PUCRS

FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

LUANA MÜLLER

UMA ABORDAGEM SEMIÓTICA PARA APOIAR PROGRAMADORES INICIANTES DURANTE O PROCESSO DE REÚSO E DE APROPRIAÇÃO DE CÓDIGOS-FONTE

Porto Alegre
2017

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

Ficha Catalográfica

M999a Müller, Luana

Uma abordagem semiótica para apoiar programadores iniciantes durante o processo de reúso e de apropriação de códigos-fonte / Luana Müller . – 2017.

114 f.

Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientadora: Profa. Dra. Milene Selbach Silveira.
Co-orientadora: Profa. Dra. Clarisse Sieckenius de Souza.

1. reúso de código-fonte. 2. programadores iniciantes. 3. apropriação. 4. Engenharia Semiótica. I. Silveira, Milene Selbach. II. de Souza, Clarisse Sieckenius. III. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecário responsável: Marcelo Votto Texeira CRB-10/1974

Luana Müller

Uma abordagem semiótica para apoiar programadores iniciantes durante o processo de reúso e de apropriação de códigos-fonte

Tese apresentada como requisito parcial para obtenção do grau de Doutor em Ciência da Computação do Programa de Pós-Graduação em Ciencia da Computação, Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 8 de dezembro de 2017

BANCA EXAMINADORA:

Prof. Dr. André Luís Alice Raabe (UNIVALI)

Prof. Dr. Alberto Barbosa Raposo (DI/PUC-Rio)

Prof. Dr. Márcio Sarroglia Pinho (PPGCC/PUCRS)

Profa. Dra. Milene Selbach Silveira (PPGCC/PUCRS - Orientadora)

Profa. Dra. Clarisse Sieckenius de Souza (DI/PUC-Rio – Co-Orientadora)

AGRADECIMENTOS

Apesar de estarem no começo deste documento, estas palavras são as últimas palavras escritas nesta Tese, e representam assim, o fechamento deste ciclo, iniciado em 2011, quando através do mestrado iniciei minha vida na pós-graduação. Este ciclo foi de aprendizado, reflexão e crescimento (pessoal e profissional), e seu encerramento deve-se ao apoio de muitas pessoas, as quais expresso aqui, meus agradecimentos.

Em primeiro lugar, agradeço as minhas queridas orientadoras, Milene Selbach Silveira e Clarisse Sieckenius de Souza. Milene, obrigada por tudo que tu me proporcionaste durante estes muitos anos de orientações. O valor que agregaste em minha vida acadêmica e pessoal é imensurável. Obrigada pelos muitos "balões", pelos divertidos almoços e cafés, pelos inúmeros conselhos, e por toda tua dedicação, carinho e amizade. Clarisse, obrigada por ter me acolhido junto ao teu grupo, obrigada pelo apoio e carinho, e obrigada por ter me dado a honra de receber tuas orientações, conselhos, ideias. Serás sempre uma grande inspiração para mim.

Ao meu "noivarido", Bernardo, obrigada pelo amor, apoio, paciência e carinho, e aos meus pais, Claudir e Marlene, obrigada por sempre me apoiarem e acreditarem em mim.

Obrigada em especial às queridas Angelina Ziesemer, Caroline Queiroz e Luciana Espindola, por todas as ideias, contribuições, bolos e cafés e, a teacher/friend Carolina Paz, por todo o apoio com revisões, traduções e ombro-amigo.

E obrigada a todos os demais colegas e amigos que me acompanharam neste período: agradeço à Lucia Giraffa, sempre disposta a me ajudar e me apoiar em todos os momentos e situações; ao João Batista, pela ajuda com as ferramentas de análise de similaridade e pelas muitas ideias, aos demais ex-colegas e professores da PUCRS, sempre apostos e dispostos; e aos colegas de laboratório, em especial a Daniele Souza, pela amizade, companheirismo, e pelas muitas caronas. Agradeço também aos amigos Vini Cassol e Marlon Oliveira, pelas muitas "figurinhas" trocadas sobre a vida de pós-graduando.

Obrigada também a todos que participaram dos estudos conduzidos durante este trabalho, pelo tempo e atenção disponibilizados em prol desta pesquisa.

E por fim (mas não menos importante), obrigada PUCRS por todos estes anos de acolhida.

UMA ABORDAGEM SEMIÓTICA PARA APOIAR PROGRAMADORES INICIANTES DURANTE O PROCESSO DE REÚSO E DE APROPRIAÇÃO DE CÓDIGOS-FONTE

RESUMO

Durante o processo de desenvolvimento de *software*, a prática de reúso é comumente utilizada por diversos motivos e de diversas formas. Dentre estas formas, destaca-se a cópia de trechos de códigos-fonte, desenvolvidos por outros programadores, que são incorporados a um novo código-fonte e adaptados para se adequar ao novo contexto. No entanto, apesar de o programador conseguir atingir desta forma seus objetivos, muitas vezes ele realiza este reúso sem compreender o código-fonte reusado, e, por consequência, sem compreender seu próprio código-fonte, criado com reúso. Embasando-nos na teoria da Engenharia Semiótica, sob a perspectiva da área de *Human-Centered Computing*, observamos este código-fonte como uma *interface* que entrega ao seu usuário (neste caso, o programador que está reusando um código-fonte) uma mensagem implícita do *designer* (neste caso, o programador que escreveu o código-fonte sendo reusado) e observamos que podem haver diversas implicações comunicativas decorrentes da falta de compreensão do *designer* sobre a mensagem que está sendo emitida. Essas implicações fazem-se ainda mais presentes no caso dos programadores iniciantes, que ainda estão construindo seu pensamento computacional. Nesta tese, conduzimos uma investigação sobre como e porque programadores iniciantes reusam códigos-fonte e como interpretam seus códigos-fonte quando estes foram construídos reusando outros códigos-fonte. Além disso, oferecemos uma ferramenta conceitual baseada no *template* de metacomunicação da Engenharia Semiótica para apoiar estes programadores na compreensão e apropriação de trechos de códigos-fonte para reúso. Observamos, através dos estudos para análise da ferramenta, que ela pode não apenas apoiar os programadores na compreensão dos códigos-fonte, mas, também, em sua reflexão sobre questões não comumente pensadas, relacionadas a aspectos comunicativos destes códigos. Esperamos que esta reflexão os conscientize da importância destes aspectos, em um cenário em que seus códigos podem ser, em algum momento, reusados por outros programadores, em um ciclo contínuo de desenvolvimento e comunicação.

Palavras chave: reúso de código-fonte, programadores iniciantes, apropriação, Engenharia Semiótica

A SEMIOTIC-BASED APPROACH TO SUPPORT NOVICE PROGRAMMERS DURING THE PROCESS OF REUSE AND APPROPRIATION OF SOURCE CODES

ABSTRACT

During the process of software development, the reuse of materials is often performed in several ways and for different reasons. Among these ways, we highlight the reuse of source code developed by other programmer, which are embedded in a new source code and adapted in order to fit into this new context. However, even though programmers are able to achieve their goals by doing so, many times they reuse a source code without properly understand it, and, consequently, they do not understand their own source-code, generated through reuse. Based on Semiotic Engineering theory under the Human-Centered Computing perspective, we can see this source code as an interface which delivers to its user (in our case, the programmer who is reusing a source code) an encoded message from its designer (in our case, the programmer who developed the source code being reused), and, thus, we observe that there may have been several communicative implications resulting from the lack of the designer's understanding about the message he is delivering. Those implications are even more perceived when it comes to novice programmers, who are still building their computational thinking. In this thesis we investigated on how and why novice programmers reuse source codes and how they understand their own source codes while they build them by reusing other programmers' source codes. Furthermore, we offer a conceptual tool based on Semiotic Engineering's metacommunication template to support those programmers understanding and appropriating source codes. Besides that, we observe, through the studies conducted in order to analyze our tool, that not only can it support programmers understanding a source code, but also support them reflecting upon questions not usually addressed by them, related, for instance, to communicative aspects of their source codes. We hope this reflection makes programmers aware of the importance of those aspects, considering a scenario where their source codes may be reused by other programmers, in a continuous cycle of development and communication.

Keywords: source code reuse, novice programmers, appropriation, Semiotic Engineering

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	Application Programming Interface
CMC	Comunicação Mediada por Computadores
EUD	*End-User Development*
IHC	Interação Humano-Computador
HCC	*Human Centered Computing*

SUMÁRIO

1 INTRODUÇÃO

Aprender a programar pode ser uma tarefa difícil, seja no papel de um programador iniciante, que almeja esta atividade como sua profissão no futuro, ou no papel de usuário final que visa aprender a programar para conseguir atingir seus objetivos pessoais. Em ambos os casos, é preciso enfrentar as dificuldades de se aprender a pensar computacionalmente (Wing, 2006), ou seja, aprender a formular problemas e projetar suas soluções de uma forma que computador seja capaz de executar, tendo como características a capacidade de decomposição e abstração de problemas, reconhecimento de padrões, pensamento recursivo, além de outras. Kelleher e Pausch (Kelleher & Pausch, 2005) citam que, além de aprender a criar soluções estruturadas para seus problemas e compreender como programas são executados, programadores iniciantes também precisam lidar com a sintaxe das linguagens de programação e os seus comandos, tendo dificuldade em compreender como traduzir para o computador as suas intenções. Durante esse processo de aprendizado, programadores iniciantes usam frequentemente exemplos de códigos-fonte que possam apoiá-los de algum modo, e muitas vezes reusam este código-fonte (Neal, 1989), incorporando-o ao seu próprio código, fazendo (ou não) as readequações necessárias para que este atinja seus objetivos.

Tanto programadores iniciantes quanto usuários finais que atuam com programação, em algum nível, enfrentam as dificuldades que a pouca experiência com programação lhes causa. Desta forma, observamos que é possível enxergar estes dois grupos como um só, e usar a literatura de ambos os casos para compreender melhor como reúso de código-fonte ocorre (Grigoreanu, et al., 2012).

Aos usuários finais que atuam com desenvolvimento de *software*, dá-se o nome de *End-User Developers.* Tal nomenclatura vem do termo *End-User Development* (EUD), que refere-se ao conjunto de métodos, técnicas e ferramentas que permitem aos usuários criarem, modificarem ou estenderem *software* (Lieberman, et al., 2006) (De Souza, 2017) e, normalmente, refere-se a desenvolvedores não profissionais (Ko, et al., 2011) (Segal, 2007) (Costabile, et al., 2008). Porém qualquer um que esteja desenvolvendo um *software* está, de fato, atuando como um usuário final de outros artefatos de *software*, tais como ambientes de desenvolvimento (IDEs), linguagens de programação, APIs, *frameworks*, entre outros (De Souza, et al., 2016) (Myers, et al., 2016).

Neste trabalho queremos estudar programadores como usuários de códigos-fonte, em particular códigos-fonte escritos por outros programadores e que podem ser reusados. Queremos estudar este reúso sob a perspectiva na qual computadores são vistos como uma forma de mídia (Kammersgaard, 1988) (De Souza, 2005) (Georgakopoulou, 2011), pela qual pessoas podem se comunicar umas com as outras. No processo de comunicação humana usamos signos[1]. Estes signos podem se apresentar na forma de palavras, gestos, símbolos, sons, imagens, entre outros, que são usados pelo emissor da mensagem para expressar algo ao receptor desta, o qual interpreta esses signos para entender o significado desta mensagem. Na comunicação mediada por computadores (CMC), essa comunicação pode ocorrer através dos signos presentes na interface de um sistema: por meio destes signos o *designer* comunica aos usuários suas intenções, cabendo a este usuário interpretar estes signos e tentar desta forma compreender o significado da mensagem do *designer*, fenômeno este nomeado como metacomunicação, postulado pela teoria da Engenharia Semiótica (De Souza, 2005).

De modo a levar essa discussão para além da interface tradicional, neste caso, para dentro do ambiente de programação, nos baseamos na área de *Human-Centered Computing* (HCC)[2] (Jaimes, et al., 2007) (Bannon, 2011) (Choi, 2016). Esta área pesquisa o *design,* desenvolvimento e a implantação de diversas iniciativas que envolvem a interação entre computadores e pessoas, e, desta forma, abrange uma série de metodologias que se aplicam a qualquer campo onde pessoas interagem diretamente com artefatos computacionais (Jaimes, et al., 2007).

A área de HCC é multidisciplinar, abrangendo diversas outras áreas de conhecimento além da Ciência da Computação, tais como Sociologia, Psicologia, Ciências Cognitivas, Design Gráfico, Design Industrial, dentre outras. A construção de ferramentas computacionais focada em pessoas foca em problemas que usualmente a área de Interação Humano-Computador (IHC) não endereça, uma vez que HCC não trata somente da interação, da interface e do processo *design*, mas também trata de conhecimento, pessoas, tecnologia e tudo que os une (Jaimes, et al., 2007). Observando esta

[1] Signos são algo que em determinado aspecto ou modo, representa algo para alguém (Peirce, 1931-1958).

[2] Em português, a área é chamada de Computação Focada em Pessoas. No entanto, usaremos o termo em inglês por este ser mais amplamente difundido.

comunicação do ponto de vista da *HCC* em conjunto com ao conceito de metacomunicação postulada pela Engenharia Semiótica*,* podemos abordar essa tradicional comunicação mediada por computadores através de uma nova perspectiva: assim como no processo de interação entre usuário e interface, no qual ocorre a metacomunicação entre este usuário e o *designer* da referida interface*,* no processo de interação entre um programador e um código-fonte esta metacomunicação ocorre entre este programador e o *designer* do código-fonte que está sendo reusado. O código-fonte, neste caso, atua como interface (Myers, et al., 2016), mediando a comunicação entre estes programadores através de mensagens implícitas no código-fonte. Neste trabalho usaremos a Engenharia Semiótica como teoria norteadora, teoria esta originalmente proposta dentro do contexto de IHC, e que atualmente vem se destacando na área de HCC (De Souza, et al., 2016) e em pesquisas em relação à área de EUD (De Souza, 2017), e cuja visão em relação a CMC se alinha com o tema aqui apresentado.

Assim como uma interface tradicional, da qual devemos nos apropriar para fazer melhor uso da mesma (Carroll, et al., 2002), acreditamos que a interface representada pelo código-fonte de um programa, também deve ser compreendida e apropriada pelo programador que pretende de alguma forma (Afonso, 2015) (Myers, et al., 2016) fazer uso deste código, seja para mantê-lo, estendê-lo ou reusá-lo. Portanto, é necessário apoiar esses programadores no processo de apropriação de código-fonte, considerando que isto o levará a um melhor uso do mesmo.

Neste trabalho queremos atuar de forma a ampará-los durante o processo de reúso, ajudando-os a se apropriar deste código-fonte e da mensagem que este emite. Para tal, visamos investigar como os programadores (re)usam códigos de outros programadores quando estão desenvolvendo seus próprios *softwares*. Desejamos observar também como esse reúso de códigos ocorre no grupo de programadores que ainda estão desenvolvendo seu raciocínio computacional, fase esta em que ainda não adquiriram hábitos sólidos de programação.

1.1 Objetivos

Neste trabalho temos como objetivo principal ajudar programadores iniciantes a compreender o código-fonte que estão reusando para, desta forma, poderem fazer um uso mais adequado do mesmo.

Como objetivos secundários, visamos:

- Compreender como programadores iniciantes reusam códigos-fonte de outros desenvolvedores durante o processo de desenvolvimento de um *software*;
- Compreender como que o reúso afeta a compreensão que estes programadores têm sobre o funcionamento de seus programas, observando se houve apropriação dos códigos-fonte reusados;
- Oferecer uma ferramenta conceitual de apoio à apropriação, pela qual os programadores possam posicionar-se como receptores de uma comunicação feita por outro programador através do código (mensagem) que têm diante de si.

1.2 Metodologia

Tendo como intuito compreender os fenômenos relacionados ao problema em questão de forma aprofundada, nesta pesquisa foi utilizada uma abordagem qualitativa. De acordo com Creswell (Creswell, 2014), a pesquisa qualitativa é um meio para explorar e compreender o significado que um indivíduo ou grupo atribui para um problema humano ou um problema social, dependendo ao máximo da visão dos participantes sobre a situação que está sendo estudada.

Foram conduzidos cinco estudos, envolvendo a participação de programadores iniciantes nos quatro primeiros e de professores de programação no quinto estudo, pelos quais coletamos dados sobre seus hábitos, técnicas e impressões em relação a reúso de código e programação.

Em relação aos estudos 1, 2 e 3, estes foram realizados com o intuito de aprofundar nossos conhecimentos em relação ao problema em questão. Através das informações coletadas através destes estudos, juntamente às referências teóricas e trabalhos relacionados utilizados nesta pesquisa, foi possível delimitar o problema e traçar nossa proposta em relação a como apoiar programadores iniciantes durante o processo de reúso de código-fonte. O detalhamento sobre os estudos iniciais é apresentado no Capítulo 343.

Os estudos 4 e 5 (apresentados na seção 4.2), por sua vez, foram realizados com intuito de coletar opiniões sobre a ferramenta proposta nesta pesquisa. Durante o Estudo 4, programadores iniciantes foram convidados a utilizar a ferramenta proposta e esboçarem suas opiniões sobre a mesma, e durante o Estudo 5 coletamos opiniões de professores de programação sobre a aplicabilidade da ferramenta durante o processo de ensino de programação.

A coleta de dados foi feita por meio de materiais fornecidos pelos participantes, entrevistas (em todos os estudos) e um grupo de foco. A partir destes dados foi feito um levantamento de indicadores qualitativos relacionados às questões abordadas em cada um destes estudos.

Para uma melhor compreensão de como os estudos ocorreram e como se relacionam com as demais etapas desta pesquisa, a Figura 1 ilustra todas as etapas deste trabalho.

Figura 1: Etapas da pesquisa

1.3 Organização do Trabalho

Este trabalho está organizado em seis capítulos, sendo, o primeiro, a Introdução aqui apresentada. No capítulo 2, apresentamos o Referencial Teórico deste trabalho composto pelos temas de reúso de *software* e exemplos de código-fonte, apropriação, Semiótica e Engenharia Semiótica e também os trabalhos relacionados. A seguir, no capítulo 3, apresentamos os três estudos conduzidos com intuito de compreender o problema em questão. No capítulo 4, apresentamos nossa proposta, uma ferramenta conceitual para apoio à compreensão e apropriação de código-fonte, assim como o relato sobre os dois estudos conduzidos para analisá-la. O capítulo 5 apresenta nossa discussão sobre os resultados dos estudos conduzidos para análise da ferramenta, a importância de ferramentas que apoiem a reflexão, e sobre as consequências esperadas desta

reflexão. Por fim, apresentamos as Considerações Finais (capítulo 6), onde são apresentadas nossas contribuições, limitações, e nossos próximos passos e sugestões de trabalhos futuros, seguidos da relação de Referências utilizadas neste trabalho, Anexos e Apêndices.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentadas nossas bases teóricas, ou seja, conceitos fundamentais para a compreensão deste trabalho. Além disso, apresentaremos trabalhos que de alguma forma endereçam os problemas relacionados ao reúso e apropriação de código-fonte, como ele ocorre quando é feito por programadores iniciantes que estão desenvolvendo programas, seja com o objetivo de aprender programação, seja com o objetivo de aprender a solucionar problemas de seu dia-a-dia com programação.

2.1 Reúso de *software* e exemplos de código-fonte

Krueger (Krueger, 1992) define reúso de *software* como o processo de criar *software* a partir de um *software* existente ao invés de construí-lo do zero. Ao reusar *software,* os programadores estão reusando também o conhecimento incorporado no artefato que foi previamente desenvolvido, podendo, ou não, este ter sido desenvolvido com o objetivo de ser reusado (Sojer, 2011). O artefato de *software* mais comumente reusado é o código-fonte, porém outros artefatos como *designs*, arquiteturas, planos de projeto, requisitos, cenários de teste, interfaces e documentação, também podem ser reusados.

Segundo Hoadley (Hoadley, et al., 1996), o reúso de *software* pode ocorrer de três maneiras distintas: (1) Invocação de código (*code invocation*), que ocorre quando há o reúso funções e procedimentos; (2) Clonagem de código (*code cloning*), que ocorre quando são copiadas as linhas de programas-exemplo, modificando o código para adequar-se à especificação do problema; e (3) Reúso de *templates*, que ocorre quando são aplicados padrões aprendidos. Sojer (Sojer, 2011) classifica o reúso de código de uma forma diferente, porém complementar (Figura 2) a classificação anteriormente citada. Para o autor o reúso pode acontecer de duas maneiras diferentes: (1) Reúso de fragmentos (*snippet reuse*) e (2) Reúso de componentes (*componente reuse*). Na primeira abordagem, programadores procuram por fragmentos de código em um *software* existente e os usam em seus novos *softwares.* Dentro desta abordagem, o reúso de código pode ocorrer quando múltiplas e contínuas linhas de código são reproduzidas (*code scavenging*), ou quando uma estrutura composta por um grande bloco de código é usada como uma estrutura, sendo que neste caso vários detalhes são removidos, mas o modelo de *design* é mantido (*design scavenging*). Já a

segunda abordagem, classifica o reúso de componentes que foram explicitamente desenvolvidos para este propósito, e que além de codificados, foram testados, documentados e, algumas vezes, certificados. Dentro de reúso de componentes, uma das formas que os programadores têm para interagir com estes componentes, é através de sua *Application Programming Interface* (API).

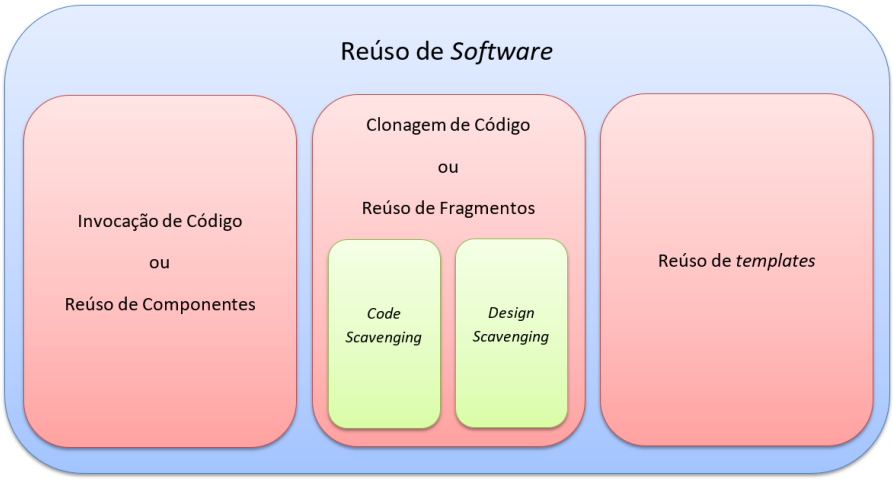

Figura 2: Formas de reúso de *software* segundo Hoadley (Hoadley, et al., 1996) e Sojer (Sojer, 2011)

Porém, muitas vezes os programadores não estão familiarizados ou não se recordam como utilizar funções provenientes destas APIs. Para aprender sobre estas funções e como interagir com elas, eles comumente usam a documentação oficial da API, fóruns, ou outras fontes de informações, sendo que, em algumas destas fontes, eles encontram fragmentos de código fontes que servem como exemplo para ampará-los no uso (Moreno, et al., 2015). Neste trabalho nosso principal tópico de interesse está particularmente em reúso de código-fonte, não abordando reúso de *templates.* Desta forma, usaremos no decorrer deste trabalho as nomenclaturas de Sojer (Sojer, 2011) por apresentar uma especificação mais detalhada sobre como a clonagem de código pode ocorrer.

De forma geral, observa-se que exemplos são importantes (senão essenciais) no processo de aprendizagem. No processo construtivo do aprendizado, estudantes convertem palavras e exemplos em habilidades, como a resolução de problemas (Chi, et al., 1989). Durante o processo de ensino e aprendizagem de programação, os exemplos podem ser usados com diversos propósitos, tais como apresentar uma estrutura da linguagem, como implementar um algoritmo para solucionar um problema, ou demonstrar um padrão/estilo de programação (Neal, 1989). Além do propósito direto de ilustrar/demonstrar um novo conceito, exemplos frequentemente servem para o propósito adicional de “vender” o conceito. Segundo Malan e Halland (Malan & Halland, 2004), o

estudante deve compreender a importância do conceito ou simplesmente continuará a programar sem aplicá-lo ao seu código. Alguns estudantes de programação iniciam a solução de um problema identificando palavras-chave na especificação do programa que possam ser relacionadas com exercícios previamente feitos e resolvidos (Gaspar & Langevin, 2007). Complementar aos exercícios prévios, a internet atualmente oferece aos programadores uma vasta gama de conteúdos que podem ser facilmente acessados e que podem conter exemplos que correspondam muito similarmente (quando não totalmente) às intenções do programador.

O uso de exemplos mantém-se constante na vida de profissionais de programação. O trabalho de Neal (Neal, 1989) reporta que programadores com diversos níveis de experiência programam estudando, reusando ou revisando programas (ou fragmentos de programas) que outros programadores escreveram previamente. No entanto, este exemplo fornecido para auxiliar o programador no entendimento de determinado conceito é reusado muitas vezes sem haver um completo entendimento sobre o que o mesmo faz (Maalej, et al., 2014), caso que pode ser observado tanto quando há um reúso de fragmento de código, quanto quando há o reúso de um componente (sendo que neste caso, o programador reúsa um exemplo que instrui como interagir com tal componente).

Neste trabalho, independente do fim ao qual o código-fonte foi destinado (para ser reusado, como um componente, para exemplificar um conceito, como um exemplo de código-fonte, ou outros fins diversos), nosso interesse é compreender como programadores iniciantes os utilizam, se e como extraem dele seus significados, como incorporam isso ao seu próprio código-fonte, e por consequência, seu próprio *discurso*[3]. Em outras palavras, queremos compreender se e como programadores iniciantes se apropriam de códigos-fonte para reusá-los. Portanto, visando aprofundar nossas bases teóricas, a seção a seguir apresenta conceitos relacionados à apropriação.

[3] Nesta pesquisa usaremos o termo *discurso* para representar aquilo que está sendo falado ou escrito pelo autor usando a linguagem, sendo essa linguagem uma expressão em linguagem natural, fórmulas e comandos escritos em uma linguagem de programação ou trechos do discurso de outros autores.

2.2 Apropriação

Apropriação pode ser definida, da perspectiva sociocultural, como o processo de pegar algo que pertence a outro(s) e fazer disso seu (Wertsch, 1998). Do ponto de vista tecnológico, apropriação é definida como a forma como usuários avaliam e adotam, adaptam e integram uma tecnologia em suas práticas diárias (Carroll, et al., 2002). Para Stevens, Pipek e Wulf (Stevens, et al., 2009), apropriação de tecnologias deve ser interpretada não somente como um fenômeno que, de alguma forma, ocorre quando o *software* está atuando em seu domínio, mas também como uma rede de atividades realizadas continuamente por usuários para fazer este *software* funcionar em um novo ambiente, tomando este artefato como material e também como um objeto significativo.

Dourish (Dourish, 2003) diz que que a apropriação é similar à customização, mas refere-se à adoção de padrões de tecnologia e à transformação em um nível mais profundo. Apropriação envolve customização (que é a reconfiguração explícita de tecnologias para adequarem-se a determinadas necessidades), mas também pode simplesmente envolver fazer uso de uma tecnologia para outros propósitos, além daqueles para os quais ele foi desenvolvido.

Carroll e coautores (Carroll, et al., 2002) definem em seu trabalho um Modelo de Apropriação Tecnológica (MTA – *Model of Technology Appropriaton*). De acordo com os autores, da mesma forma que a tecnologia é capaz de moldar as práticas dos usuários, ela também é moldada por eles. O modelo apresentado é composto por três níveis, iniciando-se no momento em que os usuários são apresentados a esta nova tecnologia e enfrentam a decisão de usá-la ou não. Após decidirem pela adoção desta tecnologia, os usuários entram em um nível mais aprofundado de uso, sendo este, o processo de apropriação em si, no qual eles testam, avaliam e adaptam a tecnologia, adequando-a as suas necessidades. Por fim, o último nível ocorre quando os usuários integraram esta tecnologia em suas práticas e ela é considerada estável.

Dentro deste âmbito, os códigos-fonte de programas também são tecnologias e, desta forma, os usuários também precisam se apropriar deles se visarem, de alguma forma, fazer uso dos mesmos. Nesta pesquisa observamos o código-fonte de um programa não apenas como palavras, escritas em uma linguagem de programação, através das quais, soluciona-se computacionalmente um problema. Nós os observamos também sob a perspectiva lançada pela Engenharia Semiótica, que considera interfaces de programas como meios que possibilitam a comunicação do *designer*

desta interface e seu usuário. Essa mensagem pode ser interpretada pelo usuário de diversas formas, formas estas que irão ditar como este usuário irá se apropriar desta interface e que tipo de uso, adaptação ou extensão, ele fará da mesma. Baseando-nos na teoria da Engenharia Semiótica (De Souza, 2005), nossa teoria norteadora, o código-fonte de um programa também é uma interface, que possibilita uma comunicação implícita entre o programador que escreveu o código-fonte que está sendo reusado, e o programador que está reusando este código-fonte (Afonso, 2015). Assim como em uma interface tradicional, a interpretação e compreensão desta mensagem irão apoiar o programador ao apropriar-se do código, possibilitando fazer um melhor uso do mesmo.

De forma a melhor compreendermos tais aspectos comunicativos e a importância de se compreender essas mensagens implícitas, na seção a seguir apresentamos conceitos relacionados à Semiótica e Engenharia Semiótica.

2.3 Semiótica e Engenharia Semiótica

A Semiótica é a ciência que estuda os signos, sendo um signo “algo que em determinado aspecto ou modo, representa algo para alguém” (Peirce, 1931-1958). Ainda, segundo a definição da Semiótica de Peirce[4], os signos são resultado da associação entre uma expressão e um conteúdo, sendo esta associação realizada por alguma mente (individual ou coletiva, humana ou não). Além disso, a Semiótica visa compreender o processo de significação, processo este que determina como os signos se constituem e ocorrem na realidade. Apropriando-se destas definições, Umberto Eco insere a Semiótica de Peirce em um contexto de cultura e comunicação social, estudando como os signos participam do processo de significação e comunicação (Eco, 1976). Durante o processo de comunicação, a maioria dos códigos usados são provenientes de sistemas de significação, sendo estes sistemas resultantes de uma convenção cultural, ou seja, um certo conjunto de elementos

[4] São tidos como precursores dos estudos sobre semiótica o linguista francês Fernand Saussure e o filósofo e lógico americano Charles Sanders Peirce. Apesar de haver pontos coincidentes em seus estudos, as teorias de ambos possuem diversas diferenças principalmente em relação a definição dos signos. Neste trabalho usamos como referência as definições de Semiótica de Peirce.

expressivos que são sistematicamente correlacionados, por uma coletividade social, a um determinado conjunto de conteúdos.

Seguindo a linha de apropriação que Eco faz de Peirce, De Souza e coautores (De Souza, 2005) (De Souza, et al., 2016) trazem esta discussão para as áreas de IHC e HCC, enfatizando que signos são associações entre uma expressão (podendo ser essa expressão computacional) e um conteúdo (podendo ser este conteúdo computacional), mediados por alguma mente (podendo esta mente ser individual ou coletiva, humana ou não). Assim, a teoria da Engenharia Semiótica trata do fato e da forma como artefatos computacionais introduzem novos signos ou sistemas de signos no universo do usuário, e como estes passam a determinar um processo de comunicação especial, mediado por computação. A teoria possibilita caracterizar e entender os fenômenos envolvidos no *design*, uso e avaliação de sistemas interativos. A interação de pessoas com artefatos computacionais é um processo de comunicação dos produtores destes artefatos (*designers* e desenvolvedores) com os usuários finais, no qual é "dito" para estes usuários como eles podem ou devem interagir com o artefato a fim de realizar uma série de objetivos ou causar uma série de efeitos possíveis, computacionalmente codificados no *design* e comportamento do artefato. Este processo de comunicação ocorre por meio da interface do sistema que, durante a interação, representa os *designers*, atuando como seu "*proxy*" e repassando aos usuários a mensagem que estes elaboraram.

Para a Engenharia Semiótica todos os artefatos computacionais são criados para expressar a visão do *designer* sobre o sistema fazendo, assim, que a Interação Humano-Computador (IHC) seja um caso de CMC. Por meio de uma interface, o *designer* envia uma mensagem ao seu usuário, sendo essa mensagem imutável e enviada em uma única vez. Essa mensagem diz ao usuário como ele deve se comunicar com a interface para atingir certos objetivos. Assim, pode-se dizer que existe uma *metacomunicação* entre *designers* e usuários, ou seja, tecnicamente há comunicação sobre a comunicação, onde os elementos da interface devem ser interpretados/decodificados pelo usuário para chegar ao significado/mensagem que está sendo expresso. Cabe ao usuário aprender e compreender a linguagem da interface, para conseguir assim comunicar ao artefato suas intenções. No caso da IHC, esta mensagem comunica ao usuário como ele pode realizar determinadas tarefas e, assim, se apropriar do sistema. A mensagem passada pelo preposto de *designer* é elaborada para "dizer" ao usuário:

"Este é o meu entendimento de quem você é, do que aprendi que você quer ou precisa fazer, de que maneiras prefere fazer, e por quê. Este, portanto, é o sistema que projetei para você, e esta é a forma como você pode ou deve utilizá-lo para alcançar uma gama de objetivos que se encaixam nesta visão".

Apesar de tradicionalmente ter sido proposta como uma teoria de IHC, a Engenharia Semiótica vem ampliando sua área de interesse e pesquisa, indo além das tradicionais interfaces, e observando aspectos comunicativos em, por exemplo, ferramentas para modelagem de *software* e APIs (De Souza, et al., 2016) (Ferreira, et al., 2015) (Bastos, et al., 2017) (Lopes, et al., 2017) (Afonso, 2015), posicionando desenvolvedores de *software* como usuários, e levando suas investigações para a área de pesquisa de HCC, estando assim alinhada com a visão que estamos usando nesta pesquisa.

Para melhor compreender o cenário envolvido, a seção a seguir apresenta trabalhos que de alguma forma se relacionam ao tema abordado nesta pesquisa.

2.4 Trabalhos relacionados

Em relação reúso de código-fonte, sabe-se que tanto os programadores iniciantes quanto os experientes os utilizam, e que este reúso pode ocorrer de diferentes maneiras (Seção 2.1). Um de nossos interesses é compreender como os programadores, com foco para o grupo de programadores com pouca experiência, aqui tratados como programadores iniciantes, compreendem, interpretam, apropriam-se (ou não) e reúsam exemplos de códigos fontes.

O trabalho de Rosson, Ballin e Nash (Rosson, et al., 2004) aborda os desafios e oportunidades da programação informal de sites realizada por programadores não profissionais, que por necessidade desenvolviam ou mantinham conteúdo on-line como parte de suas atividades diárias. Dentre entes desafios e oportunidades estão questões relacionadas ao reúso e adaptação de código proveniente de outros programadores. Os autores observaram que os participantes do estudo usaram poucas vezes a estratégia de "copiar e colar", mas usavam códigos de outras pessoas como um modelo para algo que eles estavam tentando aprender. Inclusive, um dos participantes do estudo cita a estratégia como inversa ao aprendizado, ao reportar que "Eu não gosto de copiar por que eu gosto de aprender como eu mesmo posso fazer". Os autores defendem que não se sabe se tal comportamento se estende a programadores "normais", porém que, uma vez que para estes programadores informais o código de outros serve como veículo de aprendizado, ferramentas de

desenvolvimento deveriam suportar de alguma forma perguntas em relação a como estes códigos funcionam, o que poderia facilitar o reúso, permitindo a eles copiar os códigos com maior "segurança".

Durante o estudo, diversos casos de *code-scaveging*[5] ocorreram, mas os participantes observaram que estes acarretaram problemas, tais como "bagunçar o código", uma vez que o código copiado muitas vezes não se adequava à "indentação"[6] do restante do código, ou continham elementos como espaços em branco. Um dos participantes, no entanto, descreveu um episódio de reúso que ocorreu com pleno sucesso, onde ele copiou todo o projeto de outro desenvolvedor, e expandiu e editou as páginas existentes, de modo a criar o seu próprio site.

Em relação ao auto-reúso, quando os participantes reusavam e adaptavam seus próprios códigos feitos anteriormente, os autores observaram que o comportamento era diferente, e os participantes não hesitavam em reusar os projetos como um modelo (*design scaveging*) para iniciar um novo projeto.

Finalmente, sobre a qualidade destes projetos, os participantes reportaram majoritariamente que, se a página aparenta estar "OK" ela será publicada, mostrando assim, que não há uma preocupação sobre a qualidade absoluta do site criado.

No trabalho de Brandt et al. (Brandt, et al., 2009), foram conduzidos dois estudos que visavam compreender como e porque programadores experientes buscavam recursos on-line quando estavam programando, e como estes eram utilizados. Eles observaram que os participantes usavam a internet como uma ferramenta para *aprender* um novo conceito, *lembrar* detalhes sintáticos de um conceito já conhecido, ou *esclarecer* um conhecimento que eles já possuem.

Como ferramenta de aprendizado, os participantes da pesquisa usavam a internet como meio de aprendizado sobre tecnologias com as quais não estavam familiarizados. Eles usavam a internet para localizar um tutorial, que depois de encontrado, tem seu código-fonte utilizado como uma

[5] Quando o reúso ocorre através da reprodução de múltiplas e contínuas linhas de código.

[6] No contexto de programação, a indentação (recuo de texto em relação a sua margem) é utilizada com o objetivo de ressaltar a estrutura do algoritmo, aumentando assim a legibilidade do código. Em algumas linguagens a indentação é usada para definir a hierarquia dentre os blocos de código e é essencial para o funcionamento correto do programa.

ferramenta para “aprender fazendo”. Após encontrar um tutorial que eles acreditam ser útil, imediatamente era iniciada uma experimentação dos seus exemplos de código, feito mesmo antes da leitura do conteúdo do tutorial. De acordo com os pesquisadores atuantes na pesquisa, isso pode provavelmente dar-se ao fato de tutoriais geralmente possuírem uma grande quantidade de textos, o qual consumiria um considerável tempo para leitura e entendimento. Um dos participantes da pesquisa afirmou que acreditava ser menos custoso, para ele, “pegar” o primeiro código encontrado e ver quão útil ele poderia ser, ao invés de somente ler sua descrição e textos explicativos associados. Outro ponto observado era que os participantes geralmente iniciavam a adaptação deste código fonte, adequando-o ao contexto de suas aplicações, antes mesmo de compreender como ele funcionava. Ao final, foi contabilizado que cerca de ⅓ dos códigos presentes nos projetos dos participantes foram copiados da Internet.

Em relação ao uso da Internet como ferramenta para lembrar, foi observado que os participantes usavam a Internet como uma “memória externa”, a qual eles a tinham como alternativa para memorizar trechos de códigos rotineiramente usados. Em um caso ocorrente durante a pesquisa, após copiar, da documentação oficial da linguagem de programação PHP, seis linhas de código necessárias para fazer a conexão com banco de dados, o participante foi questionado pelo pesquisador sobre já ter feito isso antes. A resposta do participante foi que, além de já ter feito isso diversas vezes, ele nunca se preocupou em aprender os comandos, uma vez que ele sabia que eles sempre estariam disponíveis na Internet.

Sobre o uso da Internet como recurso para esclarecer um conhecimento, foi observado, em muitos momentos, que os participantes precisavam de um pedaço de informação para ajuda-los a mapear suas ideias sobre uma situação específica. Foi observado também que, quando reusado para este fim, os participantes copiavam e colavam linhas provenientes da Internet, incorporando-as em seus códigos, e não as testavam imediatamente. Porém, eles frequentemente cometiam algum erro ao adaptar este código para o seu novo contexto, e, devido ao fato de assumirem que o código copiado estava correto, eles partiam para a codificação de outra funcionalidade, sem testar a anterior. Quando finalmente os testes eram executados e os erros encontrados, os participantes erroneamente assumiam que estes estavam sendo causados pelos códigos mais recentemente introduzidos, o que dificultava o rastreamento destes erros.

O trabalho de Ichinco e Kelleher (Ichinco & Kelleher, 2015) foca em programadores iniciantes e visa compreender um pouco mais sobre os desafios relacionados ao uso de exemplos, e eliciar os obstáculos existentes e quais as estratégias usadas por estes programadores para conseguir utilizar um exemplo. O estudo foi conduzido com 18 crianças e adolescentes, com idade média de 11.4. No estudo, os participantes deveriam, usando o ambiente de desenvolvimento *Looking Glass*[7], criar seis diferentes animações 3D. Os participantes receberam seis programas a serem concluídos, sendo que cada um focava em diferente conceito de programação, e eles deveriam alterar o programa dado de modo a criar uma animação específica. Também foi fornecido aos participantes um exemplo de código para cada tarefa que deveriam executar, sendo que este exemplo simulava um exemplo encontrado on-line.

Um dos obstáculos percebidos pelos pesquisadores foi em relação à compreensão do exemplo, o que por muitas vezes dificultava ou o seu uso, ou que os participantes pudessem gerar novas ideias a partir deles. Em alguns casos, os participantes sequer compreenderam como este exemplo estava relacionado à tarefa que eles deveriam realizar, e devido a isto, nem consideraram usá-lo para tentar ter ideias a partir dele. Outros obstáculos estavam relacionados à compreensão do próprio código-fonte criado por eles: por vezes eles acreditavam saber como concluir a tarefa, porém suas ideias estavam incorretas, ou por vezes não compreendiam como o seu código funcionava, uma vez que o comportamento do programa não era o esperado.

Os pesquisadores observaram que, algumas vezes, os participantes demoravam a atingir o "ponto de percepção"[8] porque focavam em executar o exemplo ao invés de ler o exemplo e tentar compreende-lo. Além disso, foi observado que o fato de não compreender como o exemplo estava relacionado ao código que eles estavam construindo, os participantes se sentiam desencorajados a tentar compreende-lo.

Também relacionado a nossa pesquisa, o trabalho de Hoadley e colaboradores (Hoadley, et al., 1996) apresenta dois estudos sobre quando, porque e como programadores iniciantes reúsam

[7] *https://lookingglass.wustl.edu/*

[8] Os autores definiram o chamado "ponto de percepção" como o momento no qual o participante percebia qual parte do exemplo deveria ser utilizada na tarefa.

códigos-fonte. Durante os estudos, os autores objetivaram observar se o ato de descrever uma explicação (sumarização) sobre os códigos-fonte aumentaria a probabilidade de reúso e se havia relação entre a qualidade da sumarização em relação ao reúso. Além disso, investigaram também se as crenças destes programadores sobre reúso influenciavam de alguma forma seus desempenhos. Participaram dos estudos alunos de disciplinas introdutórias do curso de Ciência da Computação da Universidade da Califórnia.

Os autores observaram que, para reusar códigos-fonte durante a resolução de novos problemas, os programadores devem acreditar que o reúso é possível e desejável. Dos participantes dos estudos, cerca de 20% rejeitavam o reúso de código-fonte, e alguns deles viam reúso como uma forma de plágio, enquanto outros simplesmente não confiavam em códigos escritos por outros programadores. Condizente a suas crenças, estes participantes reusaram códigos em cerca de somente 5% dos casos estudados. Por outro lado, os demais 80% dos participantes se mantiveram neutros ou a favor do reúso, alegando que reúso é eficiente, reduz a complexidade e simplifica a depuração[9].

Neste segundo grupo, foi observado que a compreensão do código-fonte influencia na frequência e na forma de reúso. Os códigos-fonte compreendidos abstratamente foram reusados mais do que os códigos compreendidos algoritmicamente[10], e códigos-fonte sumarizados algoritmicamente foram menos reusados que aqueles sumarizados incorretamente, concluindo desta forma que crenças favoráveis ao reúso, combinadas a habilidade de fornecer sumarizações abstratas levam a reúso mais frequente.

Ao reusar códigos-fonte, os autores definem que este reúso pode ser feito através de clonagem de código (*code cloning*) ou invocação de código (*code invocation*) (estes termos foram abordados previamente na Seção 2.1). Os participantes que sumarizaram os códigos-fonte de forma

[9] Depuração, também conhecido pelos tempos *debug* ou *debugging*, é o processo usado para encontrar e reduzir defeitos num aplicativo de *software*.

[10] Os autores classificaram as sumarizações feitas pelos participantes como abstratas (aquelas onde o participante descreveu corretamente a relação entre entradas e saídas da função, independentemente de como isso era realizado), algorítmicas (aqueles onde o participante descrevia corretamente as ações da função, sem especificar a relação entre entradas e saídas) e incorretas.

abstrata foram mais propensos a reusar códigos através de invocação do que clonagem, além disso, estes participantes muitas vezes viam os códigos-fonte sendo reusados como um padrão.

Por fim, os autores sugerem que cursos de computação devem dar mais ênfase a compreensão de código-fonte como forma de "melhorar" o reúso. Para alguns alunos, a compreensão algorítmica do código pode parecer suficiente e, assim, eles não procuram por representações mas abstratas. Para criar essa visão abstrata, professores devem apoiar os alunos, oferecendo a eles sumarizações abstratas e incluir, como parte das atividades de ensino, a atividade de leitura e sumarização de códigos-fonte. Este trabalho nos mostra que existe uma relação direta entre a compreensão do código-fonte e o reúso do mesmo, apresentando que, nestes casos, o uso de técnicas de sumarização pode apoiar os programadores iniciantes neste processo de compreensão. Além disso, nos mostra que são necessárias formas de guiar os programadores durante essa sumarização, para que ela seja feita em um nível abstrato ao invés de um nível algorítmico.

Outra forma comumente utilizada de reúso é fazendo-o através de APIs, sendo, estas, ferramentas que foram construídas com o objetivo de serem reusadas, tendo sido testadas e documentadas, e, após, disponibilizadas aos programadores que desejam utilizar alguns dos recursos implementados por elas (Sojer, 2011). Esta característica faz com que as APIs sejam um dos componentes mais reusados durante o processo de desenvolvimento de *software*. Linguagens como JAVA e .Net oferecem recursos com diversas classes que executam tarefas dos mais variados tipos. Essas APIs oferecem uma interface que possibilita aos desenvolvedores acessar estes recursos de forma simples, através de abstração de alto-nível.

No entanto, devido ao crescimento e diversificação das APIs, atualmente é necessário também endereçar questões sobre a sua usabilidade. Robillard (Robillard, 2009) conduziu uma pesquisa com 80 programadores profissionais da Microsoft, endereçando questões sobre os obstáculos percebidos por eles que poderiam dificultar o uso de uma API. Deste grupo, 49% eram desenvolvedores nos níveis de júnior a intermediário. Em relação a como estes desenvolvedores aprendiam a interagir com as APIs, 78% indicaram que aprendiam lendo a documentação, 55% aprendiam usando exemplos, 34% faziam experimentos usando a API, 30% liam artigos e 29% questionavam os colegas.

Sobre os obstáculos que dificultavam o aprendizado sobre como usar as APIs, eles foram classificados conforme apresentado a seguir:

- Recursos: Obstáculos causados pela inadequação ou ausência de recursos que possibilitem aprender sobre a API. Dentre eles, estão exemplos insuficientes ou inadequados, problemas gerais sobre a documentação, ausência ou inadequação de conteúdos na documentação, falta de referências sobre como usar a API para realizar uma tarefa específica, formato usado para apresentar esses recursos, e, falta de informações sobre os aspectos de alto-nível da API, como seu *design.*
- Estrutura: Obstáculos relacionados à estrutura ou *design* da API, causando problemas relacionados a testes, *debugging* e tempo de execução.
- *Background*: Obstáculos relacionados às experiências anteriores do desenvolvedor.
- Ambiente técnico: Obstáculos relacionados ao ambiente em que a API está sendo utilizada.
- Processo: Obstáculos relacionados a problemas no processo, tais como tempo e interrupções.

É importante mencionar que 50 dos respondentes forneceram respostas que foram categorizadas como obstáculos de “Recursos”, sendo que 20 destas respostas foram classificadas na subcategoria “Exemplos”, sendo este o item mais mencionado. Considerando que a maioria dos respondentes da pesquisa eram programadores iniciantes, o fato reforça a necessidade que este público tem em usar exemplos durante o processo de desenvolvimento, sendo inclusive amplamente difundidas abordagens que sugerem exemplos aos programadores para facilitar sua interação com as APIs (Alnusair, et al., 2010) (Holmes, et al., 2010) (Moreno, et al., 2015), o que nos retorna a questões iniciais sobre como estes exemplos são ou deveriam ser utilizados durante o processo de desenvolvimento.

Com o advento da área de HCC, questões relacionadas a aspectos humanos têm sido levantadas em relação a diversas etapas e artefatos usados no processo de desenvolvimento de *software*. Conforme mencionado anteriormente (Seção 2.3), a Engenharia Semiótica vem contribuindo para esta área, pesquisando e fornecendo meios para apoiar o estudo de aspectos comunicativos nestes artefatos. Na tese de doutorado de Afonso (Afonso, 2015) é proposta uma nova abordagem para investigação e discussão do design de APIs, na qual a é usada Engenharia

Semiótica como teoria norteadora. Em sua pesquisa, o autor vê as APIs como um processo de comunicação que ocorre entre o designer desta API, e o programador que a está utilizando, no qual este designer expressa ao programador, através da API, uma mensagem codificada que explica como ele deve se comunicar com o sistema e como ele pode usar as funcionalidades da API.

O autor defende que, apesar de serem executados por uma máquina, *softwares* têm sua criação e processos de uso profundamente baseados em interpretação humana, uma vez que a maioria dos *softwares* são escritos *por* humanos e *para* humanos. Assim, programadores devem ter um bom entendimento sobre as abstrações presentes nestas APIs e serem capazes de articularem isso em prol da construção suas próprias abstrações. Sobre exemplos de código, o autor salienta que estes são úteis para ilustrar como melhor usar este artefato, porém eles também podem dificultar a compreensão do programador, pois pode haver diferenças entre o contexto de uso ilustrado pelo exemplo, e o contexto no qual o programador almeja utilizar a API.

O trabalho propõe um *framework* conceitual baseado em semiótica e teorias cognitivas, que salienta os aspectos pragmáticos envolvidos na comunicação entre os desenvolvedores da API e seus usuários, comunicação esta que se dá através da interface de programação. O framework pode ser usado como uma ferramenta epistêmica para apoiar o desenvolvimento de APIs e trabalha com três dimensões comunicativas, sendo elas *intenção, efeito* e *falha,* onde *intenção* refere-se a visão de *design* codificada no artefato pelo desenvolvedor da API, *efeitos* refere-se aos potenciais resultados comunicativos resultantes da interação do programador com a API, e, *falha*, que refere-se a detecção de possíveis rupturas de comunicação, resultando em efeitos indesejados no programa de quem está usando a API. Além disso, seu trabalho endereça questões profundas sobre *design*, computabilidade e pragmática, tal como a importância de se estabelecer um contrato pragmático entre o *designer* e o usuário da API, que deve estender as especificações de comportamento dos tradicionais contratos de *software,* que estabelecem pré-condições, a serem atendidas pelo usuário, e pós-condições, a serem atendidas pelo *designer*, adicionando também as informações em relação as intenções desta API, prevenindo assim que ocorram problemas na comunicabilidade do artefato.

Parte deste trabalho, desenvolvido na tese de Afonso (Afonso, 2015), compõe o livro *Software Developers as users: Semiotic Investigations in Human-Centered Software Development* (De Souza, et al., 2016). O livro em questão apresenta suas contribuições para o avanço da área de HCC usando

a teoria da Engenharia Semiótica, anteriormente mencionada, fornecendo um conjunto de ferramentas conceituais e metodológicas que apoiam a investigação sobre como significados humanos se manifestam durante o desenvolvimento e uso de *softwares*. Este conjunto de ferramentas, denominado *SigniFYI (Signs For Your Interpretation)*, engloba cinco ferramentas para apoiar o estudo de significados inscritos em artefatos de *software*, sendo a *SigniFYIng APIs* baseada no trabalho de Afonso. Dentre as demais ferramentas, destacam-se, relacionadas a esta pesquisa, a *SigniFYIng Message*, uma ferramenta conceitual que fornece uma estrutura central para caracterização do processo de CMC. Esta ferramenta elicita as categorias de significado emergentes do *template* de metacomunicação da Engenharia Semiótica (apresentado na seção 2.3), e organiza estas categorias na forma de um formulário de estrutura de metacomunicação (Anexo A).

Outra das ferramentas é a *SigniFYIng Interaction*, classificada como uma ferramenta metodológica cujo propósito é apoiar a avaliação da qualidade da metacomunicação transmitida pelos signos da interface que está sendo avaliada. Por fim, a *SigniFYIng Models,* também uma ferramenta metodológica, tem o objetivo de apoiar a investigação de problemas que desenvolvedores possam ter com sistemas de modelagem de *software*.

Além disso, destacamos também que aspectos de reúso são amplamente discutidos dentro da área de EUD. No trabalho de Ko e colaboradores (Ko, et al., 2011) é apresentada uma revisão sistemática sobre o estado da arte da área de *End-User Software Engineering*[11]. Neste trabalho, são discutidas questões em relação ao reúso de *software* por usuários finais atuando com programação e em relação também às formas de apoiá-los neste processo. Os autores ressaltam que existem diferenças entre os desafios enfrentados por programadores experientes e usuários finais atuando com programação. Uma delas, por exemplo, é o fato de que as APIs desenvolvidas para uso profissional normalmente objetivam otimizar sua flexibilidade (ou seja, sua capacidade de atender diversos domínios diferentes), enquanto usuários finais normalmente precisam de apoio para alcançar objetivos de um domínio específico. Neste ponto, podemos observar um perfil

[11] A área de *End-User Software Engineering* é uma área derivada da área de EUD, que visa investigar também formas apoiar o usuário final que está atuando com programação a melhorar a qualidade dos *softwares* que estão desenvolvendo.

concordante com o do programador iniciante que, normalmente, guia-se por atingir seus objetivos, não levando em conta questões profundas sobre como irá atingi-los.

Estes estudos nos mostram que, muitas vezes, programadores reusam exemplos para apoiar suas atividades e seu aprendizado, porém, às vezes, não compreendem como o código de exemplo funciona, e acabam por introduzir erros nos programas que estão desenvolvendo. Também foi possível observar que em alguns casos, o exemplo adequado deixa de ser usado devido à falta de compreensão sobre o mesmo, compreensão esta que acreditamos que poderia ser adquirida através de um processo de reflexão sobre o código-fonte. Vimos em diversos trabalhos que muitas vezes o foco do programador está em observar o comportamento funcional do programa, executando-o. Porém, com exceção do trabalho de Afonso (Afonso, 2015) (Bastos, et al., 2017), não encontramos outros trabalhos que foquem em apoiar o programador a compreender os possíveis aspectos comunicativos de seu código-fonte.

Observamos, também, a existência de diversas pesquisas que visam, de alguma forma, recomendar exemplos de código-fonte para programadores, mas não encontramos pesquisas que enderecem questões que apoiem o programador a compreender este exemplo. Além disso, não encontramos em nenhumas destas pesquisas dados sobre os efeitos do reúso de exemplos e suas diferenças nos casos nos quais houve e nos quais não houve compreensão e apropriação sobre o mesmo, principalmente em relação à sobre como estes programadores interpretam os códigos-fonte escritos por eles quando houve reúso.

No próximo capítulo, iremos apresentar dois estudos que nos apoiaram na coleta de dados sobre como programadores iniciantes interpretam seus programas quando reusaram exemplos de código-fonte – com e sem apropriação do mesmo – e um estudo em que visamos observar se há diferença em relação a essa interpretação quando o reúso é feito através de APIs.

3 ESTUDOS PARA COMPREENSÃO DO PROBLEMA

No intuito de compreender melhor e delimitar o problema, identificando como programadores iniciantes reúsam código-fonte, se e como se apropriam dos códigos reusados, e os efeitos disso sobre como eles mesmos compreendem os programas feitos, conduzimos três estudos com programadores iniciantes com perfis diversos. A Figura 3 ilustra onde tais estudos estão localizados dentro das etapas desta pesquisa.

Figura 3: Localização dos estudos de compreensão do problema na pesquisa

O Estudo 1 foi conduzido com alunos de uma disciplina introdutória sobre programação de um curso de graduação na área de Computação, sendo este o estudo precursor deste trabalho. Através dele fizemos as primeiras observações relacionadas a reúso e seus efeito na apropriação. No Estudo 2, conduzido com alunos com o perfil de usuários finais, observamos como o reúso ocorre entre programadores com perfil EUD, ou seja, pessoas que estão aprendendo programação para apoiar a resolução de problemas de seu dia-a-dia. Por fim, o Estudo 3, conduzido com a participação de alunos de um curso sobre de programação *mobile*, foi realizado a fim de observar como o reúso ocorre quando o foco deste reúso está em componentes do tipo API, sendo, estes, componentes que são desenvolvidos com o intuito de serem reusados.

Os referidos estudos e seus detalhes são apresentados nas subseções a seguir.

3.1 Estudo 1: Reúso de códigos-fonte e compreensão da mensagem de metacomunicação

O Estudo 1, aqui descrito, foi não somente um estudo que nos permitiu fazer as primeiras observações em relação ao nosso objeto de estudo (reúso de código-fonte por programadores iniciantes), mas também o caso observado que fez com que a nossa questão de pesquisa emergisse.

Ao final do semestre letivo, durante uma das aulas de algoritmos e programação oferecida ao semestre inicial dos cursos de Bacharelado em Ciência da Computação e em Sistemas de Informação, foi proposto aos estudantes um exercício no qual desenvolveriam um programa para gerenciar uma livraria. Através deste programa deveria ser possível cadastrar clientes, livros e vendas. Foi apresentado pelo professor[12] aos alunos o código-fonte de uma possível solução (uma vez que se tratava de um exercício de aula, não valendo nenhuma nota) e este foi disponibilizado através do ambiente virtual de aprendizado usado no apoio às aulas.

Ainda no decorrer desta mesma disciplina, os alunos deveriam desenvolver o projeto final da disciplina, no qual precisavam construir um programa em Java para avaliar jogos educacionais, permitindo o cadastro de avaliadores, jogos e avaliações. Além disso, o programa deveria disponibilizar relatórios com informações gerais e estatísticas sobre as avaliações.

Após a entrega dos trabalhos, observamos em diversos projetos finais construídos pelos alunos, trechos de código-fonte semelhantes, e por muitas vezes iguais, aos do código-fonte do projeto da Livraria. Em outras palavras, os alunos atuaram como usuários de um código escrito por outra pessoa, neste caso, o professor. A partir desta observação, os primeiros passos em direção a construção desta pesquisa foram dados, levando-nos a tentar melhor compreender os aspectos relacionados ao reúso de código por alunos de programação.

Para melhor compreender o cenário, usamos este caso como nosso primeiro objeto de estudo, realizo após a conclusão do semestre letivo, no qual os códigos foram analisados com mais detalhes e entrevistas foram conduzidas com o objetivo de melhor compreender como ocorreu este processo de apropriação.

[12] No caso deste estudo, o professor em questão era a própria autora desta tese.

Observando o processo de reúso do ponto de vista da Engenharia Semiótica, queremos levar em consideração, também, a apropriação que programadores têm sobre a mensagem de metacomunicação emitida pelo *software* que está sendo criado. Desta forma, ao reusar códigos, consideramos que programadores estão não apenas copiando e colando linhas de comando, mas estão estendendo esta ação ao discurso implícito no código que está sendo reusado.

Portanto, neste estudo, nossos objetivos foram analisar: (1) como programadores iniciantes reúsam códigos-fonte feitos por outra pessoa, (2) como eles compreendem e se apropriam disso para criar um novo código-fonte, (3) como eles interpretam este código-fonte.

Os detalhes sobre a metodologia usada e os resultados são apresentados nas seções a seguir.

3.1.1 Metodologia de coleta e análise dos dados

A análise dos códigos-fonte produzidos pelos alunos foi feita em duas etapas: (1) análise dos códigos para identificar se e como eles reusaram o exemplo; e (2) entrevista com alunos selecionados de acordo com os resultados da etapa anterior.

Durante a primeira etapa de análise, foi feita a verificação dos níveis de similaridade entre o código da Livraria e o código-fonte dos trabalhos desenvolvidos pelos alunos. Para tal, nós utilizamos a ferramenta Moss[13], desenvolvida e mantida pela Universidade de Stanford, capaz de calcular as métricas de similaridade entre textos, sendo estes textos, no caso deste estudo, os códigos-fonte dos alunos.

Durante a análise dos códigos, observamos que diversos alunos reusaram os códigos do projeto Livraria para construir seus programas. O mapeamento feito por eles foi:

- O código que no projeto Livraria era usado para cadastro de clientes, foi reusado para o cadastro de avaliadores.
- O código que no projeto Livraria era usado para cadastro de livros, foi reusado para o cadastro de jogos.

[13] https://theory.stanford.edu/~aiken/moss/

- O código que no projeto Livraria era usado para cadastro de vendas (relacionando livros a clientes), foi reusado para o cadastro de avaliações (relacionando jogos a avaliadores).

Foram selecionados, por fim, para uma análise mais aprofundada, quatro códigos-fonte, sendo dois deles os mais semelhantes ao exemplo, e dois deles, os mais diferentes; dada atenção especial aos que mais haviam reutilizado o projeto livraria, visando compreender as condições em que esse reúso foi feito.

Então, foram conduzidas atividades específicas contando com a participação dos autores[14] dos códigos selecionados:

1. Os autores foram convidados a observarem diversos programas (cada um recebeu sete diferentes códigos-fonte do projeto final, incluindo seu próprio código-fonte) e identificarem o seu, assim como descrever as características presentes em seus programas que lhes ajudaram na identificação.
2. Após, foi solicitado aos participantes que explicassem algumas partes do programa que eles identificaram como sendo o seu. Devido ao tamanho dos códigos-fonte dos programas (entre 585 a 4100 linhas), foram selecionados alguns trechos representativos de cada um. Os programas foram estruturados em diversas classes, sendo cada classe (ou conjunto de classes) responsável por uma característica específica do programa. A maior parte dos códigos era composta por uma classe de interface responsável por, além da interação com o usuário, comunicar-se com as três classes de gerenciamento do programa. Cada classe de gerenciamento era responsável por registrar, pesquisar e remover cadastros criados com base em classes de modelo. Por exemplo: a classe "Cadastro de Clientes" era responsável por adicionar, pesquisar e remover cadastros criados a partir da classe nomeada Cliente. Portanto, para esta etapa foram usadas:
   a. Funções de adição, remoção e pesquisa: retiradas tantos da classe de interface, quanto das classes de gerenciamento, totalizando seis funções.

[14] Todos os participantes entrevistados assinaram o Termo de Consentimento Livre e Esclarecido no qual autorizaram o uso dos dados coletados durante as entrevistas para fins acadêmicos.

b. Função principal: função retirada da classe de interface, responsável por criar um menu e disponibiliza-lo aos usuários, além de ser responsável pela chamada de outras funções da interface.
c. Função autoral: uma função que não tivesse correspondente no projeto livraria, ou seja, uma função totalmente original, criada pelo próprio participante.

Observamos que os métodos listados para serem explicados não possuíam informações diretas sobre qual classe/parte do código eles eram provenientes, e os participantes eram convidados a tentar identificar de qual classe o método havia sido retirado.

3. Na última etapa da entrevista os participantes eram convidados a responder um questionário no qual perguntas a respeito de uso de exemplos, comunicação e autoria eram abordadas (Apêndice A).

   O questionário se encerrava convidando o participante a preencher o *template* de metacomunicação da Engenharia Semiótica, completando as sentenças do mesmo (*Este é o meu entendimento de quem você é..., O que aprendi...*, etc) de acordo com o que os participantes julgavam que seu código fazia.

   Os dados coletados foram analisados individualmente para melhor compreender os diferentes cenários envolvidos.

3.1.2 Perfil dos participantes

Na primeira fase, usando a ferramenta Moss, identificamos o percentual de similaridade de 15 projetos desenvolvidos pelos 23 alunos[15] [16](QuestJogos) com o projeto Livraria. A ferramenta Moss usa algoritmos para identificar trechos similares dentre textos, e é capaz de detector não somente trechos iguais, mas também padrões (Schleimer, et al., 2003). Assim, mesmo que o aluno alterasse itens como nomes de variáveis, funções, ou alterasse suas distribuições dentro do código,

[15] Todos os 23 alunos autorizam formalmente o uso dos códigos-fonte de seus programas e assinaram o Termo de Consentimento Livre e Esclarecido.

[16] A diferença entre o número de trabalhos e o número de alunos, deve-se ao fato de que alguns trabalhos foram desenvolvidos em duplas.

a ferramenta era capaz de identificar que estas partes foram reusadas. Os resultados da análise são apresentados na Tabela 1.

Tabela 1: Similaridade dos códigos com o projeto de exemplo

Projeto do participante[17]	Linhas similares
Participante 1 (44%) *	323
Participante 2 (35%)	194
Participante 3 (33%) *	267
Participante 4 (31%) *	182
Participante 5 (27%)	209
Participante 6 (21%)	161
Participante 7 (20%)	144
Participante 8 (18%)	123
Participante 9 (18%)	118
Participante 10 (13%)	156
Participante 11 (12%) *	99
Participante 12 (9%) *	62
Participante 13 (6%)	130
Participante 14 (3%) *	72
Participante 15 (1%) *	41

A partir destes resultados, decidimos por remover da análise aqueles que fizeram o trabalho em duplas, restando os participantes assinalados na Tabela 1 com um asterisco (*) ao lado do nome. No intuito de aprofundar a pesquisa, selecionamos códigos de dois extremos: Participante 1 (P1) e Participante 3 (P3), sendo esses os mais similares ao exemplo, e Participante 14 (P14) e Participante 15 (P15), sendo estes, os mais diferentes do exemplo (Tabela 2).

Tabela 2: Perfil dos participantes do estudo 1

Participante	Curso	Gênero	Similaridade
Participante 1 (P1)	Ciência da Computação	Masculino	44%
Participante 3 (P3)	Bacharelado em Matemática	Masculino	33%
Participante 14 (P14)	Ciência da Computação	Feminino	3%
Participante 15 (P15)	Sistemas de Informação	Masculino	1%

[17] O termo Participante pode representar tanto um trabalho individual quanto um trabalho feito em dupla.

3.1.3 Resultados

Foram procuradas as mudanças feitas pelos participantes em seus códigos-fonte (mudanças em relação ao projeto Livraria). Observou-se que os participantes P14 e P15 tinham em seus projetos diversas diferenças, tais como a organização do projeto, interface (tanto em nível de código, quanto em nível de apresentação para o usuário), nomes de classes, distribuição de classes, nomes de funções, e operações das funções. Por outro lado, os participantes P1 e P3 reusaram o projeto Livraria, tendo modificado apenas alguns detalhes: P1 modificou somente o padrão dos nomes das classes e P3 modificou o código de algumas classes, alterando um pouco seu funcionamento. Foi possível observar que estes dois projetos eram muito semelhantes entre si, uma vez que ambos foram baseados no mesmo exemplo, e as mudanças feitas foram quase irrelevantes. Um exemplo dos códigos produzidos pode ser visto na Figura 4, apresentada a seguir.

Cadastro de Livros do projeto Livraria

```
import java.util.*;
public class CadLivro
{
ArrayList<Livro> listaLivros = new ArrayList<Livro>();

public ArrayList<Livro> getListaLivros(){return listaLivros; }

public boolean adicionar(Livro l){
return listaLivros.add(l);
}

public boolean remover(int cod){
Livro liv = pesquisaLivro(cod);
if(liv == null){
return false;
} else {
return listaLivros.remove(liv);
}
}

public Livro pesquisaLivro(int cod){
for(Livro li : listaLivros){
if(li.getCodIdentificacao() == cod){
return li;
}
}
return null;
}
}
```

Cadastro de Jogos do projeto de P1

```
import java.util.*;

/**
* Essa classe cria uma lista de Jogos.
*
* @author (          )
* @version (novembro/2014)
*/

public class ListaJogos
{
ArrayList<Jogo> listaJogos = new ArrayList<Jogo>();

public ArrayList<Jogo> listaJogos(){
return listaJogos;
}

public boolean adicionar(Jogo j){
return listaJogos.add(j);
}

public boolean remover(String nome){
Jogo jog = pesquisaJogo(nome);
if(jog == null){
return false;
} else {
return listaJogos.remove(jog);
}
}

public Jogo pesquisaJogo(String nome){
for(Jogo j : listaJogos){
if(j.getNomeJogo().equalsIgnoreCase(nome)){
return j;
}
}
return null;
}


}
```

Cadastro de Jogos do projeto de P3

```
import java.util.*;

public class CadJogo{
ArrayList<Jogo> listaJogos = new ArrayList<Jogo>();

public ArrayList<Jogo> getListaJogos(){
return listaJogos;
}

public boolean adicionar(Jogo j){
return listaJogos.add(j);
}

public boolean remover(String nomeJogo){
Jogo jog = pesquisaJogo(nomeJogo);
if(jog == null){
return false;
} else {
return listaJogos.remove(jog);
}
}

public Jogo pesquisaJogo(String nomeJogo){
for(Jogo j : listaJogos){
if(j.getNomeJogo().equalsIgnoreCase(nomeJogo)){
return j;
}
}
return null;
}
}
```

Figura 4: Comparativo entre o exemplo fornecido pelo professor, e os códigos produzidos pelos alunos

Apesar disso, o projeto de P3 tinha cerca de 75 linhas a mais em relação ao projeto de P1. Isto porque este participante incluiu uma série de funções com o objetivo de gerar relatórios com as informações registradas no sistema, sendo isso algo que não constava no escopo do projeto Livraria.

Após a análise dos códigos, os participantes foram convidados a participarem de uma entrevista. Durante a primeira etapa da entrevista, os participantes foram convidados a tentarem encontrar seus códigos dentre outros 7 códigos-fonte fornecidos. Todos os códigos fornecidos foram produzidos pelos outros participantes, com o mesmo objetivo. Alguns destes códigos eram similares ao projeto Livraria e, consequentemente, similares entre si. Nesta etapa, visamos observar os traços autorais que os participantes reconheceriam como sendo deles.

O participante P1 foi quem reusou mais linhas do exemplo (44%). Durante a etapa de busca pelo seu código, foi solicitado a ele que expressasse seus pensamentos em voz alta para que fosse possível tomar notas sobre seus comentários[18]. Logo que iniciou a busca, o participante reportou que acreditava ter encontrado seu código. No entanto, o participante havia se equivocado e confundido seu código com o código de outro participante (P2), que, assim como ele, havia feito um alto reúso do exemplo (35%). De acordo com o participante, ele percebeu seu erro, pois seu código deveria conter determinadas linhas no código da interface que não estavam presentes. Por fim, ele agrupou quatro códigos-fonte, e relatou que todos eram muitos semelhantes (seu próprio código e também os de P2, P3 e P12). No entanto, ele conseguiu identificar seu código devido à presença de nomes das variáveis mnemônicos.

O próximo entrevistado foi o participante P3 que recebeu o mesmo conjunto de códigos dado ao participante anterior. Assim como o anterior, o código de P3 também possuía várias linhas reusadas do projeto Livraria (33%). No entanto, diferente do participante anterior, este participante teve poucas dúvidas em relação ao código escrito por ele. Ele levou algum tempo comparando seu código com o código do participante P2, e então, apontou seu código corretamente. De acordo com o próprio participante, ele foi capaz de identificar seu código devido à forma como uma palavra estava escrita.

Por outro lado, a participante P14 diferia dos demais anteriormente apresentados, pois além de possuir um código extremamente diferente do código do projeto Livraria (3%), seu código possuía também diversos itens autorais que não haviam sido introduzidos em aula. A participante iniciou sua busca descartando o código que pertencia a P15 e logo, apontou seu próprio código. No

[18] O mesmo procedimento foi feito com os demais participantes.

entanto, somente para ter certeza, ela olhou brevemente todos os códigos restantes e manteve sua primeira escolha. Ela relatou que a forma como os menus da interface estavam organizados foi o principal indicativo que aquele código havia sido feito por ela.

Por fim, o último participante entrevistado foi o participante P15, sendo este aquele que havia produzido o projeto mais diferenciado dos demais, possuindo somente 1% de similaridade com projeto Livraria. Logo que ele iniciou a procura, ele apontou o projeto de P1 como sendo o seu próprio código (acreditamos que o motivo do engano se deva ao fato de o projeto Livraria ter sido construído em aula com a participação dos alunos). Porém, logo ele se recordou que ele havia criado diferentes áreas em seu programa (área de administrado e área do usuário). Então, ele reiniciou sua busca procurando por essa característica e após alguns poucos minutos encontrou seu código.

A etapa seguinte da entrevista consistiu em uma explicação feita pelos participantes sobre partes do código escolhido por eles (mesmo que tivessem feito uma escolha errônea). Porém, como citado anteriormente, todos os participantes identificaram corretamente seus códigos.

O primeiro participante, P1, não foi capaz, por muitos momentos, de explicar o que o trecho de código fazia ou de que parte do projeto o trecho havia sido retirado. Além disso, o participante forneceu uma explicação superficial que em diversos momentos se resumiu a uma leitura linha-a-linha do código, focando em detalhes sobre a sintaxe e a semântica do projeto, mas não detalhes relacionados ao papel do trecho de código em seu programa.

Em contraste, o participante P3, o qual também havia desenvolvido um programa muito similar ao projeto Livraria, deu uma explicação totalmente diferente. A explicação por ele dada era rica em detalhes, mostrando que ele possuía consciência sobre o papel e a localização dos trechos de código apresentados. Ele foi questionado a explicar algumas linhas de código que ele havia inserido no programa, linhas estas classificadas como originais, por não haver relação entre elas e os códigos do projeto Livraria. O participante tinha conhecimento sobre as linhas citadas e também o papel que estes trechos desempenhavam no programa. Esse fato apresenta um nível diferente de apropriação em relação ao participante P1, uma vez que P3 não só utilizou o exemplo como uma base para criar o seu programa, mas ele também foi capaz de estendê-lo e adicionar novas funcionalidades.

Os demais participantes, P14 e P15, não utilizaram o exemplo como base para a construção de seus programas, que eram completamente diferentes dos exemplos trabalhados em aula. Ambos foram capazes de explicar os trechos apresentados, assim como a localização destes trechos dentro do programa.

Na última etapa deste estudo foi conduzida uma entrevista endereçando algumas questões finais. Sobre a forma como eles escolhem um exemplo para ajudá-los (quando isso é feito), P1 explicou que sua preferência é por exemplos que se refiram ao mesmo domínio da aplicação que ele está desenvolvendo. Caso um exemplo nesta categoria não fosse encontrado, então seriam procurados outros exemplos abordando as operações internas que ele necessita desenvolver.

O participante P3 reportou que ele somente busca por exemplos que se refiram a um domínio de aplicação semelhante ao que ele está desenvolvendo, porém justificou que esse tipo de exemplo serve como um esqueleto que ele pode utilizar para começar a construir seu próprio programa, somente sendo necessária a substituição de algumas linhas de código.

Para este participante, um exemplo serve como uma base que ele pode ir modificando e aprimorando até atingir seu objetivo. Ambos P1 e P3 relataram que usam exemplos sempre que necessário, copiando e colando, e apenas modificando aspectos relevantes do código. P1 complementou dizendo que, caso o código usado como exemplo seja muito complexo, então ele irá utilizá-lo somente como uma fonte de referência, mas ainda assim, copiando e colando pequenas partes.

Sobre os demais participantes, P14 e P15, eles reportaram um uso diferente de exemplos. Ambos os participantes relataram que escolhem exemplos com base em suas operações, e, além disso, os utilizam somente como uma fonte de referência. P14 relatou que um bom exemplo pode ser utilizado para ajudar na compreensão da lógica do problema, indiferente da linguagem de programação ou domínio da aplicação. Ocasionalmente ela usa exemplo “copiando e colando”, mas quando feito isso, ela afirma tentar antes compreender como o trecho de código utilizado funciona. P15 relatou que ele usa exemplos quando não tem outra opção e precisa resolver um problema. Em ambos os casos, os participantes indicaram que sua principal opção é por compreender os significados do código antes de utilizá-los; desta forma, eles não apenas usam o exemplo, mas também o compreender e se apropriam dele.

Como encerramento da entrevista, os participantes eram convidados a completar o *template* de metacomunicação da Engenharia Semiótica. Nós dividimos o *template* em quatro partes, como descrito a seguir:

1. *Este é o meu entendimento sobre quem você é…*
2. *O que aprendi que você quer ou precisa fazer, de que maneiras prefere fazer, e por quê...*
3. *Este, portanto, é o sistema que projetei para você…*
4. *Esta é a forma como você pode ou deve utilizá-lo para alcançar os objetivos...*

Os participantes P3, P14 e P15 conseguiram identificar com clareza o usuário com quem estavam se comunicando, descrevendo que ele era *"uma pessoa que está buscando novas ferramentas p/ aplicação didática*" (P3), *"uma pessoa comum, aluno ou professor"* (P14) *ou "um professor avaliando uma nova ferramenta de ensino que lhe foi apresentada"* (P15).

Em relação ao "O que aprendi que você quer ou precisa fazer, de que maneiras prefere fazer, e por quê..." as mensagens dos participantes P3 e P14 são precisas em relação ao que o usuário precisa. Eles relatam que seu usuário "*precisa selecionar um aplicativo que melhor se adeque aos seus alunos e possua um bom nível de conhecimento a ser transmitido de forma clara e objetiva, com uma interface agradável que atraia a atenção do aluno*" (P3) e que este usuário *"quer armazenar e manipular informações sobre jogos"* (P14).

E os participantes P3 e P15 dizem ter projetado *"um sistema que permite que você identifique de onde são as outras pessoas que optaram por determinado aplicativo, sua idade, suas qualificações e seu parecer com relação ao mesmo"* (P3) e um sistema com o qual *"o administrador poderá ter um pequeno banco de dados a respeito dos participantes, capaz de organizar e transformar estes em informação útil. Para o usuário será uma forma simples, porém eficaz de expor suas percepções sobre as ferramentas educacionais avaliadas"* (P15). E que seus usuários alcançarão os objetivos do sistema *"Oferecendo* [este sistema] *em instituições que possuam laboratórios de informática ou estejam desenvolvendo aplicações para este fim* [desenvolvimento de jogos educacionais]*"* (P3) ou, de forma mais específica em relação a interação com o sistema, ao "*seguir os menus,* de *forma intuitiva"* (P14 e P15).

Já as mensagens de P1 eram genéricas e possuíam poucas informações em relação ao sistema em si, o que também foi observado nas mensagens de P15. Já as mensagens de P3 e P14 eram

precisas, claras e objetivas, e mostravam a apropriação que eles possuíam das mensagens que estavam passando a seus usuários.

Podemos concluir, por meio destes resultados, que ambos P1 e P3 usaram o exemplo como "esqueleto" para construção de seus programas, alterando por demanda os itens necessários para adaptar o código ao novo domínio. Destes casos, por exemplo, observa-se que um dos participantes (P1) não estava totalmente consciente da mensagem passada por seu programa, e que ele não tinha amplo conhecimento sobre a lógica de seu próprio código.

Se observarmos este ocorrido do ponto de vista abordado pela Engenharia Semiótica, podemos dizer que a mensagem de metacomunicação passada pelo preposto do designer (àquele que por meio da interface do sistema irá comunicar as intenções do designer) ao usuário (nesta situação representada pelo participante P1) é composta por duas mensagens: a mensagem emitida pelo *designer* do projeto Livraria, e sua própria mensagem. Porém essas mensagens, apesar de complementarem-se, são desconexas, uma vez que o participante P1 não se apropriou devidamente da mensagem usada como base de seu código-fonte.

Porém, outro dos casos analisados (P3) tratava de um participante que, apesar de ter feito o reúso total do código do exemplo, apropriou-se da mensagem de forma exemplar, mostrando ciência de como o seu sistema funcionava, e da localização de trechos aleatórios. Este participante elaborou sua própria mensagem de metacomunicação, usando como base o *template* de metacomunicação definido pela Engenharia Semiótica. Quando comparada com as mensagens de metacomunicação definidas por participantes que criaram programas totalmente autorais (sem o uso de exemplos) a mensagem deste participante destaca-se por conseguir atingir um nível de detalhamento tão preciso quanto (em alguns itens maior) a dos demais.

Neste caso, observamos que a mensagem passada pelo preposto do designer, apesar de composta por duas mensagens de metacomunicação diferentes, como no caso anterior, é diferente do primeiro caso. Neste caso a mensagem de metacomunicação do exemplo é incorporada à mensagem do participante.

Unimos as respostas dadas ao *template* de metacomunicação dos participantes (P3 e P14) formando uma mensagem que, em nossa opinião, retrata os sistemas, e por sua vez, retrata que os participantes têm consciência da mensagem que estão passando: "*Você é uma pessoa, aluno ou*

professor (P14) ***que está buscando novas ferramentas para aplicação didática, porém com receio de optar por algo inapropriado, recebendo auxílio de outros usuários*** *(P3).* ***Você precisa selecionar um aplicativo que melhor se adeque aos seus alunos e possua um bom nível de conhecimento a ser transmitido de forma clara e objetiva, com uma interface agradável que atraia a atenção do aluno*** *(P3). Para isso precisará cadastrar as pessoas, que respondem questionários, e jogos, no caso o objeto do questionário. Por fim, você poderá consultar, de diferentes formas as estatísticas sobre os questionários (P14).* ***Desta forma, projetei para você um sistema que permite que você identifique de onde são as outras pessoas que optaram por um determinado aplicativo, sua idade, suas qualificações e seu parecer com relação ao mesmo*** *(P3). Você pode seguir os menus, de forma intuitiva. Inicialmente, terá que cadastrar os respondentes e os jogos. A partir disso, os questionários poderão ser respondidos (pelos respondentes, sobre os jogos). Por fim, você poderá consultar as informações geradas, a partir de estatísticas e relatórios. (P14)*".

O destaque deste *template* certamente recai sobre as falas de P3 (em destaque no parágrafo anterior), que mesmo tendo feito reúso do exemplo Livraria, mostra apropriação sobre a mensagem que desejava passar a seu usuário. Isso nos mostra que o reúso de exemplos pode e terá impactos diferentes na mensagem de metacomunicação quando àquele que a emite não se apropria da mensagem do exemplo.

Com este estudo conduzido juntamente a alunos de programação podemos ver que mesmo com total reúso de um código de exemplo, desenvolvido por outra pessoa, o *designer* (neste caso, os alunos) pode se apropriar disso e compreender a mensagem que está passando para seus usuários. Também analisamos, neste estudo, a forma como autores de programas que não reusaram códigos interpretavam a mensagem que estavam passando aos seus usuários, o que nos trouxe insumos em relação a como programadores, nunca antes apresentados a conceitos de Engenharia Semiótica e CMC, compreendem esta comunicação.

3.2 Estudo 2: Reúso de códigos-fonte

O estudo anterior nos apresentou dois cenários distintos, onde um mesmo exemplo foi reusado de maneiras distintas, tendo participantes com pouquíssimo entendimento sobre o seu código, enquanto outros compreendiam com profundidade o programa feito, tendo inclusive a capacidade de estendê-lo. Para compreender melhor o cenário onde o programador em questão

não se apropria do exemplo, decidimos conduzir este novo estudo, abordando desta vez participantes que tenham perfil EUD, ou seja, programadores que estariam fazendo um programa para apoia-los em problemas de seu dia-a-dia, sendo eles próprios consumidores da mensagem que emitem. Este estudo foi, então, conduzido com alunos de uma disciplina de introdução a programação oferecida ao 2º semestre dos cursos de Engenharia Civil e Engenharia de Produção. Diferente dos participantes do estudo anterior, estes participantes não possuem – em suas grades curriculares - diversas disciplinas sobre programação e Engenharia de *Software*, disciplinas estas nas quais posicionamentos sobre boas práticas de programação são muitas vezes abordados, e nas quais questões sobre reúso de código-fonte podem vir a ser abordadas.

Em síntese, neste estudo, visamos analisar (1) como programadores iniciantes reusam códigos-fonte feitos por outra pessoa, (2) como eles compreendem e se apropriam deles para criar um novo código-fonte, e (3) como eles interpretam este código-fonte.

3.2.1 Metodologia de coleta e análise de dados

Ao final do semestre letivo, como forma de apresentar os recursos da linguagem relacionados a comandos de repetição, a professora responsável pela disciplina apresentou aos alunos alguns exemplos. Aproximadamente duas semanas após este evento, foi solicitado aos alunos um trabalho, feito em aula, no qual eram solicitados três diferentes exercícios. Os códigos-fonte dos trabalhos produzidos pelos participantes foram utilizados como base deste estudo. Salientamos que foram somente utilizados na nossa análise somente os códigos dos participantes que autorizaram seu uso.

Este estudo foi conduzido em duas etapas: (1) análise dos códigos-fonte disponibilizados pelos alunos participantes para identificar se e como eles reusaram os exemplos; (2) e entrevistas (gravadas em áudio) com os seis alunos que aceitaram participar desta etapa.

Durante a primeira etapa, analisamos os níveis de similaridade entre o código da disponibilizado pela professora como exemplo e o código-fonte dos trabalhos desenvolvidos pelos

alunos. Para tal, nós utilizamos a ferramenta JPlag[19][20], desenvolvida e mantida pelo Departamento de Informática da Karlsruhe Institute of Technology, destinada à identificação de plágio em códigos-fonte.

Os exemplos apresentados em aulas anteriores e que poderiam ser de alguma forma, reusados pelos alunos eram os seguintes:

1. O código-fonte e projeto de tela de um programa que contava a quantidade de divisores de um número;
2. O código-fonte e projeto de tela de um programa que, dados 2 números inteiros, calculava a média aritmética dos múltiplos de 5 existentes entre eles (exclusive);
3. O código-fonte e projeto de tela de um programa para escrever os primeiros 20 termos da série de Fibonacci.

Em relação aos exercícios solicitados aos alunos, estes eram:

1. Escrever um programa que leia um número inteiro (X) e verifique se este número é primo ou não.
2. Escrever um programa que, dados 2 números inteiros (A e B), calcule a quantidade dos: múltiplos de 3, se A for ímpar; dos múltiplos de 4, se B for par.
3. Escrever um programa que calcule o somatório dos *n* termos da série de Fibonacci, sendo *n* fornecido pelo usuário.

Durante esta etapa, observamos que diversos alunos reusaram os códigos produzidos pela professora, destacando-se o exercício relacionado à sequência de Fibonacci, no qual a maioria dos alunos não fez mudanças consideráveis em relação ao exemplo, ou deixaram em seus códigos, inclusive, trechos desnecessários. Portanto, decidimos usar este exemplo (relacionado à sequência de Fibonacci) e os códigos-fonte feitos para este exercício como base para a etapa a seguir.

[19] https://jplag.ipd.kit.edu/

[20] No decorrer da análise deste estudo, observamos certa instabilidade no aplicativo Moss, usado durante o estudo anterior. Optamos por usar a ferramenta JPlag, que faz a identificação de similaridade de modo semelhante ao Moss.

O exemplo que havia sido apresentado aos alunos algumas semanas antes da solicitação do trabalho consistia em um programa que escrevia em uma lista os 20 primeiros termos da sequência de Fibonacci. Já a tarefa solicitada pedia um programa que somasse os *n* primeiros termos da sequência, e apresentasse como resultado somente esta soma, sendo *n*, um valor informado pelo usuário. Em diversos dos casos nos quais ocorreram reúso, os alunos mantiveram o padrão de nomes de variáveis X, Y e Z, utilizado pela professora, e mantiveram no projeto de tela o componente gráfico responsável por mostrar os termos da sequência, embora este não fosse mais necessário para o programa solicitado. Um exemplo deste caso por ser visto na Figura 5, onde o comando *ListBox3.AddItem(Z)*, presente no exemplo, era desnecessário ao programa solicitado.

No entanto, nosso interesse principal não é abordar questões relacionadas a plágio, ou questionar a quantidade de linhas reusadas pelo aluno, mas sim compreender como o reúso foi feito e se o programador iniciante se apropriou do código reusado, compreendendo seu significado e suas funcionalidades.

Exemplo

```
Dim CONT, X, Y, Z As Integer
X = 0
Y = 1
ListBox3.Clear
ListBox3.AddItem (Y)
For CONT = 1 To 20
Z = X + Y
ListBox3.AddItem (Z)
X = Y
Y = Z
Next
```

Código produzido pelo Participante 1

```
Dim cont, x, y, Z, Total As Integer
x = 0
y = 1
n = InputBox("Quantos numeros na serie de fibonacci?")
ListBox3.Clear
ListBox3.AddItem (y)
For cont = 1 To n
Z = x + y
ListBox3.AddItem (Z)
x = y
y = Z
Total = Total + Z
Next

ListBox3.AddItem ("TOTAL = " & Total)
```

Figura 5: Comparativo entre o exemplo fornecido pelo professor, e o código produzido pelo aluno

Assim, com o objetivo de compreender melhor como este reúso ocorreu, convidamos alguns destes alunos para uma entrevista, durante a qual:

1. Os alunos foram convidados a explicarem o programa criado, seu objetivo, a lógica do cálculo da sequência de Fibonacci, e explicarem (nos casos em que isso ocorria) porque mantiveram o componente gráfico responsável por apresentar todos os termos, quando

o solicitado era apenas a soma dos mesmos, e porque mantiveram nomes tão poucos mnemônicos para as variáveis, tais como X, Y e Z[21].

2. Após, os alunos foram convidados para uma entrevista semiestruturada na qual perguntas a respeito de técnicas de desenvolvimento, (re)uso de exemplos e se e como percebem a possibilidade de uma comunicação através de seus programas (Apêndice B).

Todas as entrevistas foram gravadas em áudio e transcritas para posterior análise.

3.2.2 Perfil dos participantes

Na primeira fase, usando a ferramenta JPlag, identificamos o percentual de similaridade entre 90 códigos-fonte feitos por 31 alunos[22] e o exemplo dado pela professora. Como anteriormente mencionado, observamos que o exemplo relacionado à soma dos termos da sequência de Fibonacci havia sido reusado por muitos alunos e que, em muitos casos, as alterações no exemplo haviam sido mínimas. Portanto, decidimos usar este caso particular como base para este estudo. Dos 31 alunos participantes, 30 deles haviam feito o referido exercício (Fibonacci). Os resultados da análise são apresentados na Tabela 3.

[21] A professora responsável pela turma disse que usou tais termos ao apresentar o exemplo para os alunos, porém tinha consciência que eles iam contra o padrão de nomenclatura recomendável para variáveis.

[22] Ao todo, 31 alunos autorizam formalmente o uso dos códigos-fonte produzidos para o trabalho e assinaram o Termo de Consentimento Livre e Esclarecido (totalizando assim, 90 códigos-fonte).

Tabela 3: Similaridade dos códigos com o exemplo

Participante	% de similaridade
Participante 1	93%
Participante 2	90%
Participante 3	87%
Participantes 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	84%
Participante 14	81%
Participante 15	78%
Participantes 16, 17, 18, 19	75%
Participantes 20, 21, 22	71%
Participante 23	62%
Participante 24	54%
Participante 25	46%
Participantes 26, 27	40%
Participante 28	22%
Participantes 29, 30	18%

Na sequência, contatamos os 18 participantes que haviam disponibilizado seus e-mails e apresentado interesse em participar da segunda etapa. Seis dos alunos contatados aceitaram participar da entrevista, sendo eles os participantes 5, 7, 12, 20, 28 e 30[23]. O perfil dos participantes é apresentado na Tabela 4.

Tabela 4: Perfil dos participantes do estudo 2

Participante	Curso	Experiência em programação	Similaridade
Participante 5 (P5)	Engenharia Civil	Sem experiência	84%
Participante 7 (P7)	Engenharia Civil	Sem experiência	84%
Participante 12 (P12)	Engenharia Civil	Sem experiência	84%
Participante 20 (P20)	Engenharia de Produção	Sem experiência	71%
Participante 28 (P28)	Engenharia Civil	De outras aulas	22%
Participante 30 (P30)	Engenharia de Produção	Sem experiência	18%

[23] Todos os seis participantes entrevistados assinaram o Termo de Consentimento Livre e Esclarecido no qual autorizaram o uso dos dados coletados durante as entrevistas para fins acadêmicos.

3.2.3 Resultados

Conforme apresentado na Tabela 4, dos seis participantes entrevistados, quatro deles possuíam códigos muitos semelhantes ao do exemplo disponibilizado pela professora. Os códigos dos participantes P5, P7, P12 e P20 possuíam, respectivamente, 84%, 84%, 84% e 71% de similaridade, enquanto os códigos dos participantes P28 e P30 possuíam, respectivamente, 22% e 18%.

Em relação ao primeiro grupo, dos que possuíam códigos com alta similaridade, apenas P12 não mostrou traços de apropriação, não sabendo explicar como seu programa funcionava ou explicar como os termos da sequência de Fibonacci eram calculados. Inclusive, o programa encontrado não somava os termos e apenas gerava-os, não atingindo assim o enunciado proposto. Em relação aos demais, um dos casos que mais se destaca foi o de P5, que, já no início da entrevista, afirmou que "*não adianta falar muito, pois esse daí foi praticamente copiado, pois ela já tinha feito isso, seria somente perder tempo, pois eu já tinha entendido como é que era e eu simplesmente adicionei a soma a mais. Nada de extra*". No entanto, o mesmo participante soube explicar diversos aspectos de seu programa, demonstrando ter não somente reusado esse código, mas apropriado-se do mesmo. Este participante afirmou ter reusado o código para economizar tempo, e afirmou que "*em aula, porque não tem muito tempo, às vezes o professor toca monte de coisa e tem que fazer! Mas quando o cara vai tentar aprender por ele mesmo em qualquer tempo livre dele, ele vai tentar fazer, acredito eu, completamente diferente do que professor fez*".

Estes quatro participantes mantiveram o padrão de nomenclatura XYZ, com pouca significação dos termos. Questionamos os participantes sobre o porquê terem usado tal nomenclatura. Com exceção do participante P5, que já havia anteriormente mencionado que manteve o código do exemplo, os demais participantes afirmaram que tais nomenclaturas eram mais genéricas, inclusive referindo-se a elas como abreviações. O participante P20 mencionou que tal escolha poderia dificultar o entendimento de outras pessoas sobre seu código: "*Eu acho mais fácil identificar assim. Fica até difícil para entender por quem não conhece, mas para mim fica mais fácil de entender*". Sobre o componente do tipo *ListBox*, o mesmo encontrava-se presente nos códigos de P5, P7 e P12. O participante P12 não soube explicar porque o item encontrava-se na interface de seu programa, enquanto P7 justificou que, mesmo o programa tendo como objetivo apresentar a soma dos termos, o componente que mostrava tais termos permitia a ele fazer a *validação* do resultado: "*eu informo*

que eu quero ver até o número 20, aí eu vou ter a ordem do Fibonacci do um até 20, aí depois em baixo, eu vou ter o somatório de todos. Aí eu posso, por exemplo, ver se ele ia estar certo. Porque aí eu tenho todos os números e posso conferir para ver se o resultado está certo".

Em relação ao segundo grupo composto por P28 e P30, ambos com programas com baixos índices de similaridade com o exemplo, apesar de ambos terem feito um programa autoral, sem semelhança com nenhum outro feito pela turma, ambos não cumpriam o solicitado e não geravam e somavam os termos da sequência de Fibonacci.

Além deste questionamento especificamente relacionados ao código da sequência de Fibonacci, questionamos os participantes sobre técnicas de desenvolvimento, (re)uso de exemplos e se, e como, percebem a possibilidade de uma comunicação através de seus programas.

Sobre os primeiros passos em direção à construção de uma aplicação, observamos que os participantes iniciam a aplicação através do mapeamento do problema. Este mapeamento do problema inclui um mapeamento E/P/S (entrada-processamento-saída) e uso de papel e caneta como ferramenta de apoio. Sobre os passos para o mapeamento do problema, P7 citou que "*primeiro eu vejo o que eu vou ter como resposta, e o que eu vou ter que informar no programa, e o que já está informado. Eu primeiro vou olhar os dados que já estão ditos, o que está no enunciado, e o que eu vou ter que escrever para fazer ele. E descobrir as variáveis que eu vou ter para depois resolver o programa*". Em relação ao uso de *papel e caneta*, P30 menciona que "*e eu gosto de organizar no papel antes, mas tem um amigo meu que é formado em engenharia de computação, me ajudou a estudar para a prova e me ajudou a fazer os exercícios, e ele também escrevia no papel, então, e ele deu uma dica para eu sempre fazer no papel antes que é mais fácil de resolver*" e P12 diz que "*eu tento primeiro fazer no papel. Eu acho mais fácil esclarecer as dúvidas*".

Sobre uso de exemplos, considerando quais exemplos utilizar, foram citados os exemplos de aulas, na qual se incluem exemplos feitos pelo professor e reúso de códigos desenvolvidos anteriormente. Já sobre por que usar exemplos alguns participantes mencionam que usam para compreender o problema e otimizar o programa. Em relação a compreender o problema, participante P28 diz "*eu vou atrás de exemplos. Até porque muitas vezes eu preciso analisar uma célula de carga, coisa assim, então eu preciso saber algumas variáveis que ela vai me expor. Então eu preciso de exemplos*" e o participante P7 afirma "*eu tenho os exemplos que a professora passou em aula, que ela faz e aí quando eu estou fazendo programa eu dou uma olhada no dela para ver*

qual é a lógica que ela seguiu para chegar naquele resultado". Em relação a otimizar o programa, o participante P20 ainda menciona que durante o uso de um exemplo "*eu vou olhando e vou fazendo em cima do que eu já fiz. Na verdade, nessa hora eu já fiz um programa, então eu vou arrumar o meu*".

Sobre *como* os exemplos são usados, observamos que são usados como modelo de código (o que pode ser corroborado pela sentença falada por P20 no final do parágrafo anterior) e também como reúso de código, o que ocorre quando o programador opta por fazer uma cópia exata de um código e somente fazer alterações no mesmo para que funcione. Em relação ao uso como modelo de código, P12 afirma que "*eu acho mais fácil pegar e usar ele só como consulta, porque senão as vezes eu acabo deixando passar alguma coisa, fica uma conta que não era para estar ali ou alguma variável que não era para estar ali. Então eu acho melhor refazer o que eu preciso e só olhar. Então, eu uso ele como consulta mesmo*". Já em relação a reúso de código, P5 mencionou que "*as primeiras vezes eu sempre copio, mas depois quando tu vai fazendo várias vezes seu cérebro já fixa e aí eu não copio mais. [...] Na verdade, eu faço assim, o programa está aqui em cima, e eu tenho aquela linha embaixo, e eu vou escrevendo olhando o de baixo. Eu mudo poucas coisas, até chegar num ponto que eu não preciso mais olhar para o de baixo, e nos próximos eu já estou fazendo. É assim que eu fiz para aprender programação*".

Outra abordagem detectada para a construção de programas foi tentativa e erro. P20 diz que "*eu não tinha uma base; eu tinha que ir pela intuição. Tentativa e erro né, eu ia fazendo e arrumando*" e P30 diz "*eu vou fazendo lá da maneira que eu acho que vai funcionar, nem sei se funciona esse código, de fato*" (lembrando que o programa deste último participante não atendia o requisito solicitado, apesar de executar sem problemas ou erros).

Por fim, questionamos os participantes sobre se, e como percebem a comunicação através de seus programas. Os participantes citam uma comunicação via código, na qual estariam comunicando-se com outros programadores: "*acho que só estudantes da própria área. Ou alguém que tenha dúvida, que use o meu exemplo como outro exemplo. Até porque a programação, eu acho que quanto mais compartilhada e quanto mais clara for, acho que serve de exemplo para os outros assim como serviu para mim os meus exemplos*" (P12); "*eu acho que a pessoa não vai entender que eu usei X e Y*" (P20). Neste último trecho o participante cita a existência desta ruptura na comunicação, uma vez que a escolha dos nomes das variáveis não favorece o entendimento de

outras pessoas. Já P5 e P28 mencionam uma comunicação via interface: "*acredito que com o público em geral, clientes...*" (P5), "*eu me preocupo com as telas de impressão, não sei se é isso, mas esse é o grande objetivo para a pessoa fazer um entendimento e ver o que está sendo feito ali*" (P28). Por outro lado, P7 fala sobre seu programa referindo-se a si mesmo como o usuário: "**i*rei* *informar um número, e* ***eu ia*** *contar desse número até aquele número na ordem do Fibonacci*", em linha com o que discutimos sobre EUP.

Quanto às características que os participantes acreditam serem perceptíveis a outros foram mencionadas características via código, como a estruturação do código e nomenclatura das variáveis, e características via interface, como a forma como os itens gráficos estão apresentados na interface.

Os dados aqui coletados corroboram o que observamos durante o Estudo 1, sendo possível observar que programadores iniciantes com perfil EUD também reusam códigos-fonte muitas vezes sem compreender seus objetivos e seus aspectos internos. E, assim como no estudo anterior, é possível observar casos onde o reúso não afetou a compreensão do programador sobre seu programa, tendo ele se apropriado da mensagem emitida pelo código-fonte reusado.

3.3 Estudo 3: Reúso de APIs

O estudo anterior nos apresentou informações que corroboram os achados no Estudo 1, no entanto, em nenhum destes estudos houve reúso de APIs, sendo estes, componentes de *software* criados para serem reusados por outros programadores. Conforme observado por Afonso (Afonso, 2015), APIs são interfaces que carregam consigo a mensagem do *designer,* mensagem esta que precisa ser compreendida pelo programador que irá utilizar a API para que este consiga fazer um melhor uso dos recursos disponibilizados. Além disso, observamos no trabalho de Robillard (Robillard, 2009) que a inadequação ou ausência de recursos que possibilitem ao programador aprender sobre a API, é um dos obstáculos encontrados por eles durante o uso destes componentes.

Desta forma, decidimos conduzir um último estudo, observando os aspectos relacionados a reúso e apropriação, quando isso ocorre através deste tipo de componente, conduzido com 20 participantes de um curso de programação oferecido para alunos do ensino médio de escolas de Porto Alegre e região.

O principal objetivo deste curso era oferecer aulas gratuitas sobre lógica, conceitos de programação, desenvolvimento de aplicações para dispositivos *mobile* (usando Android), assim como conteúdo de *marketing*, como *design thinking*, tendências em tecnologia da informação e prototipação. Ao final do curso, os alunos deveriam desenvolver um projeto no qual teriam que propor e desenvolver uma aplicação *mobile* que, de alguma forma, fosse capaz de auxiliar em problemas da sociedade.

O curso foi criado e é oferecido por uma organização sem fins lucrativos em parceria com a PUCRS. Teve duração de cerca de seis meses com aulas três vezes por semana, e foram oferecidas no total 20 vagas para alunos de escolas públicas porto-alegrenses. A seleção dos 20 alunos era feita em três etapas: (1) uma prova on-line sobre lógica; (2) uma avaliação, realizada presencialmente, sobre lógica aplicada a programação; e, (3) uma entrevista.

Durante o desenvolvimento dos seus projetos finais, os alunos se organizaram em cinco times com quatro participantes, e cada time era responsável por definir uma ideia para o aplicativo, decidir detalhes do projeto e desenvolver o aplicativo *mobile*. Ao final do curso, todos os projetos foram apresentados a uma banca avaliadora para escolha do melhor projeto.

Devido ao fato de o principal objetivo do projeto final ser o desenvolvimento de uma aplicação *mobile*, identificamos que tais projetos tinham um grande potencial para reúso de código. Ao construir uma aplicação deste tipo, desenvolvedores precisam usar APIs para realizar algumas tarefas, como, por exemplo, requisitar acesso ao GPS, câmera ou microfone do dispositivo. Por tal razão, decidimos conduzir este estudo com este grupo específico de alunos.

3.3.1 Metodologia de coleta e análise de dados

Tendo como objetivo final do curso a construção de uma aplicação *mobile*, conduzimos entrevistas semiestruturadas na fase final do curso, durante a última semana de desenvolvimento dos projetos (o desenvolvimento do projeto foi feito no período de três semanas e meia). As entrevistas foram conduzidas com os quatro membros de cada grupo ao mesmo tempo, totalizando cinco entrevistas.

Diferente dos estudos anteriores, neste estudo não houve uma etapa de análise dos códigos elaborados, comparando-os com determinados exemplos de código. Nosso objetivo era

compreender como o reúso de exemplos ocorria (ou não) em um cenário no qual os participantes precisavam utilizar diversas APIs.

A entrevista[24] iniciava-se com um breve questionário pelo qual coletamos dados sobre os perfis dos participantes (idade, escolaridade e experiência prévia sobre programação). Após, seguia-se com a entrevista, com contava com seis questões norteadoras (Apêndice C).

Todas as entrevistas foram gravadas em vídeo e transcritas para serem analisadas posteriormente.

3.3.2 Perfil dos participantes

A amostra foi composta pelos 20 estudantes do projeto. Os dados sobre seus perfis estão na Tabela 5. Observa-se que os participantes possuíam, em média, 17 anos de idade, e somente cinco deles eram do sexo feminino. A maior parte dos estudantes (12) estava cursando a 3ª série do ensino médio, seis cursavam a 2ª série e somente dois estavam cursando a 1ª série[25].

Em relação à experiência com programação, nove deles relataram que tinham experiência prévia com programação. Em relação a esta experiência, um deles relatou que já programava a cerca de cinco anos usando HTML e PHP. Outros dois relataram serem também estudantes de um curso técnico de informática, e que já estavam no mesmo há três anos. Outro participante havia tido experiência com a linguagem LUA programando *scripts* para jogos dois anos antes do curso, e em relação aos demais, todos tinham menos de um ano de experiência com HTML, Phyton, JavaScript e C#. Os demais 11 participantes nunca haviam programado antes do curso.

[24] Todos os estudantes autorizam o uso dos dados e assinaram o Termo de Consentimento Livre e Esclarecido para tal. Para aqueles com inferior a 18 anos, a autorização foi fornecida pelo professor responsável pelo projeto.

[25] No sistema de educação do Brasil o ensino médio é composto por três anos, sendo cada ano equivalente a uma série escolar.

Tabela 5: Perfil dos participantes do estudo 2

Participante	Idade	Gênero	Série	Possui experiência prévia?
G1P1	16	M	2ª	Sim
G1P2	17	M	3ª	Não
G1P3	18	F	3ª	Não
G1P4	17	F	3ª	Sim
G2P1	16	M	3ª	Não
G2P2	18	M	3ª	Não
G2P3	16	M	3ª	Sim
G2P4	17	M	3ª	Não
G3P1	18	M	2ª	Não
G3P2	17	M	3ª	Não
G3P3	16	M	1ª	Não
G3P4	16	M	2ª	Sim
G4P1	22	M	2ª	Sim
G4P1	16	M	2ª	Sim
G4P2	17	M	3ª	Não
G4P3	18	M	3ª	Não
G5P1	16	F	2ª	Não
G5P2	17	M	1ª	Sim
G5P3	17	F	3ª	Sim
G5P4	17	F	3ª	Sim

3.3.3 Resultados

Iniciamos a entrevista questionando os participantes em relação aos primeiros passos dados por eles ao iniciarem o desenvolvimento do aplicativo. Sobre isso, observamos que apesar de ser um curso introdutório sobre programação, os participantes apresentaram um alto nível de planejamento.

De acordo com as respostas dos participantes, o desenvolvimento de um aplicativo inicia-se pelo planejamento e pela prototipação do mesmo. Eles também mencionaram que priorizaram as funcionalidades a serem desenvolvidas no projeto de acordo com os seus conhecimentos prévios, o tempo disponível, e a importância da funcionalidade para o projeto, assim como mencionaram a necessidade de distribuir as tarefas dentre os membros do time, alocando tarefas para cada um deles, e de pesquisar sobre as APIs que poderiam ajuda-los no desenvolvimento da aplicação.

Em relação às ferramentas externas usadas por eles, a maioria destacou a necessidade de APIs, sendo as mais mencionadas as APIs do Google e do Facebook, usadas para fazer o *Login* da aplicação. Eles também mencionaram a necessidade de usar algumas bibliotecas desenvolvidas por

outros programadores. Em um dos grupos, um participante mencionou que a API era o ponto central do aplicativo, falando que "*basicamente é API, o nosso programa*" (G2P3). Todos os participantes mencionaram que nunca haviam utilizado APIs ou outra fonte externa de códigos antes do curso.

Com relação à necessidade de usar estas ferramentas, alguns grupos reportaram a relação com a funcionalidade que eles queriam desenvolver, e, para um dos grupos a necessidade ocorreu ao precisar desenvolver um detalhe operacional, como abrir uma janela do tipo *pop-up* ou fazer um tipo diferente de rolagem.

Em relação às formas como eles aprenderam a interagir com essas ferramentas, eles relataram que usaram a documentação da API, assim como tutoriais oferecidos nesta documentação, fóruns on-line, exemplos usados pelos professores do curso, e vídeo tutoriais. Essas informações estão em concordância com o que foi apresentado por Brandt et al. (Brandt, et al., 2009), que observou que a internet era utilizada como ferramenta para apoiar no aprendizado sobre tecnologias com as quais os programadores não estavam familiarizados. Também estão alinhadas com o trabalho de Robillard (Robillard, 2009), que observou que programadores aprendiam a utilizar APIs através de sua documentação, através de exemplos, através de experimentos, através de artigos, e através da ajuda de outras pessoas. Nossos participantes também citaram que seus conhecimentos prévios, aprendidos no decorrer do curso, foram um dos recursos usados.

Também questionamos os participantes sobre como eles procuravam por materiais que pudessem auxilia-los a interagir com as APIs. Descobrimos que era durante a procura sobre como desenvolver determinada funcionalidade que eles descobriam que precisariam de uma API. Somente um dos grupos relatou ter uma ideia sobre qual a estratégia algorítmica necessária, e usou isso para guiar a busca. Neste último caso, o grupo reportou que geralmente eles procuram exemplos após já terem desenvolvido os seus códigos, para verificar se é possível aperfeiçoa-los.

Sobre sua compreensão sobre os códigos reusados, todos os grupos mencionaram ter investido tempo e esforço para tentar entender os códigos, porém contentavam-se em entender somente o essencial para que conseguissem usar esses códigos. Um dos participantes mencionou que "*a gente sabe o que ela faz, mas não sabe como que ela faz*" (G3P1). Outro grupo usou praticamente a mesma frase. Os participantes relataram que, em relação a como eles reúsam exemplos, eles costumavam copiar e colar pequenos blocos de código fonte (*code scavenging*),

copiando códigos até mesmo de vídeo tutoriais. Segundo os participantes, eles reusaram códigos desta maneira, pois percebiam que os exemplos encontrados não eram inteiramente o que eles precisavam, porém somente algumas partes, que eles precisavam adaptar para poder reusar. Eles também comentaram que não possuem total confiança nos materiais disponíveis na Internet "*porque geralmente as linhas vêm quebradas, alguma coisa vem errada, nunca dá pra copiar e colar da internet*" (G2P3). Outro grupo citou que após reusar estes códigos diversas vezes, eles aprenderam como fazer, e conseguiam codifica-los sem a necessidade do exemplo, o que pode ser observado como uma possível apropriação, uma vez que o programador tornou isso parte de si.

Este estudo nos trouxe de volta ao caso abordado nos outros dois estudos anteriores: para usar a API, muitas vezes os programadores iniciantes necessitam de exemplos de códigos-fonte que mostre a eles como esta API deve ser utilizada, o que nos leva novamente ao questionamento sobre como estes exemplos estão sendo usados por eles. Também, tendo em vista a natureza da API, de não usualmente disponibilizar ao programador acesso a seu código-fonte (apenas meios para fazer as devidas chamadas aos recursos disponibilizadas), temos poucos meios para determinar como houve a apropriação de uma mensagem da qual o programador não teve acesso. Em contrapartida, este estudo complementa informações coletadas nos estudos anteriores, mostrando que estes programadores iniciantes têm por hábito e estão dispostos a investir tempo em relação a planejamento e reflexão sobre os programas que serão desenvolvidos.

Na seção seguinte apresentaremos uma discussão baseada nas informações encontradas nos estudos e como tais informações podem nos apoiar na construção de uma proposta que os apoie em atingir os objetivos deste trabalho.

3.4 Discussão

Neste trabalho temos como objetivo principal ajudar programadores iniciantes a compreender o código-fonte que estão usando reusando. Queremos apoiá-los no chamado processo de apropriação, pelo qual farão deste código-fonte reusado algo realmente seu, compreendendo seu funcionamento e como esse funcionamento se relaciona com os objetivos do programa que almejam desenvolver.

Desta forma, como objetivos secundários, visamos:

- Compreender como programadores iniciantes reúsam códigos-fonte de outros desenvolvedores durante o processo de desenvolvimento de um *software*;
- Compreender como que o reúso afeta na compreensão que estes programadores têm sobre o funcionamento de seus programas, determinando se houve apropriação dos códigos reusados;
- Oferecer uma ferramenta conceitual de apoio à apropriação, através da qual os programadores possam posicionar-se como receptores de uma comunicação feita por outro programador através do código (mensagem) que têm diante de si.

Antes de elaborar uma proposta, precisávamos conhecer melhor o cenário relacionado a reúso de código. Ao procurar por trabalhos relacionados ao tema, encontramos trabalhos que nos apoiaram e que nos mostravam que, muitas vezes, a preocupação do programador está somente em se seu código-fonte funcionará e cumprirá o esperado, e não está em compreender como seu código-fonte e os códigos reusados funcionam. Além disso, estes trabalhos nos mostraram que o reúso de códigos-fonte por programadores é uma ação comumente realizada, tanto por programadores iniciantes quanto por experientes. Por fim, o trabalho de Afonso (Afonso, 2015) lança uma luz em direção à importância de compreender a mensagem de metacomunicação presente em códigos-fonte de APIs, no entanto, seu foco está na emissão desta mensagem, por parte do *designer* da API, e não na recepção desta mensagem, por parte do programador. E não foram encontrados trabalhos que visassem observar como programadores iniciantes interpretavam seus códigos-fonte quando estes foram construídos usando trechos de códigos de outros programadores.

Considerando os estudos realizados, observamos nos estudos 1 e 2 que o reúso de exemplos pode afetar na compreensão que o programador tem sobre seu próprio código-fonte, sendo possível observar casos em que o programador não soube explicar como um código-fonte, escrito por ele próprio, funcionava. Porém, estes mesmos estudos nos mostram que outros programadores foram capazes de se apropriar do código-fonte reusado, incorporando-o a seu programa e compreendendo a relação disto com seus objetivos. Para nós, a apropriação é o produto final que visamos alcançar, criando cenários onde os programadores têm consciência sobre o funcionamento de seus códigos-fonte, mesmo que eles tenham sido desenvolvidos usando as “palavras” de outro programador.

Com os estudos realizados, observamos, também, que o reúso de código-fonte por parte de programadores iniciantes ocorre de formas diversas e por motivos diversos. Em relação as formas como ocorrem, eles podem ocorrer na forma de *design scavenging*, onde um grande bloco de código-fonte é copiado e usado como um esqueleto para o desenvolvimento de um programa e também na forma de *code scavenging*, onde os blocos copiados são menores e pontuais. Uma observação interessante é a de que, em diversas situações, os programadores citaram preferir realizar a cópia do código-fonte de forma manual, copiando linha-a-linha, de forma a evitar que trechos de código desnecessários sejam adicionados aos seus códigos-fonte, atividade que os obriga a efetuar uma leitura também linha-a-linha, para desta forma, efetuar a cópia.

Em relação aos motivos que levam programadores-iniciantes a reusar exemplos de código-fonte, observamos que, em diversos casos, os exemplos são buscados como uma forma de compreender o problema que estão enfrentando ou até mesmo para otimizar códigos já feitos. Porém, após escolher o exemplo adequado, a quantidade de tempo disponível para o desenvolvimento torna-se um fator decisivo para que o programador decida quanto tempo irá investir tentando entender como este exemplo funciona.

Mas apesar disso, observamos que os participantes estão dispostos a investir tempo para planejar o desenvolvimento de seus programas, sendo citado por eles o mapeamento de entradas, processamento e saídas e, frequentemente, usando de algum tipo de prototipação para apoiá-los. Além disso, foi observado que alguns destes programadores tem consciência de que em determinadas situações o usuário do programa que fizeram pode vir a ser outro programador, que usará seu programa como um exemplo, e reiniciará o ciclo de interpretação deste exemplo (Figura 6).

Deste modo, observamos a possibilidade de atuar durante esta etapa de planejamento, adicionando uma nova etapa neste processo, etapa esta que apoie estes programadores enquanto estão realizando a “leitura” do código-fonte que almejam reusar. Programadores iniciantes estão usando programação para diversos fins, e acreditamos que os códigos-fonte produzidos por eles tragam, além de uma solução computacional para um problema, também uma mensagem que expressa suas intenções, preferencias de codificação, e até mesmo, traços de sua personalidade. Visamos atuar no processo de compreensão de códigos-fonte que serão reusados, de forma a apoiar este programador a melhor visualizar e compreender a mensagem que foi expressa pelo

programador que escreveu este código-fonte, e apoiando eles em uma reflexão sobre como essa mensagem pode ser adaptada de forma a se alinhar com a mensagem que ele deseja emitir.

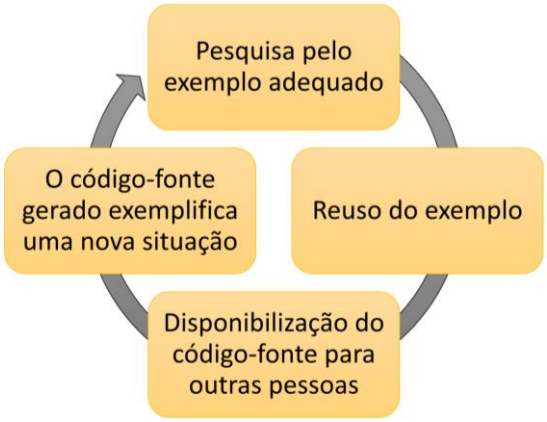

Figura 6: Ciclo de reúso de exemplos

A partir destas observações e das premissas usadas neste trabalho, elaboramos a proposta de uma ferramenta de apoio a apropriação que possa ser utilizada por programadores iniciantes quando eles precisam utilizar um exemplo de código. Os detalhes sobre nossa proposta são apresentados na seção a seguir.

4 FERRAMENTA CONCEITUAL PARA APOIO À COMPREENSÃO E APROPRIAÇÃO DE CÓDIGOS-FONTE

A comunicação pode ser mediada por computadores, considerando-se estas máquinas como uma forma de mídia (Kammersgaard, 1988) (De Souza, 2005) (Georgakopoulou, 2011), por meio da qual pessoas podem se comunicar umas com as outras. Kammersgaard (Kammersgaard, 1988) diz que, na comunicação humana, podemos fazer distinção entre os níveis de expressão e de significado. O emissor da mensagem usa signos para expressar o significado ao receptor, e este receptor interpreta esses signos para entender o significado. Na CMC, De Souza (De Souza, 2005) afirma que, por meio dos signos presentes na interface de um sistema, o designer comunica aos usuários suas intenções, cabendo a este usuário interpretar o discurso que se manifesta de forma explícita e implícita na interface.

Observando esta comunicação do ponto de vista da HCC*,* área que abrange estudos que se aplicam a qualquer campo onde pessoas interagem diretamente com artefatos computacionais (Jaimes, et al., 2007), podemos abordar a tradicional CMC através de uma nova perspectiva: a perspectiva do código-fonte, que atua como interface, carregada de símbolos e significados, e o ponto de vista do programador, que atuando como *designer*, precisa através desta interface manifestar seus objetivos ao usuário, e o ponto de vista do programador, que atuando como usuário deste código-fonte, precisando aprender a interagir com ele, compreendendo a mensagem que seu *designer* expressa, e assim, compreendendo como usá-lo.

Observamos que, durante a construção de artefatos computacionais, a equipe de programação habitualmente reúsa códigos de exemplos, incorporando ao seu discurso o discurso de um terceiro, mas não necessariamente compreendendo totalmente o significado deste discurso incorporado. Pesquisas mostram que programadores, mesmo os mais experientes, têm por hábito ignorar a etapa de compreensão do exemplo a fim de atingir seus objetivos mais rapidamente (Maalej, et al., 2014) (Gaspar & Langevin, 2007). Para nós, um dos problemas do reúso de exemplos/códigos ocorre quando os programadores não se apropriam da mensagem que está sendo expressa pelo trecho que estão reusando, e apenas replicam algo que nem mesmo eles compreenderam, havendo assim uma ruptura na comunicação desde o ponto da emissão, onde o próprio emissor não compreende a mensagem que está expressando.

Através da apropriação tornamos nosso algo que pertence a outro(s) (Wertsch, 1998) e assim construímos identidades que são moldadas pela forma como incorporamos o discurso pertencente ao outro no nosso próprio discurso (Coastley e Doncaster (2001) apud (Abasi, et al., 2006)). Consideramos a apropriação um processo que vai além da simples incorporação: ao nosso ver, a apropriação abrange a interpretação e compreensão desta mensagem antes da inclusão da mesma à mensagem que o programador quer expressar.

Umberto Eco (Eco, 1968), discute a questões sobre autoria e interpretação, afirmando que obras (poéticas, textuais, plásticas, musicais) são obras abertas, nas quais o autor oferece aos que dela desfrutam, algo que pode e será interpretado de forma e por pessoas diferentes. Assim, cada interpretação gera uma nova obra de autoria do intérprete. O autor cita que "todas as interpretações são definitivas, no sentido que cada uma delas é, para o intérprete, a própria obra, e temporárias, no sentido que cada intérprete sabe da necessidade de aprofundar continuamente a própria interpretação". Observamos que estas mesmas questões são pertinentes ao reúso de código-fonte, e que ao nos apropriarmos do discurso presente em um código-fonte reusado, e fazermos disso nosso, o produto final gerado pela interpretação decorrente do processo de apropriação, é também de nossa autoria. Desta forma, observamos que, ao optar por reusar um código-fonte, o programador deve atribuir a ele sua própria interpretação, visualizando as possíveis formas de transformar este código-fonte em algo novo, interpretação essa que pode ser alterada ou aprofundada a cada nova interação.

Assim, temos como objetivo mediar este processo de interpretação e compreensão de códigos-fonte, de forma a apoiar estes programadores iniciantes durante o processo de apropriação. Para isto, usamos como base a teoria da Engenharia Semiótica, e o *template* de metacomunicação oferecido pela teoria. Optamos pela Engenharia Semiótica como teoria norteadora devido a sua visão em relação à metacomunicação ocorrente em interfaces e como a emissão (por parte do *designer*) e a interpretação (por parte do usuário) desta mensagem de metacomunicação irão definir e apoiar as formas como esta interface será utilizada. Além disso, apesar de ser uma teoria originalmente proposta para o contexto de IHC, observa-se que métodos e abordagens originalmente propostos para esta área vêm se mostrando efetivos em apoiar desenvolvedores de *software* durante diversas etapas do processo de desenvolvimento de

software, incluindo-se como uma teoria colaboradora para a área de HCC (De Souza, et al., 2016) (Afonso, 2015) (Bastos, et al., 2017) (Ferreira, et al., 2015) (Lopes, et al., 2017).

Os detalhes sobre como se propõe efetuar esta mediação são apresentados nas seções a seguir.

4.1 Descrição da Ferramenta

Conforme observamos durante os estudos, o reúso de código-fonte é uma prática comum dentre programadores iniciantes, destacando-se as práticas de *code scavenging* (quando há o reúso de pequenos blocos de código-fonte) e *design scavenging* (quando há o reúso de um grande bloco de código-fonte que é usado como um esqueleto para o desenvolvimento de um novo código-fonte). Estas práticas levam estes programadores a, em muitos casos, não compreender como, de fato, os códigos-fonte de seus programas atuam, além de muitas vezes, escrever tais códigos-fonte sem a devida atenção em relação a detalhes que, no futuro, possam apoiar sua reinterpretação sobre o código-fonte (caso precise utilizar ele em um momento futuro) e a interpretação e apropriação por parte de outros programadores.

Os estudos realizados também apontaram que, apesar de o hábito de realizar um *mapeamento de ideias* ser comum entre os programadores iniciantes, através do qual, antes de escrever o respectivo código-fonte, eles listam e organizam as características esperadas do programa que querem construir, tais como entradas, saídas e objetivos, o fator *tempo* foi apontado como um dos motivos pelos quais eles optam por reusar soluções prontas e reusá-las sem total compreensão sobre o funcionamento destas soluções.

Baseando-nos em nossa teoria norteadora, a Engenharia Semiótica, esse código-fonte carrega consigo aspectos comunicativos, ou seja, através do código-fonte, passamos uma mensagem para alguém, sendo este alguém, neste contexto, outro programador que irá atuar como usuário deste código-fonte. Observamos que a Engenharia Semiótica oferece teorias e métodos que apoiam o *designer* no desenvolvimento e análise do discurso que está sendo emitido, porém não há ferramentas que apoiem o usuário posicionando-o como “ouvinte” deste discurso. Observamos que, por exemplo, primeiramente nós aprendemos a ouvir, aprendendo quais sons são usados em nosso idioma e de que maneira se organizam, e somente após, aprendemos a falar (Ferrante, et al.,

2009). Portanto, nossa proposta visa apoiar os programadores iniciantes a “ouvir” o discurso existente neste código-fonte.

Desta forma, o objetivo deste trabalho é ajudar programadores iniciantes a compreenderem o código que estão (re)usando. A fim de alcançar este objetivo oferecemos uma ferramenta conceitual para apoiá-los durante a apropriação de um código-fonte, com a qual estes programadores posicionam-se como receptores de uma comunicação feita por outro programador através do código (mensagem) que têm diante de si.

Para tal, propomos o uso do *template* de metacomunicação fornecido pela Engenharia Semiótica (De Souza, 2005), originalmente oferecido para que *designers* de sistemas interativos pudessem refletir sobre a mensagem passada ao usuário através de seu preposto. O *template* oferecido pela Engenharia Semiótica é apresentado a seguir:

“Este é o meu entendimento de quem você [usuário] *é, do que aprendi que você* [usuário] *quer ou precisa fazer, de que maneiras prefere fazer, e por quê. Este, portanto, é o sistema que projetei para você, e esta é a forma como você pode ou deve utilizá-lo para alcançar uma gama de objetivos que se encaixam nesta visão”.*

De modo semelhante ao apresentado pela ferramenta *SigniFYIng Message* (seção 2.4), propomos o uso do *template de metacomunicação* para apoiar o processo de CMC, porém, diferente do que foi proposto pelo *template* e pelo formulário oferecido pela ferramenta (Anexo A), não queremos posicionar o programador como *designer* do código-fonte que está sendo desenvolvido, mas sim como usuário do código-fonte que está sendo utilizado. Com isso, queremos apoiar *a* compreensão de códigos-fonte, e, assim, mediar a apropriação destes pelo programador que os almeja reusar. Para facilitar o uso do *template* para este fim, o dividimos e identificamos os objetivos de cada uma das questões levantadas, sendo assim definido:

- *Este é o meu entendimento de quem você é*: Refletir sobre possíveis usuários para o programa gerado pelo código-fonte;
- *O que você quer ou precisa fazer, de que maneiras prefere fazer, e por quê:* Especificar o objetivo e domínio do programada gerado pelo código-fonte usado como exemplo;
- *Este, portanto, é o sistema que projetei para você, e esta é a forma como você pode ou deve utilizá-lo*: Detalhar o funcionamento do código-fonte e o papel das Entradas e Saídas.

Com base nos pontos identificados, propomos o uso de um questionário com perguntas que visam guiar o programador na compreensão de determinados aspectos do código-fonte que ele deseja utilizar.

O questionário elabora as seguintes perguntas:

- Quem seria um possível usuário final para o programa deste código?
- Qual o objetivo final deste programa?
- Em quais situações ele pode ser usado?
- Quais as informações que o programa recebe como entrada e o que elas representam para atingir o objetivo final?
- Quais as operações feitas, e o que elas representam para atingir o objetivo final?
- Quais as saídas apresentadas pelo programa, e o que elas representam dentro do objetivo final do programa?
- Você proporia alterações no exemplo para:

  (1) tornar ele mais claro para você ou para um outro programador?

  (2) tornar ele mais claro para o seu usuário, para que ele melhor compreenda o que o programa faz?

Para analisar a aplicabilidade desta ferramenta durante o processo de reúso de códigos-fonte, foram realizados dois estudos, sendo o primeiro estudo um grupo de foco com programadores iniciantes, no qual eles foram convidados a utilizar o questionário e, após, foram entrevistados sobre suas possibilidades; e, o segundo estudo, uma entrevista feita com professores de cursos universitários da área de computação, aos quais a proposta foi apresentada e suas opiniões coletadas.

4.2 Estudos para análise da proposta

A Figura 7 apresenta a localização destes estudos em relação às etapas desta pesquisa. Como pode ser observado, os estudos foram conduzidos simultaneamente pois, diferente dos estudos anteriores, eles endereçavam participantes diferentes, sob perspectivas diferentes.

Figura 7: Localização dos estudos para análise da proposta na pesquisa

Os referidos estudos e seus detalhes são apresentados nas subseções a seguir.

4.2.1 Estudo 4: Grupo de foco com alunos de programação

Para compreender os aspectos práticos do uso da ferramenta conceitual proposta e coletar a opinião dos potenciais usuários da mesma, conduzimos este estudo com alunos de primeiro ano de cursos de graduação na área de Informática, estudo este no qual a ferramenta proposta foi utilizada e posteriormente, os alunos foram entrevistados, expressando suas opiniões. Os detalhes sobre a metodologia e resultados são apresentados a seguir.

4.2.1.1 Metodologia de coleta e análise dos dados

O estudo iniciava-se pela apresentação dos objetivos, assim como uma breve explicação aos participantes sobre etapas que seriam conduzidas. O grupo de foco foi conduzido em três etapas, ocorridas sequencialmente em uma mesma tarde:

1. Na primeira etapa os participantes[26] responderam a um questionário para levantamento dos seus perfis e que contava também com questões sobre hábitos de programação por eles adotadas (Apêndice D).
2. Na segunda etapa os participantes foram convidados a escolherem um exemplo de código e utilizarem a ferramenta, respondendo às perguntas propostas pela mesma, considerando o exemplo escolhido. Os exemplos foram disponibilizados nas linguagens de programação C e Java (Apêndice E) e abordavam os seguintes tópicos:
    a. Sorteio de X valores (sendo X um valor especificado pelo usuário) dentro de um intervalo de 1 até N (sendo N um valor especificado pelo usuário), sem que haja o sorteio de números repetidos.
    b. Implementação de uma lista simplesmente encadeada para armazenamento de valores inteiros informados pelo usuário, e, ao final, apresentação dos números contidos na mesma.
    c. Criptografia de um texto informado pelo usuário, usando a técnica de criptografia Cifra de César, usando um número inteiro, informado pelo usuário, para definir a rotação do alfabeto.

    Todos os códigos de exemplo traziam consigo uma breve descrição do objetivo do programa (semelhante às descrições apresentadas anteriormente). Os códigos continham variáveis com nomes pouco mnemônicos, não usavam sinal de chaves ({ }) quando era possível, e não estavam completamente indentados (a falta de indentação não prejudicava de forma alguma a execução do programa, que funcionava corretamente e atingia o objetivo proposto)[27]. Optamos pelo uso de tais características, pois queríamos observar como isso impactaria nas respostas dos participantes na última pergunta do questionário

[26] Todos os participantes do grupo de foco assinaram o Termo de Consentimento Livre e Esclarecido no qual autorizaram o uso dos dados coletados durante as entrevistas para fins acadêmicos.

[27] A indentação de código, assim como os sinais de chaves, facilitam a compreensão do programador sobre a estruturação do código e a hierarquia dos comandos (quais comandos dependem do resultado ou da execução de comandos anteriores para serem executados).

(sobre possíveis alterações que fariam no código para melhorar a compreensão do mesmo).

3. Na terceira e última etapa, o grupo de participantes foi entrevistado em conjunto (entrevista gravada em vídeo), entrevista essa semiestruturada, com perguntas que nos possibilitaram coletar suas opiniões sobre o uso da ferramenta (Apêndice F).

4.2.1.2 Perfil dos participantes

Participaram do grupo de foco oito programadores iniciantes, estudantes dos primeiros níveis de diferentes cursos de graduação da área de Informática, recrutados por conveniência. Seus perfis são apresentados na Tabela 1, sendo a idade variada entre 18 e 26 anos e todos os participantes do sexo masculino.

Tabela 6: Perfil dos participantes do estudo 4

Participante	**Idade**	**Curso**	**Semestre**
P1	18 anos	Engenharia de Computação	2º
P2	19 anos	Engenharia de Computação	1º
P3	25 anos	Engenharia de Computação	1º
P4	26 anos	Ciência da Computação	1º
P5	22 anos	Engenharia de Computação	1º
P6	18 anos	Engenharia de Computação	2º
P7	19 anos	Engenharia de *Software*	2º
P8	20 anos	Sistemas de Informação	1º

4.2.1.3 Resultados

Questionamos os participantes sobre seus primeiros passos na construção de uma aplicação, e, segundo os participantes, o desenvolvimento de uma nova aplicação é iniciado pelo planejamento e compreensão do problema. Destacamos aqui os comentários dos participantes P4, que disse que costuma "*planejar a estrutura para se organizar*", P6, que disse "*primeiramente mentalizo a resolução do problema, em seguida, faço o teste de mesa e por fim, implemento a solução testando sua funcionalidade*" e de P7 que tem por hábito "*desenhar um esboço de arquitetura que utilizarei posteriormente na implementação*". Também observamos a necessidade de compreensão de detalhes técnicos envolvidos no problema, tais como bibliotecas necessárias e detalhes

relacionados ao sistema operacional, mencionado por P1: "*Definir qual é o compilador correto para o sistema operacional, além de quais bibliotecas e funções posso usar*".

Sobre a origem dos exemplos reusados, eles vêm geralmente de exemplos da internet e também exemplos das aulas. Sobre como o reúso ocorre, observou-se que os alunos costumam adaptar os exemplos, a fim de adequá-los ao novo contexto necessitado por eles. O participante P5 mencionou "*procuro um exemplo de código e adapto até compilar*". Já o participante P3 citou o reúso de exemplos como algo que só deve ser feito após já se ter uma compreensão sobre como estes códigos funcionam: "*procuro evitar utilizar exemplos, pois embora não seja interessante recriar a roda, como um programador iniciante penso que, ainda que futuramente não vá ter esse hábito, atualmente devo saber como se faz a roda*".

Em relação às situações em que eles percebem que precisam de exemplos, foram citados casos de problemas complexos e bloqueio intelectual, ou seja, quando eles não sabem o que devem fazer, por onde começar, ou não conseguem mais evoluir a partir de certo ponto. Um dos participantes ainda citou que costuma recorrer a exemplos "*após tentar solucionar a primeira vez e não obtiver sucesso*" (P6). Também foi mencionado pouco domínio sobre a problema/tecnologia como, por exemplo, pouco conhecimento sobre a linguagem de programação empregada, conforme P8: "G*eralmente quando estou fazendo algum projeto que não entendo muito do assunto e das linguagens de programação utilizadas*".

Cinco dos oito participantes mencionaram que já haviam em algum momento utilizado um exemplo sem entender completamente como ele funcionava. A justificativa dada por eles foi pouco prazo disponível para concluir um projeto ou para casos específicos, como leitura e gravação de dados em arquivos texto. Eis, alguns dos relatos dos participantes: "*Por muitas vezes acabo jogando no código métodos que não serão de fato utilizados para solução do problema, e, portanto, a compreensão deles sobre eles é bem menor*" (P1); "*Sim, no desenvolvimento de um sistema web, para fazer determinada funcionalidade, apenas copiei um código encontrado no Stack Overflow que fazia exatamente o que estava tendo dificuldade em fazer. Não entendi completamente a funcionalidade dele*" (P3). Ainda neste âmbito, os participantes mencionaram que tentam entender o código que irão reusar, sendo dito inclusive por um deles (P6) que após a finalização dos prazos, costuma revisar o código para compreender melhor os trechos onde houve reúso: "*em alguns trabalhos com prazos apertados a última saída é pegar alguns exemplos para ao menos entregar o*

trabalho e após a data trabalhar em cima do mesmo com mais tempo". Um dos participantes mencionou que, nestes casos, é comum que códigos desnecessários fiquem contidos no programa. Os participantes que mencionaram não reusarem códigos sem antes compreendê-los, falaram que usam exemplos apenas para consulta e que usam "*justamente para entender o funcionamento do que é necessário*" (P5) ou que ao menos tentam usar o código após ter uma maior compreensão do que o mesmo faz.

Quanto à possibilidade de, por meio do programa criado, eles estarem se comunicando com alguém, os receptores mais frequentemente mencionados foram outros programadores. P5 mencionou que "*chamamos os meios para programar de linguagem de programação, então acredito sim que existe uma comunicação com alguém, seja uma pessoa que vai ler seu código ou um computador que irá compilar o código*". P2 mencionou que estaria se comunicando "*provavelmente com um programador que algum dia terá que ler o código, e assim, através de comentários //, consiga entender melhor o que está se passando ali*". Também foi mencionada a comunicação com o usuário, dando destaque para a citação de P1: "*acredito que com a pessoa que irá usar o programa no final das contas. Se eu faço uma aplicação pra ajudar um comerciante a organizar seu estoque por exemplo, eu espero que minha aplicação se comunique com ele da forma mais fácil dele compreender*". Um dos participantes (P8) citou ambas as formas de comunicação, observando que, dependendo de como e quem utilizará o programa, haverá um nível diferente de comunicação: "*Com a pessoa que utiliza a aplicação e/ou o programador que lê o código*".

Por fim, sobre a possibilidade de haver essa comunicação implícita nos seus programas, usando como base seus últimos programas desenvolvidos, todos os participantes mencionaram que já haviam, sim, pensado sobre isso, preocupando-se com indentação, comentários, e uso de variáveis e funções mnemônicas em seus códigos-fonte para facilitar esta comunicação. P3 justificou o uso de tais práticas: "*sempre penso nisso, por acreditar ser um hábito essencial para o bom desenvolvimento de softwares extensíveis e manuteníveis*". Este mesmo participante mencionou que além de outros programadores, ele mesmo pode necessitar usar tal código no futuro, e que comentários poderão ajudá-lo a entender rapidamente o que ele estava pensando durante a construção do sistema.

Em relação ao uso da ferramenta proposta, todos os participantes mencionaram que as informações do questionário os ajudaram a mapear outras formas de reúso. P1 mencionou "*No caso*

que era a lista encadeada, eu poderia colocar outro campo, por exemplo 'nome', 'idade', e aí simularia uma 'pessoa'. No caso ali era uma estrutura, mas adaptando para classes, eu poderia imaginar como uma pessoa". Nesta situação o participante claramente menciona a adaptação deste código-fonte para um novo contexto. Sua afirmação foi complementada por P8: "*eu responderia mais ou menos o que o P1 também respondeu, reaproveitar o código para fazer alguma outra coisa*". P3 menciona que as perguntas em relação a quais as entradas e as saídas e como elas se relacionam com o objetivo final do programa apoiam o entendimento do programador sobre o que o código está efetivamente fazendo: "*eu acho que as perguntas mais cabais seriam aquela que pergunta os objetivos, as entradas, e as saídas, se elas exercem o objetivo final do programa. Acho que com elas tu consegue mapear para utilizar*". E o participante P7 complementa afirmando "*Eu acho que se houvesse uma necessidade, seria uma forma um pouco mais, pelo menos para mim, seria mais intuitivo de verificar, para aquela necessidade, como eu poderia utilizar o código que tu nos apresentou... Então acho que seria mais simples*", sendo necessidade a qual ele se refere a necessidade de reusar tal código.

Ao serem questionados sobre problemas encontrados nos programas e quais possíveis modificações poderiam torná-los de mais fácil entendimento, foi mencionada a estruturação do código-fonte, incluindo a modularização e o posicionamento dos sinais do tipo chaves ({ }) que determinam, em algumas linguagens de programação, onde um determinado módulo de código inicia e encerra, e quais os comandos pertencentes ao módulo. Algumas das menções dos participantes a este ponto: "*Eu acredito que se fosse mais separado, que se a função principal fosse mais alto-nível e tu separasse em funções certinho, tipo, definir qual o próximo da lista, etc, o próximo nodo que ele vai referenciar, seria mais fácil de entender* " (P1), "*Acho que os nomes das variáveis poderiam também ser mais explicativos*" (P7) e "*E uma coisa que eu reparei, na do sorteio, no finalzinho, o* for *estava escrito todo em uma linha. Isso não modifica nada no funcionamento, mas tem gente se ver isso já 'buga'. Tem uns colegas meus, que eu, quando escrevo um* for *ou* if*, alguma coisa, eu deixo a chaves do lado e fecho embaixo, e daí tem gente que olha e diz 'ah, porque que tu não abre embaixo e fecha embaixo?'... E tem gente que já se perde com isso. Imagina um* for *que junta com um* print*, tudo na mesma linha, acho que alguém já poderia não entender*" (P8). Um dos participantes (P4) mencionou que, ao modularizar o programa, o programador pode focar apenas em compreender os parâmetros necessários para o uso do módulo. Sobre isso ele diz que "*tem partes ali que ele faz mais de uma vez, aí se de repente tivesse uma função e soubesse o que*

seria o parâmetro, seria mais fácil do que tu estar interpretando toda a vez, o que ele está fazendo ali". Desta forma o programador pode encarar cada módulo como uma espécie de caixa-preta, da qual ele não precisa saber o funcionamento interno, mas sim, detalhes em mais alto nível, como: o que o módulo faz, qual a saída apresentada, e quais as entradas que devem ser fornecidas para que ele atinja o objetivo estabelecido. O participante P1 mencionou questões relacionadas ao reúso de componente vs. reúso de fragmentos: "*Uma coisa é tu reaproveitar a classe ou uma função inteira, outra coisa, é no caso ali, onde ele não é nada modularizado, tu pegar aquilo ali e jogar no teu código, aquilo ali só vai te atrapalhar, mas se fosse uma função, se tivesse uma função que adiciona mais um objeto ou um inteiro na lista, e tu aproveitar aquela função, acontece muito, dar ctrl+c e ctrl+v na função, tu nem se dá muito conta do que acontece, tu entende só o alto nível*". Ainda em relação à *estruturação,* participante P4 menciona também questões relacionadas à padronização: "*vamos supor que tenha um método com cada padrão, e aí até tu sair de um padrão e entender o outro padrão, tu já perdeu muito tempo. Então, se tivesse um padrão tu não perderia tempo em estar pensando como é o outro padrão, e aí pode aumentar o teu desempenho também*".

Ainda sobre modificações que poderiam tornar os códigos-fonte mais fáceis de ser compreendidos, os participantes mencionaram que o código deveria ter alguma documentação, referindo-se à presença de comentários para explicar o funcionamento interno do código, nomes mnemônicos para variáveis e funções. Neste sentido, participante P7 mencionou que "*eu não vejo problema na 'gambiarra[28]', contanto que ela seja comentada. Ao menos tu sabe o que a gambiarra faz: 'aqui está uma gambiarra. Ela faz tal coisa'*". P3 menciona que mesmo uma boa estruturação de código pode não ser suficiente: "*às vezes, se eu faço de uma maneira para que eu entenda, não quer dizer que um terceiro vai entender, porque a gente escreve conforme os nossos vícios e entendimentos daquilo e acho que é de suma importância para a programação em geral que as pessoas comentem melhor os seus códigos porque os códigos*".

Considerando como agiriam em relação a estes problemas caso fossem reusar os códigos, o participante P7 afirmou que não arrumaria e outros participantes (P5 e P6) explicaram que o tempo disponível seria um fator determinante para decidir como fariam este reúso. Sobre os motivos para

[28] Solução improvisada para resolver um problema ou para remediar uma situação de emergência.

realizar correções no código ao fazer reúso, os participantes destacaram auto reúso, reúso por terceiros e para solicitar ajuda. Quanto ao auto reúso destaca-se o comentário do participante P4 que cita o possível uso do código no futuro como motivo para fazer as correções necessárias. Segundo ele "*eu iria arrumar, porque eu poderia perder mais tempo depois pensando sobre o que está acontecendo, do que já arrumar e aí não perder mais tempo*". Complementar a isso, P7 mencionou "*Eu penso em mim mesmo daqui a três semanas, por exemplo. Se eu vou conseguir entender, outra pessoa também vai conseguir*". Quanto ao reúso por terceiros, podemos destacar as falas de P6, que menciona que "*o código só está realmente pronto quando qualquer pessoa pode entender*" e P3 que menciona "*os códigos que são desenvolvidos tem muitos remendos, justamente porque a pessoa que está desenvolvendo não está entendendo aquele código ali direito, mas ela tem que fazer uma funcionalidade, ela tem que estender aquele software, ela tem que fazer ele ser mais manutenível, e daí por não entender, vi os remendos e daí fica aquelas 'gambiarras'*". E quanto às correções para solicitar ajuda, destacam-se comentários dos participantes P5 e P7 sobre a necessidade frequente de se solicitar apoio de programador. Eles mencionam "*sempre tu pede a opinião de alguém, tu pede a ajuda de alguém para ajudar para criar o código, então sempre é bom quando alguém olhar o que tu já fez, entender o que tu fez, para poder te ajudar ou poder opinar no teu código*" (P5) e *"[...] aí eu vou olhar o código do cara... 'o que é que isso faz? E porque que tu fez isso?'*" (P7).

Sobre a eficiência do uso da ferramenta em apoia-los a melhor compreender o exemplo que está sendo utilizado e sobre se, e porque o usariam em seus dia-a-dia, os participantes mencionaram que a ferramenta lhes apresentou novas perspectivas. Segundo P8, "*Acho que não a entender, mas sim a raciocinar de uma maneira diferente o que tu está vendo. Porque quando tu vê isso no dia-a-dia, tu vê, 'aqui tem um scanner, aqui tem duas entradas, aqui óh, está gerando um número randômico e imprime o número'. Tranquilo, mas aí tu vai ler e 'quem é que pode usar isso?'... aí tu fica tipo... 'bah, eu nunca pensei nisso!'... eu nunca tinha pensado porque que uma pessoa pode usar um sorteador, ou 'o que a entrada faz, o que a saída faz, e porque elas são importantes?' ou 'no que as entradas são utilizadas dentro do programa?' são coisas que quando tu lê o código ou quando tu estás escrevendo um código tu não para pra pensar. Mas, o que tu deu pra gente pensar, são coisas fora do que a gente está acostumado*". P5 menciona que "*acho que ajuda, porque tem que entender porque a gente faz o código, para o que a gente está aqui, fazendo isso*" e P3 diz que "*acho que seria*

uma coisa essencial no dia-a-dia. Acho que responde as duas perguntas fundamentais: para o que e para quem".

P1 menciona que "*eu acredito que está se perdendo um pouco com esse negócio de metodologia ágil, é uma coisa muito assim, de 'ahh, vamos programar!' e sai programando, e sai tentando resolver um problema um atrás do outro, e não pensa quanto à que direção tu está seguindo e acaba gerando até 'código porco' por causa disso, vai lá e só vai digitando as coisas. Esse questionamento seria bom no final das contas, mas por conta de tu querer ver a solução de imediato, e não se dar ao trabalho de pensar.*" Outros participantes mencionaram questões sobre o tempo investido no uso da ferramenta, porém afirmam que "*na realidade tu não perde tempo, mas ganha tempo. Tu perdes no começo, mas ganha depois*" (P5) e "*eu acho que inicialmente, isso seria mais pelo tempo, mas depois que tu pega o hábito de fazer isso durante o que tu está fazendo, tu já está pensando*" (P8). Finalizando, o participante P8 comenta sobre o uso continuo desta prática, não somente dentre programadores iniciantes, mas entre programadores profissionais, em seus dia-a-dia. Ele menciona: "*é uma boa prática, tu tem que manter sempre às boas práticas, desde que tu aprendes a programar, tu faz, tudo junto, aglomerado e o cara fala, 'separa o que tu está fazendo, dá um espaço em branco entre as funções, dá um espaço em branco para separar o que é entrada do que é saída, começar a comentar...', são todas boas práticas que tu vai mantendo, tu vai aprendendo tudo e no fim faz um aglomerado de conhecimento*".

Com este estudo, pudemos coletar algumas opiniões dos participantes sobre a ferramenta proposta, e, assim, observar as primeiras evidências em relação a sua aplicabilidade e em relação às consequências geradas pela reflexão promovida através do seu uso. Para aprofundar nossa coleta de evidencias entrevistamos também professores de programação, sendo esta entrevista descrita na seção seguinte.

4.2.2 Estudo 5: Entrevista com professores de programação

Durante o estudo apresentado previamente observamos e coletamos a opinião de uma das partes envolvidas no processo de ensino e de aprendizagem: os alunos, sendo, neste caso, alunos de disciplinas iniciais de programação, tidos aqui como programadores iniciantes. No entanto, durante o processo de aprendizado também temos como envolvidos os professores, sendo sua opinião essencial para analisar a viabilidade desta proposta, uma vez que seriam eles que

observariam - na prática - os resultados que esperamos obter com a aplicação da ferramenta conceitual.

Portanto, conduzimos este estudo, paralelamente ao Estudo 4, no qual entrevistamos professores de disciplinas variadas de programação. Os detalhes sobre as entrevistas são apresentados a seguir.

4.2.2.1 Metodologia de coleta e análise de dados

Dando continuidade à nossa análise da aplicabilidade da proposta, conduzimos uma entrevista semiestruturada com professores de programação. As entrevistas foram conduzidas individualmente e gravadas em áudio para futura análise.

Durante as entrevistas foram feitas perguntas sobre hábitos de reúso praticados pelos seus alunos, observados em sala de aula. Além disso, apresentamos a eles nossa proposta e os questionamos sobre suas opiniões a respeito do que lhes foi apresentado. As perguntas norteadoras da entrevista estão disponibilizadas no Apêndice G.

4.2.2.2 Perfil dos Professores

Em relação aos perfis dos participantes, a amostra foi composta por cinco professores de programação, recrutados por conveniência, com perfis apresentados na Tabela 7.

Tabela 7: Perfil dos participantes do estudo 5

Participante	**Tempo como professor de programação**	**Gênero**
P1	23 anos	Feminino
P2	5 anos	Masculino
P3	17 anos	Masculino
P4	21 anos	Masculino
P5	15 anos	Feminino

4.2.2.3 Resultados

Questionamos estes professores em relação a como usualmente trabalham com exemplos de códigos-fonte em sala de aula. Em relação a isto, eles mencionaram que normalmente antes do exemplo, vem uma motivação, podendo esta motivação ser um problema pontual ou até mesmo

um domínio a ser modelado, e, após, este problema é descontruído e um exemplo de código-fonte para solução é construído com a participação dos alunos. Foi mencionado que este exemplo não precisa ser construído com o uso de um código-fonte, podendo ser simplificado e sumarizado fazendo uso de pseudocódigo[29]. Em relação a isso, P4 diz que "[...] *normalmente tu tem um problema para resolver, então eu parto sempre da outra ponta: parto do problema que está sendo resolvido e 'mastigo' ele até ficar muito claro o que tem que ser feito, e aí nesse o 'o que tem que ser feito', finalmente se chega no código ou até ficar claro o suficiente para dizer 'bom, nem precisamos ir para o código, pois todo mundo já entendeu que esses dois laços assim e assim, resolveriam o problema'*".

Em relação a se apresentam ou ensinam algo em sala de aula relacionado a reúso de código-fonte, foi mencionado pelos professores que usualmente referem-se a reúso em sala de aula como reúso de componentes, ou seja, APIs e funções programadas de forma a serem reusadas através de chamadas. Sobre isso, P1 menciona que *"costumo dizer, principalmente agora, vendo a disciplina de Fundamentos, é que a gente tem que fazer métodos pontuais, pensando na reutilização das coisas*". Porém, observamos também que alguns professores reúsam exemplos como forma de introduzir novos conceitos, incrementando um código-fonte, acrescentando nele novas linhas pertencentes a este novo conceito, nos quais se *"começa com passos simples que depois vão incrementalmente dificultando"* (P2). Eles também citam que costumam reforçar em sala de aula a importância de *"documentar muito bem o que eles fizeram, e como utilizaram para depois poder reutilizar"* (P2), referindo-se à documentação de código-fonte para facilitar o reúso no futuro, e que apresentam aos alunos, após introduzir um exemplo, algumas situações nas quais os mesmos conceitos trabalhados pelos exemplos podem ser utilizados. Sobre este último ponto, P4 cita que costuma aborda-lo explicando aos alunos que *"esse jeito de pensar aqui resolvia o problema X, como o problema Y é parecido, talvez um jeito similar resolva"*.

Ainda sobre reúso de código-fonte, questionamos aos professores se eles costumam observar esta prática dentre seus alunos, especialmente através de reúso de fragmentos. Os professores

[29] Pseudocódigo é uma forma genérica de escrever um algoritmo, utilizando linguagem natural que pode ser compreendida por qualquer pessoa sem necessidade de conhecer a sintaxe de uma linguagem de programação.

mencionaram observar tal hábito nos alunos, especialmente dentre os programadores iniciantes, onde muitas vezes o código gerado é se assemelha a uma "colcha de retalhos", usando fragmentos de diversos outros códigos-fonte para gerar assim um novo. Sobre este tópico, P1 menciona que "*por isso, surge a questão de eu não querer prova com consulta, porque senão eles fazem uma colcha de retalhos que mais ou menos se aproxima daquilo que tu quer, e aí tu não consegue avaliar direito porque tu não consegue enxergar o que o aluno sabia daquilo que ele não sabia*". Complementar a isso, é mencionado que os alunos, com certa frequência, fazem este reúso sem compreender (e sem tentar compreender) os códigos-fonte sendo reusados, focando apenas em fazer o código-fonte funcionar, conforme mencionado por P3 e P5: "*sim, o problema é que muitos se baseiam na cópia do exemplo e não na adaptação para solucionar o problema*"; "*às vezes eles até entendem, mas às vezes eu acho que eles não entendem, eles simplesmente copiam, e se está funcionando e faz o que devia fazer, é isso*".

Apresentamos a estes professores a proposta apresentada neste trabalho e pedimos a eles que expressassem suas opiniões e quais as consequências, visualizadas por eles, que o uso da ferramenta conceitual teria. Sobre suas opiniões, dividimos elas em prós e contras levantados pelos participantes. Em relação aos prós, os participantes mencionaram que a proposta apoia o aprendizado, apoia o processo de abstração, apoia as disciplinas futuras (que necessitam deste processo de abstração) e geram insumos para os professores. Em relação ao apoio ao aprendizado e apoio a abstração, foram mencionados "*Pro iniciante sim. O que tu pode pegar é o básico de todas essas áreas... acho que tu pode deixar mais plana, entendeu. Numa linguagem mais simples, para ele entender, para que depois ele possa nessas mais complexas*" (P1) e "*eu acho interessante que eles tentem entender, porque não adianta eles pegaram um trecho pronto e o dia que eles se depararem com o problema eles vão ter que ir lá e copiar novamente. É bom saber fazer né?!*" (P5). Sobre o apoio a disciplinas futuras, P3 menciona que em disciplinas mais avançadas do curso, lecionadas por ele, há a necessidade do uso de formalizações, pois, segundo o participante, exigem que o aluno tenha desenvolvido anteriormente a capacidade de "*pensar de uma maneira mais geral*", sendo este pensamento abstrato sobre seus programados algo que "*ele deveria fazer, mas ele não faz*". Além disso, foi mencionado que a proposta poderia apoiar a geração de insumos para professores, ou seja, apoiar o levantamento de dados e informações que podem apoiar futuramente os professores. Sobre isso P4 menciona que "*se tu quer mudar a vida desses alunos de primeiro semestre eu acho que não consegue, justamente pelo que eu acabei de te falar, mas se tu quer*

coletar informações para informar os professores das maneiras mais eficientes... 'Olha as reações foram mais eficientes para o estilo dois, e as recomendações maiores dizem respeito a nome de variável, pois indentação eles não dão a mínima bola'". Este participante sugere o uso da ferramenta para apoiar a análise de códigos-fonte, comparando diversas soluções para um mesmo problema, e observando como o uso determinadas otimizações de código-fonte podem impactar a compreensão que os programadores têm sobre estas soluções.

Sobre o trecho "*justamente pelo que eu acabei de te falar*", este participante levantou o ponto motivacional para que os alunos usem a ferramenta proposta, sendo este um dos contras observados: "*eu teria muita dúvida se isso vai entrar muito no sangue deles, no sentido de acostumar a se preocupar. Me parece ser o tipo de preocupação que ao se virar de costa volta rapidamente a ser tudo como era antes. Meu medo seria esse*". Este ponto foi mencionado por outros participantes: "*a questão é o que motivaria eles a fazer isso?! Isso é o que eu me pergunto*" (P5). Em contrapartida, P1 menciona que se houver valorização, talvez os alunos passem a se preocupar mais com isso, o que poderia ser a motivação para o uso da ferramenta: "*A gente fala, mas... talvez até um pouco seja uma falha nossa também... a gente fala, mas talvez não cobra tanto como deveria. Se tu valorizar, eu acho que o aluno vai mais atrás. Mas de maneira geral, eles não gostam de escrever muito, e documentação é escrever. Eles acham que é perder tempo com coisas que são fáceis, que na verdade fazem toda a diferença quando tu vai usar*".

Em relação aos demais contras, os professores mencionaram que a proposta seria interessante para apoiar programadores iniciantes, somente, devido ao fato de, segundo eles, a proposta não poder ser usada para apoiar a compreensão de problemas complexos, tratados por programadores de níveis mais avançados. P1, cita que, em alguns casos, problemas complexos são difíceis de serem simplificados, e exigem, para seu entendimento, um conhecimento prévio sobre outros assuntos: "*Tem certos problemas que a solução é complexa, e não vão ser estruturas básicas que vão resolver isso, por mais que tu tente simplificar a linguagem, sem uma boa base anterior, o iniciante não acompanha. Tu não vai conseguir fazer isso, por mais que tu simplifique, porque pressupõe um conhecimento que ele ainda não tem*". Além disso, P1 salienta que, frequentemente, as nomenclaturas usadas dentro do código-fonte fazem referência a nomenclaturas clássicas adotadas por determinado conteúdo, e que mesmo que sejam alteradas, não irão apoiar a compreensão: "*no meio da solução um nome de um método que ele faz todo o sentido dentro da*

técnica, se a pessoa não entende o que é que é a técnica, não vai fazer diferença o quão significativo tu colocou aquilo, porque ela não tem a bagagem suficiente para entender". P2 também salienta esta necessidade de conhecimentos prévios, mesmo dentro do grupo de programadores iniciantes: "*Acho que depende do nível do iniciante, o cara que está recém iniciando, não vou dizer primeiro semestre, mas as primeiras aulas de programação, eu acho que não cabe*". Outro ponto contra observado por P1, é que a proposta não vai conseguir apoiar o programador na enumeração de todos os possíveis cenários de uso do exemplo: "*Então tem vários desafios no que tu colocou ali... enumerar tudo eu acho difícil, eu fico pensando em técnicas às vezes que tu mistura coisas... a gente faz muito isso na pesquisa. Tu aplicaste a técnica para um X, daqui a pouco tu está estudando aquilo e 'bah, mas aquele negócio cabe aqui também. E se eu fizer isso acho que dá um resultado e nunca ninguém pensou nessa relação... nem o cara que fez'*". Porém, neste ponto salientamos que o objetivo da ferramenta não está em fazer o programador pensar em todas as possíveis formas de aplicação de determinado código-fonte, mas sim, direcioná-lo para que faça uma reflexão a respeito destas possibilidades, uma vez que, estas possibilidades são de certa forma, os possíveis usuários da solução que pretendem utilizar.

Fora os prós e contras, observamos também comentários sobre o uso da ferramenta conceitual e seu papel em apoiar o programador em compreender a necessidade de melhorar seu código-fonte e criar a consciência de que seu código-fonte pode vir a ser reusado por outro programador: "*Acho que isso ajuda primeiro a ele deixar o código dele mais claro*" (P1); "*Eu acho que a primeira consequência é ele se dar conta que o código vai ser reutilizado por ele ou por outro, esse é um grande problema já que eles usam o código deles e de outros, só que às vezes eles nem entendem*" (P2); "*então eles também começam a tentar ter terem um padrão para escrever o código, não documentação, mas até como Java mesmo propõe: nomes de métodos ou nomes de variáveis. Eu vejo que tem muitos que não seguem, eles não se preocupam em colocar assim. [...] Às vezes nem indentado está [...]. Se nem usam a indentação que é fundamental para entender a estrutura do código, imagina o resto*" (P5).

Por fim, concluímos com a sugestão dada por P3, que irá ser acrescentada a nossa lista de próximos passos da pesquisa: "*Não sei o quanto tu já analisou os posts do Stack Overflow, aqueles mais bem cotados, mas é bom olhar, pois isso aqui que tu colocou, vai ver se aqueles caras que*

recebem o maior número de notas, que são os que mais contribuem para solucionar dúvidas, como é que eles conseguem explicar um código para ser utilizado em situações e problemas".

Com este estudo, concluímos as investigações iniciais em relação a aplicabilidade da ferramenta conceitual proposta neste capítulo. Os estudos nos apresentaram indícios de que a ferramenta pode colaborar como mediadora do processo de apropriação de exemplos de código-fonte, porém, pode ser refinada e melhorada, de modo a melhor atender as necessidades dos alunos e expectativas dos professores. No próximo capítulo, discutiremos sobre a proposta deste trabalho, de forma geral, usando a nossa teoria norteadora, a Engenharia Semiótica como guia de análise dos resultados obtidos e analisando como este trabalho se difere dos trabalhos relacionados. Ainda, discutiremos sobre os resultados de ambos estudos 4 e 5, avaliando como as observações feitas contribuem para a análise da aplicabilidade da ferramenta e, como esta nos apoia em atingir nosso objetivo de ajudar programadores iniciantes a compreender o código-fonte que estão reusando para, desta forma, poderem fazer um uso mais adequado do mesmo.

5 DISCUSSÃO

Aprender a programar e adquirir o assim chamado Pensamento Computacional (Wing, 2006) pode ser uma tarefa complexa. Os programadores precisam enfrentar diversos desafios, tais como aprender a sintaxe de uma linguagem de programação, aprender a criar soluções estruturadas e escrever estas soluções usando a linguagem aprendida, criando assim um código-fonte, e aprender como este código-fonte será executado (Kelleher & Pausch, 2005). Além destes desafios, os programadores frequentemente precisam compreender códigos-fonte criados previamente por outras pessoas. No caso de programadores experientes, a necessidade de lidar com códigos-fonte produzidos por outros é frequente durante o processo de desenvolvimento de *software* (Roehm, et al., 2012), além de práticas de reúso de *software* serem amplamente difundidas como forma de elevar a produtividade, através do uso de APIs e funções, padrões de projetos, ou blocos de códigos (Krueger, 1992). Já no caso de programadores iniciantes, este público necessita exemplos de códigos-fonte que ilustrem a eles como implementar um algoritmo para solucionar um problema, demonstrar um padrão/estilo de programação, ou apresentem determinado aspecto de uma linguagem de programação (Neal, 1989). Assim como ocorre com programadores experientes, os programadores iniciantes reusam estes exemplos de código-fonte (Neal, 1989), incorporando-os ao seu próprio código, e efetuando as readequações necessárias para que este atinja um novo objetivo.

Para conduzir nossa investigação nos apropriamos da teoria da Engenharia Semiótica e suas recentes contribuições para a área de HCC, deste modo, visualizando este código-fonte como uma interface, através da qual há uma conversa entre o programador que escreveu o código-fonte que está sendo reusado e o programador que está reusando este código. Usando esta perspectiva, nos permitimos ver que este código-fonte carrega um discurso implícito que incorpora as intenções do programador em relação a como, por quem, e onde este código-fonte pode ser usado.

A Engenharia Semiótica fornece métodos e ferramentas que apoiam o *designer* (sendo este *designer,* em nosso caso, o programador) na construção e avaliação deste discurso, mas não especificamente meios que apoiem o usuário (sendo este usuário, um outro programador) a compreender e apropriar-se deste discurso. Deste modo, neste trabalho temos como objetivo principal ajudar programadores iniciantes a compreender o código-fonte que estão reusando para, desta forma, poderem fazer um uso mais adequado do mesmo.

Como objetivos secundários, visamos:

- Compreender como programadores iniciantes reusam códigos-fonte de outros desenvolvedores durante o processo de desenvolvimento de um *software*;
- Compreender como que o reúso afeta a compreensão que estes programadores têm sobre o funcionamento de seus programas, observando se houve apropriação dos códigos-fonte reusados;
- Oferecer uma ferramenta conceitual de apoio à apropriação, pela qual os programadores possam posicionar-se como receptores de uma comunicação feita por outro programador através do código (mensagem) que têm diante de si.

Em relação ao primeiro e segundo objetivos secundários, estes foram endereçados durante a discussão apresentada na seção 3.4. No capítulo 4, apresentamos a ferramenta conceitual que caracteriza nosso terceiro objetivo secundário. Assim, iniciamos a discussão apresentada neste capítulo com foco nos estudos 4 e 5, avaliando como os resultados encontrados contribuem para a análise da aplicabilidade da ferramenta.

Para conduzir essa discussão, vamos remeter a alguns dos pontos apresentados previamente, iniciando por um dos prós apresentados pelos professores entrevistados no estudo 5: o fato de a ferramenta apoiar a abstração. Conforme apresentado pelo trabalho de Hoadley e colaboradores (Hoadley, et al., 1996), a capacidade de realizar uma sumarização abstrata de um determinado código-fonte aumenta a chance de o programador reusá-los. Os autores também observaram em seu estudo que, ao desenvolver uma sumarização abstrata, os programadores ficam mais inclinados a reusarem este código através de invocação de funções, ao invés de clonagem de código. Conforme vimos nos trabalhos de Rosson e colaboradores (Rosson, et al., 2004) e Brandt e colaboradores (Brandt, et al., 2009), os participantes de suas pesquisas ou não reusavam um código alegando que isso "bagunçaria" seus próprios códigos-fonte, ou reusavam sem testá-lo ou compreender previamente como ele funcionava, inserindo assim erros em seus programas decorrentes da adição de trechos desnecessários. Ao reusar um código-fonte por invocação (também referido por Sojer (Sojer, 2011) como reúso de componentes) tais problemas seriam minimizados, senão evitados totalmente. Já no trabalho de Ichinco e Kelleher (Ichinco & Kelleher, 2015), em diversos casos os exemplos fornecidos para auxiliar os programadores em suas tarefas não eram usados por eles pelo fato de não compreenderem como estes exemplos se relacionavam com a tarefa que estava sendo

feita. Ao realizar uma sumarização abstrata, e desta forma ter uma visão mais alto-nível dos objetivos destes exemplos, acreditamos que estes programadores conseguiriam, provavelmente, visualizar a possibilidade de reúso esperada pelos autores.

Porém, muitas vezes os programadores julgam que o conhecimento a nível algorítmico sobre o que determinado código faz é suficiente (Hoadley, et al., 1996), no entanto, este conhecimento em alguns casos se resume na capacidade de traduzir as linhas de código-fonte escritas em uma linguagem de programação, para uma linguagem natural, de mais fácil entendimento, o que não apoia este programador na compreensão em alto-nível esperada que ele atinja. Além disso, mesmo que este programador consiga perceber a necessidade de uma compreensão abstrata do código-fonte, ela pode não é feita porque ele não sabe como atingi-la. Neste ponto, nossa proposta visa atuar como um mediador, que apoie este programador no processo reflexivo necessário para que ele possa atingir o conhecimento abstrato, conhecimento este que irá apoiá-lo de diversas maneiras, incluindo o reúso de *software*.

Um dos questionamentos dos professores, classificado por nós como um possível ponto contra a ferramenta, foi em relação a motivação dos alunos para utilizar a ferramenta. Em relação a este questionamento, observamos a resposta de um dos participantes (P8) do estudo 4, que menciona que a ferramenta traz à tona uma reflexão que deveria ser levada como uma boa prática. Os participantes ressaltaram pontos sobre os benefícios que a "*perda de tempo*" em usar a ferramenta traria, uma vez que este tempo gasto na compreensão do problema seria economizado em situações futuras, nas quais os programadores precisariam investir um tempo considerável reinterpretando esse código. Um dos pontos citados por P8 foi em relação em "*entender porque a gente faz o código, para o que a gente está aqui, fazendo isso*". Nossos estudos para compreensão do problema mostraram, em diversas situações, que os programadores iniciantes estão dispostos a investir tempo no planejamento de suas atividades. No entanto, este tempo é tido por eles como um dos fatores que determinam como irão se portar durante a atividade de desenvolvimento de um código-fonte. Conforme seus prazos, eles decidirão realizar ou não este planejamento e também estudar ou não um código-fonte que desejam reusar, tentando compreendê-lo adequadamente, considerando-se, também, que o reúso é, em muitos momentos, feito com intuito de se economizar tempo no processo de desenvolvimento.

No entanto, vimos durante o estudo 5, que muitas vezes o processo percorrido por um aluno de programação para construir determinado código-fonte não é valorizado por seus professores, que avaliam somente o produto final entregue por este aluno. Sobre isso, a professora P1 citou que "*Se tu valorizar, eu acho que o aluno vai mais atrás".* Complementar a isso, P2 citou que "*muita gente acaba trabalhando assim, que não é a melhor forma. É tu ver a lógica meio que junto com o desenvolvimento, então tu já começa a desenvolver e recém entendeu a lógica. Isso é um problema porque isso faz com que tu produza códigos meio obscuros*". Este fato foi corroborado pelos alunos participantes do estudo 4, podendo ser visto na fala de P1: "*sai tentando resolver um problema um atrás do outro, e não pensa quanto à que direção tu está seguindo*". Tais comentários podem corroborar com um ponto crítico do processo de ensino e aprendizagem de programação, que não visa dar ao aluno o tempo necessário para construção do raciocínio abstrato que irá se fazer necessário durante o momento do reúso.

Em outro dos pontos contra, foi observado o fato de a ferramenta não conseguir abordar todas as possíveis formas de reúso de um código-fonte, porém salientamos aqui que o objetivo da ferramenta não está em fazer o programador pensar em **todas** as possíveis formas de aplicação de determinado código-fonte, pois, como mencionado por P1 durante o estudo 5, alguém poderá fazer um reúso totalmente inesperado de seu código-fonte. Nosso objetivo está em apoiar a reflexão do programador em relação a outras formas de reúso. Durante o estudo 4, os participantes mencionam que a ferramenta os apoiou a mapear as possíveis formas de reúso, e, conforme mencionado por P8, os ajudou a "*raciocinar de uma maneira diferente*". Este participante ainda complementa: *Porque quando tu vê isso no dia-a-dia, tu vê, 'aqui tem um scanner, aqui tem duas entradas, aqui óh, está gerando um número randômico e imprime o número'. Tranquilo, mas aí tu vai ler e 'quem é que pode usar isso?'... aí tu fica tipo... 'bah, eu nunca pensei nisso!'... eu nunca tinha pensado porque que uma pessoa pode usar um sorteador, ou 'o que a entrada faz, o que a saída faz, e porque elas são importantes?' ou 'no que as entradas são utilizadas dentro do programa?' são coisas que quando tu lê o código ou quando tu estás escrevendo um código tu não para pra pensar. Mas, o que tu deu pra gente pensar, são coisas fora do que a gente está acostumado*". Complementar a esta afirmação, P5 diz que a ferramenta ajudou a "*pensar além do código*". Um de nossos objetivos com esta ferramenta era posicionar o programador como usuário de um código-fonte, usuário este capaz de observar aspectos que vão além das questões de sintaxe e semântica, aspectos relacionados a como, quem, e onde este código pode ser utilizado, aspectos que os façam pensar sobre seus

possíveis usuários (sendo estes usuários finais ou outros programadores), e aspectos que os posicionem como ouvintes do discurso implícito no código, identificando as relações dos comandos de entrada, processamento e saída, na composição do objetivo final do código-fonte.

Por fim, os professores envolvidos no estudo 5, citam o fato de a ferramenta colaborar apenas com programadores iniciantes, uma vez que não veriam como problemas complexos, tratados por programadores profissionais, poderiam ser simplificados e explicados em uma linguagem abstrata. No entanto, acreditamos que, ao apoiar programadores iniciantes na criação deste raciocínio abstrato, apoiamos não somente o reúso de código-fonte, mas lançamos luz sobre a metacomunicação presente em códigos-fonte, luz esta que pode refletir-se em benefícios, como, por exemplo, mostrar a estes programadores a importância de manter seus códigos claros, pensando na possibilidade de um terceiro programador precisar reusá-lo. Este ponto foi mencionado pelos participantes P1, P2 e P5, do estudo 5 e também foi salientado pelos participantes do estudo 4.

Em diversos momentos, durante os estudos, podemos ver a grande valorização e foco da programação como um "mero" solucionador de problemas. No entanto, programação pode ser vista também como uma forma de comunicação (Bastos, et al., 2017), carregada de características daquele programador (Monteiro, 2015) (Monteiro, et al., 2015), usada por diversos tipos de profissionais (De Souza, et al., 2016). Um dos participantes do estudo 4 lança uma crítica às metodologias de desenvolvimento atuais, que além de possuírem o foco na solução do problema, usam as chamadas metodologias ágeis: "*eu acredito que está se perdendo um pouco com esse negócio de metodologia ágil, é uma coisa muito assim, de 'ahh, vamos programar!' e sai programando, e sai tentando resolver um problema um atrás do outro, e não pensa quanto a que direção tu está seguindo e acaba gerando até 'código porco' por causa disso, vai lá e só vai digitando as coisas".* Tal questionamento sobre o impacto de metodologias ágeis é abordado por De Souza e colaboradores (De Souza, et al., 2016), que, citando Dyba e colaboradores (Dyba, et al., 2014), observam que além da falta de ferramentas para captura, análise a apresentação de informações sobre as quais desenvolvedores de *software* possam refletir, a maioria dos projetos de desenvolvimento de *software* não apoiam ativamente esta reflexão, nem reservam recursos financeiros e tempo para isso.

O processo de resolução de um problema pode ser visto de diferentes maneiras. Em seu livro, intitulado *The Reflective Practitioner: How Professionals Think in Action,* Donald Schön (Schön, 1983) nos apresenta uma alternativa ao assim chamado racionalismo técnico. Segundo a descrição do autor, o racionalismo técnico define-se pelo entendimento de que o conhecimento profissional é a aplicação de técnicas e teorias científicas a problemas. Ou seja, é visto que, para um determinado problema, haverá uma gama de soluções ou métodos científicos definidos capaz de solucioná-lo, não possibilitando ao profissional questionar ou mudar isso. Schön observa que, na época de sua publicação, este era o modelo dominante abordado pelas Universidades, e propõe uma nova abordagem, denominada reflexão em ação. Nesta abordagem, cada problema é diferente do outro, gerando assim processos de *design* e soluções únicas e o profissional é livre para explorar ideias, compará-las e representá-las de alguma forma. Ao representar essas ideias, o profissional "conversa" com tal representação, em um processo nomeado como conversa com materiais. Segundo o autor, a partir da representação de suas ideias, o profissional obtém uma condição para melhor avaliá-las e verificar se a solução utilizada satisfaz seus objetivos e, se for o caso, ele pode experimentar outros tipos de soluções que melhor os satisfaçam. Neste processo, os materiais com os quais "conversamos" também nos "respondem", um processo denominado por ele de *talk-back.* Ao refletir sobre esta resposta, o profissional pode encontrar novas ideias sobre a situação, o que irá levá-lo para uma reformulação da solução. Ao elaborar tais conceitos, Schön não se referia diretamente a profissionais de tecnologia, mas atualmente seus conceitos vem guiando discussões entre esse público.

Em seu trabalho, Pescio (Pescio, 2006) apropria-se dos conceitos de Schön e discute a existência destes materiais dentro do ambiente de desenvolvimento de *software*, salientando que eles podem se apresentar em diversos formatos, como textos, diagramas ou códigos-fonte. Segundo o autor, os desenvolvedores de *software* deveriam, às vezes, parar e pensar, sem se preocupar com suas métricas de performance, revisando o estágio atual de seus projetos, e realizando as críticas e reformulando suas soluções. Em um trecho de seu artigo, Pescio cita um exemplo relacionado ao estilo de codificação: em muitas linguagens de programação, tem-se a opção de se declarar todas as variáveis no início do bloco, sem a necessidade de atribuir um valor para a mesma (sendo este bloco o código-fonte todo, ou uma função que o compõe). Muitos programadores, que aprenderam a programar usando linguagens que permitiam tal estilo, preferem utilizá-lo, sob a alegação de que isso facilita a identificação e o tipo de cada variável, uma vez que há a necessidade de se olhar

somente o início do código. Segundo o autor, ao fazermos isso estamos “silenciando” o material: se você está perdido em seu código-fonte (ou seja, se você está em outro ponto do código-fonte e não consegue facilmente ver a declaração das demais variáveis), seu material está dizendo a você “*eu estou errado. Meu criador não me deu estrutura suficiente; Eu sou uma lista extensa de instruções. Por favor, me particione em funções menores.*”. Neste ponto observamos a importância de ferramentas que apoiem o diálogo do *designer* de *software* com estes materiais, sendo o trabalho de De Souza e colaboradores (De Souza, et al., 2016) um interessante exemplo deste tipo de abordagem. Sobre a ferramenta conceitual proposta neste trabalho, observamos que ela apoia este processo de reflexão, colaborando com a construção do raciocínio reflexivo do programador, e migrando-o da ideia de um racionalismo técnico para a ideia de reflexão em ação.

Além de sua função como ponto de apoio para a construção de um raciocínio reflexivo, existem outras contribuições a serem consideradas. Nosso público alvo, os programadores iniciantes, precisam lidar com todos os desafios existentes da aquisição do pensamento computacional em conjunto com o aprendizado de uma linguagem de programação e seus aspectos sintáticos e semânticos. Durante o estudo 5, um dos participantes lança uma breve crítica ao sistema de ensino de programação, que em muitos locais apresenta a lógica de programação já em associação com uma linguagem, não dando ao aluno o tempo necessário para que possa estruturar seu pensamento computacional para, somente após, lidar com os desafios impostos pela linguagem de programação. Neste ponto observamos a contribuição da nossa ferramenta como uma ponte para apoiar o aluno no processo de significação entre a lógica, a sintaxe e semântica do código-fonte.

E esta necessidade de reflexão é ainda mais importante no cenário atual no qual não apenas programadores iniciantes, que estão se preparando para trabalhar com desenvolvimento de *software* como profissão, mas usuários finais que atuam com programação, em algum nível, enfrentam as dificuldades que a pouca experiência com programação lhes causa (Burnett, et al., 2004) (Burnett, 2009) (De Souza, et al., 2011).

6 CONSIDERAÇÕES FINAIS

Ao nos apropriarmos das interfaces tradicionais, estamos não somente as adotando e as integrando ao nosso dia-a-dia, mas também as adaptando às nossas necessidades e práticas e, em diversas situações, as usando para outros propósitos, diferentes daqueles para os quais elas foram desenvolvidas (Stevens, et al., 2009) (Dourish, 2003).

Ao usarmos o termo interface, normalmente o associamos às tradicionais interfaces de *softwares*, usadas pelos assim chamados usuários finais, para, por meio delas, interagir com o sistema e alcançar os objetivos esperados. Sob o ponto de vista da nossa teoria norteadora, a Engenharia Semiótica, durante essa interação é entregue ao usuário uma mensagem, codificada nos signos presentes na interface*,* que expressa as intenções do *designer* em relação a aspectos sobre quem deve usar esta interface, e como e porque utilizá-la. Neste contexto, o programador é normalmente posicionado como o *designer*, sendo ele aquele que expressa a mensagem para o usuário.

No entanto, observamos que este programador, durante o processo de construção de um *software,* atua como usuário de diversas ferramentas, das quais estamos particularmente interessados nos códigos-fonte produzidos por outros programadores, que são (re)usados durante este desenvolvimento. Assim como uma interface tradicional, o código-fonte de um programa também carrega consigo a mensagem daquele que o escreveu, porém observamos que, , como uma medida de economia de tempo, o programador em alguns casos abdica-se de tentar compreender este código-fonte, apenas integrando-o ao seu próprio código-fonte e fazendo apenas as adaptações necessárias para adequá-lo ao novo contexto.

Além disso, o processo de reúso pode ser visto como um ciclo contínuo, pois frequentemente o código-fonte produzido com reúso de outros, será ele próprio reusado, em algum outro momento, por outro programador. Portanto, observamos que, ao fazer este reúso, sem a adequada compreensão, e reutilizá-lo sem a devida apropriação, são geradas possíveis rupturas na metacomunicação, uma vez que o próprio autor do código-fonte pode não compreender completamente a mensagem que está expressando.

Considerando as perspectivas e fatos apresentados, esta tese abordou reúso de código-fonte, por programadores iniciantes, sendo estes, programadores que estão ainda construindo seu

pensamento computacional (Wing, 2006), aprendendo a programar, seja com o objetivo de exercer essa profissão, ou com o objetivo de resolver seus próprios problemas do cotidiano.

Este documento apresentou o objetivo principal desta tese como ajudar programadores iniciantes a compreender o código-fonte que estão reusando para, desta forma, poderem fazer um uso mais adequado do mesmo. Nos baseamos na ideia que, ao reusar um código-fonte de um outro programador, só estamos realmente nos apropriando do mesmo se, após compreender este código-fonte em um alto nível e compreender os aspectos implícitos do discurso do programador que o desenvolveu, observarmos que este discurso expressa de certo modo aquilo que eu, programador, também gostaria de expressar em meu código-fonte.

No intuito de investigar este fenômeno, foram realizados 5 estudos, sendo três deles destinados a nos fornecer um melhor entendimento do problema, e dois deles destinados a verificar a aplicabilidade da ferramenta conceitual proposta nesta pesquisa.

Sobre os três primeiros, estes foram conduzidos junto a programadores iniciantes, observando aspectos de reúso presentes em seus códigos e buscando coletar informações sobre seus hábitos de reúso de códigos-fonte, como e porque este ocorria, além de observar como interpretavam seus próprios códigos-fonte, quando estes eram compostos por fragmentos de outros. Em relação aos dois últimos, estes foram conduzidos respectivamente junto a programadores iniciantes e professores de programação, visando compreender de que maneiras a ferramenta conceitual proposta poderia colaborar para um reúso de código-fonte consciente.

Como contribuições deste trabalho, destacamos as informações coletadas sobre as motivações dos programadores iniciantes para reusar códigos-fonte de outros programadores; as discussões sobre questões relacionadas a reúso de código-fonte e como e porque este deve ser feito somente se houver a compreensão sobre o discurso presente no mesmo e seus efeitos na apropriação deste código-fonte; e, uma ferramenta conceitual para apoiar a compreensão e a apropriação deste discurso, além de apoiar o programador na visualização de seu papel como usuário da interface representada pelo código-fonte.

Esperamos, com esta pesquisa, contribuir para a área de HCC, alertando estes programadores, ainda durante processo de construção de seu pensamento computacional, sobre a importância de compreenderem o que estão desenvolvendo, e que essa compreensão se relaciona não somente

aspectos cognitivos, mas também a aspectos comunicativos de seu código. Esperamos, também, contribuir para com a Engenharia Semiótica, apresentando o uso da teoria para apoiar a recepção de uma mensagem de metacomunicação. E, por fim, esperamos contribuir com o desenvolvimento do ensino e aprendizagem de programação, apresentando a programadores e professores uma perspectiva na qual programação pode ser vista como mais do que uma ferramenta para solução de problemas, mas também como uma ferramenta de comunicação e expressão.

6.1 Limitações

Em relação as limitações da pesquisa, destacamos que a ferramenta proposta se destina a apoiar a abstração de problemas simples, que podem ser estruturados nas questões abordadas pela mesma. A ferramenta irá, portanto, apoiar majoritariamente programadores iniciantes no processo de compreensão de um código-fonte para reúso, podendo, porém, ser usada por programadores de diversos níveis em problemas de baixa complexidade.

Outra limitação é que, durante nossos estudos sobre a aplicabilidade da ferramenta proposta, apenas coletamos opiniões de programadores iniciantes e de professores de programação, não tendo sido realizado um acompanhamento sobre o uso da ferramenta em um ambiente de ensino que nos possibilitasse a observação de seus efeitos em relação as contribuições no processo de aquisição de pensamento computacional.

Destacamos ainda, como limitação, a ausência de uma revisão da literatura, feita através de processo sistemático, que nos possibilitasse uma visão apurada do estado da arte em relação a reúso e apropriação de códigos-fonte.

6.2 Próximos passos e Trabalhos Futuros

Como próximos passos da pesquisa, observa-se a necessidade de serem trabalhados os pontos destacados como sendo nossas limitações, refinando a ferramenta conceitual de modo que possa apoiar tipos diversificados de problemas. Além disso, é preciso um acompanhamento do uso desta ferramenta como apoiadora no processo de aquisição de raciocínio computacional, observando como a reflexão incentivada pelo seu uso contínuo conscientiza o programador em relação à importância de compreender o que está expressando através de seu código-fonte.

Outro possível próximo passo seria seguir a sugestão dada por um dos professores entrevistados (P3) durante o estudo 5: "*Não sei o quanto tu já analisou os posts do Stack Overflow, aqueles mais bem cotados, mas é bom olhar, pois isso aqui que tu colocou, vai ver se aqueles caras que recebem o maior número de notas, que são os que mais contribuem para solucionar dúvidas, como é que eles conseguem explicar um código para ser utilizado em situações e problemas*". Neste sentido, sugerimos a inspeção das postagens melhor avaliadas do site *Stack Overflow*[30], analisando como estes programadores se posicionam ao tentar explicar a outros programadores seus códigos-fonte.

Observamos também que nossa ferramenta é focada em apoiar o programador iniciante durante o reúso de código-fonte, porém há a potencialidade de utilizar estar ferramenta para apoiá-lo, também, durante o processo de construção de um código-fonte, sem envolvimento de qualquer tipo de reúso. Acreditamos que a ferramenta, se aprofundada, pode apoiar o programador na visualização dos aspectos comunicativos presentes em seu código, sua importância, e como construir um código-fonte tendo em estes aspectos em mente.

Outra possibilidade de aprofundamento desta pesquisa seria a realização de estudos sobre as rupturas comunicativas observadas por programadores quando estes estão atuando como usuários de um código-fonte que foi escrito sem que seu programador (programador do código-fonte que está sendo reusado) levasse em consideração tais aspectos comunicativos. Ou seja, compreender mais amplamente quais as consequências seriam sofridas por terceiros ao usarem um código desenvolvido sem a devida apropriação.

Por fim, conforme observamos em nossos trabalhos relacionados, existem diversos ambientes que buscam recomendar códigos-fonte para serem reusados, porém, nenhuma das ferramentas encontradas fornece meios para apoiar o programador na apropriação destes códigos. Destacamos, então, a possibilidade de implementação de uma ferramenta de recomendação de códigos-fonte para reúso que se baseie nos aspectos metacomunicativos destes códigos, apoiando os

[30] https://pt.stackoverflow.com/

programadores a encontrarem códigos-fonte que tenham objetivos, a nível abstrato, condizentes com o que o programador almeja alcançar.

REFERÊNCIAS

ABASI, A. R.; AKBARI, N.; GRAVES, B. "Discourse appropriation, construction of identities, and the complex issue of plagiarism: ESL students writing in graduate school". *Journal of Second Language Writing*, vol. 15-2, 2006, pp. 102-117.

AFONSO, L. M. "Communicative Dimensions of Application Programming Interfaces (APIs)", Tese de Doutorado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2015, 152p.

ALNUSAIR, A.; ZHAO, T.; BODDEN, E. "Effective API navigation and reuse". In: Proceedings of 2010 IEEE International Conference on Information Reuse and Integration (IRI), 2010, pp. 7-12.

BANNON, L. "Reimagining HCI: Toward a more Human-Centered Perspective". *Interactions*, vol. 18-4, 2011, pp. 50-57.

BASTOS, J. A. D. M.; AFONSO, L. M.; DE SOUZA, C. S. "Metacommunication Between Programmers Through an Application Programming Interface: A semiotic analysis of date and time APIs". In: Proceedings of 2017 IEEE Symposium on Visual Languages and Human-Centric Computing, 2017, pp. 213-221.

BRANDT, J.; GUO, P. J.; LEWENSTEIN, J.; DONTCHEVA, M.; KLEMMER, S. R. "Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code". In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2009, pp. 1589-1598.

BURNETT, M. "What Is End-User Software Engineering and Why Does It Matter?" In: Proceedings of the 2nd International Symposium on End User Development, 2009, pp. 15-28.

BURNETT, M.; COOK, C.; ROTHERMEL, G. "End-user software engineering". *Communications of the ACM*, vol. 47-9, 2004, pp. 53-58.

CARROLL, J.; HOWARD, S.; PECK, J.; MURPHY, J. "A Field Study of Perceptions and Use of Mobile Telephones by 16 to 22 year Olds". *Journal os Information Technology Theory and Application*, vol. 4-2, 2002, pp. 49-62.

CHI, M. T. H.; BASSOK, M.; LEWIS, M. W.; REIMANN, P.; GLASER, R. "Self-explanations: How students study and use examples in learning to solve problems". *Cognitive Science*, vol. 13-2, 1989, pp. 145-182.

CHOI, S. "Understanding people with human activities and social interactions for human-centered computing". *Human-centric Computing and Information Sciences*, vol. 6-9, 2016, 10p.

COSTABILE, M. F; MUSSIO, P.; PROVENZA, L. P.; PICCINNO, A. "End users as unwitting software developers". In: Proceedings of the 4th International Workshop on End-User Software Engineering, 2008, pp. 6-10.

CRESWELL, J. W. "Research Design: qualitative, quantitative, and mixed methods approaches". SAGE Publications, Inc., 2014, 273p.

DE SOUZA, C. S. "The semiotic engineering of human-computer interaction". The MIT Press, 2005, 312p.

DE SOUZA, C. S. "Semiotic Engineering: A cohering theory to connect EUD with HCI, CMC and more". In: PATERNÒ, F.; WULF, V. (Eds). New Perspectives in End-User Development. Springer International Publishing, 2017, pp. 269-305.

DE SOUZA, C. S.; CERQUEIRA, R. F. G.; AFONSO, L. M.; BRANDÃO, R. R. M.; FERREIRA, J. S. J. "Software Developers as users: Semiotic investigations in Human-Centered Software Development". Springer International Publishing, 2016, 142p.

DE SOUZA, C. S.; GARCIA, A. C. B.; SLAVIERO, C.; PINTO, H.; REPENNING, A. "Semiotic traces of computational thinking acquisition". In: Proceedings of 3rd International Symposium on End-User Development, 2011, pp. 155-170.

DOURISH, P. "The Appropriation of Interactive Technologies: Some Lessons from Placeless Documents". *Computer Supported Cooperative Work*, vol. 12-4, 2003, pp. 465-490.

DYBA, T.; MEIDEN, N.; GLASS, R. "The Reflective Software Engineer: Reflective Practice". *IEEE Software*, vol. 31-4, 2014, pp. 32-36.

ECO, H. "A theory of semiotics". Indiana University Press, 1976, 354p.

ECO, U. "Obra Aberta". Perspectiva, 1968, 352p .

FERRANTE, C.; BORSEL, J. V.; MEDEIROS, M.; PEREIRA, B. "Análise dos processos fonológicos em crianças com desenvolvimento fonológico normal". *Revista da Sociedade Brasileira de Fonoaudiologia*, vol. 14-1, 2009, pp. 36-40.

FERREIRA, J. J.; DE SOUZA, C. S.; CERQUEIRA, R. "Why and how to investigate interaction design of software development tools". *SBC Journal on Interactive Systems*, vol. 6-1, 2015, pp. 48-65.

GASPAR, A.; LANGEVIN, S. "Restoring 'coding with intention' in introductory programming courses". In: Proceedings of 8th ACM SIGITE Conference on Information Technology Education, 2007, pp. 91-98.

GEORGAKOPOULOU, A. "Computer-mediated communication". In: ÖSTMAN, J.; VERSCHUEREN, J. Pragmatics in Practice. [S.l.]: John Benjamins Publishing, 2011, 326p.

GRIGOREANU, V.; BURNETT, M.; WIEDENBECK, S.; CAO, J.; RECTOR, K.; KWAN, I. "End-user debugging strategies: A sensemaking perspective." *ACM Transactions on Computer-Human Interaction*, vol. 19-1, 2015, 28p.

HOADLEY, C. M.; LINN, M. C.; MANN, L. M.; CLANCY, M. J. "When, why and how do novice programmers reuse code?" In: Proceedings of the Sixth Workshop on Empirical Studies of Programmers, 1996, pp. 109-130.

HOLMES, R.; ROBILLARD, M. P.; WALKER, R. J.; ZIMMERMANN, T. "Recommending source code examples via API call usages and documentation". In: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, 2010, pp. 21-25.

ICHINCO, M.; KELLEHER, C. "Exploring Novice Programming Example Use". In: Proceedings of 2015 IEEE Symposium on Visual Languages and Human-Centric Computing, 2015, pp. 63-71.

JAIMES, A.; GATICA-PEREZ, D.; SEBE, N.; HUANG, T. S. "Human-Centered Computing: Toward a Human Revolution". *Computer*, vol. 40-5, 2007, pp. 30-34.

KAMMERSGAARD, J. "Four different perspectives on human–computer interaction". *International Journal of Man-Machine Studies*, vol. 28-4, 1988, pp. 343-362.

KELLEHER, C.; PAUSCH, R. "Lowering the Barriers to Programming: A taxonomy of programming environments and languages for novice programmers". *ACM Computing Surveys*, vol. 37-2, 2005, 83-137.

KO, A. J.; ABRAHAM, R.; BEKWITH, L.; BLACKWELL, A.; BURNETT, M.; ERWIG, M.; SCAFFIDI, C.; LAWRANCE, J.; LIEBERMAN, H.; MYERS, B.; ROSSON, M. B.; ROTHERMEL, G.; SHAW, M.; WIEDENBECK, S. "The state of the art in End-User Software Engineering". *ACM Computing Surveys*, vol. 43-3, 2011, pp. 21-44.

KRUEGER, C. W. "Software Reuse". *ACM Computer Surveys*, vol. 24-2, 1992, pp. 131-183.

LOPES, A.; CONTE, T.; DE SOUZA, C. S. "Analisando a Comunicabilidade de Casos de Uso". In: Proceedings of the 16th Brazilian Symposium on Human Factors in Computing Systems, 2017, pp. 119-128.

MAALEJ, W.; TIARKS, R.; ROEHM, T.; KOSCHKE, R. "On the Comprehension of Program Comprehension". *ACM Transactions on Software Engineering and Methodology*, vol. 23-4, 2014, 38p.

MALAN, K.; HALLAND, K. "Examples that can do harm in learning programming". In: Proceedings of 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications, 2004, pp. 83-87.

MONTEIRO, I. T. "Autoexpressão e engenharia semiótica do usuário-designer", Tese de Doutorado, Departamento de Informática, PUC-Rio, Rio de Janeiro, 2015, 312p.

MONTEIRO, I. T.; DE SOUZA, C. S.; TOLMASQUIM, E. "My program, my world: Insights from 1st-person reflective programming in EUD education". In: Proceedings of the 5th International Symposium on End-User Development, 2015, pp. 79-91.

MORENO, L.; BAVOTA, G.; DI PENTA, M.; OLIVETO, R.; MARCUS, A. "How can I use this method?" In: Proceedings of the 37th International Conference on Software Engineering, 2015, pp. 880-890.

MYERS, B. A.; KO, A. J.; LATOZA, T. D.; YOON, Y. "Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools". *Computer*, vol. 49-7, 2016, pp. 44-52.

NEAL, L. R. "A System for Example-Based Programming". In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 1989, pp. 63-68.

PEIRCE, S. "Collected Papers of Charles Sanders Peirce". Harvard University Press, 1931-1958, vol. 1-8.

PESCIO, C. "Listen to Your Tools and Materials". *IEEE Software*, vol. 23-5, 2006, pp. 74-80.

ROBILLARD, M. P. "What Makes APIs Hard to Learn? Answers from Developers". *IEEE Software*, vol. 26-6, 2009, pp. 27-34.

ROEHM, T.; TIARKS, R.; KOSCHKE, R.; MAALEJ, W. "How Do Professional Developers Comprehend Software". In: Proceedings of the 34th International Conference on Software Engineering, 2012, pp. 255-265.

ROSSON, M. B.; BALLIN, J.; NASH, H. "Everyday Programming: Challenges and Opportunities for Informal Web Development". In: Proceedings of 2004 IEEE Symposium on Visual Languages and Human-Centric Computing, 2004, pp. 123-130.

SCHLEIMER, S.; WILKERSON, D. S.; AIKEN, A. "Winnowing: Local Algorithms for Document Fingerprinting". In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, pp. 76-85.

SCHÖN, D. "The Reflective Practitioner: How Professionals Think in Action". Basic Books, 1983, 374 p.

SEGAL, J. "Some Problems of Professional End User Developers". In: Proceedings of 2007 IEEE Symposium on Visual Languages and Human-Centric Computing, 2007, pp. 111-118.

SOJER, M. "Reusing Open Source Code". Springer Publisher, 2011, 288p.

STEVENS, G.; PIPEK, V.; WULF, V. "Appropriation Infrastructure: Supporting the Design of Usages". In: Proceedings of the 2nd International Symposium on End User Development, 2009, pp. 50-69.

WERTSCH, J. V. "Mind as Action". Oxford University Press, 1998, 224p.

WING, J. M. "Computational Thinking". *Communications of the ACM*, vol. 49-3, 2006, pp. 33-35.

ANEXO A

FORMULÁRIO DE ESTRUTURA DE METACOMUNICAÇÃO FORNECIDO PELA FERRAMENTA CONCEITUAL *SIGNIFYING MESSAGE* (De Souza, et al., 2016).

I. O que o desenvolvedor acredita sobre…

a. O perfil do usuário,

b. Os objetivos do usuário,

c. As necessidades do usuário,

d. As preferências do usuário,

e. A lógica de contexto do usuário?

II. O que o desenvolvedor pretende e espera a respeito de…

a. Descrição do sistema,

b. Funcionalidade do sistema,

c. Modo de uso do Sistema,

d. Lógica de design do sistema?

III. O que o desenvolvedor oferece e suporta para modos alternativos de uso que são compatíveis com o design do sistema?

APÊNDICE A

QUESTIONÁRIO DO ESTUDO 1

1. Uma determinada especificação de sistema é dada a você. Quais seus primeiros passos na construção da aplicação?
2. Você costuma usar exemplos?

   ( ) Sim

   ( ) Não

   ( ) Às vezes

   a. Se sim ou às vezes, em que situações os exemplos são usados

   ( ) Reúso de código – você utiliza o código do exemplo, para, sobre ele, modificar os itens necessários para chegar ao objetivo de seu programa.

   ( ) Modelo de código – você utiliza o código de exemplo, para, durante a construção de seu código, usar ele como consulta.

   Caso tenha marcado ambas ou use exemplos para outros fins, explique em quais situações os utiliza.

   b. Se sim, como você escolhe um exemplo para ser usado?

   ( ) Exemplos de mesmo domínio – códigos que tratem sobre o mesmo assunto/domínio do sistema que você está construindo.

   ( ) Exemplos de mesmo funcionamento – códigos que tratem sobre assuntos diferentes do sistema que você está construindo, porém tem funcionamento (cadastros, integrações e etc) semelhantes ao que você quer no seu sistema.

   Caso tenha marcado ambas ou use exemplos para outros fins, explique em quais situações os utiliza.

3. Pensando em seu programa como uma forma de comunicação, com quem você está (ou acredita estar) se comunicando?

4. Você percebe se características suas (de escrita/vocabulário/humor/personalidade) transcendem o *software*, chegando até àquele com quem você está se comunicando?

   ( ) Sim

   ( ) Não

   Se sim, dê um exemplo de quando tais características se apresentam (use seu código como fonte).

5. Imagine que você está tendo a oportunidade de falar com seu usuário. Você tem a oportunidade de apresentar para ele o sistema. Observe seu código, e com base nele, tente completar os itens abaixo.
   a. Este é o meu entendimento de quem você é...
   b. Isto é o que aprendi que você quer ou precisa fazer, de que maneiras prefere fazer, e por quê...
   c. Este é o sistema que projetei para você...
   d. Esta é a forma como você pode ou deve utilizá-lo para alcançar os objetivos...

APÊNDICE B

QUESTÕES DAS ENTREVISTAS DO ESTUDO 2

As entrevistas conduzidas no Estudo 2, foram norteadas pelas seguintes questões:

a. Uma determinada especificação de sistema é dada a você. Quais seus primeiros passos na construção da aplicação?
b. Você costuma usar exemplos? Se sim: Em que situações os exemplos são usados? Como você escolhe um exemplo para ser usado?
c. Pensando em seu programa como uma forma de comunicação; com quem você está se comunicando?
d. Você percebe se características suas (de escrita/vocabulário/humor/personalidade) transcendem o *software*, chegando até aqueles com quem está se comunicando? Dê exemplos.

APÊNDICE C

QUESTÕES DAS ENTREVISTAS DO ESTUDO 3

As entrevistas conduzidas no Estudo 3, foram norteadas pelas seguintes questões:

a. Após decidirem o domínio da aplicação e seu foco principal, quais foram os primeiros passos para iniciar o desenvolvimento da aplicação?
b. Vocês precisaram usar alguma ferramenta externa, como uma biblioteca, framework, ou API para desenvolver sua aplicação? Se sim, o que elas fazem?
c. Vocês já haviam usado este tipo de ferramenta?
d. Vocês podem explicar como estas ferramentas funcionam no seu programa?
e. Como vocês aprenderam a interagir com esta ferramenta?
f. Fora estes casos, houveram outras situações nas quais tiveram que usar um exemplo de código?

APÊNDICE D

QUESTIONÁRIO DE PERFIL DO ESTUDO 4

Técnicas e hábitos de programação

Este grupo de foco tem como objetivo investigar técnicas e hábitos de programação de programadores iniciantes. Este estudo está relacionado a uma pesquisa de doutorado do Programa de Pós-Graduação em Ciência da Computação da Faculdade de Informática da PUCRS.

Os dados aqui informados serão utilizados para fins de pesquisa e como base para futuras publicações e divulgações sobre o tema. Parte deste estudo será gravado em vídeo para posterior análise, vídeos estes que não serão divulgados, nem mesmo anonimamente.

Os dados coletados (respostas do formulários, falas, e observações do pesquisador) serão utilizados preservando o anonimato dos participantes em todo e qualquer documento divulgado em foros científicos ou pedagógicos.

* Required

Se estiver de acordo que utilizemos estes dados, conforme acima descrito, marque a opção abaixo e siga respondendo a pesquisa. Caso contrário, agradecemos seu interesse. *

☐ Concordo plenamente com os termos acima.

Informe seu nome completo: *

Your answer

Sobre você

Qual a sua idade? *

Your answer

Qual seu curso? *

- ○ Ciência da Computação
- ○ Engenharia da Computação
- ○ Sistemas de Informação
- ○ Engenharia de Software

Em qual semestre você está?

Your answer

Quando uma determinada especificação de sistema é dada a você, quais são seus primeiros passos na construção da aplicação? *

Your answer

Você costuma usar exemplos? Se sim, conte-me como você faz para (re)usá-los. *

Your answer

Em quais situações você percebe que geralmente precisa de um exemplo? *

Your answer

Você já usou um exemplo sem entender completamente como ele funciona? Comente. *

Your answer

Se pensássemos que através de seu programa você pode estar se comunicando com alguém, com quem você está (ou acredita estar) se comunicando? *

Your answer

Você já havia pensado nisso? Tendo como base seus últimos programas desenvolvidos, você diria que quem for usar seu programa vai saber facilmente o que ele faz? Por que? Se quiser, explique por meio de um exemplo de um programa que tenha desenvolvido? *

Your answer

Dentro deste tópico, existe algo mais que deseje nos falar? Comente.

Your answer

APÊNDICE E

EXEMPLOS USADOS NO ESTUDO 4

Sorteio de valores (Linguagem C)

```c
#include <stdio.h>

/*Implementa um programa para realizar o sorteio de X valores, dentro de
uma amostra N, sem que hajam números repetidos no sorteio*/
int main(){
    int i, qnt, sample;
    srand(time(NULL));

    printf("Informe a quantidade que deseja sortear: ");
    scanf("%d", &qnt);
    printf("Informe o tamanho da amostra: ");
    scanf("%d", &sample);
    int valores[qnt];
    int sel[sample+1];
    for(i=0; i<sample+1; i++)
    sel[i] = 0;
        for(i=0; i<qnt; i++){
            int sort=(rand()%sample)+1;
            if(sel[sort] != 1){
            valores[i]=sort;
            sel[sort] = 1;
            }
        }
    for(i=0; i<qnt; i++)
    printf("%d ", valores[i]);
}
```

Sorteio de valores (Linguagem Java)

```java
package exemplolista;

import java.util.Scanner;
/*Implementa um programa para realizar o sorteio de X valores, dentro de
uma amostra N, sem que hajam números repetidos no sorteio*/

public class ExemploSorteio {
    public static void main(String [] args){
        Scanner in = new Scanner(System.in);
        System.out.println("Informe a quantidade que deseja sortear: ");
        int qnt = in.nextInt();
        System.out.println("Informe o tamanho da amostra: ");
        int sample = in.nextInt();

        int[] valores = new int[qnt];
        int[] sel = new int[sample+1];

        for(int i=0; i<qnt; i++){
            int sort=((int)(Math.random()*sample))+1;
            if(sel[sort] != 1){
                valores[i]=sort;
                sel[sort] = 1;
            }
        }
        for(int i=0; i<qnt; i++) System.out.print(valores[i] + " ");
    }
}
```

Lista encadeada (Linguagem C)

```c
#include <stdio.h>
/*
    Implementa uma lista simplesmente encadeada para armazenamento de
    valores inteiros. Ao fim, apresenta os valores armazezados.
*/
struct Numero{
    int v;
    struct Numero *p;
};
typedef struct Numero numero;
int main(void){
    numero *l = (numero *) malloc(sizeof(numero));
    l->p = NULL;
    printf("Cadastrando numeros...\n\n");
    do{
        printf("Informe: ");
        int n;
        scanf("%d", &n);
        if(n<0)break;

        numero *novo = (numero *) malloc(sizeof(numero));
        novo->v = n;
        novo->p = NULL;

        if(l->p == NULL){
            l->p = novo;
        } else {
            numero *tmp = l->p;
            while(tmp->p != NULL)
                tmp = tmp->p;
            tmp->p = novo;
        }
    } while(1);
    printf("\nValores adicionados\n");
    numero *tmp = l->p;
    while(tmp != NULL){
        printf("%d ", tmp->v);
        tmp = tmp->p;
    }
}
```

Lista encadeada (Linguagem Java)

```
package exemplolista;
import java.util.Scanner;
/*
    Implementa uma lista simplesmente encadeada para armazenamento de
    valores inteiros. Ao fim, apresenta os valores armazezados.
*/
public class ExemploLista {
    public static class Numero{
        int v;
        Numero p;
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        Numero l = new Numero();
        l.p = null;

        System.out.println("Cadastrando numeros... \n");

        do{
            System.out.print("Informe um numero: ");
            int n = in.nextInt();

            if(n<0)break;

            Numero novo = new Numero();
            novo.v = n;
            novo.p = null;

            if(l.p == null){
                l.p = novo;
            } else {
                Numero tmp = l.p;
                while(tmp.p != null)
                    tmp = tmp.p;
                tmp.p = novo;
            }
        } while(true);

        System.out.println("\nValores adicionados\n");
        Numero tmp = l.p;
        while(tmp != null){
            System.out.println(tmp.v + " ");
            tmp = tmp.p;
        }
    }
}
```

Cifra de César (Linguagem C)

```c
#include <stdio.h>
#include <string.h>
/*
Implementa um código simples de criptografia, alterando as palavras
de um texto para N valores à frente, sendo N, um valor (chave)
definido pelo usuário.
*/
int main(){
    char alfanumeric[42] = " abcdefghijklmnopqrstuvxwyz1234567890.,!?\n";
    char text[500];
    printf("Informe um texto: ");
    fgets(text,499,stdin);
    printf("Informe a chave (use valores positivos para criptografar e negativos para descriptografar): ");
    int key;
    scanf("%d",&key);
    char newtext[500];
    int i;
    for(i=0;i<500 ;i++){
        if(text[i] == '\0')break;
        int newIndex = strchr(alfanumeric,text[i])-alfanumeric + key;
        if(newIndex >= 42){
        newIndex-=42;}
        else if (newIndex<0){
        newIndex+=42;}
        newtext[i] = alfanumeric[newIndex];
    }
    printf("\n\n%s", newtext);
}
```

Cifra de César (Linguagem Java)

```java
package exemplolista;
/*
Implementa um código simples de criptografia, alterando as palavras
de um texto para N valores à frente, sendo N, um valor (chave)
definido pelo usuário.
*/
import java.util.Scanner;
public class SimpleCript {
    public static void main(String [] args){
        Scanner in = new Scanner(System.in);
        String alfanumeric = "abcdefghijklmnopqrstuvxwyz1234567890.,!? ";
        System.out.println("Informe um texto: ");
        String text = in.nextLine().toLowerCase();
        System.out.println("Informe a chave (use valores positivos para criptografar e negativos para descriptografar): ");
        int key = in.nextInt();
        String newText = "";
        for(int i=0;i<text.length();i++){
            int newPos = alfanumeric.indexOf(text.charAt(i)) + key;
            if(newPos >= alfanumeric.length()){
            newPos -= alfanumeric.length();}
            else if(newPos < 0){
            newPos += alfanumeric.length();}
            newText+=alfanumeric.charAt(newPos);
        }
        System.out.println(newText);
    }
}
```

APÊNDICE F

QUESTÕES DAS ENTREVISTAS DO ESTUDO 4

A entrevista conduzida no Estudo 4, foi norteada pelas seguintes questões:

a. Vocês acham que essas informações (do template) lhes ajudam a mapear formas de reúso para este código? Quais?
b. Vocês identificaram coisas no exemplo que poderiam ser reescritas para tornar mais fácil compreender como ele funciona? O que?
c. Em uma situação normal de reúso, vocês replicariam esses mesmos problemas no código de vocês?
d. Acreditam que essas perguntas lhes ajudariam a ter uma melhor compreensão sobre o exemplo?
e. Vocês vêm vantagens em usar isso no seu dia-a-dia, quando forem usar um exemplo de código?
f. Vocês vêm desvantagens em usar isso no seu dia-a-dia? Porque você NÃO usaria?

APÊNDICE G

QUESTÕES DAS ENTREVISTAS DOS ESTUDOS 5

[Introdução]

- Quantos anos de experiência com ensino de programação?

[Uso de exemplos]

- Como você trabalha com exemplos em sala de aula?
    - Explica a sintaxe (comandos)?
    - Explica a semântica (blocos como um todo)?
    - Explica a pragmática (como posso aplicar isso em outras situações)?
    - Ensina a reusar (adaptação de um programa em outro)?
- Percebe se os alunos costumam se basear em exemplos para fazer os programas?
- Percebe se os alunos costumam reusar os exemplos sem entender “direito” o que estão fazendo?

[Explicação da abordagem]

- O que você acha desta abordagem (meu trabalho)?
- Quais as consequências possíveis desta reflexão?