Ferramentas de trechos de código: uma revisão da literatura

Rodrigo Vieira de Moraes [1]

Resumo: Este artigo apresenta uma revisão da literatura acerca das ferramentas que auxiliam a localização de trechos de código para reutilização. Vinte ferramentas foram pesquisadas e suas características são apresentadas a partir de uma visão arquitetural. Diferentes estratégias utilizadas pelas ferramentas foram identificadas e são apresentadas as vantagens e desvantagens de cada uma. Por fim, o trabalho sugere alguns trabalhos futuros para suprir algumas deficiências identificadas na área.

***Palavras-chave*:** Documentação de API. Exemplos de código. Reutilização de software.

Abstract: *This paper presents a literature review of tools that assist the location of code snippets for reuse. Twenty tools were researched and their features are described from a architecture view. Differents strategies adopted by the tools were identified and the advantages and drawbacks of each are presented. Lastly, this paper suggests some future works to overcome some identified issues in the field.*

***Keywords*:** *API documentation. Code samples. Software reuse.*

1 Introdução

Uma API, *Application Programming Interface*, é uma interface que desenvolvedores de software utilizam para realizar tarefas específicas quando não pretendem envolver-se em detalhes de implementação ou em detalhes de um domínio de problema sobre o qual não dispõem de conhecimento suficiente [1]. Uma API suporta reutilização, provê alta abstração que facilita as tarefas de programação e ajuda a padronizar a experiência de programação provendo uma forma única de acessar e interagir com os recursos que a API encapsula. Entretanto, as APIs têm se tornado cada vez maiores e mais complexas, tornando difícil a sua utilização [2].

Um dos obstáculos críticos ao utilizar uma API é a falta de recursos de aprendizagem ou a presença de recursos de aprendizagem inadequados. Um dos principais recursos de aprendizagem de uma API é a sua documentação, que, por muitas vezes, é incompleta ou ineficiente em termos de suporte ao aprendizado da utilização da API. Umas das principais recomendações a respeito do conteúdo da documentação de uma API é de que a documentação deve conter bons exemplos de como utilizar a API [2]. Um trecho de código exemplificando a utilização de uma parte da API é capaz de suportar diferentes atividades de aprendizagem, tal como entender as propostas da API, seus contextos e seus protocolos de utilização [3]. Não se pode deixar de destacar também a natureza imediatista dos exemplos. Programadores, muitas vezes, utilizam exemplos de código sem mesmo entendê-los e testá-los, é uma maneira rápida de cumprir uma determinada tarefa. Há programadores que utilizam a Web como uma memória externa, ou seja, alguns não memorizam os trechos de código necessários para realizar determinadas tarefas de programação, pois sabem que tais trechos de código sempre estarão disponíveis na Web, bastando procurá-los e copiá-los quando necessário [4].

O processo de reutilização de trechos de código é dividido em três fases: localização, quando o programador requisita, por meio de alguma ferramenta, um conjunto de trechos de código relevantes para a tarefa de programação na qual ele esteja trabalhando; seleção, quando o programador investiga os trechos de código retornados pela ferramenta na fase de localização a fim de selecionar aquele trecho que seja mais adequado; e a integração, quando o programador finalmente adapta o trecho de código selecionado ao contexto de programação

[1]Programa de Pós-graduação em Ciência da Computação da UFSCar Sorocaba (PPGCCS), UFSCar, Campus Sorocaba - Rodovia João Leme dos Santos (SP-264), Km 110 - Sorocaba (SP) - Brasil
{rodrigovmoraes@gmail.com}

http://dx.doi.org/10.5335/rbca.v8i3.5933

do projeto de software em desenvolvimento [5]. Ferramentas para suportar esse processo têm sido propostas e investigadas em trabalhos acadêmicos. Com o objetivo de compreender o atual desenvolvimento dessas ferramentas, trabalhos acadêmicos que apresentaram ferramentas que suportam o processo de reutilização de trechos de código foram analisados. Abaixo, na Tabela 1, são listadas as ferramentas e as respectivas referências para os trabalhos analisados:

Tabela 1: Ferramentas analisadas

Prospector	[6]	*Sourcerer*	[7]
Strathcona	[8]	*XSnippet*	[9]
Mica	[10]	*Assieme*	[11]
MAPO	[12]	*Blueprint*	[13]
eXoaDocs[2]	[14]	*Redprint*	[15]
PropER-Doc	[16]	*Fishtail*	[17]
APIExample	[18]	*Codelets*[3]	[19]
SnipMatch	[20]	*APIMiner*	[21]
UsETeC	[22]	*SPARS-J*	[23]
Krugle	[24]	*Open Hub Code Search*	[25]

Fonte: elaborado pelo autor

Por meio dessa análise, foi possível identificar as estratégias adotadas pelas ferramentas, as quais são apresentadas neste artigo. Na próxima seção (Seção 2), é apresentada uma visão arquitetural das ferramentas investigadas. Essa visão arquitetural guiará, nas seções subsequentes (da Seção 3 até a Seção 6), a apresentação das características das ferramentas analisadas. Por fim, na Seção 8, o artigo é concluído por meio da sumarização das características encontradas nas ferramentas analisadas, também são discutidos possíveis trabalhos futuros que poderão sanar algumas deficiências identificadas.

2 Visão arquitetural

Neste artigo, é utilizada a denominação *CSTs - **C**ode **S**nippets **T**ools* - para referenciar o conjunto de ferramentas que auxiliam a localização de trechos de código para reutilização. Na Figura 1, é apresentada a arquitetura das ferramentas *CSTs*, que guiará ao longo deste artigo a apresentação das características das ferramentas analisadas.

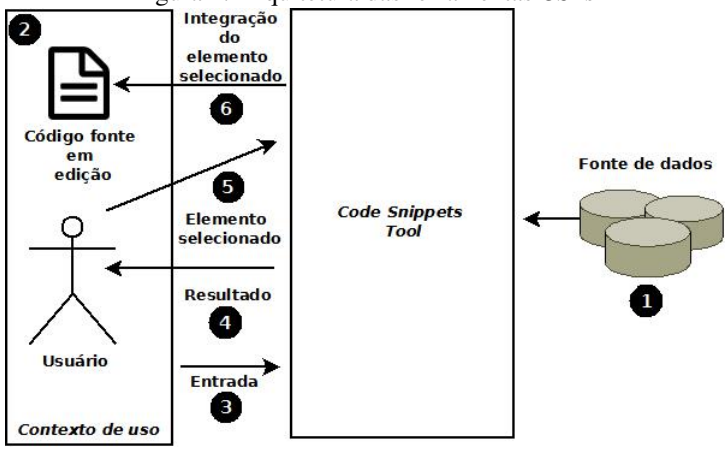
1. As ferramentas *CSTs* extraem os trechos de código a serem apresentados ao usuário a partir de uma fonte de dados. As diferentes fontes de dados utilizadas pelas ferramentas analisadas serão apresentadas na Seção 4.

2. O *contexto de uso* é o ambiente no qual a ferramenta está inserida e em que o usuário interage com a ferramenta. Algumas ferramentas *CSTs* são integradas a uma IDE[4], p.ex. a ferramenta *Blueprint* [13]. Outras, que não são integradas a uma IDE, são ferramentas web e o usuário necessita utilizar um browser a fim de acessar a ferramenta, p.ex. a ferramenta *Assieme* [11]. O usuário é um programador que está realizando alguma tarefa durante a edição de um código fonte em um projeto de software em desenvolvimento e necessita de auxílio a fim de utilizar alguma API de interesse. O *contexto de uso* será discutido na Seção 3.

3. O usuário incita a ferramenta por meio de uma ação explícita que informa a intenção de localizar um trecho de código relevante para a tarefa que ele esteja trabalhando. A entrada pode ser algo digitado pelo usuário, por exemplo uma query[5] contendo palavras chaves [11], ou informações do *contexto de uso*, como informações do código fonte em edição na IDE [6], ou uma query complementada com informações no *contexto de uso* [13]. Os diferentes tipos de entrada serão apresentados na Seção 5. Em algumas ferramentas, o

[2] *eXoaDocs* não é o nome da ferramenta apresentada em [14], mas a ferramenta apresentada gera páginas HTML que são a documentação de uma API complementada com trechos de código. A essa documentação complementada os autores dão o nome de *eXoaDocs*. No trabalho, a ferramenta apresentada não recebe uma denominação. Aqui a ferramenta é referenciada como *eXoaDocs* por uma questão de simplicidade.
[3] *Codelets* não é o nome da ferramenta apresentada em [19]. *Codelets* é utilizado no trabalho para denominar trechos de código que são associados a um editor gráfico que tem como objetivo auxiliar o usuário a editar os parâmetros do trecho de código. No trabalho, a ferramenta apresentada não recebe uma denominação. Aqui a ferramenta é referenciada como *Codelets* por uma questão de simplicidade.
[4] ***I**ntegrated **D**evelopment **E**nvironment*, um ambiente integrado para desenvolvimento de software.
[5] Termo utilizado na área Recuperação da Informação (RI) para designar a informação produzida pelo usuário como entrada para um sistema de busca [26, p. 26-27].

Figura 1: Arquitetura das ferramentas *CSTs*



Fonte: elaborado pelo autor

usuário não necessita realizar uma ação explícita para que a ferramenta produza um resultado, mas a ferramenta produz um resultado automaticamente quando determinado evento ocorre no *contexto de uso*, como por exemplo é o caso da ferramenta *Redprint* [15], que automaticamente exibe trechos de código extraídos a partir de páginas web de acordo com termos (palavras-chaves) presentes na linha onde o cursor esteja posicionado durante a edição de um arquivo de código fonte na IDE.

4. A ferramenta apresenta o resultado ao usuário. O resultado contém os trechos de código que a ferramenta julga serem relevantes para o usuário baseada nas informações da entrada. Neste momento, o usuário necessita inspecionar o resultado apresentado pela ferramenta e decidir qual trecho de código, entre os trechos de código apresentados, é o mais adequado para a tarefa de programação que ele esteja trabalhando.

5. O usuário seleciona um trecho de código entre os trechos de código apresentados no resultado.

6. O trecho de código selecionado pelo usuário é então integrado no código fonte em edição. Algumas ferramentas oferecem facilidades para a integração do trecho de código selecionado, enquanto em outras ferramentas o usuário necessita copiar o conteúdo do trecho de código, colá-lo no código fonte em edição e realizar as adaptações necessárias manualmente. Na Seção 6, será discorrido sobre o suporte que as ferramentas oferecem para auxiliar a integração do trecho de código selecionado.

3 Contexto de uso

O *contexto de uso* é o ambiente no qual a ferramenta está inserida e em que o usuário, normalmente um programador que está desenvolvendo um software, interage. Em relação ao *contexto de uso*, entre as ferramentas *CSTs* apresentadas nos trabalhos analisados, é possível identificar dois grupos: um grupo de ferramentas web e outro grupo de ferramentas integradas a uma IDE.

Ferramentas web possuem a vantagem de serem independentes de uma IDE [13, p. 513]. Por exemplo, um programador que esteja desenvolvendo um software na linguagem Java através da IDE *Eclipse* [27] e outro programador utilizando a IDE *Netbeans* [28] ambos podem utilizar a mesma ferramenta *CST* a fim de localizar trechos de código. Exemplos de ferramentas web são: *Assieme* [11], *APIExample* [18], *Krugle* [24] e *Open Hub Code Search* [25]. Já as ferramentas integradas a uma IDE podem explorar o contexto de programação da IDE que o usuário está desenvolvendo um software para extrair informações que servem como entrada ou que contribuem para a entrada a fim de localizar um trecho de código, p.ex. a ferramenta *XSnippet* [9]. Mais detalhes de como as ferramentas integradas a uma IDE utilizam informações no contexto de programação como entrada serão apresentados na Seção 5.

Uma outra vantagem das ferramentas web é que para o usuário acessar uma ferramenta web, esse utiliza um navegador web e navegadores web fornecem um ambiente rico de interação proporcionado pela possibilidade de navegação entre as páginas. O usuário pode, por meio de hiperlinks, acessar páginas relacionadas com uma página sendo visualizada e, também, pode voltar para as páginas acessadas anteriormente. Além disso, navegadores web são customizados e configurados de acordo com o perfil do usuário, pois possuem recursos como os favoritos, organização de páginas em abas, os cookies, os certificados de segurança e o histórico de navegação [29, p. 65]. Embora IDEs modernas, como o *Eclipse*, possuem um navegador web incorporado, este normalmente não é um navegador completo. Por exemplo, o navegador web incorporado ao *Eclipse* não possui favoritos, histórico, organização em abas e possui um espaço de utilização na IDE bastante limitado, consequentemente raramente é utilizado pelos desenvolvedores [29, p. 67]. Portanto, há um dilema, ao escolher integrar uma ferramenta à IDE, o projetista da ferramenta *CST* está abrindo mão, além da independência da IDE, de um ambiente de interação rico proporcionado pelos navegadores web, a favor de um ambiente (IDE) com uma interface de interação mais acoplada e próxima da tarefa que o programador esteja trabalhando no momento [13, p. 514]. Essa proximidade com a tarefa que o usuário está trabalhando no momento evita o problema da troca de contexto entre o ambiente de programação e o navegador web. Essa troca de contexto pode provocar uma distração para o usuário e diminuir sua produtividade [30, p. 62]. Em uma ferramenta integrada à IDE, o usuário não necessita abrir o navegador web e suas ações são realizadas dentro da própria IDE [17, p. 49].

Com o objetivo de utilizar as informações presentes no contexto de programação na IDE e ao mesmo tempo aproveitar os recursos de interação fornecidos por um navegador web, a ferramenta *Codetrail* [29] implementa um canal de comunicação entre a IDE *Eclipse* e o navegador *Firefox*[6]. Entre outras funcionalidades disponibilizadas pelo *Codetrail*, a ferramenta disponibiliza uma aba no navegador *Firefox*. Esta aba é automaticamente atualizada para que o usuário navegue pela documentação dos elementos (pacotes, classes, métodos, etc) no contexto de programação na IDE *Eclipse* com os quais o programador esteja trabalhando no momento. Se o programador está editando um código fonte na IDE *Eclipse* e o cursor aponta para um elemento, por exemplo para um método, a documentação do elemento é automaticamente exibida no navegador *Firefox*. Embora seja importante citar a ferramenta, devido à sua particularidade em integrar informações presentes em ambientes diferentes (navegador web e IDE), *Codetrail* não se enquadra em uma ferramenta *CST*, pois seu foco é apresentar itens de documentação dos componentes de uma API e não trechos de código.

Sobre a dependência das ferramentas integradas a uma IDE, seria interessante a possibilidade que as ferramentas integradas a uma IDE fossem facilmente migradas para outras IDEs. Do ponto de vista acadêmico, isso seria útil para realizar estudos exploratórios que atingissem uma gama maior de usuários e ajudaria a compreender as ferramentas para estilos de programação diferentes e em arquiteturas de IDEs diferentes. Uma estratégia a fim de aumentar a portabilidade das ferramentas seria a criação de uma arquitetura modularizada. Módulos que não necessitassem serem acoplados à IDE poderiam ter o código fonte reutilizado entre IDEs diferentes. Somente módulos que possuíssem recursos dependentes da IDE seriam acoplados à IDE e, portanto, teriam uma versão para cada IDE, por exemplo um módulo que exibisse resultados em componentes gráficos específicos de uma IDE. Módulos não acoplados à IDE poderiam ser implementados em uma camada de software que seria executada em um processo do sistema operacional separado do processo da IDE a fim de desacoplá-los da arquitetura da IDE. Os módulos acoplados à IDE, então, se comportariam como um adaptador e se comunicariam com esta camada de software, rodando em um processo separado do sistema operacional, por meio de algum protocolo de comunicação padrão, como por exemplo o TCP/IP. Tal arquitetura é discutida em [17, p. 50], no entanto a construção de uma ferramenta implementando a arquitetura foi deixada para trabalhos futuros.

4 Fonte de dados

A fonte de dados é de onde a ferramenta *CST* extrai os trechos de código, os quais são apresentados ao usuário.

[6] <http://www.mozilla.org/firefox/products/>

4.1 Projetos de código aberto

Um conjunto de projetos de software disponíveis na Internet que são cada vez mais populares são os projetos de código aberto [31], os quais são utilizados por algumas ferramentas *CSTs* a fim de extrair trechos de códigos e apresentá-los ao usuário [12, 25, 32]. Um dos desafios ao coletar trechos de código de projetos de código aberto pela Internet é que não há um padrão na distribuição dos códigos fontes presentes nesses projetos. Cada repositório de projetos de código aberto utiliza um controlador de versão diferente e um protocolo de download diferente. O sistema de busca de trechos de código *Sourcerer* [33] ameniza esse problema por meio de uma arquitetura baseada em módulos, para cada projeto de software, é criado um módulo denominado *Code Crawler*, que trata as particularidades para extrair as informações do projeto de software. Os módulos *Code Crawlers* no sistema *Sourcerer* devem seguir um padrão de implementação para que possam ser facilmente integrados ao sistema quando há a necessidade de extrair trechos de código de um novo projeto de código aberto.

4.2 Páginas web

Ferramentas que consideram apenas os códigos fontes de projetos de código aberto ignoram trechos de código presentes em páginas web, como páginas de documentação, fóruns, blogs e páginas de tutoriais das APIs. Tais páginas, além de apresentarem trechos de código como exemplos para ensinar a utilização da API, contêm textos explicativos que contribuem para o entendimento desses trechos de código. Os termos presentes em uma página web contendo um trecho de código colabora com a eficiência da busca realizada pela ferramenta *CST* ao permitir que o usuário busque pelo trecho de código utilizando um termo (uma palavra chave) presente na página, não limitando a busca somente pelos termos presentes no trecho de código [11, 13]. Além disso, os trechos de código contidos em páginas web geralmente são cuidadosamente confeccionados por um especialista da API com a intenção de demonstrar o uso da API, logo há uma tendência de serem mais concisos [11, p. 15]. Aproveitando o potencial de trechos de código presentes nas páginas web, trabalhos anteriores apresentaram ferramentas *CSTs* que extraem trechos de código de páginas web para serem apresentados como resultados das buscas realizadas pelos usuários [10, 11, 13, 18]. Tais ferramentas utilizam um buscador web, como o *Google*, a fim de buscar por páginas web que contenham trechos de código. Os principais desafios enfrentados por essas ferramentas são a identificação de quais páginas web possuem trechos de código e o isolamento e a extração de segmentos de uma página web que correspondam aos trechos de código. Entre as dificuldades encontradas ao extrair trechos de código a partir de páginas web é possível citar: um segmento de texto de uma página web pode conter elementos sem uma separação clara entre qual parte representa o trecho de código de interesse e qual parte é apenas um texto; uma página web pode conter um trecho de código em uma linguagem de programação semelhante à linguagem de interesse, porém diferente e irrelevante para o usuário, por exemplo, uma ferramenta com o objetivo de apresentar trechos de código Java poderia encontrar trechos de código C++; trechos de código podem apresentar elementos distrativos, como "...", e que não fazem parte da linguagem de programação; outro elemento distrativo que pode aparecer em trechos de código em páginas web são as numerações das linhas [11, p. 18].

As principais técnicas para identificar trechos de código em páginas web envolvem a aplicação de métodos de mineração de dados, mais especificamente métodos de classificação de dados [34, p. 285-382], que consistem em extrair características de um segmento de texto, como a presença ou não de determinadas palavras reservadas em uma linguagem de programação (*if*, *else*, *for*, *while*, etc), presença ou não de linhas terminadas com ponto e vírgula e a presença ou não de chaves e colchetes. Tais características são, então, utilizadas como entrada para um algoritmo de classificação que retorna se o segmento de texto é um trecho de código ou não. Esses métodos são baseados em heurísticas e, portanto, não são livres de falhas. Por exemplo, se o objetivo da ferramenta é identificar trechos de código Java em páginas web, um algoritmo de classificação poderia classificar de forma errônea um trecho de código escrito em C++ por possuírem características semelhantes, ou ainda, um segmento de texto poderia ser classificado como trecho de código por fazer muitas referências para elementos de trechos de código [13, p. 517]. Uma técnica utilizada, principalmente, para diminuir os falsos positivos, segmentos de texto classificados como trechos de código quando na realidade não são, é a utilização de um *parser* adaptativo [18, p. 594]. Um *parser* adaptativo é um *parser* com o objetivo de compilar trechos de código, no entanto não é muito rigoroso em relação à sintaxe da linguagem de programação. Ele é capaz de realizar pequenas correções no trecho de código a fim de obter uma compilação bem sucedida. Tais modificações podem incluir, por exemplo, a remoção de caracteres ilegais, a remoção de uma linha identificada como uma linha não contendo sentenças válidas, a remoção de números presentes no início das linhas e a inserção de trechos de código soltos (fora de um método e

fora de uma classe) para dentro de um método e para dentro de uma classe para que possam ser compilados.

4.3 Testes unitários

Métodos baseados na mineração de repositórios de projetos de código aberto ou de páginas web tendem a produzir muitos códigos irrelevantes (falsos positivos), principalmente pelo fato de tentar extrair elementos relevantes a partir da análise de uma grande quantidade de informações irrelevantes. Há um dilema na escolha do tamanho da fonte de dados escolhida para extrair as informações. Se a fonte de dados é muito pequena, elementos relevantes podem deixar de serem encontrados. Se a fonte de dados for muito extensa e diversificada, como páginas web, a ferramenta poderá produzir muitos resultados irrelevantes. Em [35], é sugerido que trechos de código possam ser extraídos dos testes unitários do próprio projeto de desenvolvimento da API de interesse, a fim de contornar essa deficiência. Um teste unitário é um trecho de código com o objetivo de testar um componente de uma API, como uma classe ou um método [36]. O conjunto de testes unitários de um projeto de software é um conjunto conciso e relevante a respeito da API sendo testada. Eles apresentam não somente os cenários de uso de uma classe, mas também as melhores práticas para utilizar a classe, pois são normalmente confeccionados pela mesma organização que projetou e implementou a classe. Além disso, nem sempre há códigos fontes disponíveis na Web de projetos que utilizem a API de interesse. Por exemplo, há casos que a API de interesse foi recentemente publicada ou os projetos de software que utilizam a API podem ser projetos privados de empresas que não possuem o código aberto por interesses comerciais [35].

O potencial da utilização dos testes unitários como fonte de trechos de código como exemplos de utilização de uma API foi explorado em [37]. No trabalho, foi realizado um experimento envolvendo tarefas de programação que utilizavam componentes de uma determinada API e foram disponibilizados testes unitários do próprio projeto de desenvolvimento da API para os participantes. Foi observado que a utilização dos testes unitários como exemplos pode ser útil para enfrentar dificuldades na utilização da API, no entanto o desenvolvedor pode ter dificuldades para encontrar o cenário de uso correto de uma API dentro dos testes unitários e, portanto, é necessário ferramentas para apoiar esse processo.

Muitos são os desafios para extrair trechos de código a partir dos testes unitários [35]. Testes unitários são conceitualmente divididos em três fases: configuração, execução e verificação. A configuração contém sentenças para criar um objeto da classe a ser testada e colocar esse objeto em um estado inicial. A execução chama métodos para o objeto criado na configuração a fim de alterar o estado do objeto. E, por fim, a verificação contém sentenças para verificar se o objeto encontra-se no estado esperado. Na Figura 2, é apresentado um exemplo de teste unitário. As linhas marcadas com S1 e S2 são linhas de configuração. Linhas marcadas com E1 e E2 são linhas de execução e linhas marcadas com O1 e O2 são linhas de verificação. O teste unitário testa a classe *ArrayList*, no entanto apresenta, no mesmo teste unitário, dois cenários de uso de *ArrayList*, um cenário para adicionar um elemento (linhas marcada com S1, E1 e O1) e outro para remover um elemento (linhas marcadas com S2, E2 e O2). Para que esse trecho de código possa ser utilizado como um exemplo de utilização da classe *ArrayList*, seria necessário separar esses dois cenários. Assim, teria-se dois trechos, um que mostrasse como adicionar um elemento e outro que mostrasse como remover um elemento. Além disso, certas sentenças são relevantes apenas nos testes unitários, por exemplo as sentenças de verificação. Seria interessantes que tais sentenças não fossem exibidas em um trecho de código apresentado como exemplo de utilização da classe *ArrayList*. Porém, separar os cenários e remover sentenças irrelevantes não é uma tarefa trivial. Linhas podem pertencer ao mesmo cenário, como as linhas 1 e 2. A divisão configuração, execução e verificação é apenas conceitual, não há uma regra que garanta que uma determinada linha realmente pertença à fase de configuração, à fase de execução ou à fase de verificação [38].

Em [22], testes unitários foram analisados a fim de estabelecer padrões de como eles são codificados. Dessa análise foram definidas algumas heurísticas a fim de separar as fases dos testes unitários (configuração, execução e verificação) e separar os cenários de uso dos componentes testados. As heurísticas foram então utilizadas para implementar uma ferramenta denominada *UsETeC — Usage Examples from Test Code*, que é uma ferramenta baseada em documentação, que aceita como entrada a documentação da API no formato *Javadoc* e produz como resultado a documentação complementada com trechos de código que demonstram a utilização da API (ferramentas baseadas em documentação serão abordadas na Subseção 5.4).

Figura 2: Exemplo de um teste unitário

```
1  S1 S2 List array = new ArrayList();
2  S1 S2 String item = "bob";
3  E1 array.add(item);
4  O1 int size = array.size();
5  O1 assertEquals(1, size);
6  O1 assertTrue(array.contains(item));
7  E2 array.remove(item);
8  O2 assertFalse(array.contains(item));
```

Fonte: [38]

4.4 Trechos de código cadastrados manualmente

Há ferramentas que não extraem trechos de código automaticamente de uma fonte de dados, mas confia em um repositório de trechos de código cadastrados manualmente [19, 20]. Um tipo especial de trechos de código cadastrados manualmente são os *templates* de código. *Templates* de código são trechos de código contendo diretivas de integração [20], as quais são utilizadas por uma ferramenta, normalmente um plugin ou uma funcionalidade nativa da IDE, para realizar a integração do trecho de código no código fonte em edição durante a ativação do *template* pelo usuário. Um exemplo de IDE com suporte aos *templates* de código é o *Eclipse* [27]. O *Eclipse* disponibiliza um conjunto de interfaces gráficas a fim de facilitar a manutenção desses *templates* (criação, edição, remoção e migração). Uma vez que um *template* de código esteja cadastrado na IDE do desenvolvedor, é possível que o desenvolvedor ative o *template* durante a edição de um código fonte Java. Mais sobre *templates* de código será discutido na Seção 6.

Trechos de código cadastrados manualmente possuem a mesma vantagem dos trechos de código extraídos em páginas web mencionados anteriormente, tais trechos de código (cadastrados manualmente) são confeccionados a fim de ensinar e facilitar a utilização de uma API e, portanto, há uma tendência de possuir uma qualidade melhor e serem mais concisos. Diferente dos trechos de código extraídos de páginas web, trechos de código cadastrados manualmente não possuem os riscos associados à extração automática de uma fonte de dados. Em trechos de código extraídos de páginas web, o algoritmo de extração baseia-se em heurísticas [18, p. 594]. Não há uma garantia que um trecho de código prático e útil será extraído da página web. O mesmo risco existe ao extrair trechos de código a partir dos testes unitários [35]. Por outro lado, a desvantagem é que, para os trechos de códigos cadastrados manualmente, existe um custo envolvido tanto na criação como na manutenção dos trechos de código [20, p. 219-220]. Logo, o conjunto de trechos de código cadastrados manualmente possuem uma tendência de serem incompletos. Ou seja, para uma determinada API, há o risco de não haver trechos de código confeccionados para a API ou o conjunto de trechos de código confeccionados para a API poderá não cobrir todos os cenários de uso da API.

5 Entrada

Nesta seção, são apresentados os diferentes tipos de entrada utilizados pelas ferramentas *CSTs — **C**ode **S**nippet **T**ools*. A entrada é utilizada por uma ferramenta *CST* como critério para decidir quais elementos são relevantes para o usuário.

5.1 Query baseada em palavras-chaves

Um tipo de entrada utilizado pelas ferramentas *CSTs* são as palavras-chaves. Como um exemplo, há a ferramenta *Assieme* [11], que aceita como entrada palavras-chaves e retorna componentes de uma API (pacotes, classes e métodos) associados às palavras-chaves digitadas pelo usuário. *Assieme* infere a relação de uma palavra-chave e um componente de uma API baseada nos textos de páginas web contendo trechos de código que referenciam o componente da API. Por exemplo, para a query *"output acrobat"*, a ferramenta retorna o pacote *com.lowagie.text.pdf*,

pacote da API *iText*[7] para manipulação de arquivos PDF, desde que a ferramenta tenha encontrado alguma página web cujo o conteúdo possua os termos *output* e *acrobat* e cujo o conteúdo contenha um trecho de código que faça referência ao pacote *com.lowagie.text.pdf*. Uma vez que a ferramenta tenha exibido os componentes da API (pacotes, classes e métodos) baseada nas palavras-chaves digitadas, o usuário pode selecionar um dos componentes para que sejam exibas as páginas web contendo os trechos de códigos que façam referência ao componente selecionado. Outra ferramenta que utiliza palavras-chaves é a ferramenta *Mica* [10], a qual redireciona uma query digitada pelo usuário contendo palavras-chaves para o *Google* e processa as páginas retornadas a fim de apresentar ao usuário somente aquelas que possuam trechos de código Java. A vantagem de utilizar uma entrada baseada em palavras-chaves é que é um método simples e intuitivo para o usuário. Não há a necessidade de o usuário possuir informações detalhadas, como informações estruturais, de forma antecipada. O usuário precisa apenas conhecer algum termo que semanticamente esteja relacionado com o trecho de código desejado [39, p. 253].

5.2 Query baseada em informações específicas do domínio

Query baseada em palavras-chaves é um tipo de entrada utilizada por sistemas de busca de um domínio mais genérico, como por exemplo os sistemas de busca de páginas web *Google*[8] e *Bing*[9]. Utilizar uma query baseada somente em palavras-chaves em uma ferramenta específica de um domínio, como as ferramentas *CSTs*, é uma estratégia que ignora o potencial de utilizar critérios mais detalhados e mais próximos do domínio considerado, no caso o domínio de desenvolvimento de software. Logo, algumas *CSTs* utilizam informações mais específicas do que as palavras-chaves. Por exemplo, a ferramenta *APIExample* [18] aceita como entrada o identificador de uma classe e retorna trechos de código que façam referência para a classe especificada.

Um outro exemplo de ferramenta que utiliza queries mais específicas é o sistema de busca de trechos de código denominado *Sourcerer* [7]. *Sourcerer* aceita queries em diferentes modos. As queries são baseadas em palavras-chaves, no entanto, o resultado depende do modo escolhido. Seis modos de pesquisa são definidos: *C (Component), C-use (Component USE), F (Function), F-use (Function USE), Fi-C (Control Structure Fingerprint) e Fi-T (Type Fingerprint)*. Os modos *C (Component)* e *F (Function)* são utilizados para buscar por trechos de código que declaram, respectivamente, as classes ou os métodos relacionados com as palavras-chaves digitadas na query. Os modos *C-use (Component USE)* e *F-use (Function USE)* são utilizados para buscar por trechos de código que utilizam, respectivamente, as classes ou os métodos relacionados com as palavras-chaves digitadas na query. *Fi-C (Control Structure Fingerprint)* e *Fi-T (Type Fingerprint)* são modos utilizados para recuperar trechos de código baseado em critérios estruturais mais detalhados. *Fi-C (Control Structure Fingerprint)* permite criar queries com restrições de acordo com as estruturas de controle presentes nos trechos de código. Por exemplo, por meio do modo *Fi-C* é possível especificar uma query com a seguinte semântica: "procure por trechos de código de métodos contendo mais de 50 sentenças *switch*". *Fi-T (Type Fingerprint)* possibilita criar queries com restrições de acordo com a estrutura de classes, métodos, atributos, construtores, etc. Por meio do modo *Fi-T* é possível, por exemplo, especificar uma query com a seguinte semântica: "procure por trechos de código que definem classes que possuam campos do próprio tipo e que não tenham métodos declarados".

5.3 Utilizando informações no contexto de programação

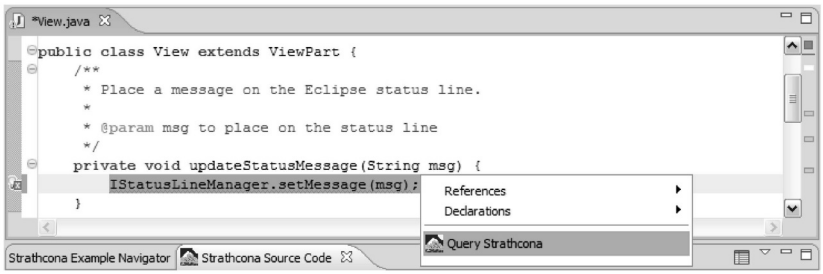
Uma query mais específica, com informações detalhadas do trecho de código desejado, pode produzir resultados mais precisos, se comparado com queries menos específicas, como as queries baseadas somente em palavras-chaves. No entanto, o usuário pode não possuir as informações necessárias para formular uma query mais específica. Além disso, uma ferramenta que depende de queries mais específicas está subordinada à disposição do usuário em formular queries que, normalmente, são maiores e mais complexas [39, p. 253].

A fim de poupar o usuário de formular queries longas e complexas, algumas ferramentas integradas à IDE se beneficiam do acesso às informações presentes no contexto de programação para, automaticamente, formular queries com informações específicas do domínio de desenvolvimento de software. Por exemplo, *Stratchona* [8] permite que o usuário selecione um trecho de código em um código fonte em edição na IDE *Eclipse* e solicite que a ferramenta realize uma busca por trechos de código estruturalmente semelhantes ao trecho de código selecionado.

[7] <http://itextpdf.com/api>
[8] <http://www.google.com>
[9] <http://www.bing.com>

Stratchona considera informações estruturais do trecho de código selecionado e informações estruturais da classe que contém o trecho de código selecionado. As informações estruturais consideradas pela ferramenta são quais métodos o trecho de código selecionado chama, quais tipos e quais atributos de classe são referenciados pelo trecho de código e qual classe é estendida pela classe que contém o trecho de código selecionado. Por exemplo, na Figura 3 é exibido um cenário de uso da ferramenta, no qual o usuário seleciona um trecho de código como entrada para uma query. Do trecho de código selecionado são extraídas as seguintes informações estruturais: é chamado o método *IStatusLineManager.setMessage(String)*, os tipos *IStatusLineManager* e *String* são referenciados e a classe que contém o trecho de código selecionado estende a classe *ViewPart*. Para buscar trechos de código presentes em seu repositório estruturalmente semelhantes ao trecho de código selecionado, *Stratchona* define quatro heurísticas de busca: *INHERITS*, *CALLS*, *USES* e *REFERENCES*. *INHERITS* procura por trechos de código de métodos cuja classe estenda a mesma classe estendida pela classe que contém o trecho de código selecionado (no exemplo apresentado, é a classe *ViewPart*). *CALLS* procura por trechos de código que chamem os mesmos métodos. *USES* procura por trechos de código que façam referência para os mesmos tipos. E *REFERENCES* procura por trechos de código que façam referência para os mesmos atributos de classes. Após o conjunto de trechos de código serem recuperados para cada heurística, os trechos de código são ranqueados pela quantidade de heurísticas que os retornaram. Por exemplo, um trecho de código retornado por ambas heurísticas *INHERITS* e *CALLS* é melhor ranqueado do que um trecho de código retornado somente pela heurística *USES*.

Figura 3: Uma query na ferramenta *Stratchona*



Fonte: adaptado de [8, p. 954]

Outras informações utilizadas como entrada por ferramentas integradas à IDE apresentadas nos trabalhos analisados incluem: o nome do método e da classe que estão sendo editados [12]; nomes dos elementos (classes e métodos) do projeto de software que o usuário tenha recentemente interagido (editado ou acessado) [17]; tipos das variáveis no escopo do código fonte em edição [6]. Há ferramentas que utilizam informações no contexto de programação a fim de complementar uma query baseada em palavras-chaves, por exemplo, a ferramenta *Blueprint* [13, p. 515] aceita queries baseadas em palavras chaves e complementa a query com o nome e a versão da linguagem de programação do código fonte em edição na IDE. A ferramenta *SnipMatch* [20, p. 220] utiliza os tipos e os nomes das variáveis no escopo do código fonte em edição na IDE a fim de ranquear e filtrar trechos de código retornados por uma query baseada em palavras-chaves.

5.4 Ferramentas baseadas em documentação

Há ferramentas que têm o objetivo de complementar a documentação de uma API. Tais ferramentas não recebem uma entrada específica do usuário, mas recebe a documentação completa da API no formato *Javadoc* e produz uma documentação complementada com trechos de código. Os trechos de código são extraídos a partir da Web [14], dos projetos de código aberto [21] ou dos testes unitários do próprio projeto de desenvolvimento da API [22]. Para cada método presente na documentação da API, a documentação do método é automaticamente complementada com trechos de código que demonstram a utilização do método.

6 Integração do elemento selecionado

Após o usuário julgar que determinado trecho de código apresentado no resultado de uma ferramenta *CST* é útil para a tarefa de programação que ele está trabalhando, é necessário adaptá-lo ao contexto do software que está sendo desenvolvido [5]. Adaptações necessárias podem incluir a renomeação de variáveis, inclusão de dependências [20] e eliminação de sentenças do trecho de código original [4, 40].

Entre as ferramentas *CSTs* analisadas, as que extraem trechos de código automaticamente de alguma fonte de dados, como a Web (Subseção 4.2) e os projetos de código aberto (Subseção 4.1), não apresentam funcionalidades que facilitam a adaptação do trecho de código no contexto de programação [7, 11, 18]. Mesmo quando tais ferramentas são integradas a uma IDE [8, 12, 15], o usuário necessita copiar o trecho de código apresentado pela ferramenta, colá-lo e adaptá-lo manualmente. A exceção é a ferramenta *Blueprint* [13], que, automaticamente, cola um trecho de código apresentado no código fonte em edição, no entanto, também não apresenta funcionalidades para auxiliar a adaptação do trecho de código, como renomear variáveis, resolver dependências ou configurar parâmetros.

De outra forma, ferramentas baseadas em *templates* de código apresentam funcionalidades que facilitam a adaptação dos trechos de código. *Templates* de código são trechos de código cadastrados manualmente contendo diretivas de integração e são suportados por IDEs modernas, como por exemplo pela IDE *Eclipse*. As diretivas de integração são sentenças que informam à ferramenta como o trecho de código deverá ser adaptado no contexto do código fonte em edição ao ativar o *template* (após o usuário ter realizado uma busca pelo *template* desejado). Entre as principais diretivas de integração suportadas pelos *templates* do *Eclipse*, é possível citar a diretiva *newName*, que possibilita declarar uma variável no trecho de código definido no *template* e instruir o *Eclipse* a renomear a variável caso outra variável com o mesmo nome esteja presente no escopo do código fonte em edição durante a ativação do *template*; a diretiva *import*, que instrui o *Eclipse* a importar determinados pacotes e determinadas classes ao ativar o *template* e a diretiva *var*, que declara um parâmetro para o *template*. Parâmetros de *templates* são substituídos por seus respectivos argumentos durante a ativação do *template* e tem como função reaproveitar o trecho de código em contexto diferentes.

Figura 4: Exemplo de *template* da ferramenta *SnipMatch*

```
1  ${import(java.io.File)}
2  ${import(java.io.BufferedReader)}
3  ${import(java.io.FileReader)}
4  ${import(java.io.IOException)}
5  ${import(java.util.ArrayList)}
6
7  ${helper}
8  class FileLineReader {
9     public static String[] readAllLines(File file) {
10       ArrayList<String> lines = new ArrayList<String>();
11       try {
12          BufferedReader br = new BufferedReader(new FileReader(file));
13          String line = null;
14
15          while ((line = br.readLine()) != null) {
16             lines.add(line);
17          }
18          br.close();
19       }
20       catch (IOException e) { return null; }
21       return lines.toArray(new String[lines.size()]);
22    }
23 }
24 ${endHelper}
25
26 String[] ${freeName(lines)} = FileLineReader.readAllLines(${fileObject});
```

Fonte: adaptado de [20, p. 223]

A ferramenta *SnipMatch* [20] estende a ideia utilizada no *Eclipse* e propõe um sistema de busca de *templates* baseado em palavras-chaves, diferente da funcionalidade nativa do *Eclipse*, que busca os *templates* por meio somente de um nome. A ferramenta também propõe novas diretivas de integração. Uma dentre as diretivas de integração propostas é a diretiva *helper*, que possibilita declarar classes utilitárias para um *template*. Classes utilitárias são classes contendo métodos que oferecem alguma funcionalidade ao trecho de código definido pelo *template*.

Na Figura 4, é apresentado um exemplo de *template* da ferramenta, o qual declara uma classe utilitária de nome *FileLineReader*, declarada na linha 8 e utilizada da linha 26. A classe utilitária é delimitada pelas diretivas de integração *helper*, na linha 7, e *endHelper*, na linha *24*. Durante a ativação do *template*, a ferramenta adiciona a classe utilitária no projeto de software que o usuário esteja trabalhando na IDE, caso a classe ainda não tenha sido adicionada ao projeto. Isso garante que o trecho gerado pelo *template*, o qual depende da classe utilitária, funcione corretamente.

Em [19], é proposto a utilização de *templates* denominados *Codelets*. *Codelets* são *templates* associados a uma interface gráfica que auxilia a edição dos valores dos parâmetros do *template*. A interface gráfica é construída pelo próprio criador do *Codelet*. No trabalho, uma ferramenta com suporte aos *Codelets* é implementada e integrada à IDE *ACE*[10]. Os *Codelets* são associados a palavras-chaves e, durante a edição de um arquivo código fonte, o usuário pode procurar, por meio das palavras-chaves, por um *Codelet* desejado e acioná-lo. Ao acionar o *Codelet*, o editor gráfico de parâmetros associado é apresentado ao usuário para que esse possa configurar os valores dos parâmetros do *Codelet*. De acordo com os valores configurados, o trecho de código é gerado e colado no código fonte em edição. O editor gráfico de parâmetros associado ao *Codelet* continua associado ao trecho de código gerado pelo *Codelet*, ou seja, mesmo após a ativação do *Codelet* e após o trecho de código ter sido já colado em um código fonte, o usuário pode retornar ao trecho de código a fim de alterar os valores dos parâmetros por meio do editor gráfico de parâmetros.

7 Resultados

Na Tabela 3, estão sumarizadas as características encontradas nas ferramentas analisadas. Os nomes das colunas e os respectivos valores são acrônimos e estão descritos na Tabela 2. Cada coluna da Tabela 3 representa um elemento da arquitetura apresentada na introdução deste artigo. Na Tabela 2, são também relacionadas as seções que abordaram neste artigo cada elemento da arquitetura, assim como as respectivas subseções. Células não preenchidas na Tabela 3 representam ferramentas cujos trabalhos não apresentaram informações suficientes para classificar a ferramenta.

Tabela 2: Acrônimos da Tabela 3

Colunas			**Valores**		
Acrônimo	**Descrição**	**Seção**	**Acrônimo**	**Descrição do valor**	**Subseção**
CONT	**Cont**exto de Uso	3	IDE	Integrada à uma **IDE**	—
			WEB	Ferramenta **web**	—
FDADOS	**F**onte de **Dados**	4	PCA	**P**rojetos de **C**ódigo **A**berto	4.1
			PW	**P**áginas **w**eb	4.2
			TU	**T**este **u**nitário	4.3
			TCCM	**T**rechos de **c**ódigo **c**adastrados **m**anualmente	4.4
ENT	**Ent**rada	5	PC	**P**alavras-**c**haves	5.1
			IED	**I**nformações **e**specíficas do **d**omínio	5.2
			ICP	**I**nformações no **c**ontexto de **p**rogramação	5.3
			JDOC	***Java*doc**	5.4
			PC + ICP	Combinação de **p**alavras-**c**haves e **i**nformações no **c**ontexto de **p**rogramação	5.1 5.3
INT	**Int**egração do elemento selecionado	6	NA	**N**ão **a**presenta funcionalidades para adaptar o elemento apresentado no contexto de programação	—
			A	**A**presenta funcionalidades de adaptação	—

Para o *contexto de uso*, há uma dicotomia entre as ferramentas integradas a uma IDE e as ferramentas web. Ferramentas integradas a uma IDE podem explorar o contexto de programação da IDE em que o usuário está desenvolvendo um software, a fim de extrair informações que servem como entrada ou que contribuam para a entrada da ferramenta *CST*. Além disso, ferramentas integradas a uma IDE possuem uma interface de interação mais próxima da tarefa que o programador esteja trabalhando no momento, evitando dessa forma a troca de contexto, pois

[10] *ACE - **A**jax.org **C**loud9 **E**ditor*, IDE para desenvolvimento de arquivos JavaScript, HTML e CSS <https://ace.c9.io/>.

Tabela 3: Ferramentas e suas respectivas características

Nome da Ferramenta	CONT	FDADOS	ENT	INT
Prospector [6]	IDE	PCA	ICP	NA
Sourcerer [7]	WEB	PCA	IED	NA
Strathcona [8]	IDE	PCA	ICP	NA
XSnippet [9]	IDE	PCA	ICP	NA
Mica [10]	WEB	PW	PC	NA
Assieme [11]	WEB	PW	PC	NA
MAPO [12]	IDE	PCA	ICP	NA
Blueprint [13]	IDE	PW	PC + ICP	NA
eXoaDocs [14]	WEB	PCA	JDOC	NA
Redprint [15]	IDE	PW	ICP	NA
PropER-Doc [16]	—	PCA	IED	NA
Fishtail [17]	IDE	PW	ICP	NA
APIExample [18]	WEB	PW	IED	NA
Codelets [19]	IDE	TCCM	PC	A
SnipMatch [20]	IDE	TCCM	PC + ICP	A
APIMiner [21]	WEB	PCA	JDOC	NA
UsETeC [22]	WEB	TU	JDOC	NA
SPARS-J [23]	WEB	PCA	IED	NA
Krugle [24]	WEB	PCA	IED	NA
Open Hub Code Search [25]	WEB	PCA	IED	NA

Fonte: elaborado pelo autor

as ações do usuário são realizadas dentro da própria IDE. De outra forma, as ferramentas web são independentes arquiteturalmente das IDEs, isso possibilita que usuários de diferentes IDEs utilizem a mesma ferramenta *CST*. Ainda, o usuário de uma ferramenta web pode aproveitar de todo um ambiente rico de interação proporcionado pelos navegadores web.

Somente uma ferramenta dentre as ferramentas analisadas, a ferramenta *UsETeC* [22], extrai trechos de código a partir de testes unitários. A utilização dos testes unitários como uma fonte de dados para extrair trechos de código para uma ferramenta *CST* é uma estratégia promissora, sobretudo se for considerado os argumentos apresentados em [38] e os resultados empíricos apresentados em [37]. No entanto, muitos são os desafios para extrair trechos de códigos que sirvam de exemplos da utilização de uma API. Novas heurísticas devem ser criadas com o objetivo de realizar estudos comparativos e obter melhores algoritmos para a extração de trechos de código a partir dos teste unitários.

Para a integração do trecho de código selecionado no contexto do projeto do software sendo desenvolvido, foi possível identificar que não são todas as ferramentas que apresentam facilidades para a adaptação do trecho de código selecionado pelo usuário no contexto de programação (após o usuário ter realizado uma busca pelo trecho de código desejado). Adaptações necessárias podem incluir a renomeação de variáveis, inclusão de dependências e a configuração de parâmetros. Ferramentas que extraem trechos automaticamente de uma fonte de dados, como a partir de páginas web ou a partir de projetos de código aberto disponíveis na Internet, não apresentam facilidades para adaptar um trecho de código selecionado pelo usuário. De outro modo, ferramentas que trabalham com *templates* de código, trechos de código cadastrados manualmente contendo diretivas de integração, apresentam funcionalidades para auxiliar o usuário, como a inclusão automática de dependência, a renomeação de variáveis já declaradas e a sugestão de argumentos para os parâmetros do *template*. No entanto, como os *templates* de código são cadastrados manualmente, o custo de criação e manutenção desses *templates* pode não justificar a vantagem de utilizá-los.

8 Conclusões

Neste artigo, foi discorrido sobre as características das ferramentas *CSTs - Code Snippets Tools* - apresentadas nos trabalhos analisados. As características foram abordadas em relação a quatro aspectos: o *contexto de uso*, que é o ambiente onde a ferramenta está inserida; a fonte de dados de onde a ferramenta extrai os trechos de código a serem apresentados ao usuário; a entrada utilizada pela ferramenta para decidir quais trechos de código são relevantes ao usuário e as facilidades oferecidas pelas ferramentas a fim de auxiliar a adaptação de um trecho de código selecionado pelo usuário no contexto do projeto do software em desenvolvimento, após o usuário ter realizado uma busca pelo trecho de código desejado.

Trabalhos futuros poderão explorar técnicas, a fim de reduzir a dependência arquitetural das ferramentas *CSTs* integradas a uma IDE, como por exemplo a solução sugerida em [17, p. 50], que é a criação de uma arquitetura modularizada tal que os módulos que não necessitem serem acoplados à IDE podem ter o código fonte reutilizado entre IDEs diferentes e somente módulos que sejam acoplados à IDE possuem uma versão para cada IDE. Ainda, a estratégia adotada em [29], que implementa um canal de comunicação entre a IDE *Eclipse* e o navegador *Firefox*, também poderá ser explorada para as ferramentas *CSTs*, a fim de utilizar as informações presentes no contexto de programação na IDE e ao mesmo tempo aproveitar os recursos de interação fornecidos pelo navegador web. Isso criaria uma solução convergente entre as ferramentas integradas a uma IDE e as ferramentas web.

Para a integração do trecho de código, trabalhos futuros poderão explorar estratégias que auxiliem a criação e a manutenção desses trechos de código cadastrados manualmente. Apenas como um exemplo, uma ferramenta poderia, a partir de páginas web ou a partir de projetos de código aberto, extrair e apresentar trechos de código ao usuário. O usuário poderia, então, por meio do auxílio da ferramenta, adicionar diretivas de integração a um determinado trecho de código selecionado, o qual se tornaria um *template* de código cadastrado em um repositório, o qual poderia ser utilizado no ambiente de programação. Uma outra solução para os trechos de códigos cadastrados manualmente seria recorrer ao *Crowdsourcing* [41]. A ferramenta *CST* poderia possibilitar a criação de trechos de código públicos em uma arquitetura cliente-servidor. Um trecho de código público cadastrado na ferramenta por um usuário seria acessível a todos os outros usuários da ferramenta, maximizando, dessa forma, a quantidade de trechos de código disponíveis para cada usuário. Embora essa estratégia tenha sido explorada em [20], trabalhos futuros poderão explorar essa estratégia com o objetivo de apresentar resultados de estudos de casos que tragam um entendimento melhor das vantagens de utilizar o compartilhamento e a criação colaborativa de trechos de código entre os usuários de uma ferramenta *CST*.

Referências

[1] ORENSTEIN, D. *QuickStudy: Application Programming Interface (API)*. 2000. Disponível em: <http://www.computerworld.com/s/article/43487/Application_Programming_Interface>. Acesso em: 4 abr. 2016.

[2] ROBILLARD, M. P. What makes apis hard to learn? answers from developers. *Software, IEEE*, IEEE, v. 26, n. 6, p. 27–34, 2009.

[3] MCLELLAN, S. et al. Building more usable apis. *Software, IEEE*, v. 15, n. 3, p. 78–86, 1998.

[4] BRANDT, J. et al. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In: ACM. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. [S.l.], 2009. p. 1589–1598.

[5] HOLMES, R. et al. The end-to-end use of source code examples: an exploratory study. In: *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. [S.l.: s.n.], 2009. p. 555–558.

[6] MANDELIN, D. et al. Jungloid mining: helping to navigate the api jungle. *ACM SIGPLAN Notices*, ACM, v. 40, n. 6, p. 48–61, 2005.

[7] BAJRACHARYA, S. et al. Sourcerer: a search engine for open source code supporting structure-based search. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*. New York, NY, USA: ACM, 2006. (OOPSLA '06), p. 681–682.

[8] HOLMES, R.; WALKER, R.; MURPHY, G. Approximate structural context matching: an approach to recommend relevant examples. *Software Engineering, IEEE Transactions on*, v. 32, n. 12, p. 952–970, Dec 2006.

[9] SAHAVECHAPHAN, N.; CLAYPOOL, K. Xsnippet: mining for sample code. In: *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*. New York, NY, USA: ACM, 2006. (OOPSLA '06), p. 413–430.

[10] STYLOS, J.; MYERS, B. et al. Mica: a web-search tool for finding api components and examples. In: IEEE. *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*. [S.l.], 2006. p. 195–202.

[11] HOFFMANN, R.; FOGARTY, J.; WELD, D. S. Assieme: finding and leveraging implicit references in a web search interface for programmers. In: ACM. *Proceedings of the 20th annual ACM symposium on User interface software and technology*. [S.l.], 2007. p. 13–22.

[12] ZHONG, H. et al. Mapo: mining and recommending api usage patterns. In: *ECOOP 2009–Object-Oriented Programming*. [S.l.]: Springer, 2009. p. 318–343.

[13] BRANDT, J. et al. Example-centric programming: integrating web search into the development environment. In: ACM. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. [S.l.], 2010. p. 513–522.

[14] KIM, J. et al. Towards an intelligent code search engine. In: *AAAI*. [S.l.: s.n.], 2010.

[15] BHARDWAJ, A. P.; LUCIANO, D.; KLEMMER, S. R. Redprint: integrating api specific instant example and instant documentation display interface in ides. In: ACM. *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology*. [S.l.], 2011. p. 21–22.

[16] MAR, L. W.; WU, Y.-C.; JIAU, H. Recommending proper api code examples for documentation purpose. In: *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*. [S.l.: s.n.], 2011. p. 331–338.

[17] SAWADSKY, N.; MURPHY, G. C. Fishtail: from task context to source code examples. In: *Proceedings of the 1st Workshop on Developing Tools As Plug-ins*. New York, NY, USA: ACM, 2011. (TOPI '11), p. 48–51.

[18] WANG, L. et al. Apiexample: an effective web search based usage example recommendation system for java apis. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.], 2011. p. 592–595.

[19] ONEY, S.; BRANDT, J. Codelets: linking interactive documentation and example code in the editor. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2012. (CHI '12), p. 2697–2706.

[20] WIGHTMAN, D. et al. Snipmatch: using source code context to enhance snippet retrieval and parameterization. In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: ACM, 2012. (UIST '12), p. 219–228.

[21] MONTANDON, J. E. et al. Documenting apis with examples: lessons learned with the apiminer platform. In: IEEE. *Reverse Engineering (WCRE), 2013 20th Working Conference on*. [S.l.], 2013. p. 401–408.

[22] ZHU, Z. et al. Mining api usage examples from test code. In: IEEE. *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. [S.l.], 2014. p. 301–310.

[23] THE SPARS PROJECT & OSAKA UNIVERSITY. *SPARS-J*. 2016. Disponível em: <http://demo.spars.info/j/>. Acesso em: 4 abr. 2016.

[24] ARAGON CONSULTING GROUP. *Krugle*. 2016. Disponível em: <http://www.krugle.com/>. Acesso em: 4 abr. 2016.

[25] BLACK DUCK SOFTWARE. *Open Hub Code Search*. 2016. Disponível em: <http://code.openhub.net/>. Acesso em: 4 abr. 2016.

[26] BAEZA-YATES, R.; RIBEIRO-NETO, B. *Modern information retrieval: the concepts and technology behind search*. [S.l.]: Addison Wesley, 2011.

[27] ECLIPSE FOUNDATION. *Eclipse Kepler*. 2016. Disponível em: <http://www.eclipse.org/ide/>. Acesso em: 4 abr. 2016.

[28] ORACLE CORPORATION. *Netbeans 8.0*. 2016. Disponível em: <http://netbeans.org>. Acesso em: 4 abr. 2016.

[29] GOLDMAN, M.; MILLER, R. Codetrail: connecting source code and web resources. In: *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*. [S.l.: s.n.], 2008. p. 65–72.

[30] YE, Y.; FISCHER, G.; REEVES, B. Integrating active information delivery and reuse repository systems. In: ACM. *ACM SIGSOFT Software Engineering Notes*. [S.l.], 2000. v. 25, n. 6, p. 60–68.

[31] DESHPANDE, A.; RIEHLE, D. The total growth of open source. In: *Open Source Development, Communities and Quality*. [S.l.]: Springer, 2008. p. 197–209.

[32] BAJRACHARYA, S. et al. Sourcerer: a search engine for open source code. In: *International Conference on Software Engineering (ICSE 2007)*. [S.l.: s.n.], 2007.

[33] BAJRACHARYA, S.; OSSHER, J.; LOPES, C. Sourcerer: an infrastructure for large-scale collection and analysis of open-source code. *Science of Computer Programming*, v. 79, p. 241 – 259, 2014. Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010).

[34] HAN, J.; KAMBER, M. *Data Mining: concepts and techniques*. Second. 500 Sansome Street, Suite 400, San Francisco, CA 94111: Morgan Kaufmann Publishers, 2006.

[35] GHAFARI, M. et al. Mining unit tests for code recommendation. In: *Proceedings of the 22Nd International Conference on Program Comprehension*. New York, NY, USA: ACM, 2014. (ICPC 2014), p. 142–145.

[36] OSHEROVE, R. *The art of unit testing: with examples in C#*. [S.l.]: Manning Publications Company, 2013.

[37] NASEHI, S.; MAURER, F. Unit tests as api usage examples. In: *Software Maintenance (ICSM), 2010 IEEE International Conference on*. [S.l.: s.n.], 2010. p. 1–10.

[38] GHAFARI, M. Extracting code examples from unit test cases. In: IEEE. *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. [S.l.], 2014. p. 667–667.

[39] HENNINGER, S. Retrieving software objects in an example-based programming environment. In: ACM. *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*. [S.l.], 1991. p. 251–260.

[40] KIM, M. et al. An ethnographic study of copy and paste programming practices in oopl. In: *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on*. [S.l.: s.n.], 2004. p. 83–92.

[41] BRABHAM, D. *Crowdsourcing*. [S.l.]: MIT Press, 2013.