Apoio à Rastreabilidade no contexto de Evolução de Software: um mapeamento Sistemático

Claudio A. S. Lelis[1]

[1]Programa de Pós Graduação em Ciência da Computação (PGCC) - Universidade Federal de Juiz de Fora (UFJF) - R. José Lourenço Kelmer - Martelos - 36.036-330 - Juiz de Fora - MG - Brasil

lelis@ice.ufjf.br

Resumo. *A Evolução de software tem se destacado como um dos principais tópicos na engenharia de software e manutenção. A ratreabilidade de software visa destacar o relacionamento entre os artefatos, necessário para a correta compreensão do software. Dentro deste contexto, o respectivo artigo propõe um mapeamento sistemático da literatura, com o objetivo de identificar e classificar publicações relevantes sobre o assunto, sugerindo pontos de discussão.*

1. Introdução

O presente trabalho contém um estudo baseado em mapeamento sistemático, com intuito de identificar e classificar a literatura relevante sobre o assunto, sugerindo pontos de discussão, bem como, identificar e relacionar ferramentas, contempladas pela literatura técnica, que oferecem suporte à geração de rastreabilidade no código fonte, no contexto de manutenção e evolução de software.

O mapeamento sistemático é um mecanismo que permite ao pesquisador fornecer uma visão geral sobre a área de pesquisa que está investigando, levando em conta fatores como a quantidade de pesquisas publicadas sobre a área pesquisada, e os tipos de pesquisa disponíveis [Steinmacher e Gerosa, 2012]. Já um protocolo de mapeamento é o método que o pesquisador define na formalização de buscas sobre a área de pesquisa alvo de investigação [Kitchenham, 2004].

O documento está organizado da seguinte forma: Seção 2 descreve o planejamento do mapeamento sistemático. Seção 3 apresenta a condução do mapeamento e na Seção 4, as análises realizadas a partir das questões estabelecidas. Seção 5 considera as ameaças à validade desta pesquisa. Por fim, a Seção 6 apresenta as considerações finais.

2. Planejamento do mapeamento sistemático

O protocolo desenvolvido foi constituído de objetivo, questão de pesquisa, PICOC (*Population, Intervention, Control, Outcome, Context*), *string* de busca, critérios de

inclusão e exclusão de artigos, critérios de seleção das bases de dados de publicações e as bases escolhidas. Esta seção está organizada conforme o protocolo utilizado.

O objetivo deste trabalho é identificar e relacionar ferramentas, contempladas pela literatura técnica, que oferecem suporte à geração de rastreabilidade no código fonte, no contexto de manutenção e evolução de software. As técnicas, métodos e abordagens implementadas ou utilizadas por ferramentas era um objetivo secundário.

2.1 Questões de pesquisa

Mediante o objetivo apresentado, foram derivadas 2 questões de pesquisa que nortearam o estudo. Estas perguntas definem as faces de classificação do estudo e abordarão temas relacionados à rastreabilidade de software e a utilização de métodos, ferramentas e técnicas voltados para esta área.

• *Q1: Quais ferramentas promovem a rastreabilidade de software em artefatos de código fonte?*

Esta questão tem como objetivo identificar as ferramentas atuais utilizadas para a rastreabilidade de software e geralmente são usadas para resolver a geração, manutenção e armazenamento das ligações de rastreabilidade.

• *Q2: Quais técnicas, métodos, abordagens as ferramentas aplicam/utilizam para gerenciar a rastreabilidade?*

Esta questão tem como objetivo identificar as técnicas e métodos aplicados e abordagens utilizadas pelas ferramentas que geralmente são usadas para resolver problemas para gerenciar a rastreabilidade e manter seus relacionamentos.

2.2 Estratégia de pesquisa

Os estudos primários selecionados devem contribuir diretamente para a resposta das questões de pesquisa do mapeamento para que esses trabalhos não sejam desconsiderados pela estratégia de busca [Kitchenham, 2004].

A definição das palavras-chave de busca, utilizadas na pesquisa, foram feitas com base na estratégia PICOC [Petticrew and Roberts, 2006], conforme Tabela 1.

Tabela 1. PICOC

PICOC	**KEY WORDS**
População (*Population*)	*Configuration management, Version control, Versioning, source code*
Intervenção (*Intervention*)	*Traceability, Traceability Management*
Controle (*Control*)	Não definido
Resultado (*Outcome*)	*Tool, plugin, Approach*
Contexto (*Context*)	*Software Maintenance, Software Evolution*

Mediante as palavras-chave identificadas foi possível derivar uma *string* de busca geral, que foi usada de base para gerar as *strings* específicas contemplando as particularidades sintáticas de cada base escolhida:

((configuration AND management) OR (version AND control) OR (source AND code)) AND ((traceability) OR (traceability AND management)) AND (tool OR plugin OR approach) AND ((maintenance) OR (software AND evolution)).

Para a seleção dos resultados foram definidos critérios para a inclusão e exclusão de artigos, os quais auxiliaram a avaliação para seleção dos artigos. Os critérios adotados são dispostos na Tabela 2.

Tabela 2. Critérios de inclusão e exclusão de artigos

Tipo	Descrição
Inclusão	Trabalhos que apresentem somente estudos experimentais, estudos de caso ou avaliações de alguma ferramenta, serão incluídos bem como o trabalho de origem.
Inclusão	Estudos que apresentem técnica ou método, que foi implementado ou em uso numa ferramenta.
Exclusão	Capítulos de livro, chamada para congressos.
Exclusão	Estudos que não puderem ser acessados completos.
Exclusão	Estudos que não tratem da rastreabilidade na área de Ciência da Computação.
Exclusão	Estudos relacionados a código fonte, mas não à rastreabilidade.
Exclusão	Estudos que não apresentem uma ferramenta, *plugin* ou abordagem para promover, gerenciar rastreabilidade no código fonte.
Exclusão	Estudos relacionados à rastreabilidade, mas não no código fonte.

Para selecionar as bases de busca por publicações, certos critérios foram levados em consideração. As bases adotadas no mapeamento cumprem os seguintes requisitos:

- são capazes de interpretar expressões lógicas e similares;
- permitem busca pelo texto completo ou em campos específicos (titulo, *abstract*);
- estão disponíveis no instituto dos pesquisadores;
- cobrem a área de pesquisa em interesse por este mapeamento: ciência da computação;
- não possuem limite de palavras e combinações na *String* a ser buscada.

De acordo com esses requisitos foram utilizadas as bases Scopus, ScienceDirect, IEEEXplore, disponíveis no portal de periódicos da CAPES, acessíveis na UFJF e também são sugeridas em revisões sistemáticas no domínio de engenharia de software e recomendadas por Kitchenham (2004).

3. Condução do Mapeamento Sistemático

Foi utilizado para o armazenamento de artigos, a ferramenta desenvolvida no contexto de mestrando da UFJF, de nome Parsifal[1]. Esta ferramenta importa padrões ***bibTex*** e é utilizada por diversos pesquisadores em suas revisões sistemáticas. A escolha da ferramenta se deu em função dos recursos disponibilizados pela mesma, os quais são descritos abaixo. A ferramenta está disponível na Web, facilitando o processo de uso, sem a necessidade de instalação de dependências e quaisquer requisitos não funcionais, como, por exemplo, máquinas virtuais (Java), utilizadas em ferramentas tradicionais.

Por meio da ferramenta podemos criar um projeto de revisão sistemática, dividido em 4 (quatro) passos, os quais encontram-se dispostos e explicados abaixo:

Review – Dados da revisão como Titulo, Autor, Descrição da revisão.

Planning – São definido os objetivos, as questões de pesquisa o PICOC, as palavras chaves, String de Busca e as bases desejadas, bem como os critérios de inclusão e exclusão.

Conducting – São definidos nesse passo, os processos de condução do revisão. Pode-se importar os arquivos padrão ***bibTex*** das bases de dados e fazer a exclusão dos artigos não desejáveis. São apresentados os artigos de uma forma geral ou separados por base o que permite a classificação e filtragem dos mesmos. E possível também acessar o abstract dos artigos bem como obter informações relevantes como Titulo, Autor, Meio de Publicação e Ano de escrita.

Reporting – É possível por meio dessa opção obter os gráficos de artigos por base, artigo aceitos e rejeitados, bem como o gráfico de artigos por ano.

Os artigos que foram importados na ferramenta supracitada, foram extraídos das bases já anteriormente citadas. A maioria das bases permitem a seleção de todos os artigos resultantes da *string* de busca aplicada, e sua exportação para um único arquivo no formato ***bibTex.*** Após a exportação dos artigos para o formato, foram importados para a ferramenta Parsifal, todos os artigos separados por base.

Após esses passos, foi possível, realizar diversas análises e o prosseguimento do processo adotado no mapeamento sistemático. O resultado das importações, bem como o gráfico de valores totais por base e os critérios de exclusão e inclusão, são detalhados na seção seguinte.

3.1 Execução da pesquisa

Após a conclusão do protocolo definido na seção anterior, foi iniciada a execução do protocolo conforme descrita nessa seção, que é dividida de acordo com a execução de cada etapa do protocolo definido.

3.1.1 Execução da busca nas bases

A pesquisa abrangeu todos os trabalhos publicados até 2015 (inclusive). A *String* de busca definida foi calibrada e aplicada em cada base de dados digital. A *String* em cada base foi aplicada aos campos título do artigo, *abstract* e palavras-chave. A distribuição

[1] HTTP://parsif.al

dos artigos retornados entre as bases é apresentada na Tabela 3, indicando que a primeira pesquisa devolveu **180** resultados.

Tabela 3 – Resultados obtidos pela *string* de busca

Base	Retorno
IEEEXplore	74
ScienceDirect	9
Scopus	97

3.1.2 Execução do processo Exclusão

O primeiro passo do processo de inclusão e exclusão foi apoiado pela ferramenta Parsifal que identificou, automaticamente, um total de 50 artigos duplicados, e desta forma foram excluídas as duplicações.

A segunda etapa foi a aplicação dos critérios de inclusão e exclusão estabelecidos no protocolo sobre os 130 artigos resultantes da etapa anterior. Nessa etapa, os campos analisados foram o título , o abstract (resumo) e a introdução, desta forma aprofundando o nível de detalhamento a respeito de cada artigo proporcionando uma análise mais criteriosa e embasada. Foram excluídos um total de 48 artigos. O gráfico da Figura 2 mostra o número de artigos excluídos para cada critério de exclusão.

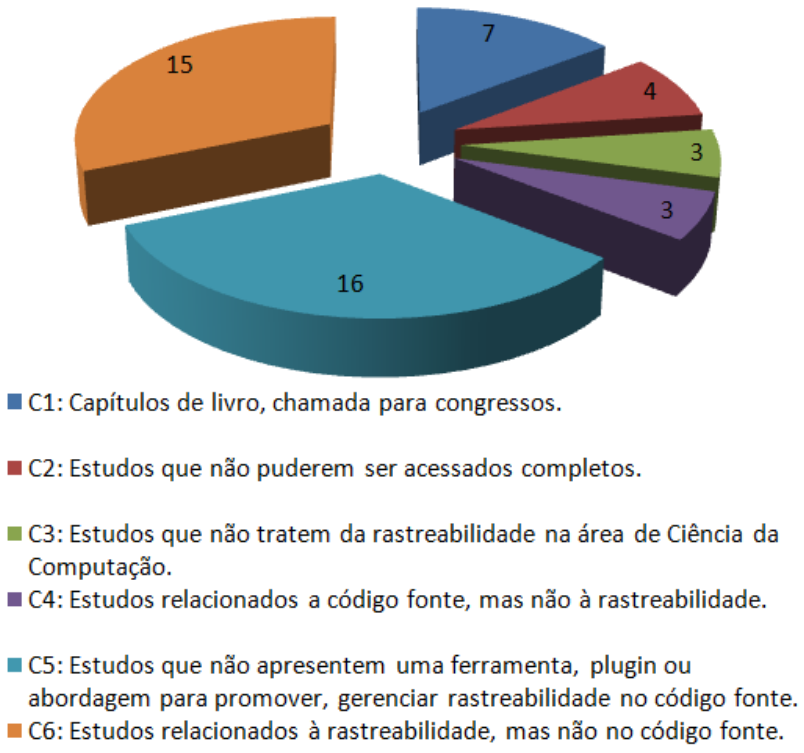

Figura 2 – Artigos excluídos por critério

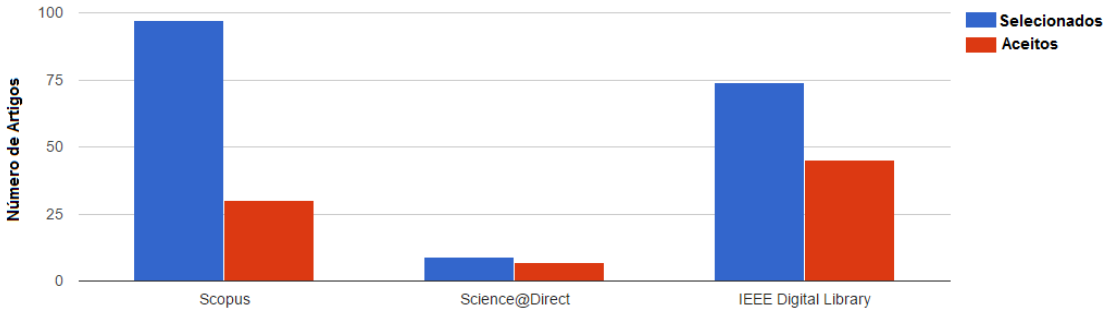

Figura 3 – Artigos Aceitos por base

Enquanto a Figura 3 demonstra o número de artigos totais aceitos dentre os selecionados em cada base.

4. Apresentação dos Resultados

Mediante a finalização do processo de seleção dos artigos, foi realizada uma análise quanto aos resultados obtidos. Na sequência serão descritas as análises efetivadas, bem como as constatações obtidas por meio desta verificação.

4.1 - Variações de publicações ao longo do ano:

O gráfico na Figura 4 demonstra a variação do número de publicações ao longo dos anos.

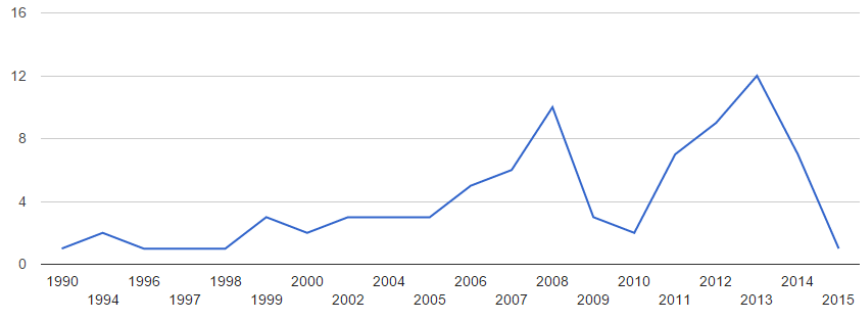

Figura 4 – Gráfico de publicações por ano

Analisando as informações pode-se notar que o assunto de rastreabilidade sempre esteve em voga e ganhou atenção nos últimos anos, com o crescimento da complexidade dos sistemas e a necessidade de se manter o relacionamento entre os artefatos.

4.2 Análise dos Resultados – Questão 1

As ferramentas e abordagens obtidas foram avaliadas e catalogadas, objetivando responder a primeira questão de pesquisa definida no protocolo. Foi possível identificar ferramentas com foco diferenciado quanto aos artefatos envolvidos nos *links* de

rastreabilidade. Esta informação foi útil para auxiliar na definição de grupos e facilitar a representação dos resultados. Cada subseção representa um destes grupos identificados.

Q1: Quais ferramentas promovem a rastreabilidade de software em artefatos de código fonte?

4.2.1 Código e Requisitos

A rastreabilidade dos requisitos garante que o código fonte permanece consistente com a documentação e que todos os requisitos foram implementados. Com a evolução do software, recursos são adicionados, removidos ou modificados, e o código se afasta de suas exigências originais. Assim, as abordagens de recuperação de rastreabilidade torna-se necessária para restabelecer as relações de rastreabilidade entre os requisitos e código fonte. Esta seção apresenta trabalhos que relaciona o código fonte com os requisitos elicitados e demais artefatos gerados na fase de levantamento de requisitos do ciclo de desenvolvimento do Software.

Coparvo é uma abordagem apresentada em [Ali et al. 2011] complementar para abordagens existentes de recuperação de rastreabilidade para programas orientados a objetos. Reduz os links falsos positivos recuperados pelos processos tradicionais de recuperação de rastreabilidade, reduzindo assim o esforço de validação manual. Coparvo assume que as informações extraídas de diferentes entidades (ou seja, nomes de classe, comentários, variáveis de classe, ou assinatura de métodos) são diferentes fontes de informação, eles podem ter diferentes níveis de confiabilidade em na rastreabilidade de requisitos e cada fonte de informação pode agir como um especialista recomendando ligações diversas.

STRADA (Scenario-based TRAce Detection and Analysis) [Egyed et al. 2007] é uma ferramenta que ajuda os engenheiros de software a explorar links de rastreabilidade para o código fonte por meio de testes. Embora o teste é predominantemente feito para garantir a correção de um sistema de software, STRADA demonstra um benefício secundário vital: ao executar o código-fonte durante o teste ele pode ser ligado aos requisitos e às features, estabelecendo assim, a rastreabilidade automaticamente.

Trustrace [Ali et al. 2013] é uma abordagem que foi implementada e recupera rastreabilidade entre requisitos e código fonte. Usa fontes heterogêneas de informação para descartar e reclassificar os *links* indicados através de expressões regulares e correspondência sintática entre termos. Composta por 3 partes. É capaz de minerar repositórios de software (CVS e SVN) para relacionar requisitos e código fonte utilizando comentários dos *commits*. Possibilita a mineração também de dados de sistemas de solicitação de mudança. Possui processamento automatizado por meio da sequência de restrições impostas para aceitar ou recusar os *links* sugeridos.

Em [Penta et al. 2002] é proposta uma abordagem e um processo para recuperar ligações de rastreabilidade entre código fonte e documentos de texto livre em um sistema de software desenvolvido com o uso extensivo de COTS, middleware e código gerado automaticamente. A abordagem baseia-se em um processo que filtra informações recolhidas a partir de artefatos de baixo nível. Essa filtragem de informação é realizada de acordo com uma taxonomia de fatores que afetam os métodos de recuperação de links de rastreabilidade. Para automatizar o processo de recuperação de rastreabilidade algumas ferramentas tiveram de ser implementadas: um script que extrai dos requisitos duplas de palavras e ocorrências além de interpretar código fonte

C++ e extrair identificadores e suas ocorrências. Outro script que retira do requisito e código expressões indesejadas (artigos números preposições). Um analisador morfológico baseado em um tessauro construído pelo dissionário de termos presentes e extraído dos requisitos e do código. Um script para remover os identificadores gerados automaticamente do código fonte. E enfim, uma ferramenta que implementa a estimativa de probabilidade que calcula a proximidade do vocabulário.

4.2.2 Código e Modelos

Relaciona o código fonte com os diferentes modelos e demais artefatos gerados na fase de projeto do ciclo de desenvolvimento do Software.

[Fiutem e Antoniol, 1998] e [Antoniol et al. 1999] Apresenta uma abordagem para verificar a conformidade do modelo OO com relação ao código fonte. O processo funciona em artefatos de design expressos em notação OMT e aceita código fonte C++. Ele recupera o projeto do código em formato "as-is", compara o projeto recuperado com o projeto real e ajuda o usuário a lidar com as incoerências, apontando as regiões de código que não combinam com o design. O processo de recuperação explora expressão regular e edição e cálculo de distância para preencher a lacuna entre o código e design. Segundo o mesmo autor em outro trabalho [Antoniol et al. 1999] a saída é uma medida de similaridade associado a cada classe compatível, além de um conjunto de classes incompatíveis. A visualização gráfica do projeto com cores diferentes associadas a diferentes níveis de compatibilidade é fornecido como um suporte para atualizar o projeto e melhorar a sua rastreabilidade ao código.

DeMIMA [Guéhéneuc e Antoniol, 2008] é uma abordagem para identificar semi-automaticamente micro-arquiteturas que são semelhantes a decisões de projeto no código fonte e para assegurar a rastreabilidade desses micro-arquiteturas entre a implementação e design. DeMIMA consiste em três camadas: duas camadas objetivam recuperar um modelo abstrato do código fonte, incluindo relações de classes em binário, e uma terceira camada para identificar padrões de design no modelo abstrato.

srcTracer [Hammad et al. 2011] é uma ferramenta para identificar automaticamente as alterações de *design* provenientes de alterações no código. A ferramenta descobre quando uma alteração de código em particular pode ter quebrado a rastreabilidade para o projeto e dá detalhes sobre o que mudou no *design*. A partir destes resultados, o documento de *design* pode ser atualizado manualmente. Utiliza a representação srcDiff para identificar o que mudou no código fonte.

Uma abordagem é apresentada [Mirakhorli et al. 2012] para o rastreamento de requisitos arquiteturais significativos, especificamente aqueles implementados com o uso de soluções comuns. No primeiro passo do algoritmo um classificador identifica todas as classes relacionadas com uma determinada solução, e, em seguida, estabelece o mapeamento entre a solução relevante e essas classes. Um classificador mais rigoroso é utilizado em conjunto com a análise estrutural para identificar o subconjunto de classes que desempenham papéis claramente definidos na solução (por exemplo, emissor e receptor). Essas classes detectadas são então mapeadas para Modelos de informações de rastreabilidade (tTIMs) que apoiam o processo de rastreabilidade e conectam as classes classificadas para um conjunto relevante de decisões do projeto, requisitos e outros

artefatos relacionados. Um plugin para o Eclipse estava em processo de finalização para implementar o algoritmo.

4.2.3 Código e Código

Relaciona o código fonte com outros componentes de código como documentos, módulos e demais artefatos gerados na fase de codificação do ciclo de desenvolvimento do Software.

4.2.4 Código e Teste

Relaciona o código fonte com os casos de teste e demais artefatos gerados na fase de teste do ciclo de desenvolvimento do Software.

É apresentado uma abordagem em [Qusef et al. 2010] baseado em Analise de Fluxo de Dados (Data Flow Analysis (DFA)) com o objetivo de recuperar links de rastreabilidade entre os casos de teste unitário e o código fonte. Em particular, a abordagem recupera como classes testadas, todas as classes que afetam o resultado da última instrução executada em cada método da classe de teste unitário.

4.2.5 Código e Mudança

Relaciona o código fonte com as requisições de mudança e demais artefatos gerados na fase de manutenção do Software.

Em [Sureka et al. 2011] é apresentado uma abordagem para integrar as duas bases de dados provenientes do sistema de controle de versão (SVN, CVS) e do sistema de gerenciamento de defeitos (BugZilla, JIRA). A proposta é baseada no modelo Fellegi-Sunter (FS) e ultrapassa algumas desvantagens identificadas a partir da análise de algumas abordagens anteriores.

4.2.6 Código e Artefatos

Relaciona o código fonte com dois ou mais tipos de artefatos de diferentes fases do ciclo de desenvolvimento, elencados nas categorias anteriormente apresentadas.

RayTracer [Gardner 1994] é uma ferramenta que fornece serviços de rastreabilidade para o processo de desenvolvimento de software. Engenharia de software avançado requer serviços de rastreabilidade que satisfaçam as exigências de dados de software, ferramentas e equipes. Fornece suporte a dados como documentos digitados, ancoras e links entre componentes. Estende as notações de várias ferramentas de desenvolvimento para permitir aos usuários fazerem anotações em seus artefatos na forma de ancoras e links. Apoia os membros da equipe permitindo que acessem os dados simultaneamente gerados pelo processo de desenvolvimento e os links e ancoras arquivados no repositório da ferramenta.

Radix [Ni et al. 1994] Radix é uma ferramenta para usuários que preparam documentos no ambiente UNlX. Um conjunto de comandos Radix é inserido em um documento de entrada para destacar entidades que podem ser: requisitos individuais, modelos, casos de teste, etc. Estes comandos suportam atributos como palavras-chave, referências, pesos, proprietários, etc. Posteriormente, a rastreabilidade é construída com base neste mecanismo de marcação e um banco de dados relacional associada à ferramenta.

Existem vários utilitários disponíveis para numerar automaticamente as etiquetas e consultar os documentos marcados. Ferramentas também estão disponíveis para gerar o mapeamento gráfico de todas as entidades em um conjunto de documentos e está ligada ao sistema de mensagens de e-mail, possibilitando avisar as mudanças aos responsáveis.

Em [Antoniol et al. 1999] é apresentada uma abordagem para estabelecer e manter vínculos de rastreabilidade entre o código-fonte e documentos de texto livre. A premissa é que os programadores usam nomes significativos para os itens do programa, tais como funções, variáveis, tipos, classes e métodos. Acreditamos que o conhecimento das aplicações do domínio que os programadores processam ao escrever o código é muitas vezes capturado pelos mnemônicos para identificadores; portanto, a análise desses mnemônicos pode ajudar a associar conceitos de alto nível com os conceitos do programa, e vice-versa. A abordagem é aplicada a software escrito em linguagem C ++, para relacionar classes a seções do manual. A abordagem também foi aplicada [Antoniol et al. 2000] à linguagem Java, para traçar o relacionamento entre as classes e os requisitos funcionais.

Em [Lago et al. 2004] apresentam uma extensão de uma ferramenta de software comercial que possui integração com módulos de implementação JAVA, para suportar rastreabilidade top-down bem como bottom-up em famílias de produtos. Sendo possível rastrear desde o mapa de features da família até os arquivos de implementação.Com o código fonte, a rastreabilidade é feita entre o mapa de componentes do produto e os arquivos de implementação pela associação de cada decisão de projeto com os artefatos de implementação que os resolvem como: componentes executáveis, módulos de código fonte e arquivos de documentação associados. Ainda é estabelecido rastreabilidade entre os *features* da família e os *features* do produto e, entre o *feature* do produto e decisões de projeto, como componentes, classes e interfaces. A migração para o ambiente Eclipse ainda não era uma realidade.

Kagdi et al. (2007) apresentam uma abordagem baseada em heurísticas para gerar e manter *links* de rastreabilidade entre artefatos de software, através da mineração e análise do histórico de versões de um software. A abordagem consiste em descobrir conjuntos de artefatos (por exemplo, código fonte e documentação) que mudam com frequência. A ideia defendida é que, se *commits* são feitos para diferentes tipos de arquivos, em conjunto, e com alta frequência, então existe uma alta probabilidade de que haja um vínculo de rastreabilidade entre eles.

ADAMS RE-TRACE [De Lucia et al. 2008] é uma ferramenta para recuperação de rastreabilidade baseada na técnica de recuperação de informação LSI [Deerwester et al. 1990]. A ferramenta foi desenvolvida, integrada ao ADAMS [De Lucia et al. 2006] um sistema gerenciador de artefatos de baixa granularidade, como um *plugin* do ambiente Eclipse. No código fonte, *links* de rastreabilidade são gerados no nível de classe, em outros artefatos gerenciados como casos de teste e casos de uso, um nível baixo de granularidade pode ser alcançado. A geração dos *links* é feita de modo semiautomático já que o usuário precisa especificar inicialmente a categoria e os artefatos a serem analisados e como a lista de *links* candidatos será formada. Em seguida, deve-se confirmar ou eliminar os *links* candidatos como julgar adequado. É possível armazenar os dados evitando duplicação de dados e processamento desnecessário.

TYRION é uma abordagem que foi instanciada e apresentada em [Diaz et al. 2013] capaz de recuperar links de rastreabilidade entre documentação e código fonte através

da informação de propriedade de código, ou seja, qual desenvolvedor manipulou certo trecho de código. Inicialmente, se extrai o autor de cada componente de código-fonte e para cada autor identifica-se o "contexto" que ele trabalhou. Assim, para uma dada consulta a partir da documentação externa é calculada a semelhança entre ele e o contexto dos autores. É utilizada uma técnica padrão baseada em IR para recuperar as classes que se relacionam a uma consulta específica, e são recompensadas todas as classes desenvolvidas pelos autores que tenham o seu contexto mais semelhante à consulta, impulsionando sua semelhança com a consulta formando um ranking.

Lehnert et al. (2013) apresentam uma abordagem holística que combina análise de impacto, modelagem multi-perspectiva, e análise horizontal de rastreabilidade e suporta modelos de projeto, código fonte e casos de teste. A abordagem foi implementada na ferramenta EMFTrace que suporta código fonte Java, casos de teste JUnit e modelos UML, URN, OWL, e modelos de features. Apesar de automática um esforço manual é necessário na criação e manutenção da detecção de dependências e regras de análise de impacto. É baseado em EMF e o EMFStore que fornecem um meta-modelo homogêneo e um modelo de repositório, respectivamente.

MOTCP [Islam et al. 2012] acrônimo de Multi-Objective Test Case Prioritization é uma ferramenta que implementa uma técnica de priorização de teste multi-objectivo com base numa análise de três dimensões de casos de teste: (i) estrutural; (ii) funcional; e (iii) o custo. Considera ao mesmo tempo informações estruturais e funcionais. Identifica a ordenação de casos de teste como um problema de otimizão multi-objetivo para equilibrar as dimensões consideradas no que diz respeito às ligações de rastreabilidade entre os artefatos de software, código fonte e especificações de requisitos. Links de rastreabilidade são recuperados por meio da técnica de recuperação de informação (IR), indexação semântica latente (LSI).

AdDoc [Dagenais e Robillard 2014] é um sistema open source de recomendação gera rastreabilidade entre código fonte e a documentação, no caso o manual das versões do software. Capaz de gerar automaticamente recomendações para elementos recém adicionados e que precisam ser documentados, além de identificar referências a elementos de código deletados ou em desuso que precisam ser corrigidos. Através dos chamados padrões de documentação é feita a inferência e a recomendação é realizada. Esse padrão seria um conjunto coerente de elementos de código referenciados pela documentação. A partir da análise de duas versões do manual, identifica-se métodos ou classes em desuso ou que foram retiradas da versão assim uma recomendação para correção pode ocorrer.

4.2.7 Código e Informativos

Listas de discussão, e-mails, constituem uma gama de informações valiosas para grupos de desenvolvedores na tomada de decisões.

LASCO (Linking e-mAils and Source Code) [Mazzeo et al. 2013] é uma ferramenta implementado como um plugin para Eclipse que busca gerar ligações entre e-mails e o código fonte. Inicialmente o email é dividido nas partes de que é composto: o cabeçalho, a mensagem atual (o corpo), e as frases de mensagens anteriores (citação). A relevância destas partes é ponderada por meio de um modelo probabilístico baseado em recuperação de texto. O modelo BM25F foi utilizado na implementação da solução. Basicamente, o sistema leva em consideração para gerar a rastreabilidade, medidas de

similaridade entre termos encontrados nos e-mails e o código fonte (por exemplo, classe).

4.3. Análise dos Resultados – Questão 2

Para responder a segunda questão de pesquisa, as técnicas e métodos foram analisados e categorizados.

Q2: Quais técnicas, métodos, abordagens as ferramentas aplicam/utilizam para gerenciar a rastreabilidade?

4.3.1 Baseadas em IR (Informational Retrieval)

Um crescente número de ferramentas se baseiam na técnica de IR para apoiar atividades de análise. ADAMS [De Lucia et al. 2006] ou FLAT3 [Sevage et al. 2010] são exemplos de ferramentas que geralmente complementam as técnicas típicas de análise de código estático com abordagens baseadas em IR com a suposição implícita de que as mesmas palavras são usadas sempre que se refere a um conceito particular.

A Informação léxica fornecida por programadores nos identificadores de código fonte é crucial para muitas ferramentas de análise e manutenção de software, mas a fim de ser devidamente explorado, identificadores devem ser processados, dividindo palavras concatenadas e expandindo abreviações. Em [Corazza et al. 2012] é apresentada uma nova técnica capaz de mapear um determinado identificador ao conjunto de palavras de um dicionário correspondente. São utilizados 4 fontes de palavras sendo uma delas os próprios comentários presentes no código fonte do sistema analisado. A técnica baseia-se num algoritmo de correspondência de padrões e é adequado para lidar com tanto a separação de um identificador em "tockens" quanto a expansão das abreviações que podem ser encontradas. A proposta foi implementada em um protótipo aplicado a sistemas de software de código aberto principalmente em C/C++. A técnica pode melhorar as ferramentas de rastreabilidade no código fonte baseadas em IR por, entre outras, tratar das questões de ambiguidade entre termos.

Outra proposta, apresentada em [Guerrouj, 2013], foca na normalização do vocabulário do código fonte para o apoio à compreensão e a qualidade do software. A abordagem proposta, explora informação de contexto para associar o vocabulário encontrado no código fonte com o encontrados nos demais artefatos de software. A normalização consiste de duas tarefas: a divisão e expansão dos identificadores de código-fonte.

Novas técnicas também são propostas como em [Capobianco et al. 2009] chamado B-spline, baseada em análise numérica para recuperar links de rastreabilidade entre documentação e código fonte. A abordagem proposta modela a informação contida em um artefato de software por curvas de interpolação particulares chamados B-splines, que plotam os termos mapeados e sua frequência no artefato analisado. Em seguida, o a similaridade entre artefatos é computada pelo calculo da distancia das curvas de interpolação correspondentes. Note-se que o uso de uma curva que interpola o conjunto de pontos que representam um artefato permite levar em conta as relações entre os termos para mitigar os problemas de sinonímia e polissemia.

Na tentativa de melhorar a recuperação de rastreabilidade baseado em IR uma nova abordagem é proposta em [Panichella et al. 2013]. Utiliza o feedback fornecido pelo engenheiro de software na classificação de ligações candidatas para regular o efeito do uso de informações estruturais. Especificamente, a abordagem considera apenas informações estruturais, quando os vínculos de rastreabilidade dos métodos de IR são verificados pelo engenheiro de software e classificados como ligações corretas.

4.3.2 Baseada em Ontologia

Algumas vezes produtos de software podem ser liberados sem uma documentação detalhada, ficando boa parte em linguagem natural sem padrão.

É proposta em [Yoshikawa et al. 2009] uma técnica baseada em ontologia para a recuperação de vínculos de rastreabilidade entre uma especificação de feature de um produto de software, em sentença de linguagem natural, e o código-fonte do produto. As relações entre as estruturas de código-fonte e domínios de problemas são importantes para detectar automaticamente fragmentos de código associados com as descrições funcionais escritas na forma de frases simples. Um passo da abordagem é modelar o conhecimento no domínio do problema em ontologias de domínio. A detecção de fragmentos de código que se correspondem com as sentenças é feita pelo uso de relações semânticas entre as ontologias, além das relações entre métodos invocados, e a similaridade entre um identificador no código e as palavras nas sentenças.

Outro trabalho nesse sentido é o de Zhang et al. (2008) focados reestabelecer as ligações de rastreabilidade entre código fonte existente e a documentação para apoiar a manutenção de software. Eles apresentam uma nova abordagem que resolve este problema através da criação de representações ontológicas formais tanto para a documentação quanto para os artefatos de código. Sua abordagem recupera links de rastreabilidade no nível semântico, utilizando informação semântica e estrutural encontrada em vários artefatos de software. As relações implícitas entre os artefatos são descobertas e mantidas através das inferências feitas pelas máquinas de inferência "reasoners" que suportam essas ontologias ligadas.

4.3.3 Baseada em Visualização

Uma abordagem implementada num estudo de caso como ferramenta para o Eclipse e utilizando a visualização "Treemap", é apresentada em [Bettenburg et al. 2012]. Baseado na ideia de que se uma parte específica de código é discutida entre os desenvolvedores então ela é importante ou especial. Utiliza detecção de clone como uma instância específica em uma abordagem baseada em pesquisa de código estabelecendo ligações entre fragmentos de código que são discutidos por desenvolvedores e o código fonte real de um projeto. A abordagem foi comparada às técnicas clássicas em análise do log de mudanças e a recuperação de informação.

Em [Chen et al. 2012] foi implementado uma abordagem objetivando fornecer visualização de rastreabilidade de modo eficiente. Pela combinação de duas representações gráficas, treemap e hierarchical tree, exibem a estrutura geral de links de rastreabilidade e fornecem uma visão detalhada de cada link de modo altamente escalável e interativo. Um protótipo da abordagem foi desenvolvido. Este protótipo é perfeitamente incorporado dentro da IDE Eclipse. Ele extrai automaticamente as relações entre seções em documentos e classes no código-fonte e visualiza estes links recuperados. Os usuários podem usar a funcionalidade fornecida não só no Eclipse IDE, mas também como uma ferramenta stand-alone.

5. Ameaças à validade

O presente trabalho possui limitações quanto a sua validade, pois embora tenha sido definido um procedimento sistemático para a realização das etapas de inclusão e ou

exclusão, esta deve ser feita por pares e apenas em comum acordo deve haver exclusão ou inclusão. Apenas o autor foi responsável pela execução e desse procedimento, portanto, existe uma possibilidade de erro de interpretação, sobretudo nas etapas preliminares de inclusão/exclusão.

As questões de investigação definidas neste estudo não podem fornecer cobertura completa da área de rastreabilidade de software. As perguntas foram definidas do ponto de vista de quais são as métricas, medidas, critérios e fatores estão sendo utilizadas para calcular o grau de reputação de uma entidade no contexto de manutenção e evolução de software, do ponto de vista que se acreditou ser importante para a pesquisa.

Mesmo com a tentativa de ser obter um *string* de busca abrangente e que atendesse as necessidades da pesquisa não foi possível com ela abordar todos os estudos possíveis na área de rastreabilidade de software. Utilizando-se das técnicas de *snow-balling* [Budgen et al. 2008] pode-se recuperar muitos artigos importantes e interessantes à pesquisa que não foram abrangidos na *string* de busca.

6. Considerações Finais

O objetivo deste trabalho foi realizar uma revisão sistemática que ajuda nas pesquisas a respeito de rastreabilidade de software e o que está sendo utilizado no contexto de manutenção ou evolução de software, identificando assim literaturas relevantes para este contexto bem como autores, ferramentas, abordagens e técnicas e métodos relacionados a essas ferramentas, apresentadas nos resultados da pesquisa.

Foi observado que a rastreabilidade é mantida entre diferentes tipos de artefatos gerados ao longo do desenvolvimento do software, não ficando restrito aos requisitos. Vale destacar a preocupação entre o código fonte e os casos de teste com o objetivo de facilitar a estruturação dos testes a serem conduzidos e a escassez de abordagens que relacionem o código fonte com outros níveis de abstração do próprio código fonte.

Foi demonstrado um gráfico de publicações relevantes por ano que remete à importância de pesquisas na área, e como vem crescendo ao longo dos anos.

Foi respondido ao longo do estudo uma importante questão, ao se descobrir as diferentes técnicas e métodos que estão sendo utilizadas pelas ferramentas na geração e manutenção dos links de rastreabilidade, além de outras que poderiam servir de oportunidades futuras de pesquisas.

Esses resultados, contem fontes de informações que podem ser utilizadas em trabalhos futuros, realizados na área de rastreabilidade nas atividades de evolução de software no processo de manutenção.

Referências

Steinmacher, I. Chaves, A.P. Gerosa, M.A. (2012), Awareness Support in Distributed Software Development: A Systematic Review and Mapping of the Literature. Journal of Computer Supported Cooperative Work.

Kitchenham, Barbara. (2004), Procedures for performing systematic reviews. Technical Report, TR/SE-0401, Department of Computer Science, Keele University, Keele,UK.

Petticrew, M., Roberts, H. (2006) Systematic Reviews in the Social Sciences: A Practical Guide. Blackwell Publishing, Malden.

D. Budgen, M. Turner, P. Brereton, B. Kitchenham (2008) Using mapping studies in software engineering, in: PPIG'08: 20th Annual Meeting of the Psychology of Programming Interest Group, Lancaster University.

Savage, T., Revelle, M., & Poshyvanyk, D. (2010, May). FLAT 3: feature location and textual tracing tool. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2 (pp. 255-258). ACM.

De Lucia, A., Fasano, F., Oliveto, R., & Tortora, G. (2006, March). ADAMS: advanced artefact management system. In Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on (pp. 2-pp). IEEE.

Corazza, A., Di Martino, S., & Maggio, V. (2012, September). LINSEN: An efficient approach to split identifiers and expand abbreviations. In Software Maintenance (ICSM), 2012 28th IEEE International Conference on (pp. 233-242). IEEE.

Guerrouj, L. (2013, May). Normalizing source code vocabulary to support program comprehension and software quality. In Proceedings of the 2013 International Conference on Software Engineering (pp. 1385-1388). IEEE Press.

Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A., & Panichella, S. (2009, October). Traceability recovery using numerical analysis. In Reverse Engineering, 2009. WCRE'09. 16th Working Conference on (pp. 195-204). IEEE.

Panichella, A., McMillan, C., Moritz, E., Palmieri, D., Oliveto, R., Poshyvanyk, D., & De Lucia, A. (2013, March). When and how using structural information to improve ir-based traceability recovery. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on(pp. 199-208). IEEE.

Yoshikawa, T., Hayashi, S., & Saeki, M. (2009, September). Recovering traceability links between a simple natural language sentence and source code using domain ontologies. In Software Maintenance, 2009. ICSM 2009. IEEE International Conference on (pp. 551-554). IEEE.

Zhang, Y., Witte, R., Rilling, J., & Haarslev, V. (2008). Ontological approach for the semantic recovery of traceability links between software artefacts.IET software, 2(3), 185-203.

Bettenburg, N., Thomas, S. W., & Hassan, A. E. (2012, March). Using fuzzy code search to link code fragments in discussions to source code. InSoftware Maintenance and Reengineering (CSMR), 2012 16th European Conference on (pp. 319-328). IEEE.

Chen, X., Hosking, J., & Grundy, J. (2012, September). Visualizing traceability links between source code and documentation. In Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on (pp. 119-126). IEEE.

Ali, N., Guéhéneuc, Y. G., & Antoniol, G. (2011, October). Requirements traceability for object oriented systems by partitioning source code. InReverse Engineering (WCRE), 2011 18th Working Conference on (pp. 45-54). IEEE.

Mazzeo, L., Tolve, A., Branda, R., & Scanniello, G. (2013, March). Linking E-Mails and Source Code with LASCO. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on (pp. 421-424). IEEE.

Dagenais, B., & Robillard, M. P. (2014). Using Traceability Links to Recommend Adaptive Changes for Documentation Evolution. Software Engineering, IEEE Transactions on, 40(11), 1126-1146.

Islam, M. M., Marchetto, A., Susi, A., Kessler, F. B., & Scanniello, G. (2012, September). MOTCP: A tool for the prioritization of test cases based on a sorting genetic algorithm and Latent Semantic Indexing. In Software Maintenance (ICSM), 2012 28th IEEE International Conference on (pp. 654-657). IEEE.

Lehnert, S., Farooq, Q. U. A., & Riebisch, M. (2013, March). Rule-based impact analysis for heterogeneous software artifacts. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on(pp. 209-218). IEEE.

Diaz, D., Bavota, G., Marcus, A., Oliveto, R., Takahashi, S., & De Lucia, A. (2013, May). Using code ownership to improve ir-based traceability link recovery. In Program Comprehension (ICPC), 2013 IEEE 21st International Conference on (pp. 123-132). IEEE.

Lago, P., Niemelä, E., & Van Vliet, H. (2004, March). Tool support for traceable product evolution. In Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on (pp. 261-269). IEEE.

Kagdi, H., Maletic, J., & Sharif, B. (2007, June). Mining software repositories for traceability links. In Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on (pp. 145-154). IEEE.

De Lucia, A., Oliveto, R., & Tortora, G. (2008, May). Adams re-trace: traceability link recovery via latent semantic indexing. In Proceedings of the 30th international conference on Software engineering (pp. 839-842). ACM.

Antoniol, G., Canfora, G., De Lucia, A., & Merlo, E. (1999, October). Recovering code to documentation links in OO systems. In Reverse Engineering, 1999. Proceedings. Sixth Working Conference on (pp. 136-144). IEEE.

Antoniol, G., Canfora, G., Casazza, G., & Merlo, E. (2000). Tracing object-oriented code into functional requirements. In Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on (pp. 79-86). IEEE.

Ni, D. C., Martinez, J., Eccles, J., Thomas, D., & Lai, P. K. M. (1994, December). Process automation with enumeration and traceability tools. InIndustrial Technology, 1994., Proceedings of the IEEE International Conference on (pp. 361-365). IEEE.

Gardner, F. K. (1994, June). RayTracer: traceability for software engineering. In Assessment of Quality Software Development Tools, 1994, Proceedings., Third Symposium on (pp. 224-232). IEEE.

Sureka, A., Lal, S., & Agarwal, L. (2011, December). Applying fellegi-sunter (fs) model for traceability link recovery between bug databases and version archives.

In Software Engineering Conference (APSEC), 2011 18th Asia Pacific (pp. 146-153). IEEE.

Qusef, A., Oliveto, R., & De Lucia, A. (2010, September). Recovering traceability links between unit tests and classes under test: An improved method. In Software Maintenance (ICSM), 2010 IEEE International Conference on (pp. 1-10). IEEE.

Mirakhorli, M., Shin, Y., Cleland-Huang, J., & Cinar, M. (2012, June). A tactic-centric approach for automating traceability of quality concerns. InProceedings of the 34th International Conference on Software Engineering(pp. 639-649). IEEE Press.

Hammad, M., Collard, M. L., & Maletic, J. I. (2011). Automatically identifying changes that impact code-to-design traceability during evolution.Software Quality Journal, 19(1), 35-64.

Guéhéneuc, Y. G., & Antoniol, G. (2008). Demima: A multilayered approach for design pattern identification. Software Engineering, IEEE Transactions on,34(5), 667-684.

Fiutem, R., & Antoniol, G. (1998, November). Identifying design-code inconsistencies in object-oriented software: a case study. In Software Maintenance, 1998. Proceedings., International Conference on (pp. 94-102). IEEE.

Antoniol, G., Canfora, G., & De Lucia, A. (1999). Maintaining traceability during object-oriented software evolution: a case study. In Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on(pp. 211-219). IEEE.

Antoniol, G., Tonella, P., & Fiutem, R. (1999). Evolving object oriented design to improve code traceability. In Program Comprehension, 1999. Proceedings. Seventh International Workshop on (pp. 151-160). IEEE.

Ali, N., Gueneuc, Y. G., & Antoniol, G. (2013). Trustrace: Mining software repositories to improve the accuracy of requirement traceability links.Software Engineering, IEEE Transactions on, 39(5), 725-741.

Egyed, A., Binder, G., & Grunbacher, P. (2007, May). STRADA: A tool for scenario-based feature-to-code trace detection and analysis. In Companion to the proceedings of the 29th International Conference on Software Engineering(pp. 41-42). IEEE Computer Society.

Penta, M. D., Gradara, S., & Antoniol, G. (2002). Traceability recovery in RAD software systems. In Program Comprehension, 2002. Proceedings. 10th International Workshop on (pp. 207-216). IEEE.

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., & Harshman, R. A. (1990). Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41(6), 391-407.

De Lucia, A., Fasano, F., Oliveto, R., & Tortora, G. (2006, March). ADAMS: ADvanced Artefact Management System. In Proceedings of the Conference on Software Maintenance and Reengineering (pp. 349-350). IEEE Computer Society.