5 Validação e Estudo de Caso

5.1. Introdução

Diariamente, um desenvolvedor necessita lidar com um grande número de questões acerca dos sistemas que ele mantém ou desenvolve. Algumas destas perguntas podem ser facilmente respondidas pelas próprias ferramentas utilizadas no desenvolvimento, como por exemplo, onde um determinado método é chamado (**Figura 37**) ou qual são os defeitos abertos sem responsável (**Figura 38**). Porém, como mencionamos anteriormente, algumas perguntas podem envolver o escopo de mais de uma de ferramenta, um exemplo é descobrir quais foram as modificações no código que implementam uma determinada tarefa ou corrigem um defeito.

Figura 37 Visualização das Chamadas de um Método na IDE Eclipse

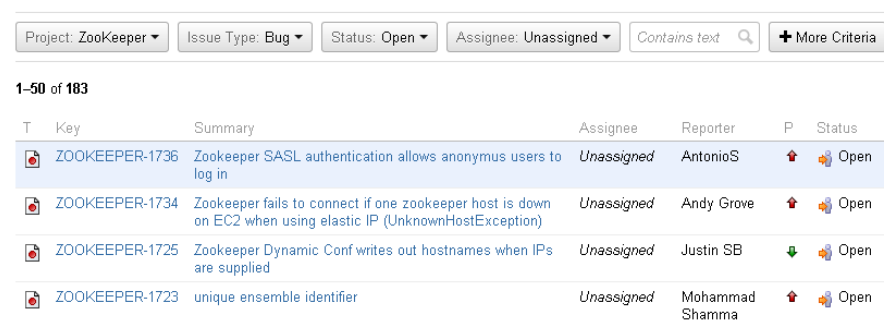

Figura 38 Visualização de todos os defeitos abertos e sem um responsável na ferramenta Atlassian Jira

Neste contexto, diversas entrevistas e experimentos foram conduzidos em estudos na literatura (FRITZ e MURPHY, 2010) (LATOZA e MYERS, 2010) (ALWIS e MURPHY, 2008) para identificar quais destas questões são mais recorrentes entre desenvolvedores de software. Estas questões foram apontadas, pelos entrevistados, como perguntas críticas que muitas vezes consomem tempo considerável para serem respondidas, por conta da falta de apoio das ferramentas existentes (FRITZ e MURPHY, 2010). Apesar da dificuldade de respondê-las, as respostas podem ser utilizadas como base na tomada de determinadas decisões, como por exemplo, quem deve ser alocado para executar uma correção ou qual modificação deve ser postergada devido ao impacto que ela pode causar em outros sistemas. Ou seja, uma melhor compreensão dos sistemas de software modificados é essencial para que suas manutenções possam ser executadas de forma minimizar possíveis problemas.

Partindo desta lacuna, demonstraremos quais perguntas e de que modo a abordagem apresentada consegue respondê-las, bem como, quais são as perguntas que somente este trabalho é capaz de responder ou que responde de forma mais rica pelo conjunto de dados utilizado. Além disso, apresentaremos exemplos de novas questões que esta abordagem consegue responder por conta da sua capacidade de expressividade de informações. Depois, um estudo de caso sobre a utilização da plataforma em repositórios de software reais será apresentado. Neste estudo apresentamos o resultado da execução de parte das consultas apresentadas anteriormente.

5.2. Perguntas Frequentes

Na tabela abaixo, apresentamos um subconjunto das perguntas apresentadas nos estudos citados anteriormente e que estão relacionadas diretamente ao código-fonte de um projeto. Indicamos quais delas a abordagem deste trabalho consegue responder completamente ou parcialmente através de uma consulta SPARQL nos dados extraídos. Para comparação, apresentamos a mesma avaliação para duas outras abordagens de mineração, Evoont (TAPPOLET, 2010) e SEC (HYLAND-WOOD, 2008). Um detalhamento destes e de outros trabalhos serão apresentados no próximo capítulo.

Perguntas	Capacidade de Resposta		
	Abordagem Atual	**Evoont**	**SEC**
Quem alterou este código, categorizado por pessoa?	Sim	Parcial[1]	Não[2]
Para quem atribuir uma revisão de código? Quem tem conhecimento para realizar a revisão de código?	Sim[3]	Parcial[1]	Não[2]
Quem alterou classes que eu modifico?	Sim	Parcial[1]	Não[2]
Quem está utilizando esta API [que eu vou modificar]?	Sim	Sim	Sim
Quem são os criadores desta API [que eu vou modificar]?	Sim	Parcial[1]	Não[2]
Quem é o dono deste pedaço de código? Quem modificou por último?	Sim	Parcial[1]	Não[2]
Com quem falar quando se tem que trabalhar com pacotes que nunca se trabalhou?	Sim	Parcial[1]	Não[2]
Qual é a classe mais popular? [Qual classes tem sido a mais modificada]	Sim	Parcial[1]	Não[2]
Qual outro código que eu trabalhei utiliza esta função utilitária?	Sim	Parcial[1]	Não[2]
Qual código recentemente modificado está relacionado a mim?	Sim	Parcial[1]	Não[2]

Perguntas	Capacidade de Resposta		
	Abordagem Atual	Evoont	SEC
Quem causou a quebra da construção? (Quem é responsável pela quebra dos testes?)	Sim	Não[4]	Não[4]
Quem é dono deste caso de teste? (Quem resolveu a última tarefa que corrige o caso de teste?)	Sim	Não[4]	Não[4]
O que mais foi modificado quando este código foi modificado ou inserido?	Sim	Parcial[1]	Não[2]
Como este código interage com bibliotecas?	Sim	Não[5]	Não[5]

Tabela 13 Perguntas Frequentes dos Desenvolvedores

[1] Evoont não considera todos os conjuntos de alterações que um determinado arquivo de código-fonte tem.

[2] SEC não considera informações das ferramentas de controle de versão.

[3] Nesta versão, a plataforma não considera fórmulas para calculo de expertise.

[4] É necessário o uso de informações da execução de construções automatizadas

[5] É necessário o uso de informações das dependências do projeto.

A seguir, apresentamos de que modo as informações extraídas foram utilizadas para a criação das consultas SPARQL que respondem as perguntas apresentadas. De modo a facilitar a leitura, a definição de cada consulta SPARQL pode ser encontrada no Apêndice E.

1. **Quem alterou este código, categorizado por pessoa?**

   Esta pergunta foi traduzida como uma consulta que busca todos os donos de conjuntos de alterações que causaram impacto em uma ou mais entidades de código-fonte que fazem parte do código em questão.

2. **A quem atribuir uma revisão de código? Quem tem conhecimento para realizar a revisão de código?**

   A consulta que responde esta pergunta foi produzida de forma semelhante à reposta da pergunta anterior. A diferença está na ordenação das pessoas, que neste caso utiliza o total de impactos e a data em que ocorreram.

3. **Quem alterou classes que eu modifico?**

   Esta consulta consiste na pesquisa de todos desenvolvedores que criaram impactos em entidades do tipo "Classe" que o usuário corrente também tenha causado impacto.

4. **Quem está utilizando esta API [que eu vou modificar]?**

   Esta consulta recebe como entrada um conjunto de entidades que formam uma determinada API e pesquisa todos os relacionamentos entre estes elementos e os outros que fazem uso da API.

5. **Quem são os criadores desta API [que eu vou modificar]?**

   Esta consulta recebe como entra um conjunto de entidades que formam uma determinada API e pesquisa todos os impactos de adição sofridos por estes elementos. Após isto, agrupa os impactos e ordena de forma decrescente pelo número de impactos causados por pessoa.

6. Quem é o dono deste pedaço de código? Quem modificou por último?

Esta consulta recebe como entrada uma entidade de código-fonte e retorna a última pessoa que causou um impacto que modificou esta entidade.

7. Com quem falar quando se tem que trabalhar com pacotes que nunca se trabalhou?

Esta consulta recebe como entrada uma determinada versão de uma biblioteca e retorna os desenvolvedores que alteraram código adicionando referências para entidades presentes na biblioteca.

8. Qual é a classe mais popular? [Qual classes tem sido a mais modificada]?

Esta consulta pesquisa qual a classe que sofreu mais impactos em sua estrutura.

9. Qual outro código que eu trabalhei utiliza esta função utilitária?

Esta consulta pesquisa todos os métodos modificados pelo usuário e que invocam a função utilitária em questão.

10. Qual código recentemente modificado que está relacionado a mim?

Esta consulta pesquisa todas as entidades de código-fonte modificadas pelo usuário atual e que foram modificadas nos últimos 10 conjuntos de alterações por outras pessoas.

11. Quem causou a quebra da construção? (Quem é responsável pela quebra dos testes?)

Esta consulta pesquisa todos os desenvolvedores que foram os criadores dos conjuntos de alterações, incluídos na construção quebrada, que modificaram entidades de código-fonte utilizadas nos métodos dos casos de testes que falharam.

12. Quem é dono deste caso de teste? (Quem resolveu a última tarefa que corrige o caso de teste?)

Esta consulta pesquisa o desenvolvedor que resolve a última tarefa associada com um conjunto de alterações que modificou um caso de teste e fez com que este saísse de seu estado de erro para sucesso.

13. O que mais foi modificado quando este código foi modificado ou inserido?

Esta consulta busca quais as entidades que tiveram impactos no mesmo conjunto de alterações que as entidades em questão.

14. Como este código interage com bibliotecas?

Esta consulta busca quais as entidades têm relacionamentos com entidades que fazem parte de alguma biblioteca.

5.3. Exemplos de Novas Perguntas

De forma a demonstrar a expressividade dos dados extraídos pela plataforma, apresentamos três novas perguntas que podem ser elaboradas a partir dos dados extraídos:

1. **Quais entidades no projeto podem ser diretamente afetadas por uma mudança em uma versão de uma biblioteca utilizada pelo projeto?**

Esta consulta recebe como parâmetro a versão atual de uma determinada biblioteca utilizada no projeto e a versão para a qual se deseja atualizar. A partir disto, pesquisa todas as entidades de código-fonte do projeto que referenciam entidades que estão diferentes entre as duas versões da biblioteca. Abaixo a consulta SPARQL correspondente.

```
PREFIX sm:<http://semancti/>

SELECT DISTINCT ?element ?libraryElement ?impactType
WHERE {
?changeSet a sm:ChangeSet.
?changeSet   sm:elements ?element.
?element ^sm:isRelationshipOf ?relationship.
?relationship a sm:Relationship.
?relationship rdf:value ?libraryElement.
?libraryElement (^sm:impactOfAfter|^sm:impactOfBefore) ?impact.
?impact sm:contextTo $libraryVersion2.
?impact sm:contextFrom  $libraryVersion1.
?impact sm:impactType ?impactType
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
  }
```

Quadro 1 Consulta SPARQL que retornar entidades afetadas pela mudança de versão de uma biblioteca do projeto

2. **Quais defeitos corrigidos na nova versão da biblioteca afetam diretamente o projeto em questão?**

Esta consulta recebe como parâmetro a versão atual de uma biblioteca utilizada no projeto e a versão para qual se deseja atualizar e pesquisa todas as entidades que invocam métodos que foram diretamente afetados por defeitos que foram corrigidos na nova versão.

```
PREFIX sm:<http://semancti/>

SELECT ?bug ?element WHERE {
?changeSet a sm:ChangeSet.
?changeSet   sm:elements ?element.
?element ^sm:isRelationshipOf ?relationship.
?relationship a sm:Relationship.
?relationship rdf:value ?libraryElement.
?libraryElement  (^sm:impactOfAfter|^sm:impactOfBefore)  ?impact.
?impact sm:contextTo $libraryVersion2.
$libraryVersion2 ^sm:resolvedIn ?bug.
?bug    sm:issueType    'Bug'.        ?impact    sm:contextFrom
$libraryVersion1. ?impact sm:impactType ?impactType
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
}
```

Quadro 2 Consulta SPARQL que retorna os elementos afetados por defeitos corrigidos em uma nova versão de uma biblioteca utilizada no projeto

3. Quais defeitos que podem ter sido causados devido à atualização de uma biblioteca?

Esta consulta recebe como parâmetro a nova versão da biblioteca e pesquisa todos os defeitos que ocorreram em entidades afetadas diretamente pela atualização da versão da biblioteca.

```
PREFIX sm:<http://semancti/>

SELECT ?bug WHERE {
?bug a sm:Issue.
?bug sm:issueType 'Bug'.
?bug ^sm:commitIssue ?changeSet.
?changeSet a sm:ChangeSet.
?changeSet   sm:elements ?element.
?element ^sm:isRelationshipOf ?relationship.
?relationship a sm:Relationship.
?relationship rdf:value ?libraryElement.

?libraryElement  (^sm:impactOfAfter|^sm:impactOfBefore)  ?impact.
?impact  sm:contextTo  $libraryVersion2.  ?impact  sm:contextFrom
```

```
$libraryVersion1. ?impact sm:impactType ?impactType
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' }
```

Quadro 3 Consulta SPARQL que retorna os elementos que podem ter sido afetados por defeitos devido à atualização da versão de uma biblioteca do projeto

5.4. Estudos de Caso

Nesta seção vamos descrever os experimentos e resultados que realizamos em um repositório real de software. O projeto escolhido foi o Apache Mahout[85], que é uma biblioteca de aprendizagem de máquina mantida pela fundação Apache. A escolha deste projeto foi feita devido as suas características tecnológicas, pois foi escrito em Java, utiliza o Apache Maven para gerenciamento de dependência, a integração contínua do projeto é executada no Jenkins e utiliza o Atlassian Jira para controle de defeitos e melhorias. Ele contém cerca de 1820 commits e 1210 *issues* em seis anos de projeto.

5.4.1.1. Conjunto de Dados Analisados

Abaixo apresentamos uma tabela com as algumas informações da execução o processo de extração. Utilizamos uma máquina com processador 2.1GHZ, 8GB de RAM e executando o Sistema Operacional Ubuntu 10 LTS.

Período de Conjuntos de Alterações Analisado:	Janeiro de 2012 a Outubro de 2012
Número de Conjunto de Alterações Analisados:	208
Número de Modificações em Arquivos:	1510
Número de Desenvolvedores e Contribuidores Envolvidos:	12
Número de Defeitos e Demandas Resolvidos no Período:	82
Número de Triplas RDF Geradas:	68.386.832
Tempo de Processamento	170 horas

Tabela 14 Informações da execução do processo de extração

[85] http://mahout.apache.org/

5.4.1.2. Resultados das Consultas Executadas

Consulta: Quem alterou classes que eu modifico?

Como entrada desta consulta, escolhemos o desenvolvedor que mais realizou alterações estruturais no período dos conjuntos de alterações extraídos. A consulta abaixo retorna este indivíduo:

```
PREFIX sm:<http://semancti/>

SELECT  ?developer (COUNT (?impact) AS ?total)
WHERE {
?developer ^sm:commitCommiter ?changeSet.
?changeSet ^sm:contextTo ?impact.
?impact sm:impactOfAfter  ?classe.
?classe a sm:Class.
FILTER NOT EXISTS { ?impact sm:impactType 'NOTHING' } }
GROUP BY ?developer
ORDER BY DESC(?total)
LIMIT 1
```

Quadro 4 Consulta SPARQL que retorna o desenvolvedor que realizou o maior número de modificações nos elementos de código-fonte do projeto

A consulta acima retornou como resultado o desenvolvedor **Sean Owen**. Abaixo podemos observar a página[86] do GitHub que corrobora o resultado:

86 https://github.com/apache/mahout/blob/trunk/core/src/main/java/org/apache/mahout/common/AbstractJob.java

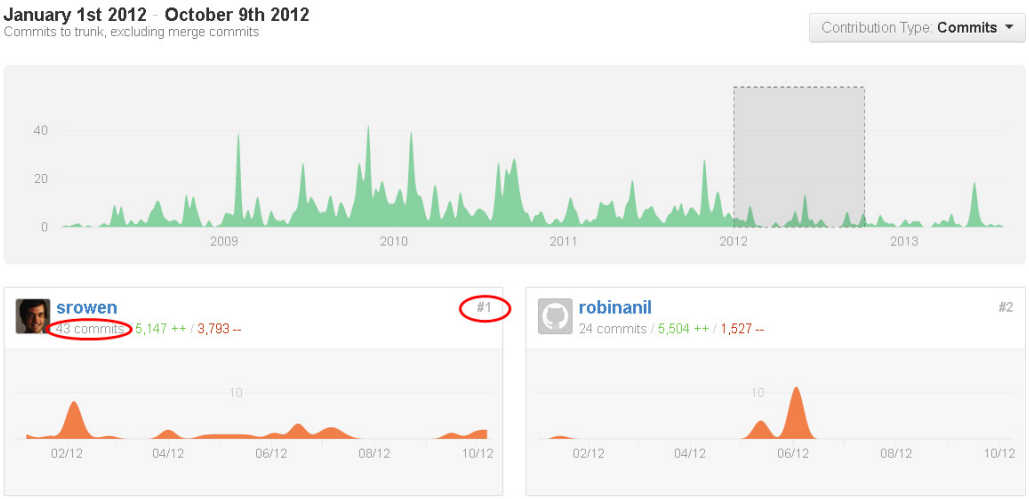

Figura 39 Imagem do GitHub que demonstra o desenvolvedor Sean Owen como o desenvolvedor com maior número de commits no período de tempo

A execução da consulta **"Quem alterou classes que eu modifico?"** com o desenvolvedor **Sean Owen** como parâmetro obteve o seguinte resultado:

Desenvolvedor	Número de Classes
Tom Pierce	51
Jeff Eastman	37
Grant Ingersoll	28
Paritosh Ranjan	20
Ted Dunning	9
Sebastian Schelter	7
Dmitriy Lyubimov	7

Tabela 15 Resultado da Consulta "Quem alterou classes que eu modifico?"

Consulta: Quem está utilizando esta API [que eu vou modificar]?

Como entrada desta consulta escolhemos os dois métodos mais invocados no projeto. A seguinte consulta retorna estes métodos.

```
PREFIX sm:<http://semancti/>

SELECT ?element (COUNT(?invokation ) as ?total)
WHERE {
?changeSet a sm:ChangeSet.
?changeSet  sm:elements ?element .
?element a sm:Method.
?element ^sm:isRelationshipOf ?isInvokedBy.
?isInvokedBy a sm:IsInvokedBy.
?isInvokedBy  rdf:value ?invokation.
?invokation a sm:Invokation.
?invokation ^sm:impactOfAfter ?impact.
 ?invokation ^sm:isRelationshipOf ?relationship.
?relationship a sm:IsInvokationOf.
?relationship rdf:value ?methodInvokes.
MINUS {
?invokation a sm:Invokation.
?invokation ^sm:impactOfAfter ?impact.
?impact sm:impactType 'NOTHING'
}
MINUS {
?invokation a sm:Invokation.
?invokation ^sm:impactOfAfter ?impact.
?impact sm:impactType 'REMOVE'
}
MINUS {
?invokation a sm:Invokation.
?invokation ^sm:impactOfAfter ?impact.
?impact sm:impactType 'MODIFY'
}
MINUS {
?invokation a sm:Invokation.
?invokation ^sm:impactOfAfter ?impact.
?impact sm:impactType 'CHANGE'
}
}
GROUP BY ?element
ORDER BY DESC(?total)
LIMIT 2
```

Quadro 5 Consulta SPARQL que retorna os dois métodos mais invocados no projeto

Resultado da Consulta:
org.apache.mahout.common.AbstractJob.getOption(String)
org.apache.mahout.common.AbstractJob.addOption(String)

Figura 40 Tabela com os métodos mais invocados no projeto

A execução da consulta **"Quem está utilizando esta API [que eu vou modificar]?"** com os dois métodos mais invocados como parâmetro obteve o seguinte resultado[87]:

- **Método:** org.apache.mahout.common.AbstractJob.getOption(String)

Método Afetado
org.apache.mahout.utils.SplitInput.parseArgs(String[])
org.apache.mahout.clustering.spectral.kmeans.SpectralKMeansDriver.run(String[])
org.apache.mahout.common.AbstractJob.getOption(String,String)
org.apache.mahout.classifier.sgd.TrainASFEmail.run(String[])
org.apache.mahout.utils.clustering.ClusterDumper.run(String[])
org.apache.mahout.clustering.minhash.MinHashDriver.run(String[])
org.apache.mahout.fpm.pfpgrowth.FPGrowthDriver.run(String[])

Tabela 16 Resultado da Consulta "Quem está utilizando esta API?" para o método "org.apache.mahout.common.AbstractJob.getOption(String)"

- **Método:** org.apache.mahout.common.AbstractJob.addOption(String)

Método Afetado
org.apache.mahout.common.AbstractJob.addInputOption()
org.apache.mahout.common.AbstractJob.addOutputOption()
org.apache.mahout.common.AbstractJob.parseArguments(String[])
org.apache.mahout.utils.SplitInput.parseArgs(String[])
org.apache.mahout.clustering.spectral.kmeans.SpectralKMeansDriver.run(String[])
org.apache.mahout.utils.clustering.ClusterDumper.run(String[])
org.apache.mahout.clustering.minhash.MinHashDriver.run(String[])

Tabela 17 Resultado da Consulta "Quem está utilizando esta API?" para o método "org.apache.mahout.common.AbstractJob.addOption(String)"

[87] Por conta do número de métodos retornados, limitamos o resultado para os 10 primeiros.

Consulta: Para quem atribuir uma revisão de código? Quem tem conhecimento para realizar a revisão de código?

Para esta consulta utilizamos como entrada os dois métodos escolhidos anteriormente:

- org.apache.mahout.common.AbstractJob.getOption(String)
- org.apache.mahout.common.AbstractJob.addOption(String)

O resultado desta consulta pode ser visto abaixo:

Desenvolvedor	Método
Grant Ingersoll	org.apache.mahout.common.AbstractJob.addOption(String)
Grant Ingersoll	org.apache.mahout.common.AbstractJob.getOption(String)
Sean Owen	org.apache.mahout.common.AbstractJob.addOption(String)
Sean Owen	org.apache.mahout.common.AbstractJob.getOption(String)

Tabela 18 Resultado da Consulta “Para quem atribuir uma revisão de código”

A confirmação deste resultado pode ser vista nas informações fornecidas pela página do GitHub[88] da classe que contem estes dois métodos:

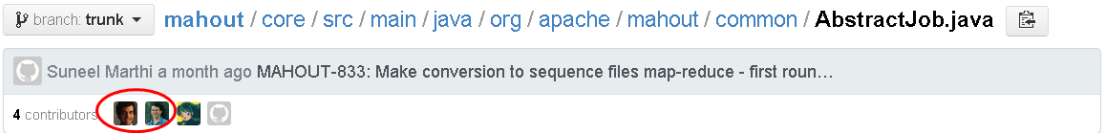

Figura 41 Imagem da Página do GitHub que indica os dois desenvolvedores retornados pela consulta como os principais contribuidores da classe

88 https://github.com/apache/mahout/blob/trunk/core/src/main/java/org/apache/mahout/common/AbstractJob.java

Consulta: Com quem falar quando se tem que trabalhar com pacotes que nunca se trabalhou?

Como entrada para esta consulta utilizamos uma versão da biblioteca Apache Lucene[89] utilizada no projeto Mahout. Esta biblioteca é utilizada para a criação de aplicações que necessitem de funcionalidades *"full text search"*. Ela é altamente escalável e fornece diversos algoritmos de pesquisa em sua distribuição. Abaixo demonstramos o resultado da consulta para a biblioteca Lucene Core na versão 3.5.0.

Desenvolvedor:
Paritosh Ranjan
Jeff Eastman
Robin Anil

Tabela 19 Listagem de desenvolvedores que já trabalharam como a biblioteca Lucene

[89] http://lucene.apache.org/core/