Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Automação de Feedback para Melhorar a Nomeação de Identificadores de Alunos

Marcos Antônio Silva Nascimento

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Educação em Ciência da Computação

Ph.D. Jorge César Abrantes de Figueiredo (Co-orientador)
Ph.D. Eliane Cristina de Araújo (Orientadora)

Campina Grande, Paraíba, Brasil

N244a Nascimento, Marcos Antônio Silva.
Automação de feedback para melhorar a nomeação de identificadores de alunos / Marcos Antônio Silva Nascimento. – Campina Grande, 2019.
129 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2019.
"Orientação: Profa. Dra. Eliane Cristina de Araújo, Prof. Dr. Jorge César Abrantes de Figueiredo".
Referências.

1. Educação em Ciência da Computação. 2. Ensino de Programação. 3. Feedback Automático. I. Araújo, Eliane Cristina de. II. Figueiredo, Jorge César Abrantes de. III. Título.

CDU 004(07)(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

AUTOMAÇÃO DE FEEDBACK PARA MELHORAR A NOMEAÇÃO DE IDENTIFICADORES DE ALUNOS

MARCOS ANTÔNIO SILVA NASCIMENTO

DISSERTAÇÃO APROVADA EM 25/11/2019

ELIANE CRISTINA DE ARAÚJO, Dra., UFCG
Orientador(a)

JORGE CESAR ABRANTES DE FIGUEIREDO, Dr., UFCG
Orientador(a)

JOÃO ARTHUR BRUNET MONTEIRO, Dr., UFCG
Examinador(a)

ROBERTO ALMEIDA BITTENCOURT, Dr., UEFS
Examinador(a)

CAMPINA GRANDE - PB

Resumo

A legibilidade do programa é uma característica fundamental da qualidade de software. Portanto, fornecer feedback oportuno sobre a nomeação de identificadores pode ajudar a melhorar o treinamento de programadores novatos em educação de programação. De fato, vários estudos presentes na literatura da engenharia de software indicam que um código-fonte que contém nomes bem escolhidos de identificadores é mais fácil de entender e menos propenso ao erro em tarefas de manutenção.

Entretanto, devido aos inúmeros estudantes aprendendo a programar nos dias de hoje em cursos de programação, fornecer feedback manual sobre a qualidade dos nomes de identificadores torna-se proibitivo por causa da alta carga de trabalho imposta aos instrutores. Durante a pesquisa deste mestrado, nós propusemos um método inovador para gerar e fornecer feedback automático sobre a avaliação da adequação dos nomes de identificadores em contraste às verificações de convenções de nomenclatura. Nosso desafio foi fornecer a estudantes feedback elaborado e oportuno para ajudá-los a escolher nomes mais apropriados para denotar identificadores de seus códigos.

Nós implementamos e avaliamos nossa proposta em um curso de programação introdutória. Inicialmente, investigamos e verificamos que é possível encontrar automaticamente nomes considerados inapropriados de identificadores de código, com a intenção de usar esta informação para gerar o feedback automático. Em sequência, nós investigamos e testemunhamos que podemos gerar e fornecer o feedback automático para estudantes, de modo a ajudá-los na escolha de nomes de identificadores de seus códigos de melhor qualidade.

A principal contribuição desta pesquisa é que podemos fornecer o feedback automático para encorajar alunos a melhorar os nomes que eles escolhem para denotar identificadores de seus códigos. Com isso, poderemos ajudar estudantes a melhorar a nomeação dos identificadores e, em consequência, a legibilidade do seu programa, desde suas primeiras experiências de codificação.

Palavras-chave: Educação em Ciência da Computação; Ensino de Programação; Feedback Automático.

Abstract

Program readability is a fundamental tenet of software quality. Thus, providing timely feedback on identifier naming can help to improve novice programmer training in programming education. Indeed, several studies in software engineering literature indicate that a source code that contains well-chosen identifier names is easier to understand and less error-prone in maintenance tasks.

However, due to the numerous students are learning to program nowadays in programming courses, providing manual feedback on identifier name quality become prohibitive due to the highest workload imposed on instructors. During this master research, we proposed an innovative method to generate and provide automated feedback on identifier name appropriateness assessment in contrast to the checking of naming conventions. Our challenge was to provide to students timely and elaborated feedback to help them to choose more appropriate names to denote their code identifiers.

We implemented and evaluated our proposal in an introductory programming course. Firstly, we investigated and verified that it is possible to automatically find names considered to be inappropriate code identifiers, with the intent of using this information to generate automated feedback. In sequence, we investigated and witnessed that we can generate and give automated feedback to students so that we can help them to choose better quality code identifier names.

The main contribution of this research is that we can provide automated feedback to students in order to encourage them to improve names chosen by them to denote their code identifiers. So, we can help students to improve identifier naming and, in consequence, their program readability, since their early coding experiences.

Keywords: Computer Science Education; Programming Teaching; Automated Feedback.

Agradecimentos

Os agradecimentos desta dissertação de MSc. são inúmeros. Primeiro, agradeço à vida pela incrível oportunidade de conduzir esta pesquisa de mestrado. Esta oportunidade permitiu-me conhecer melhor em termos de engenheiro de software, pesquisador, e pessoa humana no mundo.

Segundo, agradeço a todos os meus orientadores, os diretamente e indiretamente responsáveis pela minha orientação, a professora Eliane Cristina de Araújo e os professores Dalton Dario Serey Guerrero e Jorge César Abrantes de Figueiredo. Os meus agradecimentos vão aos acompanhamentos frequentes e oportunos, os quais eram sempre seguidos de muita atenção, paciência, conversa e troca de conhecimentos.

Terceiro, agradeço a todos os amigos e companheiros da pós-graduação que sempre estiveram ao meu lado. Especialmente, agradeço aos colegas do Laboratório de Práticas de Software (SPLAB), Rafael Pontes, Raul Andrade, Jaziel Moreira e Guilherme Gadelha, as secretárias da pós-graduação, e Kaio Oliveira do Laboratório de Sistemas Distribuídos (LSD). Os meus mais sinceros agradecimentos vão aos conhecimentos compartilhados, a disponibilidade em conversar e ajudar fornecendo dicas e, além disso, apoio quando necessário.

Ainda, agradeço a todos os meus familiares mais próximos como meus pais, Geraldo do Nascimento e Rosimere Gonçalves da Silva Nascimento, os meus irmãos e minhas tias. Primeiro, por toda a paciência, incentivo, suporte incondicional e principalmente por acreditarem em mim.

Por fim, agradeço "à parcela da população brasileira que, sem saber nem poder, financiou minha formação", mantendo as palavras do orientador o Prof. Dalton Dario Serey Guerrero, a Coordenação de Aperfeiçoamento Pessoal (CAPES) e as demais entidades que patrocinaram e/ou auxiliaram a condução deste trabalho de alguma forma. Gratidão!

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Estamos preocupados com a forma como programadores novatos nomeiam os identificadores em seus programas, pois essas construções são usadas por eles para referenciar, por exemplo, variáveis e métodos. Há vários estudos na área de engenharia de software que indicam que a qualidade dos nomes de identificadores impacta diretamente na qualidade do software [BBL76, But09, BW10, BW08, BWYS09, LMFB06]. Há estudos que apontam que a pobre qualidade dos nomes de identificadores pode revelar, por exemplo, a presença de bugs de software [AAT+12].

Abebe e colegas (2012) estudaram a relação entre a qualidade dos nomes de identificadores e a presença de bugs. Eles investigaram o efeito de utilizar LBS (no inglês, *Lexicon Bad Smells*) para realizar a predição de falhas de software em três sistemas de código-fonte aberto (ArgoUML, Rhino, e Eclipse). Como resultado, encontraram que LBS é um forte indicador de bugs de software. Isto significa que a qualidade dos nomes de identificadores não afeta apenas a qualidade do vocabulário de código-fonte, mas também outras dimensões da qualidade de todo o software, como a funcionalidade.

Por este motivo, é importante escolher nomes apropriados para denotar identificadores, pois eles auxiliam o leitor na tarefa de entender como o software funciona, facilitando a sua leitura, entendimento e manutenção [Mar08]. Boehm e colegas (1976) referenciam a facilidade de leitura e entendimento de software como sendo "legibilidade" no modelo de qualidade de software deles. Adicionalmente, eles ainda reportam que a legibilidade é uma característica necessária para a manutenibilidade de software [BBL76], pois tarefas de manutenção de software podem se tornar desafiadoras e propensas ao erro caso o código seja

difícil de ler e entender. No estudo de Boehm e colegas (1976), define-se "manutenibilidade" como sendo a facilidade de realizar tarefas de manutenção de software.

Diante da importância de escrever programas com nomes apropriados de identificadores, defendemos que investir tempo no fornecimento de instrução referente às boas práticas de programação pode impactar positivamente na vida profissional de programadores inciantes. Neste sentido, o presente trabalho tem como principal objetivo orientar programadores novatos a melhorar a nomeação de identificadores, ajudando-os a escolher nomes apropriados para denotar identificadores dos seus programas.

Shute (2008) define feedback formativo como toda informação comunicada ao aluno que pretende modificar o seu pensamento ou comportamento, visando melhorar sua aprendizagem [Shu08]. Ela afirma que essa informação se origina em resposta a alguma ação realizada por parte do aluno. Levando em consideração essa definição, para atender o objetivo deste trabalho, nós propomos o fornecimento de feedback formativo sobre a avaliação da adequação dos nomes de identificadores que seja útil, padronizado e oportuno, visando a melhoria dos léxicos usados em construções de nomes de identificadores.

Desse modo, nossa proposta destina-se a oferecer orientação adequada e personalizada para as necessidades do aluno, que vai além de verificações baseadas em padrões da linguagem, ou convenções de nomenclatura, propostas em trabalhos relacionados. Pois, defendemos que podemos melhorar a legibilidade dos programas escritos pelos alunos através da escolha de termos significativos para denotar identificadores de seus códigos. Por exemplo, muitos trabalhos relacionados destinam-se a melhorar nomes de identificadores que não empregam o uso de separados de sublinhado ou de caso de camelo, sem levar em conta se eles referenciam conceitos do problema de programação. Com isso, poderemos ajudar o aluno a escolher nomes apropriados de identificadores através do fornecimento de feedback, esperando que ele revisite seu programa para renomeá-los, se necessário. Em consequência, poderemos melhorar a formação profissional do aluno, ajudando-o a melhorar a sua maneira de nomear identificadores desde suas primeiras experiências de codificação.

Neste trabalho, focamos no fornecimento do feedback para o programa que está funcionalmente correto, isto é, quando atende aos requisitos funcionais básicos determinados pelo texto da especificação do problema. Neste sentido, consideramos que um dado programa está funcionalmente correto quando ele passa em todos os casos de teste propostos pelo ins-

trutor que projetou a tarefa de programação. Focamos nesses programas pois, de acordo com a técnica de desenvolvimento dirigida por testes chamada TDD (no inglês, *Test Driven Development*) [Bec03], eles estão aptos para serem refatorados para melhoria de sua qualidade. Mapeando para a técnica de TDD, os programas que nos interessam são aqueles que apresentam "Barra verde".

1.1 Problema Geral e Solução Proposta

Tipicamente, programadores novatos treinam habilidades de programação, enquanto aprendem conceitos básicos, em cursos de programação introdutória realizando muitas tarefas de programação [A+17]. Considerando que cada aluno produz um programa para cada tarefa, fornecer o feedback manual sobre a avaliação da adequação dos nomes de identificadores sobre inúmeros programas pode se tornar uma prática tediosa, propensa ao erro e ineficiente em princípio.

Adicionalmente, pode se tornar uma tarefa proibitiva considerando a crescente demanda de alunos buscando por cursos de programação. Nos últimos anos, o número de alunos matriculados em cursos de programação tem crescido continuamente, levando a um aumento dos deveres dos professores além do tempo de professor-aluno [KJH16, Wil15, AM05, A+17]. Considerando essas questões, acreditamos que pode ser inviável para instrutores fornecerem acompanhamento um a um que seja oportuno, padronizado e personalizado. Em consequência, programadores novatos carecem de feedback sobre a avaliação da adequação dos nomes de identificadores de seus programas.

Diante desses problemas, propomos um método inovador para atender ao objetivo de fornecer feedback automático referente à avaliação da adequação dos nomes de identificadores. Para fornecer o feedback automático, realizamos três passos bem simples: a criação do vocabulário de referência, a avaliação da adequação dos nomes de identificadores e a geração das mensagens de feedback.

Inicialmente, realizamos a criação do vocabulário de referência. Este vocabulário é composto pelo conjunto de termos extraídos a partir de um material instrucional já produzido pelos instrutores para a tarefa de programação: o texto da especificação do problema. Em nosso método, usamos os termos desse vocabulário como a referência para encontrar nomes

apropriados de identificadores escritos no código do aluno. A lógica por trás do uso desses termos é simples: eles abrangem o jargão típico do domínio do problema ao qual o programa do aluno está relacionado. Neste sentido, quanto mais os nomes de identificadores do programa incluem esses termos, mais fácil de lê-lo, entendê-lo e mantê-lo.

Em sequência, realizamos a avaliação da adequação dos nomes de identificadores, ao passo que geramos as mensagens de feedback automático. Para avaliar a adequação dos nomes de identificadores, detectamos os identificadores extraídos a partir do código-fonte do estudante que não empregam os termos do vocabulário de referência em suas construções. Depois disto, geramos e fornecemos, para o estudante, mensagens de feedback apontando os identificadores considerados inapropriados. Essas mensagens dão conselhos sobre quais identificadores do seu programa poderiam ser renomeados, visando melhorar a legibilidade do software.

1.2 Visão Geral desta Dissertação

Durante esta pesquisa de mestrado, investigamos o potencial da geração e fornecimento de feedback automático sobre a avaliação da adequação dos nomes de identificadores para estudantes. Nosso desafio foi entregar mensagens de feedback elaborado [Shu08] e oportuno de forma a ajudá-los a refletir e melhorar sobre a qualidade dos nomes de identificadores de seu programa. Por este motivo, nossa intenção era propor um método inovador que fosse capaz de substituir a avaliação manual de instrutores, gerando e fornecendo feedback automático de identificadores semelhante à forma como eles o fazem manualmente.

Focamos em prover feedback automático sobre a avaliação da adequação dos nomes de identificadores de programas escritos por estudantes que estão funcionalmente corretos, visando ajudá-los a renomear seus nomes para a melhoria de sua qualidade. Com objetivo de investigar o potencial da geração e fornecimento de feedback automático, conduzimos estudos de pesquisa, como estudo exploratório, estudo de caso descritivo, experimento e *survey*, para reunir evidências para responder às seguintes questões de pesquisa (QP):

> QP1: É possível gerar feedback automático sobre avaliação da adequação dos nomes de identificadores baseado na descrição do problema de programação?

QP2: Estudantes podem melhorar a adequação dos nomes de identificadores de seus programas com ajuda do feedback automático e oportuno?

QP3: Estudantes acham que o feedback automático e oportuno é útil em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas?

Como ponto de partida, investigamos o potencial de usar descrições de problemas de programação para avaliar a adequação dos nomes de identificadores. Utilizamos uma descrição de problema de programação para encontrar nomes apropriados de identificadores de código em programas escritos por alunos. Em sequência, contrastamos os resultados obtidos com os da avaliação manual de instrutores. A partir daí, constatamos o potencial de usar descrições de problemas de programação para avaliar a adequação dos nomes de identificadores e fornecer a geração de feedback automático.

Depois disso, desenvolvemos uma ferramenta como prova de conceito para instanciar o método proposto e construir conhecimento sobre sua eficácia: *IQCheck* (do inglês, *Identifier Quality Check*). O desenvolvimento de *IQCheck* permitiu que nós conduzíssemos um experimento *in situ*, com estudantes de um curso de programação introdutória, para avaliar a eficácia de prover o feedback automático no código final deles. Sob configuração experimental, avaliamos a efetividade de fornecer mensagens de feedback automático com *IQCheck* para alunos de um grupo experimental, em contraste a alunos de um grupo de controle.

1.3 Resultados e Conclusões

Os resultados que alcançamos com o nosso estudo mostram que podemos usar a descrição de um problema de programação para encontrar nomes apropriados de identificadores de código em programas escritos por alunos. Descobrimos que os identificadores encontrados utilizando a descrição do problema de programação assemelha-se, até certo ponto, aos identificadores julgados como apropriados pela avaliação manual de instrutores. Isto significa que podemos usar os termos do texto da especificação do problema como a referência para avaliar a qualidade de identificadores de código e em consequência, gerar o feedback automático.

Além disso, descobrimos que podemos prover o feedback automático com *IQCheck* para

ajudar alunos a refletir e a melhorar a adequação dos nomes de identificadores de seus programas. Observamos que o feedback automático promove reflexão dos alunos do grupo experimental, ajudando-os a melhorar a qualidade dos seus programas. Nós testemunhamos que os estudantes que usaram *IQCheck* tenderam a melhorar a adequação dos identificadores de seu código, com 51,7% deles escolhendo nomes melhores depois do recebimento do feedback. Nós consideramos esses resultados como positivos embora existam espaços para ajustes e customizações que podem melhorar o método proposto.

1.4 Contribuições

Inicialmente, esperávamos observar alguma melhoria da qualidade dos nomes de identificadores de programas. Contudo, os resultados encontrados em nossos estudos vão além disso. Pois, observamos que o feedback automático não apenas pode ajudar estudantes a melhorar a nomeação de seus identificadores, mas também estimular a reflexão crítica sobre esse feedback. Para resumir, enumeramos abaixo as contribuições deste trabalho:

1. Um método para gerar e fornecer mensagens de feedback automático para ajudar estudantes a melhorar a nomeação de identificadores de seu código-fonte. Deste ponto em diante, vamos nos referir à Melhoria da Nomeação de Identificadores através da sigla MNI;

2. Uma ferramenta prova de conceito chamada *IQCheck* que foi desenvolvida para instanciar o método proposto e avaliar sua efetividade em fornecer ajuda para alunos, gerando mensagens de feedback automático. Atualmente, *IQCheck* está publicamente disponível na internet para uso e propósitos de pesquisa;

3. Este trabalho também apresenta um conjunto de lições apreendidas no decorrer do processo de automatizar o feedback para MNI, com o objetivo de fornecê-lo para alunos de um curso de programação introdutória.

1.5 Organização da Dissertação

Esta dissertação está organizada como segue:

Capítulo 2 - Background: Este capítulo apresenta a fundamentação teórica necessária para a condução desta pesquisa de mestrado. Ele apresenta o histórico das áreas de educação em ciência da computação e engenharia de software que nos motivou a prosseguir com essa pesquisa. Além disso, ele apresenta o arcabouço teórico relacionado aos conceitos que usamos para desenvolver este trabalho.

Capítulo 3 - A Solução Proposta: Este capítulo apresenta detalhes do método proposto para gerar o feedback de MNI. Ele apresenta uma visão geral de *IQCheck* e sua estratégia para fornecer o feedback automático para MNI.

Capítulo 4 - Metodologia: Este capítulo sumariza o roteiro desta pesquisa e discute a metodologia que nós adotamos para avaliar nossas afirmações e construir conhecimento sobre o método proposto para gerar o feedback de MNI.

Capítulo 5 - Sobre o Uso da Descrição do Problema de Programação: Este capítulo apresenta e demonstra em detalhes que é possível atender ao objetivo de gerar feedback automático para MNI. Ele apresenta a nossa investigação do potencial de usar termos obtidos da descrição de um problema de programação como a referência para encontrar nomes apropriados de identificadores de código.

Capítulo 6 - Sobre o Uso das Mensagens de Feedback Automático: Este capítulo apresenta e demonstra em detalhes que é possível atender ao objetivo de gerar mensagens de feedback com *IQCheck* para ajudar estudantes na MNI. Além disso, ele apresenta em detalhes a nossa investigação do potencial de gerar as mensagens de feedback de *IQCheck* feita com estudantes que usaram a nossa ferramenta. Estes alunos usaram *IQCheck* para obter feedback automático de qualidade sobre os programas produzidos por eles em suas atividades.

Capítulo 7 - Discussão: Este capítulo resume a discussão sobre o feedback para MNI e as ideias que emergem desta pesquisa de mestrado. Ele descreve as observações práticas de utilizar nossa proposta e as implicações pedagógicas provenientes disto. Além disso, ele descreve as limitações da solução proposta e algumas ameaças à validade de nossas conclusões.

Capítulo 8 - Conclusões: Este capítulo vai concisamente sintetizar este trabalho enfatizando o que foi feito e porque isso foi bom. Além disso, ele vai referenciar algumas oportunidades e trabalhos futuros que podem ser feitos para estender e melhorar esta pes-

quisa.

Capítulo 2

Background

Neste Capítulo, nós apresentamos a fundamentação teórica necessária para a condução desta pesquisa de mestrado visando tornar esta dissertação auto-contida. Ele apresenta o histórico das áreas de educação em ciência da computação e engenharia de software que nos motivou a prosseguir com essa pesquisa. Além disso, ele apresenta o arcabouço teórico relacionado aos conceitos que usamos para desenvolver este trabalho. Inicialmente, apresentamos na seção 2.1 algumas considerações sobre a nomeação de identificadores que levamos em consideração ao oferecer a nossa proposta. Em sequência, apresentamos na seção 2.2 uma visão geral de estudos presentes nas áreas de educação em ciência da computação e engenharia de software relacionados à nossa proposta. Nesta última seção, apresentamos sistemas e métodos existentes para avaliação da qualidade dos nomes de identificadores de código que estão relacionados ao nosso trabalho.

2.1 Considerações Sobre a Nomeação de Identificadores

É importante avaliar e fornecer feedback sobre a nomeação de identificadores, pois a escolha e o emprego de nomes pouco significativos no código-fonte pode prejudicar a qualidade do software. A escolha e o emprego de bons nomes, por outro lado, pode contribui positivamente para a qualidade do software, pois é largamente entendido que eles podem tornar tarefas humanas voltadas para a compreensão e manutenção mais fáceis e menos propensas a erros [DLDPO11].

Caprile e Tonella (2000) estudaram a importância de identificadores para o entendimento,

eles apontam que seus nomes estão entre as fontes mais importantes de informação para entender os componentes do código-fonte [CT00]. Martin (2008) faz, em seu livro intitulado como Código Limpo, a seguinte recomendação sobre nomes para a escrita de um código limpo, fácil de ser entendido e modificado:

> "Names in software are 90 percent of what make software readable. You need to take the time to choose them wisely and keep them relevant. Names are too important to treat carelessly." [Mar08]

A despeito disso, desenvolvedores tendem a escolher e empregar nomes de identificadores pouco significativos. Sneed (1996) estudou a maneira como desenvolvedores escolhem nomes de identificadores em sistemas legados. Curiosamente, ele observou que desenvolvedores tendem a nomear funções/procedimentos e estruturas de dados de maneira arbitrária [Sne96]. O que significa que, eles não relacionam nomes de identificadores ao conteúdo semântico de funções/procedimentos e estruturas de dados.

Haiduc e Marcus (2008) estudaram projetos de software de código aberto para investigar o uso de termos do domínio do sistema no código-fonte. Eles descobriram que apenas cerca de 40% dos termos do domínio do sistema eram empregados como nomes de identificadores no código-fonte [HM08]. Isso significa que, apesar do código-fonte ser desenvolvido para atender ao propósito de um domínio específico, desenvolvedores aparentemente não empregam o jargão típico relacionado ao contexto do problema.

Os resultados dos estudos acima constituem uma motivação importante para o nosso trabalho, compartilhando considerações sobre a necessidade de investir tempo em atividades voltadas para MNI (Melhoria da Nomeação de Identificadores). Por este motivo, nossa proposta busca fornecer feedback oportuno e personalizado a alunos em cursos de programação, ajudando-lhes a fazer a escolha de nomes significativos para denotar identificadores.

Para fornecer instrução adequada e personalizada para alunos, procuramos a base teórica sobre estilo de programação para que pudéssemos orientar nosso trabalho, estudando boas práticas de programação referenciando a nomeação de identificadores. Como nossa proposta destina-se a ajudar o aluno a escrever um código que contém nomes que facilitam sua leitura, entendimento e manutenção, buscamos por boas práticas de programação propostas por Martin (2008). Nós vamos nos abster de discutir mais detalhadamente todas as recomendações

propostas por ele aqui, já que não é nossa intenção fornecer uma pesquisa abrangente sobre todas as boas práticas para escrita de código limpo nesta seção. Em resumo, a boa prática de programação referenciando a nomeação de identificadores proposta por ele que adotamos é: escolha de nomes descritivos. Martin apresentou e discutiu esta prática da seguinte forma:

> "Don't be too quick to choose a name. Make sure the name is descriptive. Remember that meanings tend to drift as software evolves, so frequently reevaluate the appropriateness of the names you choose. This is not just a "feel-good" recommendation." [Mar08]

A ideia por trás da escolha de nomes descritivos para denotar identificadores é bastante simples: sempre que eles são cuidadosamente escolhidos, eles "sobrecarregam a estrutura do código com descrição" [Mar08]. O que ajuda o leitor a entender em uma maneira fácil: o contexto ao qual o código está inserido, e o problema ao qual ele se propõe a resolver, descrevendo a estratégia da solução adotada. Considerando essa boa prática de programação, nossa proposta adere, quando propomos e implementamos nossa ferramenta prova de conceito - *IQCheck*, à avaliação da adequação dos nomes de identificadores escritos por alunos.

Além disso, nossa proposta também baseia-se em uma conjectura discutida em 2007 por Lucia e colegas, e Poshyvanyk e Marcus que: a similaridade entre artefatos de alto nível e código-fonte é um indicador da boa qualidade de comentários e identificadores do código-fonte [LFOT07, PM07]. A nomeação de identificadores baseada em especificação de software, conforme proposto por Lucia e colegas, e Poshyvanyk e Marcus, também é um padrão de codificação recomendado em estudos da área de engenharia de software para gerar avaliação automática de qualidade de nomes de identificadores [ACC$^+$02, DLDPO11, LFB06, ES15]. Neste sentido, *IQCheck* gera e fornece feedback oportuno sobre a adequação de identificadores, avaliando se seus nomes estão baseados em um conjunto de termos específicos referenciando o jargão do domínio do problema. Isto é parte da nossa proposta e será detalhado no Capítulo 3 desta dissertação.

2.2 Trabalhos Relacionados

Nesta seção, apresentamos uma visão geral de estudos presentes nas áreas de educação em ciência da computação e engenharia de software relacionados à nossa proposta. Nós apresen-

tamos sistemas e métodos existentes para avaliação da qualidade de nomes de identificadores de código que estão relacionados ao nosso trabalho. Na subseção 2.2.1, nós apresentamos uma visão geral breve de ambiantes de aprendizagem baseados em computadores que estão em linha com nossa proposta nesta pesquisa. Nós discutimos sobre Sistemas de Avaliação Automáticos - SAA no contexto de educação de programação que provém para alunos feedback no processo de aprendizagem. Esses trabalhos estão relacionados ao nosso no sentido de que a nossa estratégia para prover feedback precisou estar associada com um SAA para condução dos estudos desta pesquisa. Na subseção 2.2.2, sumarizamos algumas estratégias existentes para avaliar a qualidade dos nomes de identificadores de código. O mapeamento e a discussão dessas estratégias foram importantes para nosso trabalho, no sentido que elas revelaram a oportunidade de conduzir esta pesquisa.

2.2.1 Sistemas Automáticos de Avaliação

SAAs são sistemas baseados em programas de computador que têm sido desenvolvidos desde 1960 [DLO05], sendo capazes de gerar feedback automático oportuno para dar suporte à correção de programas escritos por alunos no processo de aprendizagem. Alguns exemplos de SAAs são: *Web-CAT* [Edw03], *Mooshak* [RSKPFVI14], *MarmoSet* [SHP+06], *PASS* [TR97], entre outros.

SAAs têm sido largamente utilizados, em cursos de programação introdutória envolvendo um grande número de alunos matriculados, para gerar e fornecer feedback formativo sobre avaliação de programas escritos por alunos ao responderem tarefas de programação. Além de substituir a avaliação humana realizada manualmente por instrutores, SAAs viabilizam o fornecimento de feedback consistente, objetivo, e oportuno para programas escritos por alunos em respota a muitas tarefas de programação [AM05, DLO05, IAKS10].

Em geral, SAAs implementam funcionalidades similares [IAKS10]. Tipicamente, a funcionalidade mais comum desses sistemas é a geração e o fornecimento de feedback sobre avaliação da corretude funcional do código. Uma estratégia simples para gerar este tipo de feedback inclui: executar no código do aluno um conjunto predefinido de casos de teste, fornecido pelo instrutor, e apresentar o resultado da comparação entre a saída esperada e a observada a partir do código.

Para conduzir um dos estudos desta pesquisa, a nossa estratégia para gerar e fornecer

feedback precisou estar associada com um SAA desenvolvido *in-house* para o curso de programação introdutória da nossa universidade - UFCG. TST é um conjunto de ferramentas da linha de comando que realiza o teste e a verificação automática da corretude funcional de programas escritos, pelos alunos do nosso curso, em resposta a problemas de programação. TST permite verificar as saídas observadas de um dado código, executando um conjunto predefinido de casos de testes baseados em entrada/saída ou testes baseados em scripts. Por padrão, TST pode ser usado com linguagens de programação como Python, Javascript (node) e Java. No entanto, é fácil configurá-lo para suportar outras linguagens de programação também.

Nós associamos nossa proposta com TST para facilitar a condução de um estudo empírico visando testá-la com alunos. Como nossa ferramenta prova de conceito - *IQCheck* era completamente independente de TST, a intenção era tornar a sua forma de uso o mais semelhante possível da forma de uso de TST, o SAA ao qual os alunos do nosso curso estavam habituados a utilizar. Isso iria dar ao aluno a oportunidade de dedicar um menor tempo para aprender como fazer uso de *IQCheck* sem recrutar esforço cognitivo excessivo dele. Nossa principal motivação era usar *IQCheck* para fornecer instruções em forma de mensagens de feedback que seguissem um estilo de conversa, visando induzir o aluno a criar um senso de parceria com nossa ferramenta [May08]. Como consequência, o aluno poderia se esforçar para entender o que realmente estava sendo aconselhado pela mensagem de feedback.

2.2.2 Avaliação Automática de Qualidade de Identificadores

SAAs podem ajudar alunos dando-lhes dicas sobre a corretude funcional do seu programa. No entanto, afirmamos que alunos carecem de dicas sobre a qualidade dos nomes de identificadores de seu código. Há estudos presentes nas áreas de educação em ciência da computação e engenharia de software relacionados ao nosso trabalho que revelam algumas estratégias existentes para avaliar a qualidade de nomes de identificadores de código.

Na área de engenharia de software, Lawrie e colegas (2006) usaram a similaridade textual entre artefatos de software relacionados para avaliar a qualidade do software [LFB06]. Neste estudo, eles propuseram uma ferramenta que usa processamento de linguagem natural para gerar avaliação de qualidade, instanciando a abordagem proposta por eles. Essa ferramenta foi capaz de caracterizar a qualidade de um programa através da verificação da similaridade textual entre os termos obtidos a partir de identificadores e comentários do seu código.

Lucia e colegas (2011) relataram que desenvolvedores poderiam ser solicitados a melhorar significativamente a qualidade dos nomes de identificadores, se o ambiente de desenvolvimento fornecesse informações sobre a verificação da similaridade textual entre o vocabulário do código-fonte e da documentação do software relacionado [DLDPO11]. Nós não encontramos estudos na área de educação em ciência da computação que usam essa verificação para avaliar a qualidade dos nomes de identificadores. Entretanto, citamos alguns estudos relacionados ao nosso trabalho, utilizando outros tipos de checagem nos contextos de detecção de plágio [CCK11] e avaliação de aspectos estruturais do código do estudante [NGV10, TR97, ASF16].

Há muitos estudos na área de educação em ciência da computação que avaliam ferramentas para gerar feedback sobre o estilo de programação. Esses estudos estão relacionados ao nosso trabalho, no sentido de que nossa proposta destina-se a avaliar um dos critérios do estilo de programação dos programas de alunos: a qualidade do nome do identificador. Por exemplo, ASSYST [JU97], CheckStyle [Che01], PMD [PMD01], STYLE [Ree82], Style ++ [AMUJ04] e entre outras, são ferramentas baseadas em verificações de convenções de nomenclatura que podem avaliar a qualidade do nome do identificador. Essas ferramentas são semelhantes entre si, no sentido de que elas usam o número de caracteres, ou o tamanho do nome, como o único indicador de que o identificador escolhido pelo aluno é descritivo. Basicamente, elas julgam um identificador como "bom" quando o tamanho de seu nome é maior ou igual a um tamanho mínimo predefinido. Nossa proposta difere dessas ferramentas em sua metodologia, no sentido de que é baseada em palavras obtidas do texto de especificação do problema como a referência para julgar identificadores como apropriados em vez de usar apenas o tamanho do nome deles.

Araújo e colegas (2017) relatam que instrutores podem fornecer feedback automatizado sobre a avaliação da qualidade do código do aluno, comparando-o com um código de referência proposto pelo instrutor [A$^+$17]. A lógica por trás disso é simples: o código de referência pode conter a linha de base de qualidade esperada do código do aluno. A metodologia da nossa proposta assemelha-se à metodologia da proposta deles, no sentido de gerar e fornecer feedback automatizado sobre a avaliação da adequação de identificadores, comparando seus nomes com os termos do texto da tarefa de programação: o vocabulário de referência.

O trabalho de Glassman e colegas (2015) é o mais próximo ao nosso em relação à in-

tenção de fornecer feedback sobre a avaliação da qualidade dos nomes de identificadores, mas sua abordagem requer anotações humanas fornecidas por um instrutor através de uma interface de usuário [GFSM15]. A interface deles permite que um instrutor envie feedback semiautomático personalizado sobre a qualidade dos identificadores escolhidos pelo aluno com base nos valores que essas variáveis podem assumir durante a execução do programa. Nosso trabalho é semelhante ao seu estudo mas difere em sua metodologia, no sentido de que nosso trabalho visa gerar feedback automatizado sobre a avaliação da qualidade de identificadores, mas é baseado na verificação entre os nomes deles e os termos do vocabulário de referência.

Capítulo 3

A Solução Proposta

Nesta dissertação, defendemos que fornecer feedback formativo sobre avaliação da qualidade dos nomes de identificadores para estudantes novatos pode ajudar a melhorar o treinamento e a aprendizagem de escrita/desenvolvimento de software com qualidade. Durante a pesquisa deste mestrado, observamos que fornecer este tipo de feedback para esses alunos é benéfico, pois pode ajudá-los a refletir e melhorar a qualidade da escrita dos nomes de identificadores de seus programas. Isto significa que este tipo de feedback é útil e adequado para ajudar a escrever nomes apropriados de identificadores, auxiliando na escrita de programas mais legíveis.

Neste Capítulo, apresentamos uma visão geral de *IQCheck*, a ferramenta prova de conceito desenvolvida para instanciar a ideia geral da nossa proposta de gerar e fornecer feedback automático sobre a adequação dos nomes de identificadores. Afirmamos que *IQCheck* é capaz de usar a descrição de um problema de programação para avaliar a adequação dos nomes de identificadores e gerar mensagens de feedback automático. Além disso, afirmamos que o feedback gerado por *IQCheck* é capaz de ajudar na formação de alunos, pois ele ajuda a escolher nomes mais apropriados para denotar identificadores de programa.

3.1 *IQCheck*

Na Figura 3.1, apresentamos uma visão geral de *IQCheck*. Como o primeiro passo da sua utilização para uma tarefa de programação, precisamos criar o vocabulário de referência para a descrição dessa tarefa. Na Figura 3.1, criamos o vocabulário de referência realizando o

processo de normalização textual da especificação da tarefa problema, o qual inclui a extração de palavras, a remoção de *stop-words* e a transformação das palavras resultantes. Na subseção 3.1.1, apresentamos mais detalhes de como *IQCheck* normaliza o texto e produz esse vocabulário. Sempre que o aluno solicita as mensagens de feedback de *IQCheck*, ele gera um bloco de mensagens de aviso referenciando nomes de identificadores em seu código que poderiam ser renomeados. Para tanto, *IQCheck* usa a verificação de uso de termos do vocabulário de referência em construções de nomes de identificadores no seu código. Na Figura 3.1, geramos as mensagens de aviso após realizar a verificação de uso de termos do vocabulário de referência, a qual requer a extração de nomes, a separação e remoção de *stop-words* entre nomes compostos e a transformação de nomes. Na subseção 3.1.2, explicamos mais detalhes de como instrutores e estudantes podem usar *IQCheck*.

Figura 3.1: A Solução Proposta - *IQCheck*

3.1.1 Normalização Textual

Normalização de texto é o processo de transformar um texto original, usando tokenização, segmentação ou uma outra técnica, para encontrar os termos canônicos que ele contém. Este processo é comumente usado em pesquisa de recuperação de informação e já tem sido utilizado em trabalhos na área de engenharia de software para avaliar a qualidade de identificadores [ACC+02, LFB06, DLDPO11].

Similarmente a esses trabalhos, *IQCheck* normaliza o texto da especificação da tarefa problema para a criação dos termos do vocabulário de referência. A ideia é criar termos que estão relacionados ao jargão do domínio do problema com caracteres apropriados para denotar nomes de identificadores, pois o texto pode conter léxicos sem conteúdo relevante ou com letras acentuadas que não são aplicáveis à nomeação de identificadores.

Durante a normalização textual, *IQCheck* usa módulos de NLTK (do inglês, *Natural Language ToolKit*) [NLT17] para criar os termos, em português, do vocabulário de referência. NLTK é o kit de ferramentas em Python focado no processamento de texto escrito usando Linguagem Natural. Inicialmente, *IQCheck* usa o modulo Tokenize [Tok01] para aplicar a tokenização do texto e extrair léxicos que denotam palavras. Em sequência, *IQCheck* usa um método *built-in* de Python e o módulo Unidecode [Uni90] para converter as palavras com letras maiúsculas e acentuadas em outras com letras minúsculas e não acentuadas, respectivamente.

Depois disso, *IQCheck* usa a lista de *stop-words* comuns em português de NLTK para aplicar a detecção e remoção de *stop-words*, que são palavras de alta frequência com pouco conteúdo léxico (e.g., preposições, artigos, números, e sinais de pontuação). *IQCheck* encontra e remove essas palavras pois elas não têm conteúdo relevante ou descritivo para serem usadas como nomes de identificadores dentro do código-fonte do aluno, no sentido de comunicar ao leitor conceitos do problema de programação. Além de *stop-words*, *IQCheck* usa o módulo RegEx [RE90] para aplicar a detecção e remoção de caracteres especiais, palavras formadas por letras repetidas, seguidas por números e tendo um tamanho menor ou igual a dois (e.g., "`\n`", "$", "%", "programm", "n1", ".py"). *IQCheck* encontra e remove palavras tendo este tamanho de acordo com o *threshold* estabelecido por Yates e Neto [BYRN99] para detecção de *stop-words*.

Finalmente, *IQCheck* usa o módulo RSLP [RSL01] para aplicar a stemização, isto é, a extração de radicais de palavras, convertendo plurais em singulares, transformando formas verbais conjugadas em infinitivos, e removendo sufixos de adjetivos. *IQCheck* aplica a stemização a essas palavras pois elas podem incluir plurais, verbos conjugados e adjetivos, que são palavras flexionadas em seu tronco, base ou raiz. Por exemplo, "contador" e "contagem" são duas palavras flexionadas do mesmo tronco. Portanto, elas iriam ser transformadas no tronco único "cont" sem os seus sufixos "ador" e "agem".

3.1.2 Geração e Fornecimento das Mensagens de Feedback Automático

Construímos *IQCheck* com o propósito de avaliar a validade da nossa proposta com alunos do nosso curso de programação introdutória em um estudo empírico: um experimento. Por este motivo, construímos *IQCheck* para avaliar a adequação dos nomes de identificadores de códigos escritos em Português, a língua nativa de alunos e instrutores do curso de programação introdutória.

Como nossa intenção inicial era permitir o funcionamento de *IQCheck* como um plugin de TST, utilizamos o código-fonte da uma ferramenta previamente desenvolvida para estar associada ao TST que está publicamente disponível no Github[1] como um *framework* para desenvolver *IQCheck*. O código-fonte de *IQCheck* também está publicamente disponível no GitHub[2] sob a licença AGPL-3.0. Portanto, ele pode ser acessado, estudado, modificado e utilizado para propósitos de pesquisa. Como *IQCheck* tem seu próprio processo de instalação, ela deve ser instalada em cada ambiente do usuário. A partir de um repositório público no GiThub, é possível fazer download e instalação de *IQCheck* em um ambiente Linux para funcionamento em conjunto com TST. O uso de *IQCheck* é bastante simples. Em resumo, esta ferramenta é invocada através de um comando padrão de TST, que é usado pelo aluno para receber mensagens de feedback sobre a adequação dos nomes de identificadores do código implementado por ele em resposta à descrição de um dado problema de programação.

Os instrutores podem fazer uso de *IQCheck* quando escrevem um novo texto de especificação de problema, executando um comando próprio para criar o vocabulário de referência. Após ser invocado, *IQCheck* cria os termos desse vocabulário aplicando o processo de normalização do texto da especificação da tarefa problema e os salva um arquivo chamado *iqcheck.json*. Nós nos referimos a estes termos como "vocabulário de referência", pois *IQCheck* os utiliza como a referência para encontrar nomes de identificadores considerados inadequados para este problema de programação. *IQCheck* automaticamente detecta o arquivo contendo o texto.

Após isso, já é possível gerar e fornecer as mensagens de feedback automático sobre a qualidade dos nomes de identificadores. Encorajamos o aluno a solicitar as mensagens de feedback de *IQCheck* quando ele tiver escrito um programa em resposta à descrição do

[1] https://github.com/elianearaujo/tst-qcheck
[2] https://github.com/marcosasn/tst-iqcheck

problema de programação. Além disso, é esperado que o aluno faça uso do *IQCheck* solicitando mensagens de feedback automatizadas quando se certificar de que seu programa está sintaticamente correto.

Sempre que o aluno solicita as mensagens de feedback de *IQCheck*, ele usa um módulo Python chamado AST (do inglês, *Abstract Syntax Trees*) [AST90] para analisar o código do aluno e um algoritmo implementado para extrair léxicos que denotam identificadores escritos pelo aluno como nomes de variáveis e métodos. Em sequência, *IQCheck* remove nomes duplicados para aplicar a detecção, divisão, remoção de *stop-words* de nomes compostos de dois ou mais termos, usados para construir o identificador, separados pelos separadores de sublinhado ou de caso de camelo. Depois disso, *IQCheck* aplica a stemização aos termos separados resultantes, ou nomes, para obter troncos dos termos flexionados ou em inglês, pois muitos alunos do curso podem usar esse idioma além do português. Ao aplicar a stemização ao termo em inglês, *IQCheck* o transforma em um tronco que pode estar relacionado a qualquer termo do vocabulário de referência, já que muitas palavras do idioma inglês e português são construídas com base em uma mesma raiz comum, o latim. Por exemplo, após a stemização, o nome de identificador "contador_a" iria ser transformado em dois outros nomes "a" e "cont", este último sem o seu sufixo "ador".

Depois disso, *IQCheck* encontra todos os nomes de identificadores considerados apropriados/positivos/bons quando suas construções incluem a presença de pelo menos um dos termos salvos em *iqcheck.json* ou inapropriados/negativos/ruins quando de outra forma. Em seguida, *IQCheck* gera um bloco de mensagens de aviso na interface da linha de comandos dando conselhos referentes aos identificadores inapropriados/negativos/ruins, informando que eles poderiam ser renomeados escolhendo nomes mais apropriados. Essas mensagens fornecem dicas sobre quais construções de identificadores do código do estudante poderiam ser renomeadas, no sentido de melhorar a comunicação de conceitos do problema de programação, a legibilidade do programa e sua compreensão em uma maneira *bottom-up* [Pen87b, Pen87a].

3.1.3 Demonstração de Uso

Para demonstrar o uso de *IQCheck* mostramos a especificação do problema da tarefa de programação "Maior Torcida" na Tabela 3.1. Como um primeiro passo, iremos informar a

IQCheck para criar o vocabulário de referência através do processo de normalização textual deste texto. Em sequência, iremos requisitar as mensagens de feedback de *IQCheck* sobre a adequação dos nomes de identificadores dentro do programa mostrado na Figura 3.2.

Tabela 3.1: Especificação do Problema da Tarefa de Programação Maior Torcida

Na maior cidade do interior do nordeste, a disputa pela maior torcida do estado é acirrada. Para resolver essa peleja, é preciso que você implemente um programa que leia a quantidade de torcedores do primeiro time nos 5 setores do estádio e a quantidade de torcedores do segundo time nos mesmos 5 setores do estádio. Você deve somar a quantidade de torcedores de cada time e imprimir o time que mais leva torcida ao estádio. **Entrada:** Seu programa deve ler 10 números da entrada padrão. Os primeiros 5 números lidos representam a quantidade de torcedores do primeiro time em cada um dos cinco setores do estádio. Os últimos 5 números lidos representam a quantidade de torcedores do segundo time nos setores. **Saída:** Seu programa deve imprimir uma mensagem identificando o time com mais torcedores. Caso haja empate, apenas imprima "Empate.".

Inicialmente, iremos informar a *IQCheck* para criar o vocabulário de referência. Na interface da linha de comando abaixo, apresentamos o comando necessário para este propósito.

```
$ tst iqcheck -s
```

Listing 3.1: Chamada de *IQCheck* para Criar o Vocabulário de Referência

Em sequência, iremos solicitar as mensagens de feedback de *IQCheck* sobre a adequação dos nomes de identificadores dentro do programa mostrado na Figura 3.2. Para gerar e fornecer as mensagens de feedback automático sobre os nomes dos identificadores deste programa, *IQCheck* irá utilizar os termos do vocabulário de referência salvos no arquivo *iqcheck.json*. É conveniente avaliar a adequação dos nomes de identificadores desse programa, pois ele representa a primeira versão correta implementada por um dado estudante em resposta ao texto mostrado na Tabela 3.1.

```
1  #coding: utf-8
2  a = 0
3  b = 0
4  for i in range(5):
5      ent = int(raw_input())
6      a += ent
7  for k in range(5):
8      ent = int(raw_input())
9      b += ent
10 if a > b:
11     print "O primeiro time leva mais torcedores ao estádio."
12 elif b > a:
13     print "O segundo time leva mais torcedores ao estádio."
14 else:
15     print "Empate."
```

Listing 3.2: Primeiro Programa Correto

Quando solicitamos as mensagens de feedback de *IQCheck*, ele gera na interface da linha de comando o bloco de mensagens de aviso mostrado na Figura 3.3 com relação aos identificadores nas linhas 2, 3, 4, 5, 7 e 8. Além desses identificadores, *IQCheck* poderia ter marcado como inapropriado outros como "cont_torcida1", "time1", "digito", "num1" e "setor5". *IQCheck* poderia ter apresentado mensagens de aviso sobre a adequação desses identificadores como, nestes casos, seus nomes não incluem nenhum dos termos do vocabulário de referência salvos no arquivo *iqcheck.json*. Diante disto, esses nomes não parecem ser adequados para serem utilizados como identificadores, pois eles não comunicam conceitos do problema de programação apresentados no texto mostrado na Tabela 3.1. *IQCheck* não demora muito para gerar e fornecer na interface da linha de comando o bloco de mensagens de aviso. Examinamos o seu desempenho medindo a performance de nossa ferramenta e descobrimos que ela leva menos que 1 minuto (1,076 segundos) para fornecer o feedback.

```
$ tst iqcheck code.py
# code.py

**5 Advertência(s)**

### Identificadores
- *a* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *i* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *k* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *b* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *ent* não parece ser um nome adequado. Use palavras da descrição da tar
efa de programação.
```

Listing 3.3: Primeira Chamada de *IQCheck*

Após isso, renomeamos o antigo nome do identificador da linha 2 de "a" para "torcida_a", pois este último parece ser mais apropriado que o primeiro levando em consideração o texto mostrado na Tabela 3.1. Quando solicitamos as mensagens de feedback de *IQCheck* pela segunda vez, ele gera na interface da linha de comando um outro bloco de mensagens de aviso mostrado na Figura 3.4 com relação aos identificadores nas linhas 3, 4, 5, 7 e 8. Em resposta ao texto mostrado na Tabela 3.1, poderíamos ter escolhido outros nomes para construir os identificadores deste código, baseados em palavras como "torcida", "numero", "time", "torcedor" e "estadio".

```
$ tst iqcheck code.py
# code.py

**4 Advertência(s)**

### Identificadores
- *i* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *k* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *b* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *ent* não parece ser um nome adequado. Use palavras da descrição da tar
efa de programação.
```

Listing 3.4: Segunda Chamada de *IQCheck* após Renomear um Nome de Identificador

Capítulo 4

Metodologia

Neste Capítulo, sumarizamos o roteiro desta pesquisa e discutimos a metodologia que adotamos para prover o método para gerar e fornecer mensagens de feedback automático de MNI (Melhoria da Nomeação de Identificadores). Inicialmente, descrevemos e sumarizamos o roteiro dos nossos estudos. Em sequência, apresentamos/fornecemos uma visão geral do contexto em que esta pesquisa estava inserida e as estratégias que utilizamos para coletar e filtrar os dados utilizados em nossas análises. Além disso, expomos brevemente ferramentas estatísticas empregadas nos estudos. Por fim, discutimos os métodos que seguimos para avaliar nossas afirmações e construir conhecimento sobre o método proposto.

4.1 Roteiro da Pesquisa

O último objetivo desta pesquisa era orientar programadores novatos a melhorar a nomeação de identificadores, ajudando-os a escolher bons nomes para denotar identificadores dos programas escritos por eles. Como ajuda, propusemos a geração de dicas em forma de mensagens de feedback através do método proposto. Como esse tipo de feedback destina-se a melhorar identificadores previamente nomeados, ele referencia a adequação dos nomes sobre o programa proposto por um aluno em resposta a um problema de programação especificado no texto da questão.

Ao aplicar o método proposto em programas escritos por alunos, é natural relembrar tarefas de verificação e validação da área de engenharia de software. No contexto de engenharia de software, tarefas de verificação e validação são comumente realizadas por equipes

de qualidade visando verificar a qualidade do software (i.e., se ele atende ou não aos requisitos funcionais e não-funcionais esperados pelo cliente). Relembrando, requisitos funcionais incluem a especificação de funcionalidades que o software deve oferecer para o cliente, isto é, incluem a especificação do que ele fará (e.g., cadastro de produto, busca de produto, envio de e-mail para o cliente). Requisitos não-funcionais, por outro lado, incluem a especificação de características desejáveis do software, promovidas durante o oferecimento de suas funcionalidades, ou seja, incluem a especificação de como ele fará (e.g., eficiência, legibilidade, manutenibilidade). Tanto os requisitos funcionais quanto os não-funcionais precisam ser implementados pelos desenvolvedores do software, pois eles irão permitir que o software atenda aos propósitos do cliente.

Mapeando para nosso estudo, podemos associar tarefas de verificação, referentes à nomeação de identificadores, realizadas por equipes de qualidade às avaliações realizadas por instrutores de um curso de programação. Tipicamente, essas equipes estão preocupadas em verificar e assegurar a boa qualidade da nomeação de identificadores, pois bons nomes de identificadores podem contribuir positivamente para a legibilidade e manutenibilidade do software. Por este motivo, muitos instrutores acham importante investir em estratégias para promover o ensino de boas práticas de programação, avaliando e fornecendo feedback sobre a adequação dos nomes escolhidos pelo aluno para denotar identificadores de seu programa. Neste sentido, nossa proposta buscou ser capaz de avaliar a adequação dos nomes de identificadores de maneira mais próxima possível da forma como instrutores o fazem manualmente.

Inicialmente, conduzimos um estudo exploratório com o objetivo de examinar como instrutores realizam avaliações humanas sobre a adequação dos nomes de identificadores de códigos de alunos. Neste estudo, avaliamos a viabilidade de automaticamente encontrar nomes apropriados de identificadores em programas enviados em resposta a uma descrição de problema de programação. Em seguida, dirigimos um estudo de caso descritivo com o objetivo de investigar e demonstrar o potencial de usar essa mesma descrição para encontrar os nomes apropriados de identificadores, segundo a avaliação humana de qualidade do primeiro estudo.

Depois disso, conduzimos um experimento *in situ* visando investigar e demonstrar que podemos ajudar alunos a refletir e melhorar a adequação dos nomes de identificadores dos seus programas com *IQCheck*. Por fim, conduzimos uma outra avaliação humana após o

experimento através de um *survey* para coletar e reunir dados sobre a experiência de uso de alunos que tinham usado *IQCheck*. Neste estudo, diagnosticamos a utilidade das mensagens de feedback de *IQCheck* em fornecer ajuda no processo de MNI.

4.2 Métodos

Nesta seção, descrevemos os procedimentos gerais que aplicamos aos estudos empíricos conduzidos ao longo desta pesquisa. Apesar de estarem em diferentes categorias: estudo exploratório, estudo de caso descritivo, experimento e *survey*, eles compartilham algumas características que são agrupadas nesta seção. Adicionalmente, algumas particularidade metodológicas de cada estudo são detalhadas mais profundamente nos seus capítulos respectivos.

4.2.1 Contexto

Esta pesquisa de mestrado aconteceu na UFCG no curso de graduação de Ciência da Computação. Ao longo desta pesquisa, coletamos e analisamos programas produzidos por estudantes durante a interação deles com TST no curso de Laboratório de Programação 1 - LP1. LP1 é o primeiro curso de Ciência da Computação, então podemos considerá-lo equivalente ao curso CS1 (do inglês, *Computer Science 1*).

Este trabalho aconteceu sob supervisão de meus orientadores. Eles faziam parte do corpo docente do curso de Ciência da Computação e tinham experiência de ensino em LP1. O pessoal acadêmico de LP1 foi composto de quatro instrutores graduados e estudantes não-graduados que forneciam suporte como assistentes dos professores ou tutores dos alunos. Dentre os instrutores graduados, dois deles eram meus orientadores nesta pesquisa.

Atualmente, LP1 engloba algumas particularidades que vale a pena serem mencionadas dado que elas também faz parte do contexto do estudo. LP1 tem adotado as estratégias de ensino *flipped classroom* e *mastery learning* [LRBC+17] com avaliação contínua e individualizada. *Flipped Classroom* ou simplesmente sala de aula invertida, é um método pedagógico instrucional que visa permitir a aprendizagem ativa de alunos. Esta estratégia busca empregar: aulas em vídeo assíncronas, tarefas de casa envolvendo problemas práticos, e atividades em sala de aula sobre resolução de problemas com ajuda de instrutores. *Mastery learning* é

uma outra estratégia de ensino que pode ser usada em conjunto com a estratégia de *flipped classroom*. *Mastery learning* permite que instrutores acompanhem alunos com necessidades de aprendizagem específicas fornecendo-lhes instrução adequada. Esta estratégia de ensino advoga que o aluno precisa alcançar um nível de domínio em um conhecimento pré-requisito antes de seguir adiante para aprender um novo conhecimento.

Para aplicar esta estratégia de ensino, LP1 é dividido em 10 unidades temáticas e permite que alunos coexistam no mesmo curso estando em diferentes unidades de acordo com o ritmo de aprendizagem deles. Por isso, cada uma das unidades temáticas de LP1 têm objetivos de aprendizagem específicos que os alunos tem que dominar. Semanalmente, instrutores aplicam exames de avaliação individualizada. Esses exames são compostos de um conjunto de tarefas de programação, de várias unidades, similares às feitas durante as aulas de laboratório e em casa. Este tipo de exame aplicado semanalmente permite que os instrutores avaliem continuamente se um dado aluno domina ou não o conhecimento necessário para que ele siga para a unidade subsequente.

Para cada unidade temática do curso, há um conjunto de tarefas de programação. *Flipped classroom* permite que alunos possam estudar o conteúdo de LP1 e realizar tarefas envolvendo problemas práticos em casa usando o sistema TST. Além disso, também permite que alunos possam realizar atividades em sala de aula sobre resolução de problemas usando TST com ajuda de instrutores. *Mastery learning* permite que estudantes destravem o seu acesso a uma unidade subsequente quando eles têm resolvido corretamente 65% do número total de tarefas da unidade corrente deles. Além disso, permite que os instrutores possam fornecer instrução personalizada de acordo com o conhecimento que os alunos têm dominado no curso.

LP1 estimula a produção de código Python através de muitas tarefas de programação. Naturalmente, esta é a linguagem de programação adotada pelos instrutores de LP1 para ensinar conceitos introdutórios de programação. Automaticamente, TST atribui aleatoriamente tarefas de programação para alunos matriculados no curso de acordo com a unidade corrente deles. O sistema permite que um aluno acesse tarefas de programação, faça check-out delas, submeta programas para elas, e receba feedback sobre a corretude funcional dos programas submetidos. Desde 2017, esta pesquisa foi inscrita no comitê de ética em pesquisa envol-

vendo humanos sob o número CAAE 70198817.3.0000.5182 [1].

4.2.2 Coleta de Dados

Em geral, os estudos empíricos conduzidos ao longo desta pesquisa de mestrado foram realizados com programas coletados no curso de LP1. Estes programas foram produzidos pelos alunos matriculados em LP1 durante a interação entre eles e o material instrucional através de TST. Além desses programas, também coletamos *logs*, que foram produzidos por *IQCheck* sempre que algum aluno requisitou as suas mensagens de feedback durante a condução do experimento.

Coletamos e usamos em nossos estudos textos de especificação de problema tipicamente utilizados como tarefas de programação pelos instrutores no curso. Esses textos foram especialmente úteis para nos permitir usar *IQCheck* para gerar arquivos *iqcheck.json* contendo os vocabulários de referência. Esses mesmos arquivos foram usados por *IQCheck* para gerar e fornecer as mensagens de feedback de MNI.

Adicionalmente, também usamos em nossos estudos dados referentes a opiniões produzidas por humanos. Em geral, coletamos e usamos dados de opiniões produzidos por humanos em um *survey* com alunos e outro com instrutores. Usamos dados de opiniões produzidos pelos quatro instrutores graduados de LP1 e de minha orientadora. Esses dados foram fornecidos por eles quando participaram de um *survey* visando registrar a opinião deles sobre a adequação dos nomes de identificadores de programas escritos por alunos. Além disso, também usamos dados de opiniões produzidos por alunos que usaram *IQCheck* e consumiram suas mensagens de feedback.

Além da coleta, realizamos a filtragem de dados. O processo de filtragem foi feito de diferentes formas de acordo com o objetivo do estudo empírico. Em geral, nós usamos informação gerada por TST para identificar os programas que poderíamos utilizar em nossos estudos. Nós usamos TST para separar programas funcionalmente corretos de incorretos como ele automaticamente testa a corretude funcional de programas.

[1] Mais informações sobre o estudo podem ser encontradas em http://plataformabrasil.saude.gov.br/

4.2.3 Análises

As métricas que utilizamos nos estudos conduzidos ao longo desta pesquisa de mestrado geraram dados quantitativos. Nós analisamos esses dados para obter resultados usando medidas estatísticas de resumo e estimação. Nós adotamos a recomendação de Gardner e Altman (1986) sobre o uso de estimação ao invés de testes de hipóteses visando computar intervalos de confiança em contraste ao estudo de p-valores [GA86].

Em regra, usamos métodos estatísticos não paramétricos para fazer estimação. A motivação por trás disso é que não tínhamos a intenção de fazer qualquer presunção sobre a normalidade dos dados que estávamos estudando. Além disso, para nossos dados, nem sempre era válida a premissa necessária para utilizar métodos estatísticos paramétricos, pois algumas vezes eles não mostravam seguir o modelo de uma distribuição normal. Para fazer estimação, utilizamos técnicas de *bootstrap* não paramétrico ordinário para computar valores de métricas e medidas estatísticas de resumo. A partir desses valores computados, pudemos computar intervalos de confiança não paramétricos. Para utilizar técnicas de *bootstrap* não paramétrico, tínhamos apenas que, quando possível, nos certificar que a amostra observada era representativa da sua população [Efr87].

Em resumo, a técnica de *bootstrap* (no inglês algo como "alça de botina"), ou *bootstrapping*, permite computar valores de uma dada métrica ou medida estatística. A partir dos valores computados, é possível estimar a variância deles e obter o intervalo de confiança que contém o valor da medida observada na população dos dados. Isto é possível através da geração de um modelo populacional que é construído a partir de uma amostra de observações e reamostragens, de modo a imitar como as observações foram obtidas. A estimação de valores de métricas e medidas estatísticas de resumo através de intervalos de confiança nos permitiram apresentar resultados diretamente na escala de medida dos dados. A partir desses resultados, nós pudemos verificar a validade de nossas hipóteses [GA86].

Para computar valores de métricas e medidas estatísticas usando intervalos de confiança, nós priorizamos o uso do método BCA (do inglês, *Bias Corrected and Accelerated*) [Dix01]. Nós computamos valores de métricas e medidas estatísticas em cada amostra de reamostragem com 2000 reamostramentos [Efr87] para obter intervalos de confiança por este método.

Em seu estudo, Efron (1987) afirma que é ideal utilizar pelo menos 1500 reamostragens utilizando a técnica de *bootstrap* para computar este tipo de intervalo de confiança em um

nível de 95% de confiança. Em contraste a outros métodos para gerar intervalos de confiança usando o método de *bootstrap*, isto equivale a 50% a mais do que o número inicialmente proposto (1000 reamostramentos).

4.2.4 Conclusões dos Estudos

Nós próximos capítulos iremos apresentar estudos e resultados de nossa pesquisa sobre geração de feedback formativo visando MNI. Cada capítulo refere-se a uma ampla questão de pesquisa e detalha uma sequência de estudos empíricos para investigá-la. Nós decidimos seguir uma ordem lógica dos estudos conduzidos nesta pesquisa de mestrado com o intuito de enfatizar as afirmações que concluímos como resultado do esforço da pesquisa.

As principais contribuições desta pesquisa estão listadas a seguir na forma de afirmações ou *claims*. Levamos em consideração achados presentes em estudos das áreas de educação em ciência da computação e engenharia de software com o intuito de construir conhecimento sobre elas.

- C1 - Nós geramos feedback automático sobre avaliação da adequação dos nomes de identificadores baseado na descrição do problema de programação;
- C2 - Estudantes melhoram a adequação dos nomes de identificadores de seus programas com ajuda do feedback automático e oportuno;
- C3 - Estudantes acham que o feedback automático e oportuno é útil em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas.

No Apêndice A, apresentamos os principais estudos conduzidos para reunir evidências que dão suporte a C1 e C2 em um artigo, o qual foi submetido para publicação no SBIE (Simpósio Brasileiro de Informática na Educação): "Giving Automated Feedback About Student Code Identifiers: a Method Based on the Description of Programming Problem".

Capítulo 5

Sobre o Uso da Descrição do Problema de Programação

Neste Capítulo, vamos investigar e demonstrar que é possível atender ao objetivo de gerar de forma automática feedback sobre a adequação dos nomes de identificadores usando a descrição do problema de programação. Inicialmente, apresentamos nosso primeiro estudo, no qual convidamos instrutores de um curso de programação introdutória para fornecerem avaliações humanas sobre a adequação dos nomes de identificadores de códigos de alunos. Em sequência, apresentamos o segundo estudo em que contrastamos os identificadores julgados, como apropriados e inapropriados, da avaliação humana com os identificadores julgados como apropriados, segundo a técnica proposta.

O ponto de partida desta pesquisa é a geração da avaliação automática de qualidade que irá compor mensagens de feedback automático. Para examinar a viabilidade da geração da avaliação automática de qualidade, conduzimos nosso primeiro estudo: uma investigação exploratória, em que examinamos como instrutores realizam avaliações humanas sobre a adequação dos nomes de identificadores de códigos de alunos. No sentido de criar o *baseline* utilizado para validar o método que propomos, convidamos instrutores, de um curso de programação introdutória, para avaliarem a adequação dos nomes de identificadores tendo em vista a legibilidade. Assim, utilizamos o julgamento dos nomes apropriados e inapropriados como o *baseline* para determinar a forma como instrutores avaliam a qualidade dos identificadores de código. Deste ponto em diante, iremos nos referir às avaliações realizadas por instrutores em cursos de programação como "avaliação humana". Nós propusemos a

seguinte questão de pesquisa:

> QP1: Como a avaliação humana julga a adequação dos nomes de identificadores de código, tendo em vista a legibilidade?

Para responder a esta questão de pesquisa, examinamos nosso *baseline* para constatar quais as características que a avaliação humana leva em consideração ao julgar identificadores de código como apropriados, contrastando-os com os inapropriados. A partir dos resultados, fundamentamos a conjectura/inspiração desta pesquisa:

> "Nomes apropriados para denotar identificadores de código são, em grande parte, construídos usando palavras da descrição do problema de programação."

Com esta conjectura em mente, conduzimos a segunda investigação empírica deste Capítulo: um estudo de caso descritivo. Neste estudo, verificamos e confirmamos a validade da nossa conjectura de que a maioria dos nomes apropriados de identificadores são constituídos de termos da descrição do problema de programação. A partir dos resultados, chegamos à conclusão que podemos gerar mensagens de feedback automático sobre a adequação dos nomes de identificadores considerados inapropriados, segundo o texto da descrição do problema. Deste ponto em diante, vamos nos referir ao método aqui proposto, de encontrar nomes apropriados de identificadores de código no programa de alunos, através da verificação da presença de termos do vocabulário de referência, como "avaliação de qualidade automática". A questão de pesquisa que dirigiu o estudo de caso descritivo foi:

> QP2: A avaliação de qualidade automática, em certa medida, se assemelha à avaliação humana ao julgar nomes apropriados de identificadores de código?

Neste estudo, avaliamos o quanto o julgamento de nomes apropriados de identificadores da avaliação automática, extraídos de programas funcionalmente corretos, assemelha-se ao julgamento da avaliação humana. Para tanto, contrastamos os julgamentos de ambos métodos de avaliação e examinamos o quanto eles eram próximos.

5.1 Contexto

Os estudos deste Capítulo aconteceram de forma organizada e controlada na UFCG, no curso de graduação em Ciência da Computação, no contexto do curso de LP1. Os estudos apresentados nas seções 5.2 e 5.3 aconteceram durante os semestres acadêmicos de 2017.2 e 2018.1, respectivamente.

Em ambos estudos, usamos dados reunidos e coletados usando TST em uma edição do curso anterior - 2017.1. O conjunto de dados usado nos estudos foi composto de 125 nomes de identificadores extraídos a partir de 58 programas funcionalmente corretos. Estes programas foram submetidos por diferentes estudantes como soluções para uma tarefa de programação dirigida por um texto de especificação do problema. Para conduzir este estudo, convidamos todos os instrutores de LP1, incluindo meus orientadores. Todos eles já haviam ensinado em turmas do curso de LP1 e, portanto, possuíam experiência em fornecer avaliação humana sobre programas escritos por alunos desse curso.

5.2 Avaliação Humana de Qualidade

Nossa investigação exploratória visando examinar como a avaliação humana julga a adequação dos nomes de identificadores de código, teve como contribuição a fundamentação da seguinte conjectura:

> "Nomes apropriados para denotar identificadores de código são, em grande parte, construídos usando palavras da descrição do problema de programação."

5.2.1 Métodos

Sem formular nenhuma hipótese prévia, conduzimos uma investigação exploratória sobre o *baseline* do estudo para examinar como a avaliação humana julga a adequação dos nomes de identificadores de código. A questão de pesquisa que motivou o presente estudo foi:

> QP1: Como a avaliação humana julga a adequação dos nomes de identificadores de código, tendo em vista a legibilidade?

Com esta questão de pesquisa em mente, inspecionamos o *baseline* do estudo procurando por características em nomes de identificadores de código desejáveis pela avaliação humana ao julgá-los como apropriados. Para obter tais características, comparamos nomes de identificadores julgados como apropriados em contraste aos inapropriados. Com o objetivo de constatar possíveis diferenças entre os primeiros e os últimos, propomos um conjunto de métricas as quais utilizamos como indicadores da qualidade das construções dos nomes de identificadores. Neste sentido, essas métricas geram dados quantitativos referentes ao que é considerado apropriado e inapropriado nas construções dos nomes de identificadores julgadas pela avaliação humana. Por fim, realizamos uma análise comparativa baseada nos valores calculados das métricas, com o intuito de contrastar os identificadores em termos de sua qualidade. Isso nos permitiu entender melhor o que construções apropriadas de identificadores contêm que diferem das inapropriadas, fornecendo insights sobre a aptidão de usar a avaliação de qualidade automática.

Estes são os passos para a criação do *baseline* do estudo: Primeiramente, convidamos cinco instrutores experientes, do mesmo curso de programação, para avaliar a adequação dos nomes de identificadores de código. Para coletar a avaliação humana, cada um deles foi convidado a responder um questionário de *survey*. Neste *survey* foi apresentada a descrição do problema de programação e uma questão perguntando a opinião do instrutor, para cada identificador, se "(este) contribui para a legibilidade do programa". Para capturar a opinião dos instrutores, nosso questionário apresentou um conjunto predefinido de respostas possíveis, as quais foram posteriormente mapeadas para um valor da escala de Likert [Lik32] variando de 1-5: "discordo fortemente (1); discordo (2); neutro (3); concordo (4);" e "concordo fortemente (5)". No design deste estudo, o valor correspondente à opinião dos instrutores foi atribuído à variável independente *score*. Em média, foi de 13 minutos, variando de 10-80 minutos, o tempo mediano gasto pelos instrutores para avaliar a adequação dos nomes de identificadores de código. Em sequência, usamos o *alpha* de Cronbach para medir o grau de consistência interna dos valores de *scores* atribuídos pelos instrutores, com o intuito de avaliar o nível de concordância inter-instrutor da avaliação humana. Baseados nos valores de *score*, calculamos o valor de *alpha* (*alpha* de Cronbach de 0,854). O valor obtido foi maior do que 0,8, o que significa um alto grau de consistência interna de acordo com a regra geral para interpretar valores desta métrica. A partir disso, chegamos à conclusão que a avaliação

humana tem um alto nível de concordância e que a qualidade dos identificadores foi avaliada de forma similar entre os instrutores [LVNT15]. Depois disso, usamos a mediana, que é uma medida estatística local de resumo para criar um valor médio de *score* sem distorcê-lo por nenhum valor de *score* extremamente grande ou pequeno, como cada identificador foi avaliado por cinco instrutores.

Por fim, com o propósito de julgar nomes de identificadores, classificamos todos aqueles que apresentaram um valor médio de *score* maior do que o valor neutro como "verdadeiro" e todos outros valores como "falso". Neste estudo, este primeiro e último correspondem aos nomes apropriados e inapropriados de identificadores, respectivamente. Neste sentido, utilizamos esta classificação como uma variável dependente, usando-a como um fator. A partir desta classificação, criamos o *baseline* do estudo, o qual utilizamos como um oráculo da qualidade dos nomes de identificadores baseado na avaliação humana.

5.2.2 Métricas

Abebe e colegas (2012) propuseram que os léxicos usados na construção do identificador pode ajudar a entender o software [AAT+12]. Por esse motivo, propomos três métricas, baseadas na construção do identificador, para gerar dados quantitativos visando caracterizar e diferenciar nomes “verdadeiros” de “falsos”. Estas são as métricas propostas: o número de caracteres (*nc*), o número de termos (*nt*) e o número de termos descritivos (*ntd*).

A primeira, *nc*, basicamente conta o número de caracteres que fazem parte da construção do nome do identificador, não importando se o caractere corresponde a uma vogal, consoante ou um outro símbolo. Esta métrica tem sido utilizada em trabalhos relacionados conforme discutimos no Capítulo 2, com o intuito de fornecer feedback sobre o estilo de programação em termos da qualidade de identificadores. Por esse motivo, também levamos em consideração essa medida para diferenciar identificadores "verdadeiros" de "falsos", pois acreditamos que ela possa indicar que este primeiro grupo é mais descritivo do que este último em alguns casos. A segunda e terceira, *nt* e *ntd*, são relativamente similares na intenção de contar sequências de caracteres que denotam palavras ou termos. No entanto, elas diferem no sentido de que *ntd* vai mais além, pois conta o número de termos que são descritivos segundo a descrição do problema de programação. Neste sentido, *ntd* conta o número de termos usados na construção do identificador que representam radicais ou flexões de palavras desta descri-

ção. Por exemplo, para o nome "tamanho_tabuleiro", os valores computados de *nc*, *nt* e *ntd* seriam, respectivamente, 17, 2 e 2. Pois, este nome contém 17 caracteres e 2 sequências de caracteres que denotam, além de palavras, termos descritivos de acordo com a descrição do problema de programação.

Além dos valores de cada uma das métricas acima mencionadas, mensuramos quantitativamente a frequência em que a construção do nome do identificador foi usada por todos os alunos em seu código (*fu*), ou em outras palavras, o número de vezes que um dado identificador apareceu em nosso *baseline*.

Adicionalmente, decidimos categorizar as construções dos nomes de identificadores levando em consideração o idioma dos seus termos. Em uma visão geral do *baseline*, percebemos que esta categorização era necessária, pois alguns alunos usaram termos em inglês, além do seu idioma nativo, na construção dos nomes de identificadores. Por esse motivo, usamos o idioma dos termos do identificador para classificar o idioma da sua construção, com a intenção de examinar se o uso do inglês ou português poderia diferenciá-la de "verdadeiro" ou "falso". Neste sentido, usamos o idioma do identificador como um fator neste estudo. Apresentamos esta métrica e as demais empregadas no estudo na Tabela 5.1.

Tabela 5.1: Lista de Métricas Usadas para Diferenciar Nomes de Identificadores

Acrônimo	**Descrição**
NC	Número de caracteres
NT	Número de termos
NTD	Número de termos descritivos
FU	Número de vezes em que foi usado
IDIOMA	Idioma de termos

5.2.3 Coleta de Dados

O *baseline* do estudo inclui 125 nomes de identificadores classificados por nós de acordo com o valor médio dos *scores* atribuídos na avaliação humana sobre a adequação dos nomes de identificadores de código. Encontramos que, dentre os 125 nomes de identificadores, 69 (55,2%) deles foram classificados como "verdadeiros" e 56 (44,8%) deles foram classificados como "falsos". Vale a pena recordar que classificamos como "verdadeiros" aqueles

nomes de identificadores que contribuem para a legibilidade dos programas escritos em resposta a uma descrição de problema de programação, a qual será mostrada mais adiante neste documento. Além disso, dentre os 125 nomes de identificadores, descobrimos que um total de 106 (84,8%) deles foram construídos usando termos em Português, 14 (11,2%) foram construídos usando um idioma que não pudemos identificar e 5 (4,0%) foram construídos usando termos em Inglês.

5.2.4 Resultados e Análises

De acordo com os valores computados de *fu*, dentre os 125 identificadores, 77 (61,6%) das construções usadas pelos alunos são diferentes entre si. Estes identificadores obtiveram valores de *fu* igual a um. Isto significa que, há construções de identificadores que muitos alunos não usam, embora muitos deles tenham usado alguns identificadores específicos uma ou mais vezes. Por exemplo, os nomes de identificadores "i", "linha", "j" e "n" foram usados pelos alunos em seus códigos 44, 35, 20 e 19 vezes, respectivamente. De acordo com a métrica *fu* e observando o comportamento descrito na Figura 5.1, podemos ver que os identificadores "verdadeiros" e "falsos" são similares entre si. Nos *boxplots* de *fu*, apesar de alguns *outliers*, em média, os identificadores "verdadeiros" e "falsos" não mostram uma diferença em seu valor mediano, pois o valor mediano de ambos é igual a 1 nos dois grupos de identificadores.

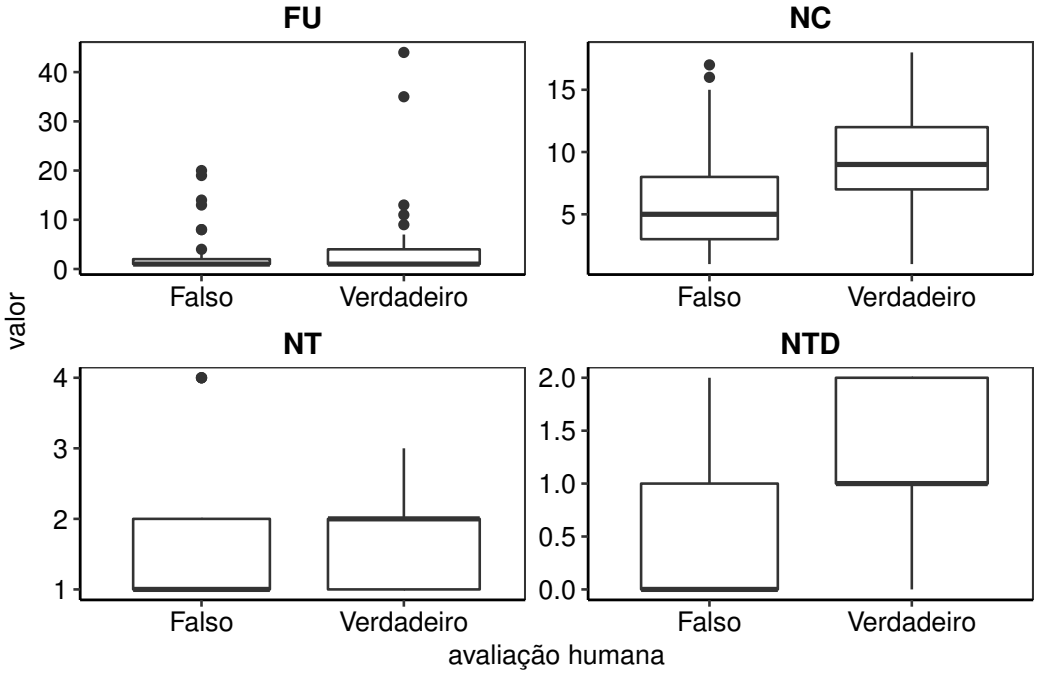

Figura 5.1: Contrastando a Avaliação Humana com as Métricas *fu*, *nc*, *nt* e *ntd*

De acordo com os valores computados de *nc*, dentre os 125 identificadores, 14 (11,2%) deles foram construídos usando letras únicas, o que significa que eles têm apenas um caractere. Apesar dessas construções, muitos estudantes usaram identificadores com 3 ou mais letras. Por exemplo, "tabuleiro_invalido" é um nome de identificador que foi construído usando 18 caracteres, incluindo letras, vogais e separador de sublinhado. Em contraste à métrica *fu*, a análise comparativa entre os identificadores "verdadeiros" e "falsos" usando a métrica *nc* revelou resultados promissores para diferenciá-los. Na Figura 5.1, nos *boxplots* de *nc*, podemos ver que eles são diferentes em relação ao seu número de caracteres, pois, apesar de alguns *outliers*, em média, identificadores "verdadeiros" apresentam um valor mediano maior (9) em contraste aos identificadores "falsos" (5).

De acordo com os valores computados de *nt*, dentre os 125 identificadores, 120 (96,0%) deles, ou seja, a maioria, foram construídos usando um ou dois termos. Em contraste com estes identificadores, alguns alunos usaram construções com três ou quatro termos. Além disso, os nomes de identificadores com quatro termos foram considerados inapropriados pela avaliação humana. No *baseline* do estudo, estes identificadores são: "nNumeroDeLinhas", "nQuantiaDeBola" e "nQuantiaDeXis". Acreditamos que eles foram julgados como "ruins",

pois foram construídos usando uma convenção de nomenclatura inapropriada, incluindo o emprego de preposição e termos redundantes para representar um mesmo conceito (e.g., os termos "n" e "Numero" parecem representar a ideia de contagem, número ou quantidade no identificador "nNumeroDeLinhas"). Além disso, de acordo com a métrica *nt*, a análise comparativa entre os identificadores "verdadeiros" e "falsos" também mostrou que eles são diferentes. Na Figura 5.1, nos *boxplots* de *nt*, apesar de um *outlier*, em média, identificadores "verdadeiros" apresentam um valor mediano maior (2) em contraste aos identificadores "falsos" (1). Isso significa que, além da métrica *nc*, *nt* também parece ser relevante na avaliação humana para diferenciar identificadores apropriados dos inapropriados.

De acordo com os valores computados de *ntd*, dentre os 125 identificadores, constatamos que 49 (39,2%) deles foram construídos sem o uso de termos descritivos, mas um total de 76 (60,8%) deles, ou seja, a maioria, foram construídos usando um (35,2%) ou usando dois (25,6%). Inicialmente, esperávamos que construções de identificadores com um ou dois termos descritivos poderiam ser julgadas como "boas" pela avaliação humana e, na verdade, a maioria são. Analizamos os identificadores "verdadeiros" e "falsos" para constatar a relação deles com a métrica *ntd*. Descobrimos que, dentre os identificadores "verdadeiros", 57 (82,6%) deles foram construídos usando um ou dois termos descritivos e 12 (17,4%) deles foram construídos usando nenhum. Este comportamento contrasta com o dos identificadores "falsos", pois dentre eles 19 (33,9%) foram construídos usando um ou dois termos descritivos e 37 (66,1%) deles foram construídos usando nenhum. Por exemplo, o nome "tamanho_tabuleiro", que inclui os termos descritivos "tamanho" e "tabuleiro", pode ser considerado apropriado para denotar o identificador dentro do código do aluno, pois estes termos permitem comunicar conceitos do problema de programação ao leitor. Na Figura 5.1, nos *boxplots* de *ntd*, em média, os identificadores "verdadeiros" apresentam um valor mediano mais alto (1) em contraste aos identificadores "falsos" (0). Isso significa que, tendo em vista a legibilidade, os identificadores apropriados são, de fato, construídos usando termos da descrição do problema de programação.

Para avaliar se os nomes de identificadores "verdadeiros" e "falsos" têm a mesma distribuição de valores de *fu*, *nc*, *nt* e *ntd*, nós examinamos, para cada uma destas métricas, se as distribuições de seus valores diferem em seu valor mediano. Para estimar o valor médio de diferença de cada uma destas métricas, nós aplicamos o método não paramétrico *bootstrap*

comum para calcular o intervalo de confiança de *bootstrap* da diferença com base em 2000 reamostragens usando o método de aproximação normal de primeira ordem. A partir da análise do intervalo de confiança, nós obtivemos em um nível de confiança 95,0% os seguintes valores medianos de diferença (*fu*=[-0,6; 1,1]), (*nc*=[-6,0; -2,1]), (*nt*=[-1,0; -1,0]) e (*ntd*=[-1,2; -0,8]). O que significa que, diferentemente dos valores de *fu*, os valores de *nc*, *nt* e *ntd* dos nomes de identificadores "verdadeiros" são maiores em média do que os valores destas métricas dos nomes de identificadores "falsos".

Por fim, decidimos selecionar para uma análise qualitativa mais aprofundada apenas os identificadores categorizados com o idioma português e inglês. Como resultado, acabamos filtrando as 14 construções de identificadores que não pudemos categorizar o idioma, pois consideramos que elas não poderiam mostrar alguma diferença entre os dois grupos de identificadores. Chegamos a esta conclusão verificando que, dentre as 14 construções, apenas uma (7,1%) delas tinha sido classificada como "verdadeiro" de acordo com a avaliação humana. No *baseline* do nosso estudo, esta construção única de nome corresponde ao identificador "i". Dentre as 14 construções de identificadores, 13 (92,9%) delas eram referente às que tinham sido classificadas como "falso".

Como resultado da comparação entre os identificadores classificados como em português ou inglês, descobrimos que ambos são apropriados de acordo com a avaliação humana. Em média, identificadores categorizados tanto como português quanto inglês não mostram diferença um do outro, pois apresentam o mesmo valor mediano de *score*, o qual é 4. Analisamos manualmente os identificadores "verdadeiros" e "falsos" para constatar o idioma usado na sua construção. Descobrimos que, dentre os identificadores "verdadeiros", 65 (95,6%) deles foram formados por termos em português e 3 (4,4%) foram formados por termos em inglês. Por outro lado, dentre os identificadores "falsos", 41 (95,3%) deles foram formados por termos em português e 2 (4,7%) foram formados por termos em inglês. Apresentamos estes números na Tabela 5.2, juntamente com os números de identificadores que não pudemos classificar o seu idioma (-), com o objetivo de relacionar cada um dos números deste estudo.

Tabela 5.2: Distribuição do Número de Identificadores Classificados pelo Idioma

	-	Inglês	Português	Total
Verdadeiro	1	3	65	69
Falso	13	2	41	56
Total	14	5	106	125

5.2.5 Discussão

Com o objetivo de discutir estes resultados, devemos lembrar a questão de pesquisa que nos motivou a conduzir este estudo: Como a avaliação humana julga a adequação dos nomes de identificadores de código, tendo em vista a legibilidade. Calculamos os valores das métricas *fu*, *nc*, *nt* e *ntd* para lançar luz sobre as características desejáveis pela avaliação humana ao julgar nomes apropriados de identificadores de código. Como as distribuições dos valores das métricas *nc*, *nt* e *ntd*, computados a partir dos identificadores "verdadeiros" e "falsos", diferem em seu valor mediano, podemos usar estas métricas para realizar o julgamento objetivo das características desejáveis pela avaliação humana.

A partir da análise comparativa das distribuições dos valores da métrica *nc*, podemos afirmar com significância estatística adequada que o número de caracteres incluídos na construção do identificador é uma característica desejável pela avaliação humana. No estudo, a avaliação humana priorizou as construções de nomes de identificadores tendo uma maior quantidade de caracteres. Este resultado fortalece a ideia de usar o número de caracteres para fornecer feedback sobre nomes inapropriados de identificadores. Vale a pena recordar que, em trabalhos relacionados, o feedback é tipicamente entregue ao aluno com o propósito de incentivá-lo a escrever nomes com um maior número de caracteres. **O que nos leva a acreditar que, até certo ponto, quanto maior o número de caracteres incluídos na construção do identificador, melhor ele será**.

Além do uso de um maior número de caracteres, também é importante levar em consideração os termos empregados na construção do nome de identificador. **A partir da análise comparativa das distribuições dos valores das métricas *nt* e *ntd*, podemos concluir com significância estatística adequada que o uso de termos, além do uso de termos descritivos, também são características desejáveis pela avaliação humana**. No estudo, a

avaliação humana priorizou, em grande parte, não apenas as construções de nomes de identificadores tendo um maior número de termos, mas também tendo um maior número de termo da descrição do problema de programação. Para estes dados, isso significa que há evidências que nos permitem fundamentar a conjectura de que nomes apropriados para denotar identificadores de código são, em grande parte, construídos usando palavras da descrição do problema de programação.

Na seção 5.3, nós iremos reportar o estudo de caso descritivo em que visamos verificar e confirmar a validade desta conjectura. Neste sentido, iremos utilizar o *baseline* deste estudo, com o intuito de confirmar a legitimidade do julgamento de nomes apropriados de identificadores fornecido pela avaliação de qualidade automática, supondo que esta pode encontrá-los.

5.3 Avaliação Automática de Qualidade

Nossa iniciativa visando examinar se a avaliação de qualidade automática assemelha-se à avaliação humana, tinha como objetivo contrastar o julgamento de nomes apropriados de identificadores fornecido pelos métodos de avaliação automático e manual. Para atender à este objetivo, nós encontramos nomes apropriados de identificadores, através do julgamento fornecido pela avaliação de qualidade automática, para compará-los ao *baseline* do estudo anterior.

5.3.1 Métodos

Neste estudo, conjecturamos se nomes apropriados de identificadores, para a legibilidade do código, são aqueles que estão baseados em termos da descrição do problema de programação. A lógica por trás do uso de tais termos é que desta maneira as construções de nomes de identificadores poderão favorecer a legibilidade de código, comunicando conceitos do domínio do problema da tarefa de programação proposta.

O que nos leva a crer que é possível usar a estratégia adotada pela avaliação de qualidade automática que propomos para avaliar a sua aptidão em avaliar automaticamente a adequação dos nomes de identificadores. Pois, ela pode nos ajudar a identificar bons termos para nomear identificadores de código de alunos, usando os termos do vocabulário de referência. Vale a pena recordar que esses termos são extraídos a partir do texto da especificação do programa

proposto por um instrutor quando ele cria uma tarefa de programação. Adicionalmente, a avaliação de qualidade automática usa a verificação do emprego desses termos, que são do vocabulário de referência, em construções dos nomes de identificadores do código do estudante para realizar a avaliação.

Considerando essa estratégia, nós examinamos se usando-a poderíamos avaliar nomes de identificadores os verificando e julgando como apropriados similarmente à forma como instrutores fazem. Nós conduzimos o estudo motivado pela seguinte questão de pesquisa:

> QP2: A avaliação de qualidade automática, em certa medida, se assemelha à avaliação humana ao julgar nomes apropriados de identificadores de código?

Para responder à esta questão de pesquisa, nós investigamos se o julgamento de bons nomes de identificadores da avaliação de qualidade automática, fornecida pela estratégia proposta, era semelhante ao julgamento da avaliação humana através do *baseline*. Para examinar isso, nós avaliamos a qualidade de identificadores de código escritos por estudantes usando a avaliação de qualidade automática.

Como o primeiro passo dessa avaliação, nós obtivemos o vocabulário de referência da tarefa problema normalizando o texto da especificação do problema e reunindo os termos resultantes. Subsequentemente, nós avaliamos os identificadores de código dos estudantes verificando se eles empregavam esses termos ou não. Como resultado dessa avaliação de qualidade automática, nós fomos capazes de julgar e classificar identificadores como "positivo" todos os nomes considerados serem apropriados, e como "negativo" todos os outros nomes considerados serem inapropriados, como definido na subseção 3.1.2. Em sequência, comparamos os nomes de identificadores classificados como apropriados, contrastando-os aos julgados como apropriados e inapropriados do *baseline* do estudo.

Nós mostramos o texto da especificação do problema usado para obter o vocabulário de referência usado para avaliar identificadores de código na Tabela 5.3. Como um exercício de programação, o texto escrito em português pede ao estudante para implementar um programa em Python para detectar e informar o jogador que é bem sucedido em colocar "x" ou "o" em uma linha horizontal, vertical, ou diagonal em um tabuleiro quadrado de dimensão "n", lembrando o Jogo da Velha.

A ilustração apresentada na Figura 5.2 mostra um exemplo de como a metodologia deste

Tabela 5.3: Especificação do Problema da Tarefa de Programação Linger

Linger é um jogo de tabuleiro em que dois jogadores se enfrentam para conquistar o maior número de pontos. Os jogadores são representados por xis **x** e bola **o**. O tabuleiro do jogo é um quadrado de lado **n**. As regras do jogo são complexas, e não trataremos delas aqui. Entretanto, para determinar qual jogador ganhou ou se houve empate basta contar os pontos: a quantidade de ocorrências de **x** ou **o** no tabuleiro. Quem tiver mais pontos, ganha. Pede-se que você escreva um programa que, dado um tabuleiro *Linger* válido, verifique e informe quem ganhou e com quantos pontos ou se houve empate. **Entrada:** O programa deve ler da primeira linha da entrada um número **n**, maior que 1, que representa o tamanho do lado do tabuleiro. Em seguida, receberá as linhas contendo o símbolo que existe em cada casa do tabuleiro. Assuma que apenas **x** e **o** serão informados no conteúdo das linhas do tabuleiro. **Saída:** Deve apresentar uma mensagem informando qual jogador foi o vencedor, a quantidade de pontos feita ou se houve um empate. Caso o tabuleiro informado pelo usuário seja inválido (quando há uma linha maior ou menor que o lado **n**), o programa deve emitir uma mensagem. Observe os exemplos de entrada e saída.

estudo foi aplicada. O primeiro oval representa um conjunto de seis identificadores obtidos a partir do conjunto de identificadores usados neste estudo. Nós queremos contrastar a avaliação de qualidade automática, aqui proposta, com a avaliação manual dos instrutores. Os termos "cont_x", "cont_o", "num_bolas" e "valid", na sequência da seta de avaliação automatizada, são julgados como "positivo", significando que estes nomes empregam termos como "cont", "num", "bola" ou "valid". Estas palavras são parte do vocabulário de referência obtidas a partir do processo de normalização do texto mostrado na Tabela 5.3. Assim, os termos "cont_x", "cont_o", "num_bolas" e "valid" parecem ser bons identificadores de acordo com a avaliação automatizada. Os termos "cont_x", "cont_o", "i" e "num_bolas", na sequência da seta de avaliação humana, são julgados como "verdadeiro", significando que eles são bons identificadores de acordo com a avaliação dos instrutores.

Resumindo a análise comparativa de ambos os métodos de avaliação, os identificadores

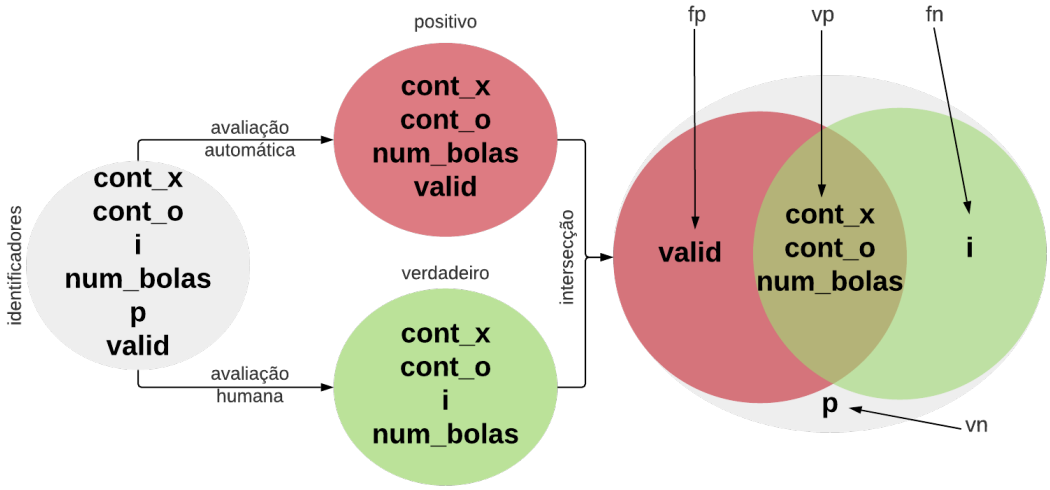

Figura 5.2: Contrastando a Avaliação Manual dos Instrutores com a Avaliação Automática de Qualidade

julgados como "positivo" e "verdadeiro" são contrastados na sequência da seta de interseção. Os nomes "cont_x", "cont_o" e "num_bolas" são identificadores "verdadeiros positivos" (*vp*), significando que eles são bons identificadores de acordo com a avaliação automatizada e a avaliação dos instrutores. O nome "i" é um identificador "falso-negativo" (*fn*), significando que, embora ele é julgado como "verdadeiro" pelos instrutores, ele é julgado como "negativo" pelo método de avaliação proposto. Por outro lado, o nome "valid" é um identificador "falso-positivo" (*fp*), embora ele é julgado como "positivo" pela avaliação automatizada, ele não é julgado como "verdadeiro" pelos instrutores. Finalmente, o nome "p" é um identificador "verdadeiro negativo" (*vn*), significando que ele é um identificador ruim de acordo com ambos métodos de avaliação.

5.3.2 Métricas

Nós avaliamos o quão similar é o julgamento de nomes apropriados de identificadores da avaliação de qualidade automática, em contraste ao *baseline* do estudo, usando duas métricas aceitas largamente da área de recuperação da informação: *Precision* e *Recall* [FBY92].

Precision mensura, no contexto do nosso estudo, a fração do "número de identificadores julgados como positivo pela avaliação de qualidade automática, que são identificadores verdadeiros de acordo com o *baseline* do estudo" pelo "número total de identificadores julgados como positivo pela avaliação de qualidade automática". *Precision* computa a corretude da

avaliação de qualidade automática em detectar identificadores verdadeiros. *Recall*, por outro lado, mensura a fração do "número de identificadores julgados como positivo pela avaliação de qualidade automática, que são identificadores verdadeiros de acordo com o *baseline* do estudo" pelo "número total de identificadores verdadeiros de acordo com o *baseline* do estudo". *Recall* computa a completude da avaliação de qualidade automática quando detecta identificadores verdadeiros.

A avaliação destes aspectos, corretude e completude, vão lançar luz sobre QP2. Dado isso, temos as seguintes hipóteses concretas:

> $H1_1$: A avaliação de qualidade automática proposta em contraste com a avaliação humana tem corretude aceitável, considerando seu valor computado de *precision*.
>
> $H2_1$: A avaliação de qualidade automática proposta em contraste com a avaliação humana tem completude aceitável, considerando seu valor computado de *recall*.

Para verificar estas hipóteses, nós consideramos a opinião de instrutores que participaram do estudo sobre quão aplicáveis são os níveis de corretude e completude da avaliação de qualidade automática em um curso de programação. As hipóteses nulas correspondentes a cada uma das hipóteses alternativas mencionadas, considera que os valores computados das métricas não são aceitáveis para a corretude e completude, respectivamente. Nós atribuímos os rótulos $H1_0$ e $H2_0$ para estas hipóteses. Neste sentido, *precision* e *recall* são consideradas variáveis dependentes neste estudo.

5.3.3 Coleta de Dados

Como um resultado da comparação entre a avaliação manual dos instrutores e a avaliação automatizada da qualidade, nós calculamos o número de identificadores considerados ser verdadeiro positivo (*vp*), falso-positivo (*fp*), falso-negativo (*fn*) e verdadeiro negativo (*vn*), conforme definido na subseção 5.3.1. Estes valores são apresentados na Tabela 5.4.

Tabela 5.4: Matriz de Confusão

		Avaliação Humana		
		verdadeiro	falso	total
Avaliação Automática	positivo	57 (*vp*)	19 (*fp*)	76
	negativo	12 (*fn*)	37 (*vn*)	49
	total	69	56	125

5.3.4 Resultados e Análises

Com base nos dados da Tabela 5.4, nós calculamos os valores de *precision* ($p = \frac{vp}{vp+fp}$ = 75,0%) e *recall* ($r = \frac{vp}{vp+fn}$ = 82,6%). Com o objetivo de coletar evidências estatísticas, nós estudamos esses mesmos dados estimando os valores de *precision* e *recall*. Inicialmente, nós usamos o método não paramétrico ordinário *bootstrap* para recalcular os valores de *precision* e *recall* 2000 vezes [Efr87] ao reamostrar 125 identificadores, identificados como *vp*, *fp*, *fn* e *vn* a partir do conjunto original de identificadores, com substituição.

A partir desses valores recalculados de *precision* e *recall*, nós calculamos os intervalos de confiança de *bootstrap* dessas métricas usando o método BCA. A partir das análises dos intervalos de confiança, nós obtivemos em um nível de confiança de 95,0% um valor de *precision* de [63,7%; 83,5%] e um valor de *recall* de [71,8%; 90,4%]. O valor de *recall* é relativamente maior que o valor de *precision*, corroborando os resultados obtidos com os dados originais.

5.3.5 Discussão

Com o objetivo de discutir esses resultados, nós devemos lembrar a questão de pesquisa que inicialmente nos intrigou: Se a avaliação de qualidade de forma automática, em certa medida, se assemelha à avaliação humana ao julgar nomes apropriados de identificadores de código. Nós calculamos os valores das métricas *precision* e *recall* para realizarmos o julgamento objetivo sobre a corretude e a completude da avaliação automatizada da qualidade. **Em nosso método de avaliação automatizado, a completude é maior que a corretude. Além disso, nós enfatizamos que os valores da corretude e completude são relativamente próximos da totalidade, o que significa que o método de avaliação automatizado**

proposto é promissor para encontrar bons identificadores.

Como o valor estimado de *recall* é [71,8%; 90,4%], nós podemos afirmar com significância estatística adequada que a avaliação automatizada proposta pode julgar corretamente a maioria dos nomes apropriados de identificadores julgados como "verdadeiro" pela avaliação humana. Embora o valor estimado de *precisão* [63,7%; 83,5%] seja ligeiramente inferior ao valor de *recall*, nós podemos afirmar que a avaliação automatizada proposta pode julgar corretamente a maioria dos nomes apropriados de identificadores julgados como "verdadeiro" de forma similar à avaliação humana. Isso significa que há evidências de que a avaliação automatizada da qualidade assemelha-se à avaliação humana, considerando os aspectos de corretude e completude.

A partir desse resultado e de acordo com os instrutores participantes do estudo, nós afirmamos que o método de avaliação automatizado proposto tem níveis aceitáveis de corretude e completude. Portanto, uma ferramenta que instancia o método proposto tendo esses níveis é aplicável ao curso de programação introdutória onde conduzimos esta pesquisa. Em consequência, nós rejeitamos as hipóteses nulas $H1_0$ e $H2_0$ em favor das alternativas.

5.3.6 Limitações

Como a avaliação automatizada proposta não abrange todos os aspectos levados em consideração pela avaliação humana, então é esperado que a primeira não seja totalmente semelhante à segunda. Em consequência, nosso método de avaliação automatizado proposto poderia julgar identificadores *fn* (falso-negativo) e *fp* (falso-positivo). Nós avaliamos manualmente cada um desses identificadores para descobrir em que casos o julgamento da avaliação automatizada de qualidade diverge da avaliação humana.

Observamos que os estudantes escolheram como identificadores em seus códigos, acrônimos, abreviações, sinônimos, termos em inglês ou de loop. Assim, categorizamos os identificadores *fn* em: sinônimo, abreviação, idioma, e letra única. Apesar desses identificadores serem julgados como "verdadeiro" pelos instrutores, eles foram julgados como "negativo" pela avaliação automatizada. Como cada um dos identificadores *fn* não emprega termos do vocabulário de referência, nós categorizamos um identificador *fn* como:

- Sinônimo: Identificadores semanticamente relacionados ao vocabulário de referência.

No contexto do estudo, o nome "pontuacao_x" é um exemplo, pois ele emprega o léxico "pontuacao". Embora este léxico não faça parte do vocabulário de referência, ele é sinônimo de "contagem", palavra flexionada em um dos termos do vocabulário de referência: "cont";

- Abreviação: Identificadores incluindo algum léxico abreviado do mesmo vocabulário (e.g., o nome "qtd_o", pois o léxico "qtd" é acrônimo de "quantidade", palavra flexionada em um dos termos do vocabulário de referência: "quant");

- Idioma: Identificadores incluindo algum léxico do mesmo vocabulário em outro idioma (e.g., o nome "count_o", pois o léxico "count" é a tradução em inglês de um dos termos do vocabulário de referência: "cont");

- Letra única: Identificadores denotando letra única (e.g., o termo "i", pois ele corresponde a um caractere único).

A figura 5.3 sumariza, a fração do "número de identificadores *fn* de uma dada categoria", pelo "número total de identificadores *fn*". A partir desta figura, nós observamos que 50,0% dos identificadores *fn* incluem sinônimos dos termos do vocabulário de referência. Isso significa que, embora esses identificadores não incluam termos do vocabulário de referência, uma quantidade significativa deles emprega sinônimos de seus termos.

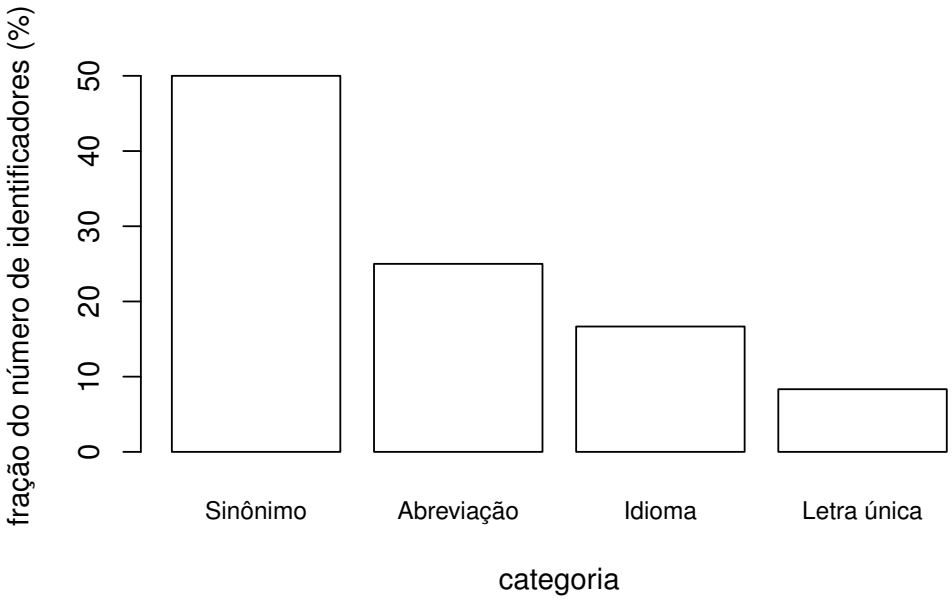

Figura 5.3: Fração do Número de Identificadores *fn* em cada Categoria

Além disso, nós categorizamos todos os identificadores *fp* em: ambíguo, convenção de nomeação, e enganoso. Nestes casos os estudantes escolheram como identificadores em seus códigos nomes vagos, enganosos ou mal formados. Apesar desses identificadores serem julgados como "positivo" pela avaliação automatizada, eles foram julgados como "falso" pelos instrutores. Como cada um dos identificadores *fp* empregam termos do vocabulário de referência, nós categorizamos um identificador *fp* como:

- Ambíguo: Identificadores incluindo léxicos vagos semanticamente (e.g., o termo "cont", pois não permite a um leitor entender especificamente o que realmente se pretende contar);
- Convenção de nomeação: Identificadores que não empregam adequadamente convenção de nomenclatura (e.g., o nome "nQuantiaDeBola", pois reúne léxicos acompanhados de um *stop-word*: "De");
- Enganoso: Identificadores que podem enganar o leitor (e.g., o nome "tamanhoLado", pois permite conduzir um leitor ao engano do quê realmente se trata o lado em questão).

A figura 5.4 sumariza, para cada uma dessas categorias, a fração do "número de identificadores *fp* de uma dada categoria", pelo "número total de identificadores *fp*". A partir desta figura, nós observamos que 52,6% dos identificadores *fp* incluem léxicos vagos semanticamente do vocabulário de referência. Isso significa que, embora esses identificadores incluam termos do vocabulário de referência, uma quantidade significativa deles incluem léxicos que não permitem a um leitor entender completamente o real propósito deles.

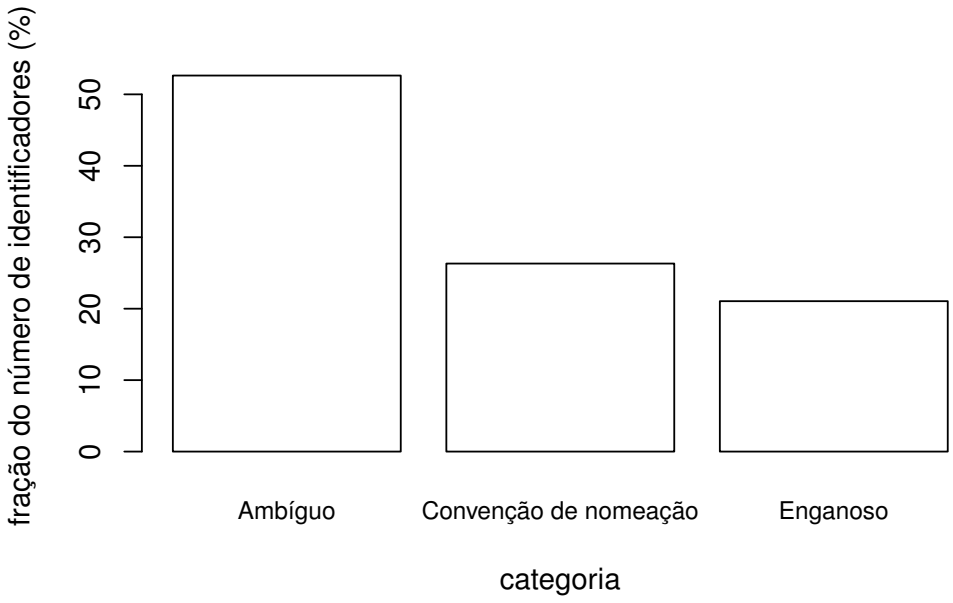

Figura 5.4: Fração do Número de Identificadores *fp* em cada Categoria

5.3.7 Conclusão

A partir dos resultados do contraste dos métodos de avaliação, percebemos que há divergências entre alguns dos identificadores avaliados.

Em geral, esses identificadores parecem ter sido mal escolhidos pelos alunos já que empregavam termos descontextualizados ou indiretamente relacionados aos termos do vocabulário de referência. Em uma avaliação manual de tais termos, observamos que eles não seguiam convenção de nomeação ou empregavam sinônimos, abreviações, acrônimos ou termos em inglês do vocabulário de referência.

Quando propomos nosso método de avaliação automatizado, nós esperávamos que seu

julgamento de nomes apropriados de identificadores apresentasse alguma semelhança com o julgamento da avaliação humana. Além disso, nós também esperávamos que pudesse haver aspectos levados em consideração pelo julgamento da avaliação humana que não fossem observados pelo julgamento da proposta. Este comportamento era esperado uma vez que a avaliação automatizada não pretendia simular perfeitamente o julgamento de nomes apropriados de identificadores da avaliação humana. Ela pretende, tão somente, usar termos do vocabulário de referência como um padrão para a qualidade de nomeação de identificadores.

Diante disso, este estudo mostra que há casos em que o julgamento de nomes apropriados de identificadores da avaliação humana e da avaliação automatizada não são idênticos, mas, em geral, os julgamentos fornecidos pela segunda são significativos. A linha de fundo é que este resultado nos dá confiança de que a estratégia da avaliação automatizada proposta pode gerar julgamentos de nomes apropriados de identificadores úteis e representativos do julgamento da avaliação humana. Neste sentido, nós podemos confiar no método de avaliação automatizado proposto, usando a descrição do problema de programação para gerar o feedback sobre a adequação dos nomes de identificadores.

Capítulo 6

Sobre o Uso das Mensagens de Feedback Automático

Neste Capítulo, nós discutiremos a efetividade de gerar e fornecer mensagens de feedback automático sobre programas de alunos quando eles respondem problemas de programação. Nós vamos investigar e demonstrar que é possível atender ao objetivo de fornecer dicas, sobre a adequação dos nomes escolhidos pelo aluno para denotar identificadores de seu código, gerando e fornecendo as mensagens de feedback de *IQCheck*. Nós reportaremos um experimento *in situ* conduzido com estudantes do curso de LP1 divididos em grupos de controle e experimento. A principal diferença desses grupos é que: apesar de terem recebido instrução para melhoria da qualidade dos nomes de identificadores, apenas os alunos do grupo experimental fizeram uso de *IQCheck*. No estudo, investigamos se nossa ferramenta poderia ajudar alunos a produzir nomes de identificadores de melhor qualidade.

Adicionalmente, apresentamos um estudo conduzido com os estudantes do grupo experimental com o objetivo de avaliar a utilidade das mensagens de feedback de *IQCheck*. Nós apresentamos a opinião de alunos que usaram nossa ferramenta para obter mensagens de feedback automático sobre os identificadores produzidos por eles em seus programas. Assim, nós podemos delinear as consequências de prover mensagens de feedback de *IQCheck* para a MNI de estudantes.

6.1 Contexto

Os estudos que nós reportamos neste Capítulo aconteceram no semestre acadêmico de 2018.1. Eles ocorreram em uma atividade extraclasse, realizada no Laboratório de Ciência da Computação 3 - LCC3 na UFCG, no curso de graduação em Ciência da Computação, no contexto do curso de LP1. Nós vamos resumir brevemente a natureza dos estudos.

Na seção 6.2, nós detalhamos um experimento controlado para avaliar a ideia de utilizar nossa proposta para gerar e fornecer feedback sobre avaliação da adequação de identificadores para alunos. Em sequência, na seção 6.3, nós reportamos um *survey* realizado com alunos que adotaram as mensagens de feedback para obter orientação personalizada no processo de melhoria da qualidade de identificadores. Os estudos reportados nestas seções contam com programas produzidos por alunos que consentiram em participar deles, preenchendo os Termos de Consentimento Livre e Esclarecido (TCLE) aprovados pelo comitê de ética em nossa universidade. Além desses programas, o estudo reportado na seção 6.3 conta com registros da avaliação da adequação de identificadores originados por *IQCheck* sempre que os seus usuários tivessem solicitado suas mensagens de feedback.

É importante ter em mente algumas particularidades do grupo de sujeitos que participaram dos estudos reportados. Em geral, a maioria dos alunos era do sexo masculino (86,2%), e suas idades estavam entre 17 e 46 anos. Como LP1 é dividido em dez unidades temáticas e aplica o *mastery learning*, esses alunos estavam em diferentes unidades do curso quando realizamos os estudos em meados de julho de 2018. No período de aplicação do experimento, 20,7% dos alunos tinham adquirido conhecimento de dez unidades do curso ou dominaram o conhecimento pré-requisito exigido por sete unidades.

6.2 Avaliação do Potencial das Mensagens de Feedback Automático

No Capítulo anterior, nós avaliamos a estratégia adotada pelo método proposto para avaliar automaticamente a adequação dos nomes de identificadores. Como resultado prático, encontramos que podemos utilizá-la para julgar nomes apropriados de identificadores de código, no sentido de encontrar e fornecer feedback sobre a adequação de nomes conside-

rados inapropriados, segundo a descrição do problema de programação. Define-se que um identificador tem um nome apropriado quando ele contribui para a legibilidade do código, expressando conceitos do problema de programação. Nesta seção, descreveremos o primeiro estudo conduzido com o propósito de avaliar o potencial do fornecimento do feedback automático através da nossa abordagem, um experimento. Trata-se de uma investigação empírica para demonstrar que podemos ajudar estudantes na melhoria da qualidade de identificadores através do fornecimento do feedback de *IQCheck*, a ferramenta que instancia a proposta desta dissertação. Queremos entender se estudantes que recebem o feedback de qualidade com *IQCheck* tendem a escrever um programa com uma melhor adequação dos nomes de identificadores. No estudo que é relatado nesta seção, lançamos luz sobre evidências para fundamentar a seguinte alegação:

> "Estudantes melhoram a adequação dos nomes de identificadores de seus programas com ajuda do feedback automático e oportuno."

6.2.1 Métodos

Neste estudo, conjecturamos se é possível ajudar alunos a refletir e melhorar a qualidade dos nomes escolhidos por eles para denotar identificadores de seu código. Para ajudá-los, propomos o uso de *IQCheck* para gerar e fornecer mensagens de feedback automático. A lógica por trás dessa proposta é que o feedback de *IQCheck* pode aconselhar estudantes sobre quais nomes de identificadores do seu código podem ser renomeados, com o propósito de melhorar a adequação dos nomes de seus programas. Portanto, nosso experimento foi realizado para responder a seguinte questão de pesquisa:

> QP1: As mensagens de feedback de *IQCheck* ajudam alunos a melhorar a adequação dos nomes de identificadores de seus códigos?

Em resposta a esta questão de pesquisa, nós investigamos o quão efetivas são as mensagens de feedback de *IQCheck* em ajudar estudantes a escolher nomes apropriados para denotar identificadores de seus códigos. Isto é, queremos investigar o quão eficaz é utilizar os termos automaticamente obtidos a partir do texto da especificação do problema como a

referência para melhorar a qualidade de identificadores. Para tanto, investigamos o potencial de fornecer as mensagens de feedback de *IQCheck* para ajudar alunos de um grupo de experimento em contraste a alunos de um grupo de controle.

Inicialmente, nós convidamos 58 estudantes para uma atividade extraclasse com a mesma duração de uma aula regular de laboratório de 120 minutos. Dividimos aleatoriamente esses alunos em grupos de controle e experimento. Nós então avaliamos a efetividade das mensagens de feedback de *IQCheck* em ajudar os alunos do grupo de experimento em contraste aos alunos do grupo de controle. Em nosso desenho de estudo, todos os alunos tiveram suporte de TST para testar a corretude funcional de seus programas. Contudo, apesar de termos instruído verbalmente, e também por escrito, todos os alunos a refletir e melhorar os nomes escolhidos por eles como identificadores de seus programas, apenas os alunos do grupo experimental receberam mensagens de feedback de *IQCheck*. Para receber mensagens de feedback, os alunos do grupo experimental rodaram/executaram *IQCheck* como mencionado na subseção 3.1.2.

Para conduzir este estudo, nós instrumentamos TST conectando a ele *IQCheck*. Nós escolhemos quatro problemas de programação escritos em português, tradicionalmente propostos pelos instrutores em LP1 como exercícios de programação. Estes problemas eram similares entre si, no sentido de que tinham o mesmo nível de complexidade, exigiam o domínio de conceitos de programação de estruturas condicionais e iterativas (loops for) e eram semelhantes em termos de número de palavras únicas. Em geral, essas descrições solicitavam ao aluno que implementassem um programa para realizar computação em loop sobre os principais tipos de dados do Python. Em sequência, usamos *IQCheck* para criar os vocabulários de referência a partir do texto de especificação desses problemas e salvá-los em arquivos *iqcheck.json*. Nós enviamos esses arquivos para o diretório de trabalho do aluno de modo que *IQCheck* pudesse gerar e fornecer o feedback após a solicitação dele localmente. Os nomes dos problemas e os tamanhos de vocabulários criados foram: "bolsa cnpq" (71), "converter matrícula" (59), "maior torcida" (61) e "imprime rank" (61).

Adicionalmente, entregamos e apresentamos aos alunos dois roteiros diferentes da atividade extraclasse. Os roteiros entregues aos alunos do grupo de controle e experimental podem ser encontrados nos Apêndices B e C desta dissertação, respectivamente. Esses roteiros são bem semelhantes. Ambos solicitam aos alunos que desenvolvam programas para

as tarefas de programação propostas, testem completamente e certifiquem-se de que estão funcionalmente corretos. A motivação por trás disso é que não fazia sentido sobrecarregar o aluno com a melhoria da adequação dos nomes de identificadores de seu código antes dele fazer com que o seu programa funcione efetivamente. Na sequência, os roteiros solicitam aos alunos que enviem seus programas em resposta às tarefas de programação propostas. Como a referência para escolher nomes apropriados de identificadores, os dois roteiros aconselham aos alunos que empreguem palavras relacionadas ao texto da tarefa problema fornecido. A principal diferença entre esses roteiros é que, o roteiro entregue aos alunos do grupo experimental incluiu uma seção adicional, fornecendo um breve treinamento sobre como usar nossa ferramenta: "Conhecendo o *IQCheck*".

Finalmente, contamos com a presença de três instrutores e dois alunos da pós-graduação na área de Ciência da Computação para auxiliarem na condução do experimento. Além de meus orientadores, um dos instrutores que fazia parte da equipe pedagógica do curso de LP1 e, portanto, experiente em ensino neste mesmo curso. No final, nós avaliamos a questão de pesquisa proposta sob análise quantitativa.

6.2.2 Métricas

Nós avaliamos quantitativamente o quão efetivas são as mensagens de feedback de *IQCheck* para ajudar alunos a melhorar a qualidade dos identificadores usando duas métricas: *tnai* e Δ_{tnai}. Para um determinado programa de estudante, *tnai* (taxa de nomes apropriados de identificadores) mensura a fração do "número de nomes de identificadores julgados como apropriado por *IQCheck*" pelo "número total de nomes de identificadores". Esta métrica calcula a taxa de identificadores que são bons no código do aluno. Por outro lado, Δ_{tnai} mensura para um determinado par de programas de estudante, a diferença entre o valor calculado da métrica *tnai* do último ($tnai_1$) e primeiro ($tnai_0$) programa funcionalmente correto. Esta métrica computa a diferença da taxa de identificadores julgados como bons entre o último e o primeiro programa correto. Isto é, a evolução da qualidade dos nomes de identificadores no programa que antecede e sucede suas renomeações. Desta métrica, chegamos a três situações:

1. $\Delta_{tnai} < 0$: O número de identificadores julgados como apropriado no último programa

correto é menor do que no primeiro programa correto. Isso significa que o aluno piorou a adequação dos nomes de identificadores de seu código.

2. $\Delta_{tnai} = 0$: O número de identificadores julgados como apropriado no último programa correto não difere do primeiro programa correto. Isso significa que o aluno não piora e nem melhora a adequação dos nomes de identificadores de seu código.

3. $\Delta_{tnai} > 0$: O número de identificadores julgados como apropriado no último programa correto é maior do que no primeiro programa correto. Isso significa que o aluno melhorou a adequação dos nomes de identificadores de seu código.

A avaliação quantitativa dos valores computados da métrica Δ_{tnai} vai nos permitir reunir evidências para responder a nossa questão de pesquisa: Se as mensagens de feedback de *IQ-Check* ajudam alunos a melhorar a adequação dos nomes de identificadores de seus códigos. Considerando esta questão de pesquisa, estabelecemos a seguinte hipótese:

> $H1_1$: Alunos que recebem as mensagens de feedback de *IQCheck* apresentam, em contraste aos alunos que não as recebem, um valor médio maior de Δ_{tnai} em seus programas.

A hipótese nula correspondente à hipótese alternativa listada acima é que alunos que recebem as mensagens de feedback de *IQCheck* apresentam, em contraste aos alunos que não as recebem, um valor médio igual de Δ_{tnai} em seus programas. Isso significa que, não há evidências para reforçar alguma diferença entre os grupos de controle e experimento. Nós atribuímos o rótulo $H1_0$ para esta hipótese. Neste sentido, Δ_{tnai} é considerada variável dependente neste estudo.

6.2.3 Coleta de Dados

O conjunto de dados utilizado neste estudo é composto por 231 programas submetidos como soluções pelos alunos, utilizando o sistema de avaliação automatizada utilizado em nosso curso - TST. Os programas submetidos pelos alunos tiveram a correção funcional deles testada por TST. Nós utilizamos a verificação de correção funcional dos programas como ponto de partida para filtrar e reunir os programas a serem incluídos no conjunto de dados.

Uma visão geral do conjunto de dados utilizado no estudo mostra um comportamento inicialmente esperado: muitos alunos simplesmente não chegam a entregar para uma determinada tarefa uma outra versão do primeiro programa correto. Pelo menos para o nosso conjunto de dados, alunos de ambos os grupos do experimento mostram este comportamento em uma maneira parecida. Como estamos medindo a evolução da qualidade dos nomes de identificadores no programa, para fazer parte do nosso conjunto de dados, precisamos do par de programas enviados para uma determinada tarefa de programação do experimento: o primeiro e último programa. Pois, espera-se que essas duas versões do mesmo programa tenham linhas de código que mostrem a reescrita de nomes de identificadores. Isso é suficiente para tornar duas versões do mesmo programa diferentes e, em consequência, é considerada uma diferença relevante no contexto do nosso estudo. Com isso em mente, excluímos do nosso conjunto de dados programas referente aos alunos que não enviaram essas duas versões do mesmo programa.

Durante a avaliação de QP1, definimos como entrada em nosso conjunto de dados um par composto pelo primeiro e o último programa correto enviado para uma dada tarefa de programação. Assim, contabilizamos um total de 156 entradas no conjunto de dados referindo-se aos 231 programas submetidos pelos alunos. No final, nós consideramos em nossas análises um total de 64 dessas entradas referindo-se aos primeiros e últimos programas corretos enviados pelos alunos. A Tabela 6.1 resume esses dados.

Tabela 6.1: Distribuição de Entradas Contabilizadas em Cada Grupo

	Controle	Experimento	Total
Entradas	33	31	64
Número total de entradas	81	75	156

6.2.4 Resultados e Análises

Avaliamos a eficácia das mensagens de feedback de *IQCheck* em ajudar os alunos do grupo experimental em fornecer nomes de identificadores de código de melhor qualidade, em contraste aos alunos do grupo de controle. Para avaliar isso, nós examinamos os valores calculados da métrica Δ_{tnai} para cada par de último e primeiro programa correto enviados em resposta às tarefas de programação do experimento.

Descobrimos que os valores computados da métrica Δ_{tnai} dos programas do grupo experimental eram maiores que os valores computados da métrica Δ_{tnai} dos programas do grupo de controle. Isso significa que os alunos do grupo experimental mostraram evolução da qualidade de identificadores, fornecendo nomes mais apropriados em comparação aos alunos do outro grupo. Nós mostramos os *boxplots* das distribuições dos valores de Δ_{tnai} e esse comportamento na Figura 6.1.

Na Figura 6.1, no gráfico da tarefa "bolsa cnpq", o primeiro *boxplot* (grupo de controle) é curto, apresenta uma variação mais estreita dos valores de Δ_{tnai} e valor de Δ_{tnai} mediano igual a zero, no eixo vertical, no primeiro grupo de último e primeiro programas corretos. Isso significa que os alunos do grupo de controle não mostraram melhoria na adequação dos nomes de identificadores de seus códigos como um resultado do não uso de *IQCheck* para receber suas mensagens de feedback. O segundo *boxplot* (grupo experimental) embora seja curto, ele apresenta uma variação relativamente maior dos valores de Δ_{tnai} e um valor de Δ_{tnai} mediano igual a 15,0%. Isso significa que, em comparação aos alunos do grupo de controle, os alunos do grupo experimental apresentaram melhoria na adequação dos nomes de identificadores de seus códigos como resultado do uso de *IQCheck* e suas mensagens de feedback.

Nos gráficos das tarefas "converter matrícula", "imprime rank" e "maior torcida", apesar de algumas pequenas diferenças, podemos observar um comportamento similar: apesar de alguns valores discrepantes e uma pequena variação de valores de Δ_{tnai}, os outros *boxplots* do grupo de controle apresentam o mesmo comportamento. Em contraste com os *boxplots* deste grupo, os *boxplots* do grupo experimental apresentam uma variação relativamente maior de valores de Δ_{tnai} e valores medianos de Δ_{tnai} maiores, uma vez que eles são superiores a zero. Os valores medianos de Δ_{tnai} apresentados pelos *boxplots* do grupo experimental, para as tarefas de programação "converter matrícula", "imprime rank" e "maior torcida", foram, respectivamente, 33,0%, 14,0% e 33,0%.

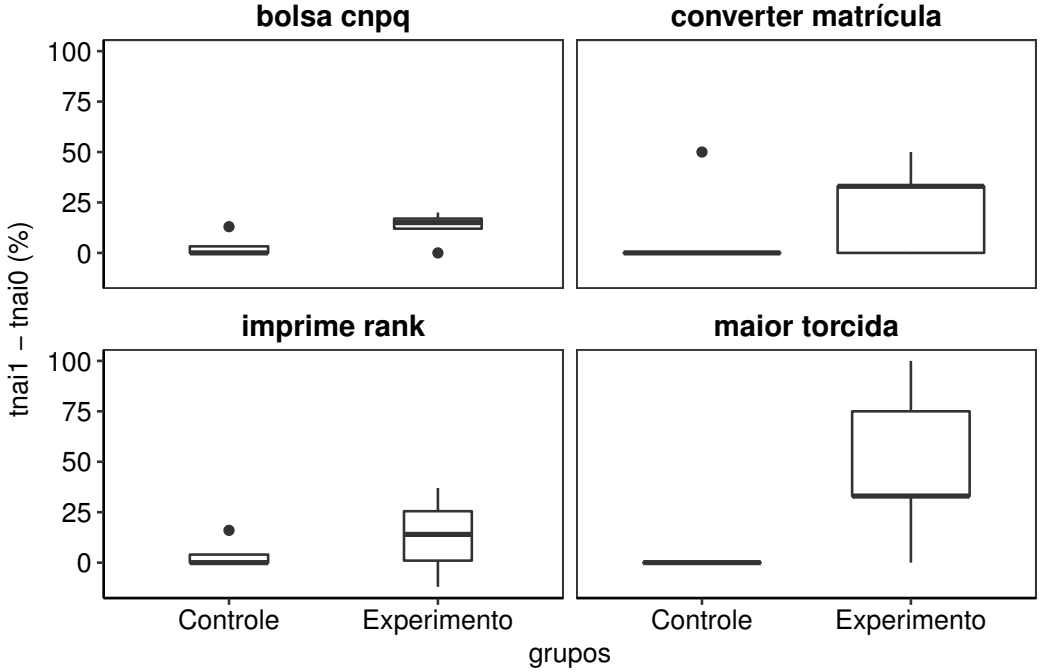

Figura 6.1: *Boxplots* das Distribuicões dos Valores de Δ_{tnai}

Para avaliar se os pares de programas de ambos os grupos do experimento têm a mesma distribuição de valores de Δ_{tnai}, nós examinamos se as distribuições de valores de Δ_{tnai} diferem em seu valor mediano. Para estimar este valor médio de diferença, nós aplicamos o método não paramétrico *bootstrap* comum para calcular o intervalo de confiança de *bootstrap* da diferença com base em 2000 reamostragens usando o método de aproximação normal de primeira ordem. A partir da análise do intervalo de confiança, nós obtivemos em um nível de confiança 95,0% um valor mediano de diferença de [-48,2%; -26,7%], o que significa que os valores de Δ_{tnai} do grupo experimental são maiores em média do que os valores de Δ_{tnai} do outro grupo.

6.2.5 Discussão

Para discutir esses resultados, nós devemos lembrar a questão de pesquisa que nos motivou a conduzir este estudo: Se as mensagens de feedback de *IQCheck* ajudam alunos a melhorar a adequação dos nomes de identificadores de seus códigos. Nós computamos os valores das métricas *tnai* e Δ_{tnai} para realizarmos o julgamento objetivo sobre a eficácia das mensagens de feedback de *IQCheck*. Como as distribuições de valores computados da métrica Δ_{tnai} dos

grupos controle e experimental diferem em seu valor mediano, nós podemos afirmar com significância estatística adequada que existe uma diferença significativa e relevante entre os grupos.

Assim, temos evidências que as mensagens de feedback de *IQCheck* são úteis e adequadas para ajudar alunos a escolher nomes melhores para denotar identificadores de seus programas. Constatamos que 15 alunos (51,7%) do grupo experimental foram capazes de escolher nomes de identificadores de melhor qualidade depois de receber as mensagens de feedback de *IQCheck*. Em contraste com o outro grupo, esse número corresponde a cinco vezes o número de alunos - 3 (10,3%) - que foram capazes de escolher nomes de identificadores de melhor qualidade sem receber as mensagens de feedback de *IQCheck*. O último e primeiro programa correto submetido por eles mostraram evolução da qualidade dos nomes de identificadores, revelando valores de Δ_{tnai} maiores que zero. Isto significa que esses alunos conseguiram melhorar a adequação dos nomes de identificadores de seus códigos. Em consequência, nós rejeitamos a hipótese nula $H1_0$ em favor da alternativa.

6.2.6 Demonstração de Casos de Sucesso

No Capítulo 3, nós mostramos na Figura 3.2, o primeiro programa correto submetido como uma solução para o texto da Tabela 3.1 por um dos alunos do grupo experimental. A partir deste ponto, nós iremos nos referir a esse aluno como A1. Quando A1 solicitou as mensagens de feedback de *IQCheck*, ele forneceu na linha de comando o bloco de mensagens de aviso com relação aos identificadores nas linhas 2, 3, 4, 5, 7 e 8.

Na Figura 6.5, nós mostramos o último programa correto submetido como solução por A1 após ele ter atuado no primeiro programa, renomeando os nomes dos identificadores aconselhados pelas mensagens de feedback de *IQCheck*. Quando A1 solicitou as mensagens de feedback de *IQCheck* para esse programa, *IQCheck* gerou na interface da linha de comando um outro bloco de mensagens de aviso mostrado nenhum aviso sobre os nomes de identificadores dele. A partir deste exemplo, nós podemos observar um comportamento surpreendente de A1 ao consumir as mensagens de feedback de *IQCheck*: A1 renomeia os nomes "ent", "i" e "k" para o nome "torcedor". Embora A1 use nomes repetidos como identificadores nesse programa, o último programa correto ainda mostra que, em contraste ao primeiro programa correto, A1 conseguiu fornecer identificadores de código de melhor

qualidade.

```
#coding: utf-8
torcida_a = 0
torcida_b = 0
for torcedor in range(5):
    torcedor = int(raw_input())
    torcida_a += torcedor
for torcedor in range(5):
    torcedor = int(raw_input())
    torcida_b += torcedor
if torcida_a > torcida_b:
    print "O primeiro time leva mais torcedores ao estádio."
elif torcida_b > torcida_a:
    print "O segundo time leva mais torcedores ao estádio."
else:
    print "Empate."
```

Listing 6.5: Último Programa Correto

Deste ponto em diante, nós vamos apresentar o caso de sucesso referente ao programa produzido por A2: um outro usuário de *IQCheck*. Na Figura 6.6, nós mostramos o primeiro programa correto submetido como uma solução por A2 para o mesmo texto.

```
1  #coding: utf-8
2  torcida1 = 0
3  torcida2 = 0
4  for i in range(5):
5      torcedores = int(raw_input())
6      torcida1 = torcida1 + torcedores
7  for x in range(5):
8      torcedores = int(raw_input())
9      torcida2 = torcida2 + torcedores
10 if torcida1 > torcida2:
11     print "O primeiro time leva mais torcedores ao estádio."
12 elif torcida2 > torcida1:
13     print "O segundo time leva mais torcedores ao estádio."
14 elif torcida1 == torcida2:
15     print "Empate."
```

Listing 6.6: A2 - Primeiro Programa Correto

Quando A2 solicitou as mensagens de feedback de *IQCheck*, *IQCheck* gerou na linha de comando o bloco de mensagens de aviso mostrado na Figura 6.7 com relação aos identificadores nas linhas 2, 3, 4, e 7.

```
$ tst iqcheck 92486fc69021fac0ec39a3bb0d36741d-1-anon.py
# 92486fc69021fac0ec39a3bb0d36741d-1-anon.py

**4 Advertência(s)**

### Identificadores
- *i* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *x* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
- *torcida2* não parece ser um nome adequado. Use palavras da descrição d
a tarefa de programação.
- *torcida1* não parece ser um nome adequado. Use palavras da descrição d
a tarefa de programação.
```

Listing 6.7: A2 - Primeira Chamada de *IQCheck*

Na Figura 6.8, nós mostramos o último programa correto submetido como solução por A2 após ele ter atuado no primeiro programa, renomeando alguns dos nomes de identificadores aconselhados pelas mensagens de feedback de *IQCheck*.

```
1  #coding: utf-8
2  soma_primeiro_time = 0
3  soma_segundo_time = 0
4  for i in range(5):
5      quantidade_primeiro_time = int(raw_input())
6      quantidade_segundo_time = int(raw_input())
7      soma_primeiro_time += quantidade_primeiro_time
8      soma_segundo_time += quantidade_segundo_time
9  if soma_primeiro_time > soma_segundo_time:
10     print "O primeiro time leva mais torcedores ao estádio."
11 elif soma_segundo_time > soma_primeiro_time:
12     print "O segundo time leva mais torcedores ao estádio."
13 elif soma_primeiro_time == soma_segundo_time:
14     print "Empate."
```

Listing 6.8: A2 - Último Programa Correto

Neste sentido, quando A2 solicitou as mensagens de feedback de *IQCheck* para esse programa, ele gerou na linha de comando o bloco de mensagens de aviso mostrado na Figura 6.9 com relação ao identificador na linha 4.

```
$ tst iqcheck 92486fc69021fac0ec39a3bb0d36741d-2-anon.py
# 92486fc69021fac0ec39a3bb0d36741d-2-anon.py

**1 Advertência(s)**

### Identificadores
- *i* não parece ser um nome adequado. Use palavras da descrição da taref
a de programação.
```

Listing 6.9: A2 - Última Chamada de *IQCheck*

Em comparação ao último programa correto de A1, o último programa correto de A2 também mostra que ele conseguiu fornecer nomes de identificadores de código de melhor qualidade. Contudo, A2 rejeita a mensagem de feedback relacionada ao nome de identificador "i", em contraste ao comportamento de A1, ao consumir as mensagens de feedback de *IQCheck*. Consideramos esta rejeição um comportamento positivo, uma vez que, o termo "i" é tipicamente empregado por programadores para denotar identificadores de loop sem prejudicar a legibilidade do programa.

6.3 Opinião de Alunos Sobre a Utilidade das Mensagens de Feedback Automático

O objetivo deste estudo foi coletar observações e comentários dos estudantes sobre a experiência do uso de *IQCheck*. A questão de pesquisa levantada por este estudo foi:

> QP1: Estudantes acham que as mensagens de feedback de *IQCheck* são úteis em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas?

Nós queremos entender melhor a percepção pós-feedback dos estudantes do estudo reportado na seção anterior a partir do seu ponto de vista. Particularmente, nós queremos entender se *IQCheck* é fácil de usar e se o feedback automatizado gerado por ele realmente é útil e relevante para ajudar o aluno a trabalhar sobre a melhoria da nomeação de identificadores.

Para conduzir este estudo, nós primeiramente convidamos os usuários de *IQCheck*, do estudo descrito na seção anterior, para participar de um questionário pós-experimento. Eles foram entrevistados respondendo um mesmo questionário composto por um conjunto de perguntas de uma forma estruturada. Em seguida, organizamos os dados coletados e sumarizamos os resultados correspondentes a cada pergunta colocada. A partir dos resultados, delineamos as conclusões sobre a opinião dos usuários de *IQCheck*. No estudo que relatamos nesta seção, reunimos indícios para apoiar a seguinte alegação:

> "Estudantes acham que o feedback automático e oportuno é útil em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas."

6.3.1 Métodos

Neste estudo, nós adotamos o método de pesquisa denominado *survey*: um método tradicionalmente usado em pesquisa social. Esta metodologia de pesquisa humana é frequentemente usada por psicólogos e sociólogos no contexto de pesquisa na área de ciência social para analisar comportamentos humanos, coletando e reunindo pensamentos, opiniões e sentimentos. Ao aplicar esta metodologia de pesquisa, nós tínhamos a intenção de coletar as opiniões dos alunos sobre o uso de *IQCheck* e suas mensagens de feedback automático. Por isso, os sujeitos convidados a participar do presente estudo eram alunos que utilizaram voluntariamente *IQCheck* para avaliar a qualidade dos nomes de identificadores de seus programas durante o estudo mencionado na seção anterior.

Inicialmente, convidamos os usuários de *IQCheck* para responder nosso questionário pós-experimento através de uma seção adicional no documento escrito, entregue e apresentado aos alunos poucos minutos antes do início do experimento. Pedimos que os alunos registrassem sua opinião após terminar as atividades de programação propostas. O *survey* foi disponibilizado através de um formulário online, composto por um conjunto predeterminado de perguntas com relação à sua experiência de uso. Durante o registro da experiência de uso

no formulário, o pesquisador tomou distância dos usuários de *IQCheck* com a intenção de reduzir sua influência na resposta natural dos alunos.

No presente estudo, nós priorizamos avaliar quantitativamente as opiniões referentes à experiência de uso de *IQCheck* dos alunos. Para capturar quantitativamente as opiniões dos alunos, cada uma das seis perguntas do nosso questionário empregaram a escala Likert de 5 pontos, referenciando um conjunto predefinido de valores de resposta variando de 1 a 5. Embora este design de questionário limite que o aluno fale livremente sobre sua experiência de uso como numa entrevista, ele nos dá a agilidade de atender muitos alunos durante o registro das opiniões, evitando o passo de suas transcrições. Na Tabela 6.2, listamos as seis perguntas do nosso questionário. Utilizamos os valores de respostas relativos à estas perguntas para reunir evidências para responder QP1. O instrumento usado para registrar as opiniões dos alunos está no Apêndice D.

Tabela 6.2: Perguntas do Questionário

Acrônimo	**Descrição**
Q1	*IQCheck* é fácil de usar?
Q2	O feedback de *IQCheck* é útil?
Q3	O feedback de *IQCheck* ajuda a pensar sobre quais nomes são bons?
Q4	O feedback de *IQCheck* é relevante para você?
Q5	O feedback de *IQCheck* fornece dicas sobre quais nomes são bons?
Q6	O feedback de *IQCheck* reforça a importância de escrever programas com bons nomes?

6.3.2 Filtragem de Opiniões

Inicialmente, nós pretendíamos contar com a análise de todas as opiniões registradas pelos alunos que usaram nossa ferramenta através do nosso questionário. De acordo com nosso conjunto de dados, nós poderíamos contar com as opiniões registradas por 28 (96.6%) usuários de *IQCheck*. Curiosamente, apenas um aluno não usou nossa ferramenta no experimento. Este mesmo aluno recusou participar do presente estudo.

Entretanto, nós observamos que uma análise mais criteriosa das opiniões coletadas deveria levar em conta que nem todas elas poderiam ser analisadas de uma mesma maneira.

Embora muitos alunos tenham registrado sua opinião, nem todos apresentaram, para pelo menos uma das atividades de programação propostas, duas versões do mesmo programa, ou sequer requisitou as mensagens de feedback de *IQCheck*.

Assim, filtramos as opiniões registradas levando em consideração o fato do aluno ter apresentado: (1) pelo menos duas versões do mesmo programa para pelo menos uma das tarefas de programação propostas no experimento, e (2) requisições de mensagens de feedback de *IQCheck*, ao passo que trabalhava na melhoria da qualidade dos nomes de identificadores de seus códigos. A segunda restrição foi fixada para garantir que estávamos analisando as opiniões registradas pelos alunos que realizaram pelo menos um ciclo de feedback: recebimento de feedback, consumo de feedback e reescrita dos nomes de identificadores de código. Embora esses critérios tenham diminuído drasticamente o número de opiniões analisadas, isso foi feito para aumentar a confiabilidade das conclusões do presente estudo e atenuar viés em relação ao aluno.

Finalmente, é importante mencionar que nós mantivemos em nosso conjunto de dados opiniões registradas por alunos que enviaram duas versões do mesmo programa, independente da última ter mostrado melhoria em consequência do feedback automático. Certamente, se nós não mantivéssemos essas opiniões, nós estaríamos enviesando nossos resultados. Após a etapa de filtragem das opiniões, nós demos sequência a análise dos resultados.

6.3.3 Métricas

Além de mensurar quantitativamente valores de resposta para cada pergunta do nosso questionário, nós decidimos categorizar as opiniões referentes aos alunos levando em consideração a utilização de *IQCheck*. Em uma análise detalhada do log de *IQCheck*, nós percebemos que esta categorização era necessária, pois nem todos alunos usaram nossa ferramenta de uma mesma maneira. Por exemplo, alguns alunos fizeram pouco uso de *IQCheck*, requisitando suas mensagens uma ou duas vezes, enquanto que outros requisitaram muitas mensagens. Por esse motivo, nós categorizamos as opiniões dos alunos quanto ao uso de *IQCheck* ao responder às tarefas de programação durante o experimento. Isso parece ser mais justo quando se conduz qualquer estudo com o objetivo de avaliar o uso de uma ferramenta.

Para categorizar as opiniões dos alunos, inicialmente usamos o coeficiente de correlação de Spearman para medir e avaliar a extensão da correlação entre a quantidade de ocorrên-

cias de uso de *IQCheck* com a quantidade de tarefas de programação respondidas. Esta correlação é relevante no sentido de que ela nos permite perceber se os alunos fazem uso da nossa ferramenta ao passo que respondem as tarefas de programação. Nós constatamos que além de relevante, esta correlação é significativa (0,86 de Spearman *rho* com p-valor < 0,01). Isto significa que os alunos fizeram uso de *IQCheck* à medida que responderam às tarefas de programação, consumindo o feedback automático sobre a qualidade dos nomes de identificadores de seus programas.

Em seguida, nós usamos a quantidade de tarefas de programação respondidas do aluno para classificar a sua proficiência em usar *IQCheck* como: "usuário" (quando ele respondeu pelo menos três tarefas de programação) ou "testador" (em caso contrário). Nós usamos a proficiência do aluno como um fator neste estudo.

6.3.4 Coleta de Dados

Por fim, nosso conjunto de dados foi composto de 11 opiniões, as quais foram obtidas a partir do processo de filtragem usando a abordagem reportada, referentes a 11 usuários de *IQCheck*. As opiniões finais reunidas incluíram alunos de um perfil diversificado de acordo com: desempenho no curso de LP1 (Unidade Temática) e uso de *IQCheck* para obter melhoria da qualidade de identificadores (Proficiência), como pode ser visto na Tabela 6.3. Naturalmente, nós preferimos alunos com os dois perfis de proficiência no uso de *IQCheck*, para que eles pudesse avaliá-lo melhor.

6.3.5 Resultados e Análises

A primeira pergunta do nosso questionário tinha intenção de verificar se os alunos achavam que *IQCheck* era fácil de usar. No experimento, nós lançamos mão de permitir que os alunos pudessem usar *IQCheck* de maneira bastante semelhante à forma como eles estavam habituados a utilizar TST. Alunos podiam requisitar mensagens de feedback de *IQCheck* fazendo sua invocação semelhante à forma como eles invocam TST.

Em resposta a (Q1) "*IQCheck* é fácil de usar?", a maioria dos estudantes (72,7%) concorda que nossa ferramenta é fácil de usar com um valor mediano de resposta de 5 (valor máximo). A Figura 6.2 mostra os *boxplots* das distribuições dos valores de resposta e esse

Tabela 6.3: Participantes do Estudo Qualitativo

Aluno	**Unidade Temática**	**Proficiência**
1	5	usuário
2	9	usuário
3	10	usuário
4	8	usuário
5	4	testador
6	4	testador
7	10	testador
8	8	usuário
9	7	testador
10	8	usuário
11	7	testador

comportamento. Na parte superior esquerda da borda desta figura, o primeiro *boxplot* é muito curto, não apresenta variação dos valores de resposta e um valor mediano igual a 5, no eixo vertical, no grupo de alunos "testador". Isso significa que, para esta pergunta, o valor de resposta 5 é um consenso entre estes alunos.

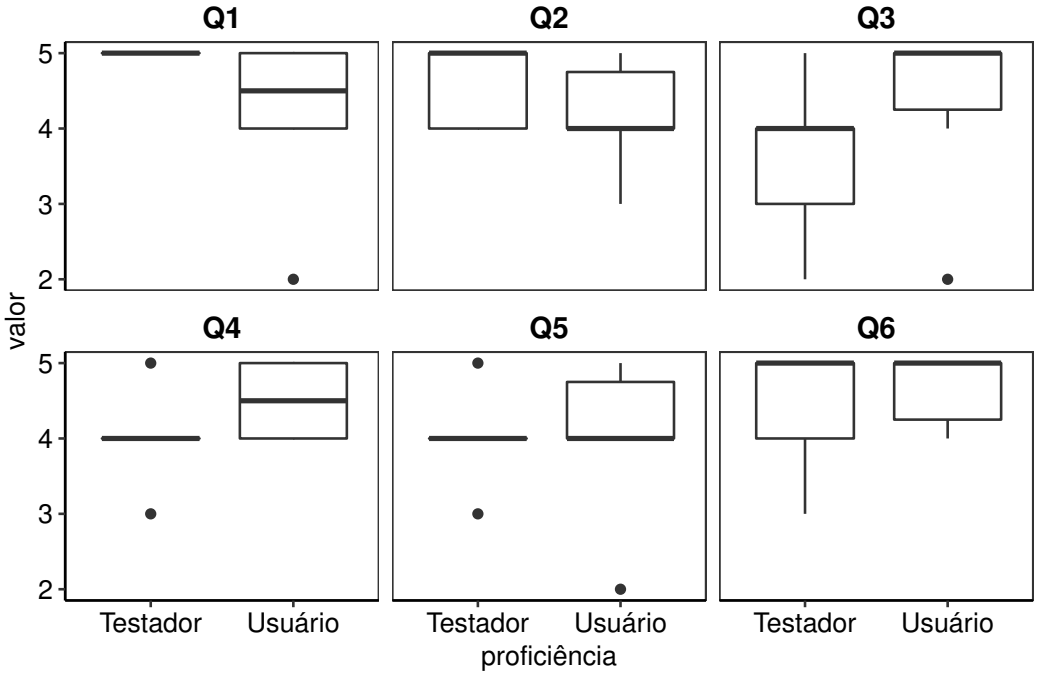

Figura 6.2: *Boxplots* das Distribuicões dos Valores de Resposta

Na segunda, terceira e quinta pergunta, nós queríamos investigar se *IQCheck* realmente gera feedback útil que ajuda alunos fornecendo dicas sobre quais nomes de identificadores podem ser renomeados para a melhoria da sua qualidade. Para investigar isso, nós propusemos três perguntas em nosso questionário: (Q2) "O feedback de *IQCheck* é útil?", (Q3) "O feedback de *IQCheck* ajuda a pensar sobre quais nomes são bons?" e (Q5) "O feedback de *IQCheck* fornece dicas sobre quais nomes são bons?".

Em resposta a Q2, Q3 e Q5, a maioria dos alunos concorda que o feedback de *IQCheck* é útil (45,5%), ajuda a pensar sobre quais nomes são bons (45,5%) e fornece dicas sobre quais nomes são bons (54,5%), com um valor mediano de resposta de 4. Com respeito a categoria dos alunos, nós podemos observar na Figura 6.2 que a pergunta Q5 apresentou um consenso entre os alunos categorizados como "usuário" e "testador" com um valor mediano de resposta de 4. Em contraste a Q5, Q2 e Q3 apresentaram valores medianos de resposta levemente diferentes entre os alunos destas duas categorias, porém, aparentemente, essa diferença não é significativa. Isto significa que, com respeito a Q2 e Q3, tanto alunos "usuário" quanto "testador" teoricamente acham que o feedback de *IQCheck* é útil, ou ajuda a pensar sobre quais nomes são bons, pois a diferença entre os valores medianos de resposta entre os alunos

destas categorias para estas perguntas não é gritante.

Na quarta e sexta pergunta, nós queríamos apurar se alunos consideravam o feedback de *IQCheck* relevante em reforçar a importância de escrever programas com bons nomes de identificadores. Para examinar isso, nós propusemos outras duas perguntas em nosso questionário: (Q4) "O feedback de *IQCheck* é relevante para você?" e (Q6) "O feedback de *IQCheck* reforça a importância de escrever programas com bons nomes?".

Em resposta a Q4 e Q6, nós percebemos que a maioria dos alunos concorda que o feedback de *IQCheck* é relevante (54,5%) e reforça a importância de escrever programas com bons nomes (63,6%) em valores medianos de resposta de 4 e 5, respectivamente. Em resposta a Q4, a Figura 6.2 mostra curiosamente que os alunos "usuário" apresentam um valor mediano de resposta superior em relação aos alunos "testador". Isto significa que os alunos "usuário" investiram tempo no uso de *IQCheck* ao responderem suas tarefas de programação porque eles acharam relevante a melhoria dos nomes de seus programas.

Finalmente, é importante destacar um comentário obtido informalmente com o aluno 11 sobre sua experiência com nossa ferramenta. Este foi o comentário reclamado por cerca de três alunos que fizeram uso de *IQCheck* durante o experimento.

> *"[...], só não gostei dela não aceitar o 'i' e o 'e' como boas variáveis, já que são tão usadas junto com o comando for, fora isso achei uma ferramenta bem útil!"*

De fato, nós percebemos que nem todos os alunos consomem o feedback em uma mesma maneira. Durante o experimento, alguns alunos reclamaram que nossa ferramenta não deveria fornecer mensagens de avisos sobre a qualidade de termos de uma única letra que denotam identificadores em declarações de loop.

Em suma, nós afirmamos que é possível ajustar a geração de mensagens de aviso de *IQCheck* para atender a esta demanda. No entanto, nossa proposta era fornecer o feedback automático como um aviso e não uma prescrição, o que significa que o aluno não deveria renomear obrigatoriamente todos os nomes de identificadores avisados por nossa ferramenta. Assim, consideramos este ajuste e outros como trabalhos futuros no Capítulo 8, pois acreditamos que é necessário avaliar as motivações para mudar parâmetros e fundamentá-los com estudos empíricos usando uma abordagem baseada em evidências.

6.3.6 Discussão

Para discutir estes resultados, nós devemos lembrar a questão de pesquisa que nos motivou a conduzir este estudo: Se estudantes acham que as mensagens de feedback de *IQCheck* são úteis em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas. Nós sumarizamos e computamos, para cada uma das perguntas do nosso questionário, o valor mediano de resposta para realizarmos o julgamento objetivo sobre a utilidade das mensagens de feedback de *IQCheck*. A partir da análise dos resultados, nós podemos afirmar que reunimos opiniões muito positivas, pois há um consenso entre os alunos que *IQCheck* é fácil de usar e seu feedback é útil para ajudá-los a melhorar a nomeação de identificadores.

Além disso, nós também reunimos algumas sugestões sobre como melhorar nossa ferramenta. Para esses dados, isso significa que, há evidências de que as mensagens de feedback automático e oportuno são úteis para alunos. Elas permitem que eles obtenham orientação personalizada de como melhorar a nomeação de identificadores de seus programas através de dicas e avisos sobre nomes de identificadores que não parecem apropriados.

Capítulo 7

Discussão

Neste Capítulo, nós resumimos a discussão sobre o feedback proposto visando a MNI (Melhoria da Nomeação de Identificadores) e as ideias que emergem desta pesquisa de mestrado. Descreveremos as observações teóricas e práticas de utilizar nossa proposta e as implicações pedagógicas provenientes disto. Além disso, destacaremos as limitações da solução proposta e algumas ameaças à validade de nossas conclusões.

Nesta dissertação, apresentamos nossa proposta de gerar feedback automático, visando fornecer feedback formativo sobre a avaliação da adequação dos nomes de identificadores. Nosso objetivo era fornecer, no momento oportuno, suporte e orientação adequada a alunos de modo a ajudá-los a trabalhar durante o processo de MNI de seus programas. Neste sentido, a orientação que propomos visa a incentivar alunos a refletirem sobre termos escolhidos por eles durante a nomeação de identificadores. Diante disso, nossa proposta pode enriquecer positivamente o fornecimento de feedback sobre programas produzidos por alunos ao responder descrições de problemas de programação.

Em nossos estudos, provemos o feedback de MNI para programas funcionalmente corretos. Desta forma, eles tinham que estar atendendo aos requisitos especificados pelo texto de especificação da tarefa problema. A partir de uma perspectiva diferente da observada em nossos estudos, o feedback de MNI também pode ser fornecido para programas funcionalmente incorretos. Neste caso, acreditamos que o feedback de MNI poderia dar ao aluno a oportunidade de entender melhor o porquê de seu código estar errado, através do exercício de renomear nomes de identificadores de seu código. Podendo, em consequência, ajudá-lo a aprimorar sua habilidade de nomeação de identificadores.

No entanto, a única limitação imposta para usar nossa proposta é que o programa esteja sintaticamente correto. Pois, ela se destina tão somente a enriquecer o feedback que é dado ao aluno que já é capaz de produzir um programa sintaticamente correto, promovendo-lhes orientação sobre como encontrar os léxicos que melhor documentem/informem como o seu software funciona. Assim, nossa proposta não se destina a se adequar ao aluno que ainda não é capaz de produzir um programa que segue a estrutura formal própria esperada pela gramática da linguagem de programação.

Com a intenção de fornecer feedback formativo e oportuno, nossa proposta pode ser aplicada em cursos de programação introdutória com instrutores centrados em ajudar alunos na escrita de programas com qualidade, por meio de sistemas de avaliação automatizado. Em LP1, nossa proposta destinou-se a melhorar a habilidade de escrita de nomes de identificadores nos programas produzidos por programadores novatos deste curso. Pois, de certa forma, os instrutores de LP1 esperam que esses alunos dominem a habilidade de escrita e leitura de programas de computador no processo de resolução de problemas.

Para viabilizar a geração e o fornecimento do feedback de MNI, nossa proposta usa o texto da descrição de um problema de programação proposto pelo instrutor. Isto é, ela não pretende impor sobre instrutores encargos adicionais, extensivos e exaustivos além das obrigações necessárias para produzir um exercício de programação. Em contrapartida, nossa proposta também impõe algumas considerações importantes.

Em primeiro lugar, nossa proposta espera que o texto da descrição do problema de programação seja bem escrito, completo e compreensível, de acordo com o instrutor que projetou esse problema. Pois, nossa proposta conta com este texto para obter o vocabulário de referência usado para encontrar nomes apropriados de identificadores de código para programas de estudantes e, em seguida, gerar e fornecer o feedback de MNI.

É importante notar que este vocabulário não é definitivo, isto é, pode ser expandido através da inclusão de novos termos que instrutores julguem relevantes para o problema de programação. Por exemplo, esse vocabulário poderia incluir "i" e "j" como termos válidos, já que são muito comuns de serem utilizados como nomes de identificadores em loops e matrizes. Ao consumir o feedback de MNI, o aluno terá a oportunidade de revisar as construções de nomes escritas por ele para denotar identificadores de seu código quando responde o problema de programação.

Além disso, o aluno pode sentir-se incentivado a fazer más escolhas de nomes de identificadores após receber o feedback de MNI. Isto pode acontecer por duas razões: (1) a mensagem de feedback de MNI que propusemos pode encorajar o aluno a usar um ou outro termo do texto da descrição do problema de programação que não faz sentido (e.g., para ser bem sucedido na avaliação de nossa proposta), e (2) nossa proposta não faz a distinção desses termos um com os outros. Na verdade, nossa proposta não pretendia fazer a distinção entre termos que são mais relevantes para denotar nomes de identificadores do que outros. Embora a solução que propusemos encontre nomes de identificadores de código que empregam termos do vocabulário de referência, ela não constata se estes termos representam conceitos de grande significado deste vocabulário. Pois, a estratégia adotada pela nossa proposta destinou-se tão somente a constatar a presença de termos do vocabulário de referência, não levando em consideração aspectos como o papel do identificador, o seu tipo de dados, ou a sua relação com outros identificadores que são sinônimos no programa.

Uma premissa desta pesquisa, e também do seu referencial teórico, é que o emprego de léxicos descritivos em construções de nomes de identificadores favorece, em parte significativa, a legibilidade do programa. Por este motivo, a estratégia adotada pela nossa proposta destinou-se tão somente a constatar o emprego desses léxicos em nomes de identificadores, constatando a presença de termos do vocabulário de referência, sem levar em consideração esses outros aspectos acima mencionados. Por este motivo, é possível que alguns programas produzidos pelos usuários de nossa proposta possam demandar um segundo ciclo de feedback com instrutores para identificar outros problemas nas construções de nomes de identificadores. Uma hipótese anedótica é que esses problemas sejam de natureza conceitual, semântica e circunstancial, que requerem uma inspeção humana manual mais aprofundada dos programas dos alunos.

Por fim, nossa proposta não se destina a se adequar ao que temos hoje: construções de nomes de identificadores em códigos com mistura de idiomas português e inglês. Naturalmente, o aluno pode misturar estes idiomas ao nomear identificadores de seus códigos, usando português e/ou inglês. Apesar disso, ao fornecer a geração do feedback de MNI, a nossa proposta faz uso da normalização textual com o intuito de mitigar esta ameaça, quando o idioma usado pelo instrutor tem, etimologicamente, uma origem que é comum ao idioma usado pelo aluno (e.g., português e inglês). Pois, a normalização textual permite

que nossa proposta uniformize léxicos derivados etimologicamente de uma mesma origem, transformando-os em uma forma única que é canônica.

Com o objetivo de propor o feedback de MNI, a trajetória desta pesquisa teve seu ponto de partida marcado pela investigação para demonstrar que podemos automatizar a avaliação sobre a adequação dos nomes de identificadores de código.

Inicialmente, nós conduzimos um estudo com instrutores para coletar e examinar a forma como eles avaliam a adequação dos nomes de identificadores. A partir desse estudo, nós observamos que os nomes de identificadores julgados como apropriados pelos instrutores apresentaram, dentre outras características, o uso de termos da descrição do problema de programação em suas construções. Em sequência, nós conduzimos uma outra investigação para examinar até que ponto a avaliação automática proposta se assemelha à avaliação humana ao encontrar nomes apropriados de identificadores. Percebemos que nossa proposta apresentou resultados muito positivos, pois ela consegue acertar, em grande parte, o julgamento da avaliação dos instrutores. Diante disso, afirmamos que podemos usar as palavras obtidas do texto de especificação do problema como a referência para avaliar a qualidade dos nomes de identificadores de código. Ao utilizar a descrição do problema, nossa proposta pode ajudar alunos a melhorar a escrita dos nomes de identificadores e, em consequência, melhorar a legibilidade de seus programas.

Como resultado prático, podemos afirmar que conseguimos usar a descrição de problemas de programação para encontrar nomes apropriados de identificadores e, em consequência, encontrar nomes considerados inapropriados para compor o feedback de MNI. Na Tabela 7.1, sumarizamos os estudos conduzidos para avaliar a aptidão de usar a descrição de problemas de programação para gerar o feedback automático sobre a adequação dos nomes de identificadores.

Tabela 7.1: C1 - Nós geramos feedback automático sobre avaliação da adequação dos nomes de identificadores baseado na descrição do problema de programação

Acrônimo	**Descrição**
O1	Proposta do uso da descrição do problema de programação para encontrar nomes apropriados de identificadores de código.
QP1	Como a avaliação humana julga a adequação dos nomes de identificadores de código, tendo em vista a legibilidade?
O2	Avaliação do contraste entre a avaliação de qualidade automática e a avaliação humana.
QP2	A avaliação de qualidade automática, em certa medida, se assemelha à avaliação humana ao julgar nomes apropriados de identificadores de código?

Motivados pelos resultados encontrados, nós projetamos e implementamos *IQCheck*, um protótipo da nossa proposta para viabilizar a geração e fornecimento do feedback de MNI usando descrições de problemas de programação. Com o apoio de *IQCheck*, nós conduzimos um experimento controlado com dois grupos de estudantes, onde um dos grupos foi introduzido o seu uso. Apesar de termos motivado os alunos de ambos os grupos para MNI, o resultado do estudo mostrou que apenas os usuários de *IQCheck* apresentaram uma melhora significativa dos seus programas neste aspecto. Isso significa que a geração de feedback forneceu orientação personalizada e útil para que eles pudessem escolher nomes de melhor qualidade, melhorando assim a adequação dos identificadores de seus códigos. Diante disso, consideramos que temos reunido evidências suficientes para dar suporte a seguinte alegação: estudantes melhoram a adequação dos nomes de identificadores de seus programas com ajuda do feedback automático e oportuno. Na Tabela 7.2, mostramos resumidamente o estudo que conduzimos visando reunir evidências para dar suporte a esta afirmação.

Tabela 7.2: C2 - Estudantes melhoram a adequação dos nomes de identificadores de seus programas com ajuda do feedback automático e oportuno

Acrônimo	**Descrição**
O1	Avaliação do potencial das mensagens de feedback automático em ajudar alunos na melhoria da adequação dos nomes de identificdores.
QP1	As mensagens de feedback de *IQCheck* ajudam alunos a melhorar a adequação dos nomes de identificadores de seus códigos?

Finalmente, aplicamos um questionário perguntando aos usuários de *IQCheck* sobre sua utilidade em fornecer ajuda na MNI. Este estudo tinha como objetivo investigar, elucidar padrões de uso da nossa proposta e opiniões pós-feedback dos alunos que usaram *IQCheck* para melhorar os nomes de identificadores escolhidos por eles. Observamos que, embora muitos estudantes tenham feito uso de *IQCheck*, poucos deles mostraram envolvimento completo com o ciclo previamente definido durante o período de observação do estudo: receber o feedback, consumi-lo, e enviar uma outra versão do código, independentemente de renomear seus nomes de identificadores. Em consequência, pudemos contar, em nossas análises, apenas com os programas produzidos pelos alunos que estavam completamente comprometidos com este ciclo, dando atenção ao feedback de MNI. Isto significa que, embora nossa proposta seja útil, ela pode contribuir pouco para o treinamento de alunos que não pretendem investir tempo e esforço em MNI [Nar08].

De acordo com os dados reunidos e nossas observações no estudo reportado, alguns alunos podem não ter mostrado MNI devido ao fato de não terem investido tempo e esforço para consumir o feedback automático. Chinn e Brewer (1993), em sua lista de fatores que fazem o efeito do feedback pequeno, aponta este possível comportamento pós-feedback como: ignorar o feedback [CB93]. No entanto, no contexto desta pesquisa, esta é uma ameaça menor, pois uma pequena fração dos alunos chegaram a ignorar nossa proposta. Em alguns casos, alunos também podem não ter conseguido usar *IQCheck* por falta de tempo, pois podem não ter conseguido implementar, para uma dada tarefa de programação, o primeiro programa funcionalmente correto. Inicialmente, esperávamos este comportamento devido ao fato de nossa amostra apresentar alunos que poderiam encontrar dificuldades para implementar programas em respostas às tarefas de programação propostas.

Dentre os alunos que aderiram o processo de MNI, também percebemos outros dois possíveis comportamentos pós-feedback: alguns alunos rejeitaram o feedback ou julgaram que ele era irrelevante [CB93]. Apesar deles serem dois fatores diferentes, eles produzem o mesmo resultado: alunos recebem o feedback de MNI mas não o leva em consideração. Apesar de diminuírem drasticamente a MNI dos programas dos alunos, consideramos que esses comportamentos produzem resultados positivos em alguns casos, pois fornecemos as mensagens de feedback de *IQCheck* como um aviso e não uma prescrição. Neste sentido, muitos alunos poderiam rejeitar algumas mensagens por achar que elas forneciam avisos irrelevantes e, em consequência, poderiam não chegar a obter a mensagem que indica que o programa não contém nenhum aviso sobre a adequação dos nomes de identificadores: "Nenhuma advertência. Parabéns!". Inicialmente, esperávamos que alunos pudessem rejeitar mensagens de feedback de *IQCheck* sobre a adequação de nomes de identificadores de letra única em declarações de loop. Chegamos a esta conclusão, pois é usual usar letras únicas para denotar identificadores em declarações de loop e, na verdade, instrutores usualmente o fazem. Assim, alunos poderiam rejeitar o feedback automático por meio da reinterpretação do aviso apresentado por ele [CB93], no sentido de que, neste caso, o feedback de MNI fornecido por nossa proposta diverge do comportamento de instrutores.

Por fim, também observamos que alguns alunos fizeram poucas renomeações de nomes de identificadores, não mostrando melhora relevante de MNI. Nós observamos isso em situações em que alunos aderiram o processo de MNI, mas não eliminaram muitos avisos de qualidade, renomeando muitos identificadores e alcançando a mensagem "Nenhuma advertência. Parabéns!". Durante o período de observação do experimento, observamos que, dentre os 29 usuários de *IQCheck*, 15 (51,7%) deles foram capazes de alcançar esta mensagem. Além de ter consumido o feedback de *IQCheck*, eles também alcançaram a mensagem "Nenhuma advertência. Parabéns!". Considerando apenas os alunos que não alcançaram esta mensagem, três apresentaram algum programa contendo avisos destacados pelo feedback de *IQCheck*. Curiosamente, esses alunos tiveram a proficiência deles em usar *IQCheck* categorizada como "usuário". Observamos que eles tinham usado nossa proposta para obter feedback em programas produzidos por eles em resposta a, pelo menos, três problemas de programação. Além disso, encontramos que dois desses alunos não apresentaram MNI em seus programas. Na prática, isso significa que, esses alunos não se engajaram em remo-

ver todos os avisos apontados pelo feedback de MNI porque eles eram bons. Isto é, eles tinham produzido programas com bons nomes de identificadores e tinham entendido como usar nossa proposta. Analisando manualmente as duas versões do programa apresentadas por esses alunos, observamos que eles fizeram poucas renomeações de nomes de identificadores, apresentando pouca MNI. As mudanças realizadas por esses alunos incluem a reescrita de um ou outro nome de identificador, mudando ou incluindo léxicos em sua construção.

É importante destacar que alguns dos não-usuários de *IQCheck* também eram bons. Observamos que, durante o período de observação do experimento, dentre os 29 não-usuários de *IQCheck*, 3 (10,3%) deles foram capazes de apresentar alguma MNI. Na prática, isso significa que esses alunos mostraram alguma MNI porque eles tinham produzidos programas com bons nomes de identificadores, encontrando um ou outro que podia ser rescrito mudando ou incluindo léxicos em sua construção sem ajuda da nossa proposta. Analisando manualmente as duas versões do programa apresentadas por esses alunos, observamos que, eles também fizeram poucas renomeações de nomes de identificadores. As mudanças realizadas por esses alunos também incluem a reescrita de um ou outro nome de identificador. Apesar de não terem usado nossa proposta para obter feedback em programas produzidos por eles, eles também puderam melhorar a qualidade dos nomes de identificadores de seus códigos. Acreditamos que os demais não-usuários de *IQCheck*, em contraste a esses alunos, podem não ter mostrado melhoria de MNI porque eles precisavam de orientação personalizada sobre como melhorar a escrita dos nomes de identificadores de seus programas.

Concluímos que a utilização de nossa proposta, através de *IQCheck*, é positiva, pois seus usuários efetivamente foram ajudados na MNI. Os usuários da nossa proposta apresentaram duas versões do mesmo programa, onde a última apresentou uma redução de mensagens de avisos. Considerando unicamente os alunos que aderiam a MNI, essa redução de avisos ocorre independentemente do aluno ter sua proficiência em usar *IQCheck* categorizada como "usuário" ou "testador". Realizamos uma última investigação para contrastar os alunos que tiveram sua proficiência categorizada como "testador" e "usuário" e testemunhamos que a MNI tende a ser maior entre os programas produzidos por este primeiro do que por este último.

Usamos o teste não paramétrico dos postos sinalizados de Wilcoxon para comparar as distribuições de valores de Δ_{tnai} dessas duas categorias de usuários. Como resultado, con-

firmamos que as distribuições de valores de Δ_{tnai} das categorias diferem em suas medianas, pois os valores de Δ_{tnai} mostrados pela primeira eram maiores, em média, do que os mostrados pela última com significância estatística adequada (p-valor = 0,02097, em um nível de significância de 0,05). Isto significa que os programas produzidos pelos usuários de *IQCheck* categorizados como "testador" mostrou uma maior diferença entre os nomes de identificadores do primeiro e último programa correto. Apesar disso, os usuários desta categoria apresentaram, em média, um número mediano menor de tarefas de programação respondidas (2) em comparação aos usuários da outra categoria (3,5). Também, usuários de *IQCheck* com sua proficiência categorizada como "testador" foram os que menos fizeram uso de nossa proposta, já que apresentaram, em média, um número mediano de 2 ocorrências de uso de nossa ferramenta em suas atividades. Este número contrasta com o número de usuários de *IQCheck* com sua proficiência categorizada como "usuário", já que apresentaram um número mediano de 4 ocorrências de uso de nossa ferramenta para obter feedback ao responder suas tarefas de programação.

Ao perguntar sobre a utilidade da nossa proposta, percebemos que há um consenso entre a maioria dos usuários da nossa ferramenta, no sentido de facilidade de uso e utilidade em fornecer ajuda na MNI. Além disso, descobrimos alguns problemas no uso de *IQCheck*. Aparentemente, alguns alunos não apresentam senso crítico sobre as mensagens de feedback de *IQCheck*, preocupando-se em eliminar todos os avisos destacados por elas. Estes alunos parecem acreditar que os avisos mostrados nas mensagens de feedback de *IQCheck* devem ser seguidos à risca, como numa "receita médica".

Em suma, podemos concluir à luz dos resultados que nossa proposta pode ajudar provadamente alunos na MNI. Nós consideramos esse resultado positivo, uma vez que o uso da nossa proposta dá a oportunidade de ajudar a melhorar o treinamento de programadores novatos. Em resumo, a Tabela 7.3 mostra o estudo que dirigimos para dar suporte a afirmação de que: estudantes acham que o feedback automático e oportuno é útil em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas.

Tabela 7.3: C3 - Estudantes acham que o feedback automático e oportuno é útil em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas.

Acrônimo	**Descrição**
O1	Avaliação da opinião de alunos sobre a utilidade das mensagens de feedback automático em prover ajuda na MNI.
QP1	Estudantes acham que as mensagens de feedback de *IQCheck* são úteis em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas?

7.1 Sobre a Sugestão de Nomes de Identificadores

Ao invés de fornecer o feedback de MNI, *IQCheck* poderia ter sido implementada para sugerir/recomendar bons nomes de identificadores para um dado problema de programação. Desta forma, nossa proposta poderia funcionar como um corretor gramatical integrado à IDE (do inglês, *Integrated Development Environment*) do aluno, esquadrinhando e destacando nomes inapropriados de identificadores em seu código. No entanto, é preciso considerar que há custos que são impostos sobre o aluno ao fornecê-lo essas sugestões. Pois, ao aceitar/recusar essas sugestões, podemos acabar colocando-o em uma posição confortável e passiva em seu processo de aprendizagem. Koedinger e Aleven (2007) usaram o termo "dilema da assistência" para se referir ao problema sobre "como ambientes de aprendizagem deveriam balancear o fornecimento e retenção de informação ou assistência para alcançar a aprendizagem ótima do estudante" [KA07].

De acordo com a lista de Koedinger e Aleven (2007), sobre os benefícios e custos do fornecimento e retenção de informações, consideramos que obtemos como benefícios o fornecimento de sugestões/recomendações de bons nomes de identificadores mais precisas, eficientes em sua comunicação e com emoção de sucesso do aluno (com ajuda). Em contrapartida, consideramos que obtemos como custos o risco dessas sugestões serem processadas de forma superficial pelo aluno, com falta de sua atenção, sem envolvimento de sua memória de longo prazo e retirando a sua chance de brilhar. Por outro lado, há muitos benefícios ao utilizar a nossa proposta. Ainda de acordo com a lista de de Koedinger e Aleven (2007), quando fornecemos o feedback de MNI, retendo as sugestões/recomendações sobre bons nomes de

identificadores, consideramos que obtemos como benefícios o fornecimento de instruções com geração de efeito, forçando a atenção, envolvendo o uso de memória de longo prazo e com emoção de sucesso do aluno (sem ajuda). Assim, damos ao aluno a oportunidade de refletir criticamente sobre a qualidade dos nomes de seus programas, pois, de certa forma, retemos sugestões para esses nomes sem "entregar o ouro".

Por este motivo, nossa proposta não se destinou a ser utilizada com o objetivo de fornecer ao aluno exemplos de bons nomes de identificadores ao responder um dado problema de programação. Além disso, observamos muitos benefícios quando fornecemos o feedback de MNI ao aluno com o *IQCheck* durante o experimento. Por exemplo, podemos citar a oportunidade que os alunos tiveram de pensar sobre a adequação de um nome, em comparação com outro, a ser usado em uma circunstância específica de seu código. Como o aluno foi impelido a refletir e melhorar os léxicos dos nomes de identificadores de seu programa, em relação a um determinado problema, então ele acabou ultrapassando os limites do seu conhecimento. Em contrapartida, consideramos que obtemos como custos os erros do aluno, seus tropeços ao fazer a reescrita dos nomes de seu programa, a sua confusão durante o processo de MNI, e sua frustração à suas falhas.

É importante destacar que nossa proposta não é uma solução única para todos os contextos. Deste modo, ela pode ser adaptada para fornecer as sugestões discutidas nesta seção após levar em consideração a sua maturidade e a abordagem pedagógica do curso. A maturidade da nossa proposta pode ser examinada após avaliar o seu uso em outros cursos de programação seguindo vários ciclos de evolução. Além disso, as decisões sobre como será a evolução da nossa proposta precisam estar baseadas na opinião de instrutores e abordagem pedagógica adotada nesses cursos.

7.2 Sobre as Implicações Pedagógicas e Oportunidades

Nesta seção, nós destacamos algumas implicações pedagógicas e oportunidades provenientes do uso da nossa proposta.

7.2.1 Aperfeiçoamento da Nomeação de Identificadores

Nossa proposta pode ajudar a melhorar o treinamento de programadores novatos em educação de computação ajudando-os a melhorar a nomeação de identificadores. Ao fornecer o feedback automático, nós damos ao aluno a oportunidade de refletir criticamente sobre os nomes de identificadores escolhidos por ele que não parecem apropriados ao responder a descrição do problema de programação. Além disso, damos a ele a oportunidade de melhorar a adequação desses nomes de identificadores, aconselhando-o a usar léxicos que podem melhorar a legibilidade de seu programa. Em consequência, espera-se que o aluno melhore a habilidade de nomeação de identificadores dele, escolhendo melhor os léxicos que ele usa na construção dos nomes de identificadores nas próximas experiências de codificação.

7.2.2 Reflexão Crítica Sobre a Nomeação de Identificadores

Através do pensamento crítico sobre a qualidade dos nomes de identificadores de seu código, é dado ao aluno a chance de tornar-se ativo no seu processo de aprendizagem. Na verdade, a oportunidade de refletir criticamente sobre a qualidade do seu próprio programa durante, ou até mesmo após, a sua produção pode contribuir no processo da auto-regulação da aprendizagem do aluno [SBCP11]. Ao considerar a melhoria dos nomes de identificadores, os alunos podem repensar o processo adotado para nomeação de identificadores de seus programas. Por sua vez, esta auto-avaliação irá contribuir para a auto-regulação do processo de aprendizagem do aluno [TWV15].

7.2.3 Aprendizagem de Boa Prática de Programação

Nossa proposta aconselha o aluno sobre quais nomes de identificadores de seu código são considerados inapropriados, segundo a linguagem da descrição do problema de programação. Neste sentido, o feedback de MNI oferece ao aluno a oportunidade de melhorar a adequação desses nomes usando termos desta especificação. Este estilo de nomeação de identificadores também é uma prática recomendada no design de software orientado ao domínio ou DDD (no inglês, *Domain-Driven Design*) [ES15].

O feedback de MNI assemelha-se à recomendação de DDD, no sentido de que esta última sugere que a equipe de desenvolvimento use no código uma linguagem que seja própria e

ubíqua do domínio do negócio. Assim, é esperado que desenvolvedores usem palavras que são próprias desta linguagem para construir nomes de identificadores do software, no sentido de associar o contexto do problema com a solução no código. Por exemplo, a solução em código executável relacionada à regra de negócio "temos que emitir a fatura para o cliente antes da data limite" poderia incluir as seguintes construções de nomes de identificadores "cliente", "fatura", "emitir" e "data_limite" [DDD10].

Diante disto, ao fornecer o feedback de MNI, estaremos dando ao aluno a chance de reler o texto da tarefa de programação para dar-lhe uma maior atenção, no sentido de encontrar nomes relevantes para serem usados como nomes de identificadores de modo a resolver um dado problema. Ao mesmo tempo, o aluno vai ter a chance de aprender na prática um bom estilo de programação desde suas primeiras experiências de codificação.

7.2.4 Discussão Sobre Adequação de Nomes de Identificadores

Por fim, nossa proposta também dá ao aluno a chace de discutir criticamente com o instrutor ou colegas sobre a adequação de nomes de identificadores. Em nossos estudos, percebemos que alguns alunos sentiram-se encorajados a discutir com os pesquisadores sobre qual era o nome mais apropriado para ser utilizado como identificador em uma circunstância ou outra do seu código. Nós chegamos a esta conclusão pois cerca de três alunos chegaram a interagir com os aplicadores do experimento controlado, chamando a atenção para uma pequena conversa sobre os nomes de identificadores de seus códigos. Esse momento de interação pesquisador-aluno trouxe uma experiência enriquecedora para o segundo, no sentido de ajudá-lo a entender qual é o papel de nomes apropriados de identificadores no desenvolvimento de seu código com qualidade.

7.3 Ameaças à Validade

É importante destacar que usamos uma descrição de problema de programação para conduzir os estudos que avaliam a aptidão de usar seu texto para gerar o feedback de MNI. Afirmamos que a escolha foi suficiente para conduzirmos os estudos no sentido de que os resultados obtidos sugerem que podemos confirmar a validade da conjectura/inspiração desta pesquisa. Com base em uma única descrição de problema de programação, os resultados obtidos em

nossos estudos sugerem que podemos constatar que grande parte dos nomes apropriados de identificadores são construídos com seus termos.

Temos que considerar que não conseguimos assegurar a validade externa da aptidão de usar esse mesmo texto para encontrar bons nomes de identificadores de código, já que levamos em consideração avaliações humanas produzidas pelos próprios instrutores de LP1. Assim, esperamos que este trabalho seja replicado em cursos de outras instituições, com outros especialistas, para que haja uma generalização dos resultados. Inicialmente, consideramos a replicação deste trabalho no contexto de cursos de CS2 (do inglês, *Computer Science 2*) como um trabalho futuro no Capítulo 8.

Uma possível ameaça é acreditar que os textos que usamos são bem escritos devido os fatores humanos. Mas, defendemos que esta ameaça é mitigada, já que usamos textos que tradicionalmente são propostos pelos instrutores de LP1 como tarefas de programação. Acreditamos que esses textos tenham sido corrigidos pelos instrutores na hipótese de reclamação de alunos sobre a escrita, completude e compreensibilidade.

É importante observar que não avaliamos os nomes de identificadores em programas produzidos por usuários e não-usuários de *IQCheck* em relação a um oráculo de nomes apropriados, que é construído de forma cega pelos instrutores de LP1, sem saber quais eram os usuários e não-usuários de *IQCheck*. Pois, a premissa da nossa proposta é que quanto mais os nomes de identificadores do código do aluno refletem os termos da descrição do problema de programação, melhor será a sua qualidade. Propusemos esta premissa levando em consideração as evidências que fundamentaram C1 e as recomendações da literatura de engenharia de software.

Além disso, buscamos comparar os usuários e não-usuários de *IQCheck* com base no que se tem hoje em dia em termos de fornecer feedback sobre os léxicos escolhidos pelo aluno em nomes de identificadores. Previamente, fornecemos aos usuários e não-usuários de *IQCheck* uma explicação sobre o que seria o experimento, alertando e avisando que os nomes dos seus programas poderiam ser avaliados. Adicionalmente, previamente entregamos aos alunos documentos escritos que forneceram instruções detalhadas sobre como eles poderiam melhorar a legibilidade de seus programas através da escrita de bons nomes de identificadores. Uma possível ameaça é acreditar que o fator ou "placebo" de alertar/avisar/pedir que não-usuários de *IQCheck* melhorassem a escrita dos identificadores, seguindo nossas orien-

tações verbais e escritas, pode não ser suficiente, já que eles não estavam sendo expostos a mensagens de feedback de qualquer natureza. Assim, não conseguimos assegurar que as renomeações de identificadores ocorreram somente pelo fato dos usuários de *IQCheck* terem sido submetidos às suas mensagens de feedback automático.

Temos que enfatizar que tínhamos a intenção de examinar as opiniões registradas pelos usuários de *IQCheck* que de fato investiram tempo em MNI, recebendo e consumindo o feedback de MNI. Desta forma, acabamos excluindo a opinião daqueles que não conseguiram efetivamente usar a nossa ferramenta. Isto é, usuários de *IQCheck* que não enviaram duas versões do mesmo programa e/ou requisitaram seu feedback. Por este motivo, não há como garantir que as opiniões analisadas retratem todos os pontos de vista possíveis sobre a utilidade da nossa proposta. Enfatizamos que a exclusão da opinião dos demais usuários de *IQCheck* não representa uma ameaça à validade de nossas conclusões. Pois, a análise de todas as opiniões mostrou, para todas as perguntas do nosso questionário, valores medianos de resposta similares aos obtidos das opiniões reportadas em nosso estudo.

Mas, consideramos a necessidade de investigar agora, para reforçarmos os resultados ou a significância do nosso relato, o porquê de alguns usuários de *IQCheck* não terem respondido o *survey* e/ou se engajado no seu ciclo. Uma hipótese anedótica é que esses fatores não signifiquem que nossa proposta é difícil de usar. Acreditamos que eles podem significar que alguns usuários de *IQCheck* podem não ter conseguido responder pelo menos uma das tarefas de programação, produzindo o primeiro programa correto necessário para dar início ao ciclo de feedback de *IQCheck*. Chegamos a esta hipótese porque apenas 2 (18,2%) das 11 opiniões analisadas eram referentes aos usuários de *IQCheck* que conseguiram responder uma única tarefa de programação.

Capítulo 8

Conclusões

Neste trabalho, nós propusemos a geração e fornecimento de feedback para melhorar a nomeação de identificadores de alunos. Nosso desafio foi entregar feedback elaborado para ajudar o aluno a escolher nomes mais apropriados para denotar identificadores do seu código, de modo a melhorar sua adequação ao responder a descrição do problema de programação. Para atender a este objetivo, nós propusemos uma estratégia para gerar e fornecer automaticamente o feedback usando descrições de problemas de programação, que já haviam sido produzidas pelos professores, para mitigar o esforço manual imposto a eles.

Nós focamos em fornecer o feedback sobre avaliação da adequação dos nomes de identificadores para programas funcionalmente corretos. Isto é, em nossos estudos, avaliamos o potencial da nossa proposta tendo como base os programas que normalmente seriam manualmente inspecionados pelos instrutores, com o propósito de encontrar nomes de identificadores considerados inadequados e passíveis de renomeação em prol de sua melhoria. A principal contribuição desta dissertação de mestrado baseia-se em lições apreendidas com a nossa proposta ao gerar e fornecer o feedback automático sobre programas produzidos por alunos de um curso de programação introdutória. Tendo como base este curso, nós conduzimos estudos empíricos de pesquisa, como estudo exploratório, estudo de caso descritivo, experimento, e *survey*, com o objetivo de lançar luz sobre evidências para apoiar as seguintes afirmações:

- *Nós geramos feedback automático sobre avaliação da adequação dos nomes de identificadores baseado na descrição do problema de programação;*

- *Estudantes melhoram a adequação dos nomes de identificadores de seus programas com ajuda do feedback automático e oportuno;*

- *Estudantes acham que o feedback automático e oportuno é útil em fornecer ajuda na escrita de nomes apropriados de identificadores de seus programas.*

Com o objetivo de avaliar o potencial da nossa proposta, nós investigamos a eficácia de usar *IQCheck* - a ferramenta prova de conceito implementada para instanciar concretamente a nossa proposta - para gerar e fornecer mensagens de feedback automático, de modo a ajudar alunos a melhorar a adequação dos nomes de identificadores no seu código final. Em suma, percebemos que é possível dar dicas úteis a alunos, através do fornecimento do feedback automático, durante a escrita dos nomes de identificadores de seus programas.

Além disso, testemunhamos que os usuários da nossa proposta sentiram-se apoiados de maneira personalizada no processo de melhoria da qualidade dos nomes de identificadores. Como resultado, eles acabaram sendo motivados a fornecer opiniões positivas sobre a utilidade da nossa proposta. A partir das opiniões e comentários, nós percebemos que alguns alunos não estavam consumindo o feedback que propusemos como esperávamos inicialmente.

8.1 Trabalhos Futuros

Esta pesquisa de mestrado traz a oportunidade de formular novas pesquisas para serem conduzidas em trabalhos futuros. Com o objetivo de dar continuidade à esta pesquisa de mestrado, novas pesquisas podem investir em direções específicas com o propósito de melhorar nosso trabalho. Outras, por outro lado, para conduzir novos estudos empíricos para investigar como o feedback que propusemos pode ajudar alunos em outras circunstâncias de seu código. Algumas sugestões para trabalhos futuros são:

- Investigação longitudinal sobre o potencial da nossa proposta. A ideia é examinar se os usuários da nossa proposta podem melhorar os nomes de seus programas dentro do ambiente de ensino-aprendizagem em longo prazo;

- Avaliação do uso da heurística atual (vocabulário de referência), incluindo termos válidos e/ou comuns para loops e matrizes, no *IQCheck* para fornecer o feedback automático sobre nomes inapropriados de identificadores (e.g., "i", "j", "k" e "l"). Além disso,

a inclusão da heurística número de caracteres, com o intuito de gerar e fornecer uma mensagem específica para nomes contendo um caractere único (e.g., "g" não parece ser um nome adequado pois é pequeno);

- Arguição sobre como podemos usar a estrutura do código do aluno, ou os nomes de identificadores do código de referência, para melhorar a adequação dos nomes de identificadores do código do aluno;

- Avaliação do uso da heurística atual (vocabulário de referência) para fornecer feedback automático sobre comentários em códigos escritos pelos alunos;

- Pesquisa sobre como podemos usar nossa proposta no contexto de cursos de CS2, os quais envolvem especificações relativamente longas de projetos de programação;

- Investigação sobre como nossa proposta pode ajudar alunos a se livrar de obstáculos encontrados no processo de produzir um programa funcionalmente correto.

Bibliografia

[A+17] Eliane Cristina de ARAÚJO et al. *Automatização de feedback para apoiar o aprendizado no processo de resolução de problemas de programação.* PhD thesis, Federal University of Campina Grande, Campina Grande, PB, BRAZIL, September 2017. [Online; accessed 8-Fevereiro-2019].

[AAT+12] S. L. Abebe, V. Arnaoudova, P. Tonella, G. Antoniol, and Y. G. Guéhéneuc. Can lexicon bad smells improve fault prediction? In *2012 19th Working Conference on Reverse Engineering*, pages 235–244, Oct 2012.

[ACC+02] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, October 2002.

[AM05] Kirsti M Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer science education*, 15(2):83–102, 2005.

[AMUJ04] Kirsti Ala-Mutka, Toni Uimonen, and Hannu-Matti Jarvinen. Supporting students in c++ programming courses with automatic program style assessment. *Journal of Information Technology Education: Research*, 3(1):245–262, 2004.

[ASF16] E. Araujo, D. Serey, and J. Figueiredo. Qualitative aspects of students' programs: Can we make them measurable? In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, Oct 2016.

[AST90] AST. Abstract Syntax Trees. https://docs.python.org/2/library/ast.html, 1990. [Online; accessed 26-January-2019].

[BBL76] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[Bec03] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[But09] Simon Butler. The effect of identifier naming on source code readability and quality. In *Proceedings of the Doctoral Symposium for ESEC/FSE on Doctoral Symposium*, ESEC/FSE Doctoral Symposium '09, pages 33–34, New York, NY, USA, 2009. ACM.

[BW08] Raymond P.L. Buse and Westley R. Weimer. A metric for software readability. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ISSTA '08, pages 121–130, New York, NY, USA, 2008. ACM.

[BW10] Raymond PL Buse and Westley R Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2010.

[BWYS09] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp. Relating identifier naming flaws and code quality: An empirical study. In *2009 16th Working Conference on Reverse Engineering*, pages 31–35, Oct 2009.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. Modern information retrieval. addison-wesley. 1999.

[CB93] Clark A Chinn and William F Brewer. The role of anomalous data in knowledge acquisition: A theoretical framework and implications for science instruction. *Review of educational research*, 63(1):1–49, 1993.

[CCK11] Daniela Chudá, Ján Chlpek, and Andrej Kumor. The impact of text preprocessing to determine the similarity in students assignments. In *Proceedings of the 12th International Conference on Computer Systems and Technologies*, CompSysTech '11, pages 417–422, New York, NY, USA, 2011. ACM.

[Che01] CheckStyle. http://checkstyle.sourceforge.net/, 2001. [Online; accessed 1-June-2019].

[CT00] Bruno Caprile and Paolo Tonella. Restructuring program identifier names. In *icsm*, pages 97–107, 2000.

[DDD10] DDD – Introdução a Domain Driven Design. http://http://www.agileandart.com/2010/07/16/ddd-introducao-a-domain-driven-design/, 2010. [Online; accessed 15-Out-2019].

[Dix01] Philip M Dixon. The bootstrap and the jackknife. *Design and analysis of ecological experiments*, pages 267–288, 2001.

[DLDPO11] Andrea De Lucia, Massimiliano Di Penta, and Rocco Oliveto. Improving source code lexicon via traceability and information retrieval. *IEEE Trans. Softw. Eng.*, 37(2):205–227, March 2011.

[DLO05] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5(3):4, 2005.

[Edw03] Stephen H Edwards. Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing (JERIC)*, 3(3):1, 2003.

[Efr87] Bradley Efron. Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82(397):171–185, 1987.

[ES15] Eric Evans and Rafał Szpoton. *Domain-Driven Design*. Helion, 2015.

[FBY92] William B Frakes and Ricardo Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice Hall PTR, 1992.

[GA86] Martin J Gardner and Douglas G Altman. Confidence intervals rather than p values: estimation rather than hypothesis testing. *Br Med J (Clin Res Ed)*, 292(6522):746–750, 1986.

[GFSM15] Elena L. Glassman, Lyla Fischer, Jeremy Scott, and Robert C. Miller. Foobaz: Variable name feedback for student code at scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 609–617, New York, NY, USA, 2015. ACM.

[HM08] Sonia Haiduc and Andrian Marcus. On the use of domain terms in source code. In *2008 16th IEEE International Conference on Program Comprehension*, pages 113–122. IEEE, 2008.

[IAKS10] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli calling international conference on computing education research*, pages 86–93. ACM, 2010.

[JU97] David Jackson and Michelle Usher. Grading student programs using assyst. In *ACM SIGCSE Bulletin*, volume 29, pages 335–339. ACM, 1997.

[KA07] Kenneth R Koedinger and Vincent Aleven. Exploring the assistance dilemma in experiments with cognitive tutors. *Educational Psychology Review*, 19(3):239–264, 2007.

[KJH16] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 41–46, New York, NY, USA, 2016. ACM.

[LFB06] D. J. Lawrie, H. Feild, and D. Binkley. Leveraged quality assessment using information retrieval techniques. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 149–158, 2006.

[LFOT07] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(4):13, 2007.

[Lik32] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[LMFB06] D. Lawrie, C. Morrell, H. Feild, and D. Binkley. What's in a name? a study of identifiers. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 3–12, June 2006.

[LRBC+17] Andrew Luxton-Reilly, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. Developing assessments to determine mastery of programming fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, ITiCSE-WGR '17, pages 47–69, New York, NY, USA, 2017. ACM.

[LVNT15] Ximena López, Jorge Valenzuela, Miguel Nussbaum, and Chin-Chung Tsai. Some recommendations for the reporting of quantitative studies. *Computers & Education*, 91(C):106–110, 2015.

[Mar08] Robert C Martin. *Código Limpo*. Elsevier/Alta Books, 2008.

[May08] Richard E Mayer. Applying the science of learning: Evidence-based principles for the design of multimedia instruction. *American psychologist*, 63(8):760, 2008.

[Nar08] Susanne Narciss. Feedback strategies for interactive learning tasks. *Handbook of research on educational communications and technology*, 3:125–144, 2008.

[NGV10] Kevin A Naudé, Jean H Greyling, and Dieter Vogts. Marking student programs using graph similarity. *Computers & Education*, 54(2):545–561, 2010.

[NLT17] NLTK. Natural Language Toolkit. http://www.nltk.org/, 2017. [Online; accessed 26-January-2019].

[Pen87a] Nancy Pennington. Empirical studies of programmers: Second workshop. In Gary M. Olson, Sylvia Sheppard, and Elliot Soloway, editors, *Empirical*

Studies of Programmers: Second Workshop, pages 100–113, Norwood, NJ, USA, 1987. Ablex Publishing Corp.

[Pen87b] Nancy Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology*, 19(3):295–341, 1987.

[PM07] Denys Poshyvanyk and Andrian Marcus. Using traceability links to assess and maintain the quality of software documentation. *Proc. of TEFSE*, 7:27–30, 2007.

[PMD01] PMD an extensible cross-language static code analyzer. http://pmd.sourceforge.net/, 2001. [Online; accessed 1-June-2019].

[RE90] RE. Regular Expression Operations. https://docs.python.org/2/library/re.html, 1990. [Online; accessed 28-March-2019].

[Ree82] Michael J Rees. Automatic assessment aids for pascal programs. *ACM Sigplan Notices*, 17(10):33–42, 1982.

[RSKPFVI14] Manuel Rubio-Sánchez, Päivi Kinnunen, Cristóbal Pareja-Flores, and Ángel Velázquez-Iturbide. Student perception and usage of an automated programming assessment tool. *Computers in Human Behavior*, 31:453–460, 2014.

[RSL01] RSLP. A NLTK Stemmer for Portuguese. https://www.nltk.org/modules/nltk/stem/rslp.html, 2001. [Online; accessed 6-April-2019].

[SBCP11] K A M Sambell, Elizabeth Barry-Cutter, and Nicholas R. Price. Rethinking feedback in higher education : an assessment for learning perspective ; learning to learn : learning to cook : the f-word, feedback. 2011.

[SHP+06] Jaime Spacco, David Hovemeyer, William Pugh, Fawzi Emad, Jeffrey K Hollingsworth, and Nelson Padua-Perez. Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. In *ACM Sigcse Bulletin*, volume 38, pages 13–17. ACM, 2006.

[Shu08] Valerie J Shute. Focus on formative feedback. *Review of educational research*, 78(1):153–189, 2008.

[Sne96] Harry M Sneed. Object-oriented cobol recycling. In *Proceedings of WCRE'96: 4rd Working Conference on Reverse Engineering*, pages 169–178. IEEE, 1996.

[Tok01] Tokenize. NLTK Tokenizer. https://www.nltk.org/api/nltk.tokenize.html, 2001. [Online; accessed 6-April-2019].

[TR97] Gareth Thorburn and Glenn Rowe. Pass: An automated system for program assessment. *Computers & Education*, 29(4):195–206, 1997.

[TWV15] Caroline F Timmers, Amber Walraven, and Bernard P Veldkamp. The effect of regulation feedback in a computer-based formative assessment on information problem solving. *Computers & education*, 87:1–9, 2015.

[Uni90] Unidecode. ASCII transliterations of Unicode text. https://pypi.org/project/Unidecode/, 1990. [Online; accessed 6-April-2019].

[Wil15] Chris Wilcox. The role of automation in undergraduate computer science education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 90–95, New York, NY, USA, 2015. ACM.

Apêndice A

Giving Automated Feedback About Student Code Identifiers: a Method Based on the Description of Programming Problem

Giving Automated Feedback About Student Code Identifiers: a Method Based on the Description of Programming Problem

Marcos Nascimento[1], **Eliane Araújo**[1], **Dalton Serey**[1], **Jorge Figueiredo**[1]

[1] Departamento de Sistemas e Computação
Universidade Federal de Campina Grande (UFCG) – Campina Grande, PB – Brasil.

{marcosantonio@copin,{eliane,dalton,abrantes}@computacao}.ufcg.edu.br

Abstract. *Providing timely feedback on identifier naming to novice programmers can help them to improve their program readability. However, due to the growth in the number of students learning to program nowadays, giving manual feedback on identifier quality become prohibitive. In this paper, we propose a method to automatically give this feedback which is correct 75.0% of the time in contrast to the instructors' assessment. We found that 51.7% of the students who received automated feedback showed their program identifier quality improvement by picking better names. It means that we can help students to improve identifier naming and consequently, their program readability from early coding experiences.*

1. Introduction

In computer programming, identifier naming is the process of choosing names to denote code identifiers, which are constructions used to refer to, for example, variables and methods. Consequently, giving feedback on identifier quality assessment to novice programmers can help them to improve a fundamental tenet of software quality: their program readability [Butler 2009]. However, it can be tedious, error-prone and inefficient for instructors to provide this feedback manually and in a timely manner. It can also become strictly prohibitive considering the growth in the number of students learning to program nowadays in programming courses [Wilcox 2015]. In this sense, generation automated feedback on identifier quality assessment is essential to aid instructors in giving personalized one-to-one feedback.

In this paper, we propose an innovative method to generate automated feedback. When answering the programming problem, we use the description provided by the instructor to assess the appropriateness of code identifiers chosen by students. The rationale behind this idea is simple: the description includes words which can be used to construct appropriate code identifiers so that they communicate fundamental concepts of the programming problem to the reader. In this sense, the more the code expresses those concepts, the easier to read, understand, and maintain. Identifier naming based on software specification, as we propose, is also a programming style recommended in software engineering literature studies intended to improve identifier quality [De Lucia et al. 2011, Lawrie et al. 2006, Evans and Szpoton 2015].

To provide automated feedback, we go through three steps. First, we use the description of the programming problem to create the terms we use as the reference to assess the appropriateness of code identifiers: the reference vocabulary. Second, we find all of the student's code identifiers constructed without the presence of using the terms

of the reference vocabulary to obtain names considered to be inappropriate. After that, we generate and provide feedback messages to the student, advising about names that could be renamed to improve software quality. We evaluated the aforementioned method conducting an in situ experiment in an introductory programming course to evaluate the effectiveness of providing automated feedback in the students' final code. We developed *IQCheck* (Identifier Quality Check) as a proof-of-concept tool that instantiates our proposal. The experiment results suggest that *IQCheck* feedback messages helps students to improve their program identifier quality. We witnessed that the students who used *IQCheck* tended to improve the appropriateness of their code identifiers, with 51.7% of the students showing better names after receiving feedback messages.

2. Related Work

There are many studies in the area of computer science education that evaluate tools to provide feedback on programming style. The works presented in these studies are related to ours regarding the intent to provide feedback on identifier quality assessment. ASSYST [Jackson and Usher 1997], CheckStyle [CheckStyle 2001], PMD [PMD 2002], STYLE [Rees 1982], Style++ [Ala-Mutka et al. 2004], among others, are tools similar to each other in the sense that they are based on the identifier name length, in characters, to provide feedback on inappropriate code identifiers. Our method differs from these tools in its methodology in the sense that to provide this feedback, it is based on the examination of student-written lexicons in code identifier constructions. Glassman et al.'s [Glassman et al. 2015] work is the closest to ours with the intent to provide feedback on inappropriate code identifiers examining their constructions. But Glassman et al.'s approach asks instructors to assess, through a user interface, the appropriateness of student-chosen identifiers based on the values those variables can take during the program execution. Our work is similar to the work presented in their study but differs in its methodology in the sense that our work aims to generate automated feedback using the description of the programming problem, without asking instructors to perform the assessment.

3. Automated Feedback Generation Method

To provide automated feedback, we go through three steps. First, we apply normalization on programming problem description to extract terms to compose a reference vocabulary. The idea is to use this reference to find appropriate code identifiers' names to student's programs. In other words, the words which could be chosen by a student to denote their code identifiers when answering the programming problem. The normalization process aims to create appropriate and relevant terms which could contribute to program readability improvement and comprehension in a bottom-up manner as the description may contain words with no relevant content or accented letters. As the first step of the normalization, we apply tokenization to the description in order to extract and obtain the lexicons denoting words. In sequence, we apply the transformation of words with capitalized and accented letters into others with non-capitalized and non-accented letters, respectively. After that, we apply the detection and removal of stop-words, which are words with little lexical content (e. g. prepositions, articles, numbers, and punctuation marks). Stop-words have no relevant or meaningful content to be written as identifiers within the code as they are unrelated to the programming problem concepts. Besides stop-words, we apply the

detection and removal of special characters, words formed by letters repeated, followed by numbers, and having a length less than two [Baeza-Yates and Ribeiro-Neto 1999]. Finally, we apply the stemming of resulting words converting plurals into singulars, transforming conjugated verb forms into infinitives, and removing suffixes from adjectives. We apply the stemming to these words as they may include plural words, conjugated verbs, and adjectives, which are inflected words to their stem. For example, "counter" and "counting" are two words inflected from the same stem, thus they would be transformed into the unique stem "count" without their suffixes "er" and "ing".

Second, we use the terms of the reference vocabulary to assess the appropriateness of student-written code identifiers when implementing their program for the original description. These are the steps of the process: Firstly, we apply the parsing of code to extract and obtain the names denoting student-chosen identifiers. In sequence, we remove duplicate names to apply the detection, division, stop-word removal of names composed of two or more terms, used to construct the identifier, separated by using the underscore or camel case separators. Secondly, we apply the stemming to resulting separated terms, or names, to obtain stems from inflected terms. Then, we find all of the identifiers constructed without the presence of using at least one term of the reference vocabulary to obtain names considered to be inappropriate. Finally, we generate and present warning messages to the student, advising which it could rename those names by better ones referencing the description of the programming problem.

4. Descriptive Case Study

In this section, we describe a study performed to investigate and demonstrate we can use the description of a programming problem for helping students to choose appropriate code identifiers.

4.1. Methodology and Data Collection

In this study, we conjectured whether we can use the words obtained from the description of a programming problem as the reference vocabulary for finding appropriate code identifiers. Considering this assumption, we examined whether using this vocabulary would allow us to find appropriate code identifiers in contrast to the way instructors do. From this point on, we will refer to the proposed method of finding appropriate code identifiers in the student program, through checking the presence of using the terms of the reference vocabulary, as "automated quality assessment". Additionally, the human assessments are those performed by the instructors of a programming course. Thus, the research question that drove our study was: **RQ1)** Does automated quality assessment, to some extent, resembles the human assessment in judging appropriate code identifiers? We evaluated the judgment of appropriate code identifiers obtained from the automated quality assessment in contrast to the study baseline using two traditional information retrieval metrics: *Precision* and *Recall*. *Precision* measures, in the context of our study, the fraction of the "number of identifiers classified as positive by the automated quality assessment, that are true identifiers according to the study baseline" by the "total number of identifiers classified as positive by the automated quality assessment". *Precision* computes the correctness of the automated quality assessment in identifying true identifiers. *Recall*, on the other hand, measures the fraction of the "number of identifiers classified as positive by the automated quality assessment, that are true identifiers according to the study baseline" by

the "total number of true identifiers according to the study baseline". *Recall* computes the completeness of the automated quality assessment when identifying true identifiers. The evaluation of these aspects, correctness and completeness, will shed light on our research question. These are the two concrete hypotheses we have formulated to evaluate the proposed technique: ($H1_1$) The automated quality assessment has acceptable correctness, considering its computed *precision* value; ($H2_1$) The automated quality assessment has acceptable completeness, considering its computed *recall* value.

To reject or accept these hypotheses, we considered the opinion of instructors took part in the study on how applicable are the levels of correctness and completeness of the proposed technique in a programming course. The null hypothesis corresponding to each alternative hypothesis listed above is that the values of the computed metrics could not be considered adequate to reinforce correctness and completeness, respectively. We assign the labels $H1_0$ and $H2_0$ to these hypotheses. In this sense, *precision* and *recall* are considered dependent variables in this study. The data corpus used in this study is composed of 125 identifiers from 58 functionally correct programs from an introductory programming course at our university in the Fall 2017. These programs were implemented in answering to a description written in Portuguese of a programming problem, asking the student to implement a program to detect and inform the player who succeeds in placing "x" or "o" in a horizontal, vertical, or diagonal row in an nxn grid recalling tic-tac-toe. The data collection was done by using an automated test-based assessment system developed in-house and tailored for the introductory programming course.

Initially, we invited five experienced instructors, from the same programming course, to assess the quality of code identifiers. We conducted a survey questionnaire asking, for each identifier, whether "(it) contributes to the program readability". To capture the instructors' opinions, this Likert-scale question, ranged from 1-5 score: "strongly disagree (1); disagree (2); neutral (3); agree (4);" and "strongly agree (5)". In our study design, *score* corresponds to the independent variable. We used Cronbach's *alpha* to measure and assess the inter-rater agreement level, or in other words, the internal consistency degree of the *scores* assigned by the instructors. The resulting *alpha* value (Cronbach's *alpha* of 0.854) was greater than 0.7 revealing that they have a high degree of agreement and that the quality of identifiers was rated similarly across the instructors [López et al. 2015]. As each identifier was evaluated by five instructors, we used the median, a location summary statistic measure, to create an average composite *score* without skewing it by any extremely large or small *scores*. After that, we classified all of the identifiers that obtained a *score* greater than the neutral value as "true" and all other values as "false". This gold-standard of "true" or "false" to identify appropriate code identifiers is our study baseline, it is used as an oracle of instructor-based identifier naming quality. In sequence, we used the automated quality assessment to assess the same code identifiers, finding appropriate and inappropriate identifiers. After that, we classified appropriate and inappropriate identifiers as "positive" and "negative", respectively, to contrast the judgment of appropriate code identifiers obtained from the automated quality assessment with the study baseline.

The illustration presented in Figure 1 shows an example of how the described methodology was applied. The first oval represents six identifiers, obtained from the data corpus used in this study, which are constructions used to refer to variables. The terms "cont_x", "cont_o", "num_bolas" and "valid", on the sequence of the automated

assessment arrow, are classified as “positive”, meaning that they are appropriate identifiers according to automated assessment. These identifiers are considered appropriate as they have the presence of using terms such as “cont”, “num”, “bola” and “valid”, which are part of the Portuguese-written description of the programming problem. The terms “cont_x”, “cont_o”, “i” and “num_bolas”, on the sequence of the instructors’ assessment arrow, are classified as “true”, meaning that they are appropriate identifiers according to human assessment. Summarizing the comparative analysis of both assessment methods, the identifiers classified as “positive” and “true” are contrasted on the sequence of the intersection arrow. The terms “cont_x”, “cont_o” and “num_bolas” are “true positive” (*tp*) identifiers, meaning that they are appropriate identifiers according to both assessment methods. The term “i” is a “false negative” (*fn*) identifier, meaning that even though it is classified as “true” by the instructors, it is classified as “negative” by the proposed automated assessment method. Conversely, the term “valid” is a “false positive” (*fp*) identifier, although it is classified as “positive” by the automated assessment, it is not classified as “true” by the instructors. Finally, the term “p” is a “true negative” (*tn*) identifier, meaning that it is a inappropriate identifier according to both assessment methods.

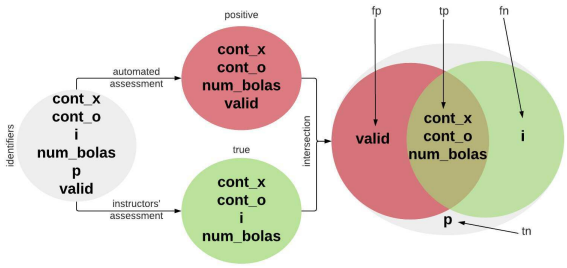

Figure 1. Contrasting instructors’ manual assessment with the automated quality assessment

4.2. Results and Discussion

As a result of the comparison, we calculated the number of identifiers considered to be true positives (*tp*=57), false positives (*fp*=19), false negatives (*fn*=12) and true negatives (*tn*=37). Based on this data, we calculated the value of *precision* ($p = \frac{tp}{tp+fp} = 75.0\%$) and *recall* ($r = \frac{tp}{tp+fn} = 82.6\%$). In order to gather statistical evidence, we studied these data estimating *precision* and *recall* values. Initially, we used the ordinary nonparametric Bootstrap method to recalculate *precision* and *recall* values 2000 times by resampling identifiers classified as *tp*, *fp*, *fn*, and *tn* from the original set of identifiers, with replacement. From these values, we computed the Bootstrap confidence intervals of these metrics using the bias-corrected and accelerated method (*bca*). From the confidence interval analysis, we obtained a *precision* value of (*bca* = [63.7%; 83.5%], 95.0% confidence level) and a *recall* value of (*bca* = [71.8%; 90.4%], 95.0% confidence level).

From these values, we can observe that in our proposed automated assessment method, completeness is greater than correctness as the estimated *recall* value is relatively greater than the estimated *precision* value. Moreover, we emphasize that the values of correctness and completeness are relatively close to totality (greater than 63.0%), which means that this is a promising method for finding appropriate code identifiers. Considering the correctness aspect, we can state with adequate statistical significance that the

proposed automated assessment can correctly find most of the appropriate identifiers classified by the human assessment. Additionally, considering the completeness aspect, we can state that the proposed method can correctly find most of the appropriate identifiers in contrast to the human assessment. For these data, this means that there is evidence that the automated quality assessment, to some extent, resembles the human assessment in judging appropriate code identifiers, considering both the correctness and completeness aspects. In addition, according to instructors took part in the study, the proposed method has acceptable levels of correctness and completeness. In consequence, we reject the $H1$ and $H2$ null hypotheses in favor of the alternatives. We manually examined identifiers considered to be *fn* and *fp* to better understand why the judgment of the automated quality assessment differed from the human assessment. The first differed from the latter, showing *fn* identifiers, in assessing the appropriateness of identifiers using acronyms, abbreviations, synonyms, English terms, and single letters. On the other hand, the first differed from the latter, showing *fp* identifiers, in assessing identifiers with misleading acronyms, two or more vague words, and inappropriate naming convention.

5. Experiment

In this section, we describe an experiment conducted to investigate and demonstrate that we can help students to improve the appropriateness of their code identifiers giving automated feedback. We report on the experience of using *IQCheck* as the proof-of-concept tool that instantiates the method detailed in Section 3 to provide feedback using Portuguese-written descriptions of programming problems.

5.1. Methodology and Data Collection

In this study, we conjectured whether we can use *IQCheck* to provide feedback messages for helping students to improve the appropriateness of their code identifiers. The rationale behind this proposal is that *IQCheck* feedback messages can advise students on which their code identifiers could be renamed picking more appropriate names. Thus, our experiment was carried out to answer the following research question: **RQ2)** Does *IQCheck* feedback messages help students to improve the appropriateness of their code identifiers? We evaluated how effective are *IQCheck* feedback messages at helping students for identifier appropriateness improvement using two metrics: *rai* and Δ_{rai}. For a given student program, *rai* (the ratio of appropriate identifiers) measures the fraction of the "number of identifiers classified as appropriate by *IQCheck*" by the "total number of identifiers." It computes the ratio of identifiers classified as appropriate within the student code. Additionally, for a given pair of student programs, Δ_{rai} measures the difference between the calculated *rai* metric value of the last and first correct functionally program. It computes the difference ratio of identifiers classified as appropriate between the last and first correct program. If Δ_{rai} value is greater than zero, then it means that the number of appropriate identifiers within the last correct program is higher than in the first one, meaning that the student managed to work to deliver more appropriate identifiers.

Initially, we invited 58 students, randomly divided into control and experimental groups, to develop programs for descriptions of programming problems, thoroughly test them, and make sure they were functionally correct. The motivation behind this is that it did not make sense to over-work the student with identifier appropriateness improvement

before they had made their program effectively work, recalling the test-driven development mantra Red Bar/Green Bar/Refactor. In sequence, we invited the students from both groups to improve on the appropriateness of their code identifiers choosing names based on the given programming problem description. We then evaluated the effectiveness of providing *IQCheck* feedback messages to the students of the experimental group in contrast to the students of the control group. The students taking part in this study were enrolled in the same course mentioned in Subsection 4.1 in the Spring 2018, most of them were male (86.2%), and their ages were between 17 and 46. The data corpus used in this study is composed of 231 functionally correct programs implemented by these students in the middle of July 2018. These programs were tested and collected using the automated test-based assessment system used in the course. Those programs were included in our data corpus as the students consented to take part in this study and filled out appropriate forms approved by the committee of ethics at our university.

To provide automated feedback generation, as the first step, we instrumented the automated test-based assessment system used in the course and plugged into it *IQCheck*. In sequence, we chose descriptions of programming problems, which are traditionally used by the instructors in the course, and used *IQCheck* to create files having the reference vocabulary of each one of those descriptions. After that, we sent those files to the working directory of the students of the experimental group so that *IQCheck* could generate feedback messages after their request. Whenever the student requests *IQCheck* feedback messages, *IQCheck* uses a Python module, Abstract Syntax Trees (AST) [AST 1990], to parse the code and an implemented algorithm to extract the names denoting student-written identifiers. To create the file with the reference vocabulary, *IQCheck* uses Python modules of Natural Language ToolKit (NLTK) [NLTK 2001] such Tokenize [Tokenize 2001] to tokenize the description, NLTK's list of Portuguese stop-words to detect and remove stop-words, and RSLP Portuguese stemmer [RSLP 2001] to stem words. In addition, *IQCheck* uses a Python built-in method to convert words with capitalized; Unidecode module [Unidecode 1990] to convert words with accented letters; and RegEx module [RE 1990] to detect and remove special characters, words formed by letters repeated, followed by numbers, and having a length less than two.

5.2. Results and Discussion

We examined the Δ_{rai} metric values computed on the last and first correct programs of the students. As a result of the comparison between the groups, we found the computed Δ_{rai} metric values of the programs of the experimental group were higher than the computed Δ_{rai} metric values of the programs of the control group. It means that the students of the experimental group delivered more appropriate code identifiers in comparison to the students of the other group. To evaluate whether the control and experimental groups have the same Δ_{rai} value distribution, we examined whether Δ_{rai} value distributions of both groups differ their median value. To estimate this difference value, we applied the ordinary nonparametric Bootstrap method to compute the Bootstrap confidence interval of the difference based on 2000 resampling using the first order normal approximation method (*ona*). From the confidence interval analysis, we obtained a difference value of (*ona* = [-48.2%; -26.7%], confidence level 95.0%), which means that the Δ_{rai} values of the experimental group are higher on average than the Δ_{rai} values of the other group.

From this difference value, we can state with adequate statistical significance that

there is a significant and relevant difference between the groups. For these data, this means that there is evidence that *IQCheck* feedback messages are useful and adequate for helping students to improve the appropriateness of their code identifiers. After receiving *IQCheck* feedback messages, 15 students (51.7%) of the experimental group were able to choose better names to denote their code identifiers. In contrast with the other group, this number corresponds to five times the number of students - 3 (10.3%) - who were able to improve their code identifiers' names without receiving *IQCheck* feedback messages. The last and first correct programs implemented by these students revealed Δ_{rai} values greater than zero. In Table 1, we show some lines of the description of a programming problem translated from the original Portuguese-written description to English. In answering to the description showed in Table 1, a student could choose names, to construct their code identifiers, based on words such as "group", "number", "team", "spectator", and "stadium". In Figure 2, we show the first version of the program implemented by a given student in answering the original description, along with the feedback messages it received, in contrast to the second version of the same program.

Table 1. Description of "Largest Fan Group" Programming Problem

In the greater city of the northeastern interior, the competition for the largest fan group of the state is competitive. To solve this issue, it is required that you implement a program that reads the number of first team's spectators in 5 stadium stands and the number of second team's spectators in same 5 stadium stands. You must sum the number of spectators from each team and print the team that brings more spectators into the stadium.

Figure 2. When the student requested feedback messages, *IQCheck* provided the warnings showed in (b) regarding the identifiers in lines 2, 3, 4, 5, 7 e 8, as they were not based on words of the original Portuguese-written description. In addition to these identifiers, *IQCheck* could have marked as inappropriate others such as "cont_torcida1", "time1", "digito", "num1", and "setor5". (c) shows the program after the student actuated on the one showed in (a), renaming the identifier name in line 2 from "a" to a more appropriate: "torcida_a".

Listing 1. (a)

```
1  #coding: utf-8
2  a = 0
3  b = 0
4  for i in range(5):
5      ent = int(...
6      a += ent
7  for k in range(5):
8      ent = int(...
9      b += ent
10 ...
```

Listing 2. (b)

```
- *a* does not appear
to be a suitable name
. You should use word
s from the programmin
g assignment descript
lion.
- *i* does not app...
- *k* does not app...
- *b* does not app...
...
```

Listing 3. (c)

```
1  #coding: utf-8
2  torcida_a = 0
3  b = 0
4  for i in range(5):
5      ent = int(...
6      torcida_a ...
7  for k in range(5):
8      ent = int(...
9      b += ent
10 ...
```

6. Discussion

Although identifier quality does not affect the program behavior, giving feedback on the quality of student-chosen identifiers can help to improve novice programmer training.

When answering the programming problem, the feedback allows students to reflect and improve on the names chosen by them to denote their code identifiers. The cycle request the feedback/receive/rename identifiers allow students to learn a good programming style, as they are pushed to improve identifier naming and their program readability. Although we had encouraged students to request the feedback after they had made sure their program was correct functionally, it also can be offered to functionally incorrect programs. In this case, we believe that the feedback allows students to understand why their code is wrong through the exercise of reading and renaming their code identifiers. It is important to observe that the proposed feedback does not intend to provide an exhaustive analysis of the appropriateness of identifiers, taking into account aspects such as the role, data type or relation with other identifiers which may be synonyms in the program. For this motive, the proposed method does not intend to make the distinction between words which are more relevant to denote identifiers than others, as it does not intend to check terms of high semantic relevance of the programming problem description. We believe that the code identifier quality issues of the programs we evaluated in this study are, in significant part, of the readability nature. However, as our empirical studies were not intended to prove this assumption, it is only an anecdotal suspicion that requires further analysis.

7. Threats to Validity

Human factors threaten our study validity as we invited instructors of a programming course to assess the quality of code identifiers (threat to construct validity) in different moments (threat to internal validity). Also, we used instructor-written descriptions of programming problems for finding appropriate code identifiers' names (threat to construct validity). The two first threats are mitigated as we shared the same assessment criterion with instructors and evaluated the inter-rater agreement level, respectively. The latter threat is diminished as the descriptions used were understandable, well-written, and complete according to the instructors. Although the conclusions should not be generalized to every course (threat to external validity), the ideas and methods proposed in this study can be adapted in different contexts. In this sense, we expected that instructors from other programming courses must take caution when applying the findings of this study.

8. Conclusion

We set out to provide timely feedback on identifier quality assessment so that we could help students to choose better names. To attain this aim, we proposed a method based on the description of programming problem. We performed a study to investigate and demonstrate that we can use this description to automatically find appropriate code identifiers in contrast to the way instructors do. After that, we conducted an experiment in a programming course to examine and prove that automatic feedback effectively helps students to improve on the appropriateness of identifiers at giving advice on which names could be renamed to improve software quality. The bottom line of this paper is that we can improve novice programmer training, giving students feedback about their code identifiers to improve identifier naming and their program readability.

References

Ala-Mutka, K., Uimonen, T., and Jarvinen, H.-M. (2004). Supporting students in c++ programming courses with automatic program style assessment. *Journal of Information Technology Education: Research*, 3(1):245–262.

AST (1990). Abstract syntax trees. https://docs.python.org/2/library/ast.html. [Online; accessed 26-January-2019].

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). Modern information retrieval. addison-wesley.

Butler, S. (2009). The effect of identifier naming on source code readability and quality. In *Proceedings of the Doctoral Symposium for ESEC/FSE on Doctoral Symposium*, ESEC/FSE Doctoral Symposium '09, pages 33–34, New York, NY, USA. ACM.

CheckStyle (2001). http://checkstyle.sourceforge.net/. [Online; acessed 1-June-2019].

De Lucia, A., Di Penta, M., and Oliveto, R. (2011). Improving source code lexicon via traceability and information retrieval. *IEEE Trans. Softw. Eng.*, 37(2):205–227.

Evans, E. and Szpoton, R. (2015). *Domain-driven design*. Helion.

Glassman, E. L., Fischer, L., Scott, J., and Miller, R. C. (2015). Foobaz: Variable name feedback for student code at scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 609–617, New York, NY, USA. ACM.

Jackson, D. and Usher, M. (1997). Grading student programs using assyst. In *ACM SIGCSE Bulletin*, volume 29, pages 335–339. ACM.

Lawrie, D. J., Feild, H., and Binkley, D. (2006). Leveraged quality assessment using information retrieval techniques. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 149–158.

López, X., Valenzuela, J., Nussbaum, M., and Tsai, C.-C. (2015). Some recommendations for the reporting of quantitative studies. *Computers & Education*, 91(C):106–110.

NLTK (2001). Natural language toolkit. http://www.nltk.org/. [Online; accessed 26-January-2019].

PMD (2002). A static source code analyzer. http://pmd.sourceforge.net/. [Online; accessed 1-June-2019].

RE (1990). Regular expression. https://docs.python.org/2/library/re.html. [Online; accessed 28-March-2019].

Rees, M. J. (1982). Automatic assessment aids for pascal programs. *ACM Sigplan Notices*, 17(10):33–42.

RSLP (2001). Stemmer. https://www.nltk.org/_modules/nltk/stem/rslp.html. [Online; accessed 6-April-2019].

Tokenize (2001). Tokenizer. https://www.nltk.org/api/nltk.tokenize.html. [Online; accessed 6-April-2019].

Unidecode (1990). Ascii transliterations of unicode text. https://pypi.org/project/Unidecode/. [Online; accessed 6-April-2019].

Wilcox, C. (2015). The role of automation in undergraduate computer science education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 90–95, New York, NY, USA. ACM.

Apêndice B

Roteiro C - Guia Prático para Ajudar Alunos do Grupo de Controle

Roteiro C

Motivação

Você sabia que os nomes das variáveis e funções afeta a qualidade do programa? Quando um programa possui bons nomes ele se torna mais fácil de ler e ser compreendido. Neste sentido, é importante utilizar bons nomes na nomeclatura deles.

Passo 1: Conhecendo a dinâmica

Escolha qualquer uma das atividades e comece a trabalhar na solução dela.

1. **Faça** login no TST Online como de costume. Acesse a página da maratona e **faça** checkout da atividade. Use o comando: ***tst checkout <código da atividade>***
2. **Leia** o enunciado do problema.
3. **Implemente** seu programa para a solução do problema dentro da pasta da atividade.
4. **Teste** seu programa. No terminal, dentro da pasta da atividade, use o comando: ***tst***
5. Após isso, envie seu programa para o servidor. Para isso, **execute** o comando: ***tst commit***
6. **Observe** se seu programa está correto.
7. Se seu programa estiver correto, é hora de verificar a qualidade dos nomes de variáveis e funções.
8. Para isso, **abra** o seu programa e **verifique** se os nomes escolhidos para denotar variáveis e funções do seu programa são adequados. Caso você ache que algum nome não é adequado, faça a renomeação dele. Durante isso, tente empregar nomes da descrição da atividade.
9. **Repita** os passos 7, e 8 até ter certeza que os nomes do seu programa são realmente adequados.
10. Depois disso, é hora de submeter seu programa para o *tst*. Para isso, execute novamente o comando: ***tst commit***

Apêndice C

Roteiro E - Guia Prático para Ajudar Alunos do Grupo de Experimento

Roteiro E

Motivação

Você sabia que os nomes das variáveis e funções afeta a qualidade do programa? Quando um programa possui bons nomes ele se torna mais fácil de ler e ser compreendido. Neste sentido, é importante utilizar bons nomes na nomeclatura deles.

Passo 1: Conhecendo o *IQCheck*

O *IQCheck* é um plugin que funciona juntamente com o *TST* capaz de avaliar a qualidade dos nomes das variáveis e métodos do seu program. Basicamente, ele avalia a adequação dos nomes escritos por você para nomeação de identificadores. Faça o seguinte:

1. **Acesse** o seguinte link e **baixe** os arquivos: *programa.py* e *iqcheck.json*.
2. **Abra** o terminal e **acesse** a pasta Downloads.
3. **Execute** o seguinte comando: ***unzip milho.zip***
4. **Execute** o comando: ***tst iqcheck programa.py***
5. **Observe** a saída no terminal e faça a renomeação dos identificadores que você julga necessário.

Legal? Agora é hora de conhecer a dinâmica da maratona.

Passo 2: Conhecendo a dinâmica

Escolha qualquer uma das atividades e comece a trabalhar na solução dela.

1. **Faça** login no TST Online como de costume. Acesse a página da maratona e **faça** checkout da atividade. Use o comando: ***tst checkout <código da atividade>***
2. **Leia** o enunciado do problema.
3. **Implemente** seu programa para a solução do problema dentro da pasta da atividade.

4. **Teste** seu programa. No terminal, dentro da pasta da atividade, use o comando: ***tst***
5. Após isso, envie seu programa para o servidor. Para isso, **execute** o comando: ***tst commit***
6. **Observe** se seu programa está correto.
7. Se seu programa estiver correto, é hora de usar o *IQCheck*. Para isso, **execute** o seguinte comando: ***tst iqcheck nome-seu-programa.py***
8. **Observe** a saída no terminal. Alguma mensagem diz que algum dos nomes não é adequado? Caso você esteja de acordo, **abra** o seu programa e faça a renomeação dos nomes apontados na mensagem. Durante isso, tente empregar nomes da descrição da atividade.
9. **Repita** os passos 7, e 8 até ter certeza que os nomes do seu programa são realmente adequados.
10. Depois disso, é hora de submeter seu programa para o *TST*. Para isso, execute novamente o comando: ***tst commit***

Passo 3: Registrando sua opinião

Clique aqui.

Apêndice D

Questionário - Opinião de Alunos Sobre a Utilidade das Mensagens de Feedback Automático

Questionário sobre sua opinião

Registre a sua opinião sobre a experiência de receber o feedback sobre nomes de variáveis e métodos do seu programa respondendo o questionário abaixo.

Sinta-se convidado para entrar em contato por e-mail para enviar dúvidas, críticas ou sugestões: marcosantonio@copin.ufcg.edu.br

*Obrigatório

Endereço de e-mail *

Seu e-mail

Nome *

Sua resposta

Em qual unidade da disciplina você está? *

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

IQCheck é fácil de usar. *

	1	2	3	4	5	
Muito difícil	○	○	○	○	○	Muito fácil

O feedback de IQCheck é útil. *

	1	2	3	4	5	
Muito inútil	○	○	○	○	○	Muito útil

O feedback de IQCheck ajuda a pensar sobre quais nomes são bons. *

	1	2	3	4	5	
Ajuda pouco	○	○	○	○	○	Ajuda muito

O feedback de IQCheck é relevante para você. *

	1	2	3	4	5	
Muito irrelevante	○	○	○	○	○	Muito relevante

O feedback de IQCheck fornece dicas sobre quais nomes são bons. *

	1	2	3	4	5	
Fornece poucas dicas	○	○	○	○	○	Fornece muitas dicas

O feedback de IQCheck reforça a importância de escrever programas com bons nomes. *

	1	2	3	4	5	
Reforça pouco	○	○	○	○	○	Reforça muito

Comentários e sugestões para a ferramenta

Sua resposta

Envie-me uma cópia das minhas respostas.

ENVIAR

reCAPTCHA
PrivacidadeTermos