1 Introdução

A fase de manutenção de um software pode ser considerada a que consome mais recursos, seja considerando custo ou tempo gasto em sua execução (WITTE, ZHANG, RILLING, 2007) (HYLAND-WOOD, 2008). Em termos gerais, a manutenção engloba tanto as alterações para a correção de defeitos, bem como melhorias e evoluções decorrentes de necessidades de adequação ao ambiente ou pela alteração de requisitos (HYLAND-WOOD, 2008). Estas modificações aumentam o tamanho e a complexidade do software desenvolvido, tornando sua manutenção em algo complexo e trabalhoso para os envolvidos no projeto (HYLAND-WOOD, 2008). Outro fato que reitera a importância desta fase é a quantidade de linhas de código que são mantidas nas organizações com o intuito de suportar seus processos de negócio, estimava-se que esse número era próximo de um trilhão de linhas no ano de 2001 (LAMMEL e VERHOEF, 2001). Observa-se ainda que este número tem aumentado no decorrer do tempo devido à crescente dependência de software em todo o mundo (HYLAND-WOOD, 2008) (SOMMERVILLE, 2004).

De acordo com (LETHBRIDGE e ANQUETIL, 1997), a pesquisa e a navegação de código-fonte são as duas principais atividades que os desenvolvedores realizam durante a fase de manutenção. Estas atividades são necessárias para compreender a estrutura geral do sistema, seja através da identificação de seus principais componentes ou da implementação associada com os mesmos (RIVA, 2000). Além do código-fonte, outras fontes habitualmente consultadas incluem gerenciadores de demandas e a documentação do software. Estas consultas têm por objetivo estabelecer a relação semântica existente entre defeitos corrigidos, requisitos e documentos de design com o código implementado no software (WITTE, ZHANG, RILLING, 2007). Grande parte das falhas que ocorrem durante esta etapa do ciclo de vida do software pode ser compreendida como a ausência da consistência entre estes diversos dados consultados.

Isto pode ser explicado em parte pelo fato de que cada tipo de artefato que compõe um sistema, seja documentação, código-fonte ou pedidos de mudanças, esteja armazenado em repositórios especializados (TAPPOLET, 2007) (TAPPOLET, 2011). Estes, apesar de armazenarem grande parte dos artefatos produzidos durante o ciclo de vida de um software, normalmente funcionam como sistemas isolados que possuem sua própria representação interna e que carecem de mecanismos para a análise detalhada de suas informações (LI e ZHANG, 2011). Isto inviabiliza a resposta a perguntas normalmente feitas durante a etapa de manutenção e que abrangem mais de um domínio de ferramenta. Um típico exemplo seria descobrir quais as modificações de código fonte foram necessárias para implementar um determinado requisito, ou quais as alterações foram feitas para corrigir um determinado defeito. Além de impactar a criação destas consultas elaboradas, a heterogeneidade destas ferramentas é um dos fatores que dificulta as pesquisas em torno da mineração de dados de repositórios de software. A Mineração de Repositórios de Software é o ramo de pesquisa que analisa as informações disponíveis em repositórios de software, como os repositórios exemplificados anteriormente. Estas análises têm contribuído para a melhoria da qualidade dos sistemas de software desenvolvidos, pois é possível desde a detecção de anomalias no projeto à predição de defeitos (TAPPOLET, 2011).

A comunidade de mineração de dados tem resolvido o problema da integração através do espelhamento dos artefatos de várias fontes em uma base de dados relacional centralizada (WÜRSCH, *et al.*, 2010). Entretanto, esta solução impõe um esquema de dados universal para todas as ferramentas que estão contribuindo com dados para análise, além de exigir a transformação de dados que não são relacionais em dados relacionais. Por exemplo, a representação da árvore de sintaxe abstrata (*Abstract Syntax Tree* ou AST)[1] do código-fonte implica na transformação de um grafo em uma representação que possa ser armazenada em uma tabela relacional (TAPPOLET, 2011). Estas características tornam o repositório centralizado em algo não extensível e inflexível, o mesmo problema encontrado nas próprias ferramentas utilizadas como fonte de dados. Como consequência, a integração de ferramentas de extração/análise criadas por diferentes grupos de pesquisa se torna uma tarefa de difícil execução, pois cada

[1] http://en.wikipedia.org/wiki/Abstract_syntax_tree

grupo realiza a análise dos mesmos artefatos, mas utilizando diferentes representações e perspectivas. Isto resulta em vários conjuntos de dados relacionados, mas que na prática são distintos (KEIVANLOO, *et al.*, 2011). Para acelerar o progresso da pesquisa e evitar retrabalho, o compartilhamento e a integração destes conjuntos de dados se tornou uma necessidade neste ramo de pesquisa (KEIVANLOO, *et al.*, 2012).

Diversos formatos genéricos de troca foram propostos na literatura com o intuito de resolver o problema compartilhamento de informações. Uma destas tentativas foi o GXL (HOLT, WINTER, SCHURR, 2000), um esforço para realizar o compartilhamento de grafos de software entre ferramentas de análise. Outro formato é o CDIF (*CASE Data Interchange Format*), um padrão de troca entre ferramentas CASE (LI, 2011). Mais tarde surgiu o formato XMI (*XML Metadata Interchange*), um formato de troca proposto pela OMG baseado em XML que é capaz de expressar diferentes modelos. Apesar da sofisticação de algum destes formatos, eles não são facilmente extensíveis e somente resolvem o problema sintático, ou seja, o formato de troca dos dados (HEBELER, *et al.*, 2011). A semântica das informações não é exportada, como por exemplo, o fato de que cada dataset utilizar identificadores aleatórios, dependentes do contexto e não padronizados (KEIVANLOO, *et al.*, 2012) (KEIVANLOO, *et al.*, 2011), tornando impossível identificar uma mesma entidade em diferentes conjuntos de dados, transformando a integração de informações em algo manual.

Recentemente, alguns trabalhos têm demonstrado que a utilização de tecnologias da Web Semântica, como representações em RDF (*Resource Description Framework*), ontologias baseadas em OWL (*Web Ontology Language*) e consultas SPARQL (*SPARQL Protocol and RDF Query Language*) podem facilitar a representação e a integração das informações extraídas de repositórios de software. Isto se deve principalmente à natureza aberta e extensível do meta-modelo RDF e o número crescente de ferramentas que suportam estas tecnologias (KEIVANLOO, *et al.*, 2011). Estas características foram fundamentais para a utilização da Web Semântica em diferentes domínios como uma plataforma de integração e de gerenciamento de conhecimento, incluindo desde o domínio de saúde ao domínio de *Life Science*. Isto se deve, em parte, ao fato de que modelos descritos em RDF permitem definir tanto o esquema de dados e como as instâncias deste esquema em um mesmo meta-modelo. Isto é

uma abordagem diferente de modelos como o SQL, onde é necessário existir de antemão o esquema para que a entrada de dados possa ser realizada. Estas características permitem estender o modelo de dados enquanto mantém-se a compatibilidade com as aplicações existentes que utilizam uma versão diferente do modelo (TAPPOLET, KIEFER, BERNSTEIN, 2007). Além do RDF, a Web Semântica também requer a criação de URIs (Universal Resource Identifier)[2] para referenciar unicamente um elemento do modelo, possibilitando que outros elementos possam referenciá-lo de forma simples e não ambígua. E, através do uso de ontologias e vocabulários, é possível formalizar os tipos de elementos existentes no modelo, bem como suas propriedades e os tipos de relacionamentos possíveis (KEIVANLOO, *et al.*, 2011). Normalmente, a definição de uma ontologia/vocabulário na Web Semântica é feita através da utilização de OWL e de RDFS (*RDF Schema*), ambos derivados do meta-modelo RDF. Embora seja possível realizar as mesmas representações utilizando outros mecanismos, a utilização destas duas tecnologias possibilita uma expressividade não existente em outras técnicas de modelagem (HYLAND-WOOD, 2008), como a verificação da consistência lógica das informações ou a derivação de novos fatos a partir de fatos existentes (TAPPOLET, 2010). Logo, a Web Semântica e suas ferramentas representam uma plataforma robusta para a integração de artefatos de software em um repositório semântico.

1.1. Limitações das Abordagens Atuais

Apesar dos avanços na área de Mineração de Repositórios de Software, encontramos alguns pontos passíveis de melhorias relacionados a:

1. **Tratamento de Grande Volume de Informações:**

Um dos grandes desafios relacionados à mineração de repositórios de software diz respeito a como lidar com o grande volume de informações geradas por estas ferramentas de forma a apresentar respostas em tempo aceitável. Por exemplo, o repositório de artefatos *Maven Central*[3] armazena por volta de 180

[2] http://en.wikipedia.org/wiki/Uniform_resource_identifier
[3] http://search.maven.org/

Gigabytes de artefatos que fornecem informações sobre bibliotecas de software, suas licenças e dependências.

2. Expressividade das Informações Relacionadas ao Código-Fonte:

Outra questão são as limitações relacionadas à expressividade de representação das informações associadas ao código-fonte de um dado projeto, como o suporte a diferentes tipos de linguagens de programação, a representação das diversas versões de um elemento de código-fonte, bem como de seus relacionamentos e dependências. Um exemplo é a relação intrínseca entre as várias revisões no controle de versão de um arquivo e as mudanças na AST do código-fonte contido nela. Normalmente, as alterações são tratadas a nível textual e não como mudanças estruturais. Uma representação mais precisa pode identificar que tipo de mudança no código um determinado desenvolvedor realizou e com isso conseguir determinar para quem atribuir uma revisão de código, ou seja, quem tem conhecimento para realizar esta tarefa.

3. Extensibilidade da Solução para Novos Tipos de Repositórios e Versões:

Normalmente estas soluções estão atreladas as ferramentas/repositórios de software inicialmente considerados pelos grupos de pesquisa. Deste modo não é algo trivial incluir novos tipos e novas versões de ferramentas às abordagens de extração. Por este motivo, a arquitetura de uma abordagem de extração deve ser organizada de forma a permitir que novas versões e novas ferramentas possam ser agregadas sem o comprometimento da estrutura geral do processo de extração.

1.2. Objetivo

Esta dissertação visa estender as abordagens atuais de mineração para endereçar as limitações apresentadas anteriormente através do uso de ferramentas e tecnologias da Web Semântica. De modo a testar esta nova abordagem, escolhemos dar suporte a um conjunto de perguntas relacionadas ao código-fonte dos projetos que carecem de repostas pelas abordagens existentes devido à falta de

dados e que também são afetadas diretamente pelas limitações apresentadas. Estas questões foram discutidas em diversos estudos que tinham por objetivo determinar quais as informações que os desenvolvedores e os gerentes consideram mais relevantes na execução de suas tarefas durante o desenvolvimento de um projeto de software. Abaixo descrevemos quais as novas informações consideradas em nossa solução em conjunto com exemplos de perguntas que são dependentes destes dados:

Informações presentes em ferramentas de gerenciamento de dependência: Um tipo de repositório normalmente não considerado pelas abordagens de mineração são as informações contidas nos arquivos utilizados por ferramentas de gerenciamento de dependência, como o Apache Maven[4] e o Apache Ivy[5]. Estas ferramentas são bastante utilizadas na comunidade de código-aberto e uma de suas principais características é manter explicitamente a relação de dependências externas de um projeto, indicando o nome do artefato e sua versão. Além disso, o projeto pode utilizar estes mesmos arquivos para declarar os artefatos que são gerados por ele e as licenças sob as quais eles são disponibilizados. Resumindo, a totalidade destes arquivos forma um grande grafo de dependências entre os projetos. Entretanto, as abordagens atuais só levam em consideração a dependência em nível de instruções da linguagem de programação em questão, como é o caso do elemento "*import*" utilizado na linguagem de programação Java. Esta instrução indica para o compilador que existe uma dependência para com uma determinada classe ou conjunto de classes de um pacto, mas não considera a versão da dependência. Uma pergunta normalmente feita por desenvolvedores e que não é possível responder sem esta informação, é descobrir quais são as pessoas que já trabalharam com uma determinada biblioteca. Ou então, verificar (semi) automaticamente se licença da biblioteca é compatível com a licença do projeto.

A utilização de dependências do projeto na criação da AST: Normalmente as abordagens que extraem informações do controle de versão não utilizam as dependências do projeto para construir a AST de cada arquivo de código-fonte,

[4] http://maven.apache.org/
[5] http://ant.apache.org/ivy/

pois, isto torna o processo de extração mais demorado e complexo, sendo necessário lidar com dependências erradas ou incompletas. Além disso, o processo de realizar o *binding*[6] entre os elementos declarados no código-fonte e os elementos presentes nas dependências é custoso por si só. Entretanto, estas informações permitem responder, por exemplo, como as dependências de um projeto interagem com ele. Mais especificamente, é possível listar quais os métodos/classes, que, se modificados em uma nova versão da biblioteca, vão afetar diretamente o código do projeto. Além disso, uma AST mais completa permite distinguir entre dois métodos quando técnicas como *method overloading*[7] ou *method overriding*[8] são utilizadas.

Dados de ferramentas de integração contínua: A Integração Contínua é uma prática amplamente adotada em projetos da comunidade de código-aberto e promove a integração frequente do código e construções automatizadas através de ferramentas de integração contínua. Esta prática ajuda na identificação precoce de problemas e consequentemente na melhoria do processo de desenvolvimento como um todo. Dentre as informações disponibilizadas podem-se destacar: os resultados das construções, os resultados dos testes executados e as entregas no controle de versão que fizeram parte das construções. Tais informações são imprescindíveis para a reposta de perguntas que envolvam as construções de um projeto, como a descoberta do causador frequente da quebra da construção ou então quem foi o responsável pela quebra dos testes automatizados.

1.3. Organização da Dissertação

Esta dissertação está organizada da seguinte forma:

- **Capítulo 2 – Fundamentos:** Neste capítulo são apresentados os principais conceitos que suportaram o desenvolvimento da nova abordagem de extração.
- **Capítulo 3 – Ontologias e Vocabulários:** Neste capítulo apresentaremos as ontologias/vocabulários que foram utilizadas, bem como as ontologias que foram desenvolvidas para este trabalho.

[6] http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/index.html#sec-bindings
[7] http://en.wikipedia.org/wiki/Function_overloading

- **Capítulo 4 – A Plataforma de Extração:** Neste capítulo discutimos a plataforma de extração capaz de minerar as informações que não são consideradas em outras abordagens. Abordaremos seus casos de uso, sua arquitetura, os diversos módulos de extração e seus relacionamentos com os vocabulários e ontologias apresentados no capítulo anterior.
- **Capítulo 5 – Validação e Estudo de Caso:** Neste capítulo apresentaremos como as questões que desenvolvedores e gerentes têm podem ser respondidas através das informações extraídas por esta abordagem. Além disso, apresentaremos a aplicação destas perguntas em um projeto de software.
- **Capítulo 6 – Conclusão e Trabalhos Futuros:** Neste capítulo são apresentados os trabalhos relacionados, contribuições e trabalhos futuros.

[8] http://en.wikipedia.org/wiki/Method_overriding#Java