Instituto de Matemática e Estatística - USP

Percepção coletiva da comunidade de desenvolvimento em relação ao dbunit

Vinícius Nascimento Silva,

Orientado por Alfredo Goldman e Maurício Aniche

São Paulo, 2018

Resumo

Neste trabalho estudamos a percepção geral da comunidade de desenvolvedores de software em relação a ferramenta dbunit. O dbunit é uma extensão do Junit feita para controlar o estado do banco de dados antes de cada teste. Ele é a principal ferramenta para integração de testes automáticos com o banco de dados em Java, além de ter versões para outras linguagens como php e .NET. Para isso analisamos perguntas e respostas no site Stack Overflow relacionadas a esta ferramenta.

Nos propusemos a responder às seguintes questões de pesquisa: Qual é o tipo de pergunta mais feita pelos usuários do dbunit no Stack Overflow. **(Q1)**; Quais são os temas mais comuns nas postagens do Stack Overflow relacionadas ao dbunit. **(Q2)**; Qual é a qualidade das respostas oferecidas pelos usuários do site. **(Q3)**; E quais são as diferenças entre os usuários do dbunit e de outras tecnologias de desenvolvimento de software no Stack Overflow. **(Q4)**.

Em relação à Q1 descobrimos que há um empate entre questões do tipo “O que?” e “Como?” e um menor número de perguntas do tipo “Por que”, mas estas são praticamente duas vezes mais visualizadas do que os outros dois tipos de questões. Descobrimos também que esse tipo de distribuição não é incomum para outras ferramentas. Em relação à Q2 encontramos três temas que se destacam: “Exceções e problemas com o Maven”, “Dúvidas gerais sobre o dbunit” e “Integração com outras ferramentas” sendo que este último tema além de ser o mais presente, concentra a maioria das perguntas do tipo “Por que”. Em relação à Q3 descobrimos que a grande maioria das perguntas são respondidas e quase metade delas tem uma resposta aceita. Essa distribuição é próxima da média do site. Finalmente em relação a Q4 descobrimos que o dbunit mostrou um aumento de perguntas no início do Stack Overflow e se estabilizou nos últimos anos, mas que este crescimento segue o padrão do próprio Stack Overflow. Vimos também que o dbunit tem um número muito pequeno de perguntas se comparado com outras ferramentas como Hibernate e Spring que aparecem frequentemente juntos com o dbunit nas perguntas do site.

Introdução

Os testes automáticos são um importante elemento na área desenvolvimento de software. Através deles é possível testar especificações de um sistema de forma rápida e fácil graças a diversas ferramentas disponíveis. Os testes automáticos são rotinas de um software que verificam se o programa em desenvolvimento se comporta da forma especificada.

Com os testes automáticos é possível detectar erros mais rapidamente, já que um sistema inteiro pode ser testado com apenas poucos comandos e em um tempo consideravelmente menor do que um grupo de pessoas testando manualmente. Além da praticidade de automatizar os testes de um sistema, há diversas outras vantagens de se utilizar testes

automáticos. Ferramentas como Cucumber[1] e Codeception[2], por exemplo, geram especificações para o código a partir dos próprios testes. Além de outras ferramentas como o Travis CI[3], ou o Gitlab[4] que automatizam o processo de instalação de um sistema em produção, utilizando testes automáticos para garantir a qualidade da instalação e outras formas de monitoramento.

Os testes automáticos podem ser divididos de acordo com o escopo do objeto em teste. Os testes de unidade são os testes que verificam o comportamento de apenas uma pequena parte do código, geralmente um único método ou função, e são mantidos pelos próprios desenvolvedores. Os testes de integração são os testes que verificam a comunicação entre dois ou mais elementos de um sistema. Estes elementos podem ser desde pequenos componentes, até diferentes softwares, estes testes também são mantidos pelos desenvolvedores. Por fim os testes de aceitação são os testes que verificam um requisito do software, do ponto de vista do negócio, portanto, o sistema é testado como um todo neste nível. Um exemplo seria um teste que simula um usuário fazendo um cadastro, ou comprando um produto.

Em todos os níveis, a rotina de um teste pode ser dividida em três partes, a preparação do contexto, a ação, e a verificação (M. Fowler, 2013; B. Wake, 2011). Durante a preparação do contexto, é criado o estado inicial necessário para gerar o comportamento que será testado. Na fase de ação, é reproduzida a ação sob teste, simulando o sistema em uso, e na fase de verificação é avaliado se o estado do sistema após a ação faz parte do conjunto de estados previstos.

A fase de preparação de contexto apresenta desafios distintos para cada tipo de teste. Os testes de unidade têm contextos simples, pois apenas testam uma pequena parte do sistema, e o seu contexto se resume em dependências entre classes e os argumentos dos métodos. Já os testes de aceitação precisam de um contexto abrangente para cada teste, todas as dependências do sistema são potencialmente parte do contexto, seja banco de dados, servidores de aplicação, servidores de serviços ou ambientes de execução. A complexidade do contexto dos testes de integração é tão intensa quanto a complexidade da própria integração que está sendo testada, e todas as dependências citadas anteriormente no contexto dos testes de aceitação são candidatas a testes de integração para validar a corretude da comunicação entre estas partes.

Em um sistema onde a persistência dos dados é importante, um contexto essencial para os testes de aceitação é o estado inicial do banco de dados. É esperado que o sistema tenha comportamento diferente de acordo com o estado inicial do banco de dados. Porém, preparar uma base de dados antes de cada teste pode ser uma tarefa complexa, principalmente se o modelo dos dados é complexo e extenso. Dependendo do cenário a ser testado, dezenas de registros com vários atributos precisam ser armazenados em um banco de dados antes do teste poder ser executado. Além disso, é importante que o estado do

[1] https://cucumber.io/ (acessado em 22/08/2017)
[2] http://codeception.com (acessado em 22/08/2017)
[3] https://travis-ci.org/ (acessado em 22/08/2017)
[4] https://about.gitlab.com/ (acessado em 22/08/2017)

banco de dados seja conhecido antes de cada teste, o que dificulta o reuso de dados entre a execução de dois testes diferentes.

Foi para lidar com esses desafios que a ferramenta dbunit[5] foi criada e, neste trabalho, esperamos conseguir avaliar como a comunidade utiliza esta ferramenta e quais desafios ela encontra. O dbunit é uma extensão do Junit feita para controlar o estado do banco de dados antes de cada teste. Existem outras ferramentas similares ao dbunit como o DbSetup [6], ou o DbFit[7], mas não há postagens o suficiente no Stack Overflow com estes termos para tirarmos conclusões sobre estas ferramentas. No momento de escrita deste trabalho há apenas 152 postagens que contém o termo “DbFit” e apenas 90 postagens contendo o termo “DbSetup”, já o termo “dbunit” é citado em mais de 2000 postagens.

Há também frameworks com ferramentas internas que possuem parte das funcionalidades do dbunit, seja o ruby on rails[8] que dá a opção de preparar o estado do banco de dados para os testes, seja frameworks de testes como o Codeception, que permite acessar o banco de dados durante os testes de integração e aceitação e fazer modificações no estado inicial. Porém, o número grande de utilizações destes frameworks para fins diversos que fogem do escopo deste trabalho dificulta a seleção de postagens e introduziria muito ruído nos dados.

Devido a esses fatores, decidimos utilizar o dbunit neste trabalho. Ele é a principal ferramenta para integração de testes automáticos com o banco de dados em Java, além de ter versões para outras linguagens como php e .NET. O dbunit permite que o desenvolvedor prepare o estado do banco de dados antes de cada cenário de teste através de arquivos escritos com uma linguagem de marcação (no caso XML), além de permitir validar o estado resultante do banco de dados na fase de verificação. O dbunit é uma ferramenta que já está no mercado há 15 anos e que continua sendo atualizada.

Questões de Pesquisa

Neste trabalho pretendemos analisar o que a comunidade tem a dizer sobre o dbunit. Para isso, coletamos e analisamos postagens feitas no site Stack Overflow[9] relacionadas com este software. O Stack Overflow é uma rede social de perguntas e respostas, na qual desenvolvedores podem tirar suas dúvidas sobre diversos temas relacionados a programação de computadores. Suas perguntas são respondidas por outros usuários e a resposta que resolver o problema é marcada pelo autor da pergunta como a solução utilizada. Além disso, usuários com uma determinada reputação podem ranquear tanto as perguntas quanto as respostas postadas no site. Eles fazem isso adicionando ou removendo um ponto, quanto mais pontos, melhor.

[5] http://dbunit.sourceforge.net/ (Acessado em 26/08/2017)
[6] http://dbsetup.ninja-squad.com/ (Acessado em 16/01/2018)
[7] http://dbfit.github.io/dbfit/index.html (Acessado em 16/01/2018)
[8] http://rubyonrails.org/ (Acessado em 16/01/2018)
[9] https://stackoverflow.com/ (Acessado em 26/08/2017)

O objetivo desta análise é apresentar uma amostragem do que está sendo discutido em relação ao dbunit para, assim, descobrir tendências nesta área, encontrar gargalos na tecnologia, sugerir melhorias na documentação, entender melhor os casos em que o dbunit é, ou não, utilizado etc. Além disso, faz parte do escopo deste projeto, comparar os resultados com trabalhos similares a este, que são relacionados a outras tecnologias da área de desenvolvimento de sistemas.

Outros trabalhos já foram feitos utilizando os dados do Stack Overflow com resultados interessantes. Rosen e Shihab (2015) verificaram que as perguntas mais frequentes de desenvolvedores de aplicações móveis são sobre a distribuição dos aplicativos, APIs de programação móvel, gerenciamento de dados, contexto e sensores, ferramentas de programação móvel, e desenvolvimento de interface de usuário. Delfim, Paixão e Cassou et al (2016) mostraram que há uma forte associação entre a quantidade de informação sobre uma API no Stack Overflow e a sua utilização em sistemas na vida real.

A principal atividade do projeto é a extração dos dados do Stack Overflow e o seu mapeamento. E, com esses dados, pretendemos responder às seguintes questões de pesquisa: **Q1**: Qual é o tipo de pergunta mais feita pelos usuários do dbunit no Stack Overflow. **Q2**: Quais são os temas mais comuns nas postagens do Stack Overflow relacionadas ao dbunit. **Q3**: Qual é a qualidade das respostas oferecidas pelos usuários do site. **Q4**: Quais são as diferenças entre as dificuldades dos usuários do dbunit e de outras tecnologias de desenvolvimento de software.

Para lidar com a **Q1** vamos classificar as postagens em três modalidades: Perguntas sobre “o quê” fazer, perguntas sobre “como” fazer e perguntas sobre “por que” fazer. As perguntas do tipo “como” são postagens nas quais o autor tem um problema para resolver e quer saber qual é a melhor forma de utilizar o dbunit para resolvê-lo. As perguntas do tipo “o que” são postagens nas quais o autor tem uma solução incorreta ou imperfeita para um problema e quer saber o que está fazendo de errado. Já as perguntas do tipo “por que” são postagens nas quais o autor conhece uma técnica do dbunit, mas quer saber o por que daquela técnica ser implementada de uma forma específica. Além disso, também consideraremos nesta categoria questões conceituais sobre a ferramenta.

Inicialmente tínhamos a pretensão de utilizar técnicas de mineração de dados para ajudar a responder esta questão. Porém, a quantidade de postagens relacionadas diretamente ao dbunit foram menores do que o estimado e não foi levado em conta que mais da metade das postagens eram respostas, não perguntas. Na sessão seguinte entraremos em maiores detalhes em relação à classificação dessas postagens.

Para lidar com a **Q2**, utilizaremos a técnica Latent Dirichlet Allocation (LDA) que encontrará grupos de palavras recorrentes nas postagens possibilitando a classificação destas por temas. Com isso, poderemos extrair os tópicos mais comuns e verificar quanto cada postagem se aproxima de cada tema. O algorítmo do LDA será melhor explicado na sessão seguinte.

Para lidar com a **Q3**, coletamos informações do próprio Stack Overflow para avaliar a qualidade das respostas, mais especificamente, se houve uma resposta marcada como solução pelo autor da postagem e o número de respostas que cada postagem recebeu dos usuários do site. Esses dados foram cruzados com a classificação de tipos para entendermos como a comunidade do dbunit se comporta com cada tipo ou tema de uma postagem.

E, para lidar com a **Q4**, comparamos os resultados obtidos com trabalhos similares que utilizam os dados do Stack Overflow para avaliar outras tecnologias de desenvolvimento de software. Com isso esperamos entender melhor os desafios que os desenvolvedores enfrentam ao utilizar esta ferramenta em comparação com o resto do ciclo de desenvolvimento. Por exemplo, será que há a mesma preocupação conceitual ao se escrever testes com dbunit do que há em outras partes do ciclo? Será que o usuário de dbunit está tendo suas dúvidas respondidas com a mesma frequência que outras tecnologias? Infelizmente não há muitos trabalhos similares a este sobre outras ferramentas de testes automáticos, portanto, faremos comparações mais gerais.

Levantamento de dados

Todo o processo de levantamento de dados foi feito com apoio de scripts em python que podem ser encontrados no repositório do github deste trabalho[10]. Os scripts estão na pasta *scripts* e nos momentos pertinentes deste trabalho citaremos quais scripts foram utilizados. Um backup dos dados extraídos durante este trabalho podem ser encontrados na pasta *backup.*

O primeiro passo para a coleta e análise de dados é decidir quais postagens do Stack Overflow são relacionadas com o dbunit. Todas as postagens do Stack Overflow estão disponíveis em XML[11]. Utilizando o script *crawler-questions.py* foram separadas qualquer pergunta que contenha a palavra dbunit em qualquer parte do corpo de texto, do título e das tags. Além disso, todas as combinações de letras maiúsculas e minúsculas foram levadas em consideração. As respostas dessas perguntas foram separadas utilizando o script *crawler-answers.py*. A quantidade de postagens e a relação entre respostas e perguntas estão detalhadas no gráfico abaixo. São postagens feitas desde a fundação do Stack Overflow em 2008 até o mês de março de 2017.

[10] https://github.com/Dhinihan/TCC (Acessado em 13/09/2017)
[11] https://archive.org/download/stackexchange (Acessado em maio de 2017)

O próximo passo foi separar as perguntas entre três tipos: as perguntas nas quais o usuário não sabe o que está causando um erro específico ou um comportamento indesejado, neste trabalho serão as perguntas do tipo **O que**; as perguntas nas quais o usuário ainda não sabe como executar alguma tarefa específica utilizando a biblioteca, neste trabalho serão as perguntas do tipo **Como**; E as perguntas nas quais o usuário sabe como resolver um problema, mas deseja saber se há uma forma melhor, ou deseja saber por que uma forma específica é utilizada, além de outras perguntas mais conceituais, neste trabalho serão as perguntas do tipo **Por que**.

O objetivo dessa separação é entender melhor como os usuários usam o Stack Overflow para se informar em relação ao dbunit. Por exemplo, se o número de perguntas do tipo **O que** são muito mais representativas, talvez as mensagens de erro da biblioteca não sejam informativas o suficiente. Já um excesso de perguntas do tipo **Como** pode indicar que a documentação não é completa o suficiente, ou que é de difícil entendimento.

Na proposta deste trabalho, o objetivo era fazer esta classificação utilizando aprendizado de máquina. Porém, o número baixo de perguntas se mostrou um problema para essa abordagem. O número de classificações manuais que teriam que ser feitas para ter material de treino substancial seria uma porção muito grande do montante de perguntas (apenas 1037). Por isso, decidimos classificar todas as perguntas manualmente e o resultado será detalhado na seção de análise dos dados.

Para executar a classificação manual, implementamos o script *classify-posts.py* que sorteia uma questão ainda não classificada e o usuário escolhe qual o tipo que aquela questão faz parte, ou se ela não é relevante para este trabalho, ou seja, se a pergunta não é referente ao dbunit diretamente. Muitas dessas perguntas que foram julgadas como não referente ao trabalho continham a palavra “dbunit” apenas como parte da configuração do sistema, ou

então perguntas que buscavam ferramentas e soluções e citavam o dbunit apenas para descartá-lo como uma possível solução.

Houveram também perguntas nas quais as soluções não eram necessariamente relacionada ao dbunit, mas o usuário que postou a pergunta ainda não havia descartado a possibilidade. Essas perguntas foram classificadas como relevantes a este trabalho. Ferramentas muito citadas nas perguntas foram: *java*, *maven, junit, arquillian, spring, unitils e phpunit*.

O próximo passo foi a extração de temas através da técnica conhecida como LDA (Latent Dirichlet Allocation). O LDA automaticamente encontra um número predeterminado de tópicos em um conjunto de textos. Para o modelo do LDA cada postagem é composta por uma mistura de tópicos, e cada tópico possui uma probabilidade de produzir uma palavra específica. Então, através de uma engenharia reversa, ele tenta descobrir a composição de tópicos que produziu cada texto do conjunto.

Para ilustrar o modelo, podemos criar um exemplo dessa suposta produção de textos através de uma mistura dos seguintes tópicos:

- **Esporte:**
    - Palavras possíveis: Bola (50%), Jogador (30%), Vitória (10%), Derrota (10%)
- **Culinária:**
    - Palavras possíveis: Pão (40%), Forno (30%), Gratinar (20%), Farinha (10%)

Então, à partir destes tópicos, vamos criar dois textos de 10 palavras, um contendo 80% Culinária e 20% Esporte e o segundo contendo 50% Culinária e 50% Esporte:

- Primeiro texto:
    - Pão Gratinar Bola Forno Gratinar Gratinar Pão Jogador Farinha Farinha
- Segundo Texto
    - Bola Derrota Derrota Pão Vitória Pão Bola Farinha Forno Pão

Nesses exemplos fica claro que o modelo não é projetado para gerar textos, mas ele é poderoso se assumirmos que os textos (E, no caso deste trabalho, as postagens) foram formados utilizando este método, o LDA pode cruzar as frequências de cada palavra nos textos, e quais palavras tendem a aparecer juntas e, assim, apresentar tópicos que poderiam ter gerado aqueles textos.

Para encontrar os tópicos, o LDA cria um estado inicial, atribuindo cada palavra que aparecem nos textos a um número predeterminado de tópicos. A partir daí ele percorre cada palavra de cada texto, assumindo que ela é a única palavra que está classificada incorretamente e recalcula a probabilidade desta palavra ser produzida por cada tópico. Este processo é repetido até que o sistema fique estável, ou seja, que as probabilidades mudem arbitrariamente pouco a cada iteração. Assim, o algoritmo retorna a probabilidade de cada palavra que aparece nos textos ser produzida por cada tópico. O número de tópicos é definido pelo usuário.

Com o baixo número de postagens, foi necessário simplificar os textos para que símbolos com pouco significado pudessem ser ignorados (como marcações de HTML, ou trechos de código). Isso foi feito com o script *process-posts.py,* que, além disso, aplicou a técnica de stemização para simplificar o texto.

Stemização consiste em reduzir as palavras de um texto à sua raiz, ou seja, reduzir flexões diversas de uma raiz semântica para o mesmo símbolo. Por exemplo: “pesquisa”, “pesquisar” e “pesquisado” se referem ao conceito de “pesquisa”, portanto, o processo de stemização transformará cada ocorrência dessas palavras em um único símbolo (“pesquis” por exemplo). Esse processo representa melhor a frequência de palavras de mesmo significado entre as postagens.

O processo de limpeza das postagens se deu da seguinte forma: O primeiro passo foi tirar qualquer marcação que não fosse parte da pergunta em si, seja exemplos de código ou *tags* html. Isso nos ajuda a encontrar os temas, pois esses símbolos não carregam significado por si mesmos e poderiam separar as perguntas por características não temáticas, como por exemplo o algoritmo poderia separar as perguntas entre aquelas que têm código ou não.

O segundo passo foi tirar as *stop words*. *Stop words* são palavras que não carregam muito significado e que apenas ajudam na comunicação. Exemplos de palavras retiradas neste momento são “I”, “it”, “my”, “the”. Deixar estas palavras no texto poderia gerar classificações indesejadas, já que aparecem com muita frequência em qualquer texto independentemente do tema.

O terceiro passo é a stemização, explicada no parágrafo anterior. E o quarto passo foi acrescentar o título no início do texto duas vezes, para que as palavras que o compõem sejam consideradas com peso duplo. Assim, as palavras que o usuário considerou mais importantes para identificar a sua questão também serão importantes para decidir o tema que a postagem pertence.

Eis um exemplo de uma postagem antes e depois deste processo:

Antes:

> Título: “DbUnit excel nullable foreign key”
>
> Corpo: “ I'm trying to set a nullable <code>BIGINT</code> foreign key in my excel file to fill the data for my <code>DbUnit</code> test, but I'm getting the following exception when leaving the cell empty:
>
> <pre><code>Caused by: org.dbunit.dataset.datatype.T (...)”

Depois do primeiro passo (remoção de símbolos e código):

> “I'm trying to set a nullable foreign key in my excel file to fill the data for my test but I'm getting the following exception when leaving the cell empty (...)”

Depois do segundo passo (remoção de *stop words*):

> “trying set nullable foreign key excel file fill data test getting following exception leaving cell empty (...)”

Depois do terceiro passo (stemização):

> “tri set nullabl foreign key excel file fill data test get follow except leav cell empti (...)”

Depois do quarto passo (adição do título duas vezes):

> “dbunit excel nullabl foreign key dbunit excel nullabl foreign key tri set nullabl foreign key excel file fill data test get follow except leav cell empti (...)”

Para extrair os temas, foi utilizado o script extract-themes.py. Decidimos extrair apenas três temas, pois com quatro ou mais, um grande número de postagens tinha menos de 50% de chances de pertencer a qualquer um dos tópicos.

Neste trabalho utilizamos também o sistema de buscas na base de dados do Stack Overflow[12]. Através desse sistema é possível fazer buscas em formato de SQL em toda a base de perguntas e respostas do Stack Overflow. Com isso foi possível comparar tendências da comunidade do dbunit com o geral do Stack Overflow. Todas as buscas foram efetuadas em outubro de 2017 e levaram em conta dados de todos os períodos disponíveis até em então.

Análise de dados

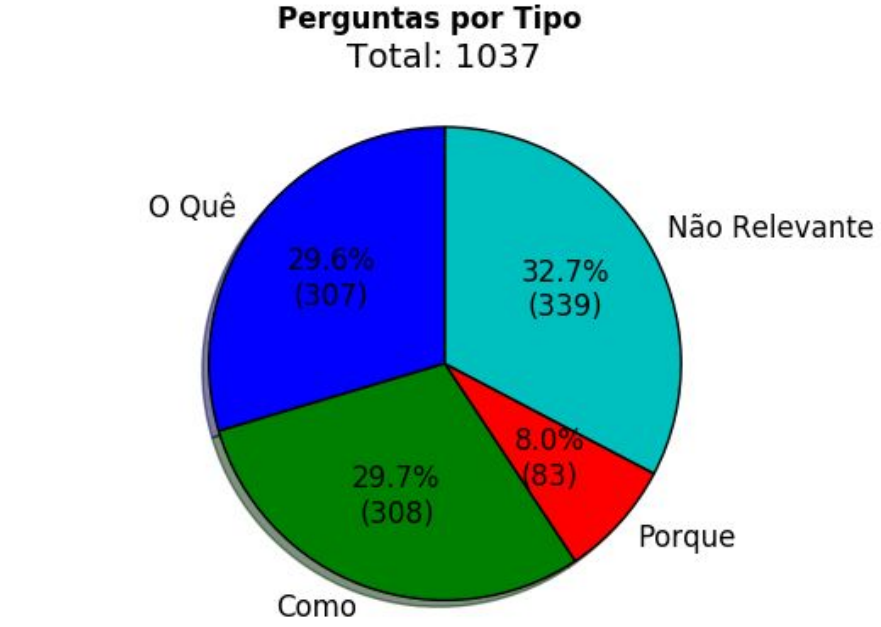
Questão 1

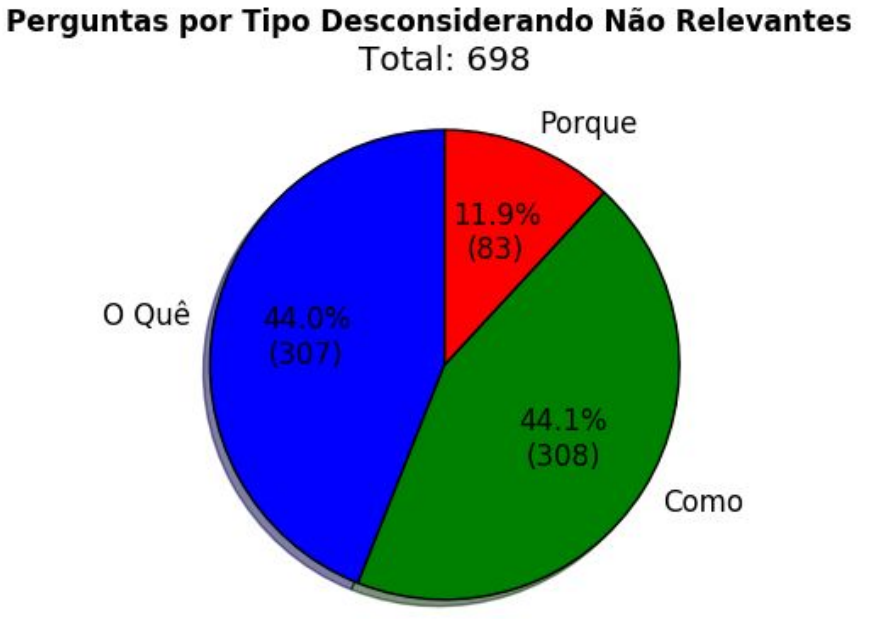
Com os dados reunidos do site Stack Overflow, conseguimos responder as perguntas propostas neste trabalho. Nesta seção detalharemos os resultados que ajudam no entendimento de cada questão.

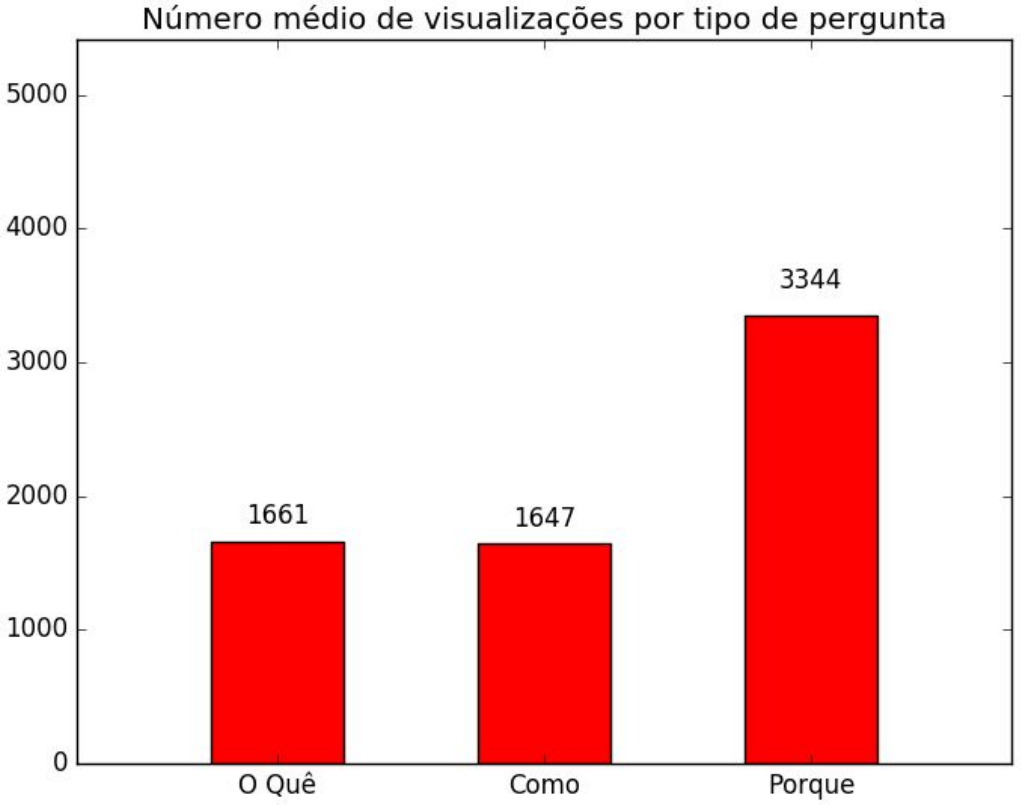
A primeira questão proposta foi: *Qual é o tipo de pergunta mais feita pelos usuários do dbunit no Stack Overflow* (**Q1**). Para entendermos melhor como se divide as questões feitas no site Stack Overflow, classificamos manualmente as postagens em 4 tipos: *O quê*, *Como, Por que* e *Não Relevante.* As postagens classificadas como não relevantes são aquelas que não têm uma relação direta com o dbunit, apenas citam a biblioteca dentro do contexto do problema. Um caso muito comum são perguntas sobre a configuração de um sistema na qual o dbunit é citado mas não é a causa do problema em si.

[12] https://data.stackexchange.com/stackoverflow/query/new (Acessada em 23/10/2017)

Nos dois gráficos seguintes (Perguntas por Tipo e Perguntas por Tipo Desconsiderando Não Relevantes) podemos ver a distribuição dos tipos de perguntas sobre o dbunit no Stack Overflow. É possível ver nestes gráficos que as perguntas do tipo *O Que* e *Como* estão praticamente empatadas e são muito mais proeminentes do que as questões do tipo *Por que*. Porém no gráfico "Número médio de visualizações por tipo de pergunta" é possível ver que as questões do tipo *Por que* são praticamente duas vezes mais visualizadas do que os outros dois tipos de questões. Uma hipótese para essa discrepância de visualizações é que algumas perguntas do tipo *Por que* envolvem diversas ferramentas além do próprio dbunit, o que potencialmente atrairia um número maior de interessados. Outra hipótese é que o pequeno número de perguntas conceituais faz com que elas apareçam com mais frequência nas buscas de usuários sobre este tipo de questão.







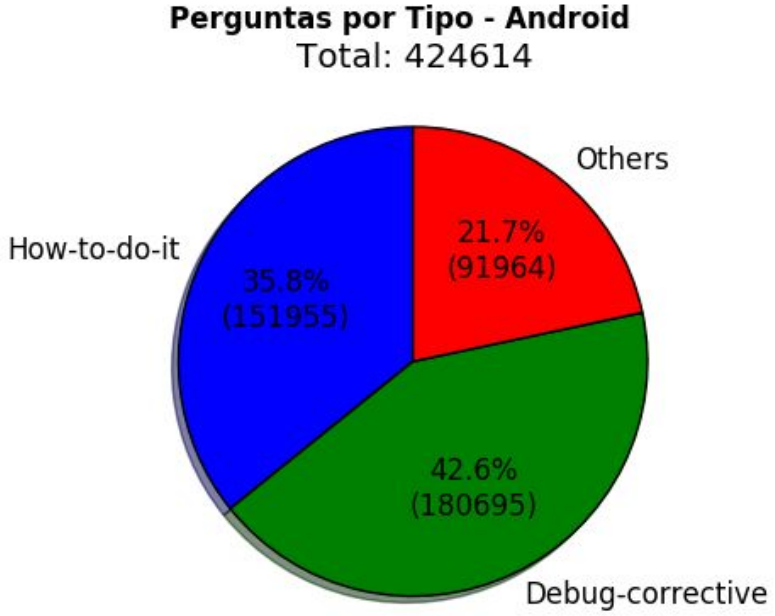
Para entender melhor essa distribuição de questões por tipo, é interessante comparar com outros trabalhos. Em um trabalho similar intitulado: *Redocumenting APIs with crowd knowledge: a coverage analysis based on question types* (Delfim et al, 2016) foram analisadas questões no Stack Overflow e dividas em três tipos: *How-to-do-it, Debug-corrective* e *Others.* Foi inspirado neste trabalho que criamos a nossa própria classificação de tipos. Apesar de o Android e o Swing serem bibliotecas mais populares e de usos muito mais gerais do que o dbunit, podemos aproveitar a classificação similar feita neste trabalho para entendermos melhor a distribuição dos tipos de perguntas do próprio dbunit e como ele se compara com dados de outras bibliotecas.

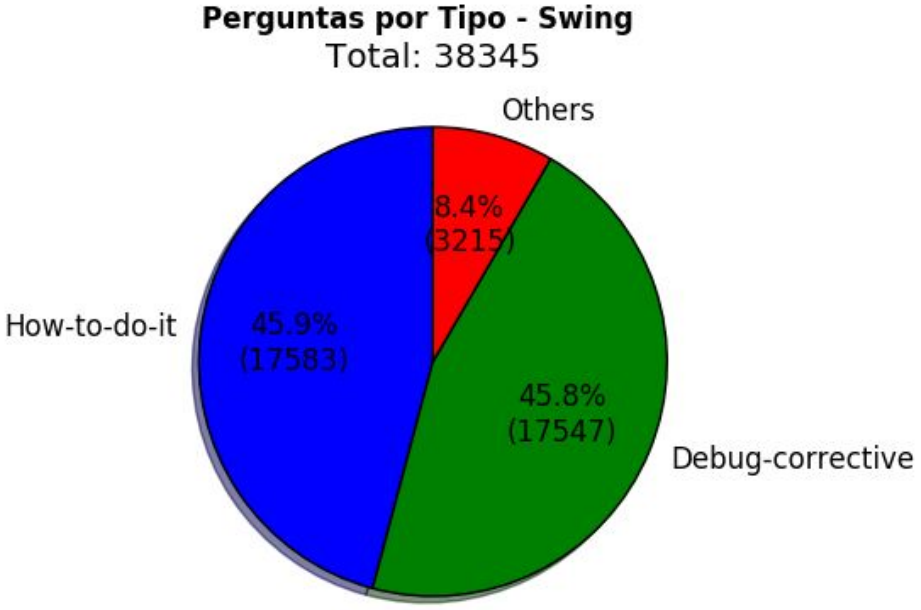
Nos dois gráficos seguintes (Perguntas por Tipo - Android e Perguntas por Tipo - Swing) é possível ver a distribuição desses tipos em questões sobre o *Android* e sobre o *Swing.* Fazendo uma correlação entre as duas metodologias de classificação, é possível correlacionar as questões do tipo *How-to-do-it* com as questões do tipo *Como* e as questões do tipo *Debug-corrective* com as questões do tipo *O que*.

Desta forma vemos que a distribuição das perguntas sobre o Swing seguem uma distribuição similar ao dbunit, já o Android tem uma distribuição diferente, com uma proporção muito maior de questões que não se encaixam nos tipos *How-to-do-it* e *Debug-corrective* e com uma pequena predominância das questões do tipo Debug-corrective em relação às questões do tipo How-to-do-it. Uma hipótese que pode ajudar a explicar esta similaridade entre o dbunit e o Swing, que não se repete em relação

ao Android, seria que a comunidade do dbunit tem uma relação mais próxima com o ambiente do Swing do com o ambiente do Android.

Os dados seguintes foram extraídos do trabalho *Redocumenting APIs with crowd knowledge: a coverage analysis based on question types* (Delfim et al, 2016):





Neste trabalho faremos mais comparações com dados gerais do Stack Overflow. Porém, infelizmente, não há muitos trabalhos sobre percepção da comunidade sobre outras ferramentas de testes automáticos, o que dificulta comparações mais precisas.

Questão 2

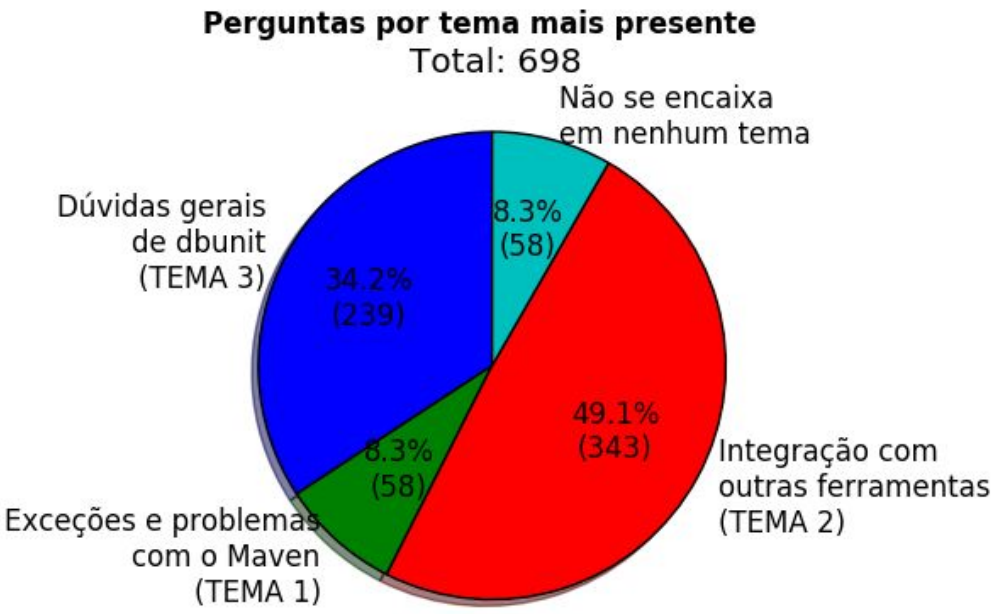
A questão “Quais são os temas mais comuns nas postagens do Stack Overflow relacionadas ao dbunit.” (**Q2**) encontramos três temas presentes através do LDA (algorítmo explicado na sessão anterior).

O **tema 1** está relacionado com as palavras: ‘*java’, ‘dbunit’, ‘org’, ‘xml’, ‘use’, ‘error’, ‘maven’, ‘test’, ‘file’* e *hsqldb*; e a pergunta que mais representa este tema pode ser encontrada neste link: https://stackoverflow.com/questions/29857599. As postagens deste tema são, no geral, relacionadas à dúvidas em relação a exceções lançadas pelo Java durante a execução de algum teste, sendo muitas delas durante algum tipo de integração com o Maven, em poucas palavras: *Exceções e problemas com o Maven.*

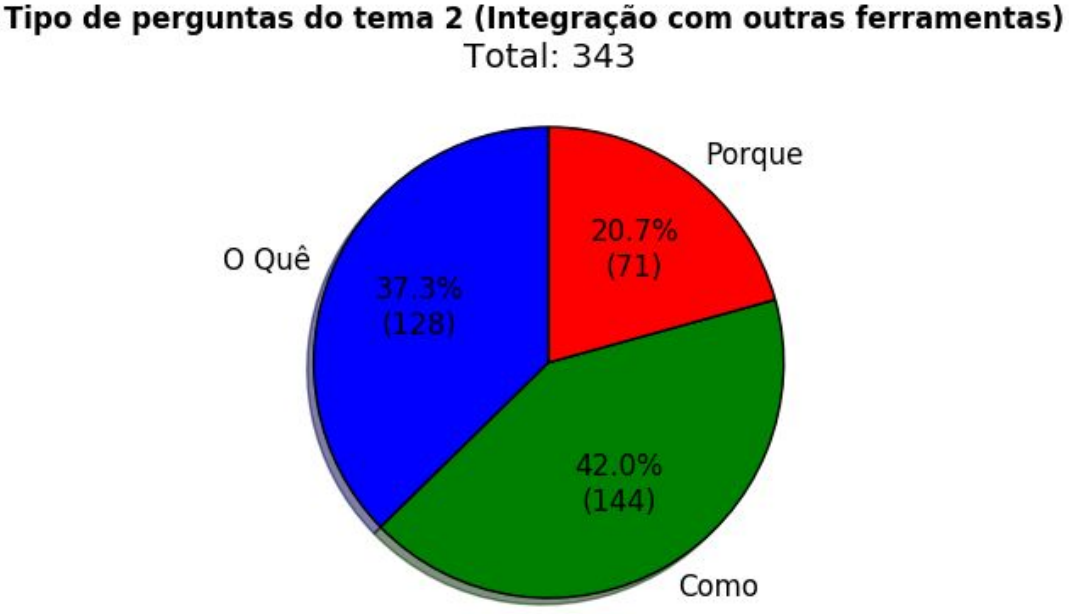
O **tema 2** está relacionado com as palavras: *‘test’, ‘database’, ‘use’, ‘dbunit’, ‘spring’, ‘data’, ‘run’, ‘class’, ‘hibernate’* e *‘method’*; e a pergunta que mais representa este tema pode ser encontrada neste link: https://stackoverflow.com/questions/33784171. As postagens deste tema são, no geral, relacionadas à integração do dbunit com outras ferramentas como Spring, Hibernate ou Arquillian, em poucas palavras: *Integração com outras ferramentas.*

O **tema 3** está relacionado com as palavras: *‘dbunit’, ‘table’, ‘xml’, ‘use’, ‘dataset’, ‘file’, ‘data’, ‘insert’, ‘column’, ‘database’*; e a pergunta que mais representa este tema pode ser encontrada neste link: https://stackoverflow.com/questions/40740545. As postagens deste tema são, no geral, relacionadas à utilização do dbunit em si e em problemas com a representação dos dados em arquivos de xml, em poucas palavras: *Dúvidas gerais de Dbunit.*

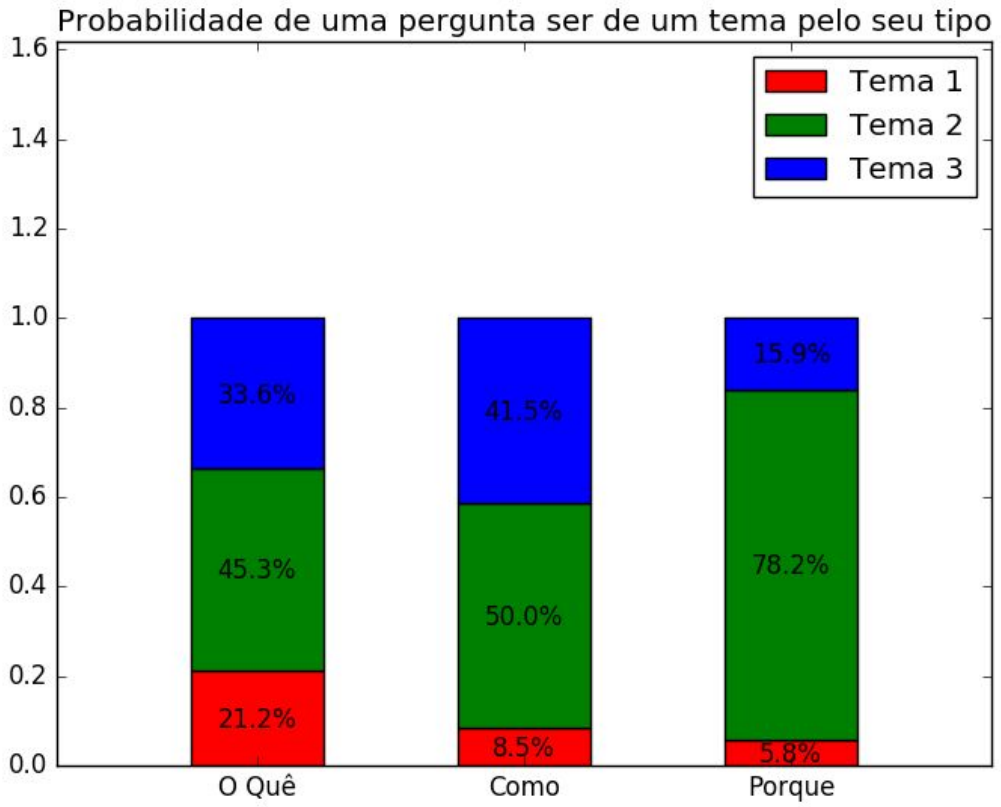
Para classificar as questões em cada tema encontrado, podemos atribuir um tema para uma postagem se ela tiver 50% ou mais probabilidade de pertencer àquele tema. No próximo gráfico é possível ver a distribuição gerada por esta classificação e a predominância de questões relacionadas à integrações com outros sistemas.



Chama a atenção essa quantidade de perguntas relacionadas à integração com outros frameworks. Analisando os tipos de perguntas que pertencem a este tema podemos ver no próximo gráfico que há uma proporção maior de perguntas do tipo *Por que.*



Mesmo sem esta classificação de 50% de probabilidade de pertencimento a um tema, e olhando apenas para as probabilidades, é possível ver no próximo gráfico que as perguntas do tipo *Por que* tem uma probabilidade grande de pertencer ao Tema 2



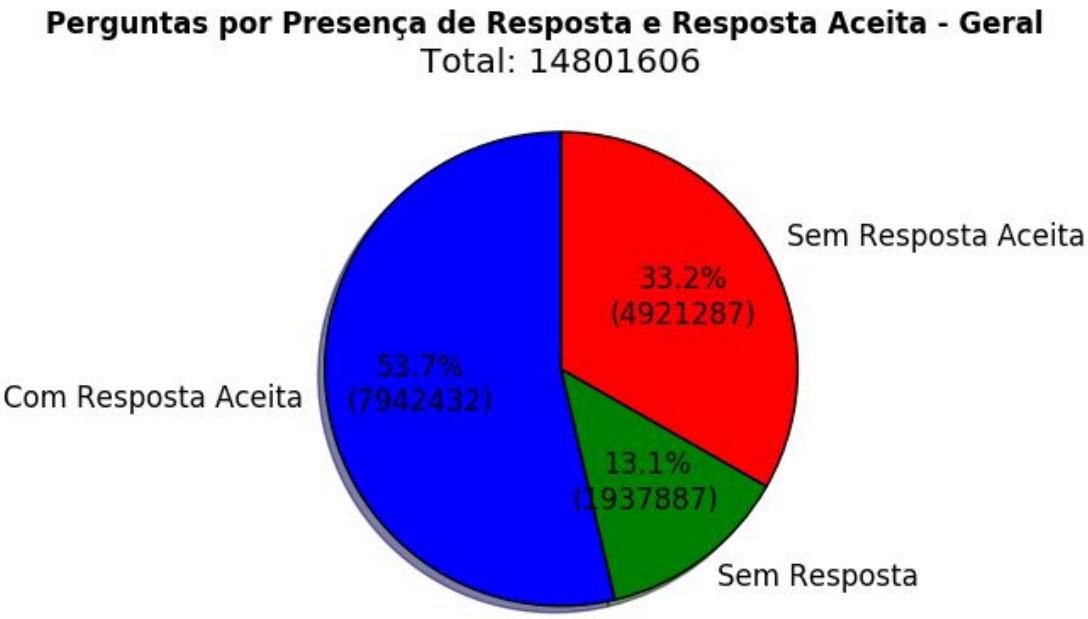
Com esses dados já é possível apreender algumas tendências das perguntas feitas em relação ao Dbunit no Stack Overflow, a mais importante é que a maioria das questões, quase metade delas, se referem a integração com outras ferramentas, principalmente as questões mais conceituais (do tipo *por que*). São quase inexistentes as questões conceituais fora deste tema. Isso é um indício interessante sobre a forma como o dbunit é utilizado, uma hipótese interessante é que o dbunit precisa de uma melhor integração com outras ferramentas, com uma interface mais simples ou mais bem documentada.

Há também um número considerável de questões envolvendo problemas de integração com a biblioteca Maven, a ponto do algoritmo do LDA considerar quase um tema por si só. O Maven é uma ferramenta de gerenciamento do Java[13], mas que também pode ser utilizada para rodar os testes automáticos.

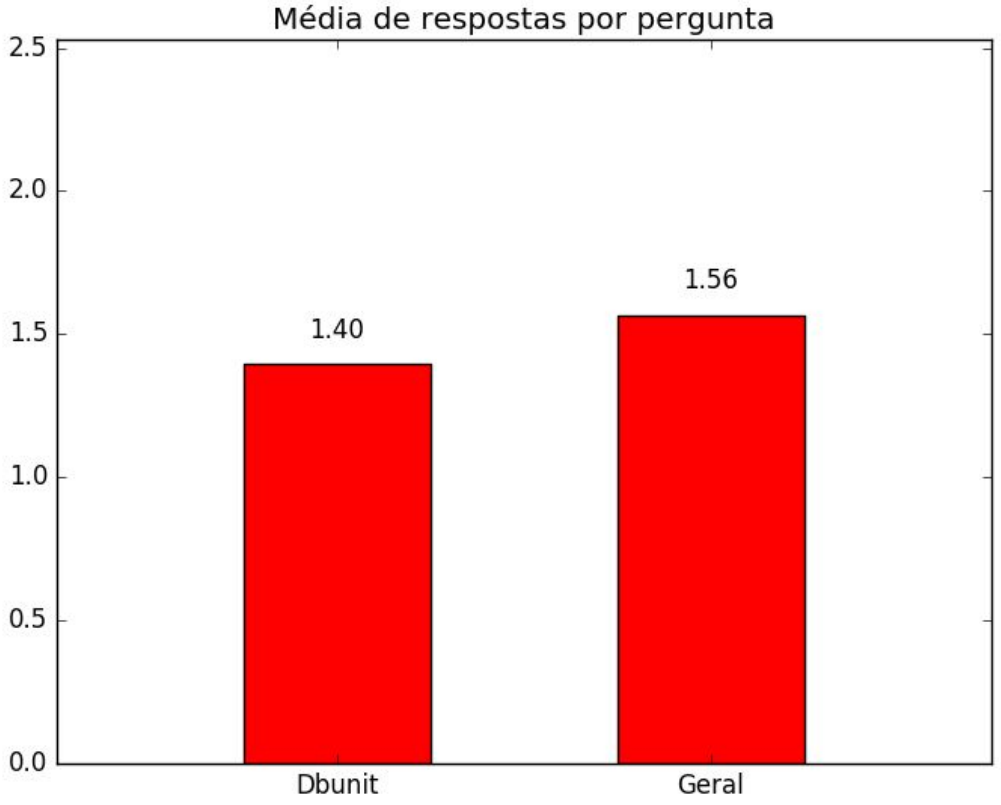
Questão 3

Para a questão “Qual é a qualidade das respostas oferecidas pelos usuários do site.” (**Q3**) encontramos dados que mostram que a qualidade das respostas está levemente abaixo do padrão do site. Nos dois gráficos seguintes podemos ver que a quantidade de respostas aceitas em média é menor para o dbunit do que a média, porém a diferença é pequena.

[13] https://maven.apache.org/ (Acessado em 13/09/2017)



A média de respostas por perguntas do dbunit está também levemente abaixo da média geral, como é possível ver no gráfico seguinte. Apesar dessas diferenças, acreditamos que o número de perguntas relacionadas diretamente ao dbunit é pequeno demais para concluir que a quantidade e, principalmente, a qualidade das respostas oferecidas apresente alguma forma de deficiência.



Questão 4

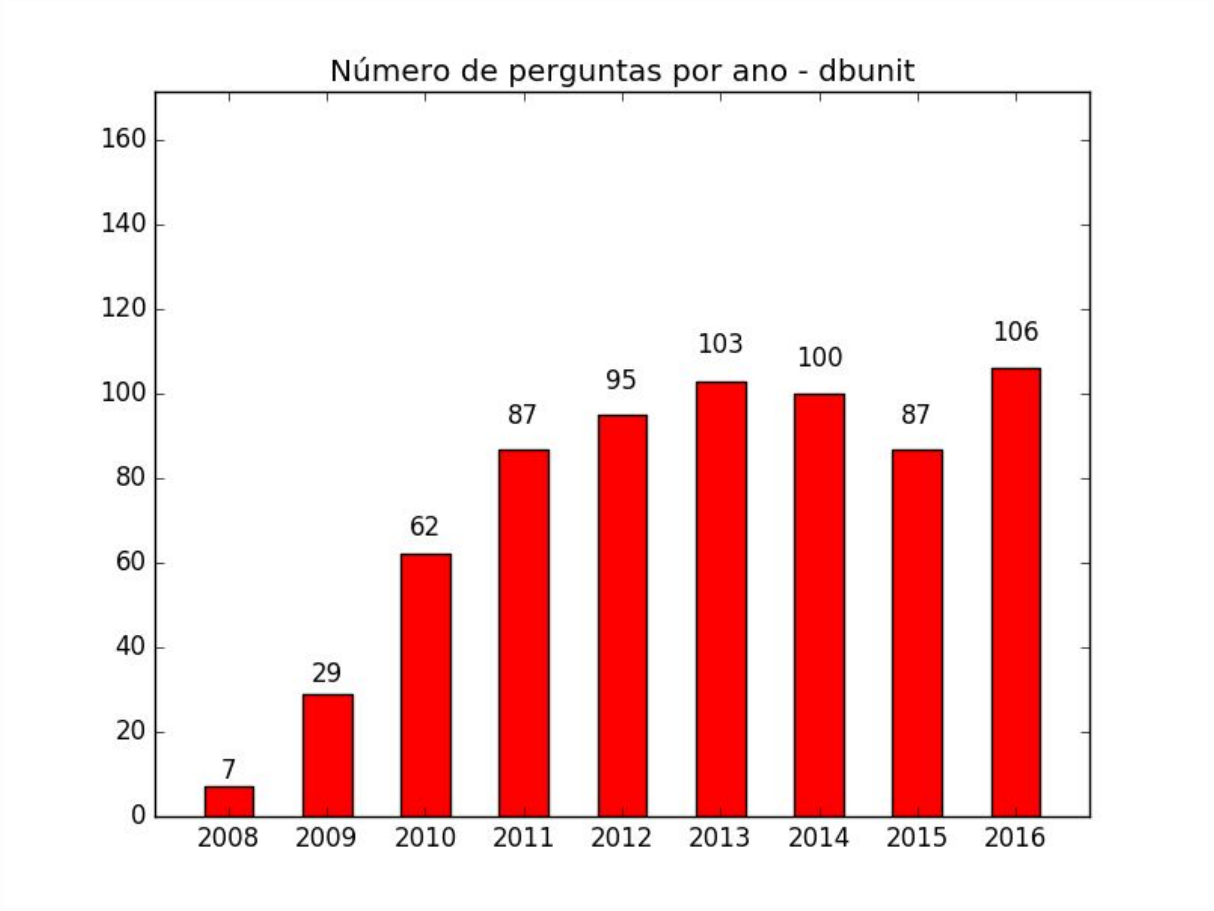
Parte da questão “Qual é a qualidade das respostas oferecidas pelos usuários do site.” (**Q4**) já apresentamos como parte das respostas às questões anteriores. Podemos voltar ao trabalho de Delfim, Paixão e Cassou et al sobre a possibilidade do uso do Stack Overflow como fonte de *Crowd Knowledge* para a documentação de APIs. Avaliando nossos resultados, é possível descartar a possibilidade deste tipo de documentação para o dbunit devido ao baixíssimo número de perguntas comparado com as ferramentas citadas no trabalho (no caso Android e Swing).

Outro aspecto é a pequena quantidade de perguntas no site sobre o dbunit. Em uma listagem com as 1000 tags mais comuns em perguntas no site[14] a tag “dbunit” tem aproximadamente 10 vezes menos perguntas que a milésima tag da lista. No caso, em 31 outubro de 2017, a milésima tag é “Lisp” com 5231 postagens enquanto o dbunit tinha apenas 525.[15] Mesmo outras ferramentas que aparecem em várias perguntas analisadas neste trabalho têm uma quantidade muito maior de perguntas no site. Alguns exemplos são: Hibernate com 67270 postagens e Spring com 120698 postagens. Se olharmos para a **Q2**, essas duas ferramentas são chaves para definir o tema mais presente dentre as postagens

[14] https://data.stackexchange.com/stackoverflow/query/749850/top-1000-tags-by-post-count (Acessado em 31/10/2017)
[15] https://stackoverflow.com/questions/tagged/dbunit (Acessado em 31/10/2017)

do dbunit. Mesmo assim, o dbunit tem um número mais de 100 vezes menor de perguntas do que essas ferramentas.

Apesar do baixíssimo número de questões no site os dois gráficos seguintes mostram que a quantidade de perguntas relacionadas diretamente ao dbunit vem crescendo, o que é um bom sinal para uma ferramenta com 15 anos de uso, porém a taxa de crescimento é inferior ao crescimento geral do site. Isso pode ocorrer devido a novas linguagens que apareceram neste período e que não são compatíveis com o dbunit, mas pode significar também, uma queda de uso e de interesse pela ferramenta.



Conclusão

Neste trabalho apresentamos diversos dados sobre o comportamento dos usuários do dbunit no site Stack Overflow e pudemos tirar algumas conclusões sobre a ferramenta com base nisso. Sabemos que a quantidade de postagens que buscam informações de *como* usar a ferramenta é praticamente igual ao número de postagens buscando encontrar o *que* está errado em uma implementação específica, o que mostra um equilíbrio na utilização do site. Comparando com uma pesquisa similar relacionada ao Android e ao Swing, podemos ver que essa distribuição é comum entre outras ferramentas.

Também podemos concluir que problemas com a integração do dbunit com outras ferramentas (como o Spring, Hibernate e o Maven) são comuns entre os usuários do site. Sugerimos aos mantenedores do dbunit e dessas ferramentas que criem interfaces mais simples e com essas integrações em mente, ou então que criem documentação e tutoriais focados em auxiliar os usuários nessas tarefas. Outro tema de pergunta comum é sobre o funcionamento do próprio dbunit, que é um tema esperado já que esse tipo de pergunta geral é comum para aprender a usar a ferramenta.

Outra conclusão que podemos tirar dessa pesquisa é que, apesar de o dbunit ser uma ferramenta lançada há mais de 15 anos e não ser uma ferramenta tão presente no Stack Overflow, a utilização do site pelos usuário do dbunit não apresenta uma desaceleração

aparente, mas que não seguiu o crescimento do próprio site. Este fato sugere que houve demanda por esse tipo de ferramenta nestes oito anos de Stack Overflow. Porém o distanciamento dos números de ferramentas relacionada ao dbunit no Stack Overflow mostra que há um potencial de crescimento para a ferramenta inexplorado. Um exemplo é o *Hibernate*, uma ferramenta que traduz o esquema do banco de dados para objetos no sistema. seus usuários poderiam se beneficiar do uso de testes de integração através do dbunit para assegurar a qualidade deste mapeamento.

Apesar da pouca quantidade de questões relacionadas ao dbunit, a quantidade de perguntas respondidas seguem a proporção das postagem do Stack Overflow, com apenas uma ligeira deficiência no número de respostas aceitas e na média de respostas por pergunta.

O dbunit é uma ferramenta que resistiu ao teste do tempo e se mantém como uma referência em testes de integração com o banco de dados. Com tantas ferramentas entrando e saindo do mercado é importante que ferramentas como o dbunit encontrem formas simples de se comunicar com as diversas opções apresentadas aos desenvolvedores na hora de construir um sistema. E é importante que a comunidade, mesmo que pequena, continue a transmitir conhecimento para qualquer pessoa interessada em aprender.

Bibliografia

“Glossary of Software Testing Terms” (**n.d.**) *http://www.csc.villanova.edu/~tway/courses/csc4700/s2004/testing_glossary.html*. Retrieved from http://www.csc.villanova.edu/~tway/courses/csc4700/s2004/testing_glossary.html.

Bill Wake (**2011**) – Arrange, Act, Assert. *https://xp123.com/articles/3a-arrange-act-assert/*. Retrieved from https://xp123.com/articles/3a-arrange-act-assert/.

Delfim, Paixão, Cassou, and Maia (**2016**) Fernanda Madeiral Delfim, Klérisson V. R. Paixão, Damien Cassou and Marcelo De Almeida Maia. Redocumenting APIs with crowd knowledge: a coverage analysis based on question types. *Journal of the Brazilian Computer Society*. 22, 1 (2016). doi: 10.1186/s13173-016-0049-0.

“Wikipedia” (**2017**) Latent Dirichlet allocation.

https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation. Retrieved from https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation.

Rosen and Shihab (**2015**) Christoffer Rosen and Emad Shihab. What are mobile developers asking about? A large scale study using stack overflow. *Empirical Software Engineering*. 21, 3 (2015), 1192–1223. doi: 10.1007/s10664-015-9379-3.

"Agile Alliance" (**2017**) What is Acceptance Testing? *https://www.agilealliance.org/glossary/acceptance/*. Retrieved from https://www.agilealliance.org/glossary/acceptance/.

"Agile Alliance" (**2017**) What is Unit Testing? *https://www.agilealliance.org/glossary/unit-test/*. Retrieved from https://www.agilealliance.org/glossary/unit-test/.

Martin Fowler (**2013**) bliki: GivenWhenThen. *https://martinfowler.com/bliki/GivenWhenThen.html*. Retrieved from https://martinfowler.com/bliki/GivenWhenThen.html.