Documentação Automatizada de APIs com Tutoriais Gerados a Partir do Stack Overflow

Adriano M. Rocha
Faculty of Computing
Federal University of Uberlândia
Uberlândia, Brazil
adriano.comp2@gmail.com

Marcelo A. Maia
Faculty of Computing
Federal University of Uberlândia
Uberlândia, Brazil
marcelo.maia@ufu.br

ABSTRACT

O reuso de software oferece benefícios durante os processos de desenvolvimento e manutenção de software. O uso de APIs é uma das maneiras mais comuns de reuso. Porém, a obtenção de uma documentação de fácil compreensão é um desafio encontrado pelos desenvolvedores. Vários trabalhos propuseram alternativas para tornar a documentação de APIs mais compreensível, ou até mais detalhada. Entretanto, não tem sido levada em consideração a complexidade de entendimento dos exemplos para tornar estas documentações adaptáveis a diferentes níveis de experiência de desenvolvedores. Neste artigo desenvolveu-se e avaliou-se quatro metodologias para gerar tutoriais para APIs a partir do Stack Overflow e organizá-los conforme a complexidade de entendimento. As metodologias foram avaliadas por meio de tutoriais gerados para a API Swing. Foi conduzido um *survey* para avaliar oito diferentes características dos tutoriais gerados. O resultado geral da avaliação foi positivo nas diversas características, mostrando a viabilidade para uso de tutoriais gerados automaticamente. Além disso, o uso de critérios para apresentação dos elementos por ordem de complexidade, a separação em partes básica e avançada, a natureza de tutorial para os *posts* e existência de código-fonte didático tiveram resultados significantemente diferente em relação à metodologia de geração escolhida.

CCS Concepts

•Software and its engineering → Software libraries and repositories;

Keywords

Redocumentação de APIs; Mineração de Repositórios; Stack Overflow

1. INTRODUÇÃO

O reuso de software tem sido indicado durante os processos de desenvolvimento e manutenção de software, pois oferece benefícios tais como: redução do tempo e custos no desenvolvimento; produtos de melhor qualidade e confiabilidade; uso eficaz de especialistas; etc. [19]. Uma das maneiras mais comuns de reuso é por meio de APIs (*Application Programming Interfaces*), que é um conjunto de padrões de programação que permite aos desenvolvedores utilizarem suas funcionalidades sem a necessidade de entender seus detalhes de implementação. Porém durante atividades de reuso de software, desenvolvedores geralmente encontram dificuldades para compreender a documentação da funcionalidade que se deseja reutilizar [23]. Atualmente existem várias APIs que podem ser utilizadas durante o desenvolvimento de software, mas geralmente os desenvolvedores têm dificuldades de utilizarem estas APIs por falta de uma documentação de fácil compreensão [17], [26]. Uma das alternativas para buscar um melhor entendimento sobre uma dada API é recorrer à internet em sites de busca por conteúdo que possa ajudar o desenvolvedor a entender as funcionalidades da API de forma intuitiva (trechos simples de código que explique de forma clara uma certa funcionalidade) [14].

O Stack Overflow (SO) é um site de perguntas e respostas (*posts*) relacionadas ao desenvolvimento de software onde muitos desenvolvedores buscam informações que possam auxiliá-los durante o processo de desenvolvimento ou manutenção de software. Os desenvolvedores podem enviar dúvidas relacionadas à programação, e geralmente os mais experientes no assunto enviam melhores respostas, de forma a esclarecer as dúvidas dos desenvolvedores menos experientes no assunto. Todo o conteúdo fica disponível para que essas dúvidas e suas respectivas respostas possam ser acessadas pela comunidade. Treude et al. [24] mostraram que o SO possui mais de 12 milhões de visitantes e 135 milhões de visualizações de páginas todo mês, isto faz com que este site seja uma fonte importante de informações que os desenvolvedores podem utilizar no processo de desenvolvimento ou manutenção de software. Os desenvolvedores têm acessado o SO em busca de informações que possam melhorar o entendimento das APIs. Um problema destes sites é que possuem um grande volume de informação, dificultando o acesso ao conteúdo que realmente ajude os desenvolvedores a esclarecerem suas dúvidas [16]. O SO oferece mecanismos de busca, tais como: barra de pesquisa; filtragem por *tags*, usuários, entre outros. Estes mecanismos auxiliam os desenvolvedores a encontrarem suas respostas, mas mesmo assim ainda retornam muito conteúdo, e no caso da barra de pesquisa, dependendo da formulação da pesquisa do usuário, esta pode excluir *posts* que poderiam ajudar no entendimento da API. Vários trabalhos [5], [7], [11], [20], [22] foram

SBES '16, September 19-23, 2016, Maringá, Brazil

DOI: http://dx.doi.org/10.1145/2973839.2973847

propostos para tornar a documentação de APIs mais compreensível, ou até mais detalhada, entretanto ainda não têm levado em consideração a complexidade de entendimento dos exemplos destas documentações, podendo o desenvolvedor perder um tempo considerável até encontrar a informação desejada.

Souza et al. [20] propuseram uma abordagem para encontrar temas de uma certa API no SO, e através destes temas construir um livro de receitas para a API. Esta abordagem é baseada na filtragem do conteúdo do SO, selecionando tópicos específicos através de um classificador para questões do tipo (*"How-to-do"*)[1],[4]. Esta abordagem não leva em consideração a complexidade de entendimento dos exemplos do livro de receitas, ou seja, pode acontecer de um capítulo do livro de receitas seja formado apenas com exemplos de difícil compreensão. Outro problema é que não há uma separação entre capítulos mais complexos e capítulos mais básicos, de caráter introdutório.

Para solucionar estas limitações, neste trabalho foram desenvolvidas e analisadas quatro diferentes metodologias para filtrar os *posts* do SO e organizá-los conforme a complexidade de entendimento. Desta forma, um desenvolvedor ainda inexperiente com uma certa API, pode começar com a leitura de um *post* mais simples (complexidade de entendimento menor), e depois de entender estes mais simples pode continuar com outros de complexidade mais alta.

O artigo está organizado como a seguir. Na Seção 2 são apresentadas as quatro metodologias propostas neste trabalho para a geração de tutoriais para uma API. O procedimento adotado na avaliação e comparação destas metodologias é apresentado na Seção 3. Na Seção 4 são apresentados e discutidos os resultados. A Seção 5 apresenta trabalhos relacionados e na Seção 6 é apresentada a conclusão.

2. METODOLOGIAS PARA GERAÇÃO DE TUTORIAIS

Neste trabalho foram desenvolvidas quatro metodologias diferentes para gerar tutoriais para uma dada API. Para cada metodologia foi dado um nome e uma sigla conforme suas principais características. A Tabela 1 mostra as principais características de cada metodologia com suas respectivas siglas.

Tabela 1: Principais características de filtragem das metodologias e suas siglas.

Sigla	Características
FHCBAC	**F**iltros + ***H**ow to* + **C**omplexidade + Divisão **B**ásico e **A**vançado + **C**obertura API.
GP	**G**oogle **P**uro.
GFC	**G**oogle + **F**iltros + **C**omplexidade.
GFHCBAC	**G**oogle + **F**iltros + ***H**ow to* + **C**omplexidade + Divisão **B**ásico e **A**vançado + **C**obertura API.

As metodologias propostas implementaram diferentes características para permitir a comparação do benefício entre incluir ou não a característica. Por exemplo, na metodologia GP não foram implementados os filtros propostos e nem o algoritmo de ranqueamento por complexidade para avaliar a falta que estes mecanismos podem fazer.

Para cada metodologia foi desenvolvida uma sequência de etapas para construir um tutorial para uma dada API, a partir do banco de dados disponível do Stack Overflow.

Na Metodologia FHCBAC são aplicados os filtros desenvolvidos neste trabalho e outros algoritmos tais como: classificador de textos *Mallet* para filtrar os posts do tipo *"How-to-do"*, LDA (*Latent Dirichlet Allocation*) - modelagem de tópicos para gerar os capítulos do tutorial e LambdaMart para ranquear os *posts* conforme sua complexidade.

Na Metodologia GP utiliza-se o LDA para definir os capítulos do tutorial para uma certa API, depois é utilizado o mecanismo de pesquisa do Google com as palavras chaves de cada capítulo, e os n primeiros resultados ranqueados pelo Google são utilizados para montar cada capítulo do tutorial.

A Metodologia GFC é híbrida, pois combina etapas da Metodologia FHCBAC com etapas da Metodologia GP. Nesta metodologia é utilizado o LDA para definir os capítulos do tutorial para uma certa API, depois é utilizado o mecanismo de pesquisa do Google com as palavras chaves de cada capítulo (etapas da Metodologia GP), em seguida são aplicados os filtros e algoritmos desenvolvidos na Metodologia FHCBAC, e finalmente é montado o tutorial.

A Metodologia GFHCBAC também é híbrida, pois combina etapas da Metodologia FHCBAC com etapas da Metodologia GP. As diferenças são as seguintes: a primeira etapa consiste em gerar o tutorial para uma certa API utilizando a Metodologia FHCBAC; aplica-se o mecanismo de pesquisa do Google com as palavras chaves de cada capítulo combinadas com a palavra *"how to create"* filtrando apenas os resultados do site *"stackoverflow.com"*, a fim de encontrar os *posts* relacionados à "criação" de objetos, por exemplo, em um capítulo relacionado à tabelas um *post* de criação interessante seria *"How to instantiate an empty table?"*. Em seguida são aplicados os filtros e os algoritmos desenvolvidos na Metodologia FHCBAC nos *posts* retornados pelo Google, e depois eles são incluídos no tutorial.

A Tabela 2 mostra as principais características das metodologias desenvolvidas neste trabalho.

2.1 Metodologia FHCBAC

A seguir serão detalhadas as principais etapas da Metodologia FHCBAC para a geração de um tutorial para uma dada API "X".

Etapa 1: Filtro 1 - Seleção dos *posts* que possuem a *tag* da API "X" e que possuem resposta aceita pelo o usuário no banco de dados do Stack Overflow.

Etapa 2: Filtro 2 - Aplicação do classificador Mallet nos *posts* filtrados na Etapa 1 a fim de selecionar apenas os *posts* do tipo *"How-to-do"*. O objetivo desta etapa é identificar entre os *posts* previamente selecionados aqueles que possuem um tipo especial de questão (*"How-to-do"*), pois apenas este tipo de pergunta é adequado para a construção de um tutorial, uma vez que está relacionada em como implementar funcionalidades no software utilizando uma tecnologia ou uma API.

Existem outros tipos de questões propostas em [12] que não estão relacionadas com tutoriais e que não deveriam ser consideradas, como: *debug-correção, preciso-saber e buscando-solução-diferente*.

O classificador do Mallet foi treinado com um conjunto de dados de treinamento com 100 exemplos aleatórios de *posts "How-to-do"* e 100 exemplos aleatórios envolvendo *posts "debug-correção", "preciso-saber" e "buscando-solução-diferente"*, con-

Tabela 2: Principais características das metodologias.

Características	FHCBAC	GP	GFC	GFHCBAC
Utiliza o LDA para definir os capítulos do tutorial	X	X	X	X
Classificador *"How-to"*	X			X
Filtro *"How-to"* do Google		X	X	X
O código-fonte da resposta do *post* deve ter pelo menos três linhas	X		X	X
O código-fonte da pergunta do *post* não pode ter mais do que três linhas	X		X	X
Ranqueamento por complexidade	X		X	X
Divisão dos capítulos em básicos e avançados	X			X
Os *posts* cobrem pelo menos um tipo da API	X			X
Utiliza o mecanismo de pesquisa do Google		X	X	X
Seleciona *posts* de criação				X

forme a classificação proposta por Nasehi et al. [12]. Os *posts "How-to-do"* foram designados como grupo "Tutorial" e os demais tipos de *posts* foram designados como grupo "Não Tutorial". Para medir a acurácia do classificador após treinamento e teste, foi dado mais 100 exemplos aleatórios de *posts* do tipo *"How-to-do"* e mais 100 exemplos aleatórios envolvendo *posts "debug-correção", "preciso-saber" e "buscando-solução-diferente"*, estes exemplos são diferentes dos exemplos usados na fase de treinamento e teste. Dentre os 200 exemplos, 187 foram classificados corretamente pelo classificador, obtendo uma acurácia de 93,5%.

Depois de treinado, o classificador é aplicado em todos os *posts* de uma determinada API. Sendo assim, espera-se que os posts classificados como "Tutorial" sejam do tipo *"How-to-do"* e portanto com uma natureza apropriada para participar de um tutorial.

Etapa 3: Filtro 3 - Seleção dos *posts* com as seguintes características:

1. O código-fonte da resposta deve cobrir pelo menos um tipo da API (Classes, Interfaces, etc.). Considera-se que o código da resposta deve ilustrar como usar algum elemento da API.
2. No corpo da pergunta deve ter no máximo três linhas de código-fonte. Normalmente, perguntas com código extenso, estão solicitando ajuda acerca de algum problema no respectivo código.
3. No código da resposta deve ter no mínimo três linhas de código-fonte. Considera-se que três linhas seja um tamanho mínimo para que uma ilustração de uso da API seja significativa.

Etapa 4: Aplicação do algoritmo de ranqueamento LambdaMART nos *posts*. Nesta etapa, coleta-se um conjunto de atributos em cada *post*. Foram definidos os seguintes atributos:

1. Quantidade de **if**s no código-fonte do post;
2. Quantidade de **for**s no código-fonte do post;
3. Quantidade de **while**s no código-fonte do post;
4. Quantidade de **switch**s no código-fonte do post;
5. Quantidade de linhas de código-fonte;
6. Pontuação do *post* no SO.

Depois de coletados os atributos dos *posts*, define-se um conjunto de dados de treinamento que será utilizado pelo algoritmo de ranqueamento. Este conjunto será formado por *posts* que possuem diferentes níveis de complexidade, conforme os valores de seus atributos. A seleção do conjunto de *posts* de treinamento é feita de maneira manual, e em seguida é gerada uma matriz onde as linhas serão formadas pelos *posts* selecionados e as colunas serão formadas pelos valores de seus atributos. Em seguida, os *posts* são ranqueados de forma manual, atribuindo um valor para cada *post* conforme os valores dos seus atributos, por exemplo, um *post* X em que há várias estruturas de repetição é mais complexo do que um *post* Y que não tem estruturas de repetição, desta forma o valor atribuído ao *post* X é maior do que o atribuído a Y. O atributo *Pontuação do post no SO* é considerado para fins de seleção de *posts* de qualidade, este atributo não influencia na complexidade do *post*. Uma vez treinado o ranqueador, o próximo passo é ranquear todos *posts* filtrados na etapa anterior.

Etapa 5: Divisão dos *posts* ranqueados em básicos e avançados. Seja n o número de *posts* ranqueados. Define-se que os *posts* básicos são formados pelos os *posts* ranqueados da posição 1 até a posição $n/2$, e os *posts* avançados são formados pelos os *posts* ranqueados da posição $(n/2+1)$ até a posição n.

Etapa 6: Execução do algoritmo de modelagem de tópicos (LDA) na documentação oficial da API "X" para gerar os tópicos que representarão os capítulos do tutorial. É necessário definir um número de tópicos onde não haja muita repetição de assuntos nos tópicos e que abordem o máximo de assuntos da API. Para isto foi proposto o Algoritmo 1.

Depois de gerados os tópicos pelo o LDA, analisa-se a necessidade de clusterizar aqueles que ainda abordam o mesmo assunto, logo em seguida, o(s) tópico(s) e/ou cluster(s) são analisados para receber um nome. Cada tópico(s) e/ou cluster(s) representará um capítulo no tutorial.

Etapa 7: Constrói os capítulos da parte básica do tutorial, conforme o Algoritmo 2.

Etapa 8: Construção dos capítulos da parte avançada do tutorial. Nesta etapa é usado o mesmo algoritmo da Etapa 7, a única diferença é que ao invés de o algoritmo executar com os *posts* básicos, ele executa com os *posts* avançados obtidos na Etapa 5 da metodologia.

Etapa 9: Montagem final do tutorial. Este algoritmo recebe como entrada os capítulos básicos e avançados retornados pelas Etapas 7 e 8. Para cada capítulo do tutorial o algoritmo cria um sumário geral contendo *links* para os outros capítulos. O algoritmo também cria para cada capítulo um sumário contendo *links* para todos os seus respectivos exemplos (*posts* selecionados na Etapa 7 ou 8). Estes sumários facilitam a navegabilidade pelo tutorial. No final desta etapa o algoritmo retorna o tutorial montado em um arquivo

Algoritmo 1 Algoritmo para encontrar um número adequado de tópicos para o LDA

```
1: numeroTopicos ← leiaNumeroTopicos();
2: executarLDA ← "sim";
3: aumentarNumeroTopicos ← "sim";
4: while executarLDA = "sim" do
5:    executarLDA ← "nao";
6:    topicos ← executaLDA(numeroTopicos);
7:    termosComunsParesTops ← calculeQuantTermosComunsPares(topicos);
8:    for i ← 1 to tamanho(termosComunsParesTops); do
9:       if termosComunsParesTops[i] ≥ 3 then
10:         numeroTopicos ← numeroTopicos − 1;
11:         executarLDA ← "sim";
12:         aumentarNumeroTopicos ← "nao";
13:      end if
14:      if aumentarNumeroTopicos = "sim" then
15:         numeroTopicos ← numeroTopicos + 10;
16:         executarLDA ← "sim";
17:      end if
18:   end for
19: end while
20: return numeroTopicos;
```

HTML, depois basta hospedar este arquivo em um servidor de hospedagem de página WEB. O usuário final acessará o tutorial online através de qualquer dispositivo que tenha acesso à internet e um navegador.

Algoritmo 2 Algoritmo para formar capítulos Básicos do Tutorial

```
1: for i ← 1 to postBasico.tamanho(); do
2:    postBasico[i].selecionado ← "nao";
3: end for
4: for i ← 1 to capitulos.tamanho(); do
5:    tiposAPI ← tiposAPICaps(capitulos[i]);
6:    for j ← 1 to postBasico.tamanho(); do
7:       for w ← 1 to tiposAPI.tamanho(); do
8:          if postBasico[j].contemTipo(tiposAPI[w]) then
9:             tiposAPI.removeTipo(tiposAPI[w]);
10:            if postBasico[j].selecionado = "nao" then
11:               capTutorial[i].incluiPost(postBasico[j]);
12:               postBasico[j].selecionado = "sim";
13:            end if
14:         end if
15:      end for
16:   end for
17: end for
```

2.2 Metodologia GP

A seguir serão apresentadas as principais etapas da Metodologia GP para a geração de um tutorial para uma dada API "X".

Etapa 1: Definição dos tópicos que organizarão o tutorial usando o algoritmo LDA. Nesta etapa é usado o mesmo algoritmo da Etapa 6 da Metodologia FHCBAC para definir um número balanceado de tópicos para o LDA.

Etapa 2: Definição dos capítulos do tutorial usando os tópicos definidos na etapa anterior. Nesta etapa é feita a mesma análise da Etapa 6 da Metodologia FHCBAC para clusterizar os tópicos que abordam o mesmo assunto da API, e em seguida, os tópico(s) e/ou cluster(s) são analisados para receber um nome. Cada tópico(s) e/ou cluster(s) representará um capítulo no tutorial.

Etapa 3: Utilização do mecanismo de pesquisa do Google, tendo como entrada as palavras extraídas dos tópico(s) e/ou cluster(s) gerados na Etapa 2 e palavras chaves como: o nome da API "X", *"how to"*, filtrando somente para o *site* stackoverflow.com. No final desta etapa, para cada capítulo do tutorial, o Google retornará a pesquisa com os *posts* do Stack Overflow ranqueados.

Etapa 4: Seleção dos n primeiros resultados ranqueados pelo mecanismo de pesquisa do Google, para cada capítulo.

Etapa 5: Similar à Etapa 9 da Metodologia FHCBAC, com exceção que não há organização do tutorial em partes básica e avançada. Os *posts* utilizados para a montagem dos capítulos são aqueles obtidos na Etapa 4.

2.3 Metodologia GFC

A seguir serão apresentadas as principais etapas da Metodologia GFC para a geração de um tutorial para uma dada API "X". Esta metodologia é híbrida, pois combina algumas etapas da Metodologia FHCBAC com algumas etapas da Metodologia GP.

Etapa 1: Executa o LDA na documentação oficial da API "X". Nesta etapa é usado o mesmo algoritmo da Etapa 6 da Metodologia FHCBAC para encontrar um número de tópicos para o LDA.

Etapa 2: Usa os tópicos gerados pelo LDA na Etapa 1 para definir os capítulos do tutorial. Nesta etapa é feita a mesma análise da Etapa 6 da Metodologia FHCBAC para clusterizar os tópicos que abordam o mesmo assunto da API, e em seguida, os tópico(s) e/ou cluster(s) são analisados para receber um nome. Cada tópico(s) e/ou cluster(s) representará um capítulo no tutorial.

Etapa 3: Utiliza o mecanismo de pesquisa do Google, dando como entrada as palavras extraídas dos tópico(s) e/ou cluster(s) gerados na Etapa 2 e palavras chaves como: o nome da API "X", *"how to"* filtrando para o *site* stackoverflow.com. No final desta etapa, para cada capítulo do tutorial, o Google retornará como resultado da pesquisa os *posts* do Stack Overflow ranqueados.

Etapa 4: Aplica o filtro desenvolvido neste trabalho para pegar os *posts* com as características:

1. No corpo da pergunta deve ter no máximo três linhas de código-fonte.
2. No código da resposta deve ter no mínimo três linhas de código-fonte.

Etapa 5: Ranqueia os *posts* através do algoritmo de ranqueamento por complexidade desenvolvido na Etapa 4 da Metodologia FHCBAC deste trabalho. Isto é feito para cada capítulo do tutorial.

Etapa 6: Seleciona os n primeiros resultados ranqueados na Etapa 5. Isto é conduzido para cada capítulo do tutorial.

Etapa 7: Similar à Etapa 5 da Metodologia GP.

2.4 Metodologia GFHCBAC

A seguir serão apresentadas as principais etapas da Metodologia GFHCBAC para a geração de um tutorial para uma dada API "X". Esta metodologia é híbrida, pois combina algumas etapas da Metodologia FHCBAC com algumas etapas da Metodologia GP.

Etapa 1: Gera o tutorial da API "X" utilizando a Metodologia FHCBAC.

Etapa 2: Utiliza o mecanismo de pesquisa do Google, dando como entrada as palavras chave extraídas dos capítulos do tutorial gerado na Etapa 1 e palavras chaves como: o nome da API "X", *"how to create"* e "Stack Overflow". No final desta etapa o Google retornará a pesquisa com os *posts* do Stack Overflow ranqueados, para cada capítulo.

Etapa 3: Aplica o filtro desenvolvido neste trabalho para pegar os *posts* com as características:

1. No corpo da pergunta deve ter no máximo três linhas de código-fonte.
2. No código da resposta deve ter no mínimo três linhas de código-fonte.

Etapa 4: Seleciona os n primeiros *posts* de criação retornados pelo Google e os inclui no tutorial gerado na Etapa 1. Isto é feito para cada capítulo do tutorial.

Etapa 5: Executa o algoritmo desenvolvido no trabalho para montar o tutorial. Este algoritmo recebe como entrada o tutorial gerado pela Etapa 1 e os *posts* de criação de cada capítulo selecionados na Etapa 4. Os *posts* de criação são inseridos em seus respectivos capítulos no tutorial, e os seus *links* também são inseridos no sumário do capítulo. O restante desta etapa é idêntico ao das demais abordagens.

Cabe ressaltar que os esforços para a geração dos tutoriais utilizando as metodologias propostas são mínimos, praticamente toda a abordagem é automática, a única etapa manual é a da escolha dos nomes dos capítulos dos tutoriais analisando visualmente os termos dos tópicos gerados pelo LDA. Apesar de esta abordagem poder ser automatizada, preferimos não o fazer devido ao baixo esforço manual e a alta capacidade do ser humano escolher um bom título dado um conjunto de termos que define o tópico.

3. AVALIAÇÃO DAS METODOLOGIAS

Foram utilizadas as quatro metodologias apresentadas anteriormente para gerar tutoriais para a API Swing. A escolha da API Swing tem vários pontos favoráveis para a avaliação das metodologias, apesar de a tecnologia desktop ter perdido espaço para tecnologias móveis e web, a popularidade do Swing entre novatos ainda é alta, é uma API conhecida que permite avaliar qualitativamente a qualidade do tutorial gerado, além de ser uma API bastante discutida no Stack Overflow. A escolha da API tem pouco impacto sob o ponto de vista de comparação entre as abordagens. As metodologias propostas neste trabalho foram desenhadas para gerar tutoriais não só para APIs, mas também para bibliotecas, linguagens de programação, ou qualquer tecnologia que contenha código-fonte como exemplos.

Cada tutorial recebeu um nome conforme as metodologias que o geram, FHCBAC: "Blue Version", GP: "Red Version", GFC: "Yellow Version" e GFHCBAC: "Green Version". Na Figura 1 é apresentada a tela inicial para o tutorial "Green Version". Na lateral esquerda, são apresentados os *links* de navegação para os capítulos, divididos em parte básica e avançada. Na parte superior central são apresentados os *links* para os elementos do Capítulo 1 (*Table*). Na parte inferior, já é apresentado o primeiro elemento do Capítulo 1, e os próximos elementos do capítulo também podem ser acessados por meio de rolagem. As quatro versões do tutorial da API Swing podem ser acessadas online[1].

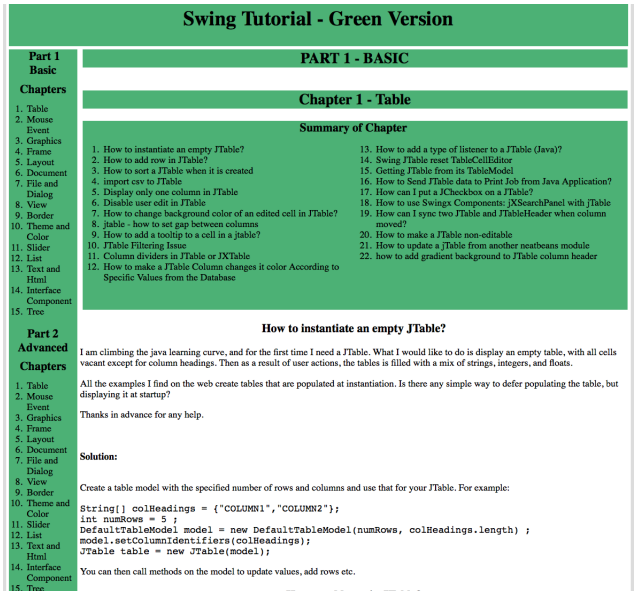

Figura 1: Visão Inicial do Tutorial Swing gerado por GFHCBAC

Para avaliar estes tutoriais foi desenvolvido um formulário online através do software Lime Survey. O *link* do formulário foi submetido a uma lista de e-mail do Programa de Pós-graduação em Ciência da Computação da UFU. O grupo possui cerca de 200 discentes. O *link* também foi enviado para 6 profissionais que trabalham na área de desenvolvimento de software. Ao todo 12 respondentes completaram a avaliação dos tutoriais.

Nas afirmações do questionário para avaliar os tutoriais foi utilizada a escala Likert em 5 níveis para medir o grau de concordância de cada afirmação que varia do extremo *Concordo fortemente* ao *Discordo fortemente*. Os participantes avaliaram os tutoriais conforme os seguintes critérios:

1. **Organização**. A afirmação solicitada para os participantes avaliarem foi: *O tutorial é organizado.* Os elementos adicionais indicados para o avaliador utilizar ao definir o *score* desta afirmação foram:
   - O sumário do tutorial está numerado.
   - Os sumários dos capítulos possuem os títulos de todos os exemplos encontrados em um determinado capítulo.
   - Os exemplos encontrados no tutorial estão organizados em pergunta seguida da solução.
2. **Intuitividade**. A afirmação solicitada para os participantes avaliarem foi: *O tutorial é intuitivo.* Os elementos adicionais indicados foram:
   - O tutorial pode ser utilizado sem nenhum tipo de treinamento.
   - Sua utilização é autoexplicativa.

[1]http://lascam.facom.ufu.br/APITutorial/SwingTutorial/

3. **Navegabilidade**. A afirmação solicitada para os participantes avaliarem foi: *O tutorial possui uma navegabilidade adequada nos sumários*. Os elementos adicionais indicados foram:
    - O sumário do tutorial possui *links* para todos os capítulos.
    - Em cada capítulo, o sumário geral pode ser acessado, possibilitando a navegação para outros capítulos.
    - Cada capítulo possui seu próprio sumário com *links* para os exemplos do respectivo capítulo.
4. **Natureza didática do código-fonte**. A afirmação solicitada para os participantes avaliarem foi: *Os exemplos do tutorial possuem códigos-fonte didáticos.*
5. **Ordem de complexidade dos elementos do tutorial.** A afirmação solicitada para os participantes avaliarem foi: *O tutorial segue uma ordem crescente de complexidade dos exemplos dos capítulos, ou seja, os exemplos mais simples aparecem antes dos exemplos mais complexos.*
6. **Partes básica e avançada**. A afirmação solicitada para os participantes avaliarem foi: *A organização do tutorial com divisão em partes básica e avançada é adequada.*
7. **Natureza de tutorial**. A afirmação solicitada para os participantes avaliarem foi: *A natureza dos exemplos é compatível com um tutorial. Em outras palavras, os exemplos explicam como utilizar funcionalidades da API, e não, por exemplo, como corrigir erros de programação.*
8. **Coesão interna dos capítulos**. A afirmação solicitada para os participantes avaliarem foi: *Os exemplos encontrados nos capítulos estão relacionados com o respectivo nome do capítulo.*

Para cada item de avaliação, os participantes foram instruídos a avaliar comparativamente as quatro versões do tutorial, sendo que, se a versão X for melhor do que a versão Y atribuir um *score* maior para X em relação à Y.

A escolha do teste estatístico para análise das diferenças entre as avaliações levou em consideração uma série de características dos dados: 1) os resultados do *survey* são valores ordinais da escala Likert, 2) a amostra analisada não é grande e os resultados não permitem assumir normalidade dos dados, 3) cada avaliador avaliou os 4 tutoriais, o que caracteriza um experimento em blocos (avaliadores). Sendo assim, adotou-se o Teste de Friedman com análise post-hoc por ser um teste não-paramétrico que não requer suposição de normalidade da população e nem de tamanho amostral elevado. Além disso, suporta dados ordinais em bloco. Foi utilizada uma função utilitária baseada no pacote *coin* disponível na CRAN (*Comprehensive R Archive Network*)[6].

4. RESULTADOS E DISCUSSÃO

Esta seção apresenta a avaliação dos tutoriais gerados pelas metodologias desenvolvidas neste trabalho. A Figura 2 apresenta os *boxplots* com os resultados de avaliação para cada critério. Para tornar a avaliação mais intuitiva foram colocados os pontos de avaliação com *jittering* em relação ao respectivo *score*, para evitar que todos os pontos ficassem na mesma posição, inviabilizando a respectiva apresentação.

A Tabela 3 mostra os resultados de comparação de diferença entre os *scores* das respectivas metodologias para cada critério. O resultado dos *p-values* menores que 0.05 para o teste de Friedman indica diferença estatisticamente significante entre as metodologias. Para identificar quais metodologias são responsáveis pelas diferenças, a Tabela 4 apresenta os resultados da análise post-hoc para o teste de Friedman.

Tabela 3: Teste de Friedman para os *scores* de cada critério agrupados por metodologia com bloco por avaliadores

Critério	p-value
Básica-Avançada	0.002
Código didático	0.049
Coesão do capítulo	0.108
Complexidade	0.004
Intuitividade	0.472
Natureza de tutorial	0.008
Navegabilidade	0.146
Organização	0.437

Tabela 4: Comparação entre metodologias por análise post-hoc para o Teste de Friedman

Par de Metodologias	Complex.	Bas.Av.	Natur.	C.Did.
GFC - FHCBAC	0.936	**0.049**	0.999	0.928
GFHCBAC - FHCBAC	0.861	0.733	0.891	0.983
GP - FHCBAC	*0.052*	**0.049**	*0.060*	0.122
GFHCBAC - GFC	0.516	**0.001**	0.839	0.761
GP - GFC	0.203	1.000	0.081	0.383
GP - GFHCBAC	**0.004**	**0.002**	**0.007**	**0.049**

Organização. Como se observa na Figura 2, os tutoriais gerados pelas quatro metodologias receberam um alto *score* para a organização, isto porque os quatro tutoriais estão organizados de forma semelhante. Como esperado, o Teste de Friedman não apontou diferença entre as metodologias. A razão para os altos *scores* deve-se aos sumários estarem numerados e possuírem os títulos de todos os exemplos encontrados em um determinado capítulo, além dos exemplos estarem organizados em pergunta seguida da solução.

Intuitividade. Este item de avaliação também apresentou altos *scores* para os quatro tutoriais, isto porque eles são autoexplicativos, o *layout* é bem simples e conta com *links* para os outros capítulos na tela inicial, conforme o usuário vai navegando pelos tutoriais, ele vai encontrando *links* que o leva direto para os exemplos oferecidos pelos tutoriais. Cabe ressaltar que houve avaliador que não concordou com a intuitividade e vários que se posicionaram de maneira neutra.

Navegabilidade. Este item também apresentou altos *scores* para os quatro tutoriais, indicando que o fato deles possuírem dois tipos de sumário, um sumário geral que pode ser utilizado para acessar qualquer um dos capítulos, e um sumário local em cada capítulo que pode ser utilizado para acessar os exemplos do respectivo capítulo mostrou-se uma navegação adequada.

Complexidade. Dos quatro tutoriais gerados pelas metodologias, para este item de avaliação, a GFHCBAC foi a que obteve *scores* mais altos, podendo ser observado pela mediana na Figura 2, além de ser estatisticamente significativa a diferença em relação à GP. A explicação para o melhor desempenho da GFHCBAC é que esta metodologia possui uma etapa de ranqueamento dos exemplos, além de

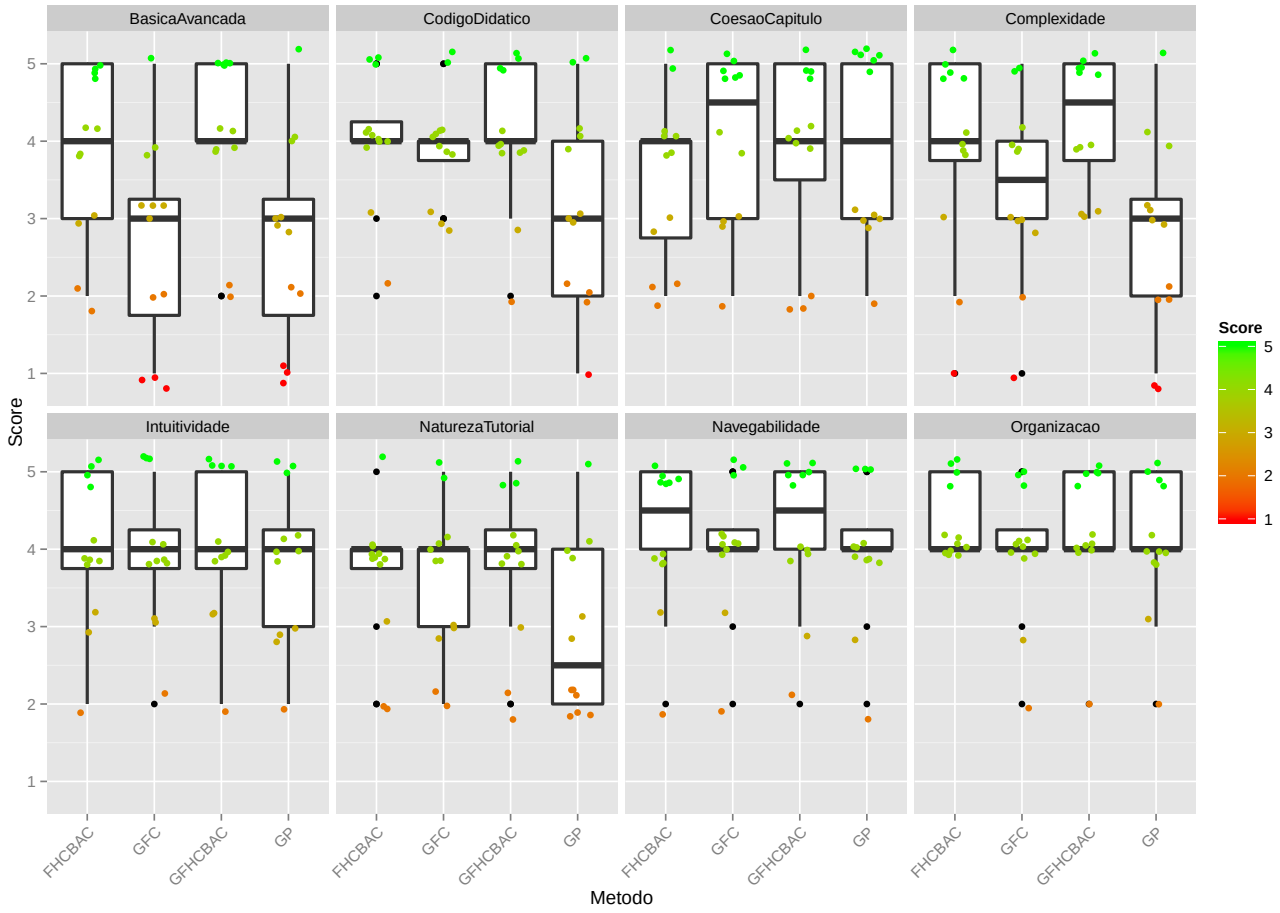

Figura 2: Avaliação dos critérios de tutoriais. 5: Concordo Fortemente ... 1: Discordo Fortemente

possuir exemplos simples de criação no começo de cada capítulo da parte básica do tutorial. A FHCBAC, a segunda com maior mediana, também possui uma etapa de ranqueamento dos exemplos. A diferença da avaliação em comparação com a GFHCBAC pode estar pelo fato da FHCBAC não gerar exemplos de criação no começo de cada capítulo básico do tutorial. O terceiro tutorial com maior mediana foi o gerado pela Metodologia GFC que utiliza o mecanismo de pesquisa do Google para ranquear os n exemplos do tutorial, em uma etapa posterior também é aplicado o algoritmo de ranqueamento por complexidade. O tutorial gerado pela Metodologia GP foi o que teve a menor mediana, podendo ser explicado por esta metodologia somente utilizar o mecanismo de pesquisa do Google para ranquear os exemplos do tutorial. Como este mecanismo não leva em consideração a complexidade dos exemplos, então não se pode esperar que estes exemplos estejam ranqueados por ordem de complexidade, podendo acontecer de um exemplo muito complexo aparecer antes de um exemplo muito simples, uma vez que o Google utiliza outras características para o ranqueamento.

Partes básica e avançada. Os tutoriais gerados pelas Metodologias GP e GFC apresentaram menores medianas entre os quatro, porque estas metodologias não dividem os capítulos dos tutoriais em partes básica e avançada. Mesmo que estas metodologias não implementam esta característica, ela foi considerada para efeitos de controle da avaliação, ou seja, para verificar se os avaliadores foram consistentes na avaliação dos tutoriais. A maioria dos avaliadores deu nota baixa para este quesito ou ficaram neutros para estas metodologias. Já as Metodologias FHCBAC e GFHCBAC dividem os capítulos em partes básica e avançada, explicando os melhores *scores*. Uma diferença estatisticamente significativa entre os grupos GP/GFC em relação aos grupos GFHCBAC/FHCBAC mostra que tal separação se mostrou efetiva. No entanto, alguns avaliadores discordaram, uma possível explicação seria por se tratar de um tutorial, eles acharam que não é adequado os tutoriais terem a parte avançada.

Natureza de tutorial. Os tutoriais gerados pelas Metodologias GFHCBAC e FHCBAC foram os que apresentaram maiores medianas, tendo o GFHCBAC sido estatisticamente maior que GP e FHCBAC com uma diferença *borderline* em relação a GP. Inclusive a GFC obteve uma diferença com p-value de 0.08 em relação à GP. Isto indica a adequação dos filtros aplicados em GFHCBAC, FHCBAC e GFC. Por exemplo, o filtro para selecionar apenas os exemplos do tipo *"How-to-do"*, o filtro que seleciona os exemplos com as características: *"no corpo da pergunta do exemplo deve ter no máximo três linhas de código-fonte"*, *"no código da resposta do exemplo deve ter no mínimo três linhas de código-fonte"*

e *"o código-fonte da resposta deve cobrir pelo menos um tipo da API(Classes, Interfaces, etc.)."*. Além de estas metodologias utilizarem o ranqueador por complexidade, que ranqueia os exemplos conforme sua complexidade, ou seja, os *posts* mais simples aparecem primeiro do que os *posts* mais complexos. Estas metodologias também dividem o tutorial em partes básica e avançada. A GFHCBAC teve um nível de concordância um pouco maior do que a FHCBAC, que pode ser explicado devido à GFHCBAC ser uma extensão da versão FHCBAC, com exemplos de criação no começo de cada capítulo. O tutorial gerado pela Metodologia GP obteve a menor mediana nos *scores*. Esta metodologia só utiliza o mecanismo de pesquisa do Google para ranquear os exemplos do tutorial, e como este mecanismo não leva em consideração a complexidade do conteúdo do exemplo, então os exemplos complexos estão misturados com os simples, sem nenhum ranqueamento por complexidade, o que indica uma deficiência na caracterização como tutorial. Além disso, não é aplicado nenhum dos filtros utilizados nas Metodologias GFHCBAC e FHCBAC. A GFC teve um nível de concordância intermediário entre as FHCBAC e GP, isto porque a GFC implementa alguns dos filtros utilizados nas Metodologias FHCBAC e GFHCBAC.

Código-fonte didático. Os tutoriais gerados pelas Metodologias GFHCBAC, FHCBAC e GFC, tiveram uma maior mediana em relação a GP. Entretanto, apenas GFHCBAC e GP se mostraram estatisticamente diferentes. Isto pode ser explicado pelo fato das metodologias que geram estes tutoriais implementarem filtros que tentam selecionar exemplos do tipo "Tutorial", conforme os critérios de cada filtro. A análise da Figura 2 indica que exemplos do tipo "Tutorial" são mais didáticos. O tutorial gerado pela Metodologia GP teve a mediana neutra, possivelmente por esta metodologia não implementar os filtros para selecionar exemplos do tipo "Tutorial", apenas utiliza o mecanismo de pesquisa do Google. Mesmo que este mecanismo não leve em consideração a complexidade dos exemplos, ou não tente identificar os exemplos do tipo "Tutorial", este mecanismo ainda retorna exemplos que ajudam os usuários a resolverem os seus problemas, mesmo que não seja da maneira mais didática possível.

Coesão interna do capítulo. Apesar de não ter havido diferença estatisticamente significativa entre os diversos tutoriais em relação à coesão dos capítulos, o tutorial gerado pela Metodologia GFC teve a maior mediana entre os quatro tutoriais. Esta metodologia utiliza o mecanismo de pesquisa do Google para buscar os exemplos do tutorial, sendo este mecanismo muito eficiente em buscar exemplos relacionados com o capítulo, uma vez que, para formular o texto da pesquisa, são considerados os termos retornados pelo LDA para caracterização do capítulo. O tutorial gerado pela Metodologia GP também teve uma mediana alta, porém muitos avaliadores ficaram neutros em relação a este tutorial, onde sua metodologia também utiliza o mecanismo do Google para buscar os seus exemplos. Os tutoriais gerados pelas Metodologias FHCBAC e GFHCBAC também tiveram uma alta mediana, porém houve algumas avaliações negativas. Isto pode ser explicado devido a estas metodologias buscarem os exemplos para o tutorial considerando o conteúdo do código-fonte destes exemplos. Por exemplo, pode acontecer de um exemplo do tutorial que está relacionado ao capítulo "Tabela" estar em um capítulo relacionado à "Botão", se este exemplo tiver botões dentro de uma tabela.

Ameaças à validade. Uma ameaça à validade externa foi a análise de somente a API Java Swing. As metodologias podem ter resultados diferentes com outras APIs de Java ou de outras linguagens. Outra ameaça à validade externa é a amostra de avaliadores que participaram do estudo. Avaliadores de outras regiões ou países podem ser mais ou menos rigorosos na avaliação dos tutoriais. Em relação à validade interna, outros fatores que poderiam interferir em melhores resultados em geral para a abordagem GFHCBAC são o nível de experiência dos avaliadores na API Swing ou a possibilidade de divergência de interpretação das afirmações. Para esta última, procurou-se mitigar por meio de afirmações simples e com auxílio à interpretação em casos onde identificou-se maior possibilidade de ambiguidade.

5. TRABALHOS RELACIONADOS

Vários autores têm trabalhado na questão de como elaborar uma documentação de uma API de forma mais simples e eficaz. Storey et al. [21] mostraram a importância das mídias sociais (wikis, blogs, redes sociais, etc.) nas atividades da Engenharia de Software, que vão desde a engenharia de requisitos e desenvolvimento até testes e documentação. Os autores citam a importância do SO como sendo um site de troca de informações e gerenciamento de trabalho colaborativo.

No trabalho de Treude et al. [24] foram discutidos as oportunidades e os desafios que os desenvolvedores de software têm ao fazerem uso de informações disponibilizadas por sites da internet, como por exemplo, o SO. Os autores mostraram que a maior parte das perguntas do SO, cerca de 90%, tem tempo médio de resposta de 11 minutos [10] (existem casos de usuários que recebem respostas em minutos, ou até mesmo em segundos após a sua postagem no site). O trabalho também mostrou que o SO possui mais de 12 milhões de visitantes e 135 milhões de visualizações de páginas todo mês (dados obtidos em 2012 pelos autores). Isto faz com que o SO seja uma fonte importante de informações que os desenvolvedores podem estar utilizando no processo de desenvolvimento ou manutenção de software.

O trabalho de Parnin et al. [14] mostra a importância das documentações de APIs feitas a partir de sites como o SO. Segundo os autores nas documentações oficiais das APIs, para uma certa funcionalidade há apenas um exemplo simples de código, sem um texto que explica melhor como utilizar a funcionalidade. Já em sites como o SO há centenas de tópicos relacionados à funcionalidade que se deseja aprender. No trabalho também foi feito um estudo para viabilizar o uso do SO para obter exemplos e explicações sobre APIs. Os resultados mostraram que 87% das classes da API do Android são referenciadas no SO, o que motiva a construção de documentações a partir das informações contidas neste site.

Robillard [17] fez um estudo onde analisou as principais dificuldades que desenvolvedores de software têm ao aprender novas APIs. O estudo foi feito com 80 profissionais de software da Microsoft através de preenchimento de questionários e realização de entrevistas. Entre estes profissionais, 78% disseram aprender sobre APIs lendo a documentação oficial, 55% através do uso de exemplos de código, 34% através de experimentação com a API, 30% através da leitura de artigos e 29% através de consulta a colegas de trabalho. Segundo o autor, os principais obstáculos na aprendizagem da API são os recursos disponíveis, como por exemplo, a do-

cumentação. Isto implica em esforços para tentar melhorar estes recursos.

Souza et al. [20] propuseram uma abordagem semiautomática para construir livros de receitas para APIs através de informações disponíveis no SO. Esta abordagem utiliza a técnica de recuperação de informação LDA para encontrar temas de uma API, e através destes são identificados os potenciais capítulos do livro de receitas. Uma limitação desta abordagem é que não há uma separação entre capítulos mais complexos e capítulos mais básicos, de caráter introdutório.

Treude e Robillard [25] apresentaram uma abordagem para aumentar automaticamente a documentação de um tipo particular de API com informações do Stack Overflow. No trabalho os autores apresentam SISE, uma nova abordagem baseada em aprendizagem de máquina que usa como características sentenças: sua formatação, sua pergunta, sua resposta, seus autores, *tags* e semelhança de uma frase que corresponde à documentação da API. Com o SISE, os autores alcançaram uma precisão de 0,64 e uma cobertura de 0,7 no conjunto de desenvolvimento. Em um estudo comparativo com oito desenvolvedores de software, eles relataram que o SISE resultou no maior número de frases que foram consideradas úteis para adicionar informações não encontradas na documentação da API.

Montandon et al. [11] desenvolveram o APIMiner, que tem o objetivo de aumentar a documentação de APIs para Java com exemplos concretos de uso. Uma versão do APIMiner para a API Android possibilitou a extração de 73.732 exemplos de código-fonte a partir de 103 projetos *open-source*. Neste trabalho o objetivo foi melhorar a documentação já existente, e apenas para a linguagem Java.

Henb et al. [5] propuseram uma técnica semiautomática para construir um FAQs (*Frequently Asked Questions*) a partir de dados disponíveis em sites de discussão sobre software e fóruns. Este trabalho também não organiza as FAQs por complexidade.

Kim et al. [7] apresentaram uma técnica para incrementar de forma automática a documentação de APIs com exemplos de código-fonte. Segundo os autores o conteúdo da maioria das documentações de APIs não possui código-fonte o suficiente. Os resultados mostraram que apenas 2% das classes (de mais de 27.000) de APIs dos Javadocs possuem exemplos de código. Isto faz com que desenvolvedores busquem na internet código-fonte que os auxiliam na aprendizagem das APIs. Os autores alertam sobre a quantidade de tempo gasta no processo de encontrar um conteúdo na internet que possa auxiliar os desenvolvedores no entendimento da funcionalidade desejada.

Long et al. [9] desenvolveram o Altair, uma ferramenta que gera automaticamente referências cruzadas de funções de APIs. A ferramenta realiza uma análise estática para extrair informações estruturais do código-fonte, e a partir desses dados calcula a sobreposição de pares.

Dagenais e Robillard [3] propuseram AdDoc, uma técnica que detecta automaticamente os padrões de documentação, isto é, conjuntos coerentes de elementos de código que estão documentados juntos.

Kim et al. [8] propuseram um sistema de recomendação de exemplo de código que retorna documentos para API extraídos na Web. Os resultados da avaliação mostram que a abordagem fornece exemplos de código com alta precisão e aumenta a produtividade do programador.

Robillard e Chhetri [18] propuseram detectar e recomendar fragmentos de documentação de API potencialmente importantes para um programador que já decidiu usar um dado elemento de API. Os autores categorizaram fragmentos de texto na documentação de API baseando se eles contêm informações que são indispensáveis, valiosas, ou nenhuma das opções. A partir dos fragmentos que contêm um conhecimento, eles extraíram padrões de palavras, e usaram esses padrões para encontrar automaticamente novos fragmentos que contêm conhecimentos semelhantes em uma documentação não analisada.

Chen e Zhang [2] propuseram conectar a documentação oficial de API com a documentação informal através da captura do comportamento de desenvolvedores que navegam na Web.

Zhong e Su [27] introduziram DOCREF, uma abordagem que combina técnicas de processamento de linguagem natural e análise de código para detectar e reportar inconsistências na documentação. Os autores utilizaram com sucesso o DOCREF para detectar mais de mil erros de documentação.

Pandita et al. [13] propuseram inferir especificações formais de texto de linguagem natural de documentações de API.

Petrosyan et al. [15] propuseram uma abordagem para descobrir seções de tutorial que explicam um determinado tipo de API. Os autores classificaram fragmentados de seções de tutorial usando classificação de textos supervisionado, baseado em características linguísticas e estruturais. Eles foram capazes de alcançar alta precisão e recall em diferentes tutoriais.

6. CONCLUSÃO

Neste trabalho apresentamos quatro metodologias diferentes para a geração de tutoriais para uma dada API a partir do conteúdo do Stack Overflow. A principal contribuição em relação aos trabalhos relacionados é que este trabalho busca gerar uma documentação para uma dada API levando em consideração a complexidade de entendimento dos exemplos. Os resultados da avaliação dos tutoriais gerados pelas metodologias propostas neste trabalho mostraram a viabilidade para o uso de tutoriais gerados automaticamente.

Cada metodologia proposta implementou um grupo de características. Entre estas características estão filtros desenvolvidos neste trabalho, algoritmos de classificação, modelagem em tópicos e ranqueamento. As metodologias FHCBAC, GFC, GFHCBAC que utilizaram os filtros propostos tiveram resultados melhores do que a metodologia GP que utiliza apenas o mecanismo de pesquisa do Google. Este mecanismo não leva em consideração a complexidade dos exemplos relacionados ao desenvolvimento de software. Isto sugere a importância do desenvolvimento contínuo de filtros especializados para encontrar conteúdos relacionados ao desenvolvimento de software. Entre as quatro metodologias, a GFHCBAC foi a que teve os melhores resultados gerais. De certa forma isto é explicado pelo fato que ela implementa de alguma maneira as características das demais metodologias, ou seja, utiliza a metodologia FHCBAC para gerar um tutorial, depois utiliza o mecanismo de busca do Google para encontrar os exemplos de criação. Cabe ressaltar que esta metodologia não foi a mais eficiente em relação à coesão do capítulo, onde o uso nativo do Google obteve melhores resultados.

Os desenvolvedores, principalmente novatos, poderão utilizar os tutoriais gerados pelas metodologias propostas, prin-

cipalmente pela metodologia GFHCBAC que se mostrou ser a mais didática na avaliação, para aprender uma nova tecnologia (APIs, linguagens de programação, bibliotecas, etc.). Desenvolvedores experientes poderiam utilizar a parte avançada dos tutoriais para aprofundar seus conhecimentos. Geralmente as documentações oficiais carecem de exemplos de uso das partes da API. Não é incomum, as documentações oficiais não possuírem exemplos simples contendo um código-fonte didático.

Como trabalho futuro, sugere-se a criação de um portal de tutoriais de API que permita a avaliação em larga escala, tanto em diversidade de APIs quanto na diversidade de avaliadores. Outra frente de investigação é a análise do impacto que tais tutoriais podem trazer para os desenvolvedores iniciantes quando comparado com o uso dos recursos didáticos mais usuais. Espera-se que a abordagem tenha um impacto maior para APIs que carecem de recursos didáticos, mas que tenham comunidade ativa no Stack Overflow.

Agradecimento. Este trabalho foi financiado parcialmente pela FAPEMIG.

7. REFERÊNCIAS

[1] E. C. Campos and M. de Almeida Maia. Automatic Categorization of Questions from Q&A Sites. In *Proc. of the 29th Annual ACM Symposium on Applied Computing, SAC'2014*, pages 641–643. ACM, 2014.

[2] C. Chen and K. Zhang. Who asked what: integrating crowdsourced FAQs into API documentation. In *ICSE Companion*, pages 456–459. ACM, 2014.

[3] B. Dagenais and M. P. Robillard. Using Traceability Links to Recommend Adaptive Changes for Documentation Evolution. *IEEE Trans. Software Eng.*, 40(11):1126–1146, 2014.

[4] L. B. L. de Souza, E. C. Campos, and M. d. A. Maia. Ranking Crowd Knowledge to Assist Software Development. In *Proc. of the 22nd Intl. Conf. on Program Comprehension, ICPC'2014*, pages 72–82, New York, NY, USA, 2014. ACM.

[5] S. Henb, M. Monperrus, and M. Mezini. Semi-automatically Extracting FAQs to Improve Accessibility of Software Development Knowledge. In *Proc. of ICSE'2012*, pages 793–803, 2012.

[6] T. Hothorn, K. Hornik, M. A. van de Wiel, and A. Zeileis. Implementing a Class of Permutation Tests: The coin Package. *Journal of Statistical Software*, 28(8):1–23, 2008.

[7] J. Kim, S. Lee, S.-w. Hwang, and S. Kim. Adding Examples into Java Documents. In *Proc. of ASE'2009*, pages 540–544, 2009.

[8] J. Kim, S. Lee, S.-W. Hwang, and S. Kim. Enriching Documents with Examples: A Corpus Mining Approach. *ACM Trans. Inf. Syst.*, 31(1):1:1–1:27, Jan. 2013.

[9] F. Long, X. Wang, and Y. Cai. API Hyperlinking via Structural Overlap. In *Proc. of ESEC/FSE '2009*, pages 203–212, New York, NY, USA, 2009. ACM.

[10] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design Lessons from the Fastest Q&A Site in the West. In *Proc. of CHI'2011*, pages 2857–2866, 2011.

[11] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente. Documenting APIs with examples: Lessons learned with the APIMiner platform. In *Proc. of WCRE'2013*, pages 401–408, Oct 2013.

[12] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What Makes a Good Code Example? A Study of Programming Q&A in StackOverflow. In *Proc. of ICSM'2012*, pages 25–34, 2012.

[13] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar. Inferring Method Specifications from Natural Language API Descriptions. In *Proc. of ICSE'2012*, pages 815–825, 2012.

[14] C. Parnin, C. Treude, L. Grammel, and M. A. Storey. Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow. Georgia Institute of Technology, Tech. Rep, 2012.

[15] G. Petrosyan, M. P. Robillard, and R. De Mori. Discovering Information Explaining API Types Using Text Classification. In *Proc. of ICSE'2015*, pages 869–879, 2015.

[16] L. Ponzanelli, A. Bacchelli, and M. Lanza. Leveraging Crowd Knowledge for Software Comprehension and Development. In *Proc. of CSMR'2013*, pages 57–66, March 2013.

[17] M. P. Robillard. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Softw.*, 26(6):27–34, Nov. 2009.

[18] M. P. Robillard and Y. B. Chhetri. Recommending reference API documentation. *Empirical Software Engineering*, 20(6):1558–1586, 2014.

[19] I. Sommerville. *Engenharia de Software*. Pearson Addison-Wesley, São Paulo, 8th edition, 2007.

[20] L. B. L. Souza, E. C. Campos, and M. A. Maia. On the Extraction of Cookbooks for APIs from the Crowd Knowledge. In *Proc. of the 28th Brazilian Symposium on Software Engineering - SBES'2014*, pages 21–30, 2014.

[21] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng. The Impact of Social Media on Software Engineering Practices and Tools. In *Proc. of the FSE/SDP Workshop on Future of Software Engineering Research, FoSER '10*, pages 359–364, 2010.

[22] J. Stylos, B. A. Myers, and Z. Yang. Jadeite: Improving API Documentation Using Usage Information. In *CHI '09 Extended Abstracts*, pages 4429–4434, New York, NY, USA, 2009. ACM.

[23] S. Subramanian, L. Inozemtseva, and R. Holmes. Live API Documentation. In *Proc. of ICSE'2014*, pages 643–652, 2014.

[24] C. Treude, F. F. Filho, B. Cleary, and M.-A. D. Storey. Programming in a Socially Networked World: the Evolution of the Social Programmer. The Future of Collaborative Software Development, 2012.

[25] C. Treude and M. P. Robillard. Augmenting API Documentation with Insights from Stack Overflow. In *Proc. of ICSE '2016*, 2016.

[26] T. Xie and J. Pei. MAPO: Mining API Usages from Open Source Repositories. In *Proc. of MSR '2006*, pages 54–57. ACM, 2006.

[27] H. Zhong and Z. Su. Detecting API Documentation Errors. In *Proc. of OOPSLA '2013*, pages 803–816, 2013.