia, por compreender profundamente a difícil tarefa de trilhar o desenvolvimento de um trabalho de mestrado, me apoiando incondicionalmente com seu carinho, a qualquer momento em que eu necessitasse.

Aos amigos do programa de mestrado UFU/FACOM, cujo apoio e amizade me ajudaram muito nessa caminhada. Em especial aos parceiros diretos desta pesquisa, Felipe Cêzar, Victor Sobreira e Rodrigo Reis pela amizade.

Aos professores do programa de mestrado UFU/FACOM.

Aos professores Marco Túlio Valente e Stéphane Julia por terem aceitado participar da minha banca de mestrado.

Obrigado!

Resumo

É fato conhecido, que mudanças são inerentes a sistemas computacionais. Todo software de sucesso passa por contínuas evoluções seja para atender as novas expectativas dos usuários, mudanças de hardware e mudanças no ambiente operacional. Contudo, um dos problemas mais significativos nas atuais práticas de evolução de software é a dificuldade dos mantenedores em encontrar seções nos códigos fontes que precisam ser alteradas ou onde novas características precisam ser inseridas. Outro problema consiste nas refatorações produzidas durante o processo de mudança que também podem representar um desafio para a localização de conceitos entre diferentes versões. Neste trabalho é proposto um mecanismo para localização de elementos de código-fonte que servirão de base para inserção de novas funcionalidades em um sistema e para comparação de elementos de código de versões diferentes sob o ponto de vista do conceito que eles implementam. Especificamente, a abordagem proposta baseia-se no alinhamento de duas sequências de chamadas de métodos obtidas da execução de determinadas funcionalidades do sistema. Os pontos de desalinhamento mostram diferenças entre a implementação de duas funcionalidades similares ou a diferença de implementação de uma funcionalidade entre duas versões diferentes do sistema. O método é validado com a análise de resultados na ferramenta CASE ArgoUML. Esta abordagem mostrou-se útil para prover localização de potenciais artefatos do código-fonte que ajudem na compreensão do sistema diminuindo a quantidade de elementos que os desenvolvedores deverão analisar.

Palavras chave: alinhamento de sequências, compreensão de programas, análise dinâmica, engenharia reversa.

Abstract

The fact that requirement changes are inherent in software projects is well known. All successful software must continually evolve to support user requests, hardware changes and environment changes. However, discovering what are the software artifacts that need to evolve or where a new feature have to be inserted is a major challenge to Software Maintenance. In addition, discovering where the source code was refactored is another problem that hinders the location of concepts in source code. This work proposes a method to locate source code artifacts that will drive the evolution process in order to add new features in the system or to compare source code artifacts extracted from different versions of the same system in order to analyse where refactorings have taken place. The approach is based on the pairwise alignment of method call sequences obtained from the execution of predefined functionalities. The misaligned points show the differences between the implementation of two similar functionalities or the differences between the implementation of some functionality in two different versions of the system. The proposed method is validated with the analysis of the ArgoUML CASE tool. Specifically, the method have proven to be useful to locate potential source code artifact that helps the system comprehension with the reduction of the number of elements that the developer should analyse.

Keywords: sequence alignment, program comprehension, dynamic analysis, reverse engineering

Sumário

Lista de Figuras	x
Lista de Tabelas	xi
1 Introdução	12
1.1 Exposição do Problema	14
1.2 Hipóteses e Proposta	14
1.3 Objetivos	15
1.4 Justificativas	15
1.5 Visão Geral da Dissertação	15
2 Engenharia Reversa e Alinhamento de Seqüências	16
2.1 Engenharia Reversa	16
2.1.1 Análise Estática e Análise Dinâmica	17
2.2 Alinhamento de Seqüências	17
2.2.1 Definições	18
2.2.2 Alfabetos, Símbolos e Seqüências	18
2.2.3 Alinhamentos	18
2.2.4 Alinhamentos de Pares de Seqüências	19
2.2.5 Pontuações	20
2.2.6 Formalização do Problema	21
2.2.7 Algoritmos de Programação Dinâmica	21
2.2.8 Algoritmo Needleman e Wunsch	22
2.3 Alinhamento de Rastros de Execução	25
2.3.1 Características de Rastros de Execução	25
2.3.2 Alinhamento Local vs. Alinhamento Global	25
2.3.3 Alinhamento de Pares de Seqüência vs. Alinhamento de Múltiplas Seqüências	26
2.3.4 Alinhamentos Não Idênticos e Com Espaços	26
2.4 Conclusão	27

3	Abordagem Proposta para Compreensão de Pontos de Variação	28
3.1	Definição de Roteiros de Execução	28
3.2	Módulo de Extração de Rastros	29
3.2.1	Marcação do Rastro	30
3.3	Alinhamento de Rastros de Execução	31
3.4	Exemplo	32
3.4.1	Definição de Roteiros de Execução	33
3.4.2	Extração de Rastros	33
3.4.3	Alinhamento dos Rastros	35
3.5	Conclusão	38
4	Estudo de Caso: ArgoUML	39
4.1	<i>Goal Question Metric</i> (GQM)	39
4.2	Caracterização da Viabilidade da Abordagem Proposta	41
4.2.1	Planejamento do Estudo	42
4.2.2	Contexto 1: Mesma Versão, Funções Iguais	43
4.2.3	Contexto 2: Mesma Versão, Funções Similares	45
4.2.4	Contexto 3: Versões Diferentes, Funções Iguais	49
4.2.5	Contexto 4: Versões Diferentes, Funções Similares	53
4.3	Avaliação da Abordagem	55
5	Trabalhos Relacionados	57
6	Conclusões e Perspectivas	60
6.1	Contribuições	60
6.2	Limitações Identificadas	61
6.3	Trabalho Futuros	61
	Referências Bibliográficas	63
	Apêndice 1: Relatório de alinhamento dos rastros do Editor Gráfico	67

Lista de Figuras

2.1	Possível alinhamento entre TAGGTCA e TAGCTA.	19
2.2	Outro alinhamento entre TAGGTCA e TAGCTA.	20
2.3	Política de pontuação.	20
2.4	Matriz de pontuação e alinhamento.	23
2.5	Inicialização das Matrizes.	23
2.6	Cálculo de uma célula.	23
2.7	Matriz de substituição.	24
2.8	Célula $C(2, 2)$ preenchida.	24
2.9	Final da segunda etapa.	24
2.10	Início da dedução do alinhamento.	25
3.1	Abordagem proposta adaptado de [dAMSP ⁺ 08].	28
3.2	Marcação de um rastro.	30
3.3	Resultado do alinhamento.	32
3.4	Aplicação alvo.	32
4.1	Estrutura Hierárquica do GQM.	40
4.2	Exemplo modelo GQM [DdSK03].	41
4.3	Esquema dos estudos realizados.	42

Lista de Tabelas

2.1	Exemplo de alinhamentos entre 3 seqüências.	19
3.1	Exemplos de rastros	31
3.2	Métodos da <i>thread</i> 1 do roteiro de execução Desenhar Reta com seus <i>timestamps</i>	34
3.3	Métodos da <i>thread</i> 1 do roteiro de execução Desenhar Círculo com seus <i>timestamps</i>	35
3.4	Métodos e identificadores.	36
3.5	Segmentos entre os rastros dos Roteiro de execução 1 e 2.	37
3.6	Análise de desalinhamento: Segmentos 1 e 2.	37
3.7	Análise de desalinhamento: Segmentos 10 e 11.	38
3.8	Métodos sugeridos para implementação.	38
4.1	Sumário do Modelo GQM	40
4.2	Sumário do alinhamento do estudo de caso 1.	44
4.3	Sumário do alinhamento do estudo de caso 2.	46
4.4	Chamada de métodos do segmento 1, roteiro 2.	47
4.5	Análise de desalinhamento: Segmento 281.	47
4.6	Análise de desalinhamento: Segmento 689.	48
4.7	Métodos desalinhados do roteiro 1 no segmento 689.	48
4.8	Método desalinhado do roteiro 2 no segmento 689.	48
4.9	Sumário dos artefatos de código a serem estudados.	49
4.10	Sumário do alinhamento do estudo de caso 3.	50
4.11	Segmento 1, Diagrama de Classe, versão 0.28.	50
4.12	Segmento 1, Diagrama de Classe, versão 0.26.	51
4.13	Análise de desalinhamento: Segmento 187.	52
4.14	Análise de desalinhamento: Segmento 843.	52
4.15	Sumário dos artefatos de código a serem estudados.	53
4.16	Sumário do alinhamento do estudo de caso 4.	54

Capítulo 1

Introdução

É fato que mudanças são inerentes a sistemas computacionais [BR00]. Todo software de sucesso passa por contínuas evoluções seja para atender às novas expectativas dos usuários, mudanças de hardware, mudanças no ambiente operacional, etc.

Contudo, prover evoluções com facilidade, rapidez e confiabilidade ainda é um grande desafio para a engenharia de software, porque as aplicações estão cada vez mais complexas. Essa complexidade é consequência da necessidade de se atender a requisitos sofisticados, o que demanda abrangentes metodologias, por exemplo, novas notações de modelagem, linguagens de programação modernas, tecnologias distribuídas, componentes reusáveis, causando impacto direto nas tarefas de manutenção de software que por sua vez consistem na maior parte do processo de evolução.

Na literatura [BR00], manutenção de software é, comumente, classificada em categorias de acordo com a razão das alterações sugeridas. Abaixo, segue tal classificação:

Perfectiva Atender pedidos do usuário para modificar funções existentes, incluir novas funções e efetuar melhoramentos gerais;

Adaptativa Adaptar o software ao ambiente;

Corretiva Identificar e corrigir erros;

Preventiva Melhorar a manutenabilidade ou confiabilidade.

Importa ressaltar que a grande maioria das tarefas de manutenção têm sua origem em novas requisições do usuário, ou seja, tratam-se de tarefas de manutenção perfectiva [Ben96, BR00].

O processo de adicionar novas funções a um sistema já existente é denominado de mudança incremental¹. As atividades da mudança incremental são as seguintes [FR07]:

¹Do inglês *Incremental Change*.

1. Iniciação: Análise da mudança requerida e identificação de conceitos relevantes;
2. Localização de conceitos: Localização de conceitos no código fonte.
3. Análise de impacto: Identificação do conjunto completo de classes que serão afetadas pela mudança;
4. *Prefactoring*: Refatoração do programa para facilitar a mudança;
5. Atualização: Implementação dos conceitos e incorporação do novo código no sistema já existente;
6. Propagação da mudança: Correção das inconsistências causadas pela propagação da mudança;
7. *Postfactoring*: Refatoração para melhorar a estrutura do sistema;
8. Nova base: Entrega no novo código, atualização da documentação e entrega da nova versão aos clientes.

Logo na fase de *Localização de conceitos* encontra-se um dos problemas mais significativos nas atuais práticas de evolução de software que é a dificuldade dos mantenedores em encontrar seções nos códigos fontes que precisam ser alteradas ou onde novas características precisam ser inseridas [LYC00]. Várias abordagens já foram apresentadas para mitigar tal problema. Técnicas estáticas, incluindo casamento de padrão, análise de dependência entre classes e recuperação de informação são detalhadas em [MRB⁺05]. Técnicas dinâmicas foram exploradas em [WS95, SdAM08]. Contudo, como encontrar artefatos exatos para efetuar a mudança, ou mesmo pontos que direcionarão as mudanças, são questões ainda abertas.

As refatorações produzidas durante o processo de mudança também podem representar um desafio para a localização de conceitos entre diferentes versões, porque o código-fonte que implementa um conceito em uma versão pode ter sido alterado em uma versão seguinte, mesmo que não haja mudança de funcionalidade.

Outro fato que se observa na prática da engenharia de software é o uso recorrente de cópias de artefatos (“clones”) pelos programadores durante as tarefas de manutenção [KBLN04]. Quando se tem novas demandas para serem adicionadas a um sistema é comum encontrar funcionalidades, já existentes, semelhantes à requerida. Portanto, é prudente que a nova funcionalidade siga a mesma estrutura das demais até mesmo para facilitar futuras manutenções desde que se utilize técnicas para gerenciar os clones com eficiência [KSNM05].

Neste trabalho é proposto um mecanismo para localização de elementos de código-fonte que servirão de base para inserção de novas funcionalidades em um sistema e para comparação de elementos de código de versões diferentes sob o ponto de vista do conceito que eles implementam. Especificamente, trata-se de uma forma ágil para prover loca-

lização de potenciais artefatos do código-fonte que ajudem na compreensão do sistema diminuindo a quantidade de elementos que os desenvolvedores deverão analisar.

1.1 Exposição do Problema

O tema principal da pesquisa está na localização de potenciais artefatos do código-fonte que servirão de base para adição de novas funcionalidades em um sistema e na comparação de elementos de código de versões diferentes sob o ponto de vista do conceito que eles implementam. Diante disto, foram buscadas respostas para as seguintes perguntas:

1. Dadas funcionalidades que implementam conceitos semelhantes no sistema:
 - (a) Como inserir uma nova funcionalidade que tenha arquitetura e implemente conceitos semelhantes de funcionalidades já existentes no sistema?
 - (b) Qual a seqüência de eventos que ocorrem nas execuções de funcionalidades similares que podem contribuir na integração de uma nova funcionalidade no sistema?
 - (c) Quais as variabilidades na execução de funções similares de um sistema?
 - (d) Quais seqüência de eventos são comuns a determinadas funcionalidades?
2. Dadas versões diferentes de um sistema:
 - (a) Como identificar a ocorrência de evolução em uma dada funcionalidade de um sistema?
 - (b) Como identificar mudanças de implementação ou arquiteturais de uma funcionalidade através de rastros de execuções?

1.2 Hipóteses e Proposta

Quando considera-se a adição de novas funcionalidades a um sistema, é possível que a nova funcionalidade tenha arquitetura semelhante a alguma pré-existente no sistema.

A hipótese deste trabalho é que, identificando artefatos semelhantes existentes em um sistema, a inserção de uma nova funcionalidade semelhante a alguma existente possa ter seu tempo e esforço reduzido já que se saberá onde será inserido um novo artefato e até como será sua forma inicial.

Além disso, identificar como e se foram feitas evoluções em versões diferentes de um sistema é um desafio para os desenvolvedores, sobremaneira quando não se tem documentação.

Logo, outra hipótese deste trabalho é que, identificando variações em código-fonte, pode-se inferir quais mudanças foram necessárias para concretizar uma evolução de uma funcionalidade.

A proposta é fundamentada no uso de técnica de alinhamento de seqüências. As seqüência são rastros de execução de software definidas por roteiros de execução.

A principal contribuição deste trabalho está em mostrar potenciais elementos de código que podem direcionar o desenvolvimento de uma nova funcionalidade, bem como a seqüência de eventos que deverão ocorrer para que a nova funcionalidade seja incorporada. Além disso, outra contribuição está na identificação de como foram feitas evoluções em uma dada funcionalidade.

1.3 Objetivos

O objetivo deste trabalho é aplicar técnica de alinhamento de rastros de execução de um programa para extrair informação do código-fonte que direcionem a compreensão de sistemas de forma a agilizar tarefas de um processo de manutenção de software.

1.4 Justificativas

As práticas de evolução de sistemas enfrentam uma demanda cada vez maior por software de qualidade e entregue em menor tempo. Nesse contexto, a compreensão do que se tem a fazer para evoluir um sistema tem custos que variam de 40% a 60% do custo total de um sistema [ABDM01].

Neste contexto, a reutilização de software representa uma área da Engenharia de Software que vem ganhando força. Ela pode ser definida como o processo de criação de sistemas de software a partir de software preexistente, trazendo consigo a promessa de aumentar a produtividade e diminuir custos do desenvolvimento, além de melhorar a qualidade do produto, através da reutilização de artefatos de software já testados e utilizados em outros contextos.

Portanto, a identificação de artefatos potenciais que direcionam e, por conseqüência, reduzam os custos dos processos de compreensão de software para prover evolução tornam-se valiosos.

1.5 Visão Geral da Dissertação

Este capítulo apresentou a motivação, proposta e hipóteses da abordagem. O capítulo 2 apresenta uma fundamentação teórica da proposta. O capítulo 3 apresenta a arquitetura da abordagem e um pequeno exemplo do mecanismo em funcionamento. O capítulo 4 descreve um estudo de caso para mostrar a viabilidade da abordagem em um sistema real. Por fim, o capítulo 5 apresenta alguns trabalho relacionados e o capítulo 6 apresenta as conclusões finais.

Capítulo 2

Engenharia Reversa e Alinhamento de Seqüências

Neste capítulo será apresentada uma revisão na literatura sobre os principais temas envolvidos na pesquisa, dentre eles, Engenharia Reversa (Seção 2.1), Análise Dinâmica (Seção 2.1.1) e Alinhamento de Seqüências (Seção 2.2).

2.1 Engenharia Reversa

A Engenharia Reversa (ER) pode ser definida como: “o processo de análise de um sistema, a fim de identificar os componentes desse sistema e seus relacionamentos e criar representações do sistema em uma outra forma ou em um nível mais alto de abstração” [CCI90].

A ER, normalmente, objetiva a extração de documentação de projeto a partir do código, representando uma atividade ou um conjunto de técnicas que visam facilitar a compreensão de um software existente [Vas07]. Neste contexto existem vários trabalhos envolvendo diferentes objetivos, como: redocumentação de software, recuperação de projeto, remodularização, recuperação de arquitetura, localização de funcionalidades no código, extração de componentes.

No presente trabalho, foi explorada a atividade de localização de funcionalidades no código. Essa atividade tem fundamental importância no processo de manutenção de software uma vez que o pré-requisito para tal atividade é o entendimento do sistema e esse entendimento está diretamente ligado ao entendimento das funcionalidades no código.

O primeiro passo para localizar funcionalidades consiste na análise dos artefatos do código. Dois tipos de análises são comumente aplicados: estática e dinâmica.

2.1.1 Análise Estática e Análise Dinâmica

A fim de identificar funcionalidades de um código tanto a análise estática como a dinâmica podem ser desempenhadas. Se por um lado a análise estática pode extrair informações básicas do software, como, por exemplo, dependência de acesso a variáveis e grafos de chamadas de procedimentos, por outro lado a análise dinâmica consiste na análise de propriedades de uma programa em execução, informando, por exemplo, as sequências de chamadas de funções, os valores de variáveis, etc.

Para se descobrir a dependência de chamadas de funções/métodos a análise estática é suficiente, mas saber quais objeto de código realmente utiliza, pode depender dos dados de entrada utilizados ou caminho escolhido para executar o programa, sendo necessária a análise dinâmica. A análise dinâmica é menos geral do que a estática, no sentido de que ela fornece informações sobre a chamada do programa com um conjunto de dados de entrada particular. Um mesmo programa com outros diferentes dados de entrada pode fornecer outras informações [ANQ00].

Neste trabalho, somente a análise dinâmica foi explorada por permitir relacionar as variações de entrada, relativas a uma determinada funcionalidade, aos acontecimentos internos dos programas [Sob].

2.2 Alinhamento de Sequências

A Biologia Molecular é uma ciência que apresentou avanços significativos nas últimas décadas. Um grande desafio encontrado pelo biólogos é a necessidade de manipular uma grande quantidade de informações geradas a partir de seus experimentos. Assim, surgiu a Bioinformática que aplica técnicas computacionais, matemáticas e estatísticas para tratar seus problemas. Um dos problemas principais neste contexto é a comparação de sequências.

A técnica mais freqüentemente usada para comparar duas ou mais sequências de nucleotídeos ou aminoácidos consiste em sobrepor uma cadeia sobre outra e buscar semelhanças e diferenças, a qual é formalmente denominada alinhamento.

Algumas informações obtidas a partir da comparação de sequências são apontadas por [OLP01]:

1. Determinação da função biológica mediante homologia de sequências;
2. Busca de padrões conservados ao longo da evolução que servem para identificar estruturas conservadas, resíduos funcionais que podem descrever uma família ou sub-família de proteínas;
3. Organização dos domínios dentro de uma família protéica;

4. Construção da estrutura molecular a partir do alinhamento de nucleotídeos com uma proteína de estrutura conhecida e, por consequência, determinar sua função biológica;
5. Estudos evolutivos para definir relações filogenéticas entre seqüências.

As primeiras abordagens para o problema usavam técnicas de programação dinâmica. Posteriormente, surgiram modelos heurísticos e probabilísticos. Técnicas de aprendizado de máquinas como redes neurais e algoritmos evolutivos foram também desenvolvidas.

2.2.1 Definições

Nesta seção apresenta-se as definições para o problema de alinhamento de seqüências. O conteúdo desta seção está baseado em [dB03].

2.2.2 Alfabetos, Símbolos e Seqüências

Um alfabeto Σ é um conjunto finito e não-vazio. Um elemento $\sigma \in \Sigma$ é chamado caractere ou símbolo. Uma *seqüência* s sobre o alfabeto Σ é uma seqüência $s = (\sigma_1, \sigma_2, \dots, \sigma_n) \in \Sigma^n$, onde $n \geq 0$ é um inteiro. Se $s = \sigma_1, \sigma_2, \dots, \sigma_n$ define-se o comprimento $|s|$ de s por $|s| = n$. Se $n = 0$ (e, portanto, $|s| = 0$), então s é chamada *seqüência vazia* e é denotada por ε . Se $n \geq 0$ é um inteiro e $\sigma \in \Sigma$, então denota-se por σ^n a seqüência de comprimento n cujos caracteres são todos iguais a σ . O *conjunto de todas as seqüências* sobre o alfabeto Σ é denotado por Σ^* , enquanto o *conjunto de todas as seqüências não-vazias* sobre Σ é denotado por Σ^+ .

Um *segmento* $s[i..j]$ de uma seqüência s , para $i \leq j$, é a seqüência definida por $s[i..j] = \sigma_i, \sigma_{i+1}, \dots, \sigma_j$. Se $i > j$, define-se $s[i..j] = \varepsilon$.

Uma *subseqüência* s' de s é uma seqüência $s' = \sigma_{i_1}, \sigma_{i_2}, \dots, \sigma_{i_m}$ em que $i_1, i_2, \dots, i_m \subseteq 1, 2, \dots, n$ e $i_1 < i_2 < \dots < i_m$. Nesse caso, também tem-se que s é uma superseqüência de s' . A diferença entre um segmento e uma subseqüência de s é que o primeiro se constitui de uma série de caracteres consecutivos de s , por outro lado, uma subseqüência pode ser constituída de caracteres que não necessariamente estejam consecutivos em s .

Se Σ é um alfabeto tal que $\sqcup \notin \Sigma$, denotamos por Σ' o alfabeto $\Sigma' = \Sigma \cup \sqcup$. O símbolo $\sqcup$ é tratado como especial e é chamado caractere branco ou espaço.

2.2.3 Alinhamentos

Sejam k um inteiro positivo e $s_1, s_2, \dots, s_k$ seqüências sobre um alfabeto Σ , com $|s_i| = n_i$, para $i = 1, \dots, k$. Um alinhamento A de $s_1, s_2, \dots, s_k$ é uma matriz $A = (A_{ij})$ de dimensões $k \times n$ com entradas de $\Sigma' = \Sigma \cup \sqcup$ (e $n \geq \max_{i=1}^k \{n_i\}$) tal que a linha A_i do alinhamento A consista exatamente da seqüência s_i com possíveis espaços (e apenas espaços) inseridos entre os caracteres de s_i .

Para ficar mais claro, pode-se reescrever a definição acima como: um alinhamento A de $s_1, s_2, \dots, s_k$ é uma matriz $A = (A_{ij})$ de dimensões $k \times n$ com entradas de Σ' tal que, para cada i , existe um conjunto $J_i = \{j_1, j_2, \dots, j_{n_i}\} \subseteq \{1, 2, \dots, n\}$, com $j_1 < j_2 < \dots < j_{n_i}$ e tal que $A_{ij_1}A_{ij_2}\dots A_{ij_{n_i}} = s_i$ e tal que para todo $j \in \{1, 2, \dots, n\} - J_i$, temos $A_{ij} = \sqcup$.

Dizemos que dois caracteres $s_i[j]$ e $s_{i'}[j']$ estão alinhados em A se $s_i[j]$ e $s_{i'}[j']$ estão na mesma coluna de A .

Seja $A = (A_{ij})$ um alinhamento de $s_1, s_2, \dots, s_k$. Um espaço¹ de s_i no alinhamento A é qualquer segmento não-vazio maximal de A_i composto apenas de espaços, isto é, um espaço de s_i em A é um segmento $A_i[j..j']$ de A_i tal que:

- $j \leq j'$;
- todos os símbolos de $A_i[j..j']$ são $\sqcup$;
- não seja possível estender $A_i[j..j']$ em A_i por meio de espaços em branco, ou seja, $j = 1$ ou $A_i[j - 1] \neq \sqcup$ e $j' = n$ ou $A_i[j' + 1] \neq \sqcup$.

Sejam $s_1, s_2, \dots, s_k$ seqüências sobre um alfabeto Σ . Denota-se por $A_{s_1, s_2, \dots, s_k}$ o conjunto de alinhamentos entre as seqüências $s_1, s_2, \dots, s_k$. Na Tabela 2.1 apresenta-se um pequeno exemplo ilustrativo dos conceitos apresentados.

A	B	$\sqcup$	$\sqcup$	$\sqcup$
A	B	C	D	E
A	$\sqcup$	C	$\sqcup$	E

Tabela 2.1: Exemplo de alinhamentos entre 3 seqüências.

2.2.4 Alinhamentos de Pares de Seqüências

Alinhamentos são uma maneira de realizar comparações entre seqüências. Na Figura 2.1 apresentamos um possível alinhamento entre as seqüências de caracteres TAGGTCA e TAGCTA.

TAGGTCA $\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup$
 $\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup$ TAGCTA

Figura 2.1: Possível alinhamento entre TAGGTCA e TAGCTA.

Embora o alinhamento acima não demonstre que as duas seqüências são parecidas temos outro possível alinhamento entre as mesmas seqüências mostrado na Figura 2.2 onde fica evidente que as seqüências tem muito em comum:

As únicas diferenças são um caracter G na primeira seqüência ocupando o lugar de um caracter C na segunda e um caracter C que está presente primeira seqüência, mas que

¹Espaço, do inglês: *gap*

TAGGTCA
TAGCT□A

Figura 2.2: Outro alinhamento entre TAGGTCA e TAGCTA.

não está presente na segunda. Além disso, percebe-se que temos um segmento de três caracteres iniciais que são iguais nas duas seqüências.

A busca por evidências que exponham as similaridades e diferenças entre as seqüências é o que interessa às tarefas de alinhamento. Por isso, o alinhamento da Figura 2.2 parece melhor, uma vez que permite uma fácil comparação entre as seqüências.

Logo para decidir qual o alinhamento é o melhor, é necessário estabelecer um conceito de “pontuação” para cada alinhamento. Tal conceito é explorado na seção seguinte.

2.2.5 Pontuações

Tradicionalmente, o problema de alinhamento de seqüências é tratado como um problema de otimização com objetivo simples. Para cada par de seqüências, o alinhamento que evidenciar a maior quantidade de semelhanças será o melhor. Uma maneira usual de atribuir pontuações a alinhamentos de duas seqüências é dada pela estratégia de, a cada coluna de um alinhamento, associar uma pontuação e definir a pontuação do alinhamento como sendo a soma das pontuações de suas colunas.

Por exemplo, na Figura 2.3 apresenta-se uma possível política de pontuação:

C □ T T A G
C G A T A G

C x C = 1	T x T = 1
- x G = -2	A x A = 1
T x A = -1	G x G = 1

Total = 1

Figura 2.3: Política de pontuação.

Se uma coluna apresentar duas bases iguais, recebe pontuação 1. Se apresentar bases diferentes, recebe pontuação -1. Finalmente, se essa coluna tiver uma base e um espaço, ela recebe a pontuação -2. Assim, o alinhamento ótimo será aquele que tiver pontuação mais alta que é definida pela similaridades entre as seqüências [Gus97]. Outras políticas de pontuação podem ser encontradas na literatura, contudo foram feitas para atender objetivos específicos da biologia.

2.2.6 Formalização do Problema

A pontuação de um alinhamento é definida como a soma das pontuações de suas colunas. Intuitivamente, se o esquema de pontuação atribui “pontuação baixa” a caracteres “semelhantes” de um alfabeto e “pontuação alta” a caracteres “diferentes” de mesmo alfabeto, então os alinhamentos de maior interesse são aqueles de “pequena pontuação”.

Problema 1.1 (Alinhamento de Pares de seqüências (APS)) Dadas duas seqüências s e t sobre um alfabeto Σ e fixado um esquema de pontuação c , encontrar um alinhamento A de s e t cuja pontuação $c(A)$ seja mínima.

Na próxima seção, apresenta-se dois algoritmos fundamentais, baseados no *paradigma de programação dinâmica*, que podem ser usados para resolver o problema de alinhamento de pares de seqüências.

2.2.7 Algoritmos de Programação Dinâmica

Um algoritmo baseado no paradigma de programação dinâmica opera resolvendo progressivamente subproblemas do problema original (geralmente, em ordem crescente de tamanho dos subproblemas) e armazenando as soluções de cada subproblema em uma tabela para um possível uso posterior. A idéia é evitar que algum subproblema seja resolvido (desnecessariamente) mais de uma vez. Outra forma de entendimento do paradigma é que ele consiste em uma maneira de executar os passos de um algoritmo recursivo em uma ordem conveniente, usando uma tabela para manter as soluções dos subproblemas já resolvidos por chamadas recursivas do algoritmo.

Um algoritmo de programação dinâmica para o Problema APS foi publicado em 1970 por Needleman e Wunsch [NW70], e foi considerado como a primeira contribuição importante em procedimentos computacionais para a comparação de seqüências. O algoritmo, muito modificado desde então por vários pesquisadores, foi melhorado em relação à complexidade de tempo e de espaço, como também foi adaptado para outros problemas. Importa destacar uma das adaptações mais importantes e populares desenvolvida por Smith e Waterman em 1981, para computar alinhamentos locais [SW81].

A saber, na literatura encontram-se basicamente duas formas diferentes de alinhamentos que são apresentados a seguir:

Alinhamento Global Para o Alinhamento Global, o total das posições de um alinhamento (s, t) é considerado. Por exemplo:

Dadas as seqüências:
 $s = G A A G G A T T A G$
 $t = G A T C G G A A G$
 Tem-se o seguinte alinhamento:
 $G A A - G G A T T A G$
 $G A T C G G A - - A G$

Alinhamento Local Um alinhamento local entre duas seqüências s e t acontece entre um segmento de s e um segmento de t . A pontuação é calculada apenas sobre a região onde os segmentos estão alinhados:

Dadas as seqüências:
 $s = A A G A C G G$
 $t = G A T C G A A G$
 Tem-se o seguinte alinhamento:
 $- - - - - A A G A C G G$
 $G A T C G G A A G - - - -$

Na seção seguinte detalha-se o algoritmo para alinhamento global Needleman e Wunsch. Esse algoritmo foi utilizado no desenvolvimento deste trabalho.

2.2.8 Algoritmo Needleman e Wunsch

O algoritmo de alinhamento de seqüências proposto por Needleman e Wunsch é utilizado como algoritmo de programação dinâmica para alinhamentos globais. Esse algoritmo foi aplicado ao problema de alinhamento de seqüências de proteínas [NW70].

O processo de alinhamento proposto é composto de três passos:

1. Inicializar uma matriz de pontuação;
2. Calcular as pontuações e preencher uma matriz de alinhamento;
3. Deduzir o alinhamento a partir da matriz de alinhamento;

Abaixo apresenta-se um pequeno exemplo dos passos do algoritmo. Importa ressaltar que no exemplo foram utilizados os conceitos de penalizações determinadas na matriz BLOSUM62 apresentada mais adiante, tal matriz trata-se de uma probabilidade específica para o alinhamento de seqüências de DNA, o que significa que podem ocorrer alinhamentos entre elementos diferentes. Outra abordagem que pode ser utilizada é que só haja alinhamentos entre elementos idênticos.

Suponha que se deva alinhar as seqüências $s = SEND$ e $t = AND$. Duas matrizes são criadas, matriz de alinhamento e matriz de pontuação como mostrado na Figura 2.4:

A inicialização da matriz de pontuação segue uma política de penalizações por espaços colocados no alinhamento. Neste exemplo, temos as penalizações variando de 0 até -40

		S	E	N	D	
		C(1,1)	C(1,2)	C(1,3)	C(1,4)	C(1,5)
A		C(2,1)	C(2,2)	C(2,3)	C(2,4)	C(2,5)
N		C(3,1)	C(3,2)	C(3,3)	C(3,4)	C(3,5)
D		C(4,1)	C(4,2)	C(4,3)	C(4,4)	C(4,5)

		S	E	N	D

Figura 2.4: Matriz de pontuação e alinhamento.

na linha e 0 até -30 na coluna, como mostrado na figura abaixo. A matriz de alinhamento é inicializada conforme com as indicações de direção mostradas na Figura 2.5:

	S	E	N	D	
	0	-10	-20	-30	-40
A	-10				
N	-20				
D	-30				

	S	E	N	D	
	done	left	left	left	left
	up				
	up				
	up				

Figura 2.5: Inicialização das Matrizes.

Na segunda etapa, a matriz de pontuação é preenchida começando da célula $C(2,2)$. A pontuação de qualquer célula $C(i,j)$ é dada pelo máximo de:

$$\max \begin{pmatrix} qdiag = C(i-1, j-1) + S(i, j) \\ qup = C(i-1, j) + g \\ qlft = C(i, j-1) + g \end{pmatrix}$$

Onde $S(i,j)$ é a pontuação da substituição de i por j , neste exemplo foi utilizado a matriz de substituição BLOSUM62, Figura 2.7, onde cada caracter i quando for substituído por j tem uma pontuação. E g é a penalização por inserção de espaços. Uma representação de como é calculada uma célula $C(i,j)$ é apresentado na Figura 2.6:

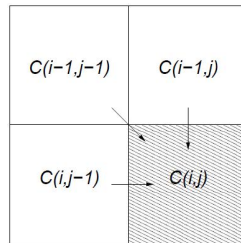


Figura 2.6: Cálculo de uma célula.

Então para a célula $C(2,2)$ teria-se:

$$\begin{aligned} qdiag &= C(1,1) + S(S, A) = 0 + 1 = 1 \\ qup &= C(1,2) + g = -10 + (-10) = -20 \\ qlft &= C(2,1) + g = -10 + (-10) = -20 \end{aligned}$$

	A	R	N	D	C	O	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
O	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	-3	-3
I	-1	-3	-3	-1	-3	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	-3	-2	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-3	-3	-3	-1	0	0	-3	-3	0	6	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2
S	0	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
T	0	0	0	-1	-1	-1	-1	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	7	-3
Y	-2	-2	-3	-2	-1	-2	-3	-2	-1	-2	-1	-3	-3	-2	-2	-2	-2	7	-1	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Figura 2.7: Matriz de substituição.

	S	E	N	D		S	E	N	D	
A	0	-10	-20	-30	-40	done	left	left	left	left
N	-10	1				up	diag			
D	-20					up				
	-30					up				

Figura 2.8: Célula $C(2, 2)$ preenchida.

O processo é feito para cada célula das matrizes, Figura 2.9.

		S	E	N	D			S	E	N	D	
A		0	-10	-20	-30	-40		done	left	left	left	left
		-10	1	-9	-19	-29		up	diag	left	left	left
N		-20	-9	-1	-3	-13		up	diag	diag	diag	left
D		-30	-19	-11	2	3		up	up	diag	diag	diag

Figura 2.9: Final da segunda etapa.

A última etapa consiste em percorrer a matriz de alinhamento começando pela célula mais a esquerda e mais abaixo da matriz, ou seja, pela última célula da matriz, Figura 2.10:

Para cada direção apontada na matriz de alinhamento uma ação é tomada conforme abaixo:

$$\left\{ \begin{array}{ll} \text{diag} = & \text{Os elementos são alinhados;} \\ \text{left} = & \text{Um espaço é introduzido na sequência da coluna;} \\ \text{up} = & \text{Um espaço é introduzido na sequência da linha.} \end{array} \right.$$

Passo 1	Passo 2	Passo 3	Passo 4
D	ND	END	SEND
D	ND	-ND	A-ND
diag	diag	left	Diag

		S	E	N	D
	done	left	left	left	left
A	up	diag	left	left	left
N	up	diag	diag	diag	left
D	up	up	diag	diag	diag

Figura 2.10: Início da dedução do alinhamento.

Enfim, o alinhamento entre s e t seria:

SEND
A-ND

2.3 Alinhamento de Rastros de Execução

Durante a execução de um sistema, um extrator de rastros intercepta as chamadas de métodos e produz um arquivo texto com a seqüência de métodos chamados para cada *thread* iniciada. Cada linha do arquivo corresponde a uma chamada de método que contém o nome do método executado e o tempo em que ocorreu (*timestamp*). As entradas para o algoritmo de alinhamento serão esses arquivos contendo a seqüência de métodos executados.

2.3.1 Características de Rastros de Execução

Os rastros de execução podem apresentar ruídos. Por exemplo, considerando as seqüências “XaaaaY” e “XaaaY”, pode-se suspeitar que tais seqüências foram geradas pelo mesmo código. A diferença pode ter sido causada por um *loop* que executou o método “a” por 4 vezes na primeira seqüência e 3 na outra. Em outro exemplo, considerando as seqüências “XaaaaY” and “XaabaY”, é possível que a chamada do método “b” foi ocasionada por uma cláusula condicional. Em ambos exemplos, os rastros podem representar a mesma funcionalidade e, no entanto, podem apresentar seqüências diferentes.

2.3.2 Alinhamento Local vs. Alinhamento Global

Para decidir qual tipo de alinhamento utilizar em um processo de alinhamento de seqüências deverá ser considerado o resultado que deseja obter. O alinhamento global é feito comparando elemento por elemento das seqüências. Se o objetivo do alinhamento é comparar seqüências com um alto grau de similaridade e com tamanhos semelhantes, o alinhamento global é o mais indicado.

O alinhamento de seqüências do tipo local é mais utilizado para os casos onde o propósito do alinhamento é encontrar uma seqüência menor dentro de uma seqüência

maior. Nesse caso a comparação é feita por fragmentos da sequência e não elemento por elemento.

Um alinhamento local de s e t é um alinhamento global de qualquer fragmento de s e qualquer fragmento de t . Um alinhamento local ótimo é um alinhamento com pontuação máxima entre todos os alinhamento de s e t .

Neste trabalho, o tipo de alinhamento escolhido foi o alinhamento global, que será utilizado para comparar sequências que possuem um alto grau de similaridade e com tamanhos semelhantes pois objetivo é encontrar características semelhantes nas sequências.

2.3.3 Alinhamento de Pares de Sequência vs. Alinhamento de Múltiplas Sequências

O alinhamento de pares de sequências é utilizado em um alinhamento global ou local, de duas sequências. No alinhamento de várias sequências é necessário fazer uma integração de técnicas já que a comparação só pode ser feita com duas sequências por vez. Os algoritmos de alinhamento de múltiplas sequências são uma generalização do alinhamento de pares, e aceitam como dados de entrada múltiplas sequências.

Os algoritmos de alinhamento de múltiplas sequências são mais indicados para sequências pequenas, pois este problema é NP-completo.

Como nesse trabalho será feito o alinhamento de rastros de execução de software, as sequências que serão comparadas possuem um tamanho grande e por essa razão o alinhamento de pares foi o escolhido.

2.3.4 Alinhamentos Não Idênticos e Com Espaços

O alinhamento de pares de sequências pode ser feito considerando políticas de substituição entre os elementos ou não.

No primeiro caso uma matriz de substituição deve ser definida, assim como mostrado na Tabela 2.7, e as pontuações dos alinhamentos são calculadas levando em consideração os valores presentes na matriz. Esse esquema de alinhamento é pertinente quando se conhece as relações entre os elementos em análise e pode-se quantificar quanto um elemento é semelhante a outro. Por outro lado, o alinhamento pode ser feito somente entre elementos idênticos, ou seja, se um elemento em análise é diferente de outro ocorre um desalinhamento.

Neste trabalho, optou-se pelo alinhamento de elementos idênticos, assim para qualquer desalinhamento que ocorrer, será inserido um espaço, ou seja, não existe penalidade para inserção de espaço.

2.4 Conclusão

Neste capítulo apresentou-se os conceitos das áreas que se insere este trabalho, ressaltando as definições sobre alinhamento de seqüências que serão utilizadas no decorrer do trabalho. No próximo capítulo é apresentada a abordagem proposta para compreensão de pontos de variação em alinhamento de rastros de execução.

Capítulo 3

Abordagem Proposta para Compreensão de Pontos de Variação

Neste capítulo será apresentada a abordagem proposta neste trabalho para compreensão de pontos de variação em código-fonte. Depois de detalhada a abordagem, segue um exemplo em um sistema de pequeno porte.

Na Figura 3.1 é apresentada uma visão geral da abordagem proposta usando um diagrama de atividade *UML*. As atividades da abordagem são detalhadas durante esta seção. Por fim, é apresentado um exemplo ilustrativo da abordagem.



Figura 3.1: Abordagem proposta adaptado de [dAMSP⁺08].

3.1 Definição de Roteiros de Execução

A primeira atividade consiste em definir os roteiros que são similares à nova característica requisitada. É uma atividade manual e tem como entrada as questões relativas ao processo de compreensão, bem como a documentação disponível sobre o sistema. Os resultados serão os roteiros de execução das características exercitadas em uma determinada versão do software.

Importa ressaltar que essa definição é de extrema importância para o processo, uma vez que o alinhamento de rastro é bem sensível às entradas definidas. Caso sejam definidas características a serem exercitadas sem alguma similaridade entre elas, pode não haver alinhamento algum, não contribuindo em nada para o processo de manutenção.

Outro ponto importante é a utilização da documentação, caso exista. A descrição de roteiros já implementados na aplicação alvo deve servir de base para determinar se existe alguma compatibilidade com o problema de compreensão em questão. Além disso, a documentação poderá ser adaptada para contemplar as questões de compreensão.

3.2 Módulo de Extração de Rastros

A técnica de extração utilizada neste trabalho para a extração de rastro está descrita em [Sob]. Trata-se de um processo apoiado por uma ferramenta de instrumentalização de sistema. Segue uma breve descrição das etapas:

Planejamento da Execução Refere-se à definição dos roteiros de execução e à definição da forma de extração dos rastros.

Definição dos Roteiros de Execução Trata-se de uma especificação dos passos a serem seguidos para execução do programa a ser analisado.

Definição da forma de Extração dos Rastros Refere-se ao número de rastros de execução extraídos e tratados. Na literatura existem abordagens que lidam com um único rastro [RHLR08], bem como outras abordagens extraem rastros distintos para cada característica [Eis05]. Neste trabalho optamos pela extração de um único rastro para representar nossa sequência.

Mecanismo de Coleta Define o mecanismo utilizado para coletar os dados durante a execução do programa. Neste trabalho, optou-se pela instrumentação baseada em *Aspectos*, por ser menos intrusiva, ou seja, causa menor alteração no comportamento original do sistema.

Compressão de Rastros Objetiva simplificar os rastros coletados, focada especialmente na remoção de eventos redundantes tais como chamadas em laços de execução, chamadas recursivas e seqüências de chamadas que se repetem no rastro [HLL02]. Tais redundâncias ocasionam desalinhamentos nos rastros sem importância no processo de compreensão de sistemas. Portanto é importante que tais redundâncias sejam retiradas para que não sejam produzidas informações irrelevantes para o desenvolvedor.

Ao final do processo de extração tem-se um rastro para cada *thread* disparada durante a execução dos roteiros que são seqüências de chamadas de métodos com seus respecti-

vos tempos de ocorrência (*timestamp*) obtidos do relógio do sistema, conforme mostrado abaixo:

Rastros: métodos e <i>timestamps</i> .		
graphicaleditor.GraphicalEditor	main	1232677520346
graphicaleditor.GraphicalEditor	<init>	1232677520362
graphicaleditor.MainFrame	<init>	1232677520658
. . .		

3.2.1 Marcação do Rastro

Ao executar os roteiros de execução o rastro pode ser marcado, paralelamente à execução, pelo usuário com o objetivo de identificar as características que estão sendo executadas no momento. A marcação consiste em notificar o ponto inicial na sequência de métodos e o final nos quais uma determinada característica ocorreu.

Na Figura 3.2 é mostrado um exemplo da marcação do rastro, onde as características *inicialização*, *desenhar reta* e *desenhar círculo* foram marcadas. A tela *Graphical Editor* representa a aplicação alvo e a tela *Trace Mark Control* representa o módulo interativo para marcação do início e fim da execução das características.

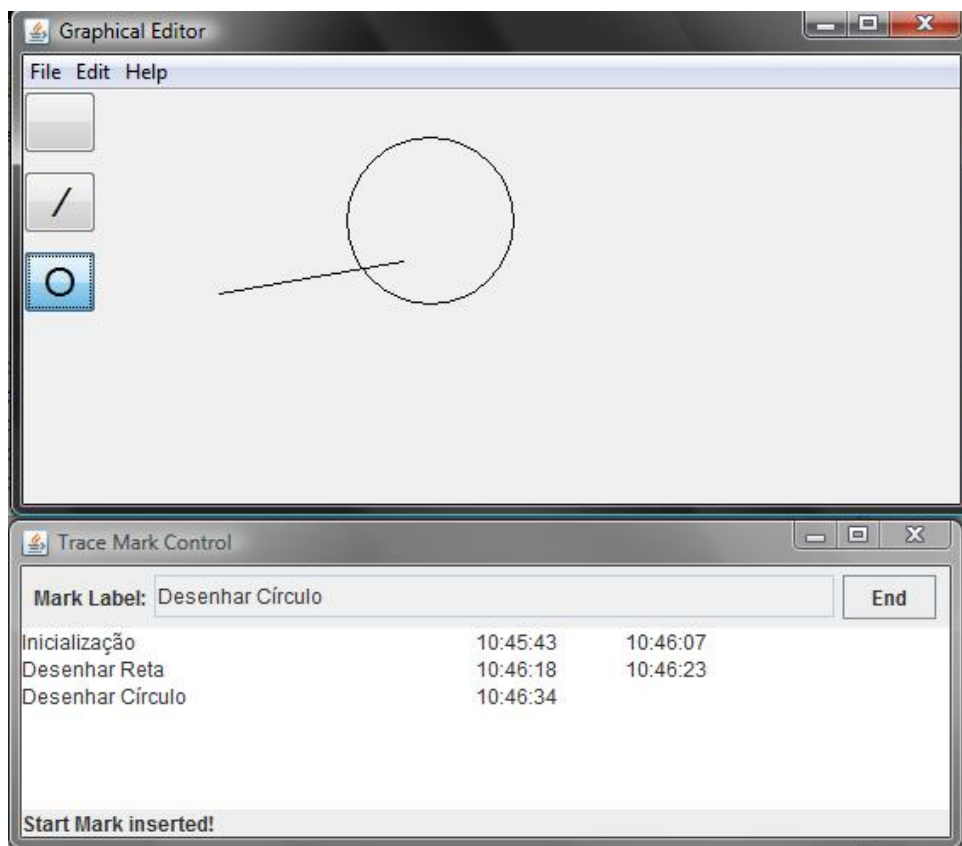


Figura 3.2: Marcação de um rastro.

Ao final do processo teremos um arquivo com as características apontadas e um intervalo de tempo que ocorreram:

Característica	Início (<i>timestamp</i>)	Fim (<i>timestamp</i>)
Inicialização	1233751543346	1233751567527
Desenhar Reta	1233751578541	1233751583659
Desenhar Círculo	1233751594797	...

3.3 Alinhamento de Rastros de Execução

Finalmente, o módulo de alinhamento de rastros é responsável por produzir os relatórios que ajudarão na compreensão desejada, seja para implementar uma nova funcionalidade ou para análise de implementações das características exercitadas. Tem como entradas dois rastros distintos de funcionalidades existentes. Na Tabela 3.1 temos dois rastros (*rastro1* e *rastro2*) com seus respectivos métodos identificados por *a, b, c, d* e *e*.

Rastros	Métodos					
Rastro1	a	b	b	c	c	e
Rastro2	a	b	c	c	d	e

Tabela 3.1: Exemplos de rastros

Ao final do alinhamento, são produzidas duas novas seqüências de mesmo tamanho referentes ao rastro com espaços inseridos nas mesmas de tal forma que métodos alinhados aparecerão na mesma posição nas duas seqüências. Estes dois rastros podem ser quebrados em segmentos que estão na mesma posição em cada rastro. As segmentações do rastro são determinadas levando em consideração se os métodos estão alinhados ou não. Na Figura 3.3 temos o primeiro segmento (*segmento1*) composto pela seqüência de métodos (*a, b,*) tanto para o *rastro1* como para o *rastro2*. Logo depois, o elemento 3 dos rastros desalinhou e assim temos um novo segmento até que algum alinhamento ocorra e assim por diante.

As variações são potenciais representações das características similares à nova a ser inserida ou podem representar algum ruído do rastro. Além disso, tem-se a identificação das seqüência de métodos que foram executados anterior e posteriormente aos métodos que variaram. Por exemplo, na Figura 3.3, para o segmento 4 que desalinhou, temos o método “*c*” executado anteriormente no segmento 3 e, posteriormente, o método “*e*” no segmento 5. Tal seqüência servirá de base para as análises de impacto que deverão ser feitas para completar o processo de mudança incremental. O escopo do presente trabalho restringe-se somente a análise da aplicação de alinhamento de rastros no processo de localização de características.

Rastro1	Rastro2	
a	a	} Segmento1
b	b	
b	—	} Segmento2
c	c	} Segmento3
c	c	
—	d	} Segmento4
e	e	} Segmento5

Figura 3.3: Resultado do alinhamento.

3.4 Exemplo

Nesta seção é apresentado uma aplicação do alinhamento de rastros entre duas características em um pequeno editor gráfico mostrado na Figura 3.4.

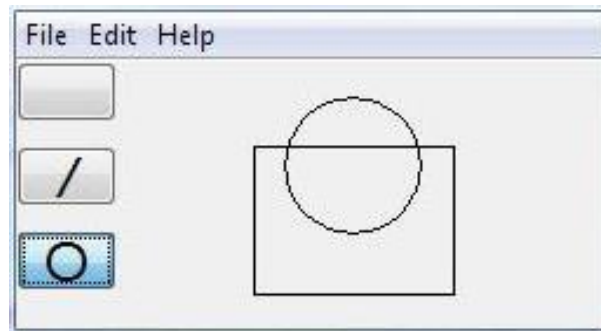


Figura 3.4: Aplicação alvo.

Possui 378 linhas de código, distribuídas em 9 classes e contendo 58 métodos. Para o exemplo, suponha que uma nova requisição, por exemplo desenhar quadrado, foi solicitada para ser implementada no sistema. Além disso, suponha que a aplicação já possua as funcionalidades ou características: desenhar reta e desenhar círculo.

Portanto, a questão de compreensão que será tratada é como descobrir onde e como os desenvolvedores deverão implementar a nova requisição?

Assim, os seguintes passos devem ser feitos:

1. Definição de roteiros de execução
 - (a) Desenhar uma reta;
 - (b) Desenhar um círculo;
2. Extração de rastros:
 - (a) Executar a funcionalidade desenhar reta;
 - (b) Executar a funcionalidade desenhar círculo;
3. Comprimir Rastros
4. Alinhar Rastros

3.4.1 Definição de Roteiros de Execução

Dado que se precisa inserir a característica *desenhar quadrado*, precisa-se definir quais outras funcionalidades o sistema possui e são similares à nova requisição. Nesta fase, informações de um especialista são importantes, caso não exista documentação do sistema. A saída dessa etapa será a descrição da funcionalidade a ser executada.

3.4.2 Extração de Rastros

Uma vez definidas as funcionalidades é preciso descrever detalhadamente como executar os roteiros de execução propostos:

Roteiro de execução 1: Executar a funcionalidade desenhar reta.

1. Iniciar aplicativo (inicialização);
2. Clicar no botão desenhar reta;
3. Clicar e arrastar o mouse sobre o painel de desenho;
4. Finalizar aplicativo.

Roteiro de execução 2: Executar a funcionalidade desenhar círculo.

1. Iniciar aplicativo (inicialização);
2. Clicar no botão desenhar círculo;
3. Clicar e arrastar o mouse sobre o painel de desenho;
4. Finalizar aplicativo.

As saídas desse processo serão rastros de execução, isto é, seqüências de chamadas de métodos para cada *thread* disparada durante a execução de cada roteiro. Como descrito na seção 3.2.1, ao executar os roteiros de execução, o usuário pode marcar o rastro para delimitar partes da seqüência que identifique alguma característica. Por exemplo, antes de executar a atividade “*Clicar no botão reta*”, adiciona-se um marcador de início e ao terminar um marcador de fim. Dessa forma, é possível identificar um segmento da seqüência que represente uma característica. Neste exemplo, marcou-se duas características para cada roteiro de execução.

No primeiro roteiro de execução tem-se as características “*Iniciar aplicativo*” que consistiu do primeiro método executado até a aplicação terminar sua inicialização e a característica denominada *reta* que consistiu na execução das atividades “*Clicar no botão reta*” e “*Clicar e arrastar o mouse sobre o painel de desenho*”.

No segundo roteiro de execução, de forma similar ao primeiro, a marcação foi feita para as características “*Iniciar aplicativo*” que consistiu do primeiro método executado até a aplicação terminar sua inicialização e a característica denominada “*desenhar círculo*” que

consistiu na execução das atividades “*Clicar no botão círculo*” e “*Clicar e arrastar o mouse sobre o painel de desenho*”.

Para o roteiro de execução 1: ocorrência de duas *threads* sendo a primeira com 14 chamadas de métodos e a segunda com 918 chamadas de métodos.

Para o roteiro de execução 2: ocorrência de duas *threads* sendo a primeira com 14 chamadas de métodos e a segunda com 516 chamadas de métodos.

O alinhamento pode ser feito para ambas as *threads*, contudo é necessário verificar as *threads* de interesse e nesse contexto as técnicas de localização de características são eficazes. Neste exemplo, devido ao pequeno porte da aplicação basta analisar os métodos contidos nas *threads*, Tabelas 3.2 e 3.3, e percebe-se que a *thread* 1 contém métodos genéricos e sem importância para identificar funcionalidades na aplicação. Outro ponto importante é que ambos roteiros de execução apresentam o mesmo comportamento, indicado pelo alinhamento perfeito dos rastros das *threads* 1 e 2.

<i>thread</i> 1: Desenhar Reta
graphicaleditor.GraphicalEditor,main,1232677520346
graphicaleditor.GraphicalEditor,<init>,1232677520362
graphicaleditor.MainFrame,<init>,1232677520658
graphicaleditor.MainFrame\$1,<init>,1232677520674
graphicaleditor.ConjuntoFormas,<init>,1232677520674
graphicaleditor.ConjuntoFormas,<init>,1232677520674
graphicaleditor.MainFrame.jbInit,1232677520674
graphicaleditor.MainFrame\$2,<init>,1232677520674
graphicaleditor.MainFrame\$3,<init>,1232677520674
graphicaleditor.MainFrame\$4,<init>,1232677520675
graphicaleditor.MainFrame\$5,<init>,1232677520675
graphicaleditor.MainFrame\$6,<init>,1232677520675
graphicaleditor.MainFrame\$7,<init>,1232677520675
graphicaleditor.MainFrame\$8,<init>,1232677520675

Tabela 3.2: Métodos da *thread* 1 do roteiro de execução Desenhar Reta com seus *timestamps*.

Por outro lado, ainda que se trate de um sistema de pequeno porte, a análise da *thread* 2 requer um esforço considerável para entendimento do rastro, afinal tem-se quase 1.000 chamadas de métodos para o roteiro de execução 1. Logo, a compressão do rastro tem um papel muito relevante neste contexto, pois diminuirá o espaço de busca.

Além disso, percebe-se que há uma diferença de tamanho considerável entre os dois rastros, 918 chamadas de métodos para o primeiro roteiro e 516 chamadas de métodos para o segundo roteiro de execução, o que reforça a necessidade da compressão do rastro já que as técnicas de alinhamento de sequência são bastante sensíveis às entradas, requerendo sequências de tamanhos similares.

<i>thread</i> 1: Desenhar Círculo
graphicaleditor.GraphicalEditor,main,1232677654141
graphicaleditor.GraphicalEditor,<init>,2,1,1232677654172
graphicaleditor.MainFrame,<init>,1232677654422
graphicaleditor.MainFrame\$1,<init>,1232677654438
graphicaleditor.ConjuntoFormas,<init>,1232677654438
graphicaleditor.ConjuntoFormas,<init>,1232677654438
graphicaleditor.MainFrame.jbInit,1232677654438
graphicaleditor.MainFrame\$2,<init>,1232677654438
graphicaleditor.MainFrame\$3,<init>,1232677654438
graphicaleditor.MainFrame\$4,<init>,1232677654439
graphicaleditor.MainFrame\$5,<init>,1232677654439
graphicaleditor.MainFrame\$6,<init>,1232677654439
graphicaleditor.MainFrame\$7,<init>,1232677654439
graphicaleditor.MainFrame\$8,<init>,1232677654439

Tabela 3.3: Métodos da *thread* 1 do roteiro de execução Desenhar Círculo com seus *timestamps*.

Após comprimir as seqüências obteve-se rastros com 43 chamadas de tamanho para o roteiro de execução 1 e 41 chamadas para o cenário 2. O método de compressão utilizado é similar ao desenvolvido em [HLL02].

Os métodos envolvidos e seus respectivos identificadores são apresentado na Tabela 3.4.

3.4.3 Alinhamento dos Rastros

Após alinhar os rastros, obteve-se duas seqüências de tamanho 54 em um total de 25 segmentos sendo que destes 9 segmentos alinharam e 16 desalinharam. Na Tabela 3.5 são apresentados os segmentos gerados, alinhados ou não e um identificador dos métodos contidos em cada roteiro de execução. As análises de cada segmento são apresentadas a seguir.

O primeiro segmento, número 0, ocorreu no rastro do roteiro de execução 1 como também no roteiro de execução 2 na mesma ordem, portanto foi um segmento que alinhoun. Verificando as informações referentes às características implementas percebe-se que se trata de um segmento que contém métodos relacionados, predominantemente, à inicialização do sistema. Sendo assim, os segmentos alinhados não trazem informações relevantes para a localização das características *Desenhar reta* ou *Desenhar círculo*.

Os rastros contém os métodos 1, 2, 3, 4, 5, 6, 7 e 8, descritos na Tabela 3.4. Os segmentos de interesse são os desalinhados, ou seja, onde ocorreu variação nas seqüências 1 e 2.

A presença do número 0 indica espaço na seqüência e, portanto, ocorreu desalinhamento. Os métodos identificados por 9, 10, 25 e 26 são, respectivamente, *graphi-*

ID.	Método
1	graphicaleditor.MainFrame\$1.paint
2	graphicaleditor.MainFrame.access\$0
3	graphicaleditor.ConjuntoFormas.desenha
4	graphicaleditor.MainFrame.access\$1
5	graphicaleditor.MainFrame.processWindowEvent
6	graphicaleditor.MainFrame\$4.mousePressed
7	graphicaleditor.MainFrame.panelDesenho _mousePressed
8	graphicaleditor.MainFrame.createForma
9	graphicaleditor.Linha.<init>
10	graphicaleditor.Linha.setAncora
11	graphicaleditor.Ponto2D.<init>
12	graphicaleditor.MainFrame\$5.mouseDragged
13	graphicaleditor.MainFrame.panelDesenho _mouseDragged
14	graphicaleditor.Linha.getAncoraX
15	graphicaleditor.Ponto2D.getX
16	graphicaleditor.Linha.getAncoraY
17	graphicaleditor.Ponto2D.getY
18	graphicaleditor.Linha.setDimensao
19	graphicaleditor.Linha.desenha
20	graphicaleditor.Linha.getAncora
21	graphicaleditor.MainFrame\$4.mouseReleased
22	graphicaleditor.MainFrame.panelDesenho _mouseReleased
23	graphicaleditor.ConjuntoFormas.add
24	graphicaleditor.MainFrame.jMenuFileExit _actionPerformed
25	graphicaleditor.Circulo.<init>
26	graphicaleditor.Circulo.setAncora
27	graphicaleditor.Circulo.getAncoraX
28	graphicaleditor.Circulo.getAncoraY
29	graphicaleditor.Circulo.setDimensao
30	graphicaleditor.Circulo.desenha
31	graphicaleditor.Circulo.getAncora

Tabela 3.4: Métodos e identificadores.

caleditor.Linha.<init>, *graphicaleditor.Linha.setAncora*, *graphicaleditor.Circulo.<init>* e *graphicaleditor.Circulo.setAncora*.

Assim, pode-se concluir que para a nova característica, no exemplo, *desenhar quadrado*, similar às características de execução em análise, teria-se, potencialmente, que implementar os métodos *graphicaleditor.Quadrado.<init>* e *graphicaleditor.Quadrado.setAncora* similares aos métodos desalinhados nos segmentos 1 e 2, e que estes métodos deveriam ser chamados na mesma ordem dos respectivos “pares”.

O processo de análise segue-se por todos os desalinhamentos ocorridos. Abaixo relata-se o desalinhamento ocorrido entre os segmentos 10 e 11.

Houve um desalinhamento nos segmentos 10 e 11 das seqüências, conforme mostrado

Roteiro 1: Desenhar Reta			Roteiro 2: Desenhar Círculo		
# Seg.	Alinhou	Métodos	# Seg.	Alinhou	Métodos
0	Sim	1 2 3 4 5 6 7 8	0	Sim	1 2 3 4 5 6 7 8
1	Não	0 0	1	Não	25 26
2	Não	9 10	2	Não	0 0
3	Sim	11 12 13	3	Sim	11 12 13
4	Não	0	4	Não	27
5	Não	14	5	Não	0
6	Sim	15	6	Sim	15
7	Não	0	7	Não	28
8	Não	16	8	Não	0
9	Sim	17	9	Sim	17
10	Não	0	10	Não	29
11	Não	18 15 17	11	Não	0 0 0
12	Sim	1 2 3 4	12	Sim	1 2 3 4
13	Não	0 0	13	Não	30 31
14	Não	19 20	14	Não	0 0
15	Sim	15	15	Sim	15
16	Não	0	16	Não	31
17	Não	20	17	Não	0
18	Sim	17 21 22 23 1 2 3	18	Sim	17 21 22 23 1 2 3
19	Não	0 0	19	Não	30 31
20	Não	19 20	20	Não	0 0
21	Sim	15	21	Sim	15
22	Não	0	22	Não	31
23	Não	20	23	Não	0
24	Sim	17 4 5 24	24	Sim	17 4 5 24

Tabela 3.5: Segmentos entre os rastros dos Roteiro de execução 1 e 2.

Segmento 0	Segmento 1	Segmento 2	Segmento 3
... 8	0 0	9 10	11 ...
... 8	25 26	0 0	11 ...
Alinhado	Desalinhados		Alinhado

Tabela 3.6: Análise de desalinhamento: Segmentos 1 e 2.

na Tabela 3.7. Os métodos envolvidos foram os de número 18, 15 e 17, nessa ordem, para o roteiro de execução 1 e o método 29 para o roteiro de execução 2.

Os métodos 18 e 29 são, respectivamente, *graphicaleditor.Linha.setDimensao* e *graphicaleditor.Circulo.setDimensao*. Intuitivamente, teria-se um método *graphicaleditor.Quadrado.setDimensao* para a nova requisição.

Contudo, no Roteiro de execução 1 ocorreram as chamadas de mais dois métodos 15 e 17 que desalinham, respectivamente *graphicaleditor.Ponto2D.getX* e *graphicaleditor.Ponto2D.getY* que embora não sejam métodos específicos da característica *desenhar*

Segmento 9	Segmento 10	Segmento 11	Segmento 12
... 17	0	18 15 17	1 ...
... 17	29	0 0 0	1 ...
Alinhado	Desalinhados		Alinhado

Tabela 3.7: Análise de desalinhamento: Segmentos 10 e 11.

reta, devem ser sugeridos para avaliação de implementação no momento de se inserir uma nova funcionalidade. Além disso, para certificar que os métodos 15 e 17 devem ser sugeridos para uma nova implementação é preciso analisar o desalinhamento para verificar a causa da não ocorrência dos métodos 15 e 17 no segmento 11 do roteiro de execução 2, desenhar círculo. A análise consiste em verificar por que o método 29, *graphicaleditor.Circulo.setDimensao* não chama os métodos 15 e 17. E, logo, percebe-se que se trata de um método auto-contido, ou seja, as funcionalidades dos métodos 15 e 17 estão implementadas no corpo do método 29.

Enfim, ao final das análises, para o exemplo, foi proposta a criação dos métodos apresentados na Tabela 3.8 abaixo:

Novos métodos:
graphicaleditor.Quadrado.<init> graphicaleditor.Quadrado.setAncora graphicaleditor.Quadrado.getAncoraX graphicaleditor.Quadrado.getAncoraY graphicaleditor.Quadrado.setDimensao graphicaleditor.Quadrado.desenha graphicaleditor.Quadrado.getAncora

Tabela 3.8: Métodos sugeridos para implementação.

Os métodos *graphicaleditor.Ponto2D.getX* e *graphicaleditor.Ponto2D.getY* devem ser adaptados para suportar a nova inserção. Uma vez que desalinharam durante o processo, mas não são métodos que implementam apenas uma funcionalidade específica das características analisadas pois tais métodos ocorreram nos dois cenários apresentados.

3.5 Conclusão

Este capítulo apresentou a abordagem proposta para compreensão de pontos de variação em alinhamento de rastros de execução, bem como ilustrou a aplicação da abordagem em um sistema de pequeno porte. No próximo capítulo, é apresentado um estudo de caso com o objetivo de avaliar a aplicação da abordagem em um sistema de grande porte.

Capítulo 4

Estudo de Caso: ArgoUML

Com o objetivo de verificar a viabilidade da abordagem proposta em um sistema real, apresenta-se neste capítulo um estudo de caso sobre a ferramenta CASE de modelagem UML, ArgoUML. Duas versões da ArgoUML foram utilizadas com o intuito de comprovar as diferentes aplicações da abordagem.

Além disso, o paradigma “*Goal Question Metric*” (*GQM*) [BCR94] foi aplicado para caracterizar e fornecer um melhor entendimento do processo e, assim, estabelecer bases para comparação com trabalhos relacionados. Uma breve explicação desse paradigma é apresentado na seção seguinte.

4.1 *Goal Question Metric* (GQM)

O GQM baseia-se na definição de objetivos a serem mensurados para que estes sirvam de rota para o estabelecimento de questões que orientarão a definição de métricas em um contexto particular [MB97]. Importa destacar que o sucesso da aplicação do GQM está diretamente ligado à definição dos objetivos, uma vez que as escolhas das métricas e posterior avaliação dos dados dependem dos objetivos como mostrado na Figura 4.1.

Briand et al. [BDR] definiram uma estrutura modelo para GQM baseado em cinco aspectos:

- O **objetivo** define o objeto, o alvo, que será estudado, isto é, o processo ou produto que será analisado.
- O **propósito** do estudo expressa a intenção que se tem ao analisar o objeto em questão. Propósitos comuns, em ordem de dificuldade, são:

Caracterização: Retratar o atual estado ou desempenho de um processo de desenvolvimento de software ou produto.

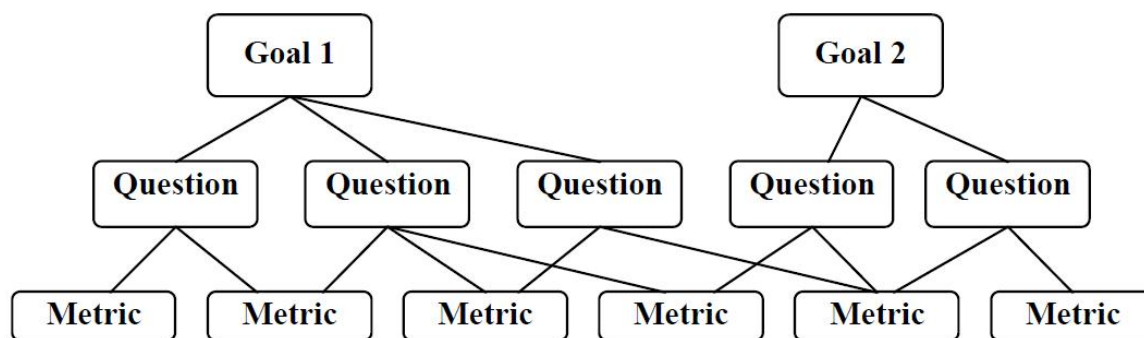


Figura 4.1: Estrutura Hierárquica do GQM.

Dimensão	Definição	Exemplos
Ojetivo do estudo	O que será analisado	teste de sistema, processo de desenvolvimento, produto final, etc
Propósito	Para que será analisado o objeto	caracterização, avaliação, predição, etc
Foco	Qual propriedade/atributo do objeto será analisado	custo, mudanças, corretude, etc
Ponto de vista	Quem usará os dados coletados	desenvolvedores, líderes de projeto, usuários, etc
Contexto	Em qual ambiente	projeto X, na corporação A, etc

Tabela 4.1: Sumário do Modelo GQM

Monitoração: Investigar tendências/evoluções de um estado/desempenho de um processo ou produto.

Avaliação: Comparar e medir a qualidade de um produto e a eficiência/eficácia de um processo.

Predição: Identificar relações entre vários fatores de processos ou produtos e através dessas relações prever atributos relevantes do processo ou produto.

Controle ou mudança: Identificar relacionamentos que influenciam o estado/desempenho de um processo ou produto.

- O **foco** define um atributo em particular do objeto em estudo que estará sujeito a um propósito. Exemplos de focos são custos, mudanças, manutenibilidade, etc.
- O **ponto de vista** identifica os papéis ou posições do pessoal que usará os resultados obtidos. Por exemplo, quem interpretará os dados coletados e usará para prever modelos.
- O **contexto** do estudo especifica o ambiente no qual o paradigma será aplicado.

Um sumário do modelo é apresentado na Tabela 4.1. Um exemplo paradigma é apresentado na Figura 4.2.

Objetivo 1 Verificar a aderência ao Processo

Propósito: Verificar
 Questão: a aplicação
 Objeto: do processo de manutenção de software
 Ponto de Vista: analisado pelo ponto de vista dos gerentes de projetos

Questão 1: Qual o grau de aplicabilidade do processo de manutenção?

Métrica 1: Densidade de execução das Tarefas

$$X = \frac{\text{N}^\circ \text{ de Tarefas Realizadas nos Ciclos}}{\text{N}^\circ \text{ de Tarefas Previstas nos Ciclos}}$$

Métrica 2: Porcentagem da Precisão dos Prazos

$$X = \frac{\text{N}^\circ \text{ de Solicitações Atendidas no Prazo} * 100}{\text{N}^\circ \text{ de Solicitações Feitas Durante o Período de Avaliação}}$$

Métrica 3: Precisão do Esforço

$$X = \frac{\text{Total de homens-hora realizado em cada tarefa da Implementação}}{\text{Total de homens-hora dimensionado em cada tarefa da Implementação}}$$

Figura 4.2: Exemplo modelo GQM [DdSK03].

4.2 Caracterização da Viabilidade da Abordagem Proposta

Esta seção apresenta os estudos que foram realizados no contexto deste trabalho. Trata-se de uma análise de viabilidade que procura verificar se a abordagem atinge os objetivos esperados.

Um dos propósitos é prever quais artefatos variantes serão relevantes para atividade de *localização de conceitos* em um processo de *mudança incremental*. Espera-se que o alinhamento de rastros de execuções aponte as variabilidades entre os roteiros de execução de forma que possamos extrair informações que facilitem a localização de conceitos no código para uma possível integração de nova funcionalidade no sistema alvo.

Outro propósito é identificar como mudanças foram implementadas para uma determinada funcionalidade em versões diferentes de um mesmo sistema. Espera-se que o alinhamento de rastros de execuções aponte as variabilidades entre os roteiros de execução de forma que as diferenças de implementações fiquem evidentes.

O estudo foi dividido em quatro contextos distintos: (1) Alinhamento de rastros extraídos da mesma versão do sistema alvo e da execução, por dois momentos distintos, da mesma função no sistema; (2) Alinhamento de rastros extraídos da mesma versão do

sistema alvo e da execução de funções similares no sistema; (3) Alinhamento de rastros extraídos de diferentes versões do sistema alvo e da execução, por dois momentos distintos, da mesma função do sistema; (4) Alinhamento de rastros extraídos de diferentes versões e da execução de funções similares. A Figura 4.3 mostra o esquema dos diferentes estudos realizados.

	Funções Iguais	Funções Similares
Versões Iguais	Contexto 1	Contexto 2
Versões Diferentes	Contexto 3	Contexto 4

Figura 4.3: Esquema dos estudos realizados.

4.2.1 Planejamento do Estudo

Objetivo do Estudo

Analisar a abordagem de alianhamento de rastros **com o propósito de** caracterizar a viabilidade em identificar variabilidades em rastros de execução relevantes ao processo de mudança incremental **do ponto de vista de** engenheiros de software **no contexto** manutenção de aplicações de grande porte.

Questões

As questões e métricas utilizadas para análise dos resultados são apresentadas abaixo:

Q1: O alinhamento de rastros diminui os esforço no processo de compreensão de rastros?

M1.1:

$$\text{Redução do Espaço de Busca (Métodos)} = \frac{|\text{Métodos Envolvidos}|}{|\text{Métodos Totais}|}$$

M1.2:

$$\text{Redução do Espaço de Busca (Classes)} = \frac{|\text{Classes Envolvidas}|}{|\text{Classes Totais}|}$$

M1.3:

$$\text{Redução do Espaço de Busca (Classes)} = \frac{|\text{Pacotes Envolvidas}|}{|\text{Pacotes Totais}|}$$

M1.4:

$$\text{Redução do Espaço de Busca (Chamadas)} = \frac{|\text{Chamadas não variantes}|}{|\text{Chamadas do rastro}|}$$

Onde:

Métodos envolvidos: métodos distintos executados.

Métodos totais: quantidade total de métodos da aplicação alvo.

Classes envolvidas: classes distintas que participaram do cenário.

Classes totais: quantidade total de classes da aplicação alvo.

Pacotes envolvidos: pacotes que participaram do cenário.

Pacotes totais: quantidade total de pacotes da aplicação alvo.

Chamadas não variantes: todas as chamadas de métodos que alinharam.

Chamadas do Rastro: todas as chamadas de métodos que ocorreram no rastro.

Q2: As variabilidades apresentadas no alinhamento de rastros apóiam a compreensão da aplicação?

M2: Análise qualitativa do especialista.

4.2.2 Contexto 1: Mesma Versão, Funções Iguais

Diante do contexto exposto, foi conduzido um estudo de viabilidade para caracterizar a abordagem de alinhamento de rastros quanto à sua aplicabilidade em identificar artefatos variantes e relevantes aos processos de mudanças incrementais.

Neste primeiro estudo de caso, supõe-se que haverá 100% de alinhamento entre os rastros, já que as seqüências são execuções extraídas de um mesmo sistema e funções iguais. Qualquer desalinhamento deve representar apenas ruídos do rastro.

Roteiro de Execução

O teste proposto consiste na criação de diagramas de classe utilizando a mesma forma de interação, ou seja, se trata de uma repetição idêntica de roteiro de execução. Duas execuções do mesmo roteiro de execução foram feitas e os rastros devidamente coletados. As etapas da execução estão sumarizadas abaixo:

1. Criar Diagrama de Classes pela barra de ferramentas (*toolbar*)
 - (a) Clicar sobre o botão correspondente para a criação de diagramas de classe na barra de ferramentas;
 - (b) Criar uma classe;

- (c) Criar outra classe;
- (d) Clicar sobre o botão “New Association”;
- (e) Clicar sobre uma classe e arrastar o ponteiro do mouse até a outra classe.

Foram marcadas no rastro as características *inicialização* do aplicativo, duas inserções de novas classes, ou seja, *nova classe* e *nova classe2*, e, por último, a característica *associação* feita entre as classes.

Resultados e Análises

Na Tabela 4.2 temos um sumário do resultado deste estudo, sendo que foram obtidos 115 segmentos: 58 alinhados e 57 desalinhados.

	Rastro 1	Rastro 2
Tamanho original	17.130	15.642
Tamanho comprimido	9.954	9.476
Quantidade de métodos na característica inicialização	326	253
Quantidade de métodos na característica Classe	3.596	3.547
Quantidade de métodos na característica Classe 2	2.867	2.860
Quantidade de métodos na característica Associação	2.956	2.599
Quantidade de métodos sem característica	209	217
Tamanho do rastro depois de alinhado	10.077	

Tabela 4.2: Sumário do alinhamento do estudo de caso 1.

O segmento número 1 desalinhou. Contudo, ao verificar métodos desalinhados percebe-se que se tratam de chamadas de métodos repetidos durante o restante da seqüência. Outro fato é que os métodos desalinhados em tal segmento ocorrem no segmento 2 na mesma ordem, o que caracteriza um *loop*. Não foi possível detectar a causa pela qual o compactador não eliminou esta repetição.

Este contexto, serve para mostrar que nem todos os desalinhamentos são relevantes, por conta dos ruídos que ocorrem durante a extração do rastro. Outro ponto importante é que mesmo utilizando técnicas de compressão de rastros, os ruídos ainda não são totalmente retirados.

Dessa forma, para as questões Q1 e Q2 os resultados do alinhamento são irrelevantes sob o ponto de vista de manutenção de software. No entanto, apontam a existência de ruídos nos rastros, uma vez que os cenários são os mesmos e desalinhamentos ocorreram.

4.2.3 Contexto 2: Mesma Versão, Funções Similares

No presente contexto, espera-se extrair informações relevantes ao processo de mudança incremental. Tais informações devem orientar os desenvolvedores, de forma mais rápida, na identificação dos potenciais métodos que serão necessários para implementar um suposto novo tipo de diagrama no ArgoUML 0.26 que será similar aos diagramas de *classe* e *caso de uso*.

Roteiro de Execução

O teste proposto consiste no alinhamento de dois rastros extraídos dos roteiros: criação de um *diagrama de classe* e na criação de uma *diagrama de caso de uso*. As execuções foram feitas conforme etapas descritas abaixo:

Roteiro de execução 1:

1. Criar Diagrama de Classes pela barra de ferramentas (toolbar)
 - (a) Clicar sobre o botão correspondente para a criação de diagramas de classe na barra de ferramentas;
 - (b) Criar uma classe;
 - (c) Criar outra classe;
 - (d) Clicar sobre o botão “New Association”;
 - (e) Clicar sobre uma classe e arrastar o ponteiro do mouse até a outra classe.

Roteiro de execução 2:

1. Criar Diagrama de Caso de Uso pela barra de ferramentas (toolbar)
 - (a) Clicar sobre o botão correspondente para a criação de diagramas de caso de uso na barra de ferramentas;
 - (b) Criar um ator;
 - (c) Criar um caso;
 - (d) Clicar sobre o botão “New Association”;
 - (e) Clicar sobre o caso e arrastar o ponteiro do mouse até o ator.

Para o roteiro de execução 1 foram marcadas no rastro as características *inicialização* do aplicativo, duas inserções de novas classes, ou seja, *nova classe* e *nova classe2*, e, por último, a característica *associação* feita entre as classes.

De forma similar, no roteiro de execução 2, foram marcadas no rastro as características *inicialização* do aplicativo, inserção de um *ator*, inserção de um *caso de uso* e, por último, a característica *associação* feita entre o ator e o caso.

Análise e Resultados

Na Tabela 4.3 temos um sumário do resultado deste estudo, sendo que foram obtidos 803 segmentos: 402 alinhados e 401 desalinhados.

	Rastro 1	Rastro 2
Tamanho original	16.679	13.632
Tamanho comprimido	9.628	8.104
Quantidade de métodos na característica inicialização	263	263
Quantidade de métodos na característica Classe	3.582	-
Quantidade de métodos na característica Ator	-	3.136
Quantidade de métodos na característica Classe 2	2.943	-
Quantidade de métodos na característica Caso de Uso	-	2.191
Quantidade de métodos na característica Associação	2.617	2.312
Quantidade de métodos sem característica	223	202
Tamanho do rastro depois de alinhado	12.646	

Tabela 4.3: Sumário do alinhamento do estudo de caso 2.

Para análise de viabilidade do alinhamento de rastros no processo de evolução de software, escolheu-se 4 segmentos do presente estudo de caso. A escolha dos segmentos foi feita levando-se em consideração a participação do segmento na implementação das características em análise. Para cada segmento escolhido é feita também a análise do segmento anterior e do segmento posterior com objetivo de identificar se a causa do desalinhamento não foi por ruído no rastro.

O primeiro segmento escolhido foi o de número 1. Para o roteiro de execução 1 (*Diagrama de Classe*) tal segmento é composto por espaços, já no roteiro de execução 2 (*diagrama de caso de uso*) ocorrem a chamada de 12 métodos distintos relatados na Tabela 4.4.

Os segmentos vizinhos 0 e 2 alinharam, assim poderia-se supor que tais métodos desalinhados seriam potenciais direcionadores para uma nova implementação de uma funcionalidade semelhante ao *diagrama de caso de uso*. Contudo, no segmento 2 ocorreram as mesmas chamadas na mesma ordem dos métodos. Logo, pode-se inferir que o desalinhamento ocorrido trata-se de um *loop* e os métodos apontados não são relevantes para um processo de evolução de software.

O segundo segmento escolhido é o de número 13, que indica participação da característica *caso de uso*, pois no roteiro 1 (*diagrama de classes*) apareceram somente espaços.

Roteiro de execução 2: Diagrama de Caso de Uso	
4	org.argouml.cognitive.ui.ToDoPane\$3.run
5	org.argouml.cognitive.ui.ToDoPane.access\$0
12	org.argouml.cognitive.ui.ToDoPane.updateCountLabel
8	org.argouml.cognitive.Designer.theDesigner
9	org.argouml.cognitive.Designer.getToDoList
13	org.argouml.cognitive.ToDoList.size
14	org.argouml.cognitive.ui.ToDoPane.formatCountLabel
15	org.argouml.cognitive.Translator.localize
16	org.argouml.application.Main\$1.i18nlocalize
17	org.argouml.i18n.Translator.localize
18	org.argouml.i18n.Translator.getName

Tabela 4.4: Chamada de métodos do segmento 1, roteiro 2.

Além disso, foi o primeiro grande desalinhamento ocorrido que consistiu em 1013 chamadas de métodos sendo 433 métodos distintos para o rastro do roteiro 2 (*diagrama de Caso de Uso*). Os segmentos 12 e 14 alinharam, logo no rastro do roteiro 1, não há outro desalinhamento correspondente e o segmento do roteiro 1 é composto apenas por espaços, então esse desalinhamento identifica métodos específicos do roteiro de execução 2.

O fato de não haver segmento correspondente no roteiro de execução 1, já que tanto os segmento posterior e anterior alinharam, é explicado pelo fato de que o ArgoUML 0.26 já inicia a aplicação com o ambiente para edição de diagrama de classe aberto, ou seja, todas as ferramentas estão carregadas na aplicação, logo a execução desta funcionalidade implícita ocorre apenas com o roteiro 2. Assim para iniciar um diagrama de caso de uso ao clicar “*Criar Diagrama de Caso de Uso*”, os métodos responsáveis pelo carregamento das ferramentas de um caso de uso são iniciados, provocando tamanho desalinhamento.

Enfim, para se inserir um novo diagrama será necessário avaliar se os métodos do segmento 13 serão necessários ao carregar os módulos do novo diagrama, assim os métodos desalinhados podem direcionar como serão os novos métodos. Portanto tais métodos são considerados relevantes.

No segmento 281 ocorreu desalinhamento, conforme mostrado na Tabela 4.5. Tais métodos estão participando da característica *criar outra classe* para o roteiro de execução 1 e *criar um caso* para o roteiro de execução 2.

	Segmento 280	Segmento 281		Segmento 282
Classes	... 158	0	159	160 ...
Caso de uso	... 158	1179	0	160 ...
	Alinhado	Desalinhados		Alinhado

Tabela 4.5: Análise de desalinhamento: Segmento 281.

Cada segmento possui apenas 1 método. No rastro do roteiro de execução 1, no segmento 281, tem-se o método de identificador 159, *org.argouml.uml.diagram.static*

`_structure.ClassDiagramGraphModel.getProject`. Já no rastro do roteiro de execução 2, no mesmo segmento, tem-se o método de identificador 1179, `org.argouml.uml.diagram.use_case.UseCaseDiagramGraphModel.getProject`.

Apenas pelos nomes dos pacotes, classes e métodos percebe-se que serão estruturas potenciais para implementação de uma nova funcionalidade semelhante.

Outro desalinhamento ocorreu no segmento 689, mostrado na Tabela 4.6, e estão relacionados a característica *nova associação* em ambos roteiros de execução.

	Segmento 688	Segmento 689						Segmento 690
Classe	... 826	0	915	916	917	916	918	248 ...
Caso de Uso	... 826	897	0	0	0	0	0	248 ...
	Alinhado	Desalinhados						Alinhado

Tabela 4.6: Análise de desalinhamento: Segmento 689.

Os métodos envolvidos no desalinhamento são mostrados na Tabela 4.7 e 4.8.

roteiro de execução 1: Diagrama de Classe	
915	org.argouml.uml.diagram.static_structure.ui.ClassDiagramRenderer.setPorts
916	org.argouml.uml.diagram.static_structure.ui.ClassDiagramRenderer. getNodePresentationFor
917	org.argouml.uml.diagram.static_structure.ui.ClassDiagramRenderer.setSourcePort
918	org.argouml.uml.diagram.static_structure.ui.ClassDiagramRenderer.setDestPort

Tabela 4.7: Métodos desalinhados do roteiro 1 no segmento 689.

roteiro de execução 2: Diagrama de Caso de Uso	
897	org.argouml.uml.diagram.ui.FigAssociation.damage

Tabela 4.8: Método desalinhado do roteiro 2 no segmento 689.

Embora a funcionalidade executada em ambos roteiros de execução seja similar ocorreram comportamentos totalmente diferentes nos dois roteiros de execução. Ainda assim, a abordagem proposta retorna os métodos envolvidos e caberá ao desenvolvedor decidir qual estrutura melhor se adequa a nova implementação.

De posse dos desalinhamentos ainda é necessário inspecioná-los manualmente para retirar segmentos que representam laços onde a compressão do rastro não conseguiu retirá-los, bastando excluir todos os segmentos repetidos e todos os segmentos que tenham repetições de subsequências.

Dos 401 segmentos desalinhados restaram 250 depois de retirados os repetidos e 219 retirando os laços não encontrados na fase de compressão. Nestes 219 segmentos que restaram existem 4.344 chamadas, as quais são definidas com 1.071 métodos distintos que estão contidos em 147 classes e estão dentro de 4 pacotes.

Na Tabela 4.9 é apresentado a quantidade de artefatos que devem ser levados em consideração no processo de evolução incremental do sistema, retratando as métricas M1.1, M1.2, M1.3 e M1.4:

	Métodos	Classes	Pacotes	Chamadas
A serem investigados	1071	147	4	4344
Totais do aplicativo	9533	1643	80	12646
Redução	88%	91%	95%	65%

Tabela 4.9: Sumário dos artefatos de código a serem estudados.

Para a questão Q2 é fato que a quantidade de métodos que deverão ser profundamente compreendidos pelos desenvolvedores foi reduzido, visto que se limitam aos pontos de variação. Diminuindo assim o esforço gastos no processo de compreensão de programas.

Outro aspecto importante é que com a segmentação dos rastros já se tem a primeira relação de dependência entre os métodos. Já pela seqüência de chamadas sabe-se quais métodos foram executados antes de depois de um ponto de variação.

4.2.4 Contexto 3: Versões Diferentes, Funções Iguais

Neste terceiro estudo de caso, o resultado do alinhamento dos rastros deve mostrar as diferenças de implementação de uma funcionalidade de uma versão em relação a outra de um mesmo aplicativo, fornecendo informações aos desenvolvedores sobre como foram feitas as evoluções de uma versão para outra.

Roteiro de Execução

O teste proposto consiste na criação de diagramas de classe utilizando a mesma forma, ou seja, se trata de uma repetição idêntica de roteiro de execução. Duas execuções do mesmo roteiro de execução foram feitas nas versões 0.26 e 0.28 do ArgoUML tendo os rastros devidamente coletados. As etapas da execução estão sumarizadas abaixo:

1. Criar Diagrama de Classes pela barra de ferramentas com a versão 0.26 e 0.28
 - (a) Clicar sobre o botão correspondente para a criação de diagramas de classe na barra de ferramentas;
 - (b) Criar uma classe;
 - (c) Criar outra classe;
 - (d) Clicar sobre o botão “New Association”;
 - (e) Clicar sobre uma classe e arrastar o ponteiro do mouse até a outra classe.

Foram marcadas no rastro as características *inicialização* do aplicativo, duas inserções de novas classes, *nova classe* e *nova classe2*, e, por último, a característica *associação* feita entre as classes.

Resultados e Análises

Na Tabela 4.10 temos um sumário do resultado deste estudo, sendo que foram obtidos 1.556 segmentos: 778 alinhados e 778 desalinhados. Retirando segmentos repetidos e laços não encontrados pelo compressor restaram 206 segmentos. Foram selecionados 4 segmentos, apresentados abaixo, levando em consideração os diferentes momentos que ocorreram.

	Rastro 1	Rastro 2
Tamanho original	16.679	14.707
Tamanho comprimido	9.628	7.546
Quantidade de métodos na característica Inicialização	263	110
Quantidade de métodos na característica Classe	3.582	2.505
Quantidade de métodos na característica Classe 2	2.943	2.082
Quantidade de métodos na característica Associação	2.617	2.807
Quantidade de métodos sem característica	209	42
Tamanho do rastro depois de alinhado	12.077	

Tabela 4.10: Sumário do alinhamento do estudo de caso 3.

O primeiro desalinhamento ocorreu no segmento número 1. Para o rastro da versão 0.26 tal segmento é composto pela chamada de 29 métodos, já na versão 0.28 ocorreu a chamada de apenas 2 métodos.

Verificando os segmentos 0 e 2, observa-se que tratam-se de segmentos alinhados, logo, pode-se deduzir que houve mudanças de implementação na inicialização do programa nos métodos do segmento 1. Os segmentos envolvidos no desalinhamento para cada roteiro de execução do segmento 1 são mostrados nas Tabelas 4.11 e 4.12.

ID	Método
1032	org.argouml.uml.diagram.static_structure.ui.UMLClassDiagramnotationAdded
1033	org.argouml.uml.diagram.use_case.ui.UMLUseCaseDiagramnotationAdded

Tabela 4.11: Segmento 1, Diagrama de Classe, versão 0.28.

Analisando os métodos executados na versão 0.28, em todas as *threads*, nota-se que em momento algum os métodos do segmento 1 da versão 0.26 foram utilizados na versão 0.28 para o cenário em questão. Ainda assim analisando os métodos contidos na versão 0.28 percebe-se que todos do segmento 1 da versão 0.26 pertencem a 0.28, ou seja, na nova versão tais métodos não são carregados ao inicializar a aplicação e nem foram utilizados no exercício de um mesmo cenário em ambas versões.

ID	Método
3	org.argouml.application.helpers.ApplicationVersion.getVersion
4	org.argouml.cognitive.ui.ToDoPane\$3.run
5	org.argouml.cognitive.ui.ToDoPane.access\$0
6	org.argouml.cognitive.ui.ToDoByPriority.toDoItemsRemoved
7	org.argouml.cognitive.ToDoListEvent.getToDoItemList
8	org.argouml.cognitive.Designer.theDesigner
9	org.argouml.cognitive.Designer.getToDoList
10	org.argouml.cognitive.ui.PriorityNode.getPriorityList
11	org.argouml.cognitive.ui.PriorityNode.getPriority
12	org.argouml.cognitive.ui.ToDoPane.updateCountLabel
8	org.argouml.cognitive.Designer.theDesigner
9	org.argouml.cognitive.Designer.getToDoList
13	org.argouml.cognitive.ToDoList.size
14	org.argouml.cognitive.ui.ToDoPane.formatCountLabel
15	org.argouml.cognitive.Translator.localize
16	org.argouml.application.Main\$1.i18nlocalize
17	org.argouml.i18n.Translator.localize
18	org.argouml.i18n.Translator.getName
19	org.argouml.i18n.Translator.loadBundle
20	org.argouml.application.events.ArgoEventPump\$1.run
21	org.argouml.application.events.ArgoEventPump.access\$0
22	org.argouml.application.events.ArgoEventPump.fireNotationEventInternal
23	org.argouml.application.events.ArgoNotationEvent.getEventType
24	org.argouml.notation.ui.NotationComboBox.notationAdded
25	org.argouml.notation.ui.NotationComboBox.refresh
26	org.argouml.notation.Notation.getAvailableNotations
27	org.argouml.notation.NotationNameImpl.getAvailableNotations
28	org.argouml.notation.NotationNameImpl.toString
29	org.argouml.notation.NotationNameImpl.getTitle

Tabela 4.12: Segmento 1, Diagrama de Classe, versão 0.26.

No segmento 137 ocorreu um desalinhamento envolvendo 99 chamadas de métodos no rastro do roteiro de execução 1, por outro lado esse mesmo segmento no roteiro de execução 2 é composto somente por 2 chamadas de métodos. Os segmentos vizinhos estão alinhados, caracterizando uma possível mudança no segmento 137.

Analisando os métodos envolvidos, tem-se para o roteiro de execução 1, que os 99 métodos do segmento 137 estão relacionados com propriedades da figura inserida no ArgoUML. Da mesma forma, no roteiro de execução 2 tem-se os métodos *org.argouml.uml.diagram.ui.FigStereotypesGroup.setBoundsImpl* e *org.argouml.uml.diagram.ui.FigStereotypesGroup.propertyChange* que certamente, também, estão relacionados com propriedades da figura inserida. Logo, podemos afirmar que embora as duas versões contenham os mesmo métodos estes, na nova versão e para este cenário, não são totalmente carregados na aplicação ao se utilizar os objetos que compõem o cenário.

Outra mudança ocorreu no segmento 187 mostrado na Tabela 4.13. Para o roteiro 1 tem-se a execução do método *org.argouml.uml.ui.foundation.core.PropPanelClass.setOrientation*, identificado pelo número 326 no rastro, por outro lado no roteiro 2 só houve espaço e os segmentos vizinhos, anterior e posterior, alinharam. Além disso, o método desalinhado em questão não aparece no rastro da versão mais nova. Analisando os métodos da versão 0.28 nota-se uma mudança arquitetural do programa. O método *setOrientation* está implementado na classe *AbstractArgoJPanel.java* e não mais foi sobre-escrito pela classe *PropPanel.java* que é pai da classe *PropPanelClass.java*.

	Segmento 186	Segmento 187	Segmento 188
Versão 0.26	... 324	326	327 ...
Versão 0.28	... 324	0	327 ...
	Alinhado	Desalinhados	Alinhado

Tabela 4.13: Análise de desalinhamento: Segmento 187.

No segmento 843 ocorreu o desalinhamento mostrado na Tabela 4.14:

	Segmento 842	Segmento 843			Segmento 844
Versão 0.26	... 208	0	0	209	178 ...
Versão 0.28	... 208	1.061	1.062	0	178 ...
	Alinhado	Desalinhados			Alinhado

Tabela 4.14: Análise de desalinhamento: Segmento 843.

O desalinhamento ocorreu entre a chamada do método 209, *org.argouml.uml.diagram.ui.FigStereotypesCompartment.getMinimumSize* da versão 0.26 e as chamadas dos métodos 1.061 e 1.062, respectivamente *org.argouml.uml.diagram.ui.FigStereotypesGroup.getMinimumSize* e *org.argouml.uml.diagram.ui.FigStereotypesGroup.getStereotypeFigs* da versão 0.28.

Por uma leitura rápida no código-fonte pode-se inferir que o método 209 sofreu alteração para o método 1.061. Além disso o método 1.062 foi inserido na versão 0.28 já que não existe na versão 0.26.

Dos 708 segmentos desalinhados restaram 206 depois de retirados os repetidos e os laços não encontrados na fase de compressão. Nestes 206 segmentos restantes são compostos por 3.510 chamadas de métodos, as quais são definidas com 559 métodos distintos que estiveram envolvidos em algum tipo de mudança de código de uma versão para outra. Tais métodos estão contidos em 148 classes e estão dentro de 8 pacotes.

Na Tabela 4.15 é apresentado a quantidade de artefatos que devem ser levados em consideração no processo de evolução incremental do sistema, retratando as métricas M1.1, M1.2, M1.3 e M1.4.

Para a questão Q2, o resultado apontou métodos que possivelmente foram suprimidos da nova versão ou estão sendo executados em outra *thread*. Ainda é possível que

	Métodos	Classes	Pacotes	Chamadas
A serem investigados	559	148	8	3510
Totais do aplicativo 0.26	9533	1643	80	12077
Totais do aplicativo 0.28	9555	1650	80	12077
Redução versão 0.26	95%	91%	90%	71%
Redução versão 0.28	95%	92%	90%	71%

Tabela 4.15: Sumário dos artefatos de código a serem estudados.

métodos da versão 0.28 possam ter incorporado as funcionalidades suprimidas. Mesmo assim, os pontos de variação entre os rastros foram úteis na identificação de mudanças de implementações.

4.2.5 Contexto 4: Versões Diferentes, Funções Similares

Neste último contexto, a princípio não se identifica o que o resultado do alinhamento poderia apresentar de útil a uma equipe de desenvolvedores, já que o alinhamento será feito entre dois rastros sem similaridade aparente. A seguir apresenta-se os roteiros de execuções e os resultados do alinhamento.

Roteiro de Execução

O teste proposto consiste na criação de um *diagrama de classe* para o primeiro roteiro de execução utilizando a versão 0.26 do ArgoUML e na criação de uma *diagrama de caso de uso* utilizando a versão 0.28 do ArgoUML no roteiro de execução 2. As execuções foram feitas conforme etapas descritas abaixo:

roteiro de execução 1:

1. Criar Diagrama de Classes pela barra de ferramentas (toolbar) com a versão 0.26
 - (a) Clicar sobre o botão correspondente para a criação de diagramas de classe na barra de ferramentas;
 - (b) Criar uma classe;
 - (c) Criar outra classe;
 - (d) Clicar sobre o botão “New Association”;
 - (e) Clicar sobre uma classe e arrastar o ponteiro do mouse até a outra classe.

roteiro de execução 2:

1. Criar Diagrama de Caso de Uso pela barra de ferramentas (toolbar)
 - (a) Clicar sobre o botão correspondente para a criação de diagramas de caso de uso na barra de ferramentas;

- (b) Criar um ator;
- (c) Criar um caso;
- (d) Clicar sobre o botão “New Association”;
- (e) Clicar sobre o caso e arrastar o ponteiro do mouse até o ator.

Para o roteiro de execução 1 foram marcadas no rastro as características *inicialização* do aplicativo, duas inserções de novas classes, ou seja, *nova classe* e *nova classe2*, e, por último, a característica *associação* feita entre as classes.

De forma similar, no roteiro de execução 2, foram marcadas no rastro as características *inicialização* do aplicativo, inserção de um *ator*, inserção de um *caso* e, por último, a característica *associação* feita entre o ator e o caso.

Análise e Resultados

Na Tabela 4.16 temos um sumário do resultado deste estudo, sendo que foram obtidos 866 segmentos: 433 alinhados e 433 desalinhados.

	Rastro 1	Rastro 2
Tamanho original	16.679	12.201
Tamanho comprimido	9.628	6.546
Quantidade de métodos na característica inicialização	263	110
Quantidade de métodos na característica Classe	3.582	-
Quantidade de métodos na característica Ator	-	2.403
Quantidade de métodos na característica Classe 2	2.943	-
Quantidade de métodos na característica Caso de Uso	-	1.716
Quantidade de métodos na característica Associação	2.617	2.283
Quantidade de métodos sem característica	223	34
Tamanho do rastro depois de alinhado	13.040	

Tabela 4.16: Sumário do alinhamento do estudo de caso 4.

O primeiro segmento desalinhado foi o de número 1. Para o roteiro 1 (*Diagrama de Classe, versão 0.26*) tais segmentos são compostos por 28 chamadas de métodos. Do outro lado, no roteiro 2 (*diagrama de caso de uso, versão 0.28*) o segmento 1 contém 2 métodos.

Contudo, nada pode ser afirmado acerca do resultado já que se tratam de métodos sem alguma semelhança. O que pode-se afirmar é que os alinhamentos ocorridos entre

os rastros tratam-se de métodos que não mudaram de uma versão para outra. Como por exemplo no segmento 3 onde ocorreu alinhamento de 10 métodos.

Enfim, o presente contexto não ajuda a responder questões relevantes nos processos de manutenção de sistemas.

4.3 Avaliação da Abordagem

Este capítulo mostrou o potencial da abordagem no processo de compreensão de elementos do código com base em roteiros de execução.

O contexto 1 foi útil apenas para demonstrar os ruídos que um rastro pode apresentar. Mesmo fazendo uso de técnicas de compressão, ruídos podem permanecer no rastro e causar desalinhamentos irrelevantes.

No contexto 4, os rastros são extraídos de roteiros semelhantes, mas diferentes a ponto do alinhamento não obter resultados significativos, ou seja, os desalinhamentos não trazem informação útil para o processo de compreensão de sistemas, pois ficaria a dúvida se este foi causado pela diferença das funcionalidades exercitadas ou diferença de versão do sistemas. Os segmentos alinhados indicariam que não houve mudança em tais métodos, mas o contexto 3 também contém a mesma informação.

Os contextos que realmente são úteis ao processo de compreensão de sistemas são o 2 e o 3.

No contexto 2, as evidências que direcionarão uma evolução em um sistema foram capturadas, ressaltando não só métodos específicos de um roteiro, mas também métodos que embora estejam presentes nos dois roteiros, desalinham em virtude da execução de métodos específicos. Com o resultados é possível até especificar *templates* das novas funcionalidades a serem implementadas.

Vale ressaltar que se considerássemos apenas os métodos distintos envolvidos dos dois rastros, então a captura dos métodos específicos de cada rastro seria facilmente obtida por disjunção de conjuntos. Entretanto, capturar métodos que são comuns aos rastros, mas que têm momentos de execuções peculiares a cada rastro não seria possível pela simples disjunção de conjuntos.

O contexto 3 demonstrou a efetividade da abordagem em mostrar as mudanças de implementação entre duas versões de um mesmo aplicativo. Os resultados indicam os métodos origem e destino envolvidos em refatoração e consegue captar novos métodos adicionados a aplicação. Neste trabalho as mudanças são apontadas, mas não são classificadas.

É sabido que avaliação baseada em estudo de caso tem um papel importante no processo de melhoramento de processos. Contudo, têm pontos fortes e fracos [KPP95]:

Pontos Fortes: Baixo custo, pode facilmente ser reproduzido em um caso real, útil para

identificar novos assuntos para serem investigados, útil para entender a causa e como fenômenos acontecem.

Pontos Fracos não tem significância estatística, apresenta mais dificuldade para se obter um bom desempenho, requer alta experiência no domínio da aplicação, dificuldade em assegurar que um projeto piloto seja representativo.

Outros tipos de estudo empíricos, por exemplo *quasi-experimentos* e experimentos controlados, devem ser produzidos para validar os resultados obtidos com o estudo de caso.

Além disso, outro problema encontrado foi o grande número de segmentos gerados. Este fato pode ser visto pelo número não tão expressivo de uma redução em torno de 40% do espaço de busca nos contextos 2 e 3. A compressão do número de segmentos seria um interessante trabalho visto que diminuiria ainda mais os pontos que os desenvolvedores teriam que compreender. Caso a compressão não pudesse ser viável, análises e processamentos adicionais nos segmentos são necessários para aumentar a escalabilidade da proposta.

Capítulo 5

Trabalhos Relacionados

Muitos aspectos deste trabalho foram inspirados por outros trabalhos em manutenção de software. Destacando os tópicos sobre recuperação de arquitetura de software, evolução de software, engenharia reversa e alinhamento de seqüências.

Ding e Medvidovic propõem um processo incremental para evolução de sistemas orientados a objetos com documentação pobre ou inexistente [DM01]. O processo tem como uma das etapas, a recuperação da arquitetura dos fragmentos do sistema de interesse, para dar suporte à reestruturação. Para a execução da etapa de recuperação parte-se de três pressupostos: conhecimento das mudanças desejadas, conhecimento das propriedades da aplicação em nível de usuário e entendimento das características arquiteturais básicas da plataforma de implementação. Os autores definem duas categorias para as atividades de recuperação da arquitetura: recuperação da arquitetura lógica (idealizada) e recuperação da arquitetura física (concreta). O trabalho é revisado com a adição de novas heurísticas para a etapa de identificação de componentes e novos estudos de caso [MJ06]. Diferentemente, o presente trabalho explorou as técnicas de alinhamento de seqüência na etapa de entendimento das implementações de características de um sistema.

Rajlich e Silva estudaram o reuso e a evolução de *arquiteturas ortogonais*, que são fragmentos de código organizados em camadas de mesmo nível de abstração [RS96]. Eles desenvolveram um processo independente de domínio de aplicação com o propósito de adaptar a arquitetura de um sistema para receber um novo conjunto de requisitos. No entanto, os autores afirmam que tal processo tem aplicação em sistemas de pequeno e médio porte, confirmando que a fazer “clusters” no código-fonte não é uma técnica efetiva em sistemas de grande porte. Já neste trabalho, o propósito foi a aplicação em sistemas de grande porte, onde as técnicas de alinhamento de seqüências possam auxiliar as atividades do processo de evolução de software, diminuindo a quantidade de artefatos que o

mantenedores terão de compreender e localizando-os mais facilmente.

No trabalho [YNCC04], **Ying et al.** apresentaram uma nova abordagem, baseada em técnicas de mineração de dados, para determinar os conjuntos de arquivos de código-fonte que foram modificados em conjunto durante uma etapa da manutenção de um sistema. A grande vantagem da abordagem consiste na não dependência de execução de roteiros diferentemente da abordagem aqui proposta.

Em relação aos processos de engenharia reversa existem várias abordagens: recuperação de arquitetura de software, visualização de código, análise de impacto, localização de características, agrupamentos de código, detecção de clone, fatiadores de código, entre outras.

Sartipi et al. apresentaram um trabalho que consistiu em recuperar a arquitetura do sistema expressa através de padrões definidos na linguagem AQL - Architectural Query Language - conjuntamente com técnicas de mineração de dados. O sistema é traduzido em um modelo de grafos para o casamento dos padrões à partir do código-fonte [SKM00]. Em outro trabalho [SDS06], é apresentado um arcabouço que combina as informações das análises dinâmica e estática. Acredita-se que a combinação das análises dinâmica e estática enriquecem a tarefa de compreensão de programas.

Outro importante trabalho [SS06], **Sartipi et al.** propuseram de recuperação de arquitetura baseada em técnicas de mineração de padrões seqüenciais que tem por finalidade identificar características de implementação de sistemas. Os padrões seqüenciais são usados para identificar o grupo de funções principais que são utilizadas para executar determinada tarefa no sistema. Esses resultados podem ser utilizados tanto na compreensão do funcionamento do sistema como no entendimento do código.

Na mesma linha, **Vasconcelos et al.** [VCW05,VW04] apresenta uma série de heurísticas para o agrupamento de classes em sistemas orientados a objeto a partir dos rastros de execução do sistema.

A análise de impacto é responsável por identificar as partes do sistema que a nova mudança afetará. Uma técnica bastante utilizada é *fatiar* o código (*Program Slicing*) em buscas de tais partes. Binkley e Gallagher apresentaram um *survey* sobre esse método [BG96]. Neste trabalho, pode-se usar futuramente um fatiador para análise de impacto usando como entradas os métodos listados nos relatório de variação.

Agrupamento ou “*clustering*” é outra técnica de mineração de dados bastante utilizada em processos de classificação de entidade de código. A idéia é extrair grupos de entidades relacionadas através de métricas de similaridade. Wiggerts [Wig97], Anquetil [AFL99] e Tzerpos [TH98] mostram diferentes aspectos dos algoritmos de agrupamento para código-fonte.

Localização de características ou conceitos em código-fonte é uma tarefa comum no suporte às atividades de evolução de software. Marcus et al. apresentaram um método de recuperação de informação denominado *Latent Semantic Indexing (LSI)* que é usado

para mapear conceitos expressos em linguagem natural a trechos relevantes do código-fonte [MSRM04]. Neste trabalho, o alinhamento de seqüências de rastros mostrou que além localizar métodos que implementam determinadas características no sistema, pode-se deduzir os métodos que deverão ser implementados numa possível evolução, bem como o alinhamento mostrou-se valioso para se compreender como funcionalidades de um sistema estão implementadas.

No campo da Biologia Molecular, a comparação entre seqüências e sub-seqüências se tornou essencial. A identificação de regiões similares em seqüências de DNA, RNA ou proteínas possibilitou mapear relações funcionais, estruturais e evolucionárias. Vários algoritmos são apresentados em [Gus97]. No presente trabalho, o algoritmo de alinhamento global, *Needleman e Wunsch*, foi adaptado para o contexto para o domínio da manutenção de sistemas.

Capítulo 6

Conclusões e Perspectivas

Este trabalho apresentou uma abordagem de apoio à compreensão de programas baseada em alinhamento de rastros de execução que pode ser aplicada nos processos de manutenção de sistemas. É composta, basicamente, por quatro etapas, a saber: definição de roteiros para análise, extração de rastros de execução, alinhamento dos rastros de execução e comparação das variabilidades apontadas.

Conforme apresentado, o alinhamento de rastros pode oferecer um maior apoio no processo de compreensão de sistemas fornecendo elementos de código que guiariam novas implementações ou identificando e caracterizando mudanças de implementações em sistemas.

O objetivo foi diminuir o esforço gasto por equipes de desenvolvimento de software para compreender um sistema, em especial, sistemas que possuem pouca ou nenhuma documentação.

6.1 Contribuições

As principais contribuições deste trabalho foram:

1. Apoio à compreensão de software, através da identificação de pontos de variação em rastros de execução: O alinhamento de rastros dá suporte a manutenção de sistemas, uma vez que identifica métodos que implementam requisitos funcionais. Os rastro é segmentado e permite a compreensão dos conceitos que foram implementados no sistema. Além disso, dado que uma funcionalidade do software precisa ser integrada, o rastro permite localizar elementos que podem ser impactados pela manutenção.
2. Localização de pontos de refatoração feitas em sistemas: O alinhamento de rastros de execução possibilita identificar e caracterizar pontos no sistema onde houveram

refatorações.

3. Estudos de caso que sugerem indícios da viabilidade do alinhamento de rastros nos processos de manutenção de sistemas.

6.2 Limitações Identificadas

Algumas limitações que identificou-se neste trabalho são:

1. A compressão dos rastros dos roteiros: A compressão foi realizada com o apoio de uma ferramenta. No entanto, mesmo após compressão, ruídos ainda persistem nos rastros. Dessa forma, o número de segmentos feito pelo alinhamento é alto. Acredita-se que esses segmentos possam ser diminuídos, visto que existem segmentos duplicados ou com grande parte dos métodos duplicados, e assim o número de elementos que deveriam ser analisados pelos desenvolvedores seria menor e por consequência o esforço gasto também.
2. Utilização de roteiros: A abordagem é aplicada em roteiros de execução e, portanto, a identificação de pontos de variação se restringe ao escopo do roteiro executado. Outros métodos podem ser necessários durante o processo de manutenção de sistemas. Para a abordagem ser completa seria necessário ter todos os roteiros para os requisitos em análise.
3. Esforço na avaliação das variabilidades identificadas: Os relatórios produzidos pela abordagem requerem grande esforço quando o número de segmentos é grande. Contudo, acredita-se que os relatórios possam ser melhorados ofertando maior agilidade na avaliação e na compreensão dos resultados.

6.3 Trabalho Futuros

Alguns trabalhos que podem ser realizados a partir deste são:

1. Alterar a forma dos relatórios: seria interessante apresentar, como resultado do alinhamento, artefatos de software para prosseguir com o processo de mudança incremental. A próxima etapa seria a análise de impacto e para isso seria necessário identificar estruturas estáticas no código.
2. Extensão da abordagem para linhas de produto: Uma vez que se pode identificar pontos de variação e pontos comuns em sistemas, talvez a abordagem poderia ser aplicada no contexto de linhas de produto, já que, basicamente, são compostas de artefatos comuns e artefatos variáveis.
3. Identificação de outras questões: Acredita-se que o alinhamento de rastros possa responder outras questões relacionadas com a manutenção de sistemas, por exemplo,

“uma característica já implementada se parece com uma sugestão de métodos a serem implementados obtidos por *alinhamento* de outras características?”

Referências Bibliográficas

- [ABDM01] Alain Abran, Pierre Bourque, Robert Dupuis, and James W. Moore, editors. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2001.
- [AFL99] Nicolas Anquetil, Cédric Fourrier, and Timothy C. Lethbridge. Experiments with clustering as a software remodularization method. In *WCRE '99: Proceedings of the Sixth Working Conference on Reverse Engineering*, page 235, Washington, DC, USA, 1999. IEEE Computer Society.
- [ANQ00] N. ANQUETIL. Curso de engenharia reversa, notas de aula, 2000.
- [BCR94] V. Basili, G. Caldiera, and H.D. Rombach. Goal question metric approach. In *Encyclopedia of Software Engineering*, pages 528–532. John Wiley & Sons, Inc., 1994.
- [BDR] Lionel C. Bri, Christiane M. Differding, and H. Dieter Rombach. Practical guidelines for measurement-based process improvement.
- [Ben96] Keith Bennett. Software evolution: past, present and future. *Information and Software Technology*, Vol. 38(11):673–680, November 1996.
- [BG96] David W. Binkley and Keith Brian Gallagher. Program slicing. *Advances in Computer*, 1(43), July 1996.
- [BR00] Keith H. Bennett and Václav T. Rajlich. Software maintenance and evolution: a roadmap. In *Conference on The Future of Software Engineering*, pages 73–87, New York, NY, USA, 2000. ACM Press.
- [CCI90] Elliot J. Chikofsky and James H. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Softw.*, 7(1):13–17, 1990.
- [dAMSP⁺08] Marcelo de A. Maia, Victor Sobreira, Klérison R. Paixão, Sandra A. de Amo, and Ilmério R. Silva. Using a sequence alignment algorithm to identify specific and common code. In Orla Greevy David Röthlisberger Andy Zaidman, Abdelwahab Hamou-Lhadj, editor, *4th International Workshop on Program Comprehension through Dynamic Analysis*, pages 6–10, 2008.
- [dB03] Rogério Theodoro de Brito. Alinhamento de seqüências biológicas. Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, 2003.

- [DdSK03] Anquetil N. Damian de Sousa K., Oliveira K.M. de. Aplicação de gqm para avaliação de processo de manutenção de software. In *Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento*, Valdivia, Chile, 2003.
- [DM01] Lei Ding and N. Medvidovic. Focus: a light-weight, incremental approach to software architecture recovery and evolution. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*, pages 191–200, 28–31 Aug. 2001.
- [Eis05] Andrew David Eisenberg. Dynamic feature traces: Finding features in unfamiliar code. In *In Proceedings IEEE International Conference on Software Maintenance (ICSM 2004*, pages 337–346. IEEE Computer Society Press, 2005.
- [FR07] Neal Febbraro and Vaclav Rajlich. The role of incremental change in agile software processes. In *AGILE '07: Proceedings of the AGILE 2007*, pages 92–103, Washington, DC, USA, 2007. IEEE Computer Society.
- [Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, January 1997.
- [HLL02] Abdelwahab Hamou-Lhadj and Timothy C. Lethbridge. Compression techniques to simplify the analysis of large execution traces. In *IWPC '02: Proceedings of the 10th International Workshop on Program Comprehension*, page 159, Washington, DC, USA, 2002. IEEE Computer Society.
- [KBLN04] Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin. An ethnographic study of copy and paste programming practices in oopl. In *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 83–92, Washington, DC, USA, 2004. IEEE Computer Society.
- [KPP95] Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger. Case studies for method and tool evaluation. *IEEE Softw.*, 12(4):52–62, 1995.
- [KSNM05] Miryung Kim, Vibha Sazawal, David Notkin, and Gail Murphy. An empirical study of code clone genealogies. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 187–196, New York, NY, USA, 2005. ACM.
- [LYC00] Yang Li, Hongji Yang, and William Chu. Generating linkage between source code and evolvable domain knowledge for the ease of software evolution. *Principles of Software Evolution, International Symposium on*, 0:196, 2000.
- [MB97] Yasuhiro Mashiko and Victor R. Basili. Using the gqm paradigm to investigate influential factors for software process improvement. *J. Syst. Softw.*, 36(1):17–32, 1997.

- [MJ06] Nenad Medvidovic and Vladimir Jakobac. Using software evolution to focus architectural recovery. *Automated Software Engineering*, 13(2):225–256, April 2006.
- [MRB⁺05] Andrian Marcus, Vaclav Rajlich, Joseph Buchta, Maksym Petrenko, and Andrey Sergeyev. Static techniques for concept location in object-oriented code. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, pages 33–42, Washington, DC, USA, 2005. IEEE Computer Society.
- [MSRM04] Andrian Marcus, Andrey Sergeyev, Vaclav Rajlich, and Jonathan I. Maletic. An information retrieval approach to concept location in source code. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)*, pages 214–223, Washington, DC, USA, 2004. IEEE Computer Society.
- [NW70] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal fo Molecular Biology*, (48):443–453, 1970.
- [OLP01] Frédéric Plewniak Jean-Claude Thierry Odile Lecompte, Julie D. Thompson and Olivier Poch. Multiple alignment of complete sequences (macs) in the post-genomic era. *Gene*, (270):17–30, 2001.
- [RHLR08] Abhishek Rohatgi, Abdelwahab Hamou-Lhadj, and Juergen Rilling. An approach for mapping features to code based on static and dynamic analysis. In *ICPC '08: Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*, pages 236–241, Washington, DC, USA, 2008. IEEE Computer Society.
- [RS96] Václav Rajlich and João H. Silva. Evolution and reuse of orthogonal architecture. *IEEE Transaction on Software Engineering*, 22(2):153–157, 1996.
- [SdAM08] Victor Sobreira and Marcelo de Almeida Maia. A visual trace analysis tool for understanding feature scattering. In *WCRE*, pages 337–338, 2008.
- [SDS06] K. Sartipi, N. Dezhkam, and H. Safyallah. An orchestrated multi-view software architecture reconstruction environment. In *Reverse Engineering, 2006. WCRE '06. 13th Working Conference on*, pages 61–70, Oct. 2006.
- [SKM00] K. Sartipi, K. Kontogiannis, and F. Mavaddat. Architectural design recovery using data mining techniques. In *Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European*, pages 129–139, 29 Feb.-3 March 2000.
- [Sob] Victor Sobreira. Uma abordagem para compreensão de programas baseada na localização de características em código fonte. Master's thesis, Faculdade de Ciência da Computação - Universidade Federal de Uberlândia.
- [SS06] Hossein Safyallah and Kamran Sartipi. Dynamic analysis of software systems using execution pattern mining. In *ICPC '06: Proceedings of the 14th IEEE International Conference on Program Comprehension*, pages 84–88, Washington, DC, USA, 2006. IEEE Computer Society.

- [SW81] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, (147):195–197, 1981.
- [TH98] Vassilios Tzerpos and Richard C. Holt. Software botryology: Automatic clustering of software systems. In *International Workshop on Large-Scale Software Composition*, pages 811–818, 1998.
- [Vas07] Aline Vasconcelos. *Uma Abordagem de apoio à Criação de Arquiteturas de Referência de Domínio baseadas na Análise de Sistemas Legados*. Tese (doutorado em engenharia de sistemas e computação), Universidade Federal do Rio de Janeiro, 2007.
- [VCW05] A. Vasconcelos, R. Cêpeda, and C. Werner. An approach to program comprehension through reverse engineering of complementary software views. In *1st International Workshop on Program Comprehension through Dynamic Analysis (PCODA)*, pages 58–62, 2005.
- [VW04] A. Vasconcelos and C. Werner. Software architecture recovery based on dynamic analysis. In *Simpósio Brasileiro de Engenharia de Software*, 2004.
- [Wig97] T. A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *WCRE '97: Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE '97)*, page 33, Washington, DC, USA, 1997. IEEE Computer Society.
- [WS95] Norman Wilde and Michael C. Scully. Software reconnaissance: Mapping program features to code. *Journal of Software Maintenance: Research and Practice*, 7(1):49–62, 1995.
- [YNCC04] Annie T. T. Ying, Raymond Ng, and Mark C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Trans. Softw. Eng.*, 30(9):574–586, 2004. Member-Murphy,, Gail C.

Apêndice 1: Relatório de alinhamento dos rastros do Editor Gráfico

Roteiro de execução 1: Desenhar desenhar reta

*** Analisando Segmento 0: 1 2 3 4 5 6 7 8

Sequencia: de 0 a 7.

-----> Característica inicialização: 5 metodos.

---> 100,000000% da Característica inicialização foi implementada no segmento 0

---> 62,500000% do segmento 0 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 8,108108% da Característica desenhar reta foi implementada no segmento 0

---> 37,500000% do segmento 0 implementa a Característica desenhar reta

*** Analisando Segmento 1: 0 0

Sequencia: de 8 a 9.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar reta: 37 metodos.

*** Analisando Segmento 2: 9 10

Sequencia: de 10 a 11.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 2

---> 0,000000% do segmento 2 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 5,405406% da Característica desenhar reta foi implementada no segmento 2

---> 100,000000% do segmento 2 implementa a Característica desenhar reta

*** Analisando Segmento 3: 11 12 13

Sequencia: de 12 a 14.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 3

---> 0,000000% do segmento 3 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 8,108108% da Característica desenhar reta foi implementada no segmento 3

---> 100,000000% do segmento 3 implementa a Característica desenhar reta

*** Analisando Segmento 4: 0

Sequencia: de 15 a 15.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar reta: 37 metodos.

*** Analisando Segmento 5: 14

Sequencia: de 16 a 16.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 5

---> 0,000000% do segmento 5 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 2,702703% da Característica desenhar reta foi implementada no segmento 5

---> 100,000000% do segmento 5 implementa a Característica desenhar reta

*** Analisando Segmento 6: 15

Sequencia: de 17 a 17.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 6

---> 0,000000% do segmento 6 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 2,702703% da Característica desenhar reta foi implementada no segmento 6

---> 100,000000% do segmento 6 implementa a Característica desenhar reta

*** Analisando Segmento 7: 0

Sequencia: de 18 a 18.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar reta: 37 metodos.

*** Analisando Segmento 8: 16

Sequencia: de 19 a 19.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 8

---> 0,000000% do segmento 8 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 2,702703% da Característica desenhar reta foi implementada no segmento 8

---> 100,000000% do segmento 8 implementa a Característica desenhar reta

*** Analisando Segmento 9: 17

Sequencia: de 20 a 20.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 9

---> 0,000000% do segmento 9 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 2,702703% da Característica desenhar reta foi implementada no segmento 9

---> 100,000000% do segmento 9 implementa a Característica desenhar reta

*** Analisando Segmento 10: 0

Sequencia: de 21 a 21.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar reta: 37 metodos.

*** Analisando Segmento 11: 18 15 17

Sequencia: de 22 a 24.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 11

---> 0,000000% do segmento 11 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 8,108108% da Característica desenhar reta foi implementada no segmento 11

---> 100,000000% do segmento 11 implementa a Característica desenhar reta

*** Analisando Segmento 12: 1 2 3 4

Sequencia: de 25 a 28.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 12

---> 0,000000% do segmento 12 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 10,810811% da Característica desenhar reta foi implementada no segmento 12

---> 100,000000% do segmento 12 implementa a Característica desenhar reta

*** Analisando Segmento 13: 0 0

Sequencia: de 29 a 30.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar reta: 37 metodos.

*** Analisando Segmento 14: 19 20

Sequencia: de 31 a 32.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 14

---> 0,000000% do segmento 14 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 5,405406% da Característica desenhar reta foi implementada no segmento 14

---> 100,000000% do segmento 14 implementa a Característica desenhar reta

*** Analisando Segmento 15: 15

Sequencia: de 33 a 33.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 15

---> 0,000000% do segmento 15 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 2,702703% da Característica desenhar reta foi implementada no segmento 15

---> 100,000000% do segmento 15 implementa a Característica desenhar reta

*** Analisando Segmento 16: 0

Sequencia: de 34 a 34.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar reta: 37 metodos.

*** Analisando Segmento 17: 20

Sequencia: de 35 a 35.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 17

---> 0,000000% do segmento 17 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 2,702703% da Característica desenhar reta foi implementada no segmento 17

---> 100,000000% do segmento 17 implementa a Característica desenhar reta

*** Analisando Segmento 18: 17 21 22 23 1 2 3

Sequencia: de 36 a 42.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 18

---> 0,000000% do segmento 18 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 18,918919% da Característica desenhar reta foi implementada no segmento 18

---> 100,000000% do segmento 18 implementa a Característica desenhar reta

*** Analisando Segmento 19: 0 0

Sequencia: de 43 a 44.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar reta: 37 metodos.

*** Analisando Segmento 20: 19 20

Sequencia: de 45 a 46.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 20

---> 0,000000% do segmento 20 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 5,405406% da Característica desenhar reta foi implementada no segmento 20

---> 100,000000% do segmento 20 implementa a Característica desenhar reta

*** Analisando Segmento 21: 15

Sequencia: de 47 a 47.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 21

---> 0,000000% do segmento 21 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 2,702703% da Característica desenhar reta foi implementada no segmento 21

---> 100,000000% do segmento 21 implementa a Característica desenhar reta

*** Analisando Segmento 22: 0

Sequencia: de 48 a 48.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar reta: 37 metodos.

*** Analisando Segmento 23: 20

Sequencia: de 49 a 49.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 23

---> 0,000000% do segmento 23 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 2,702703% da Característica desenhar reta foi implementada no segmento 23

---> 100,000000% do segmento 23 implementa a Característica desenhar reta

*** Analisando Segmento 24: 17 4 5 24

Sequencia: de 50 a 53.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 24

---> 0,000000% do segmento 24 implementa a Característica inicialização

-----> Característica desenhar reta: 37 metodos.

---> 8,108108% da Característica desenhar reta foi implementada no segmento 24

---> 75,000000% do segmento 24 implementa a Característica desenhar reta

-----> 25,000000% do segmento 24 não implementou alguma Característica.

Total de segmentos: 25

Quantidade de segmentos alinhados: 9

Quantidade de segmentos não alinhados: 16

Roteiro de execução 2: Desenhar desenhar círculo

*** Analisando Segmento 0: 1 2 3 4 5 6 7 8

Sequencia: de 0 a 7.

-----> Característica inicialização: 5 metodos.

---> 100,000000% da Característica inicialização foi implementada no segmento 0

---> 62,500000% do segmento 0 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 8,571429% da Característica desenhar círculo foi implementada no segmento 0

---> 37,500000% do segmento 0 implementa a Característica desenhar círculo

*** Analisando Segmento 1: 25 26

Sequencia: de 8 a 9.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 1

---> 0,000000% do segmento 1 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 5,714286% da Característica desenhar círculo foi implementada no segmento 1

---> 100,000000% do segmento 1 implementa a Característica desenhar círculo

*** Analisando Segmento 2: 0 0

Sequencia: de 10 a 11.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar círculo: 35 metodos.

*** Analisando Segmento 3: 11 12 13

Sequencia: de 12 a 14.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 3

---> 0,000000% do segmento 3 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 8,571429% da Característica desenhar círculo foi implementada no segmento 3

---> 100,000000% do segmento 3 implementa a Característica desenhar círculo

*** Analisando Segmento 4: 27

Sequencia: de 15 a 15.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 4

---> 0,000000% do segmento 4 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 2,857143% da Característica desenhar círculo foi implementada no segmento 4

---> 100,000000% do segmento 4 implementa a Característica desenhar círculo

*** Analisando Segmento 5: 0

Sequencia: de 16 a 16.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar círculo: 35 metodos.

*** Analisando Segmento 6: 15

Sequencia: de 17 a 17.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 6

---> 0,000000% do segmento 6 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 2,857143% da Característica desenhar círculo foi implementada no segmento 6

---> 100,000000% do segmento 6 implementa a Característica desenhar círculo

*** Analisando Segmento 7: 28

Sequencia: de 18 a 18.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 7

---> 0,000000% do segmento 7 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 2,857143% da Característica desenhar círculo foi implementada no segmento 7

---> 100,000000% do segmento 7 implementa a Característica desenhar círculo

*** Analisando Segmento 8: 0

Sequencia: de 19 a 19.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar círculo: 35 metodos.

*** Analisando Segmento 9: 17

Sequencia: de 20 a 20.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 9

---> 0,000000% do segmento 9 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 2,857143% da Característica desenhar círculo foi implementada no segmento 9

---> 100,000000% do segmento 9 implementa a Característica desenhar círculo

*** Analisando Segmento 10: 29

Sequencia: de 21 a 21.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 10

---> 0,000000% do segmento 10 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 2,857143% da Característica desenhar círculo foi implementada no segmento 10

---> 100,000000% do segmento 10 implementa a Característica desenhar círculo

*** Analisando Segmento 11: 0 0 0

Sequencia: de 22 a 24.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar círculo: 35 metodos.

*** Analisando Segmento 12: 1 2 3 4

Sequencia: de 25 a 28.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 12

---> 0,000000% do segmento 12 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 11,428572% da Característica desenhar círculo foi implementada no segmento 12

---> 100,000000% do segmento 12 implementa a Característica desenhar círculo

*** Analisando Segmento 13: 30 31

Sequencia: de 29 a 30.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 13
---> 0,000000% do segmento 13 implementa a Característica inicialização
-----> Característica desenhar círculo: 35 metodos.
---> 5,714286% da Característica desenhar círculo foi implementada no segmento 13
---> 100,000000% do segmento 13 implementa a Característica desenhar círculo

*** Analisando Segmento 14: 0 0

Sequencia: de 31 a 32.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar círculo: 35 metodos.

*** Analisando Segmento 15: 15

Sequencia: de 33 a 33.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 15

---> 0,000000% do segmento 15 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 2,857143% da Característica desenhar círculo foi implementada no segmento 15

---> 100,000000% do segmento 15 implementa a Característica desenhar círculo

*** Analisando Segmento 16: 31

Sequencia: de 34 a 34.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 16

---> 0,000000% do segmento 16 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 2,857143% da Característica desenhar círculo foi implementada no segmento 16

---> 100,000000% do segmento 16 implementa a Característica desenhar círculo

*** Analisando Segmento 17: 0

Sequencia: de 35 a 35.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar círculo: 35 metodos.

*** Analisando Segmento 18: 17 21 22 23 1 2 3

Sequencia: de 36 a 42.

-----> Característica inicialização: 5 metodos.
---> 0,000000% da Característica inicialização foi implementada no segmento 18
---> 0,000000% do segmento 18 implementa a Característica inicialização
-----> Característica desenhar círculo: 35 metodos.
---> 20,000000% da Característica desenhar círculo foi implementada no segmento 18
---> 100,000000% do segmento 18 implementa a Característica desenhar círculo

*** Analisando Segmento 19: 30 31

Sequencia: de 43 a 44.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.
---> 0,000000% da Característica inicialização foi implementada no segmento 19
---> 0,000000% do segmento 19 implementa a Característica inicialização
-----> Característica desenhar círculo: 35 metodos.
---> 5,714286% da Característica desenhar círculo foi implementada no segmento 19
---> 100,000000% do segmento 19 implementa a Característica desenhar círculo

*** Analisando Segmento 20: 0 0

Sequencia: de 45 a 46.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.
-----> Característica desenhar círculo: 35 metodos.

*** Analisando Segmento 21: 15

Sequencia: de 47 a 47.

-----> Característica inicialização: 5 metodos.
---> 0,000000% da Característica inicialização foi implementada no segmento 21
---> 0,000000% do segmento 21 implementa a Característica inicialização
-----> Característica desenhar círculo: 35 metodos.
---> 2,857143% da Característica desenhar círculo foi implementada no segmento 21
---> 100,000000% do segmento 21 implementa a Característica desenhar círculo

*** Analisando Segmento 22: 31

Sequencia: de 48 a 48.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.
---> 0,000000% da Característica inicialização foi implementada no segmento 22
---> 0,000000% do segmento 22 implementa a Característica inicialização
-----> Característica desenhar círculo: 35 metodos.

---> 2,857143% da Característica desenhar círculo foi implementada no segmento 22
---> 100,000000% do segmento 22 implementa a Característica desenhar círculo

*** Analisando Segmento 23: 0

Sequencia: de 49 a 49.

---> Não está alinhada.

-----> Característica inicialização: 5 metodos.

-----> Característica desenhar círculo: 35 metodos.

*** Analisando Segmento 24: 17 4 5 24

Sequencia: de 50 a 53.

-----> Característica inicialização: 5 metodos.

---> 0,000000% da Característica inicialização foi implementada no segmento 24

---> 0,000000% do segmento 24 implementa a Característica inicialização

-----> Característica desenhar círculo: 35 metodos.

---> 8,571429% da Característica desenhar círculo foi implementada no segmento 24

---> 75,000000% do segmento 24 implementa a Característica desenhar círculo

-----> 25,000000% do segmento 24 não implementou alguma Característica.

Total de segmentos: 25

Quantidade de segmentos alinhados: 9

Quantidade de segmentos não alinhados: 16