Pós-Graduação em Ciência da Computação

"Um Sistema de Recomendação de Código-Fonte para Suporte a Novatos"

Por

Yuri de Almeida Malheiros Barbosa

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, AGOSTO/2011

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Yuri de Almeida Malheiros Barbosa

"Um Sistema de Recomendação de Código-Fonte para Suporte a Novatos"

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Silvio Romero de Lemos Meira*

RECIFE, AGOSTO/2011

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Barbosa, Yuri de Almeida Malheiros
Um sistema de recomendação de código-fonte para suporte a novatos / Yuri de Almeida Malheiros Barbosa - Recife: O Autor, 2011.
x, 87 folhas : il., fig., tab.

Orientador: Silvio Romero de Lemos Meira.
Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2011.

Inclui bibliografia e apêndice.

1. Ciência da Computação. 2. Engenharia de software. I. Meira, Silvio Romero de Lemos (orientador). II. Título.

004 CDD (22. ed.) MEI2011 – 149

Dissertação de Mestrado apresentada por **Yuri de Almeida Malheiros Barbosa** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Um Sistema de Recomendação de Código-Fonte para Suporte a Novatos"**, orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:

Profa. Patricia Cabral de Azevedo Restelli Tedesco
Centro de Informática / UFPE

Prof. Leonardo Vidal Batista
Departamento de Informática / UFPB

Prof. Silvio Romero de Lemos Meira
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 12 de setembro de 2011.

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Eu dedico esta dissertação a minha família que me deu suporte para chegar até aqui.

Agradecimentos

Inicialmente eu gostaria de agradecer a minha família. Ao meu pai, João Batista, por me mostrar desde pequeno o prazer pelos estudos e a minha mãe, Francinete, pela grande ajuda e incentivo. Ambos sempre me apoiaram e tentaram me fornecer a melhor educação possível. Ao meu irmão Dimitri pela amizade e por ter me ajudado enquanto eu morava em Recife e ao meu irmão Igor por todos os momentos juntos.

Os amigos também foram muito importantes nesta jornada. Agradeço todas as vezes que me ouviram e que conversaram comigo, ao apoio em momentos de dificuldades e de incertezas, e sempre que de alguma forma me fizeram seguir em frente. Em especial esse agradecimento vai para os amigos Adriano, Amanda, Bruno, Eduardo e Roger.

Gostaria de agradecer também aos meus colegas de mestrado. Meu amigo desde a graduação Thiago Fernandes, a Bruno Neiva por toda ajuda e por todas as aventuras nas viagens que fizemos, e a todos os outros que conheci em Recife e que se mostraram grandes amigos, Emanoel Barreiros, Elias Queiroga e Wyllians Barbosa.

A todos os professores do Centro de Informática da Universidade Federal de Pernambuco (UFPE) que contribuíram na minha formação.

Ao meu orientador, Dr. Silvio Meira, por me aceitar como aluno e por me inspirar profundamente através de suas ideias, aulas e trabalho árduo. Também agradeço muito ao professor Alan Moraes, por todos os ensinamentos, discussões e ideias que ajudaram durante todo o mestrado. Sem o apoio dos dois eu não teria conseguido alcançar meus objetivos.

Any sufficiently advanced technology is indistinguishable from magic.

—ARTHUR C. CLARKE

Resumo

Novatos em um projeto de desenvolvimento de software costumam ter problemas nas suas primeiras tarefas, porque antes de tornarem-se produtivos como os desenvolvedores mais experientes, eles precisam aprender as ferramentas, como o código-fonte está organizado, como todo projeto funciona, entre outras atividades. Muitas vezes, para ensinar a um novato o que ele precisa, um mentor, alguém mais experiente no projeto, é alocado para guiá-lo nos seus primeiros passos.

Durante o desenvolvimento de software os desenvolvedores interagem com sistemas de controle de versão, sistemas de controle de mudanças e listas de discussão. Todas essas ferramentas gravam artefatos e a este conjunto de dados damos o nome de memória do projeto. Sistemas de recomendação podem ajudar os desenvolvedores usando a memória do projeto para recomendar artefatos importantes e assim evitar a necessidade de comunicação contínua entre as pessoas. Usando um sistema de recomendação os desenvolvedores perguntam primeiro para o computador, portanto eles só precisam perguntar para outro desenvolvedor se o sistema de recomendação falhar em guiá-lo para a resposta correta.

Este trabalho apresenta a criação de um sistema de recomendação para Engenharia de Software chamado Mentor. Seu objetivo é recomendar arquivos de código-fonte que devem fazer parte da solução de uma solicitação de mudança. Além disso são apresentados e discutidos os resultados de três experimentos realizados para comparar técnicas de atribuição de similaridade utilizando os dados dos projetos GTK+, Hadoop e GIMP. Usando o PPM para atribuir similaridade foram conseguidos resultados para recall rate entre 38,82% e 66,8%, e utilizando o LSI os resultados variaram entre 24,6% e 47,6%.

Palavras Chave: Sistemas de Recomendação, Engenharia de Software, Manutenção de Software, Teoria da Informação.

Abstract

Newcomers in a software development project often need assistance to complete their first tasks, because they need to learn how the whole project works, its architecture and how to use tools to become productive. Then a mentor, an experienced member of the team, teaches the newcomers what they need to complete the tasks.

But to allocate an experienced member of a team to teach a newcomer during a long time is neither always possible nor desirable, because the mentor could be more helpful improving the software project doing more important and hard tasks. The team interacts with version control system, bug tracking and mailing lists during the development, and all of these tools record data creating the project memory.

Recommender systems can use the project memory to help newcomers in their tasks, thus in some cases the developers don't need a mentor, they can ask first to the computer and only if the recommender system fails in guide, they ask a more experienced colleague.

This work presents Mentor, a recommender system to help newcomers to solve change requests. Mentor uses the Prediction by Partial Matching algorithm and some heuristics to analyze the change requests, and the version control data, and recommend potentially relevant source code that will help the developer. We did three experiments to compare the PPM algorithm with the Latent Semantic Indexing (LSI). Using PPM we achieved results for recall rate between 38.82% and 66.8%, and using LSI the results were between 24.6% and 47.6%.

Keywords: Recommender Systems, Software Engineering, Software Maintenance, Information Theory.

Sumário

1 Introdução

1.1 Motivação

Novatos em um projeto de desenvolvimento de software costumam ter problemas nas suas primeiras tarefas, porque precisam primeiro aprender as ferramentas, como o código-fonte está organizado, como todo projeto funciona, entre outras atividades. Até mesmo um desenvolvedor que não seja novo no projeto pode ter dificuldades em algumas tarefas, principalmente se elas estiverem relacionadas à alguma parte do projeto que ele não tenha conhecimento suficiente, e precisar de auxílio. Muitas vezes, para ensinar a um novato o que ele precisa, um mentor, alguém mais experiente no projeto, é alocado para guiá-lo nos seus primeiros passos (Hansman *et al.*, 2002).

Para ajudar, o mentor conversa com o novato, auxilia na resolução de problemas e com frequência mostra exemplos de código-fonte. Ensinar e aprender através de exemplos para resolver novos problemas é uma atividade comum na programação (Pirolli and Anderson, 1985).

Resolver o problema da falta de conhecimento em uma porção do código-fonte alocando outra pessoa para ensinar quem estiver com dificuldades nem sempre é viável. Um mentor é um papel temporário, logo ele não pode ficar ajudando outro desenvolvedor para sempre. Os custos do projeto tornam-se ainda maiores com pessoas separadas geograficamente, pois se sabe que a comunicação a distância é menos eficiente (Espinosa and Pickering, 2006). Nesse caso, se existir outra forma de auxiliar quem precisa de ajuda para entender o código-fonte, menos comunicação pode diminuir o tempo para resolver

uma tarefa (Herbsleb, 2007).

Para resolver uma solicitação de mudança um desenvolvedor precisa saber qual parte do código-fonte ele deve alterar para conseguir a solução desejada. Para isso, antes de resolver o problema, ele primeiro navega pelo código-fonte e documentos do projeto tentando encontrar informações importantes para execução da tarefa. Em um projeto muito grande, com centenas de arquivos, o trabalho de pesquisa para encontrar a porção importante de código-fonte para uma solicitação de mudança se torna muito custoso. Assim a resolução de uma solicitação de mudança pode ter atrasos imprevisíveis, pois não é possível saber o tempo que um desenvolvedor pode levar nessa busca (Cubranic *et al.*, 2005).

Durante o desenvolvimento de software os desenvolvedores interagem com sistemas de controle de versão, sistemas de controle de mudanças e listas de discussão. Todas essas ferramentas gravam artefatos e a este conjunto de dados damos o nome de memória do projeto. Sistemas de recomendação podem ajudar os desenvolvedores usando a memória do projeto para recomendar artefatos importantes e assim evitar a necessidade de comunicação contínua entre as pessoas. Usando um sistema de recomendação os desenvolvedores perguntam primeiro para o computador, portanto eles só precisam perguntar para outro desenvolvedor se o sistema de recomendação falhar em guiá-lo para a resposta correta (Herbsleb, 2007).

1.2 Definição do problema

Motivado pelos problemas citados, o objetivo do trabalho descrito nesta dissertação pode ser apresentado como:

Definir requisitos, projetar e implementar um sistema de recomendação de código-fonte para ajudar novatos na resolução de uma solicitação de mudança. Desta forma, a ferramenta guia o processo de pesquisa da solução e facilita o encontro de informações relevantes.

1.3 Visão geral da solução proposta

Para atingir o objetivo deste trabalho foi desenvolvido o Mentor, um sistema de recomendação para Engenharia de Software, que utiliza solicitações de mudança como entrada e retorna arquivos de código-fonte do projeto. Desta forma o sistema de recomendação auxilia a busca realizada pelo desenvolvedor para encontrar os arquivos relevantes que precisam ser entendidos e modificados para resolver uma solicitação de mudança. Esta seção descreve as características e a arquitetura da ferramenta.

O Mentor utiliza os artefatos criados por gerenciadores de solicitações de mudança e por controles de versão para efetuar recomendações. Eles são carregados no sistema e depois é realizada uma análise para que relações entre os artefatos sejam estabelecidas. Assim, o Mentor consegue receber como entrada uma solicitação de mudança e retornar os arquivos de código-fonte potencialmente importantes para resolução da solicitação.

O Mentor foi implementado através de um conjunto de componentes integrados que trabalham para prover as funcionalidades necessárias para o funcionamento do sistema de recomendação. A Figura 1.1 apresenta uma visão geral da arquitetura proposta.

Em linhas gerais, o *Parser* é responsável por processar os dados de gerenciadores de solicitações de mudança e de controles de versão, e gravar os dados no banco de dados do Mentor. O *Matcher* é responsável por analisar estes dados e atribuir relações entre as solicitações de mudança e as revisões do controle de versão. O *Similarity Assigner* processa as solicitações de mudança para gravar a relação de similaridade entre elas. Após o trabalho destes três componentes é possível saber quais arquivos de código-fonte estão relacionados a uma solicitação de mudança ainda não resolvida.

A arquitetura do Mentor foi projetada para ser possível trocar os componentes facilmente, assim pode-se implementar diferentes técnicas para o *Parser*, para o *Matcher* e para o *Similarity Assigner*. Basta apenas que eles tenham uma saída coerente com o funcionamento da ferramenta.

1.4 Fora do escopo

As seguintes questões não foram abordadas neste trabalho:

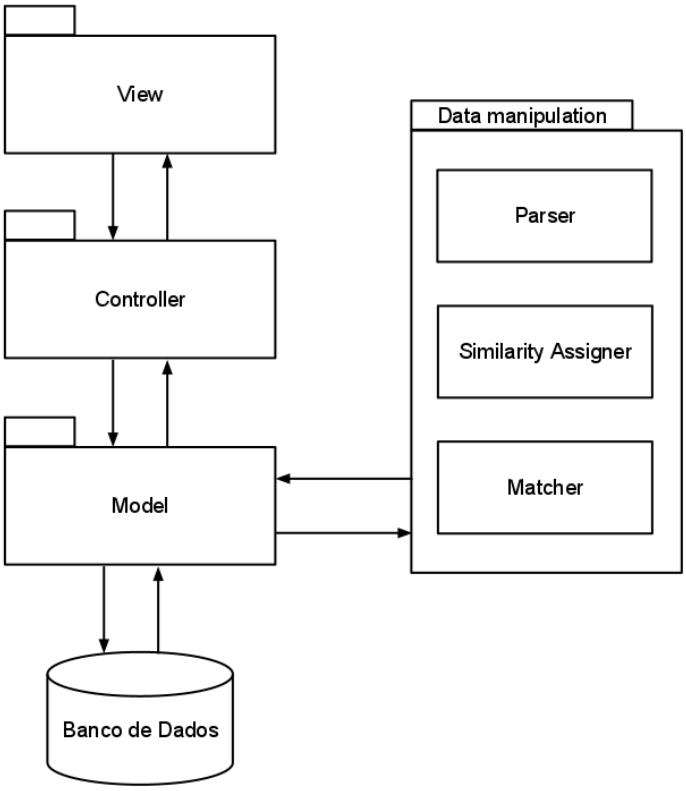

Figura 1.1 Arquitetura do Mentor

Gerenciamento de solicitações de mudança. O Mentor é um sistema de recomendação, que usa dados de solicitações de mudança de um projeto, mas está fora do escopo do trabalho ter mecanismos para o gerenciamento das solicitações de mudança de um projeto.

Integração com controle de versão. Novamente o Mentor apenas utiliza estes dados, por isso ele não possui nenhuma forma de integração com um controle de versão. A ferramenta apenas carrega estes dados para usar na sua técnica de recomendação.

Interferência nos dados carregados. Está fora do escopo do trabalho atribuir algum tipo de informação manual que ajude as recomendações. A ferramenta proposta funciona sem nenhuma interferência dos desenvolvedores, pois ela utiliza os dados como eles são.

Calcular o tempo ganho com a ferramenta. Mesmo com a proposta de facilitar o entendimento das tarefas e possivelmente diminuir o tempo para resolvê-la, a avaliação da ferramenta proposta não verificou o tempo ganho com as recomendações realizadas. O foco do trabalho está na qualidade das recomendações, não será avaliado se isso realmente causa uma diminuição no tempo de resolução de uma solicitação de mudança.

1.5 Contribuições realizadas

As seguintes contribuições podem ser listadas como resultados desta dissertação:

- A definição de requisitos, arquitetura e implementação de um sistema de recomendação de código-fonte para ajudar novatos na implementação de uma solicitação de mudança;
- O uso de compressão de dados em um sistema de recomendação;
- A avaliação empírica da qualidade das recomendações feitas pela ferramenta proposta neste trabalho através de um experimento formal, descrevendo as fases de definição, planejamento, execução, análise, interpretação e apresentação para replicação do estudo pela comunidade científica.

1.6 Organização da dissertação

Além do capítulo de Introdução, este trabalho envolve outros quatro capítulos, organizados da seguinte forma:

Capítulo 2. Apresenta todo o referencial teórico do trabalho. Conceitos de sistema de recomendação para Engenharia de Software, compressão de dados, mais especificamente o compressor *Prediction by Partial Matching* (PPM), *Latent Semantic Indexing* (LSI) e detalhes sobre os trabalhos relacionados são apresentados neste capítulo;

Capítulo 3. Descreve o sistema de recomendação desenvolvido neste trabalho, o Mentor, seus requisitos, arquitetura, implementação e uso;

Capítulo 4. Apresenta os experimentos realizados para a avaliação do Mentor e a discussão dos resultados obtidos;

Capítulo 5. Resume as contribuições deste trabalho e apresenta direções para trabalhos futuros.

2

Referencial Teórico

2.1 Sistemas de Recomendação

Sistemas de Recomendação (SR) são software que têm como objetivo ajudar os usuários a tomarem decisões que envolvem a interação com uma grande quantidade de informação. Eles recomendam itens de interesse dos usuários baseados em suas preferências, que podem ser colocadas no sistema explicitamente, por exemplo, o usuário entra com os gêneros musicais que mais gosta em uma aplicação que recomende novos artistas, ou as preferências podem ser implícitas, ou seja, a aplicação consegue descobrir de acordo com a interação do usuário o que ele mais gosta (Resnick and Varian, 1997).

Formalmente, o problema de recomendação pode ser entendido da seguinte forma. Dado C, um conjunto de entrada, S, um conjunto de todos os itens que podem ser recomendados e u sendo a função de utilidade que mede o quanto um item s é útil para um item c, deve-se, para cada item $c \in C$, escolher o item $s' \in S$ que maximiza o valor da função de utilidade 2.1 (Adomavicius and Tuzhilin, 2005).

$$\forall c \in C, s \in S : s'_c = argmax\ u(c,s) \tag{2.1}$$

Os sistemas de recomendação são classificados em categorias baseadas em como as recomendações são realizadas (Balabanovic and Shoham, 1997):

- Filtragem baseada em conteúdo;
- Filtragem colaborativa;
- Abordagem híbrida.

2.1.1 Filtragem Baseada em Conteúdo

Na filtragem baseada em conteúdo, a função de utilidade u de um item s para um usuário c é calculada de acordo com as utilidades de outros itens similares ao item s que já tenham sido definidas anteriormente. Por exemplo, em um sistema de recomendação de livros, o usuário recebe como recomendação livros que sejam similares a outros livros que ele gostou, ou seja, livros similares a livros que possuem utilidade alta para o usuário.

Esse tipo de sistema de recomendação é limitado pelas características que se consegue associar a um item. Quando tem-se itens com muita informação textual é mais fácil para um computador extrair o que é importante automaticamente. Em imagens, sons e vídeos esse processo é mais complicado, portanto muitas vezes é pedido para o usuário adicionar informações manualmente para representar itens multimídia, assim sendo possível efetuar a recomendação. É comum em vários *websites* populares o uso de metadados para representar características de fotos, vídeos, etc. Por exemplo, no YouTube os vídeos possuem título, descrição, categoria e palavras-chave, que são formas de representar o conteúdo do vídeo como texto.

Depois da análise dos seus conteúdos, da extração de características e metadados, dois itens diferentes podem ter suas características exatamente iguais, sendo impossível para o sistema diferenciar um do outro. Por exemplo, um sistema de recomendação, que represente um texto através de um vetor de N palavras mais frequentes, pode considerar um texto de um escritor famoso igual a um texto totalmente aleatório que contenha as mesmas N palavras mais comuns. Isto poderia ser muito ruim, pois o texto aleatório pode não ter nenhum valor para quem o receber como recomendação.

2.1.2 Filtragem Colaborativa

Ao invés de recomendar itens baseados em outros itens, um sistema de recomendação com filtragem colaborativa usa outros usuários para realizar o seu trabalho. A função de

utilidade u de um item s para um usuário c é calculada baseada na utilidade que outros usuários, similares ao usuário que pediu a recomendação, atribuíram ao item s. Usando novamente o exemplo de um sistema de recomendação de livros, o usuário recebe como recomendação livros que usuários parecidos com ele tenham gostado, ou seja, livros que tenham utilidade alta para quem é semelhante ao usuário que pediu a recomendação.

A maioria das técnicas utilizadas para detectar a semelhança entre os usuários se baseia na opinião deles sobre o mesmo item, por exemplo, se dois usuários gostaram de vários itens iguais, eles devem ter um gosto parecido por outros itens.

O sistema Grundy foi o primeiro sistema de recomendação com filtragem colaborativa (Rich, 1979) que propôs a criação de estereótipos baseados em poucas informações para construir modelos dos usuários. Um exemplo bastante conhecido de sistema de recomendação com filtragem colaborativa é o sistema de recomendação de livros da Amazon que ajuda os seus usuários a encontrarem livros interessantes de acordo com o que outros usuários acharam.

Como o sistema de recomendação com filtragem colaborativa exige interação do usuário com os itens, por exemplo, através da atribuição de notas, existem dois problemas comuns que aparecem nesse tipo de recomendação. O primeiro é quando um novo item é acrescentado ao sistema. Esse item no início não foi avaliado por ninguém e mesmo depois de certo tempo ele pode ter sido avaliado por poucos usuários, dessa forma o sistema não consegue recomendá-lo para muitos usuários, pois é necessário um certo número de avaliações para que o sistema consiga funcionar bem. O mesmo acontece com um usuário novo, como ele ainda não avaliou nenhum ou poucos itens o sistema não consegue encontrar outros usuários semelhantes a ele. Novamente o usuário precisa interagir com um maior número de itens para que o sistema funcione melhor.

2.1.3 Abordagem Híbrida

A abordagem híbrida é a combinação dos dois tipos de sistemas de recomendação anteriores. Ela é usada para tentar sanar os pontos fracos de cada um deles. Tem-se combinado as abordagens de diferentes maneiras na tentativa de conseguir melhores resultados que usando apenas uma ou outra.

A maneira mais simples de combinar os SRs mencionados é utilizar as abordagens

separadamente. Recomendam-se itens colaborativamente e baseado no conteúdo, podendo apresentar ambos os resultados ou de acordo com alguma métrica mostrar apenas os resultados do melhor método. O DailyLearner (Billsus and Pazzani, 2000) faz esse tipo de escolha entre os métodos de recomendação quando vai sugerir algo ao usuário.

Outra forma de combinar os tipos de SRs é adicionar características de um tipo onde o outro tipo costuma falhar. Por exemplo, em um sistema de recomendação colaborativa, pode-se através do conteúdo de um perfil preenchido pelo usuário encontrar usuários parecidos sem que eles tenham itens avaliados em comum. O Fab (Balabanović and Shoham, 1997) utiliza essa abordagem. Ou usar os usuários parecidos para filtrar inicialmente os itens que poderão ser recomendados através do conteúdo pelo sistema, assim ganhando desempenho e possivelmente evitando itens de baixa qualidade que nenhum usuário tenha gostado. Um exemplo dessa abordagem pode ser encontrado no trabalho de Soboroff and Nicholas (1999).

2.1.4 Sistemas de Recomendação para Engenharia de Software

Os desafios encarados por pessoas navegando por uma grande quantidade de informações são parecidos com os desafios enfrentados por desenvolvedores de software. Compare alguém tentando escolher entre centenas de hotéis em uma metrópole e um desenvolvedor tentando encontrar uma classe que resolverá seu problema entre centenas de classes em um componente.

Em Engenharia de Software os sistemas de recomendação são usados para ajudar os desenvolvedores a tomarem decisões e a encontrarem informações importantes para as atividades realizadas durante um projeto. Alguns exemplos de itens que são recomendados nesses sistemas são: código-fonte que precisa ser alterado para resolver algum problema ou adicionar uma nova funcionalidade, exemplos de uso de uma API, especialistas em uma determinada parte de projeto, documentação, solicitações de mudança, histórico do controle de versão, entre outros. É importante ressaltar que os sistemas de recomendação voltados para Engenharia de Software são bem mais recentes que os sistemas de recomendação utilizados na *web* que recomendam livros, filmes, músicas, notícias, etc. (Robillard *et al.*, 2010).

2.2 Compressão de Dados

Na ferramenta apresentada neste trabalho foram utilizadas técnicas de compressão de dados para efetuar recomendações. As técnicas desta área visam reduzir a representação da informação, com a finalidade de transmiti-la de forma mais eficiente. A busca por um compressor computacionalmente eficiente e que reduza consideravelmente a representação da informação é uma tarefa que vem sendo praticada há muito tempo. Isto devido ao fato de que quanto maior a compressão menor serão os requisitos de armazenamento e transmissão. Espera-se que o destinatário possa restaurar perfeitamente a mesma informação transmitida (compressão sem perdas), ou com distorções aceitáveis (compressão com perdas) (Salomon and Motta, 2009). Após a compressão de dados, pode-se obter a razão de compressão (RC), que é a divisão da quantidade de bits usados na codificação do texto original pela quantidade de bits usados na codificação do texto comprimido, conforme a equação 2.2.

$$RC = \frac{To}{Tc} \tag{2.2}$$

Onde, To e Tc são os tamanhos da mensagem original e comprimida, respectivamente. Quanto maior o valor da RC, maior é a compressão obtida, sendo que valores menores ou igual a 1 (um) significam que não houve compressão.

A Equação (2.3) é uma definição matemática para uma informação I associada a um símbolo x de uma mensagem. A informação associada ao símbolo é definida como sendo o logaritmo do inverso da probabilidade estimada para sua ocorrência, $P(x)$ (Shannon, 1948). Se a base 2 for utilizada para o cálculo, o resultado será dado em bits.

$$I(x) = \log_2(\frac{1}{P(x)}) \tag{2.3}$$

O cálculo da entropia H de uma fonte de informação pode ser definido como a informação média que a fonte produz, logo ela depende das probabilidades estimadas para a occorência dos símbolos (Salomon and Motta, 2009). Neste trabalho o cálculo da

entropia é utilizado para saber o quanto uma solicitação de mudança parece com outra. A Equação (2.4) representa o cálculo da média das informações individuais de cada símbolo x_i para uma mensagem de tamanho N.

$$H = \frac{1}{N} \sum_{i=1}^{N} I(x_i) \tag{2.4}$$

Várias técnicas de compressão de dados foram pesquisadas e constatou-se que não há um método de compressão que seja o melhor em qualquer contexto. Dependendo do arquivo a comprimir, faz-se necessário decidir sobre qual método de compressão utilizar (Salomon and Motta, 2009).

2.2.1 Modelos estatísticos

Modelos estatísticos são capazes de quantificar situações que envolvem incertezas. Como qualquer modelo, constituem uma representação simplificada de um objeto de estudo com maior ou menor fidelidade.

A compressão baseada em modelos estatísticos se divide em duas fases: criação de um modelo e a codificação. O modelo atribui probabilidades de acordo com a ocorrência dos símbolos de entrada e a codificação é feita baseada nestas probabilidades. Então símbolos que aparecem com mais frequência são associados a códigos menores para que uma boa compressão ocorra. Com melhores estimativas de probabilidade de ocorrência dos símbolos, menos símbolos serão necessários para a codificação da mensagem e melhor será a sua compressão.

Para obter melhores estimativas pode-se levar em consideração o contexto em que um símbolo está inserido, ou seja, os n símbolos que o precedem. A ideia do uso de contextos consiste em atribuir a probabilidade a um símbolo não dependendo apenas de sua frequência, mas do contexto no qual ele ocorre. Por exemplo, a probabilidade da letra h aparecer em um texto em inglês é de 5%. Entretanto, se o símbolo atual for a letra t, existe uma probabilidade muito maior do próximo símbolo ser a letra h, cerca de 30%, pois é comum no inglês a ocorrência das letras th juntas (Salomon and Motta, 2009).

2.3 Prediction By Partial Matching

O algoritmo Prediction by Partial Matching (PPM) é um método para compressão de dados baseado em modelos estatísticos. Ele é o estado-da-arte para compressão de dados sem perdas de informação (Salomon and Motta, 2009).

O modelo PPM utiliza um conjunto de no máximo k símbolos precedentes como contexto para estimar a distribuição de probabilidades condicionais para o próximo símbolo da mensagem. Dado um símbolo S e um contexto C_k de tamanho k, o PPM calcula a probabilidade condicional da ocorrência de S em C_k. Caso não haja ocorrência do símbolo S no contexto C_k, um símbolo especial de escape é codificado e é realizada uma nova busca no contexto C_{k-1}, que é a sequência de símbolos C_k reduzida de um símbolo. Caso o símbolo não seja encontrado em nenhum dos contextos, ele é codificado utilizando um modelo que considera equiprováveis todos os símbolos possíveis de ocorrer. Após a codificação do símbolo S, o modelo atualiza as probabilidades levando em consideração a ocorrência deste símbolo. Este processo é repetido para cada novo símbolo (Salomon and Motta, 2009).

A tabela 2.1 mostra o modelo gerado pelo PPM após receber informações da cadeia de caracteres “hocuspocus”, utilizando contexto com tamanho máximo $K = 2$. N indica o número de vezes que o símbolo apareceu num determinado contexto e P indica a probabilidade estimada para a ocorrência deste símbolo.

Para entender melhor o funcionamento vamos construir parte da tabela passo a passo. O primeiro símbolo da cadeia de caracteres “hocuspocus” é a letra h. Como ela não tem nenhum contexto é adicionada a letra h na tabela para $k = 0$ com N valendo 1, pois é a primeira vez que a letra h aparece, e também será adicionado um símbolo especial, o escape, com $N = 1$. Portanto, como temos dois símbolos, e cada um apareceu apenas uma vez, a probabilidade deles é de 1/2. O resultado pode ser visto na tabela 2.2.

O próximo simbolo é a letra o, ela também será adicionada em $k = 0$ com N valendo 1. Como este símbolo ainda não apareceu o escape será incrementado, ou seja, o valor de N para ele agora é 2. Atualizando as probabilidades tem-se a letra h com $P = 1/4$, a letra o com $P = 1/4$ e o escape com $P = 2/4$. A letra o é precedida pela letra h, então ela tem um contexto, assim a letra o será codificada no contexto h com probabilidade 1/2 e o escape também terá a mesma probabilidade. O modelo ficará como pode ser observado

Tabela 2.1 Modelo PPM após o processamento da cadeia de caracteres “hocuspocus”.

Contexto k=2			
Contexto	Símbolo	N	P
ho	c	1	1/2
	escape	1	1/2
oc	u	2	2/3
	escape	1	1/3
cu	s	2	2/3
	escape	1	1/3
us	p	1	1/2
	escape	1	1/2
sp	o	1	1/2
	escape	1	1/2
po	c	1	1/2
	escape	1	1/2
Contexto k=1			
h	o	1	1/2
	escape	1	1/2
o	c	2	2/3
	escape	1	1/3
c	u	2	2/3
	escape	1	1/3
u	s	2	2/3
	escape	1	1/3
s	p	1	1/2
	escape	1	1/2
p	o	1	1/2
	escape	1	1/2
Contexto k=0			
	h	1	1/16
	o	2	2/16
	c	2	2/16
	u	2	2/16
	s	2	2/16
	p	1	1/16
	escape	6	6/16

Tabela 2.2 Construção do modelo PPM para cadeia de “hocuspocus” - primeiro passo.

Contexto k=0			
	h	1	1/2
	escape	1	1/2

na tabela 2.3.

Tabela 2.3 Construção do modelo PPM para cadeia de “hocuspocus” - segundo passo.

Contexto k=1			
h	o	1	1/2
	escape	1	1/2
Contexto k=0			
	h	1	1/4
	o	1	1/4
	escape	2	2/4

O próximo passo adicionará a letra c ao modelo. O símbolo será adicionado no $k = 0$ com probabilidade 1/6 e o escape agora terá probabilidade 3/6. A letra c também será adicionada ao $k = 1$ no contexto o com probabilidade 1/2 e escape com a mesma probabilidade. Agora existem dois símbolos que precedem a letra c, então é possível adicioná-la ao modelo no $k = 2$, ela terá probabilidade 1/2 no contexto ho e escape também terá probabilidade 1/2. Veja o resultado do modelo na tabela 2.4

O procedimento continua até adicionar todos os símbolos da cadeia de caracteres “hocuspocus”. É importante lembrar que o escape não é atualizado quando um símbolo se repete num mesmo contexto.

Existe ainda um contexto especial $k = -1$ que é utilizado quando um símbolo não aparece no modelo em nenhum dos contextos. Nele todos os símbolos que não estão no modelo são equiprováveis.

No final do processo, o codificador aritmético gera uma sequência de símbolos codificados. Nela, a mensagem é representada inicialmente dentro do intervalo real [0,1). Este intervalo é alterado à medida que os símbolos e suas probabilidades são inseridos no codificador. Quanto menor for o tamanho dessa sequência em relação ao tamanho do texto de entrada, maior será a compressão obtida. Em relação ao tamanho da mensagem,

Tabela 2.4 Construção do modelo PPM para cadeia de “hocuspocus” - terceiro passo.

Contexto k=2			
ho	c	1	1/2
	escape	1	1/2
Contexto k=1			
h	o	1	1/2
	escape	1	1/2
o	c	1	1/2
	escape	1	1/2
Contexto k=0			
	h	1	1/6
	o	1	1/6
	c	1	1/6
	escape	3	3/6

quanto maior for seu tamanho, menor o intervalo e mais casas decimais são necessárias para sua representação (Witten *et al.*, 1987).

2.4 Latent Semantic Indexing

Uma deficiência fundamental nos métodos de recuperação de informação usando computadores é que as palavras usadas em uma pesquisa nem sempre são exatamente iguais às palavras do texto que possui a informação que queremos (Deerwester *et al.*, 1990). Isso acontece por causa da polissemia, ou seja, uma mesma palavra que possui vários sentidos, por exemplo, a palavra manga pode se referir à fruta manga ou à manga de uma camisa. Outro fator que causa problemas na recuperação de informação é a sinonímia, ou seja, palavras diferentes que possuem o mesmo significado, por exemplo as palavras bonita e bela.

A falha em muitos sistemas de indexação automática em resolver esses problemas pode ser atribuída a três fatores (Deerwester *et al.*, 1990). O primeiro é que a forma de identificar os termos indexados é incompleta. Os termos usados para descrever ou indexar um documento tipicamente contém apenas uma fração dos termos que os usuários procurarão, pois os documentos podem não possuir de fato todos os termos que os usuários vão procurar ou porque o procedimento de indexação intencionalmente remove

termos do documento.

O segundo fator é a falta de mecanismos adequados para lidar com a polissemia. Uma abordagem comum é usar especialistas para definir relações de polissemia entre as palavras e fazer o sistema tratar esses casos. Esta abordagem é muito custosa e ineficiente, pois muitos casos ainda poderão passar despercebidos e torna a técnica dependente de idioma. O sucesso da pesquisa é limitado pela incapacidade dos usuários de pensar em termos apropriados e pelo fato de que tais termos podem não ocorrer nos documentos ou não terem sido incluídos na indexação.

O terceiro fator é relacionado à forma como os sistemas de recuperação de informação funcionam. Normalmente esses sistemas tratam cada palavra independentemente umas das outras. Dessa forma pesquisar por muitos termos pode piorar muito os resultados se o usuário não souber escolher cuidadosamente o que procurar, pois o sistema tentará encontrar a ocorrência de todas as palavras, e seus sinônimos dependendo da técnica utilizada, no documento.

Para lidar com os problemas apresentados, o Latent Semantic Indexing (LSI) representa um documento pelo seu conceito que está oculto (latente) nos termos que o formam. Este conceito oculto não é apenas uma relação de muitos para muitos entre termos e conceitos, mas depende fortemente da coleção de documentos conhecidos pelo sistema (Papadimitriou *et al.*, 1998).

O LSI ao invés de gravar apenas as palavras-chave contidas em um documento, examina a coleção de documentos como um todo, para verificar quais possuem palavras iguais. O LSI considera documentos que possuem muitas palavras em comum semanticamente próximos e os documentos com poucas palavras em comum semanticamente distantes. Este método simples se assemelha bastante como os seres humanos procuram por conteúdo. O LSI, mesmo não sabendo nada sobre o significado das palavras, é bastante eficiente apenas utilizando os padrões de ocorrência das palavras (Yu *et al.*, 2008).

Quando é feita uma procura em um banco de dados indexado pelo LSI, o motor de pesquisa procura por valores de similaridade que foram calculados para cada palavra contida nos documentos, retornando os documentos que o motor acha que melhor estejam adequado à procura. Dois documentos podem ser semanticamente parecidos mesmo se eles não compartilharem algumas palavras-chave, pois o LSI não requer uma correspondência exata para retornar resultados úteis, podendo acontecer casos em que ele retorne

documentos relevantes que não contenham nenhuma palavra-chave procurada.

Por exemplo, supondo o uso do LSI em uma coleção de notícias, um usuário poderia realizar a busca pela palavra “carro” para encontrar notícias relacionadas a carros. Nas notícias do banco de dados é comum em textos que contenham a palavra “carro” também existir a palavra “trânsito”. Ao realizar a pesquisa serão retornados resultados de notícias com a palavra “carro”, com as palavras “carro” e “trânsito” e também podem ser retornadas notícias apenas com a palavra “trânsito” mesmo que a palavra “carro” não esteja presente. O motor de pesquisa aprende que os termos “carro” e “trânsito” estão relacionados dado um número suficiente de documentos para ele examinar.

A linguagem natural possui muita redundância, visto que existem muitas palavras sem significado semântico em um texto, por exemplo, preposições, conjunções, verbos auxiliares, entre outros. O primeiro passo para o uso do LSI é eliminar este tipo de palavra do texto dos documentos.

Em seguida deve ser gerada uma matriz termo-documento, que nada mais é que uma grande tabela com os documentos na primeira coluna e os termos na primeira linha. Para cada palavra que aparece num documento uma marcação é feita na célula da tabela correspondente ao documento e a palavra de acordo com o número de vezes que ela aparece. Ao final deste processo tem-se uma matriz com as informações de todos os documentos e através dela é possível saber quais palavras aparecem em cada documento e quais documentos contêm uma palavra.

Cada documento é representado por um vetor num espaço com o número de dimensões igual ao número de palavras existentes na matriz. Com mais de três dimensões é impossível visualizar esses vetores, mas pode-se notar que quanto mais os documentos possuírem palavras iguais, mais os vetores serão parecidos. O contrário também é válido, documentos com poucas palavras em comum serão representados por vetores bem distintos.

O próximo passo é chave no funcionamento do LSI, nele as dimensões do espaço onde os documentos estão representados são reduzidas, tentando manter as distâncias relativas entre os vetores, usando o algoritmo Singular Value Decomposition (SVD) (Grund, 1979). Esta perda de informações pode ser boa para o LSI, pois vetores similares são agrupados e vetores diferentes ficam cada vez mais distantes uns dos outros.

Para calcular a similaridade entre os vetores basta calcular o cosseno do ângulo entre

eles. Quanto maior o cosseno, ou seja, menor o ângulo, mais parecidos são os vetores e quanto menor o cosseno, ou seja, maior o ângulo, menos parecidos são os vetores. É importante ressaltar que após a aplicação do SVD alguns vetores podem passar a ter valores negativos em alguns eixos, logo o cosseno calculado não se restringe a valores positivos.

2.5 Trabalhos Relacionados

Nesta seção serão apresentados trabalhos relacionados ao que foi desenvolvido, seus objetivos, técnicas utilizadas e os artefatos usados nas recomendações.

2.5.1 Hipikat

O objetivo do Hipikat (Cubranic *et al.*, 2005) é ajudar novatos em um projeto de desenvolvimento de software através de recomendações de código-fonte, solicitação de mudança, e-mails de uma lista de discussão, documentação e pessoas que sejam relevantes à tarefa que o desenvolvedor esteja realizando. A ferramenta tem o mesmo objetivo do Mentor, tornar a memória do projeto acessível aos desenvolvedores sem a necessidade de um mentor real para guiá-los. O Hipikat usa o Latent Semantic Indexing (LSI) para analisar a semelhança dos textos, que segundo os autores do trabalho é o principal gargalo da ferramenta por causa do seu desempenho ruim. O Mentor usa o PPM, assim espera-se conseguir melhores resultados e um melhor desempenho nesse tipo de recomendação.

O Hipikat utiliza os seguintes artefatos como entrada:

- Código-fonte
- Pedidos de mudança
- E-mails
- Documentos do projeto
- Pessoas

E recomenda os seguintes artefatos:

- Código-fonte
- Pedidos de mudança
- E-mails
- Documentos do projeto
- Pessoas

Os artefatos são recomendados através da atribuição de relações entre eles. Por exemplo, um documento é ligado a outro documento que é similar a ele, assim como os dados de uma revisão que implemente a solução de uma solicitação de mudança são interligados. A seguir serão descritos os módulos da ferramenta e suas técnicas para atribuir os diferentes tipos de relações.

Log Matcher: é comum nos projetos especificar na mensagem do *commit* a que solicitação de mudança aquela operação se refere, por exemplo, a mensagem *"Fix the bug #1234"* pode ser usada no *commit* que resolve a solicitação de mudança de número 1234. Quando um *commit* é realizado no controle de versão esse módulo usa uma série de expressões regulares para tentar identificar na mensagem do *commit* se ele se refere a alguma solicitação de mudança.

Activity Matcher: normalmente após realizar o *commit* que resolve uma solicitação de mudança o desenvolvedor altera o estado da solicitação para resolvido. Ao invés de tentar associar um *commit* a uma solicitação de mudança de acordo com a mensagem de *commit*, o Activity Matcher utiliza padrões de utilização de ferramentas de gerenciamento de solicitações de mudança. Este módulo monitora as mudanças na ferramenta de gerenciamento de solicitações de mudança e procura por *commits* que foram feitos num intervalo de tempo próximo à mudança de estado da solicitação. Todos os *commits* feitos numa janela de seis minutos são associados à solicitação de mudança marcada como resolvida. O Activity Matcher complementa o trabalho do Log Matcher.

Text Similarity: esse módulo funciona em duas fases. A primeira é a indexação do texto de um novo artefato, que é transformado em um vetor documento que possui dimensões correspondentes às palavras no vocabulário. Em seguida o vetor documento é projetado num espaço semântico usando o Latent Semantic Indexing (LSI). Na segunda fase, o módulo calcula o cosseno entre vetores para inferir ligações de similaridade entre os artefatos.

CVS Check-in Package Matcher: a função deste módulo é agrupar diferentes *commits* realizados num período próximo, numa janela de seis minutos, como sendo um só. Para ser agrupado também é necessário que o *commit* seja do mesmo autor e tenha a mesma mensagem.

Thread Matcher: este módulo agrupa mensagens de uma *thread* de uma lista de discussão através do campo do cabeçalho das mensagens que especifica a que mensagem ela está respondendo. Assim, é possível recomendar mensagens de toda *thread* e não apenas uma mensagem individual.

2.5.2 Strathcona

O Strathcona (Holmes *et al.*, 2006) tem como objetivo ajudar desenvolvedores a encontrarem exemplos de código-fonte, que usem uma API, através de heurísticas baseadas na maneira como os desenvolvedores pesquisam por esses artefatos. Estes exemplos podem ser usados pelos desenvolvedores para aprender como interagir com uma API. Entretanto o Strathcona não recomenda código-fonte para ajudar na resolução de uma solicitação de mudança, o objetivo da ferramenta é um pouco diferente do objetivo do Mentor.

O Strathcona utiliza os seguintes artefatos como entrada:

- Código-fonte

E recomenda os seguintes artefatos:

- Código-fonte

O Strathcona funciona da seguinte forma. Um trecho de código-fonte é enviado pelo usuário para o servidor e ele retorna exemplos de uso de código-fonte relacionados ao código-fonte enviado. Os exemplos, que são trechos de código-fonte na linguagem que estiver sendo usada no projeto, precisam ser adicionados previamente no servidor para a ferramenta funcionar.

Tanto o código-fonte adicionado quanto o enviado como entrada da pesquisa passam por um processo de extração de informações sobre ele próprio e sobre o contexto em que

está inserido. O autor chama estas informações de contexto estrutural, que é composto de:

- Assinatura dos métodos que estão dentro do código-fonte enviado;
- O nome dos tipos que declaram os métodos do item anterior (tipos declarados);
- O nome e o tipo dos campos declarados pelos tipos declarados;
- Os tipos, métodos e campos referenciados no código-fonte enviado e
- As superclasses dos tipos declarados.

Quando o código-fonte enviado pelo usuário chega ao servidor, ele tenta encontrar exemplos similares para recomendar ao usuário. Para encontrar os exemplos, quatro heurísticas são usadas, pois tentar apenas encontrar contextos estruturais iguais ao contexto do código-fonte enviado pode resultar num resultado pobre. As heurísticas são:

- Herança: localiza exemplos no repositório que contenham tipos declarados que estendam de um mesmo tipo;
- Chamada: encontra exemplos que chamam métodos iguais;
- Uso: localiza exemplos que contenham tipos utilizados iguais aos do código-fonte enviado. Um tipo é considerado utilizado se ele for instanciado, um método dele for chamado, um campo for referenciado ou se ele for usado como parâmetro.
- Referência: retorna exemplos com declarações de métodos que se referem a um mesmo campo.

2.5.3 Codetrail

O Codetrail (Goldman and Miller, 2009) automatiza a procura de conteúdo relacionado ao projeto na *web*. A ferramenta associa o conteúdo das páginas *web* visitadas pelo desenvolvedor ao código-fonte do projeto automaticamente usando algumas heurísticas, por exemplo, relacionando URLs a palavras-chave. Associações entre páginas web e código-fonte também podem ser criadas manualmente no Codetrail. Apesar de ser uma

abordagem interessante, ela não tem como foco encontrar relações entre partes do próprio projeto como estamos propondo com o Mentor.

O Codetrail utiliza os seguintes artefatos como entrada:

- Código-fonte
- Conteúdo de sites visitados
- Histórico de sites visitados

E recomenda os seguintes artefatos:

- Código-fonte
- Documentação

O algoritmo de Smith and Waterman (1981) é utilizado neste trabalho para fazer a correspondência entre trechos de código-fonte escritos pelo programador direto no ambiente de programação e códigos-fontes que são visualizados através da navegação usando um navegador.

2.5.4 Codebook

O Codebook (Begel *et al.*, 2010) é um framework para mineração de repositórios, que utiliza um grafo que relaciona pessoas e artefatos em um projeto de desenvolvimento de software. No artigo duas aplicações foram criadas para testar o Codebook. O Hoozizat é uma ferramenta usada para encontrar especialistas e o Deep Intellisense é um *add-in* do Visual Studio que exibe uma lista de eventos ordenados cronologicamente relacionados a algum símbolo selecionado no código-fonte. O Codebook poderia ser usado para recomendar código-fonte de acordo com uma solicitação de mudança, mas, mesmo que criassem uma ferramenta que utilizasse o grafo do framework para esta finalidade, ela poderia não utilizar o PPM, que faz parte do processo de recomendação do Mentor.

O Codebook utiliza os seguintes artefatos como entrada:

- Código-fonte

- Pedidos de mudança
- E-mails
- Sites
- Pessoas

E recomenda os seguintes artefatos:

- Código-fonte
- Pedidos de mudança
- E-mails
- Sites
- Pessoas

O Codebook utiliza uma ideia muito parecida com o Hipikat para relacionar e recomendar artefatos de um projeto de software. Os artefatos são representados como nós de um grafo e a relação entre eles são as arestas do grafo. Para encontrar artefatos relacionados a um artefato de entrada basta apenas caminhar pelo grafo. Se o usuário necessitar encontrar uma pessoa relacionada a uma solicitação de mudança, o Codebook pode procurar, por exemplo, uma relação "alterado por" que liga uma solicitação de mudança a uma pessoa.

Para criar o grafo do Codebook vários *crawlers* podem ser implementados para analisar diferentes tipos de dados e depois gravar os artefatos e as relações entre eles no banco de dados. Também são gravados no banco de dados alguns metadados sobre os artefatos que são usados no mecanismo de busca do Codebook. Os *crawlers* apresentados no trabalho foram:

- Código-fonte: este *crawler* analisa as revisões do controle de versão, os arquivos alterados nelas e o código-fonte alterado nos arquivos. Os identificadores que aparecem no código-fonte são armazenados e relações de herança, chamada, referência, uso, entre outras, podem ser gravadas no banco de dados;

- Pessoas: informações das pessoas que participam do projeto são analisadas e gravadas no banco de dados sendo relacionadas por hierarquia no projeto;
- Pedidos de mudança: campos das solicitações de mudança podem ser relacionados a outros artefatos no grafo, isto pode ser definido por especialistas que estejam usando o Codebook. Por exemplo, pode ser definido que o campo que indica quem alterou a solicitação de mudança está relacionado com os artefatos Pessoas. Campos textuais das solicitações de mudança são analisados por expressões regulares para tentar encontrar relação com o código-fonte;
- Alusão textual: quando um nome ou e-mail aparece em algum campo textual, o artefato é relacionado com a pessoa correspondente.

2.5.5 Moin and Khansari

Dada uma solicitação de mudança não resolvida o trabalho de Moin and Khansari (2010) utiliza o classificador Support Vector Machine (SVM) (Vapnik, 1998) para sugerir a parte da hierarquia do código-fonte do projeto que provavelmente está relacionada com a solicitação não resolvida. As sugestões são baseadas em solicitações de mudanças feitas no passado. Os autores do trabalho não mencionam grandes problemas com a técnica utilizada, entretanto este trabalho possui uma diferença crucial para o Mentor, pois ele recomenda diretórios inteiros onde os arquivos estão contidos e não diretamente os arquivos. Esta recomendação pode continuar sendo muito vaga se um diretório possuir muitos arquivos.

Este trabalho utiliza os seguintes artefatos como entrada:

- Pedidos de mudança

E recomenda os seguintes artefatos:

- Código-fonte

O SVM é um algoritmo de aprendizagem muito usado em trabalhos de classificação. No trabalho de Moin e Khansari o SVM é treinado usando os dados de solicitações de mudança que já foram resolvidas no passado, para que sejam encontrados solicitações semelhantes e assim localizar os diretórios com código-fonte que devem ser modificados.

Tabela 2.5 Artefatos utilizados como entrada nos trabalhos relacionados.

Trabalho	Código-fonte	Pedidos de mudança	E-mails	Documentação	Sites	Pessoas
Hipikat	x	x	x	x		x
Strathcona	x					
Codetrail	x				x	
Codebook	x	x	x		x	x
Moin and Khansari		x				
Mentor		x				

Tabela 2.6 Artefatos recomendados nos trabalhos relacionados.

Trabalho	Código-fonte	Pedidos de mudança	E-mails	Documentação	Sites	Pessoas
Hipikat	x	x	x	x		x
Strathcona	x					
Codetrail	x			x		
Codebook	x	x	x		x	x
Moin and Khansari	x					
Mentor	x	x				

2.5.6 Análise Crítica

Um projeto muito grande pode ter centenas de classes e milhares de linhas de código, o que torna o trabalho de navegação no código-fonte árduo. Eliminar a maior quantidade de ruído possível e limitar o número de arquivos que o desenvolvedor precisará procurar é de extrema relevância.

De acordo com os experimentos realizados em Cubranic *et al.* (2004) recomendar artefatos de acordo com uma tarefa que será realizada é uma boa abordagem, pois os desenvolvedores parecem se importar mais com a solução imediata de um problema do que com o entendimento geral do sistema. Isso pode acontecer pela falta de tempo por causa de prazos apertados para entrega do software que são comumente vistos.

Quando um desenvolvedor busca por artefatos recomendados, ele está querendo acessar o conhecimento construído durante o desenvolvimento do projeto para que ele não precise reinventar uma solução para algo que já foi resolvido. As recomendações servem para indicar para o desenvolvedor o ponto de partida para a solução de um problema. A solução de uma tarefa pode levar muito tempo e pode não alcançar a qualidade de outra solução que já estava sendo usada, que foi desenvolvida e testada durante um período bem maior, pois já estava implementada a mais tempo.

Através da análise dos trabalhos relacionados foi possível perceber alguns padrões de uso dos artefatos pelos desenvolvedores. Uma grande ênfase é dada ao código-fonte dos projetos. Isto se deve ao fato de que as recomendações em muitos casos apresentados são usadas para o entendimento do código-fonte que deverá ser modificado para resolver alguma solicitação de mudança. As recomendações são usadas basicamente para ajudar o entendimento do desenvolvedor sobre alguma parte do projeto. O estudo qualitativo apresentado em Cubranic *et al.* (2004) mostra que as recomendações são pedidas com muito mais frequência no início da solução de uma tarefa, reforçando ainda mais a ideia de que recomendações são utilizadas para fornecer informações iniciais para que um problema seja resolvido.

Uma maneira muito utilizada para auxiliar no entendimento do código-fonte é apresentar códigos-fonte similares relacionados à solução do problema que o desenvolvedor precisa resolver. Documentos do projeto também são artefatos importantes para ajudar os desenvolvedores, eles também são recomendados por alguns dos trabalhos relacionados.

Nas tabelas 2.5 e 2.6 é mostrado um resumo dos artefatos utilizados nos trabalhos relacionados.

Alguns benefícios trazidos pela recomendação de artefatos baseados em tarefas já são conhecidos, entretanto as recomendações ainda podem ser melhoradas. Trabalhar para aumentar a precisão, a abrangência e a utilidade em geral das recomendações feitas é ainda muito importante. Também existem questões de desempenho e escalabilidade, pois existem grandes projetos que podem tornar os métodos de recomendação atuais inviáveis. Enfim, melhorar a qualidade e desempenho das técnicas de recomendação de artefatos baseados em tarefas em um projeto de desenvolvimento de software é um trabalho de extrema relevância e que poderá ajudar muitos desenvolvedores.

No Mentor é usado o PPM no processo de recomendação de código-fonte, técnica que não é utilizada em nenhum dos trabalhos analisados. Desta forma tentamos melhorar a qualidade dos resultados do SR para que os desenvolvedores consigam obter melhores recomendações.

2.6 Conclusão

Durante um projeto de desenvolvimento de software uma grande quantidade de artefatos é produzida, por exemplo, documentação, código-fonte, solicitações de mudança, discussões sobre o projeto, entre outros. Os sistemas de recomendação usados na Engenharia de Software ajudam as pessoas que participam de um projeto a tomarem decisões que dependem da análise de todo esse material criado.

Com a análise dos trabalhos relacionados foi possível identificar os problemas que os sistemas de recomendação para Engenharia de Software que recomendam código-fonte tentam resolver, quais as técnicas mais comuns e quais os problemas ainda enfrentados na criação desse tipo de ferramenta.

Nessa dissertação é proposto um sistema de recomendação de código-fonte de acordo com solicitações de mudança em um projeto de desenvolvimento de software. Usando esse SR a quantidade de informação que o usuário precisa navegar é reduzida e o SR passa a funcionar como um mentor que aponta para o usuário quais os arquivos mais prováveis precisarão ser alterados para resolver uma solicitação de mudança. Isso diminui também a necessidade de comunicação com outros integrantes, reduzindo, possivelmente, alguns

dos problemas causados pela interrupção para pedir ajudar a um colega, principalmente se ele estiver distante geograficamente.

3

Mentor

Este trabalho propõe um sistema de recomendação de código-fonte baseado no conteúdo de solicitações de mudança para resolver os problemas discutidos no Capítulo 1. O Mentor é uma ferramenta web na qual os usuários podem navegar entre as solicitações de mudança do projeto, ver os dados relacionados a elas e pedir recomendação de código-fonte para ajudar na resolução de uma solicitação de mudança. A tela inicial do Mentor com a listagem das solicitações de mudanças pode ser vista na Figura 3.1.

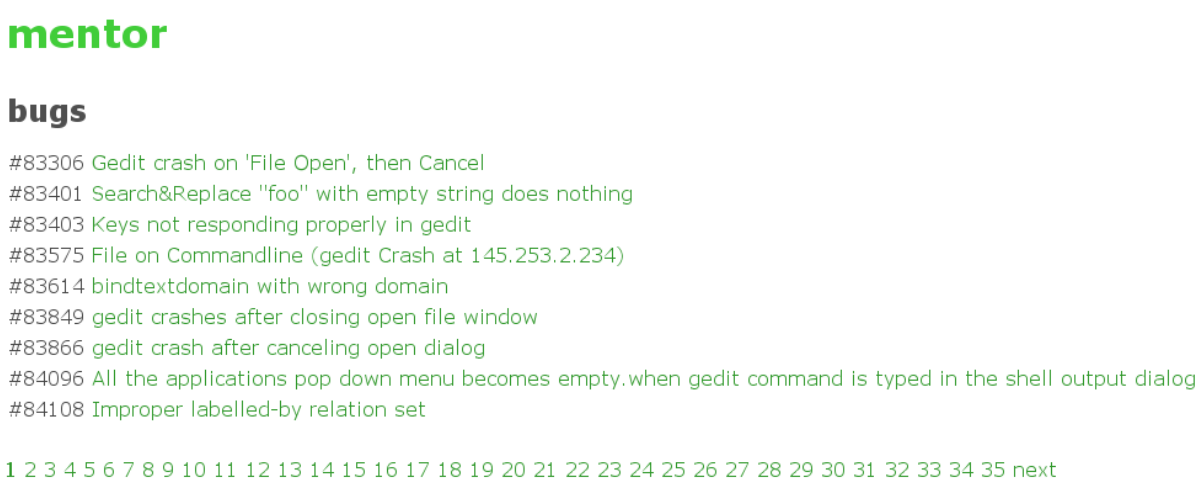

Figura 3.1 Tela inicial do Mentor

Utilizando as recomendações o desenvolvedor tem um filtro que o ajuda a encontrar mais facilmente as informações necessárias para resolver uma solicitação de mudança, evitando a perda de muito tempo, o seu desgaste durante o processo de entendimento do

problema antes do início da resolução e também evita a alocação desnecessária de um mentor para ajudá-lo.

Neste capítulo serão apresentados os requisitos, a arquitetura, os algoritmos usados e como o Mentor executa uma recomendação.

3.1 Requisitos

Foram vistos anteriormente alguns trabalhos semelhantes na área de sistemas de recomendação. A partir deles e dos problemas que o Mentor se propõe a resolver foram definidos os requisitos funcionais da ferramenta:

RF1. Carregar dados de solicitações de mudança de projetos externos: o Mentor não é um gerenciador de projetos, ele é um sistema de recomendação que funciona usando dados de projetos de software, então ele precisa carregar dados já existentes para funcionar, pois não é possível criar solicitações de mudança diretamente na ferramenta. Assim é fundamental que existam meios de executar esse carregamento e que ele possa suportar diferentes tipos de projetos, já que existem diversas ferramentas para gerenciar projetos e elas não seguem uma forma padrão para armazenar todos os seus dados.

RF2. Carregar dados de controle de versão externos: além de não gerenciar solicitações de mudança o Mentor também não gerencia os dados do controle de versão, portanto eles também precisam ser carregados no sistema de recomendação para que ele possa funcionar corretamente, dessa forma também é necessário criar um meio para carregar estes dados no Mentor.

RF3. Listagem das solicitações de mudança: ferramentas que gerenciam solicitações de mudança comumente exibem uma lista de solicitações para que o usuário possa navegar entre eles. O Mentor é um sistema de recomendação que poderia ser incorporado a um gerenciador de projetos, mas atualmente ele apenas exibe informações sobre as solicitações de mudança e recomenda código-fonte.

RF4. Exibição das informações de uma solicitação de mudança: apenas listar as solicitações não é suficiente, também é necessário analisar o seu conteúdo. O sumário, a descrição completa e os comentários feitos pelos usuários são muito importantes

para a análise da solicitação de mudança, assim é possível saber com mais detalhes do que ela realmente trata.

RF5. Listar solicitações de mudança similares e a solução de cada uma: este é o passo final da recomendação, o desenvolvedor deve receber uma lista de solicitações de mudança similares à solicitação de entrada e também deve ser possível ter acesso aos arquivos que foram alterados para resolver cada solicitação.

Os requisitos não funcionais do Mentor são:

NF1. Apresentar desempenho adequado: um problema comum em sistemas de recomendação deste tipo é o custo computacional excessivo para executar uma recomendação. Os algoritmos para este tipo de tarefa são pesados e exigem muito do computador. Por isso é necessário desenvolver algumas técnicas para que o processamento seja feito poucas vezes, assim tornando viável a recomendação de um grande número de solicitações de mudança.

NF2. Possibilitar a troca da técnica para atribuição da similaridade: o Mentor pode funcionar com diferentes técnicas de atribuição de similaridade de textos, dessa forma a ferramenta não está amarrada apenas ao uso de determinados algoritmos de recomendação. Assim é possível desenvolver diferentes técnicas de recomendação e incorporá-las ao Mentor sem que seja necessário alterar todo o sistema.

3.2 Arquitetura

A arquitetura de software é uma representação abstrata da estrutura e organização dos componentes e módulos que compõem um software e como eles interagem para formar um sistema (Kruchten *et al.*, 2006). De acordo com Clements *et al.* (2003), a arquitetura de software pode ser vista como o que faz um conjunto de partes funcionar com sucesso quando fazem parte de um todo.

A arquitetura do Mentor foi projetada para ser possível acoplar e desacoplar os componentes tornando o sistema de recomendação independente das tecnologias usadas para gerenciar projetos e para que diferentes tipos de algoritmos de recomendação pudessem ser usados. O Mentor também precisa funcionar para uma grande quantidade de dados.

As subseções a seguir apresentam uma visão geral da arquitetura, os principais módulos do Mentor, como eles estão organizados e como eles interagem.

3.2.1 Visão geral

A arquitetura geral do Mentor é apresentada na Figura 3.2. O framework Django foi utilizado no desenvolvimento da ferramenta, então sua organização é baseada na arquitetura MVC (Model-View-Controller). A representação das entidades é definida na camada *model*, as *views* cuidam da apresentação dos dados para o usuário e a camada *controller* requisita e atualiza os dados dos *models*, processa esses dados e os envia para serem exibidos nas *views*.

Três componentes foram criados para auxiliar no funcionamento do Mentor, são eles: o *Parser*, o *Matcher* e o *Similarity Assigner*. Eles se comunicam diretamente com a camada *model* para realizar tarefas importantes.

3.2.2 Model

Os *models* são responsáveis pela representação das entidades de dados usadas no Mentor. Por exemplo, existe um *model* para representar uma solicitação de mudança e outro para representar as revisões do controle de versão.

3.2.3 View

As *views* do Mentor tratam de exibir as informações de **RF3**, **RF4** e **RF5**. A *view* do **RF3** exibe uma lista com as solicitações de mudança e utiliza uma paginação para facilitar a navegação quando o número de pedidos de mudança é muito grande.

Ao clicar em um pedido de mudança da listagem o Mentor alterna para a *view* do **RF4**, Figura 3.3, que exibe os detalhes do pedido de mudança. Além do sumário que era possível visualizar na listagem geral de todos os pedidos, na *view* das informações detalhadas é possível ver a descrição do pedido de mudança, os comentários e uma opção para pedir recomendação de soluções. Esta última opção vai acionar a *view* de **RF5**.

O Mentor exibe para o usuário um conjunto de solicitações de mudança semelhantes a

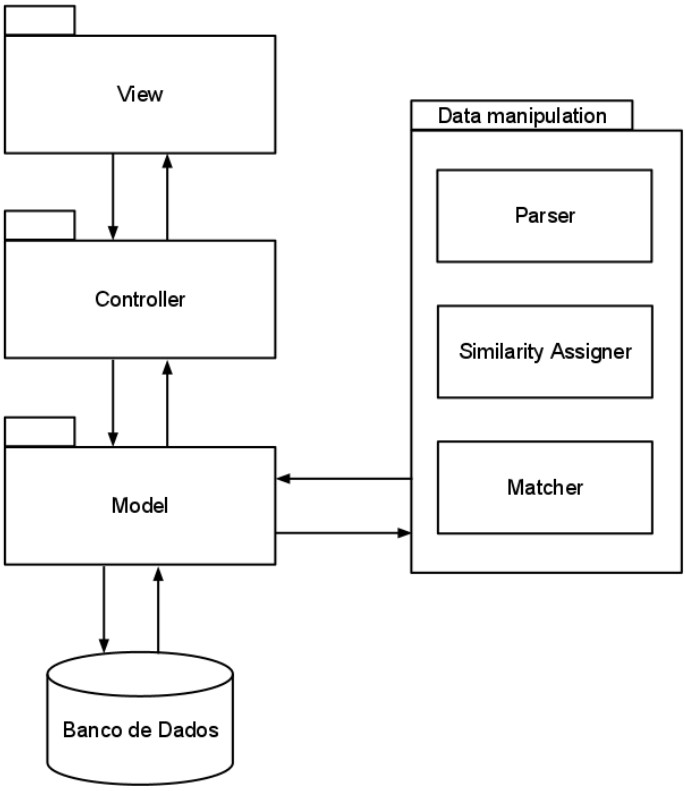

Figura 3.2 Arquitetura do Mentor

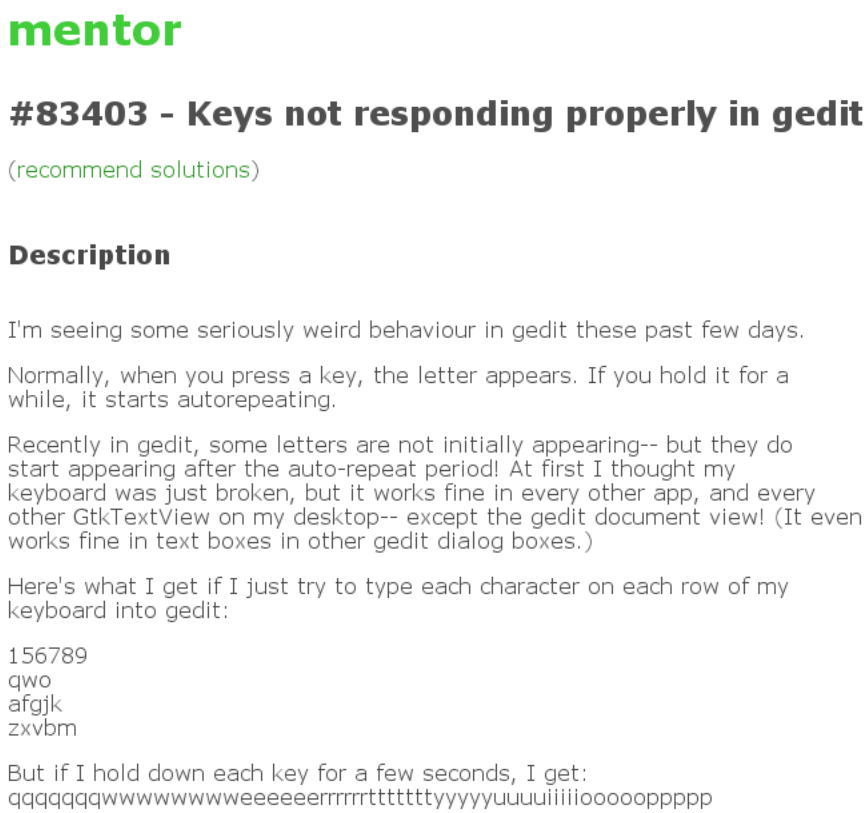

Figura 3.3 Informações detalhadas de um pedido de mudança

solicitação a qual ele pediu recomendação, assim o usuário pode julgar quais solicitações são realmente parecidas. Ao clicar em uma solicitação, o Mentor exibe uma página com mais informações da solicitação, incluindo as revisões associadas a ela e consequentemente a lista dos arquivos modificados para resolvê-la. Esta é uma visão simplificada de como funciona a recomendação do Mentor. Todas essas telas estão ligadas às *views* que satisfazem **RF5**. Veja as telas na Figura 3.4 e na Figura 3.5

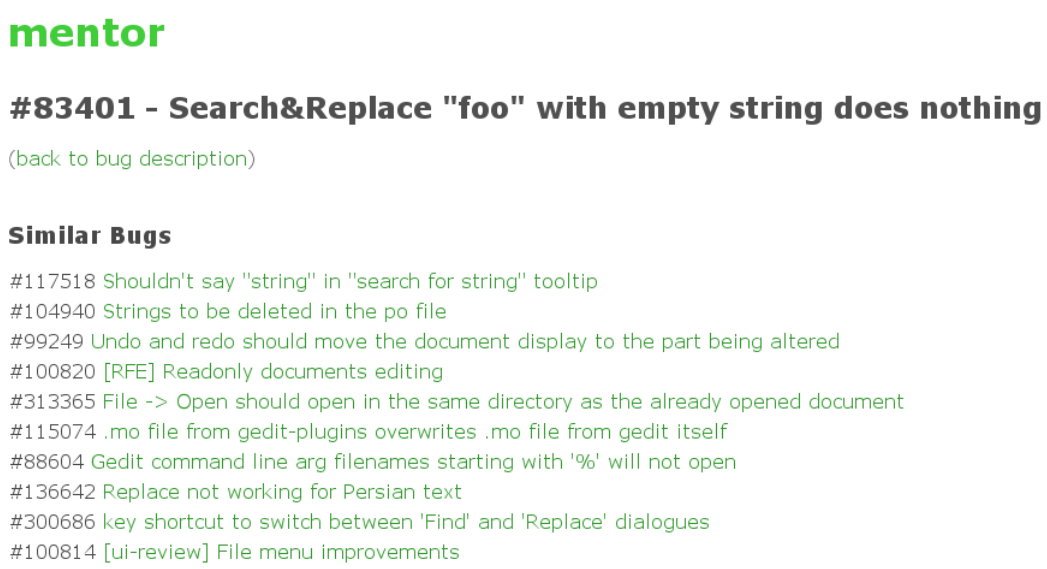

Figura 3.4 Pedidos de mudança similares

Figura 3.5 Arquivos modificados

3.2.4 Controller

O *controller* é responsável por acessar os *models*, tanto para pegar dados quanto para gravar, processar estes dados e enviar para as *views*. Para manter a velocidade do Mentor alta, o *controller* não interage com os componentes de processamento mais pesado que serão descritos a seguir, ele apenas busca dados nos modelos, faz algumas operações simples e os envia para as *view*. Isto evita que a cada clique do usuário numa funcionalidade do sistema o Mentor precise executar muitas instruções que sobrecarregariam rapidamente o computador que o estivesse executando.

3.2.5 Parser

O componente *Parser* é responsável por transformar dados de diferentes tipos de projetos no formato utilizado pelo Mentor.

Diferentes projetos de software podem usar diferentes ferramentas para gerenciar as solicitações de mudança e para controle de versão e cada ferramenta pode ter sua própria forma peculiar de armazenar os dados. Assim não é possível garantir que exista um padrão para o Mentor trabalhar, porque a organização dos dados é dependente de ferramenta, então é necessário ter um meio para transformar os dados para a forma utilizada pelo Mentor.

Diferentes tipos de *parsers* podem ser implementados, basta que no final ele gere uma lista de JSONs (JavaScript Object Notation), um padrão aberto para troca de dados, com os campos apresentados nas listagens 1, 2 e 3.

O campo “pk” é a chave primária, nesse caso um ID que começa de 1 e vai sendo incrementado a cada novo registro. O campo “model” é fixo, seu valor sempre será “bugs.bug”, pois este é o nome do modelo criado no sistema para representar uma solicitação de mudança. Dentro do campo “fields” tem-se vários campos, que representam os dados da solicitação de mudança. O campo “bug_id” é o número da solicitação de mudança, estes números variam de acordo com o projeto que estiver sendo carregado. Os campos “summary”, “description” e “comments” têm como valores o sumário da solicitação, a descrição e os comentários, respectivamente. O último campo é o “close_time”, ele recebe a data de quando a solicitação foi fechada.

```
{
    "pk" : bug_pk,
    "model" : "bugs.bug",
    "fields" : {
        "bug_id" : bug_id,
        "summary" : summary,
        "description" : description,
        "comments" : comments,
        "close_time" : close_time,
    }
}
```

Listagem 1: JSON para o modelo Bug

```
{
    "pk" : revision_pk,
    "model" : "bugs.revision",
    "fields" : {
        "revision_id" : revision_id,
        "message" : message,
        "time" : commit_time
    }
}
```

Listagem 2: JSON para o modelo Revision

O campo “pk” para as revisões funciona da mesma forma que o “pk” das solicitações de mudança. Ele é uma chave primária que começa com o valor um e vai sendo incrementado. O campo “model” também é fixo, mas nesse caso recebe o valor “bugs.revision”, que é o nome do modelo para as revisões. Dentro de “fields” tem-se três campos, “revision_id” é o identificador da revisão, “message” é a mensagem enviada pelo desenvolvedor ao efetuar o *commit* e “time” é a data em que a revisão foi criada.

```
{
    "pk": 0,
    "model": "bugs.revisionfile",
    "fields": {
        "path": "/diretorio/do/arquivo",
        "revision": 1
    }
}
```

Listagem 3: JSON para o modelo RevisionFile

Cada revisão pode ter vários arquivos associados. O JSON para o modelo dos arquivos das revisões contém o campos “pk” que é a sua chave primária e segue o mesmo padrão dos outros modelos e o campo “model” tem sempre o valor “bugs.revisionfile”. Em “fields” tem-se dois outros campos, o “path” que recebe o valor do caminho para o arquivo que o modelo representa e o “revision” que tem a chave primária da revisão a que o arquivo está associado.

O JSON é utilizado, pois com esse formato é possível adicionar dados no banco do projeto utilizando Django facilmente. Para cada solicitação de mudança, cada revisão e cada arquivo associado a uma revisão é gerado um JSON, eles são agrupados em uma lista e gravados em arquivo. Para carregar os dados no banco de uma aplicação Django usa-se o comando:

```
python manage.py loaddata nome_do_arquivo
```

Listagem 4: Comando para carregar o banco de dados

O modelo entidade-relacionamento da ferramenta pode ser visto na figura 3.6

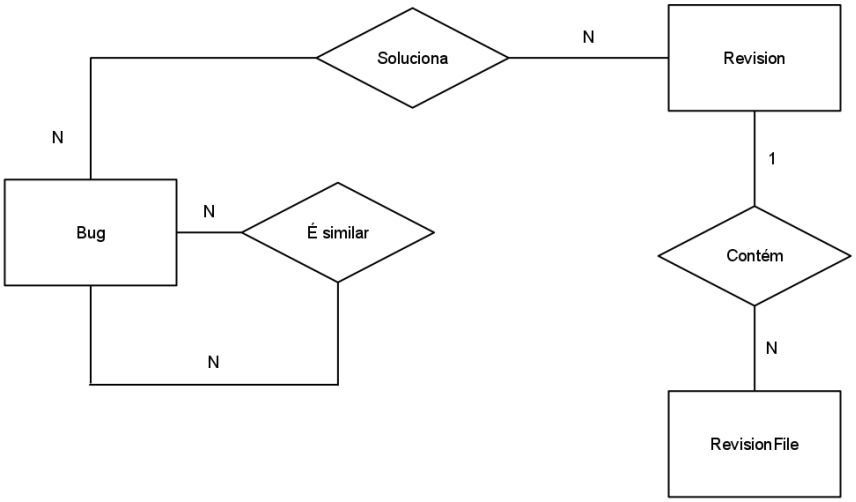

Figura 3.6 Modelo Entidade Relacionamento

3.2.6 Matcher

Para efetuar uma recomendação o Mentor precisa saber a relação entre uma solicitação de mudança e uma revisão do controle de versão. Entretanto não existe uma relação explícita de qual revisão resolve qual solicitação de mudança, inclusive existem casos em que uma revisão não está relacionada a nenhuma solicitação de mudança. Para atribuir essas relações é necessário executar o *Matcher*, que analisa todas as solicitações de mudança e todas as revisões e grava relações entre elas no banco de dados.

Em muitos projetos existe um padrão para mensagens de *commit*, as mensagens utilizadas para descrever uma revisão. Ao enviar as modificações é comum, na mensagem de *commit*, existir uma referência à solicitação de mudança relacionada. Por exemplo, se a solicitação de mudança #1234 foi resolvida, o desenvolvedor pode enviar a revisão com a seguinte mensagem "*issue #1234 solved*". Tendo o número da solicitação de mudança na mensagem de *commit* o *Matcher* consegue associar aquela revisão à solicitação de mudança.

Entretanto essa não é a única maneira de identificar a referência a uma solicitação de mudança em uma mensagem de *commit*. O desenvolvedor pode esquecer o “#” e

colocar apenas "*issue 1234 solved*" ou pode usar apenas "*#1234 solved*" já que eles sabem que um número se refere a uma solicitação de mudança. Não existe um padrão exato para mensagens de *commit*, então o *Matcher* usa a expressão regular da Listagem 5 para encontrar possíveis referências a solicitações de mudança em um mensagem de *commit*.

```
((fix|fixes|close|closes|resolve|resolves) *[:#] *[0-9]+)|
((fix|fixes|close|closes|resolve|resolves) *: *# *[0-9]+)|
((fix|fixes|close|closes|resolve|resolves) *(bug|issue|pr)
*#? *[0-9]+)| (# *[0-9]+)
```

Listagem 5: Expressão regular para as mensagens de commit

3.2.7 Similarity Assigner

O *Similarity Assigner* atribui similaridade entre solicitações de mudança. Ele analisa cada solicitação e, de acordo com uma técnica usada para calcular a similaridade entre textos, grava no banco de dados as relações de similaridade entre cada uma.

Neste trabalho foram implementadas duas técnicas distintas para o *Similarity Assigner*. O *Prediction by Partial Matching* (PPM) que é a proposta do Mentor para encontrar similaridade entre as solicitações de mudança e o *Latent Semantic Indexing* (LSI), muito usado em trabalhos similares.

O processo de identificação das solicitações de mudança similares usando o PPM é dividido em duas etapas: construção dos modelos e classificação das solicitações de mudança.

Na primeira etapa é criado um modelo PPM para cada solicitação de mudança. O modelo é criado usando os seguintes dados de uma solicitação de mudança: sumário e descrição detalhada. O contexto utilizado no PPM do Mentor é $k = 4$, pois foram realizados testes preliminares e constatou-se que o aumento do contexto além desse valor praticamente não modificava os resultados e piorava cada vez mais o desempenho da ferramenta. Uma vez criado, o modelo usado para classificação não será alterado.

Conforme descrito na seção 2.3, o modelo criado é composto por informações estatísticas sobre a ocorrência de símbolos dentro de determinados contextos. Ele será utilizado posteriormente para calcular a entropia das solicitações de mudanças presentes

no sistema de controle de mudanças.

A etapa de classificação das solicitações de mudança é iniciada após a criação dos modelos na primeira etapa. Para encontrar as solicitações de mudança semelhantes a um outra solicitação de mudança (*master*) é feito o cálculo da entropia de todas as solicitações, utilizando o sumário, descrição detalhada e os comentários, de acordo com o modelo *master*.

Cada solicitação de mudança classificada usando o modelo do *master* terá um valor de entropia associado. Ordenando a lista de entropias de forma crescente, as solicitações de mudança com menores entropias são as solicitações que deveriam ser as mais semelhantes ao *master*, então a lista gerada é uma lista ordenada pela semelhança das solicitações. Entretanto na prática a solicitação atribuída pelo PPM como mais semelhante nem sempre é a correta, pois a técnica pode errar. Sendo assim, em vez de retornar apenas o primeiro item, pode ser útil retornar uma lista com várias solicitações de mudança para que o desenvolvedor analise as sugestões do Mentor.

Para o LSI o processo é dividido em três etapas: construção da matriz termo-documento, aplicação do SVD e classificação.

Inicialmente todas as solicitações de mudança resolvidas são analisadas para gerar a matriz termo-documento, neste processo são utilizados os dados do sumário, descrição e comentários.

Cada documento representa um vetor num espaço que possui o número de dimensões iguais ao número de termos da matriz. A segunda etapa reduz as dimensões do espaço através da aplicação do SVD, assim os documentos mais parecidos poderão ficar cada vez mais próximos no espaço e os mais distintos poderão ficar mais separados.

Para finalizar, o sumário e a descrição de cada solicitação de mudança são transformados em documentos para serem comparados aos documentos gerados no primeiro passo. A cada comparação um valor de similaridade, entre 1 e -1, é calculado, onde 1 significa igualdade entre os documentos e -1 significa que não existe nenhuma relação entre elas. Dessa forma é possível ordenar as solicitações de mudança para que seja retornada uma lista das solicitações mais parecidas com uma solicitação de entrada.

3.3 Uso do Mentor

Para começar a usar o Mentor, primeiro precisa-se escolher um projeto no qual o Mentor efetuará as recomendações. Após escolher o projeto, tem-se que analisar as ferramentas utilizadas para gerenciar as solicitações de mudança e para controle de versão.

Um *parser* precisa ser criado para processar os dados das solicitações de mudança e gravá-los no Mentor. Não importa como estes dados são lidos ou como eles estão representados, a única exigência é que após todo o processo os dados sejam gravados no formato apresentado na seção 3.2.5. Outro *parser* precisa ser criado para processar os dados do controle de versão. Novamente os detalhes do processo de receber os dados do controle de versão e processá-los é totalmente dependente do parser e não influencia no funcionamento do motor de recomendação. Se os dados forem gravados da forma correta o Mentor funcionará normalmente.

Um *parser* pode ser reaproveitado em diferentes projetos se eles utilizarem uma mesma ferramenta para gerenciar as solicitações de mudança ou para controlar versão. Por exemplo, um parser que funcione para os dados exportados do Bugzilla de um projeto funciona para os dados exportados do Bugzilla de outro projeto.

Após criar os dois *parsers* eles precisam ser executados para gerar os arquivos com dados no formato JSON e em seguida deve-se carregar estes dados no banco do Mentor.

O próximo passo para preparação do Mentor é a execução do *Matcher*. Com os dados todos gravados o *Matcher* deve ser executado para que as solicitações de mudança e as revisões do controle de versão sejam analisadas e as relações entre elas gravadas no banco de dados.

Continuando, o *Similarity Assigner* deve ser executado para que as relações de similaridade sejam criadas. Para usar métodos diferentes de atribuição de similaridade basta apenas executar um *Similarity Assigner* diferente. A ordem de execução entre o *Matcher* e o *Similarity Assigner* não é importante, eles funcionam independentemente.

Após todos esses passos, o Mentor está pronto para efetuar as recomendações. Todas as relações importantes estão gravadas no banco de dados, então acessando os dados é possível recuperar as solicitações de mudança similares a outra, recuperar as revisões associadas às solicitações de mudança e também ter acesso aos arquivos modificados

por cada revisão. Tudo o que é necessário para efetuar a recomendação de código-fonte baseada em uma solicitação de mudança.

3.4 Conclusão

Este capítulo apresentou os principais aspectos da ferramenta proposta. Os requisitos e a arquitetura definidas foram definidos e o funcionamento de cada componente que compõe o Mentor foi explicado em detalhes. Por fim, o capítulo demonstrou como o utilizar o Mentor desde a sua configuração até como conseguir uma recomendação.

O próximo capítulo apresenta os experimentos realizados para a avaliação do Mentor, os resultados obtidos e a análise deles.

4

Avaliação

O capítulo 3 apresentou o Mentor, um sistema de recomendação de código-fonte baseado no conteúdo de solicitações de mudança. Entretanto ainda é necessário avaliar a eficiência das recomendações efetuadas pelo Mentor para saber se elas realmente podem ajudar e qual a melhor técnica para atribuição de similaridade.

Nas seções seguintes são mostradas as técnicas utilizadas para a avaliação da ferramenta, a avaliação em si do Mentor, os resultados conseguidos e a análise deles.

4.1 Técnicas de avaliação de software

Sjoberg *et al.* (2007) classificou os métodos de estudos empíricos em quatro grupos. São eles: experimentação, *surveys*, estudos de caso e *action research*.

Experimentação. Um experimento é uma investigação empírica que investiga tanto relações causais quanto processos causais. A identificação de relações de causa fornece uma explicação das razões de acontecimento de um fenômeno, enquanto a identificação dos processos da causa explica como um fenômeno acontece. Experimentos são realizados quando o pesquisador necessita do controle da situação com a manipulação direta, precisa e sistemática do comportamento do fenômeno estudado. Assim, experimentos são úteis para testar teorias e hipóteses.

Surveys. Um *survey* é um estudo retrospectivo da situação que investiga relacionamentos e resultados. Este tipo de estudo é útil quando o número de variáveis é muito

grande, o número de amostras também é grande e usa-se estatística rigorosa. *Surveys* são especialmente úteis para responder perguntas sobre o que, quanto, como e por que. Eles são usados quando não é possível ter controle das variáveis, quando os fenômenos de interesse devem ser estudados em seu ambiente natural e quando os fenômenos ocorrem no tempo atual ou no passado recente.

Estudos de caso. Um estudo de caso é uma investigação empírica de um fenômeno contemporâneo no seu contexto de uso real, especialmente quando as fronteiras entre o fenômeno e o contexto não são evidentes. Então, enquanto o experimento separa o fenômeno do seu contexto e o *survey* não investiga profundamente as condições de um fenômeno no seu contexto, o estudo de caso tem como objetivo cobrir as condições ligadas ao contexto em que o fenômeno está inserido. Em engenharia de software os estudos de caso ajudam a responder perguntas do tipo "qual o melhor?" (Kitchenham *et al.*, 1995).

Action research. Uma *action research* combina teoria e prática. Ela tenta fornecer resultados práticos para uma organização enquanto contribui para aquisição de novos conhecimentos. Ela pode ser caracterizada como um processo iterativo envolvendo pesquisadores e profissionais trabalhando em conjunto no diagnóstico de problemas, ações de intervenção e aprendizado reflexivo. A *action research* é muito importante para entender os resultados práticos de uma pesquisa, mas pode faltar objetividade nas atividades dos pesquisadores, pois eles podem priorizar os resultados para um cliente ou o lucro da organização em detrimento da pesquisa.

Na avaliação do Mentor é necessário um controle sobre os dados para poder testar a eficiência da ferramenta. Como temos uma grande quantidade de dados de solicitações de mudança e *logs* de controle de versão e temos o controle sobre estes dados é mais adequado utilizar experimentação para avaliar o trabalho.

4.2 Avaliação do Mentor

O planejamento do experimento seguiu o modelo proposto no trabalho de Wohlin *et al.* (2000) e também seguido por Trindade (2009), sendo dividido em cinco atividades. Na **Definição** são especificados os aspectos mais importantes do experimento e o motivo pelo qual ele é conduzido. No **Planejamento** é estabelecido como é realizado o experimento,

enquanto que na **Execução**, os dados são coletados para avaliação na fase de **Análise e Interpretação**. Para finalizar, os resultados são apresentados na fase de **Apresentação**.

4.2.1 Definição

O paradigma Goal Question Metric (GQM) (Basili *et al.*, 1994) foi adotado na definição do experimento, para formular o objetivo do estudo, as perguntas a serem respondidas e as métricas necessárias para formar as respostas.

Objetivo

O objetivo deste estudo é:

Comparar, no contexto do Mentor, a qualidade das recomendações utilizando o PPM para encontrar a similaridade entre solicitações de mudança em relação às recomendações que utilizam o LSI para encontrar este mesmo tipo de similaridade.

Pergunta

A pergunta a ser respondida é a seguinte:

P_1 As recomendações da ferramenta são melhores utilizando o PPM ou o LSI para encontrar similaridade entre as solicitações de mudança?

Métricas

M1. *Precision*: é calculada dividindo-se a quantidade de recomendações corretas pela quantidade de recomendações realizadas. Esta métrica indica se as recomendações realmente ajudam na resolução da solicitação de mudança (Wives, 1999) (Minto and Murphy, 2007).

M2. *Recall*: é calculada dividindo-se a quantidade de recomendações corretas pela quantidade de arquivos que realmente fazem parte da solução. Esta métrica indica se a quantidade de arquivos de código-fonte recomendados é igual à quantidade de arquivos

que foram modificados para resolver a solicitação de mudança (Wives, 1999) (Minto and Murphy, 2007).

M3. *Recall rate*: é definida como a porcentagem dos casos em que um item relevante procurado está presente no resultado da recomendação. Por exemplo, suponha que esteja-se procurando arquivos em 10 solicitações de mudança. Se em 7 casos houver sucesso na busca de pelo menos um arquivo de código-fonte, então a recall rate é 70% (Runeson *et al.*, 2007).

4.2.2 Planejamento

Os projetos escolhidos para execução dos experimentos precisam ter disponíveis os dados das solicitações de mudança e do controle de versão para ser possível carregar estes dados no Mentor. É importante ressaltar que a linguagem de programação utilizada nos projetos não é um fator limitante para as recomendações do Mentor.

Contexto

O objetivo do experimento é comparar as recomendações utilizando o PPM e o LSI para encontrar a similaridade entre as solicitações de mudança. Três projetos de código aberto foram escolhidos, cada um com um propósito distinto, dois deles escritos em C e um em Java. Mais detalhes sobre os projetos são dados na seção 4.2.3.

Todo procedimento realizado para o PPM é realizado da mesma forma para o LSI. As mesmas solicitações de mudança foram usadas, os mesmos dados do controle de versão e a variação da quantidade de solicitações de mudança similares recomendados também são iguais.

Participantes

Os participantes desse estudo são as solicitações de mudança dos projetos avaliados.

Hipóteses nulas

São hipóteses a serem rejeitadas pelo pesquisador da forma mais significativa possível. Para o experimento as hipóteses nulas determinam que as recomendações utilizando o PPM são iguais, de acordo com as métricas, às recomendações utilizando o LSI. Desta forma as seguintes hipóteses nulas foram geradas:

$\mathbf{H}_{n1}$ *precision* do PPM = *precision* do LSI

$\mathbf{H}_{n2}$ *recall* do PPM = *recall* do LSI

$\mathbf{H}_{n3}$ *recall rate* do PPM = *recall rate* do LSI

Hipóteses alternativas

São hipóteses com o objetivo de rejeitar as hipóteses nulas. As hipóteses alternativas do experimento foram as seguintes:

$\mathbf{H}_{a1}$ *precision* do PPM > *precision* do LSI

$\mathbf{H}_{a2}$ *recall* do PPM > *recall* do LSI

$\mathbf{H}_{a3}$ *recall rate* do PPM > *recall rate* do LSI

Variáveis independentes

Todas as variáveis controladas e manipuladas durante o experimento são chamadas de independentes. As variáveis independentes no experimento são os dados das solicitações de mudança, os dados dos controles de versão e a técnica utilizada para atribuição de similaridade. Nesse experimento teremos um fator, uma variável que é modificada para poder ser observada a sua influência no experimento, que é o algoritmo de atribuição de similaridade, e serão aplicados dois tratamentos, o PPM e o LSI.

Variáveis dependentes

As variáveis dependentes são os objetos de estudo do experimento. Seus valores variam com as mudanças feitas nas variáveis independentes. As variáveis dependentes do

experimento são a *precision*, *recall* e *recall rate* das recomendações.

Validade de conclusão

Esta validade diz respeito a relação entre o tratamento e o resultado. Para avaliarmos a *precision* e *recall* será utilizado o teste de postos com sinais de Wilcoxon para pares combinados e para avaliarmos a *recall rate* será utilizado o teste de proporção para duas amostras (Triola, 2008).

Validade interna

Se uma relação é observada entre o tratamento e o resultado, precisamos garantir que isso é uma relação causal, isto é, não é resultado de um fator que não se tenha controle ou que não tenha sido levado em consideração. Nos experimentos realizados as variáveis independentes são estáticas e a única variável independente que está sendo alterada é o fator algoritmo de similaridade.

Validade do constructo

Diz respeito a relação entre teoria proposta e os resultados observados com a execução do experimento. De acordo com a pergunta formulada na definição do experimento a causa será representada pelo fator e o resultado pelas métricas definidas previamente.

Validade externa

Verifica a capacidade do experimento em ser generalizado, ou seja, se o mesmo resultado pode ser obtido em outros projetos. Para tentar garantir a generalização, três experimentos foram realizados utilizando três projetos de naturezas distintas, com times, propósitos, linguagens de programação e tamanhos diferentes.

4.2.3 Projetos utilizados no experimento

Foram realizados três experimentos, cada um utilizando um projeto diferente. Foram escolhidos projetos de código aberto, porque disponibilizam os dados que o Mentor precisa para funcionar.

GTK+ é uma biblioteca multiplataforma para criação de interface gráfica, usada em muitos projetos, principalmente, os relacionados ao GNOME do Linux, e em diferentes plataformas, tais como PCs e dispositivos móveis. O projeto é escrito predominantemente em C, totalizando 503.161 linhas de código distribuídas entre 1.348 arquivos.

GIMP é uma ferramenta multiplataforma para criação e edição de imagens. O projeto é escrito predominantemente em C, totalizando 737.835 linhas de código distribuídas em 3.293 arquivos.

Hadoop é um *framework* mantido pela fundação Apache para processamento distribuído de grandes massas de dados usando *clusters* de computadores. O Hadoop inclui três subprojetos, *Common*, *Distributed File System* e *MapReduce*. No experimento foram utilizados os dados do Hadoop Common. O projeto é escrito predominantemente em Java, totalizando 613.481 linhas de código distribuídas em 1.003 arquivos.

4.2.4 Instrumentação e seleção da amostra

Os experimentos foram executados tendo como entrada os dados das solicitações de mudança e do controle de versão de cada um dos projetos apresentados anteriormente. Estes dados precisaram de tratamento antes da execução do experimento, conforme descrito nos parágrafos seguintes.

Dentre o universo de solicitações de mudança de cada projeto, selecionamos as tarefas mais simples, que poderiam ser adequadas a desenvolvedores novatos, ou seja, todas as solicitações de mudança marcadas pelos desenvolvedores dos projetos como "minor"ou "trivial".

Faz-se necessário também excluir as solicitações de mudança que não tiveram solução identificada, porque não foi possível associá-la a nenhuma revisão do controle de versão.

Além disso, a solução de algumas solicitações de mudança não possuem interseção com a solução de outras. Isto pode causar problemas nos resultados das recomendações, pois esta solicitação sempre receberia recomendações erradas. Para contornar esse problema, foi feito um filtro nas solicitações de mudança para que fossem usadas apenas as que modificassem arquivos que também tenham sido modificados para solução de outras solicitações.

Por fim, os dados de controle de versão foram filtrados apenas em relação à data. Para os projetos GTK+ e GIMP estavam disponíveis no repositório do *Mining Software Repositories Challenge* 2009 (Bird, 2009) as solicitações de mudança de 06/2001 a 09/2008, mas acessando o controle de versão do projeto é possível conseguir dados que compreendem um período bem maior. Como não fazia sentido utilizar os dados do controle de versão de um período anterior à criação de uma solicitação de mudança, então foi feito um filtro para excluir dados antigos do controle de versão.

A tabela 4.1 apresenta a quantidade de solicitações de mudança e revisões utilizadas após a execução dos filtros e o período em que elas estão distribuídos. Para cada projeto foram selecionados aleatoriamente um número de solicitações de mudança para receber recomendações com a finalidade de manter um intervalo de confiança de 95% e margem de erro de 5%. Utilizamos a ferramenta estatística STATDISK (Pearson Education, 2009) para realizar os cálculos. Assim, para o GTK+ foram selecionadas 252 solicitações numa população de 727, 255 dentre 753 para o GIMP e 250 dentre 714 para o Hadoop. A lista com todas as solicitações de mudança utilizadas no conjunto de treinamento e no conjunto de testes pode ser encontrada no Apêndice A.1.

Tabela 4.1 Dados utilizados no experimento

Projetos	Dados	Quantidade	Período
GTK+	solicitações de mudança revisões	727 26805	06/2001 a 08/2008
GIMP	solicitações de mudança revisões	753 29708	06/2001 a 09/2008
Hadoop	solicitações de mudança revisões	714 3049	06/2009 a 06/2011

4.2.5 Execução

Três experimentos foram executados, o primeiro com os dados do GTK+, o segundo com os dados do Hadoop e o último com os dados do GIMP. Para cada experimento foram carregados os dados das solicitações de mudança e das revisões, e logo após todo o processo de atribuição de relações foi realizado.

No processo de recomendação foram utilizados os dados do sumário, descrição e comentário das solicitações de mudança resolvidas, ou seja, as que podiam ser recomendadas, e o sumário e descrição para as solicitações que receberam recomendação. Isto foi escolhido, pois ao criar uma solicitação de mudança o desenvolvedor adiciona apenas os dados do sumário e descrição, já numa solicitação de mudança mais antiga, que talvez tenha sido resolvida, pode-se ter, além do sumário e descrição, comentários criados por outros desenvolvedores.

Para cada experimento foi realizada uma série de recomendações variando a quantidade de solicitações de mudança resolvidas apresentadas, começando com uma solicitação até dez. Para cada recomendação foram calculadas as métricas *precision*, *recall* e *recall rate*. Por exemplo, para uma recomendação A é primeiro recomendada apenas uma solicitação e os arquivos modificados para resolvê-la, assim as métricas são calculadas utilizando apenas os arquivos dessa única solicitação de mudança recomendada. Em seguida é pedida a recomendação da segunda solicitação de mudança mais semelhante a solicitação A, agora as métricas são calculadas utilizando os arquivos das duas solicitações de mudança mais parecidas. Esse processo continua até serem recomendadas dez solicitações de mudança.

Todas as solicitações de mudança utilizadas no experimento já foram solucionadas pelos desenvolvedores dos projetos previamente, entretanto no processo de recomendação não foram usados os dados das soluções das solicitações de mudança que receberam recomendações. Para analisar se uma recomendação é correta ou não foi necessário apenas comparar os arquivos das solicitações recomendadas e verificar se eles eram iguais ao da solução original.

4.2.6 Análise e interpretação

Após a execução de cada experimento, as recomendações realizadas pelo Mentor foram coletadas para o cálculo da *precision*, *recall* e *recall rate*. Foram feitas recomendações utilizando o PPM e o LSI para atribuição de similaridade entre as solicitações de mudança.

O objetivo da avaliação é verificar qual técnica para atribuição de similaridade, PPM ou LSI, consegue fornecer melhores recomendações. A seguir são apresentadas as tabelas 4.2, 4.3, 4.4, 4.5, 4.6 e 4.7 com os resultados dos três experimentos. A primeira coluna das tabelas indica a quantidade de solicitações de mudança recomendadas e as colunas seguintes trazem a média da *precision*, *recall* e *recall rate* respectivamente.

As métricas são calculadas para cada solicitação de mudança participante do experimento, então, por exemplo, recomendando uma solicitação de mudança semelhante tem-se vários valores de *precision*, *recall* e *recall rate*, um para cada solicitação. O resultado apresentado nas tabelas é a média aritmética das recomendações para todas as solicitações participantes, assim fica mais simples apresentar os resultados.

Tabela 4.2 Resultados utilizando o PPM para o projeto GTK+

Número de recomendações	Média da precision	Média da recall	Média da recall rate
1	18,7500%	18,8468%	23,4127%
2	13,4391%	25,3527%	31,3492%
3	10,5412%	29,3968%	36,5079%
4	9,0276%	35,1612%	42,4603%
5	8,1163%	38,3870%	46,0317%
6	7,2214%	39,4925%	46,8254%
7	6,6288%	42,3091%	50,0000%
8	5,9316%	43,5217%	51,1905%
9	5,5752%	45,1475%	53,1746%
10	5,2247%	45,9737%	53,9683%

O maior valor de *precision* para o GTK+ foi conseguido recomendando apenas uma solicitação de mudança, 18,75%, já para o LSI o maior valor, também recomendando apenas uma solicitação de mudança, foi 4,9592%. Através da análise dos valores brutos percebe-se que o resultado com o uso do PPM é melhor que o LSI para *precision*, entretanto esta análise não é suficiente, é necessária a realização de uma validação estatística para avaliar as hipóteses nulas.

Tabela 4.3 Resultados utilizando o LSI para o projeto GTK+

Número de recomendações	Média da precision	Média da recall	Média da recall rate
1	4,9592%	4,2606%	06,3492%
2	3,9342%	7,1240%	09,9206%
3	3,0330%	8,8151%	11,5079%
4	2,8305%	10,8794%	13,4921%
5	2,4923%	11,4101%	14,2857%
6	2,2827%	12,4076%	15,4762%
7	2,3267%	14,6894%	18,2540%
8	2,0289%	16,4780%	20,2381%
9	1,9984%	18,2683%	22,2222%
10	1,9656%	19,4291%	24,6032%

Tabela 4.4 Resultados utilizando o PPM para o projeto GIMP

Número de recomendações	Média da precision	Média da recall	Média da recall rate
1	8,2045%	8,1365%	14,5098%
2	7,1421%	13,2413%	23,1373%
3	6,1352%	16,1990%	27,8431%
4	5,1510%	18,1599%	30,1961%
5	4,0719%	18,3817%	30,9804%
6	3,7346%	19,8348%	32,9412%
7	3,5670%	21,1556%	34,5098%
8	3,2133%	22,9988%	36,4706%
9	2,9910%	23,2934%	37,2549%
10	2,7787%	24,6713%	38,8235%

Tabela 4.5 Resultados utilizando o LSI para o projeto GIMP

Número de recomendações	Média da precision	Média da recall	Média da recall rate
1	5,8685%	6,0808%	9,4118%
2	4,3099%	7,4855%	12,1569%
3	3,5313%	9,1298%	15,2941%
4	2,9819%	11,1619%	18,8235%
5	3,2136%	14,9358%	23,9216%
6	2,8211%	16,7181%	26,6667%
7	2,5593%	17,9026%	28,2353%
8	2,1960%	18,4254%	28,6275%
9	2,0572%	19,2871%	30,1961%
10	1,9696%	20,2860%	31,7647%

Tabela 4.6 Resultados utilizando o PPM para o projeto Hadoop

Número de recomendações	Média da precision	Média da recall	Média da recall rate
1	11,3363%	11,5758%	21,2000%
2	8,1523%	18,4814%	32,8000%
3	7,7853%	25,8124%	44,0000%
4	6,9537%	30,2332%	50,4000%
5	5,9870%	32,9433%	53,2000%
6	5,7859%	37,5557%	59,2000%
7	5,2238%	40,2962%	61,6000%
8	4,8348%	42,7740%	63,6000%
9	4,5226%	44,4251%	65,6000%
10	4,2596%	45,9387%	66,8000%

Tabela 4.7 Resultados utilizando o LSI para o projeto Hadoop

Número de recomendações	Média da precision	Média da recall	Média da recall rate
1	7,2149%	8,1241%	16,0000%
2	5,8055%	11,2686%	22,4000%
3	4,3976%	13,6883%	25,6000%
4	4,1377%	18,3401%	32,4000%
5	3,8100%	20,6825%	36,4000%
6	3,1283%	22,8339%	40,0000%
7	2,9159%	23,8268%	41,6000%
8	2,7451%	25,4258%	43,6000%
9	2,6426%	26,9710%	45,2000%
10	2,5892%	28,8091%	47,6000%

O Teste de Postos com Sinais de Wilcoxon para Pares Combinados (Boslaugh and Watters, 2008) foi utilizado para avaliar H_{n1}, pois as *precisions* não possuem uma distribuição normal. Com o resultado do teste foi possível rejeitar a hipótese nula H_{n1} ***precision* do PPM = *precision* do LSI**, e validar a hipótese alternativa H_{a1} ***precision* do PPM > *precision* do LSI**, para o experimento com os dados do projeto GTK+. A saída gerada pelo STATDISK é encontrada na Listagem 6.

```
Num Unequal pairs:  141

Using Approximation
Test Statistic, T: 3701,0000
Mean:              5005.5
St Dev:               485.8938
Test Statistic, z: -2,6847
Critical z:           +-1.959962

Reject the Null Hypothesis
Sufficient sample evidence to conclude the matched pairs
have differences that come from a population with a
nonzero median.
```

Listagem 6: Resultado do teste de postos com sinais de Wilcoxon para pares combinados para *precision* do GTK+

Para a métrica *recall* o PPM também consegue valores melhores que o LSI. Ambas as técnicas tem melhores resultados quando são recomendadas 10 solicitações de mudança,

o PPM atinge o valor 45,9737%, mais que o dobro do LSI que atinge 19,4291%.

Para avaliar H_{n2} também foi utilizado o Teste de Postos com Sinais de Wilcoxon para Pares Combinados, pois as *recalls* não possuem uma distribuição normal. Com o resultado do teste foi possível rejeitar a hipótese nula H_{n2} ***recall* do PPM = *recall* do LSI**, e validar a hipótese alternativa H_{a2} ***recall* do PPM > *recall* do LSI**, para o experimento com os dados do projeto GTK+. A saída gerada pelo STATDISK é encontrada na Listagem 7.

```
Num Unequal pairs:  103

Using Approximation
Test Statistic, T: 633,0000
Mean:           2678
St Dev:            303.9589
Test Statistic, z: -6,7279
Critical z:        +-1.959962

Reject the Null Hypothesis
Sufficient sample evidence to conclude the matched pairs
have differences that come from a population with a
nonzero median.
```

Listagem 7: Resultado do teste de postos com sinais de Wilcoxon para pares combinados para *recall* do GTK+

Para a métrica *recall rate* o valor máximo do PPM foi 53,9683% quando foram recomendados 10 solicitações de mudança. Isto significa que mais da metade das solicitações receberam recomendação correta de pelo menos um arquivo de código-fonte. Usando o LSI o maior valor conseguido foi 24,6032% também com 10 solicitações recomendadas.

A métrica *recall rate* tem uma natureza diferente, ela responde com que porcentagem dos casos a resposta foi positiva, ou seja, é uma medida de eficácia, por isso foi utilizado um teste de proporção para duas amostras para avaliar H_{n3}. Para o GTK+ foi possível rejeitar a hipótese nula H_{n3} ***recall rate* do PPM = *recall rate* do LSI** e validar a hipótese alternativa H_{a2} ***recall rate* do PPM > *recall rate* do LSI**. Na Listagem 8 tem-se a saída do STATDISK para o teste realizado.

No experimento com os dados do Hadoop, a maior média das *precisions* utilizando o PPM foi 11,3363%, quando apenas uma solicitação de mudança foi recomendada, já para o LSI esse valor caiu para 7,2149%. Mais uma vez foi possível rejeitar a hipótese nula H_{n1}

```
Claim:    p1 = p2

Pooled proportion: 0.3928571
Test Statistic, z: 6,7492
Critical z:        +-1,9600
P-Value:           0,0000

95% Confidence interval:
0.21232 < p1-p2 < 0.3749816

Reject the Null Hypothesis
Sample provides evidence to reject the claim
```

Listagem 8: Resultado do teste de proporção para o *recall rate* do GTK+

***precision* do PPM = *precision* do LSI** e validar a hipótese alternativa H_{a1} ***precision* do PPM > *precision* do LSI**. A Listagem 9 mostra o resultado do teste estatístico realizado no STATDISK para avaliar H_{n1}.

```
Num Unequal pairs:  181

Using Approximation
Test Statistic, T: 13264,0000
Mean:           8235.5
St Dev:            705.8667
Test Statistic, z: 7,1239
Critical z:        +-1.959962

Reject the Null Hypothesis
Sufficient sample evidence to conclude the matched pairs
have differences that come from a population with a
nonzero median.
```

Listagem 9: Resultado do teste de postos com sinais de Wilcoxon para pares combinados para *precision* do Hadoop

Para a *recall* do Hadoop, os melhores valores foram conseguidos recomendando 10 solicitações de mudança, isto é esperado dada a natureza da métrica, o PPM obteve 45,9387% e o LSI 28,8091%. A hipótese nula H_{n2} ***recall* do PPM = *recall* do LSI** é rejeitada devido aos resultados do teste estatístico e a hipótese alternativa H_{a2} ***recall* do PPM > *recall* do LSI** validada. A Listagem 10 mostra o resultado do teste estatístico realizado no STATDISK para avaliar H_{n2}.

```
Num Unequal pairs:  128

Using Approximation
Test Statistic, T: 11106,0000
Mean:           4128
St Dev:            420.4949
Test Statistic, z: 16,5947
Critical z:        +-1.959962

Reject the Null Hypothesis
Sufficient sample evidence to conclude the matched pairs
have differences that come from a population with a
nonzero median.
```

Listagem 10: Resultado do teste de postos com sinais de Wilcoxon para pares combinados para *recall* do Hadoop

O maior valor de *recall rate* entre todos os experimentos foi conseguido com a utilização dos dados do Hadoop e a técnica PPM, o valor chega a 66,8%, já com o LSI o valor chega a 47,6%. Para o Hadoop foi possível rejeitar a hipótese nula H_{n3} ***recall rate* do PPM = *recall rate* do LSI** e validar a hipótese alternativa H_{a3} ***recall rate* do PPM > *recall rate* do LSI**. Na Listagem 11 tem-se a saída do STATDISK para o teste realizado.

```
Claim:    p1 = p2

Pooled proportion: 0.572
Test Statistic, z: 4,3385
Critical z:        +-1,9600
P-Value:           0,0000

95% Confidence interval:
0.1069096 < p1-p2 < 0.2770904

Reject the Null Hypothesis
Sample provides evidence to reject the claim
```

Listagem 11: Resultado do teste de proporção para o *recall rate* do Hadoop

O último experimento executado usou os dados do GIMP e mais uma vez o PPM obteve valores brutos maiores que o LSI em todos os casos. Para a *precision* a técnica proposta neste trabalho conseguiu 8,2045% no seu melhor caso, já o LSI obteve o valor 5,8685%, ambos os resultados foram conseguidos recomendando apenas uma solicitação

de mudança. Novamente foi possível rejeitar a hipótese nula H_{n1} ***precision* do PPM = *precision* do LSI**, e validar a hipótese alternativa H_{a1} ***precision* do PPM > *precision* do LSI**. A Listagem 12 mostra o resultado do teste estatístico realizado no STATDISK para avaliar H_{n1}.

```
Num Unequal pairs:  119

Using Approximation
Test Statistic, T: 6747,0000
Mean:              3570
St Dev:               377.1008
Test Statistic, z: 8,4248
Critical z:           +-1.959962

Reject the Null Hypothesis
Sufficient sample evidence to conclude the matched pairs
have differences that come from a population with a
nonzero median.
```

Listagem 12: Resultado do teste de postos com sinais de Wilcoxon para pares combinados para *precision* do GIMP

Para a métrica *recall* os maiores valores foram conseguidos recomendando dez solicitações de mudança, o PPM obteve 24,6713% e o LSI 20,2860%, sendo possível também rejeitar a hipótese nula H_{n2} ***recall* do PPM = *recall* do LSI**, e validar a hipótese alternativa H_{a2} ***recall* do PPM > *recall* do LSI**. A Listagem 13 mostra o resultado do teste estatístico realizado no STATDISK para avaliar H_{n2} no experimento com os dados do GIMP.

O PPM obteve *recall rate* máxima igual a 38,8235%, no experimento com o GIMP, e o LSI 31,7647%, um valor menor que a técnica proposta, todavia ainda precisa-se utilizar o teste estatístico para rejeitar ou não H_{n3}.

Para o experimento com GIMP não foi possível rejeitar a hipótese nula H_{n3} ***recall rate* do PPM = *recall rate* do LSI**. Entretanto é importante ressaltar que mesmo sendo impossível rejeitar H_{n3} o valor nominal encontrado com a utilização do PPM é maior que o valor utilizando o LSI e que se fosse utilizado no teste de proporção uma significância de 0,1 ao invés de 0,05 seria possível rejeitar H_{03}. Na Listagem 14 tem-se a saída do STATDISK para o teste realizado.

O Hadoop teve os melhores resultados para *recall rate*, 66,8%, praticamente empatou

```
Num Unequal pairs:  80

Using Approximation
Test Statistic, T: 11495,0000
Mean:           1620
St Dev:            208.4946
Test Statistic, z: 47,3633
Critical z:        +-1.959962

Reject the Null Hypothesis
Sufficient sample evidence to conclude the matched pairs
have differences that come from a population with a
nonzero median.
```

Listagem 13: Resultado do teste de postos com sinais de Wilcoxon para pares combinados para *recall* do GIMP

```
Claim:    p1 = p2

Pooled proportion: 0.3529412
Test Statistic, z: 1,6679
Critical z:        +-1,9600
P-Value:           0,0953

95% Confidence interval:
-0.0121351 < p1-p2 < 0.1533116

Fail to Reject the Null Hypothesis
Sample does not provide enough evidence
to reject the claim
```

Listagem 14: Resultado do teste de proporção para o *recall rate* do GIMP

com o GTK+ no *recall*, o projeto da Apache obteve 45,93% e o do GNOME 45,97% e teve piores resultados que o GTK+ na *precision*.

A forma de utilizar o controle de versão e o gerenciador de solicitação de mudança pode ter influência nos resultados do Mentor. No Hadoop praticamente todos os *commits* fazem uma referência explícita a solicitação de mudança que ele está resolvendo, como pode ser observado na Figura 4.1, com isso as associações entre revisões e solicitações de mudança são feitas sem erros, melhorando a recomendação dos arquivos de código-fonte.

O *recall rate* é uma métrica importante neste experimento, pois o Mentor pode recomendar solicitações de mudança que alteram alguns ou exatamente os arquivos necessários, mas também alteram outros. Isto não invalida a qualidade da recomendação, pois mesmo baixando os resultados da *precision* e *recall*, a recomendação pode ser extremamente útil para o desenvolvedor.

Por exemplo, a solicitação de mudança #7300 do Hadoop foi resolvida alterando três arquivos com extensão .java (Configuration.java, TestConfiguration.java, StringUtils.java) e, utilizando o PPM, a solicitação mais parecida foi #7001, que foi solucionada modificando sete arquivos .java, entre eles o Configuration.java. Recomendando apenas uma solicitação de mudança similar, o *recall rate* tem valor 1, mas a *precision* tem apenas o valor 14,2%. Entretanto ao encontrar o arquivo Configuration.java é muito provável que o desenvolvedor também verifique o arquivo de testes TestConfigurarion.java. Também é possível que o desenvolvedor chegue no arquivo StringUtils.java partindo da classe Configuration, pois métodos dessa classe são usados com frequência no arquivo Configuration.java. Assim, pode-se perceber, que mesmo com uma *precision* 14,2%, a recomendação guia bem o desenvolvedor para resolução da solicitação de mudança.

As mensagens de *commit* do GTK+ e do GIMP não seguem um padrão bem definido como as do Hadoop, entretanto apenas o GIMP ficou com resultados muito inferiores. Isto provavelmente deve-se ao fato que as solicitações de mudança mais simples do GIMP são apenas mais diferentes entre si que as solicitações do GTK+, logo o Mentor não consegue encontrar solicitações de mudança realmente parecidas entre si e consequentemente a recomendação acaba não tendo uma boa qualidade.

Figura 4.1 Revisões do Hadoop

4.3 Conclusão

Neste capítulo foram apresentadas as fases de definição, planejamento, execução, análise, interpretação e apresentação da avaliação da qualidade das recomendações do Mentor. Analisou-se duas técnicas para encontrar a similaridade entre as solicitações de mudança, PPM e LSI, um passo fundamental da recomendação da ferramenta.

Foram realizados três experimentos, cada um com um projeto diferente, para avaliar se as recomendações utilizando o PPM para atribuição de similaridade são melhores que as recomendações utilizando o LSI.

Apenas para a métrica *recall rate* no experimento com os dados do GIMP não foi possível rejeitar a hipótese nula e concluir que os resultados com o PPM foram melhores que os com o LSI, entretanto, se comparado apenas os dados brutos, o PPM consegue um valor maior. Com os três experimentos realizados pode-se mostrar que o uso do PPM para atribuir similaridade entre textos é potencialmente melhor que o LSI.

No próximo capítulo, serão apresentadas as conclusões deste trabalho, as principais contribuições e as direções para trabalhos futuros.

5

Conclusão

O processo de desenvolvimento de software gera muitos artefatos com os quais os desenvolvedores costumam interagir frequentemente. Um tipo comum de artefato é a solicitação de mudança, a descrição de uma modificação que deve ser feita no software. Desenvolvedores que não possuem conhecimento suficiente sobre a parte do projeto relacionada à solicitação podem ter dificuldade em solucionar o problema em aberto. Nesse momento pode surgir o mentor, alguém mais experiente que ajuda o desenvolvedor iniciante no entendimento do código-fonte e dos procedimentos utilizados no processo de desenvolvimento, e na solução de diversos problemas. Ter um mentor ao lado é bom para quem estiver sendo ajudado, entretanto deslocar alguém experiente, que poderia estar resolvendo problemas complicados do desenvolvimento, para ensinar pode ser inviável para uma empresa, o custo de tirar um mentor do seu trabalho principal é grande.

Por isso foi criado e depois avaliado o Mentor, um sistema de recomendação para Engenharia de Software, com o objetivo de auxiliar desenvolvedores que não tenham conhecimento sobre onde mexer para resolver uma solicitação de mudança, ou seja, de substituir em partes o papel do mentor real. O Mentor recomenda código-fonte dada uma solicitação de mudança não resolvida, mostrando para o desenvolvedor um conjunto de arquivos que possivelmente estão relacionados com o problema apontado.

5.1 Contribuições da pesquisa

A pesquisa teve duas grandes contribuições. A primeira está relacionada a criação do sistema de recomendação para Engenharia de Software chamado Mentor e a segunda está relacionada ao experimento realizado para avaliar a ferramenta em relação às técnicas de atribuição de similaridade entre artefatos textuais, no caso do Mentor, solicitações de mudança.

O Mentor. Após serem analisados alguns trabalhos na área foram identificadas algumas ferramentas relacionadas a sistemas de recomendação para Engenharia de Software e as técnicas empregadas nessas ferramentas. O objetivo do Mentor é recomendar código-fonte dada uma solicitação de mudança não resolvida, algumas ferramentas na literatura já conseguem fazer isso, mas no Mentor foi utilizado o PPM para atribuição de similaridade para tentar melhorar os resultados dos sistemas de recomendação conhecidos.

O experimento. Para avaliar as recomendações realizadas pelo Mentor foram executados três experimentos com dados de projetos reais. Foi analisada a qualidade das recomendações da ferramenta com duas técnicas diferentes de atribuição de similaridade, com a finalidade de descobrir se o PPM poderia ser usado no contexto do problema e se ele teria resultados melhores que o LSI, uma técnica comum na literatura para procurar artefatos baseados em texto. Como resultado o experimento identificou que além de funcionar bem o PPM consegue melhorar as recomendações se comparado com o uso do LSI. Usando o PPM foram conseguidos resultados para recall rate entre 38,82% e 66,8%, e utilizando o LSI os resultados variaram entre 24,6% e 47,6%.

5.2 Trabalhos relacionados

Durante o desenvolvimento da pesquisa alguns trabalhos relacionados foram identificados. No capítulo 2 cinco trabalhos com objetivos próximos a este trabalho foram apresentados, ou seja, trabalhos que propõem sistemas e técnicas que tentam ajudar desenvolvedores em um projeto de software efetuando recomendações de artefatos. Entretanto existem diferenças entre o Mentor e os demais trabalhos. O trabalho de Goldman and Miller (2009), por exemplo, além do código-fonte, também recomenda documentação, e ele não utiliza dados de solicitações de mudança. Por outro lado, o trabalho de Cubranic *et al.*

(2005) tem exatamente o mesmo propósito deste trabalho, além de outros artefatos, ele recomenda código-fonte baseado em solicitações de mudança. A diferença chave entre o trabalho proposto e o trabalho de Cubranic *et al.* (2005) é a técnica empregada para atribuição de semelhança entre artefatos textuais, ele utiliza o LSI, o Mentor utiliza o PPM.

5.3 Trabalhos futuros

Este trabalho foi o passo inicial para o uso do PPM para predições relacionadas a Engenharia de Software. Foi possível perceber que ele funciona muito bem para o problema proposto, por isso temos muito o que investigar no futuro.

Recomendar outros artefatos. Como os trabalhos de Cubranic *et al.* (2005) e Begel *et al.* (2010) o Mentor pode ser estendido no futuro para recomendar outros tipos de artefatos. Mensagens de uma lista de discussão poderiam ser recomendadas por similaridade a outras mensagens e a solicitações de mudança utilizando o PPM. Da mesma forma poderia ocorrer a recomendação de documentação do projeto, estes artefatos contêm texto e o PPM tem potencial para recomendar bem todos eles. Talvez também seja possível recomendar pessoas envolvidas no projeto, por exemplo um especialista que conheceria os arquivos que precisariam ser alterados para resolver uma solicitação de mudança.

Aumentar especifidade da recomendação. Através dos dados de controle de versão, além dos arquivos modificados em uma revisão, é possível ter acesso às linhas de código-fonte modificadas nos arquivos. Então o Mentor poderia tentar recomendar não apenas arquivos, mas classes, métodos ou até mesmo linhas de código-fonte que influenciaram na solução de uma solicitação de mudança, basta que seja identificada que parte dos arquivos que o Mentor recomenda foram modificadas e exibir está informação para o usuário da ferramenta. O Mentor se transformaria num filtro ainda mais rigoroso, talvez facilitando ainda mais a solução de um problema em aberto.

Integração com gerenciadores de projeto. O Mentor poderia ser integrado a gerenciadores de projeto já existentes. Este tipo de ferramenta frequentemente agrega informações tanto das solicitações de mudança quanto do controle de versão, tudo que o Mentor precisa para funcionar. O sistema de recomendação funcionaria como uma

melhoria para sistemas já utilizados por desenvolvedores.

Estudo de caso O Mentor poderia ser utilizado em um estudo de caso real em larga escala. Um projeto desenvolvido utilizando código aberto poderia utilizar o Mentor para tentar atrair novos desenvolvedores colocando as recomendações em teste no mundo real. Assim poderiamos avaliar a utilidade das recomendações de forma qualitativa além da quantitativa.

5.4 Considerações finais

Muitos problemas emergem durante o desenvolvimento de um software, porque fazer software é uma tarefa complexa, porém a cada dia mais empresas passam a produzir e depender desse tipo de tecnologia. Os esforços para melhorar o processo de desenvolvimento são extremamente válidos, tanto para melhorar a qualidade do produto desenvolvido quanto para diminuir o custo. Os sistemas de recomendação para Engenharia de Software podem ajudar em ambos os casos.

O Mentor ajuda desenvolvedores novatos em um projeto de software. Se o desenvolvedor desconhece a parte do código-fonte relacionada a uma solicitação de mudança, então o sistema de recomendação aponta o caminho sem necessitar de intervenção de um desenvolvedor experiente.

Este trabalho avançou no estado-da-arte de sistemas de recomendação de código-fonte utilizando algoritmo de compressão de dados PPM para atribuição de similaridade entre solicitações de mudanças. Por fim, foram realizados três experimentos para avaliar a ferramenta, que obtiveram resultados positivos, mostrando que o Mentor é eficiente e eficaz para o que se propõe.

Referências Bibliográficas

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, **17**(6), 734–749.

Balabanovic, M. and Shoham, Y. (1997). Content-based, collaborative recommendation. *Communications of the ACM*, **40**(3), 66–72. Cited By (since 1996): 720.

Balabanović, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Commun. ACM*, **40**(3), 66–72.

Basili, V., Caldiera, G., and Rombach, H. (1994). *Encyclopedia of Software Engineering*, chapter Goal Question Metric Approach, pages 528–532. John Wiley & Sons, Inc.

Begel, A., Khoo, Y. P., and Zimmermann, T. (2010). Codebook: discovering and exploiting relationships in software repositories. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 125–134, New York, NY, USA. ACM.

Billsus, D. and Pazzani, M. J. (2000). User modeling for adaptive news access. *User Modeling and User-Adapted Interaction*, **10**, 147–180.

Bird, C. (2009). Msr challenge 2009 data. http://msr.uwaterloo.ca/msr2009/challenge/msrchallengedata.html.

Boslaugh, S. and Watters, P. A. (2008). *Statistics in a Nutshell*. O'Reilly.

Clements, P., Garlan, D., Little, R., Nord, R., and Stafford, J. (2003). Documenting software architectures: Views and beyond. pages 740–741. cited By (since 1996) 9.

Cubranic, D., Murphy, G. C., Singer, J., and Booth, K. S. (2004). Learning from project history: a case study for software development. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 82–91, New York, NY, USA. ACM.

Cubranic, D., Murphy, G. C., Singer, J., and Booth, K. S. (2005). Hipikat: A project memory for software development. *IEEE Trans. Softw. Eng.*, **31**(6), 446–465.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, **41**(6), 391–407.

Espinosa, J. A. and Pickering, C. (2006). The effect of time separation on coordination processes and outcomes: A case study. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 25.2, Washington, DC, USA. IEEE Computer Society.

Goldman, M. and Miller, R. C. (2009). Codetrail: Connecting source code and web resources. *J. Vis. Lang. Comput.*, **20**, 223–235.

Grund, F. (1979). Forsythe, g. e. / malcolm, m. a. / moler, c. b., computer methods for mathematical computations. englewood cliffs, new jersey 07632. prentice hall, inc., 1977. xi, 259 s. *Zamm-zeitschrift Fur Angewandte Mathematik Und Mechanik*, **59**, 141–142.

Hansman, C., Mott, V., Ellinger, A., and Guy, T. (2002). *Critical Perspectives on Mentoring: Trends and Issues*. Columbus, OH: Clearinghouse on Adult, Career and Vocational Education.

Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. In *FOSE '07: 2007 Future of Software Engineering*, pages 188–198, Washington, DC, USA. IEEE Computer Society.

Holmes, R., Walker, R., and Murphy, G. (2006). Approximate structural context matching: An approach to recommend relevant examples. *IEEE Trans. Softw. Eng.*, **32**(12), 952–970.

Kitchenham, B., Pickard, L., and Pfleeger, S. L. (1995). Case studies for method and tool evaluation. *IEEE Softw.*, **12**(4), 52–62.

Kruchten, P., Obbink, H., and Stafford, J. (2006). The past, present, and future for software architecture. *IEEE Softw.*, **23**, 22–30.

Minto, S. and Murphy, G. C. (2007). Recommending emergent teams. In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 5, Washington, DC, USA. IEEE Computer Society. expert.

Moin, A. H. and Khansari, M. (2010). Bug localization using revision log analysis and open bug repository text categorization. In *OSS*, pages 188–199.

Papadimitriou, C. H., Raghavan, P., and X, S. V. (1998). Latent semantic indexing: A probabilistic analysis. pages 159–168. ACM press.

Pearson Education (2009). Statdisk 11.1.0 for the triola series of statistics textbooks.

Pirolli, P. and Anderson, J. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology*, **39**, 240–272.

Resnick, P. and Varian, H. R. (1997). Recommender systems. *Commun. ACM*, **40**(3), 56–58.

Rich, E. (1979). User modeling via stereotypes. *Cognitive Science*, **3**(4), 329–354. cited By (since 1996) 175.

Robillard, M., Walker, R., and Zimmermann, T. (2010). Recommendation systems for software engineering. *IEEE Softw.*, **27**, 80–86.

Runeson, P., Alexandersson, M., and Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 499–510, Washington, DC, USA. IEEE Computer Society.

Salomon, D. and Motta, G. (2009). *Handbook of Data Compression*. Springer Publishing Company, Incorporated, 5th edition.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, **27**, 379–423.

Sjoberg, D. I. K., Dyba, T., and Jorgensen, M. (2007). The future of empirical methods in software engineering research. In *2007 Future of Software Engineering*, FOSE '07, pages 358–378, Washington, DC, USA. IEEE Computer Society.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, **147**(1), 195–197.

Soboroff, I. and Nicholas, C. (1999). Combining content and collaboration in text filtering. In *IJCAI'99 Workshop: Machine Learning for Information Filtering*.

Trindade, C. C. d. (2009). *Presley: uma ferramenta de recomendação de especialistas para apoio à colaboração em desenvolvimento distribuído de software*. Master's thesis, Universidade Federal de Pernambuco.

Triola, M. F. (2008). *Introdução à Estatística, 10a. edição*. LTC.

Vapnik, V. (1998). *Statistical learning theory*. Wiley.

Witten, I. H., Neal, R. M., and Cleary, J. G. (1987). Arithmetic coding for data compression. *Commun. ACM*, **30**, 520–540.

Wives, L. K. (1999). *Um estudo sobre Agrupamento de Documentos Textuais em Processamento de Informações não Estruturadas Usando Técnicas de "Clustering"*. Master's thesis, Universidade Federal do Rio Grande do Sul.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

Yu, C., Cuadrado, J., Ceglowski, M., and Payne, S. J. (2008). Patterns in unstructured data.

A

Apêndice

A.1 Solicitações de mudança utilizadas no experimento

Tabela A.1: Conjunto de treinamento do GTK+

53375	55798	55846	55916	57047
57680	58330	58681	59058	61160
62887	63585	66735	69115	69586
72382	72784	73093	74221	84621
84668	86567	87482	89254	89491
89972	93738	97151	103869	104873
105126	105497	105654	106532	107068
107872	108433	108615	109510	109594
109823	110027	112388	113476	113547
114065	114351	115384	115837	116023
116501	118156	119722	119754	121955
122531	123107	125504	126270	126572
127698	130701	131335	131480	132975
133485	134031	135423	135649	135845
135905	136357	137796	137974	139095
139503	140412	142571	142634	143001
143647	144226	145232	145481	149013
Continua na próxima página				

Tabela A.1 – continuação da página anterior

149404	153168	153811	153828	155400
155881	155891	157087	157675	158459
160826	161409	161789	162447	163526
164128	164448	165108	165163	165180
165714	165770	165815	168326	168688
168737	169390	169647	169729	170611
171883	171885	172422	172584	302283
303389	304844	305511	307190	307419
308677	309301	311175	311508	311633
311803	313051	313344	313627	314278
314532	314627	314878	314882	315520
315993	316001	316419	317457	318781
319627	320431	320638	322291	322493
323146	323876	323995	325521	326341
326429	329604	330073	332756	333284
333291	334098	334423	335012	336770
336784	337910	340033	340063	340135
341091	342007	343330	343444	343841
343956	344290	344707	344897	345094
345106	345176	345320	346237	346420
347711	349277	349429	349501	350665
350806	352121	353962	354035	355128
355212	360112	365364	371756	372447
373137	378632	383354	385642	389358
389581	390423	392504	392532	393166
393255	395045	396493	400530	403165
410517	414875	418673	425985	426192
429732	430746	434329	435028	435053
436576	437493	438131	438651	452056
453673	453930	454596	456258	457774
459340	461483	463510	463865	467051
468832	474696	478637	486636	501992

Continua na próxima página

Tabela A.1 – continuação da página anterior

505268	506760	514471	530255	532262
536126	540529			

Tabela A.2: Conjunto de testes do GTK+

13432	50199	50574	50686	50707	50938	50994	51382
51536	52457	52790	53371	53375	53685	55035	55170
55584	55650	55798	55846	55916	55921	56130	56361
57047	57680	58161	58330	58570	58681	58929	59058
60272	60422	60818	60863	61160	61161	61285	61602
61672	62067	62196	62887	63539	63585	64459	64500
64832	64950	64977	65490	65505	65998	66467	66590
66735	68060	68084	69115	69336	69586	69900	70283
70453	71679	72382	72784	72858	73093	73359	74221
74575	83336	83386	83459	84621	84668	85859	85860
85872	86567	87482	87888	88814	89221	89254	89491
89972	91597	91619	92637	93389	93500	93730	93738
93758	93962	95026	95446	95829	96242	96516	97125
97136	97151	97224	97510	98167	99078	99115	100524
101185	101328	101614	102758	103869	104832	104873	104991
105126	105497	105654	106532	106975	106992	107068	107398
107495	107872	108243	108433	108614	108615	109510	109594
109737	109823	110027	110775	112388	113241	113476	113547
113924	114065	114351	114747	115263	115384	115837	116023
116501	117917	118097	118156	118810	118921	119722	119754
120799	121390	121955	122531	122800	123107	125504	125612
126193	126270	126369	126572	127698	128977	129463	130043
130353	130462	130587	130701	130718	131032	131335	131480
132223	132589	132590	132975	133485	134031	134162	134164
134235	135315	135358	135423	135649	135845	135888	135905
135906	136078	136357	136533	136596	136776	137616	137796

Continua na próxima página

Tabela A.2 – continuação da página anterior

137974	138560	138966	139095	139503	139586	139785	140126
140412	141621	142444	142571	142572	142634	142734	142778
142892	142998	143001	143113	143309	143395	143647	144070
144226	145232	145481	148518	148978	149013	149193	149194
149404	149657	149931	151331	151742	152188	152622	152744
153168	153224	153567	153768	153811	153828	153876	153896
154305	154341	154390	154392	154658	155400	155881	155891
156327	156809	157087	157675	157820	158177	158420	158423
158459	159034	159037	159217	159382	159389	160162	160377
160826	161109	161195	161409	161484	161789	162447	163465
163526	164108	164128	164147	164157	164448	164725	164905
165108	165163	165180	165581	165714	165770	165815	165823
166855	166934	167000	167088	167356	168074	168128	168326
168661	168688	168737	169390	169647	169677	169729	169812
170611	171490	171762	171883	171885	172312	172422	172584
172633	172653	301798	302283	302463	303024	303168	303389
303403	303473	303834	303940	304164	304844	305511	305535
305736	305832	305986	307055	307190	307419	308111	308145
308677	309291	309296	309301	309355	309464	309466	309718
309791	309868	309910	310522	311175	311465	311508	311612
311633	311803	312600	312695	313051	313247	313344	313475
313580	313595	313627	313628	314278	314418	314532	314627
314873	314878	314882	314921	315520	315993	316001	316008
316027	316121	316419	316424	316552	316712	317258	317457
317844	318444	318654	318781	319627	320138	320431	320638
320941	321523	322113	322291	322493	322516	322757	323146
323876	323995	324996	325199	325295	325521	325782	326341
326429	326916	328428	329604	329831	330073	330420	330840
331275	331306	331338	332059	332466	332756	333284	333291
333555	334098	334423	334617	334726	334742	335012	335912
336013	336770	336774	336784	336917	337603	337910	340031
340033	340063	340110	340135	340154	340870	340924	341091

Continua na próxima página

Tabela A.2 – continuação da página anterior

341158	341625	341730	341855	342007	342481	343233	343330
343444	343630	343841	343956	344239	344290	344528	344558
344707	344765	344897	345038	345094	345106	345176	345275
345320	345321	345666	346113	346237	346420	346605	346757
346970	347032	347037	347041	347043	347065	347066	347211
347224	347710	347711	348728	349277	349429	349501	350023
350072	350665	350806	351241	351759	351820	351910	352121
353423	353962	354035	354627	355126	355128	355134	355212
355961	356255	356630	357303	357538	358864	360112	360695
365364	367529	370909	371756	372447	373137	373706	373766
375436	378462	378632	381083	383354	385642	385672	389298
389358	389581	390423	391523	391725	392017	392504	392532
393166	393255	394000	394034	394144	395045	396493	399555
399809	400530	402144	402169	403165	403753	406160	408018
410517	412357	412484	414875	415677	417389	418634	418673
421988	425985	426192	426416	429732	430746	431067	432779
434308	434329	434535	435028	435053	436269	436576	437493
438131	438651	441767	442326	445196	446042	446107	446532
447396	447883	449492	451314	451634	452056	452278	453316
453673	453930	454596	454835	455268	455666	456258	457118
457774	459313	459340	459732	460207	461483	463510	463865
465144	465516	467051	468832	469068	469868	470033	472643
473340	473463	474282	474696	475439	477278	477447	478637
479384	482034	482837	484132	485887	486636	487323	489782
490624	496277	496734	496795	498922	500672	500686	501730
501992	502295	502850	503888	504749	505268	505857	506604
506760	507953	510074	510225	511163	513580	513811	514471
515596	515824	516083	517908	517954	518166	520989	522211
523930	526575	526635	526958	526987	527260	528822	530255
531129	531754	532262	532497	532558	533761	534979	535250
535427	536126	536542	536645	536765	538505	538686	539164
539814	540318	540379	540529	540861	540915	541351	541540

Continua na próxima página

Tabela A.2 – continuação da página anterior

543590	544863	545982	547655	547846	549236	552500	

Tabela A.3: Conjunto de treinamento do GIMP

6101	21028	55261	56194	56200
58507	61092	62988	71409	71478
72288	72862	72880	89825	92194
92469	92473	94979	97214	97734
103976	106474	106595	108034	108659
108958	109078	109475	109505	109648
110443	112273	113233	113239	117148
118115	119412	119735	119940	120031
120084	120600	121074	121783	124389
124600	125237	125668	126966	127673
127676	128112	128825	131146	132145
132621	132969	133266	134748	135866
136124	136321	136676	136909	136937
137566	139000	139868	140039	142697
142944	142965	142968	142996	143243
144788	145373	147483	149139	149558
149567	150177	150213	150679	151912
151914	152820	152948	153035	153229
153751	155328	155829	155963	156765
157513	157570	157915	158407	158605
159076	159304	159376	160045	160136
160247	161274	161573	162285	162385
163670	164116	164156	164195	164240
164281	165032	165774	167267	167762
168516	169066	170973	171306	171453
171497	171562	171646	171827	172682
300608	301028	303379	304148	304701
Continua na próxima página				

Tabela A.3 – continuação da p ágina anterior

305005	306617	306675	308658	308748
309373	310426	311965	315443	315964
316148	316212	316569	317137	317355
317992	318075	318644	318652	322210
323304	325617	326700	326718	331344
331436	331473	332933	335130	336259
338378	340073	341149	342390	343121
344871	345137	345214	345926	346754
347675	347861	347945	348243	349338
349341	351656	352341	353132	354034
355178	355761	356044	357059	357436
358203	358481	362096	365436	377917
382929	383870	384673	386061	389779
392111	395384	396169	398183	398185
398851	398913	400907	407214	414656
430917	443384	444964	445559	445719
450070	463096	466402	466593	469360
472518	473265	475966	475991	478618
478657	479893	480319	483887	485119
490785	493030	495564	499255	508114
511072	511965	512529	514082	521436
522483	526679	528160	528811	530077
530519	532853	533647	536582	537960
546159	546204	548631	552104	552625

Tabela A.4: Conjunto de testes do GIMP

3340	6101	12253	21028	34635	51164	51311	51631
51632	51633	51722	51884	52385	52866	53263	53278
55261	56193	56194	56200	57853	58507	59760	61092
62988	63880	64075	64360	64658	65194	67000	67808
Continua na próxima página							

Tabela A.4 – continuação da página anterior

71409	71478	72288	72324	72673	72862	72880	72881
74248	83458	83947	83968	85201	85374	87687	87797
88905	89093	89825	90629	91384	92194	92311	92469
92473	92476	93031	94732	94979	97214	97734	98490
99473	99738	100027	101435	103976	104693	105068	106008
106474	106595	107441	107556	107587	107949	107975	108034
108659	108958	109078	109114	109322	109434	109472	109475
109505	109561	109648	109680	109681	110014	110115	110173
110232	110443	110610	111188	111351	112012	112106	112115
112156	112273	113233	113237	113239	113240	113610	114225
115712	115755	115797	116171	116241	116698	116765	117148
118081	118104	118115	118796	119204	119236	119267	119412
119735	119940	120031	120084	120162	120600	121074	121783
122379	122692	123296	123336	123882	124389	124600	124698
124972	125148	125237	125274	125668	126366	126425	126941
126966	127259	127673	127676	128112	128156	128218	128594
128825	129678	130121	130869	130912	131109	131146	131431
131490	131634	131779	131975	132028	132032	132077	132113
132145	132297	132595	132621	132898	132969	133112	133180
133266	133280	134304	134371	134512	134748	134752	135866
136124	136199	136321	136497	136575	136676	136706	136909
136937	137057	137078	137435	137502	137508	137529	137566
137753	137767	138103	138237	138524	138776	138960	138980
139000	139069	139141	139337	139571	139868	139989	140039
140115	140296	141220	141535	141598	141733	141977	142277
142279	142280	142697	142944	142965	142968	142996	143125
143243	143850	143887	144046	144168	144267	144780	144788
144972	145373	146344	147483	147603	147662	148025	148026
148300	148731	148852	148853	149099	149139	149464	149558
149567	149669	149932	150177	150213	150676	150679	150917
151227	151308	151343	151638	151912	151914	151969	152531
152820	152879	152948	153035	153199	153229	153456	153751

Continua na próxima página

Tabela A.4 – continuação da página anterior

154651	155328	155829	155963	156545	156765	157508	157511
157513	157561	157570	157572	157580	157612	157722	157879
157915	157971	158407	158425	158605	158863	159010	159064
159076	159304	159376	159578	159579	159606	159659	159816
160032	160045	160121	160136	160247	160258	160339	160990
161274	161465	161508	161572	161573	161800	162285	162385
163381	163670	163721	164116	164156	164195	164218	164240
164281	164827	164914	165002	165032	165347	165367	165461
165633	165774	166650	166829	167002	167260	167267	167339
167506	167562	167762	167893	168516	168529	168936	168981
169066	169113	169274	169656	169882	169892	170397	170543
170812	170973	171306	171374	171453	171497	171562	171567
171646	171827	172051	172347	172589	172682	173039	300282
300582	300608	301006	301028	301033	301197	302328	302400
303118	303379	303972	304148	304685	304701	305005	305990
305995	306617	306675	308131	308658	308748	308754	309373
309657	309967	310126	310291	310426	311434	311960	311965
312646	313304	313634	314719	314734	314764	315106	315211
315443	315964	316148	316212	316555	316569	317137	317339
317355	317992	318075	318081	318540	318644	318652	318942
319029	320196	320461	320462	320977	320985	321100	321511
321735	322210	322296	322343	323304	323404	324309	325268
325324	325465	325617	325745	325794	326700	326718	326988
327523	327681	328320	328683	329004	329007	329468	329987
330864	331086	331344	331436	331473	332620	332933	333401
335130	335652	335715	336259	336402	336498	338378	338684
338792	339481	339560	339759	340073	340771	341149	341319
341483	341923	342390	342455	342456	343121	343911	344871
345137	345214	345551	345926	346016	346754	346755	346878
346988	347103	347104	347105	347109	347195	347339	347675
347861	347945	348243	348249	348317	349338	349341	349614
350009	351175	351656	352214	352341	352609	353132	353381

Continua na próxima página

Tabela A.4 – continuação da página anterior

354034	354668	355178	355761	356044	356137	356260	356353
356354	356643	357059	357223	357424	357436	358203	358314
358481	358824	359833	360458	361672	362096	362586	362877
363381	363534	364852	365226	365436	367532	369656	373254
375606	377098	377811	377917	378266	378738	382929	383870
384096	384673	386061	389201	389779	392111	395384	396169
396513	398183	398185	398188	398851	398913	400389	400907
401145	401182	403449	403580	407214	410387	413347	413906
414656	417465	418284	419290	421466	430146	430917	431845
433251	433772	433902	434279	434532	435017	436815	438886
439222	443384	444960	444964	445559	445719	446681	447402
448784	449141	449729	450001	450070	451272	452836	454011
454364	457045	457286	460878	463096	463623	463729	465669
466189	466402	466593	466923	467933	468336	469360	469708
470914	471051	471565	472518	473265	475553	475966	475991
478385	478618	478657	479389	479817	479893	479974	480303
480319	480796	480799	482289	482743	483003	483887	483896
484123	485119	486283	486517	486779	489813	490198	490364
490785	490827	491272	491311	493030	495564	496772	497724
498282	498511	499161	499255	499329	500826	502374	503124
506110	507572	508114	509608	510294	511072	511965	512126
512529	514082	517285	518012	521202	521436	522087	522483
523873	524017	525747	526256	526679	528160	528811	528958
529434	530077	530216	530519	532485	532853	533356	533647
535039	536582	537960	539949	542350	542972	546159	546204
548631	549690	550911	550983	551231	552104	552127	552601
552625							

Tabela A.5: Conjunto de treinamento do Hadoop

1849	1886	3315	4675	6080
Continua na próxima página				

Tabela A.5 – continuação da página anterior

6097	6139	6145	6215	6231
6255	6269	6298	6312	6315
6344	6421	6428	6432	6460
6472	6482	6495	6496	6498
6526	6536	6562	6580	6581
6584	6586	6596	6599	6600
6603	6613	6620	6623	6627
6632	6633	6637	6642	6644
6647	6648	6649	6652	6656
6663	6667	6669	6670	6674
6682	6683	6687	6693	6706
6710	6714	6715	6724	6730
6747	6754	6756	6758	6760
6761	6763	6764	6781	6787
6796	6803	6814	6818	6825
6832	6833	6834	6835	6845
6853	6856	6859	6861	6862
6864	6870	6877	6879	6884
6885	6888	6889	6890	6892
6898	6899	6900	6903	6904
6905	6907	6912	6913	6919
6920	6922	6925	6926	6930
6932	6933	6934	6938	6939
6940	6943	6944	6947	6949
6951	6965	6975	6977	6978
6984	6987	6989	6991	6994
6995	6996	7001	7006	7009
7010	7011	7013	7014	7023
7024	7032	7034	7045	7046
7048	7049	7057	7058	7060
7070	7078	7082	7087	7091
7093	7096	7101	7102	7104

Continua na próxima página

Tabela A.5 – continuação da página anterior

7110	7112	7114	7121	7122
7126	7129	7131	7133	7140
7144	7145	7146	7151	7153
7156	7159	7162	7166	7171
7172	7174	7175	7180	7183
7187	7189	7193	7194	7202
7207	7208	7210	7214	7215
7216	7223	7224	7227	7229
7231	7233	7235	7236	7237
7238	7241	7245	7249	7250
7251	7257	7258	7265	7267
7268	7271	7272	7275	7282
7285	7286	7287	7290	7292
7296	7300	7301	7306	7316
7318	7320	7322	7323	7329
7333	7336	7337	7341	7342
7346	7349	7351	7353	7355

Tabela A.6: Conjunto de testes do Hadoop

213	338	650	776	789	795	938	1013
1033	1316	1540	1681	1740	1741	1790	1792
1798	1823	1831	1849	1862	1881	1886	1970
1972	1977	1979	2036	2043	2053	2154	2159
2281	2409	2565	2602	2642	2647	2661	2663
2672	2697	2707	2786	2795	2798	2849	2850
2851	2859	2868	2881	2948	2950	2962	2963
2964	2965	2967	2968	2969	2977	2987	2989
2999	3005	3013	3017	3020	3026	3038	3077
3133	3142	3145	3188	3192	3197	3207	3216
3276	3278	3284	3290	3305	3311	3312	3315
Continua na próxima página							

Tabela A.6 – continuação da página anterior

3367	3371	3389	3394	3435	3457	3472	3474
3475	3522	3525	3526	3550	3565	3621	3654
3676	3701	3835	3874	3912	3920	3967	3979
4013	4019	4025	4043	4162	4164	4170	4212
4214	4221	4235	4304	4313	4318	4389	4413
4441	4460	4461	4462	4463	4487	4573	4581
4617	4626	4674	4675	4709	4711	4733	4736
4745	4762	4773	4802	4816	4851	4856	4882
4893	4901	4913	4926	4938	4989	5029	5047
5051	5054	5055	5057	5114	5118	5125	5141
5158	5177	5182	5190	5210	5220	5228	5254
5265	5268	5271	5277	5301	5303	5311	5313
5318	5330	5339	5358	5360	5367	5370	5373
5383	5387	5393	5397	5401	5409	5411	5412
5441	5448	5529	5533	5539	5560	5574	5597
5611	5612	5623	5629	5636	5641	5646	5647
5648	5654	5655	5665	5673	5678	5685	5688
5691	5701	5711	5714	5718	5719	5726	5736
5746	5749	5753	5759	5769	5777	5800	5802
5803	5807	5813	5828	5833	5850	5852	5869
5871	5876	5880	5881	5882	5883	5884	5885
5898	5903	5909	5920	5921	5932	5934	5937
5959	5983	5990	6026	6028	6032	6043	6056
6057	6067	6077	6080	6091	6097	6111	6118
6125	6126	6127	6128	6134	6136	6139	6141
6145	6147	6164	6168	6178	6183	6190	6202
6213	6215	6231	6236	6248	6255	6259	6269
6277	6284	6298	6304	6312	6315	6328	6331
6333	6335	6344	6348	6352	6355	6361	6369
6376	6381	6382	6388	6393	6397	6401	6421
6427	6428	6432	6438	6450	6453	6455	6460
6463	6469	6472	6482	6493	6495	6496	6498

Continua na próxima página

Tabela A.6 – continuação da página anterior

6500	6506	6508	6511	6512	6523	6524	6526
6529	6530	6536	6539	6542	6544	6554	6557
6562	6564	6571	6575	6576	6580	6581	6584
6586	6588	6596	6599	6600	6603	6611	6613
6617	6618	6620	6623	6625	6627	6628	6632
6633	6637	6638	6639	6642	6644	6647	6648
6649	6652	6653	6655	6656	6660	6661	6662
6663	6667	6669	6670	6674	6681	6682	6683
6687	6693	6706	6708	6710	6714	6715	6718
6721	6724	6730	6734	6735	6741	6743	6746
6747	6749	6754	6756	6758	6760	6761	6763
6764	6770	6773	6774	6780	6781	6786	6787
6789	6793	6796	6803	6810	6811	6812	6814
6818	6825	6827	6832	6833	6834	6835	6843
6845	6853	6855	6856	6859	6861	6862	6864
6868	6870	6872	6877	6879	6883	6884	6885
6887	6888	6889	6890	6892	6898	6899	6900
6903	6904	6905	6907	6912	6913	6914	6917
6919	6920	6921	6922	6923	6925	6926	6928
6930	6932	6933	6934	6938	6939	6940	6943
6944	6947	6948	6949	6950	6951	6953	6954
6955	6956	6960	6965	6970	6971	6975	6977
6978	6984	6985	6987	6988	6989	6990	6991
6992	6993	6994	6995	6996	6998	7001	7006
7007	7008	7009	7010	7011	7013	7014	7019
7023	7024	7028	7032	7033	7034	7038	7042
7045	7046	7047	7048	7049	7052	7053	7055
7057	7058	7060	7068	7070	7072	7075	7078
7082	7087	7091	7093	7094	7096	7097	7098
7100	7101	7102	7104	7108	7110	7112	7113
7114	7117	7118	7120	7121	7122	7126	7128
7129	7131	7132	7133	7134	7136	7140	7143

Continua na próxima página

Tabela A.6 – continuação da página anterior

7144	7145	7146	7151	7152	7153	7154	7156
7159	7162	7166	7167	7170	7171	7172	7174
7175	7179	7180	7183	7184	7187	7189	7190
7192	7193	7194	7201	7202	7207	7208	7210
7212	7214	7215	7216	7223	7224	7225	7226
7227	7229	7230	7231	7232	7233	7235	7236
7237	7238	7241	7244	7245	7246	7247	7248
7249	7250	7251	7253	7257	7258	7259	7261
7265	7267	7268	7271	7272	7274	7275	7276
7277	7282	7283	7285	7286	7287	7288	7289
7290	7291	7292	7296	7300	7301	7302	7306
7312	7316	7318	7320	7322	7323	7325	7329
7330	7331	7333	7335	7336	7337	7341	7342
7343	7345	7346	7349	7351	7353	7355	7356
7364	7373						