UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

REYDNE BRUNO DOS SANTOS

Um Estudo Sobre Definição e Avaliação da Readability e Legibility do Código Fonte

Recife

2021

Reydne Bruno Dos Santos

Um Estudo Sobre Definição e Avaliação da Readability e Legibility do Código Fonte

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação, Engenharia de Software e Linguagens de Programação
Orientador: Fernando José Castor de Lima Filho

Recife
2021

Catalogação na fonte
Bibliotecário Cristiano Cosme S. dos Anjos, CRB4-2290

S237e Santos, Reydne Bruno dos
Um estudo sobre definição e avaliação da readability e legibility do código fonte / Reydne Bruno dos Santos. – 2021.
108 f.: il., fig., tab.

Orientador: Fernando José Castor de Lima Filho.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2021.
Inclui referências e apêndices.

1. Engenharia de Software e Linguagens de Programação. 2. Legibilidade de código. 3. Compreensão de código. 4. Compreensão de programa. I. Lima Filho, Fernando José Castor de (orientador). II. Título.

005.1 CDD (23. ed) UFPE - CCEN 2021 – 86

Reydne Bruno Dos Santos

"**Um Estudo Sobre Definição e Avaliação da Readability e Legibility do Código Fonte**"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 22/03/2021.

BANCA EXAMINADORA

Profa. Dra. Jéssyka Flavyanne Ferreira Vilela
Centro de Informática / UFPE

Prof. Dr. Rohit Gheyi
Departamento de Sistemas e Computação / UFCG

Prof. Dr. Fernando José Castor de Lima Filho
Centro de Informática/ UFPE
(Orientador)

Dedico este trabalho à minha mãe, avô e avó.

AGRADECIMENTOS

A meu orientador, Fernando Castor, pela oportunidade, pelo tempo e esforço dedicado a me orientar e me ensinar boas práticas de pesquisa e docência, além de mostrar que a academia é um excelente lugar de protagonismo.

Ao Delano Oliveira e Fernanda Madeiral, pela colaboração, orientação e pelas conversas construtivas. Esse conjunto de atividades colaborou diretamente para a construção desse trabalho.

A Fundação de Amparo a Ciência e Tecnologia do Estado de Pernambuco (FACEPE), pelo apoio financeiro, em seguida por continuar investindo e incentivando a pesquisa em nosso estado.

A professora Carla Silva por ter me orientado no ato da docência e por seus concelhos. Essas ações colaboraram diretamente na minha trajetória profissional.

A minha família, em especial minha mãe Rejane Luíza e a meus avoes Edite Luíza e Reginaldo André, pelo apoio e acolhimento nos momentos difíceis dessa etapa que está sendo concluída.

A Thiago Vinícios, pelo apoio e companheirismo. Por todos os concelhos prestados e por ter permanecido ao meu lado em momentos de alegria e tristeza desse mestrado.

À Amanda Lima, também pelo apoio e por acreditar que esse sonho seria realizado e pelo apoio emocional prestado.

A todos os amigos que fiz durante esse mestrado. Em especial, a Alessandra Aleluia, Chaina Oliveira, Davi Maia, Jéssica Silva, Horhanna Almeida e Sheyla Medeiros por todo suporte durante esse processo.

A todos os amigos que me incentivaram a iniciar o mestrado. Em especial, a Ítala Célly, Ivaldir Junior, Walter Felipe e Márcia Jôsy pelo suporte e recomendações.

A todo o corpo de funcionários do Centro de Informática da UFPE, por toda a ajuda e disponibilidade em vários momentos.

Por fim, agradeço a mim, por ter idealizado esse processo e por não ter desistido mediante aos diversos empecilhos que ocorreram.

A todos, meu muito obrigado!

RESUMO

Entender código fonte é uma atividade importante no desenvolvimento de software. Ela é fundamental durante a correção de problemas, evolução do sistema e durante otimização de código. Código que é difícil de entender pode impactar negativamente essas e outras atividades. A legibilidade do código é um dos fatores que pode estar ligado ao problema de compreensão do código. Muitos estudos em engenharia de software avaliam a legibilidade de diferentes aspectos da linguagem e práticas de programação. Esses estudos fazem suas avaliações de legibilidade através de diferentes características, por exemplo, tempo para completar uma tarefa, número de erros cometidos ou opinião do sujeito. Frequentemente, esses trabalhos utilizam vários termos para discutir sobre os elementos que tornam um trecho de código mais ou menos legível e fácil de compreender. Os termos *readability e legibility* são frequentemente utilizados como sinônimos. No entanto, esses dois termos devem ter significados claros, distintos, embora relacionados, assim como são definidos nas áreas de educação, linguística e design, onde representam aspectos diferentes do texto ou do ambiente. Assim, esse trabalho tem como objetivo estudar de forma ampla o conceito de legibilidade de código e como ele vem sendo estudado na área de Engenharia de Software. Para atingir esse objetivo, realizamos um levantamento bibliográfico sobre a temática em várias áreas do conhecimento e uma revisão sistemática da literatura na área de Engenharia de Software, onde encontramos 54 artigos relevantes para nosso estudo. Nossos resultados revelam que os conceitos de *readability e legibility* são bem definidos em áreas como Design e Linguística, mas na área de Engenharia de Software há termos diferentes que são sinônimos e o mesmo termo sendo usado de forma inconsistente em diferentes contextos. São encontrados na literatura vários estudos empíricos que avaliam o que torna o código mais legível. Entretanto, cada um apresenta um método diferente para medir a legibilidade do código. A maioria dos estudos incluídos na revisão sistemática avalia a *readability e a legibility* do código medindo a correção dos resultados dos sujeitos (83,3%) ou simplesmente pedindo suas opiniões pessoais (55,6%). Alguns estudos (16,7%) dependem exclusivamente desta última variável de resposta. Além disso, ainda existem relativamente poucos estudos que monitoram os sinais físicos do desenvolvedor, como as regiões de ativação cerebral (5%). Esses resultados não só contribuem com o conhecimento existente sobre o assunto como também deixam claro que abordagens de avaliação de compreensão de código diferentes requerem competências diferentes dos sujeitos do estudo, por exemplo, rastrear o programa versus resumir seu objetivo versus memorizar seu texto.

Palavras-chaves: Legibilidade de código. Compreensão de código. Compreensão de programa.

ABSTRACT

Understanding source code is an important activity in software development. It is essential during problem correction, system evolution and code optimization. Code that is difficult to understand can negatively impact these activities and other activities. Code readability is one of the factors that can be linked to the problem of understanding the code. Many studies in software engineering assess the readability of different aspects of programming language and practices. These studies make their readability assessments through different characteristics, for example, time to complete a task, number of mistakes made or subject's opinion. These works often use several terms to discuss the elements that make a piece of code more or less readable and easy to understand. The terms legibility and legibility are often used interchangeably. However, these two terms must have clear, distinct, yet related meanings, just as they are defined in the areas of education, linguistics and design where they represent different aspects of the text or the environment. Thus, this work aims to study the broad concept of code readability concept and how it has been studied in the area of Software Engineering. In order to achieve this objective, we carried out a bibliographic survey on the subject in various areas of knowledge and a systematic review of the literature in the area of Software Engineering, where we found 54 articles relevant to our study. Our results reveal that the concepts of legibility and legibility are well defined in areas such as Design and Linguistics, but in the area of Software Engineering there are different terms that are synonymous and the same term being used inconsistently in different contexts. Several empirical studies are found in the literature that evaluate what makes the code more readable. However, each has a different method for measuring code readability. Most of the studies included in the systematic review assessing the readability and legibility of the code by measuring the correctness of the subjects' results (83.3%) or simply asking for their personal opinions (55.6%). Some studies (16.7%) depend exclusively on this last response variable. In addition, there are still few studies that monitor the developer's physical signals, such as regions of brain activation (5%). These results not only contribute to the existing knowledge on the subject, but also make it clear that different code comprehension assessment approaches Competencies different from studies, for example, tracking the program versus summarizing its purpose versus memorizing its text.

Key-words: Code legiblility. Code comprehension. Program comprehension.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

No contexto complexo da engenharia de software, no qual sistemas são vistos como produtos, encontram-se desafios cada vez mais acentuados, por exemplo, a construção de software escaláveis e a manutenção dos sistemas complexos (BOSU et al., 2016). Na construção de um software o nome de identificadores (TEASLEY, 1994), anti-padrões de código (SANTOS; GEROSA, 2018) e estilos de formatação do código (OMAN; COOK, 1990) podem impactar a tarefa de compreensão do código e causar prejuízos ao desenvolvimento de um sistema de software (FAKHOURY et al., 2019a; STOREY, 2006; SHARGABI et al., 2020).

A tarefa de compreensão de programas é uma tarefa cognitiva de alta complexidade (LETOVSKY, 1987). A interpretação de um trecho de código depende de sub-tarefas que, por sua vez, requerem diferentes aspectos como conhecimento do domínio do problema do mundo real; conhecimento de estratégias de design e componentes de design úteis; conhecimento da sintaxe da linguagem de programação, regras de estrutura de texto e convenções de programação; conhecimento dos recursos do computador que afetam a implementação do programa; e conhecimento do usuário do programa (BROOKS, 1983; PENNINGTON, 1987).

Haja vista a importância dada pela academia nos últimos anos para o assunto de compreensão de código (BUSE; WEIMER, 2009), um estudo realizado com 222 programadores profissionais buscou medir o seu desempenho na tarefa de entender/interpretar trechos de código. Os resultados desse estudo mostrou que diferentes estruturas de código demandam diferentes quantidades de tempo para serem interpretadas por desenvolvedores (AJAMI; WOODBRIDGE; FEITELSON, 2019). Nesse mesmo sentido, o estudo de Gopstein et al. (GOPSTEIN et al., 2017) traz a conclusão de que em projetos reais e bem-sucedidos também são encontrados elementos que dificultam a compreensão do código, por exemplo operador condicional e delimitadores de bloco. Esse trabalho sugere que pequenos padrões de código impactam o processo de compreensão de código e que eles podem proporcionar mais erros.

Outros trabalhos sobre compreensão avaliam aspectos ligados ao conteúdo do código, por exemplo, o trabalho de Benander et al. (BENANDER; BENANDER; PU, 1996), estudou aspectos ligados a construção do código. Por outro lado, existe também estudos que avaliaram aspectos visuais do código, por exemplo, o trabalho de Geffen e Maoz (GEFFEN; MAOZ, 2016) que estudou estilos diferentes de ordenação de métodos. Esses trabalhos são sobre compreensão de programas e legibilidade de código, mas misturam questões diferentes. Alguns, por exemplo, avaliam questões relacionadas à identificação de elementos do programa, como recuo e tipos de delimitadores, enquanto outros avaliam como o pro-

grama funciona, por exemplo, recursão e interação. Embora possam estar relacionados, esses elementos são diferentes, um está ligado a *readability* do código e outro está ligado a *legibility*.

1.2 PROBLEMA

Alguns recentes trabalhos procuram estudar como a legibilidade de código pode afetar a compreensão do código (FAKHOURY et al., 2018; SANTOS; GEROSA, 2018). Alguns desses trabalhos abordam as temáticas de *readability* e *legibility* porém, na área de Engenharia de Software, esses termos ainda não são bem definidos e muitas vezes se sobrepõem. Em outras áreas, como Educação (LAUGHLIN, 1969), Linguística (DALE; CHALL, 1949) e Design (WEISMAN, 1981), esse cenário é diferente, uma vez que os estudos dessas áreas de pesquisa acreditam que o termo *legibility* está conectado com aspectos visuais como o tamanho da letra e o espaçamento entre os elementos do texto. Por outro lado, *readability* engloba as características que permitem ler facilmente frases, independentemente de seu significado.

Na área de Engenharia de Software os elementos ligados a características visuais do código-fonte como recuo e espaçamento e características sintáticas e semânticas do código-fonte, por exemplo construções de linguagens de programação, padrões de codificação e identificadores significativos, são consideradas como atributos da *readability* do código por alguns trabalhos. Por exemplo, o trabalho de Gopstein et al. (GOPSTEIN et al., 2017) e o trabalho de Santos et al. (SANTOS; GEROSA, 2018) que estudaram o impacto de um conjunto de práticas de codificação que continham elementos visuais, sintáticos e semânticos do código, como expressões lógicas, linhas em branco e colchetes. Ambos os trabalhos não apresentam uma distinção conceitual entre esses elementos.

Nessa dissertação é considerado que as características sintáticas e semânticas do código-fonte de um programa que afetam a capacidade dos desenvolvedores de entendê-lo durante a leitura, por exemplo, construções de programação, idiomas de codificação e identificadores significativos, afetam a *readability*. Por outro lado, as características visuais do código-fonte de um programa, que afetam a capacidade dos desenvolvedores de identificar os elementos do código durante a leitura, como quebras de linha, espaçamento, alinhamento, recuo, linhas em branco, capitalização do identificador, impacta a *legibility*. No entanto, não existem pesquisas na área de Engenharia de Software que considerem essa distinção.

Também existe a necessidade de produção de pesquisas, na área de engenharia de software, que estudem as maneiras de escrever código mais legível. Embora existam guias de estilos de programação que recomendem padrões com objetivo de melhorar a legibilidade, esses guias de estilo muitas vezes não foram construídos através de estudos empíricos como, por exemplo, o guia de estilo javascript (JAVASCRIPT, https://google.github.io/styleguide) da google e o guia de estilo c++ (C++, https://isocpp.github.io/CppCoreGuidelines).

Uma recente pesquisa de Gopstein et al. (GOPSTEIN et al., 2017) mostra que alguns guias de estilo recomendam o uso de padrões confusos.

1.3 OBJETIVO

Esse trabalho pretende produzir conhecimento sobre o conceito a avaliação dos aspectos de readability e legibility do código-fonte. Para isso, busca responder às seguintes questões de pesquisa:

RQ1 Como Engenharia de Software e outras áreas do conhecimento definem *readability* e *legibility*?

RQ2 Como podemos avaliar *code readability* e *code legibility*?

O objetivo geral deste trabalho é caracterizar de forma ampla o conceito de legibilidade de código. Isso nos permitirá entender o que vem sendo produzido na área de Engenharia de Software e auxiliará no desenho de novos estudos empíricos. Portanto esse trabalho tem como objetivos específicos:

- Identificar como trabalhos na área de Engenharia de Software avaliam legibilidade de código;
- Analisar como outras áreas, onde a legibilidade é importante, definem esse conceito;
- Propor uma definição de legibilidade de código baseada na análise do assunto em outras áreas.

Para responder as questões de pesquisa e atingir os objetivos do estudo foram realizados um levantamento bibliográfico e uma revisão sistemática da literatura a fim de identificar quais as definições sobre *readability* e *legibility* e como esses atributos estão sendo avaliados. Dessa forma, esse trabalho poderá impactar pesquisadores na elaboração de pesquisas voltadas para esses aspectos da área de compreensão de código.

1.4 ESTRUTURA DA DISSERTAÇÃO

O restante deste trabalho está organizado da seguinte maneira. No Capítulo 2 é apresentado o referencial teórico sobre compreensão de código e sobre a engenharia de software baseada em evidência. No Capítulo 3 é apresentada a metodologia utilizada neste trabalho, abordando aspectos como o planejamento, condução do trabalho e documentação. No Capítulo 4 são apresentados os resultados do levantamento bibliográfico e da revisão sistemática da literatura desenvolvida neste trabalho. Por último, no Capítulo 5 é apresentado as considerações finais deste trabalho.

2 FUNDAMENTOS

Inicialmente, para construção do referencial teórico que norteia esse trabalho foram analisados 12 artigos sobre a temática de compreensão de programa. Esses trabalhos foram encontrados através de sugestões em reuniões do grupo de pesquisa e em reuniões de orientações com pesquisadores envolvidos nesse trabalho. Com o objetivo de encontrar mais artigos relevantes para construir o referencial teórico, foi conduzida uma pesquisa na ferramenta do google scholar utilizando as palavras: “Program Comprehension”, o critério para seleção de um estudo foi guiado pela quantidade de citações recebidas. Um dos primeiros resultados retornados foi o trabalho de Mayrhauser and Vans (MAYRHAUSER; VANS, 1993).

O título desse trabalho foi utilizado para uma outra busca, dessa fez na ferramenta Connected papers. Essa ferramenta tem como objetivo ajudar pesquisadores a encontrar e explorar artigos relevantes para seu campo de trabalho. Ela faz isso montando um gráfico através de uma análise de trabalhos fortemente relacionados com o título do trabalho pesquisado. Os artigos são organizados de acordo com sua similaridade e não por ordem de citação. Os resultados dessa pesquisa foram analisados com base em sua relevância para a pesquisa. Ao final dessa análise, foram coletados um total de 18 trabalhos para serem lidos e sintetizados no referencial teórico desta pesquisa.

A última estratégia de busca para construção do referencial teórico foi a busca por trabalhos publicado na International Conference Program Comprehension (ICPC). Essa pesquisa foi realizada no mecanismo de busca da ACM Digital Library. Nesse mecanismo a pesquisa foi configurada para retornar apenas trabalhos publicados na ICPC, contendo como palavra-chave as palavras: “program comprehension”, dentro dos anos de 2014 a 2020. Foram retornados nessa busca 56 resultados, o título e resumo de cada artigo foi analisado e os mais relevantes para a pesquisa foram incluídos no referencial teórico.

A partir disso, é apresentado neste capítulo o embasamento teórico que foi usado como fonte para elaboração deste trabalho. O capítulo está organizado da seguinte forma: Na seção 2.1 são apresentados os principais conceitos sobre compreensão de código, como: características principais e um breve histórico sobre o tema. Na seção 2.2 são apresentados alguns fatores que impactam a compreensão de código. Na seção 2.3 são apresentados os conceitos acerca da Engenharia de Software Baseada em Evidência, em específico o método de revisão sistemática e seus processos.

2.1 UMA VISÃO GERAL SOBRE COMPREENSÃO DE CÓDIGO

Sabe-se que a compreensão de código é uma das atividades que mais demandam esforço e tempo de um desenvolvedor (FAKHOURY et al., 2018) e (AJAMI; WOODBRIDGE; FEITELSON,

2019). Entender um código vai além de conseguir ler o que está escrito (SCALABRINO et al., 2017). Para Ajami et al. (AJAMI; WOODBRIDGE; FEITELSON, 2019), a complexidade é um fator que pode influenciar a compreensão de código. Ela pode ser composta por: tamanho do código, elementos sintáticos, padrões de fluxo de dados, nome de variáveis, entre outros. O trabalho de Mynatt (MYNATT, 1984), colabora com essa afirmação, mostrando que recursividade, linearidade do algoritmo e tipo de estruturas de dados usados podem causar impactos na compreensão de código.

Scalabrino et al. (SCALABRINO et al., 2017), reforçam que o entendimento pode depender também de fatores externos como a experiência do programador. Todos esses trabalhos mostram a importância do estudo de compreensão de código. Eles são promissores no que tange os aspectos de impacto dessa tarefa, contudo, Beniamini et al. (BENIAMINI et al., 2017) afirma que mesmo com tantos avanços, a engenharia de software não supre todas as expectativas. Neste trabalho, o autor afirma que os códigos de difícil compreensão são mais um obstáculo no trajeto de um desenvolvedor exercendo sua função. Para entender melhor esse e outros aspectos o estudo sobre os modelos de compreensão de programas pode ser útil. Nesse sentido, as subseções a seguir discorrem sobre os principais modelos encontrados na literatura.

2.1.1 Teoria de compreensão de cima para baixo

Uma das principais teorias de compreensão de programas foi cunhada por Brooks (BROOKS, 1983). Em seu trabalho, Brooks afirma que a compreensão de código pode ser considerada uma tarefa de interpretação de um código que se baseia no conhecimento prévio do indivíduo. Neste sentido, o autor afirma que o processo de compreensão de código é um processo de reconstrução de cima para baixo, baseado em hipóteses, em que uma hipótese inicialmente vaga e geral é refinada e elaborada com base nas informações extraídas do texto do programa e de outra documentação. O autor então resume sua teoria da seguinte forma:

> *"1.O processo de programação consiste em construir mapeamento de um domínio de problema, possivelmente através de vários domínios intermediários, para o domínio de programação; 2. Compreender um programa envolve reconstruir parte ou todos esses mapeamentos; 3. Este processo de reconstrução é a expectativa impulsionada pela criação, confirmação e refinamento de hipóteses"* (BROOKS, 1983).

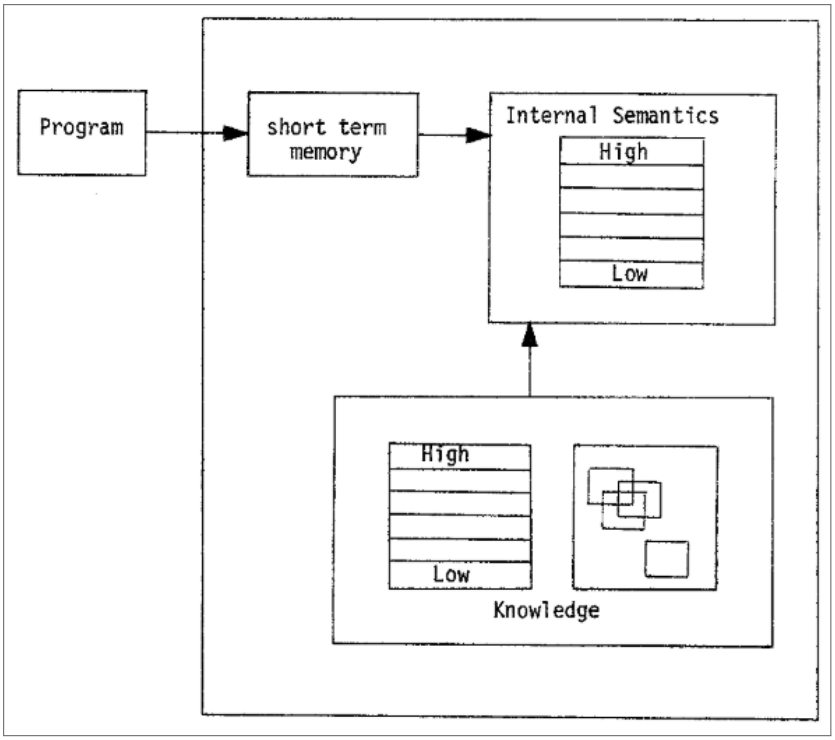
Outro trabalho que procurou estudar a teoria proposta por Brooks (BROOKS, 1983), foi o trabalho de Soloway and Ehrlich (SOLOWAY; EHRLICH, 1984), nesse trabalho os autores estudam se os programadores especialistas fazem uso de planos de programação (fragmentos de programas que representam sequências de ação) e regras de discurso (regras que especificam uma convenções em programação), mas acabam mostrando que o método

de compreensão de cima para baixo é utilizado em situações que o código é familiar. Neste estudo, os autores definem compreensão de código de forma genérica afirmando que ela é "a capacidade de entender um determinado problema de código". Os resultados desse trabalho mostram que os planos de programação e as regras de discurso desempenham um papel importante na compreensão do programa.

2.1.2 Teoria da compreensão de baixo para cima

O trabalho de Shneiderman e Mayer (SHNEIDERMAN; MAYER, 1979), apresenta a teoria de um processo de compreensão de baixo para cima. Nesse processo, o desenvolvedor constrói o entendimento de um programa através da construção de uma estrutura semântica multinível. No nível mais alto, o desenvolvedor entende o objetivo do programa, ou seja, o que o programa faz. No nível mais baixo, o desenvolvedor pode reconhecer sequências familiares de declarações ou algoritmos. Na ilustração Figura 1, é possível visualizar essa ideia proposta pelo autor. Neste trabalho, os autores também definem a compreensão de programa como a capacidade de entender determinado problema.

Figura 1 – Processo de compreensão do programa: a formação da semântica interna para um determinado programa.



Fonte: (SHNEIDERMAN; MAYER, 1979)

Outro trabalho que também descreve essa teoria de compreensão é o trabalho de Pennigton (PENNINGTON, 1987), nesse trabalho a autora procurou estudar como o conhecimento de programação e as representações mentais do programa influenciam o processo de compreensão de um desenvolvedor. Para isso, a autora realizou dois experimentos com ati-

vidades de compreensão, reconhecimento e modificação de programas. Ela observou nesses experimentos que a compreensão de programa obtida pelos desenvolvedores se constrói pelo conhecimento abstrato das estruturas de texto do programa que desempenha o papel organizador inicial na memória, aliado ao fluxo de controle ou relações procedimentais que dominam a representação da memória da macroestrutura.

2.1.3 Teoria da compreensão sistemáticas e conforme necessário (*as-needed*)

Littman et al. (LITTMAN et al., 1987), observou 10 programadores profissionais com cerca de 5 anos de experiência durante entrevistas, em que os sujeitos descreviam seus pensamentos à medida que realizavam uma tarefa. Esse estudo tinha como objetivo investigar o que significava entender um programa para desenvolvedores de manutenção de código. Os autores descrevem neste trabalho duas estratégias de compreensão. Sendo elas:

- Estratégia sistemática onde o "desenvolvedor quer entender como o programa se comporta antes de tentar modificá-lo". Essa abordagem garante que o programador leve em consideração as sub-rotinas do programa nas alterações;
- Na estratégia conforme necessário (*as-needed*) o programador tenta localizar o mais rápido possível as partes do programa que devem ser modificadas. Essa abordagem, por outro lado, não garante que o desenvolvedor identifique quais são as partes do programa que podem ser afetadas pelas modificações.

Os resultados mostram que o uso de uma estratégia de compreensão de código sistemática colaborou para que os desenvolvedores adquirissem um conhecimento causal necessário para fazer o aprimoramento no código. Embora essa estratégia seja a mais adequada, ela pode não ser a mais rápida em mostrar resultados.

O trabalho de Letovsky (LETOVSKY, 1987) gravou seis programadores profissionais em uma tarefa de adição de um novo recurso em um programa existente com o objetivo de entender quais são os processos cognitivos envolvidos na compreensão de programas. O autor mostra que fazer perguntas e supor respostas são atividades que dominam os comportamentos dos desenvolvedores em atividades de compreensão de código. O processo de fazer perguntas surge de acordo com a mistura de entendimento de cima para baixo e de baixo para cima. O processo de supor respostas surgem para responder os questionamentos estabelecidos (LETOVSKY, 1987).

2.1.4 Teoria do Modelo Integrado

O trabalho de Mayrhauser e Vans (MAYRHAUSER; VANS, 1993) procurou construir um modelo de compreensão de programa que combina os modelos de cima para baixo, de baixo para cima, o modelo sistemático e conforme necessário (*as-needed*). Esse estudo foi conduzido porque os autores observaram que a compreensão de código no processo de

manutenção de software intercala entre as estratégias de cima para baixo e de baixo para cima. O modelo proposto consiste de quatro componentes, sendo eles:

- Modelo de Programa: É uma abstração do fluxo de controle que é geralmente construído quando o desenvolvedor não conhece o código.
- Modelo de Situação: Descreve o fluxo de dados e as abstrações de um programa. Ele pode ser construído através do entendimento de baixo para cima ou conforme necessário.
- Modelo de cima para baixo (ou modelo de domínio): Esse modelo representa esquemas de conhecimento sobre o domínio da aplicação e é normalmente invocado durante a compreensão se a área de aplicação for familiar para o desenvolvedor.
- Base de Conhecimento: Refere-se a fonte de informação relacionada à tarefa de compreensão. Pode servir como fornecedor de informações como também coletor de novos conhecimentos.

Mayrhauser e Vans (MAYRHAUSER; VANS, 1993) validaram esse modelo através de um estudo envolvendo programadores profissionais de manutenção que trabalham com programas grades, ou seja, programas com várias linhas de código. Esses participantes foram solicitados a pensar em voz alta enquanto realizavam uma tarefa de compreensão de código. Os resultados mostram que a compreensão de código, em programadores de manutenção, se constrói através de níveis de abstração que alteram entre três processos de compreensão.

2.1.5 Resumo

O estudo do processo de compreensão de programa ajuda a esclarecer os pontos chaves para o desempenho do programador em tarefas de compreensão intensiva, como manutenção de programas (LETOVSKY, 1987). O trabalho de Shneiderman e Mayer (SHNEIDERMAN; MAYER, 1979), nesse sentido apresenta um modelo cognitivo de comportamento do programador que foi desenvolvido em resposta a experimentos controlados. Este modelo cognitivo separa o conhecimento sintático do conhecimento semântico e enfatiza a representação interna criada pelo programador nas tarefas de programação de composição, compreensão, depuração, modificação e aprendizagem. Esse modelo mostra que programadores realizam um processo de compreensão de código de baixo para cima. O trabalho de Pennington (PENNINGTON, 1987) também apoia essa ideia e informa que os objetivos da tarefa de um programador também podem influenciar o processo de compreensão do código de baixo para cima.

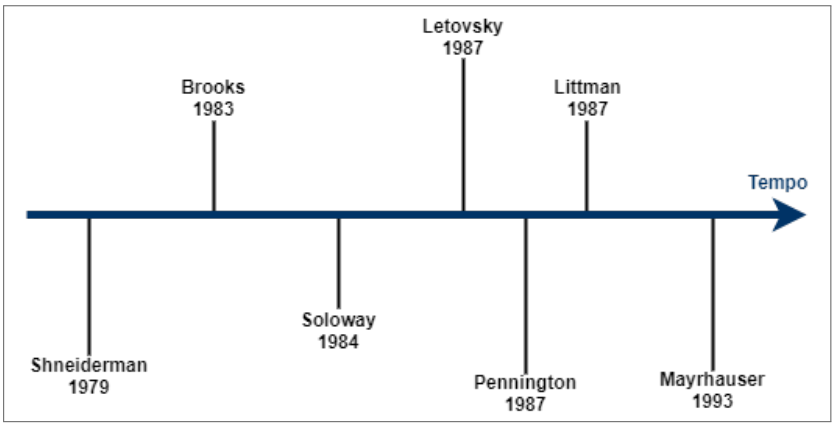
O trabalho de Brooks (BROOKS, 1983), por outro lado, conclui que o processo de compreensão de código é um processo de reconstrução de cima para baixo, baseado em

hipóteses, em que uma hipótese inicialmente geral é refinada e elaborada com base nas informações extraídas do texto do programa e de outra documentação. O trabalho de Soloway e Ehrlich (SOLOWAY; EHRLICH, 1984) também concorda que a compreensão de programas é realizada de cima para baixo e afirma que os planos de programação e as regras de discurso desempenham um papel importante na compreensão do programa.

Outros trabalhos como Littman et al. (LITTMAN et al., 1987) e Letovsky (LETOVSKY, 1987) apoiam a ideia de que o processo de compreensão pode acontecer de forma sistemática ou conforme necessário. Os autores apoiam a ideia de que a estratégia sistemática é a mais eficiente, porém pode não ser a mais ágil, para algumas atividades como manutenção de sistemas. Outros dois autores, Mayrhauser e Vans (MAYRHAUSER; VANS, 1993), apresentam um modelo que integra todos esses outros modelos acima propostos. Os autores afirmam que programadores constroem uma compreensão do sistema através de níveis de abstração que alteram entre os três processos de compreensão discutidos anteriormente.

Esses trabalhos são, ainda hoje, boas referências para o estudo de compreensão de código. A Figura 2 mostra uma linha do tempo com a distribuição de todos trabalhos considerados nessa seção. Todos eles envolveram desenvolvedores nos seus diversos níveis de experiência. Porém, as tarefas solicitadas nos experimentos, muitas vezes, diferem umas das outras. Por exemplo, no trabalho de Mayrhauser e Vans (MAYRHAUSER; VANS, 1993), os participantes tiveram que pensar em voz alta enquanto realizavam uma tarefa de compreensão de código. Enquanto no trabalho de Letovsky (LETOVSKY, 1987), os sujeitos foram filmados em uma tarefa de adição de um novo recurso em um programa existente. Isso pode explicar o fato de existirem três modelos de processos de compreensão de código diferentes. Em outras palavras, o tipo de tarefa e a metodologia utilizada para análise de dados podem influenciar a forma que as evidências foram construídas. Ou, simplesmente, pode-se afirmar que existem várias formas de compreensão de código. O que determina qual será utilizada pode ser o objetivo da tarefa, o conhecimento do programador ou ainda a formação acadêmica recebida.

Figura 2 – Linha do tempo de trabalhos que propõem modelos de compreensão de softwares.



Fonte: Elaborado pelo autor (2020)

2.2 OS FATORES DE IMPACTO PARA COMPREENSÃO DE CÓDIGO

Nos últimos anos, muitos trabalhos na área de compreensão de progamas procuraram estudar os fatores que influenciam a legibilidade do código. Muitos desses trabalhos podem ser encontrados no *International Conference on Program Comprehension* (ICPC). Uma parcela desses trabalhos envolveu experimentos com seres humanos em atividades de compreensão de código e podem ser categorizados e separados em grupos de acordo com suas semelhanças.

Uma das categorias existentes é a de **identificadores**. Nessa categoria podem ser encontrados estudos que abordam assuntos como a significância dos nomes de identificadores, como é o caso do trabalho de Beniamini et al. (BENIAMINI et al., 2017) que procurou estudar como as variáveis de letra única são usadas. Acredita-se que nome de variáveis são importantes porque ajudam na compreensão do programa, por isso devem ser significantes. Porém, algumas estruturas utilizam apenas uma única letra para nomear variáveis. Os resultados do estudo de Beniamini et al. (BENIAMINI et al., 2017) mostram que os nomes de variáveis com uma letra são pouco utilizados, mas são tão comuns quanto variáveis de tamanhos curtos. Nesse sentido, se o desenvolvedor não tiver o conhecimento e as habilidades necessárias, bons nomes de variáveis não o ajudam muito na tarefa de compreensão (BENIAMINI et al., 2017). Nesse trabalho os nomes de variáveis com letra única não apresentam fortes indícios de que são ruins para compreensão de código.

Outro trabalho que também procurou estudar identificadores foi o trabalho de Avidan e Feitelson (AVIDAN; FEITELSON, 2017), neste trabalho os autores examinaram o impacto dos nomes de variáveis. Os resultados mostraram que a nomeação de identificadores tem um forte impacto para a compreensão do código. Os nomes dos parâmetros e dos métodos apresentam mais significância comparados aos nomes de variáveis locais. Portanto, os nomes de variáveis, métodos e parâmetros devem ser uma escolha cautelosa, pois devem

refletir a sua função. Por meio desses trabalhos é possível afirmar que a significância dos nomes de identificadores e os conhecimentos individuais do programador podem afetar a tarefa de compreensão de código.

Há também trabalhos que avaliam se o uso de **espaçamento** ajuda na legibilidade do código. No trabalho de Bauer et al. (BAUER et al., 2019), por exemplo foi estudado o efeito de recuo de código. O trabalho teve como objetivo fornecer evidências empíricas para os níveis de recuo propostos em guias de estilo de codificação e mostrou que o nível de recuo não tem efeito sobre a compreensão de código porque o tempo de resolução do problema e o número de acerto não foram afetados. Os autores afirmam, portanto, que o nível de recuo pode ser considerado apenas como um fator negativo para o estilo do código (BAUER et al., 2019).

Outro trabalho que estuda os níveis de espaçamento do código é o de Santos e Gerosa (SANTOS; GEROSA, 2018), nele os autores investigaram como as práticas de codificação influenciam nos níveis de legibilidade percebidos pelos desenvolvedores. Uma das práticas estudadas foi o uso de recuo de 4 espaços. Eles não encontraram significância estatística em seus resultados que pudessem afirmar se essa prática causa efeito positivo ou negativo à legibilidade de código. Porém, esses resultados podem ser apenas questões de critério pessoal, uma vez que foi analisado apenas a opinião do desenvolvedor (SANTOS; GEROSA, 2018). Com base nesses trabalhos, que analisaram níveis de espaçamento do código, pode-se afirmar que o nível de recuo do código não é um fator de impacto negativo para a compreensão do código, mas que pode ser negativo para o estilo do código.

Também são encontrados trabalhos que estudaram características de **fluxo de controles**. Nessa categoria alguns dos trabalhos investigaram os estilos de construções, como é o exemplo do trabalho de Ajami et al. (AJAMI; WOODBRIDGE; FEITELSON, 2019) que esteve preocupado em medir como diferentes fatores sintáticos influenciam na complexidade e compreensão do código. Alguns dos resultados reportados no trabalho mostram que construções de repetição *for* são mais difíceis de entender do que construções de seleção *if*.

Outro trabalho que estudou características de fluxo de controle foi o de Santos e Sant'anna (SANTOS; SANT'ANNA, 2019) que procurou entender como a **dependência de recurso** afeta a compreensão do código-fonte do sistema configurável. Para os autores, dependências de recurso ocorrem quando diferentes recursos se referem ao mesmo elemento do programa, como por exemplo uma variável. Esse estudo também realizou um experimento com desenvolvedores de software e concluiu que as dependências de recursos afetaram a compreensibilidade do programa de diferentes maneiras, por exemplo: (i) as dependências globais e inter-procedurais exigiram mais tempo de depuração do que as dependências intra-procedurais e (ii) as dependências inter-procedurais exigiram um esforço visual maior do que os outros dois tipos de dependência. Portanto, essas evidências afirmam que a técnica de compilação condicional também impacta a compreensão de código

(SANTOS; SANT'ANNA, 2019).

Ainda na categoria dos trabalhos sobre fluxo de controle encontra-se um grupo que investigou as construções e estilos de repetição. O trabalho de Iselin (ISELIN, 1988) é um bom exemplo desse grupo. Neste estudo, os autores descrevem um experimento que visa fornecer uma visão sobre os efeitos de dois estilos de *loop* no desempenho da compreensão do programa. O primeiro tipo foi chamado de *read/process*, ou seja, o i-ésimo elemento é lido e processado na passagem pelo *loop*. O segundo tipo foi chamado de *process/read* e nele o elemento i-ésimo é processado e em seguida é lido. Os resultados dessa pesquisa mostram que o estilo de *loop* de *read/process* seria mais fácil de compreender do que *process/read*. Esses estudos sobre fluxo de controles mostram que determinadas estruturas de *loops* são difíceis de serem entendidas pelos desenvolvedores. Portanto, pode-se afirmar que a escolha desse tipo de estrutura deve ser uma tarefa cautelosa porque pode impactar negativamente a compreensibilidade do programa.

Outro grupo de estudos procurou investigar como **elementos não relacionados a código-fonte** podem afetar a compreensão do programa. O trabalho de Roehm (ROEHM, 2015), por exemplo, teve como objetivo estudar o uso da interface do usuário para compreensão do programa. Esse estudo mostra que os desenvolvedores podem interagir com a interface do usuário, a fim de reproduzir falhas, encontrar o código-fonte relevante, para testar uma mudança implementada, para acionar o depurador ou para se familiarizar com partes desconhecidas do aplicativo. Portanto, a interface do usuário pode ser um dos recursos úteis na compreensão de código.

Outros dois estudos também investigaram elementos não relacionados ao código-fonte. O primeiro foi o trabalho de Dias et al. (DIAS et al., 2020) que avaliou como duas ferramentas de visualização de código influenciam a compreensão do programa pelos desenvolvedores. Esse estudo indica que o uso das visualizações fornecidas por ferramentas de desenvolvimento de programa, como por exemplo o *Visual Studio Code* (VS Code), diminui o tempo necessário para realizar tarefas de compreensão de software. O segundo estudo é o de Shargabi et al. (SHARGABI et al., 2020) que teve o objetivo de investigar e comparar o efeito de seis tarefas no modelo mental de compreensão do programa de desenvolvedores novatos. Esse trabalho mostrou que para cada nível de conhecimento em programação, existem tarefas que são mais adequadas para a compreensão do programa. Portanto, embora esses três últimos trabalhos mencionados (ROEHM, 2015), (DIAS et al., 2020) e (SHARGABI et al., 2020) não tenham investigado características específicas do código-fonte, eles também fornecem evidências de elementos externos ao código que podem impactar a tarefa de compreensão.

2.2.1 Resumo

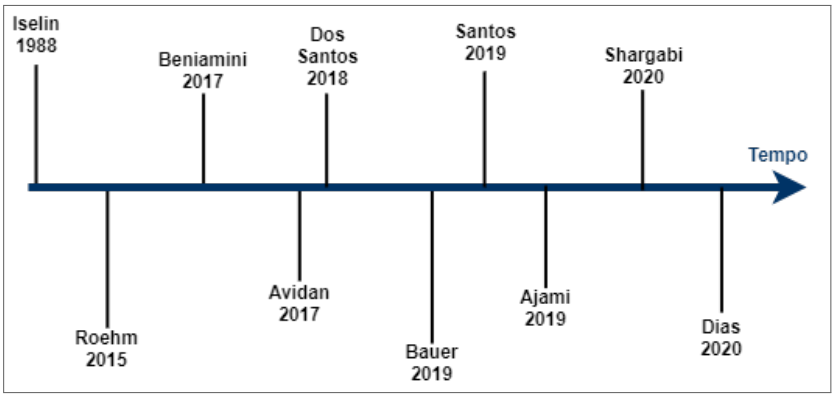
Estudar como os desenvolvedores entendem código pode permitir melhorar as técnicas e recursos existentes para compreensão do código, que por sua vez levam a redução de

tempo e custo da produção de programas. Nesse sentido, os estudos sobre fatores que influenciam a compreensão de código mostram diversos elementos sintáticos e semânticos do código que podem ser positivos ou negativos para a compreensão do código (SANTOS; GEROSA, 2018), (AJAMI; WOODBRIDGE; FEITELSON, 2019), (SANTOS; SANT'ANNA, 2019). Uma breve análise desses estudos revela que existem categorias de trabalhos que procuraram investigar assuntos específicos como fluxo de controle e nome de identificadores. Alguns trabalhos também mostram que elementos externos, como ferramentas de programação, podem ser um fator de influência (DIAS et al., 2020), (SHARGABI et al., 2020). A Figura 3, apresenta uma linha do tempo sobre os trabalhos mapeados nessa seção.

Esse fatores de influência, podem ser considerados em alguns trabalhos como fatores de qualidade do código, como é o caso da estrutura do código e da nomenclatura do identificador (SCHANKIN et al., 2018). Os trabalhos considerados nessa seção mostram que as formas de nomeação de variáveis, métodos e funções têm um grande impacto na compreensão do código. Assim como, as ferramentas de desenvolvimento podem fornecer suporte a visualização geral do software o que pode proporcionar uma colaboração útil para a tarefa de compreensão.

Outros fatores que podem impactar a compreensão são elementos ligados à *readability e legibility* do código. Um dos estudos incluídos nesse referencial teórico mostra uma série desses fatores (SANTOS; GEROSA, 2018). Trabalhos, ligados a *readability e legibility*, embora sejam importantes ainda não apresentam um conceito bem definido sobre o que é *readability e legibility*, como mostra o estudo de Oliveira et al. (OLIVEIRA et al., 2020). Portanto, estudos mais profundos sobre os aspectos de legibilidade de código são necessários. Os próximos capítulos discutem o conhecimento existente sobre esse conceito.

Figura 3 – Linha do tempo de trabalhos que estudaram fatores de influência para a compreensão de softwares.



Fonte: Elaborado pelo autor (2020)

2.3 ENGENHARIA DE SOFTWARE BASEADA EM EVIDÊNCIA

Segundo Mafra e Travassos (MAFRA; TRAVASSOS, 2006), o estado da arte sobre a indústria de software encontra-se atrasado porque frequentemente a indústria de software é acometida por anúncios de "cura" para os mais diversos problemas, isso demonstra a imaturidade desse setor industrial. Com isso, surgem dúvidas como em quais tecnologias investir, como buscar fundamentação para auxiliar nas escolhas ou como saber a procedência da solução. A engenharia de software baseada em evidência pode ser uma maneira de obter essas respostas.

Kitchenham et al. (KITCHENHAM; DYBA; JORGENSEN, 2004), apresenta o conceito de engenharia de software baseada em evidência como um recurso útil para fornecer uma síntese de trabalhos científicos sobre um determinado tópico. Os autores afirmam que a abordagem de engenharia de software baseada deve:

> "fornecer os meios pelos quais as melhores evidências atuais de pesquisa possam ser integradas à experiência prática e aos valores humanos no processo de tomada de decisão em relação ao desenvolvimento e manutenção de software." (KITCHENHAM; DYBA; JORGENSEN, 2004)

Esse estudo, apresenta evidências de como a área de engenharia de software pode se beneficiar das práticas de pesquisa baseadas em evidências. Entre esses benefícios, estão: o fornecimento de objetivos comuns para pesquisadores, ou seja, objetivos que sejam direcionados à indústria e outros *stakeholders* e um meio para que profissionais da indústria possam embasar suas decisões.

Adicionalmente, para Kitchenham et al. (KITCHENHAM; BUDGEN; BRERETON, 2015), engenharia de software baseada em evidência faz uso de procedimentos sistemáticos e bem definidos que fornece um meio apropriado para vincular a experiência ao conhecimento, e também para abordar a natureza não determinística das atividades de engenharia de software. Kitchenham et al. (KITCHENHAM; BUDGEN; BRERETON, 2015) mostram que o rigor empregado na condução de estudos baseados em evidencia é um dos pontos fortes desse tipo de trabalho.

Aliado a isso, Mafra e Travassos (MAFRA; TRAVASSOS, 2006), afirmam que a engenharia de software baseada em evidência faz uso de dois tipos de estudos para atingir um nível adequado de evidências sobre uma determinada tecnologia, sendo eles:

- Estudos Primários, que são estudos que visam caracterizar uma determinada tecnologia em uso dentro de um contexto específico. Nessa categoria são encontrados os experimentos, *surveys* e os estudos de caso;
- Estudos Secundários, que visam identificar, avaliar e interpretar todos os resultados relevantes a um determinado tópico de pesquisa, fenômeno de interesse ou questão

de pesquisa. Nessa categoria são encontradas as revisões sistemáticas e os mapeamentos.

Dentro desse cenário, a revisão sistemática da literatura é um método que faz uso de estudos secundários ou terciários e tem como objetivo pesquisar o conhecimento disponível sobre determinado assunto. Através desse método é possível sintetizar os conhecimentos existentes e encontrar com isso respostas para determinados questionamentos, assim como, áreas para potenciais estudos e lacunas de conhecimento. Assim, pode-se definir uma revisão sistemática como um estudo que busca encontrar respostas para um determinado questionamento utilizando uma análise sistemática de estudos científicos (KITCHENHAM; BUDGEN; BRERETON, 2015).

Uma revisão sistemática da literatura diferente de uma revisão *ad hoc* pode fazer uso de uma abordagem qualitativa como processo de pesquisa, envolvendo ainda uma abordagem de pesquisa interpretativista e uma análise temática como método de análise de dados (WOHLIN; AURUM, 2015). Kitchenham et al. (KITCHENHAM et al., 2009), afirma que uma revisão sistemática não tem como objetivo apenas agregar trabalhos sobre um determinado tema, essa abordagem tem também o objetivo de apoiar a construção de diretrizes baseadas em evidências.

Para Brereton et al. (BRERETON et al., 2007), o crescente número de trabalhos empíricos na área de engenharia de software faz necessária a adoção de uma abordagem sistemática para sintetizar o conhecimento existente gerado por esses trabalhos. Em seu trabalho, os autores relatam, através de suas experiências com a aplicação da abordagem de revisão sistemática, os pontos onde os processos de condução podem ser adaptados para o domínio da Engenharia de Software, assim como melhorias na infraestrutura e nas práticas da Engenharia de Software para aumentar a aplicabilidade deste método. Inicialmente, para condução de uma revisão sistemática, os autores apresentam uma estrutura guiada por três fases, são elas: o planejamento; condução da revisão; e relato da revisão. A fase de planejamento tem como objetivo a definição das perguntas de pesquisa, o desenvolvimento do protocolo da revisão e a validação do protocolo construído. A fase de condução da revisão, tem como objetivo identificar as pesquisas relevantes, selecionar os estudos primários, avaliar a qualidade desses estudos, extrair os dados necessários e sintetizar esses dados. A fase de relato da revisão tem como objetivo escrever o relatório da revisão e validar o relatório.

Além disso, Brereton et al. (BRERETON et al., 2007), apresentam uma série de recomendações que podem auxiliar na condução de uma revisão. Dentre elas, as mais importantes correspondem à definição de uma pergunta de pesquisa e a definição de um protocolo de pesquisa que é mostrado como algo fundamental para condução de uma pesquisa com esse método. Outra recomendação dos autores é que todos os pesquisadores envolvidos na pesquisa estejam também envolvidos na construção do protocolo da revisão. Os au-

tores acreditam ainda que uma validação formal do protocolo pode ser benéfica, porém investigações ainda são necessárias para comprovar essa crença.

Assim como outros métodos de pesquisa, existem alguns problemas que podem impactar a condução de uma revisão sistemática. Um deles é quanto às bases de dados escolhidas, pois muitas delas não oferecem um suporte satisfatório para pesquisas sistemáticas, como por exemplo o *google scholar*. Outro problema é quanto aos trabalhos indexados nas bases de dados. Por exemplo, um número considerável de trabalhos não fornecem um resumo de boa qualidade, isso inviabiliza que pesquisadores definam, a partir desse elemento, a relevância do trabalho para a pesquisa conduzida (BRERETON et al., 2007).

Apesar dessas limitações, a engenharia de software baseada em evidência é um excelente recurso para fornecer um conhecimento através de uma síntese de trabalhos existentes. Essa abordagem faz uso principalmente de estudos primários e secundários e o método mais popular para sintetizar esses estudos é o da revisão sistemática da literatura. Sobre esse método, na área da Engenharia de Software, diferente da área de ciências médicas, ainda são poucos os trabalhos que fazem uso desse método. Kitchenham et al. (KITCHENHAM; DYBA; JORGENSEN, 2004) (KITCHENHAM et al., 2009) e Brereton et al. (BRERETON et al., 2007) fornecem uma série de informações para condução de uma revisão sistemática na área de engenharia de software. O trabalho de Kitchenham et al. (KITCHENHAM et al., 2009) ainda apresenta um conjunto de lições que podem ajudar outros pesquisadores na utilização deste método.

3 METODOLOGIA

O método científico é o conjunto de processos ao qual uma pesquisa científica passa até ser concluída. Ele pode envolver ferramentas, processos e observações e tem como principal objetivo tornar a pesquisa confiável e possível de ser reproduzida por outros pesquisadores (KAUARK; MANHÃES; MEDEIROS, 2010). Este capítulo, apresenta o método científico utilizado na condução dessa pesquisa. São apresentados a classificação da pesquisa, e em seguida as etapas para conclusão da pesquisa que é composta pela condução de uma revisão bibliográfica e condução de uma revisão sistemática da literatura.

3.1 CLASSIFICAÇÃO DA PESQUISA

O método utilizado na condução dessa pesquisa pode ser caracterizado como de abordagem indutiva, uma vez que infere-se uma verdade através da dados particulares (MARCONI; LAKATOS, 2004). As variáveis desta pesquisa são de natureza quantitativa uma vez que métodos estatísticos foram utilizados para construção dos resultados e qualitativas por existir uma análise dissertativa sobre os resultados. Essa pesquisa também é caracterizada enquanto ao seu objetivo como exploratória e descritiva. Um método fundamental para a conclusão dessa pesquisa foi o de revisão sistemática da literatura.

A abordagem indutiva segundo Marconi e Lakatos (MARCONI; LAKATOS, 2004) é caracterizada como um processo mental que se utiliza de dados para dedução de verdades. O argumento indutivo está baseado em premissas verdadeiras que podem levar a conclusões também verdadeiras. Nessa abordagem, primeiro é feita uma observação dos fenômenos para que uma análise mais aprofundada possa ser construída, em seguida, encontra-se as relações entre o fenômeno observado e outros fatos e, por último, é construída a generalização das relações encontradas, ou seja, as respostas encontradas são generalizadas para fatos semelhantes.

A pesquisa quantitativa é construída através de interpretações de fenômenos e atribuição de significados. Consequentemente, essa abordagem faz uso da indução e é totalmente descritiva. O foco de pesquisas que fazem uso dessa abordagem estão nos processos e seus significados. A pesquisa qualitativa, por sua vez, considera que opiniões e informações podem ser quantificáveis, ou seja, traduzida em números. Essa abordagem utiliza como base técnicas estatísticas para classificar e analisar os dados da pesquisa (KAUARK; MANHÃES; MEDEIROS, 2010).

Pesquisas exploratórias tem como objetivo estudar tópicos ou assuntos que não estão claramente definidos a fim de criar uma familiaridade com o problema. Por outro lado, pesquisas descritivas, geralmente, envolvem o relacionamento entre as variáveis estudadas com o objetivo de apresentar os aspectos do objeto de estudo (KAUARK; MANHÃES;

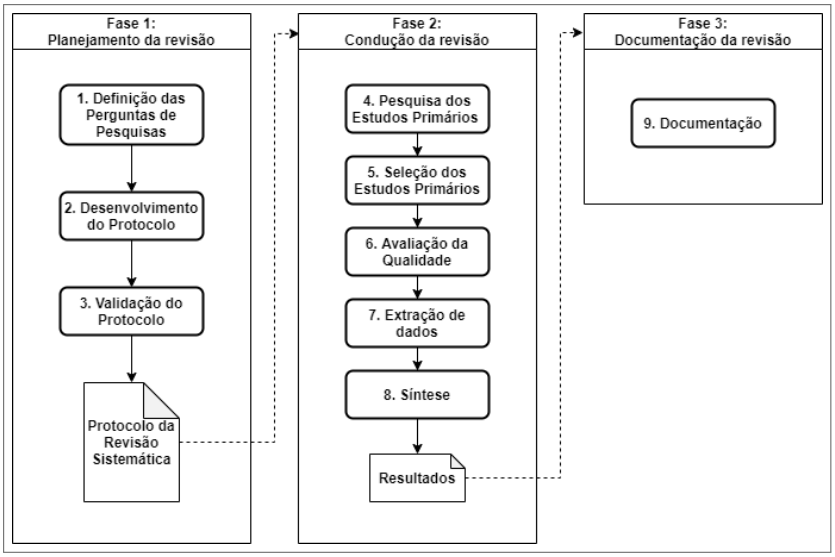
MEDEIROS, 2010).

Uma revisão sistemática deve ser conduzida seguindo um planejamento prévio contido em um protocolo. O objetivo principal desse planejamento é tornar o trabalho replicável e auditável, ou seja, que tenha de fato um valor científico agregado. Em caso contrário, tornam-se uma revisão informal e dependente de seus respectivos criadores, isso leva por consequência a diminuição da confiabilidade do estudo (KITCHENHAM; BUDGEN; BRERETON, 2015).

3.2 ETAPAS DA PESQUISA

Essa pesquisa seguiu as recomendações de Kitchenham, B. at al (KITCHENHAM; BUDGEN; BRERETON, 2015). As etapas que foram conduzidas para conclusão da pesquisa podem ser encontradas na Figura 4.

Figura 4 – Fases da Pesquisa



Fonte: Adaptado de (KITCHENHAM; BUDGEN; BRERETON, 2015)

Essa pesquisa se iniciou com a fase de planejamento da revisão sistemática. Essa fase foi composta por três etapas. A primeira corresponde a definição das perguntas de pesquisa. As quatro perguntas de pesquisas definidas nessa fase podem ser encontradas na Tabela 1. Essas perguntas foram elaboradas a partir da problemática definida para a condução deste trabalho. A segunda etapa consistiu na construção do protocolo de condução da revisão sistemática. Um protocolo de revisão sistemática é um documento que contém todos os passos que serão conduzidos em uma revisão sistemática. Nesse documento, é possível encontrar as perguntas que norteiam a revisão, as estratégias de pesquisa utilizadas assim como as strings de busca. Também é possível encontrar nesse documento informações sobre a etapa de seleção dos estudos, extração de dados e síntese dos dados, o protocolo utilizado na condução desta revisão pode ser encontrado no Apêndice A. A terceira etapa

foi caracterizada pela validação do protocolo de pesquisa, essa validação foi conduzida pelos quatro pesquisadores envolvidos nessa pesquisa. Ao final desta etapa, obteve-se como resultado um protocolo contendo informações para a condução das próximas etapas da revisão sistemática.

Tabela 1 – Perguntas de Pesquisa da Revisão Sistemática

ID	Pergunta de Pesquisa
RQ1	Quais construções de programação e expressões idiomáticas de codificação melhoram a legibilidade do código em tarefas de compreensão do programa?
RQ2	Quais elementos de estilo de codificação melhoram a legibilidade do código nas tarefas de compreensão do programa?
RQ3	Como os estudos centrados no ser humano avaliam se uma determinada abordagem é mais legível do que outra?
RQ4	Em estudos centrados em ferramentas, como a *readability* e a *legibility* são avaliadas?

Fonte: Elaborado pelo autor (2020)

A segunda fase desta pesquisa foi a condução da revisão. Essa condução teve cinco etapas em sua composição. A primeira correspondeu a pesquisa dos estudos primários. Em síntese, essa fase foi responsável por encontrar os trabalhos que abordam a temática pesquisada. A segunda etapa corresponde a seleção dos estudos primários essa fase tem como objetivo a aplicação dos critérios de inclusão e exclusão estabelecidos no protocolo da revisão. A terceira etapa corresponde à avaliação da qualidade dos estudos, essa etapa teve como objetivo a avaliar a qualidade dos estudos incluídos na revisão. A quarta etapa dessa fase foi responsável pela extração dos dados relevantes para responder às perguntas de pesquisa estabelecidas neste trabalho. A última etapa desta segunda fase foi responsável pela síntese dos resultados extraídos.

Em seguida, a terceira fase dessa pesquisa se caracterizou pela documentação dos resultados encontrados nesta pesquisa. Alguns dos resultados iniciais foram publicados no International Conference on Software Maintenance and Evolution (ICSME) (OLIVEIRA et al., 2020). Os resultados apresentados nesta dissertação correspondem ao somatório dos resultados publicados no ICSME e outros resultados encontrados. As seções a seguir explicam mais profundamente as fases da etapa de condução da revisão sistemática.

3.3 BUSCA DOS ESTUDOS PRIMÁRIOS

A fase de busca é uma das fases que compõem o conjunto dos primeiros passos para condução de uma revisão sistemática. Essa etapa é responsável por obter os estudos

primários relevantes para a pesquisa. Pode ser conduzida através de alguns métodos como, por exemplo: Busca manual, busca automática ou processo de busca através das referências dos artigos citados (KITCHENHAM; BUDGEN; BRERETON, 2015). Para essa pesquisa, a estratégia de busca adotada foi composta por três partes. A primeira consistiu de uma busca manual para reunir os estudos principais, a segunda caracterizou-se pela definição de uma string de busca genérica e a terceira e última foi caracterizada pela busca automática nos mecanismos de busca.

Na primeira estratégia, foi realizado uma busca manual de estudos com o objetivo de validar se o problema abordado é relevante para a comunidade de engenharia de software e também para encontrar termos para a definição de uma string de pesquisa genérica, esse conjunto de estudos recebe o nome de estudos sementes. Para encontrar esses trabalhos, foram realizadas pesquisas ad hoc nas seguintes conferências de engenharia de software: International Conference on Software Engineering (ICSE), Symposium on the Foundations of Software Engineering (FSE), International Conference on Mining Software Repositories (MSR), International Conference on Automated Software Engineering (ASE), International Symposium on Software Testing and Analysis (ISSTA), Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), International Conference on Software Maintenance and Evolution (ICSME), International Collegiate Programming Contest (ICPC) e International Conference on Software Analysis, Evolution and Reengineering (SANER).

Essa estratégia é também recomendada como forma de validar a completude da fase de pesquisa. Segundo Kitchenham et al. (KITCHENHAM; BUDGEN; BRERETON, 2015), definir um parâmetro de completude é importante para determinar quando a busca deve ser interrompida. Uma das recomendações é que esse parâmetro seja baseado em um grupo de artigos que são conhecidos previamente pelos pesquisadores. Sendo assim, foram analisados os artigos publicados nessas conferências durante o período de 2016 a junho de 2019 com o objetivo de selecionar os artigos que respondessem às perguntas de pesquisa deste trabalho e para capturar termos para construção da string de busca a ser utilizada na fase de busca automática. Essa análise começou em 2016 porque foi estabelecido como critério que esse conjunto de artigo deveria ser composto por recentes estudos. Também foi incluído um artigo do TSE (BUSE; WEIMER, 2010) que era conhecido de antemão por dois dos pesquisadores envolvidos nessa revisão sistemática da literatura. Isso aconteceu porque esse artigo era relevante para nossa pesquisa. Esse processo resultou em 13 artigos semente.

A segunda estratégia de pesquisa foi caracterizada pela busca automática nos mecanismos de busca avançados das bases de dados da Association for Computing Machinery (ACM) (ACM..., http://dl.acm.org/), Institute of Electrical and Electronics Engineers (IEEE) (IEEE..., http://ieeexplore.ieee.org/) e Scopus (SCOPUS, http://www.scopus.com/). Para esse segunda estratégia foi utilizado uma string de busca construída a partir da aná-

lise dos títulos e palavras chaves dos artigos do conjunto de trabalhos sementes. Foram escolhidos os termos gerais com o propósito de capturar o maior número de artigos que respondessem às perguntas de pesquisa deste trabalho. Em caso contrário, acredita-se que poderia ocorrer uma delimitação do conjunto de artigos com base em tópicos específicos. Os termos utilizados para construir a string de busca, são:

$$Title(ANY(terms))\ OR\ Keywords(ANY(terms)),$$
where $terms$ = { “code comprehension”, “code understandability”,
“code understanding”, “code readability”,
“program comprehension”, “program understandability”,
“program understanding”, “program readability”,
“programmer experience” }

Embora esse trabalho esteja interessado em estudar aspectos ligados a legibilidade, não foi incluído o termo “Legibility” na string de busca porque nos mecanismos de busca utilizados a maioria dos artigos retornados com essa palavra no título, resumo ou palavras-chave estava relacionada à outras áreas de pesquisa, entre elas linguística e linguística computacional. Consequentemente, acredita-se que esse termo é utilizado com significados diferentes do que é esperado em um artigo de Engenharia de Software. Portanto utilizá-lo causaria um efeito de inchamento no número de falso positivo. Além desse termo, também não foram incluídos os termos com "*software*", por exemplo "*software readability*", "*software comprehension*"porque a palavra *software* remete a aspectos mais amplo do que apenas ao código-fonte.

Devido às diferenças existentes nos mecanismos de busca utilizados a string de pesquisa genérica foi adaptada para cada mecanismo. Sendo assim, as strings utilizadas nas bases de dados foram:

ACM:

$$Title(ANY(terms))\ OR\ keywords.author.keyword(ANY(terms)),$$
where $terms$ = { acmdlTitle: (“code comprehension” “code understandability“ “code understanding“ “code readability“ “program comprehension“ “program understandability“ “program understanding“ “program readability“ “programmer experience“ “readability“) OR keywords.author.keyword: (“code comprehension“ “code understandability“ “code understanding“ “code readability“ “program comprehension“ “program understandingability“ “program understanding“ “program readability“ “programmer experience“ “readability“) }

IEEE:

$$Title(ANY(terms))\ OR\ Keywords(ANY(terms)),$$
where $terms$ = { ((“Document Title“: “code comprehension“ OR “code understandability“ OR “code understanding“ OR “code readability“ OR “program comprehension“ OR “program understandability“ OR “program understanding“ OR “program readability “OR“ programmer experience “OR“ readability “OR (“ Author Keywords “:“ code comprehension “OR“ code

understandability “OR“ code understanding “OR“ code readability “OR“ program comprehension “OR“ program understandability “OR“ program understanding “OR“ program readability “OR“ programmer experience “OR“ readability “)) }

SCOPUS:

$$Title(ANY(terms))\ OR\ Keywords(ANY(terms)),$$

where $terms$ = { TITLE (“programmer experience“) OR TITLE (“program comprehension“) OR TITLE (“code comprehension“) TITLE (“code readability“) OR TITLE (“code understandability“) OR KEY (“code comprehension“)) OR KEY (“program comprehension“) OR KEY (“program understandability“) OR KEY (“readability“)}

Com isso, recuperamos 1,926 documentos na base de dados da ACM (ACM..., http://dl.acm.org/), 729 na base de dados do IEEE Explore (IEEE..., http://ieeexplore.ieee.org/) e 1,909 na base de dados do Scopus (SCOPUS, http://www.scopus.com/), em 2 de setembro de 2019. Como um determinado documento pode ser recuperado por mais de um mecanismo de busca, as saídas dos motores foram unificadas para eliminar duplicatas, isso resultou em 2,843 documentos. Como recomendado por (KITCHENHAM; BUDGEN; BRERETON, 2015) essa string de busca retorna todos os 13 documentos incluídos no conjunto semente.

3.4 ANÁLISE DOS ESTUDOS

Após a fase de pesquisa, que tem como objetivo encontrar trabalhos relevantes para a revisão sistemática da literatura, deve-se conduzir uma análise com o objetivo de identificar quais são os trabalhos relevantes para responder às perguntas de pesquisa do trabalho (KITCHENHAM; BUDGEN; BRERETON, 2015). Sendo assim, essa seção apresenta o processo conduzido para análises dos estudos retornados através das buscas realizadas na fase anterior.

3.4.1 Exclusão dos Estudos

Nessa fase do processo de análise todos os 2,704 documentos retornados nas buscas foram analisados com o objetivo de remover todos os trabalhos que não seriam relevantes para o estudo. Para isso foram definidos alguns critérios de exclusão listados abaixo:

- CE1: O artigo está fora do escopo deste estudo. Não é principalmente relacionado à compreensão do código-fonte, *readability* ou *legibility*, não envolve qualquer comparação de diferentes maneiras de escrever programas, nem diretos nem indiretos, ou é claramente irrelevante para as nossas questões de pesquisa. Por exemplo, excluímos estudos com foco em design de alto nível, documentação e dependências (problemas de nível superior). A motivação para esse critério foi excluir estudos que estão nitidamente distantes do objeto de pesquisa;

- CE2: O estudo não é um artigo completo, ou seja, *short papers* e literatura cinza (por exemplo, dissertações de mestrado, teses de doutorado, monografias de conclusão de curso, artigos resumidos) ou não está escrito em inglês. Como regra geral, consideramos que os artigos completos devem ter pelo menos 5 páginas. A motivação para esse critério foi excluir artigos que poderiam ser estendidos futuramente e materiais que não tivessem uma validação científica;

- CE3: O estudo é sobre métricas de legibilidade sem uma avaliação experimental. A motivação para esse critério foi excluir artigos que não envolvesse seres humanos;

- CE4: O estudo é sobre ajudas de compreensão de programa, como visualizações ou outras formas de análise ou ajudas sensoriais (por exemplo, gráficos, execução baseada em traços, resumo de código, mineração de especificações, engenharia reversa). A motivação para esse critério foi excluir trabalhos que não focassem em fatores do código fonte;

- CE5: O estudo se concentra na acessibilidade, por exemplo, visa indivíduos com deficiências visuais ou desenvolvedores neuros diversos. A motivação para esse critério também foi excluir trabalhos que não focassem em fatores do código fonte.

Para análise dos 2,704 documentos e aplicação dos critérios de exclusão foram analisados o título e o resumo do documento, e quando necessário em alguns casos a metodologia. Os documentos que não atenderam a nenhum dos critérios de exclusão foi aceito para entrar na próxima fase. Os documentos que atenderam um ou mais critérios de exclusão passaram por uma segunda análise por um diferente pesquisador envolvido na nesta pesquisa. Essa medida foi adotada com a intenção de reduzir a ameaça de descartar potenciais estudos para pesquisa. Ao final dessa fase, todos os artigos que atenderam a no mínimo um critério de exclusão foram descartados. O total de documentos no final dessa fase foi de 524 documentos.

3.4.2 Inclusão dos estudos

Após eliminar todos os documentos irrelevantes para essa pesquisa os 524 documentos foram analisados novamente dessa vez quanto a sua inclusão no estudo. Essa etapa, semelhante a anterior teve como objetivo descartar aqueles documentos que não eram relevantes para a pesquisa, mas especificamente descartar os trabalhos que não respondem às nossas perguntas de pesquisa. Isso posto, foram aplicados os seguintes critérios de inclusão:

- CI1 (Escopo): O estudo deve estar relacionado principalmente aos tópicos de compreensão de código, *readability*, *legibility* ou código difícil de entender;

- CI2 (Metodologia): O estudo deve ser/conter pelo menos um estudo empírico, como experimento controlado, quase-experimento, estudo de caso ou pesquisa envolvendo seres humanos;
- CI3 (Comparação): O estudo deve comparar alternativas construções de linguagem de programação, expressões idiomáticas de codificação ou estilos de codificação em termos de *readability* ou *legibility* do código;
- CI4 (Granularidade): O estudo deve ter como alvos elementos de programa de baixa granularidade e atividades de programação de baixo nível/escopo limitado. Não design ou comentários, mas implementação.

A aplicação desses critérios de inclusão em um artigo, muitas vezes exigiu a leitura não apenas do título e resumo, mas também das seções de introdução, metodologia e conclusões. Isso porque os critérios de inclusão definidos, muitas vezes não eram encontrados na seção de resumo. Todos os trabalhos que violaram um ou mais dos critérios de inclusão foram anotados com "não aceitável". Semelhante à fase de exclusão, todos os artigos analisados nessa fase passaram por dupla análise envolvendo em cada uma delas diferentes pesquisadores. Para avaliar a concordância dos pesquisadores, foi calculado o coeficiente Kappa (COHEN, 1960). Nesse cálculo, foi obtido como resultado um coeficiente de k = 0,323, que é considerado uma concordância razoável (KITCHENHAM; BUDGEN; BRERETON, 2015). No final dessa etapa, todos os trabalhos marcados com "aceitável", nas duas análises, passaram para a próxima etapa, assim como todos os trabalhos marcados com "não aceitável" foram removidos da revisão. Todos os outros casos, por exemplo, um trabalho marcado em uma análise com "aceitável" e em outra análise com "não aceitável" foram discutidos por todos os pesquisadores envolvidos na pesquisa, em reuniões ao vivo para que o consenso fosse definido. Essa etapa resultou em 56 artigos.

3.4.3 Avaliação da Qualidade dos Estudos

Além da etapa de busca e aplicação dos critérios de exclusão e inclusão também foi realizada uma análise referente a qualidade dos artigos. Essa etapa segundo Kitchenham et al. (KITCHENHAM; BUDGEN; BRERETON, 2015) é responsável por avaliar os estudos quanto a sua validade e viés dos resultados. O objetivo dessa etapa era identificar e remover trabalhos que continham baixa qualidade. Para essa avaliação foi produzido um questionário, utilizando a ferramenta do google formulário, composto por nove questões. As questões, desse questionário, foram adaptadas do trabalho de Keele (KEELE et al., 2007) e foi respondido para os 54 artigos.

Esse questionário foi formado por três questões referentes ao desenho do estudo, essas perguntas tinham como propósito capturar o objetivo central do estudo e as variáveis (dependente e independentes) medidas. Quatro questões sobre a análise e rigor aplicado no

estudo, o objetivo dessas perguntas era avaliar o aspectos ligados ao método utilizado para condução do estudo. Duas questões referente as conclusões do trabalho, essas perguntas tinham como objetivo analisar aspectos ligados a como as conclusões foram reportadas.

Essas questões podem ser encontradas na Tabela 2 ou no Apêndice A. Para cada uma das questões existiram três opções de resposta, sendo elas:

- Sim (1): Deve ser pontuado com esse valor caso o artigo atenda totalmente ao critério;
- Parcialmente (0,5): Deve ser pontuado com esse valor caso o artigo atenda parcialmente ao questionamento;
- Não (0): Deve ser pontuado quando o artigo não atende ao questionamento.

Esses valores foram utilizados para calcular a pontuação de cada artigo. Essa pontuação indica se o artigo deveria ser removido da revisão ou se prosseguiria para próxima etapa. O valor máximo é, portanto, 9. Se a pontuação de um artigo for 0,5 ou melhor em todas as questões, sua pontuação geral será 4,5 ou mais. Assim, definimos que um artigo deveria ter pontuação mínima de 4,5 para ser mantido em nossa lista de artigos.

Tabela 2 – Questões de avaliação da qualidade para estudos quantitativos

ID	**Perguntas para avaliação de qualidade do estudo**
QA1	Os objetivos estão claramente definidos?
QA2	As variáveis independentes estão definidas de forma adequada?
QA3	As variáveis dependentes estão definidas de forma adequada?
QA4	Os métodos de coleta de dados estão descritos adequadamente?
QA5	Os participantes do estudo ou unidades de observação estão adequadamente descritos? Por exemplo, experiência em ES, tipo (aluno, profissional, consultor), nacionalidade, experiência em tarefas e outras variáveis relevantes.
QA6	Os dados coletados foram descritos adequadamente (por exemplo, estatísticas descritivas)?
QA7	O estudo empregou métodos estatísticos para analisar os dados?
QA8	Existe uma declaração clara de resultados, ou seja, todas as perguntas do estudo foram respondidas?
QA9	Os pesquisadores explicam as ameaças à validade do estudo?

Fonte: adaptado de Keele et al. (2007)

Nessa fase, cada artigo foi avaliado por um dos pesquisadores envolvidos na pesquisa. Para mitigar a possibilidade de avaliações errôneas e alinhar o entendimento de todos os pesquisadores envolvidos, foi realizada uma rodada de teste. Essa rodada consistiu em uma reunião ao vivo, onde cada pesquisador apresentou uma avaliação da qualidade de um trabalho e justificou aos demais pesquisadores a pontuação para cada questão. Nesta etapa os valores gerais foram mín = 5, mediana = 8, máx = 9. Portanto, nenhum artigo foi removido. A lista dos artigos incluídos nessa revisão sistemática pode ser encontrada em <https://github.com/reydne/code-comprehension-review>.

3.4.4 Detalhes Adicionais

No conjunto de artigos utilizados nesta pesquisa alguns artigos não foram capturados pela *string* de busca, mas são artigos relevantes para essa pesquisa (POSNETT; HINDLE; DEVANBU, 2011), (BENANDER; BENANDER; PU, 1996), (Kleinschmager et al., 2012), (STEFIK; SIEBERT, 2013). Esses artigos não foram retornados pela *string* de busca porque ela não os abrange. Esses artigos também não entraram no conjunto semente porque nenhum dos pesquisadores envolvidos nessa pesquisa os conheciam. Alguns desses artigos foram encontrados durante a leitura dos artigos incluídos na fase de seleção final, ou seja, na etapa de avaliação de qualidade. Esses trabalhos, portanto, podem ser considerados como trabalhos relacionados aos que foram incluídos durante o processo de triagem.

Também foi incluído um estudo que assim como esses não foi retornado na busca e que não estava sendo citado por nenhum outro artigo já incluído (SCANNIELLO; RISI, 2013). Esse artigo foi então incluído por ser conhecido por um dos pesquisadores envolvidos na pesquisa. Após a identificação desses trabalhos, as buscas não foram realizadas novamente porque os refinamentos nas strings de busca e o critério de completude serviram como estratégias para mitigar esse tipo de limitação. Todos esses estudos incluídos foram discutidos por todos os pesquisadores em reuniões ao vivo. E em seguida, esses artigos foram submetidos aos critérios de exclusão e inclusão e avaliação da qualidade. Um artigo é considerado obsoleto caso tenha sido estendido por outro artigo. Para essa pesquisa estão sendo considerados apenas os artigos estendidos ou os últimos a serem publicados.

3.5 EXTRAÇÃO DE DADOS

A etapa de extração de dados foi caracterizada pela coleta de informações que contribuíssem para responder às perguntas de pesquisa do estudo. Essa etapa foi apoiada por um formulário contendo dezessete questões. Entre elas, nove destinavam-se a recolher informações gerais sobre o trabalho, como por exemplo: ano e veículo de publicação. Seis questões eram sobre o método de avaliação, como por exemplo: qual foi a metodologia utilizada no estudo e como as variáveis dependentes estavam sendo avaliadas. Duas ques-

tões eram sobre os resultados do estudo, como por exemplo: o que os autores do estudo afirmaram ter descobertos. O formulário completo pode ser encontrado no Apêndice A.

Nessa etapa, os artigos foram divididos entre os pesquisadores envolvidos nesta pesquisa. Cada artigo foi lido por completo, isso foi feito com o objetivo de recolher as informações necessárias para responder cada pergunta contida no formulário de extração. Eventualmente, alguns artigos não respondiam a algumas das perguntas contidas no formulário de extração, para esses casos a pergunta deveria ser deixada sem resposta.

3.6 SÍNTESE DOS DADOS

Após essa etapa de extração, deve ser conduzida a etapa de síntese que consiste na sumarização, integração, combinação e comparação dos resultados recolhidos através do processo de extração de dados dos artigos primários (KITCHENHAM; BUDGEN; BRERETON, 2015). Pode-se conduzir a síntese através de abordagens quantitativas ou qualitativas, no primeiro caso os dados devem ser utilizados técnicas estatísticas. Neste trabalho, devido à não homogeneidade das obras incluídas, apresentamos uma síntese qualitativa. Nesse tipo de síntese, o objetivo é integrar resultados e conclusões em linguagem natural. Essa etapa, foi então apoiada por ferramentas de tabulação de dados e teve como principal insumo os dados recolhidos na fase anterior a de extração de dados. A próxima seção apresenta o processo conduzido para responder a primeira pergunta de pesquisa deste trabalho.

3.7 REVISÃO (NÃO SISTEMÁTICA) DA LITERATURA

Para responder à primeira pergunta de pesquisa, foi conduzido um levantamento bibliográfico ou como define John (CRESWELL, 2010), uma revisão da literatura. Segundo John (CRESWELL, 2010), essa abordagem é muito utilizada para limitar o escopo da pesquisa e pode transmitir a importância de estudar o tópico. Essa abordagem compõe também uma das primeiras etapas de uma pesquisa. A pesquisa apresentada nesta dissertação, portanto, apresenta duas revisões (não sistemáticas) da literatura, a primeira compõe o referencial teórico sobre compreensão de código, essa foi apresentada nas seções iniciais deste documento (Capítulo 2). A segunda, compõe as evidências para responder a primeira pergunta desta pesquisa (Capítulo 4).

Para construção da segunda revisão da literatura, sobre os conceitos de *readability* e *legibility* na área de Engenharia de Software, foi conduzida uma pesquisa na ferramenta do Google Scholar utilizando as palavras "*Readability*" e "*Legibility*". Essa busca teve como objetivo encontrar trabalhos relevantes para construir o referencial teórico sobre esses assuntos na área de Engenharia de Software. O critério para seleção de um estudo foi guiado pela quantidade de citações recebidas e adequação ao objetivo da pesquisa. Um dos primeiros resultados encontrados foi o trabalho de Almeida et al. (ALMEIDA et al., 2003).

O título desse trabalho foi utilizado para uma outra busca, dessa vez na ferramenta Connected papers[1]. Essa ferramenta faz uma busca por artigos relacionados e os organiza baseado em um grafo conectado onde os nós representam os artigos e as arestas sua similaridade com outros artigos. É importante destacar que a ferramenta não gera uma árvore de citações. Isso é útil para encontrar e explorar artigos relevantes em uma área de estudos que estão relacionados a um artigo semente. Os resultados dessa pesquisa foram analisados com base em sua relevância para a pesquisa. Ao final dessa análise, foram coletados um total de 7 trabalhos para serem lidos e sintetizados no levantamento bibliográfico.

A estratégia realizada para encontrar trabalhos relevantes nas áreas de Educação, Linguística, Design e Interação Humano-Computador foi semelhante à estratégia discutida nos parágrafos acima. Inicialmente as palavras “readability” e “legibility” foram utilizadas junto a outras palavras referentes a cada área, por exemplo, "text readability", "text legibility"em buscas no Google Scholar. Nesse processo, os trabalhos selecionados também seguiram os critérios de número de citações. Em seguida, os títulos e resumos foram analisados e os trabalhos relevantes para a pesquisa foram incluídos no levantamento bibliográfico. O artigo mais recente foi utilizado para conduzir uma busca no Connected papers. Nesta etapa, o processo de seleção conduzido foi semelhante ao processo conduzido para encontrar os trabalhos da área de Engenharia de Software, ou seja, para os artigos retornados por essa ferramenta o título e resumo foram lidos e os trabalhos foram excluídos quando não em conformidade com o objetivo desta pesquisa.

[1] Link de acesso para o Connected papers: <https://www.connectedpapers.com/>.

4 RESULTADOS

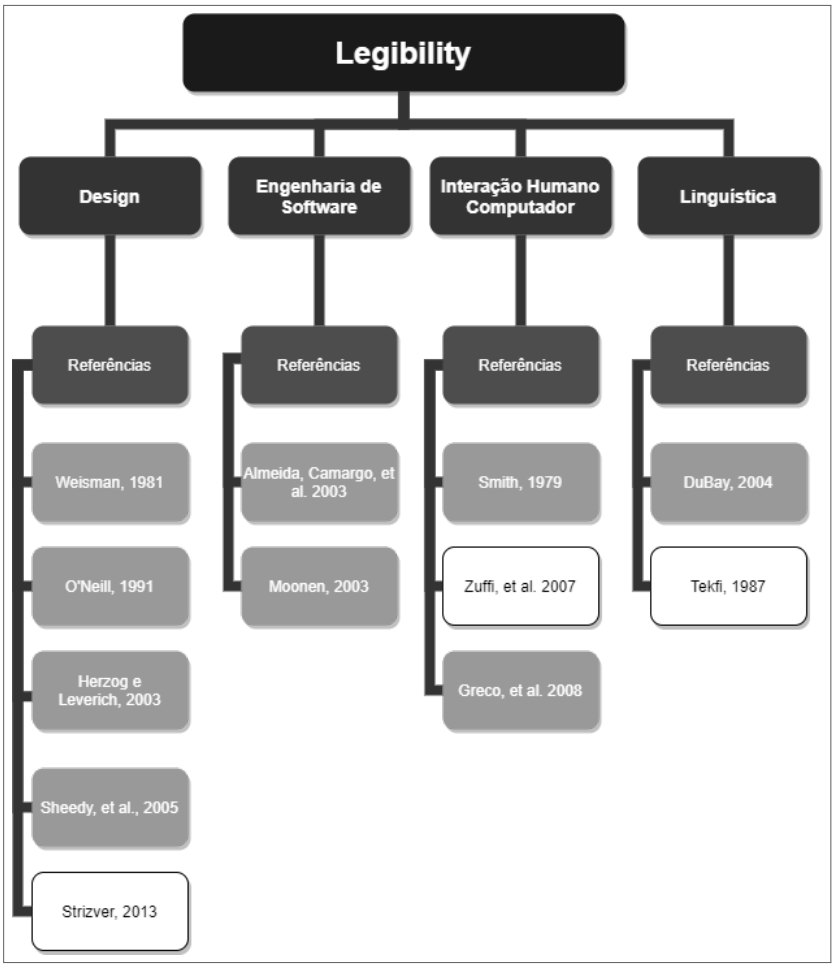
O Capítulo 4 apresenta os resultados da pesquisa que foi conduzida neste trabalho. Esse capítulo está organizado em quatro seções, são elas:

- Na seção 4.1 apresenta os resultados da pesquisa bibliográfica realizada para entender como outras áreas da ciência estudam *legibility* e *readability*. Essa seção apresenta os resultados obtidos nas áreas de Educação, Linguística, Design e Interação Humano-Computador comparados aos resultados encontrados na área de Engenharia de Software.
- A seção 4.2 mostra os resultados para a segunda pergunta de pesquisa (seção 1.3) e foi dividida em duas subseções. Na subseção 4.2.1 é apresentado uma análise descritiva sobre os resultados da revisão sistemática realizada. Essa seção apresenta dados como: a quantidade de artigos retornados e aceitos na revisão, o número de trabalhos por ano de publicação, o foco dos estudos, a avaliação da qualidade e outras características quantitativas dessa amostra de trabalhos. Na subseção 4.2.2 é apresentado uma análise das perguntas de pesquisa que conduziram a revisão sistemática da literatura, ou seja, essa seção apresenta as respostas obtidas após a fase de síntese dos resultados extraídos para a segunda questão de pesquisa dessa dissertação.
- A seção 4.3 apresenta uma discussão sobre todos os resultados apresentados. Essa conclusão leva em consideração tantos aspectos qualitativos deste trabalho como aspectos quantitativos.

4.1 RQ1. COMO ENGENHARIA DE SOFTWARE E OUTRAS ÁREAS DO CONHECIMENTO DEFINEM *READABILITY* E *LEGIBILITY*?

Para responder a essa pergunta de pesquisa foi conduzido um levantamento bibliográfico nas áreas de Educação, Linguística, Design, Interação Humano-Computador e Engenharia de Software. O método utilizado e as tomadas de decisão são discutidos no Capítulo 3. A Figura 5 e a Figura 6 apresentam os artigos incluídos neste levantamento. No total foram incluídos 29 trabalhos, entre os quais 31% desses trabalhos são apenas sobre a temática de *legibility*, 58% são apenas sobre a temática de *readability* e apenas 10% sobre as duas temáticas em conjunto. Os artigos que abordam as duas temáticas estão sinalizados sem preenchimento de cor.

Figura 5 – Gráficos de artigos sobre legibility

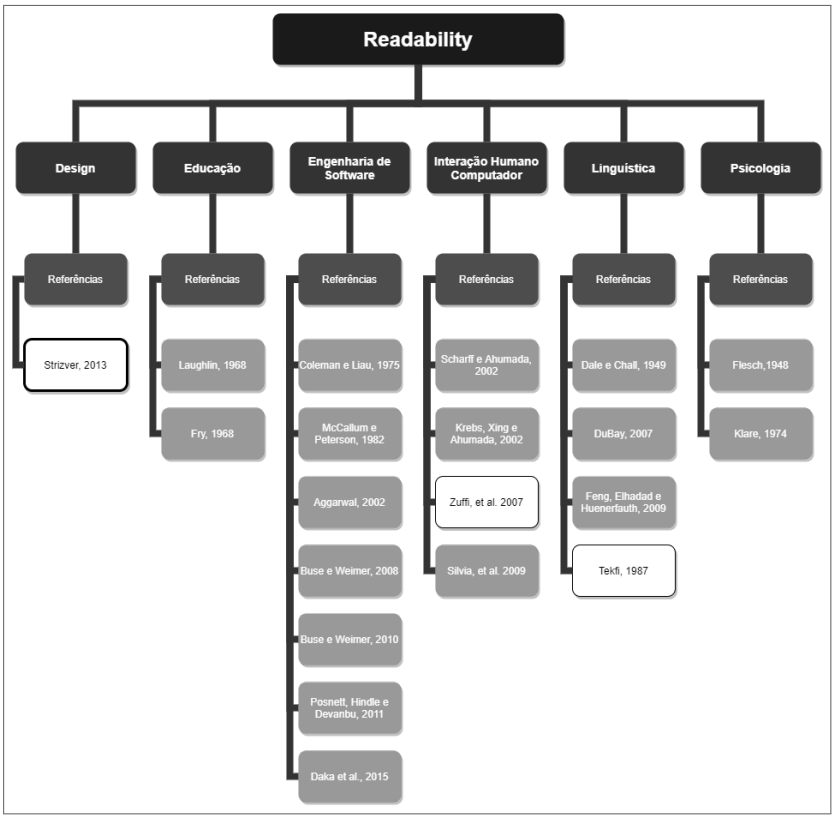


Fonte: Elaborado pelo autor (2020)

4.1.1 Definição na Área de Linguística

Na área de linguística, os conceitos de *readability* e *legibility* são definidos de forma clara e consistente e o conceito de compreensão de texto é semelhante à compreensão de programas em Engenharia de Software. Por exemplo, Gough e Tunmer (GOUGH; TUNMER, 1986) afirmam que "compreensão (não compreensão de leitura, mas compreensão linguística) é o processo pelo qual as informações léxicas (ou seja, palavras), sentenças e discursos são interpretados". No entanto, Hoover e Gough (HOOVER; GOUGH, 1990) aprofundam essa definição e afirmam que "decodificação e compreensão linguística são componentes separados da habilidade de leitura". Essa afirmação destaca a existência de dois processos distintos durante a compreensão do texto: (i) interpretar as palavras/símbolos e (ii) interpretação das sentenças formadas por eles. Essas ideias também podem ser aplicadas na área de Engenharia de Software se considerarmos que a compreensão de código é uma tarefa que abrange: (i) interpretação de símbolos e palavras reservadas da linguagem de programação e (ii) interpretação do conteúdo das estruturas de código, como o conteúdo das funções.

Figura 6 – Gráficos de artigos sobre readability



Fonte: Elaborado pelo autor (2020)

O conceito de *readability*, na área de linguística, corresponde às características que facilitam a compreensão de um texto. Por exemplo, Dale e Chall (DALE; CHALL, 1949) definem essa temática como "o sucesso alcançado pelo leitor ao ler um texto, ou seja, até que ponto o leitor entende o texto". Feng et al. (FENG; ELHADAD; HUENERFAUTH, 2009) utilizam essa mesma definição em seu trabalho e afirmam que "as habilidades de alfabetização dos leitores, suas motivações, conhecimento prévio e outras características internas desempenham um papel importante em determinar se um texto é legível para um determinado grupo de pessoas". Para Hoover e Gough (HOOVER; GOUGH, 1990) a *readability* é o que torna alguns textos mais fáceis de ler do que outros e para Tekfi (TEKFI, 1987) os estudos de *readability* se concentram nos fatores linguísticos como comprimento da palavra e da frase. Em resumo, essas afirmações evidenciam que a *readability* pode ser entendida como uma característica ligada ao conteúdo do texto.

Por outro lado, para Hoover e Gough (HOOVER; GOUGH, 1990) a *legibility* diz respeito à fonte, layout e outros aspectos relacionados à identificação dos elementos no texto. Na mesma linha, para Tekfi (TEKFI, 1987), os estudos de *legibility* estão principalmente preocupados com fatores tipográficos e de *layout*. Quando relacionamos os aspectos da Linguística com a Engenharia de Software, podemos encontrar tanto as características visuais quanto os fatores linguísticos no código-fonte. Por exemplo, Daka et al. (DAKA et al., 2015) afirmam que "a aparência visual do código em geral é referida como sua *rea-*

dability". Os autores se referem claramente à *legibility* (no sentido de design/linguística), mas empregam o termo "*readability*", possivelmente porque é mais frequentemente usado na literatura de Engenharia de Software. Podemos perceber com essas afirmações que a *legibility* na área de linguística está associada a fatores ligados à forma com que o texto se apresenta e que esse conceito é encontrado na Engenharia de Software como *readability*.

Os trabalhos sobre legibilidade, na área de linguística, mostram que os conceitos de *readability* e *legibility* correspondem a aspectos diferentes de um texto, porém ambos estão relacionados. A *readability* está ligada às características de um texto que o tornam mais fácil de ser compreendido. Por outro lado, a *legibility* está ligada às características dos elementos visuais do texto que facilitam a compreensão. Os estudos nessa área têm colaborado tanto para educação de adultos quanto de crianças e adolescentes (DALE; CHALL, 1949). A *legibility* para Tekfi (TEKFI, 1987) pode ser influenciada pelas mudanças no tamanho da fonte, seu peso (grossura dos caracteres), mudanças entre maiúsculas e minúsculas e a mudanças no espaçamento linear. Enquanto a *readability* pode ser impactada pelas habilidades de alfabetização dos leitores, suas motivações, conhecimento prévio e outras características internas do indivíduo (FENG; ELHADAD; HUENERFAUTH, 2009).

4.1.2 Definição em Educação e Psicologia

Existem trabalhos nas áreas de Educação e Psicologia que também discutem a temática de *readability*. Alguns dos trabalhos encontrados estão preocupados em estudar a criação, implementação e avaliação de fórmulas para medir e predizer a *readability* de um texto. Embora esses trabalhos estejam muito próximos aos trabalhos encontrados na área de linguística, eles foram publicados em periódicos da área de educação e psicologia e abordam um contexto referente a essas áreas.

O trabalho de Mc Laughlin (LAUGHLIN, 1969) é um exemplo disso. Para o autor, a *readability* é calculada através de uma medida de dificuldade experimentada pelas pessoas na leitura de um determinado texto. Para Flesch (FLESCH, 1948) a *readability* é definida como "um aspecto ligado à dificuldade de compreensão e a facilidade de leitura". Klare (KLARE, 1974) afirma que as fórmulas existentes calculam *readability* como um aspecto de dificuldade/facilidade de leitura. Através desses trabalhos pode-se perceber que a *readability* nessa área também está conectada com características ligadas a compreensão do conteúdo de um texto.

Existem inúmeras fórmulas para medir a *readability* de um texto, com peculiaridades e focos específicos. Muitas dessas fórmulas são complexas e agregam inúmeras variáveis como comprimento de palavra e comprimento de frase (KLARE, 1974). Na área de Engenharia de Software algumas das métricas propostas para medir a *readability* do código também são complexas e agregam um grande número de variáveis, como é o caso da métrica proposta por Buse e Weimer (BUSE; WEIMER, 2008) que utilizou 19 características do código, entre elas comentários, recuo e tamanho de identificadores. Trabalhos como

esse sugerem que tanto a *readability* de um texto quanto a *readability* do código-fonte de um programa são mensuradas através de um conjunto de variáveis que juntas se referem a compreensão do conteúdo de um texto ou do código.

Os trabalhos das áreas de Educação e Psicologia reforçam a ideia desenvolvida pelos trabalhos na área de Linguística. Esses trabalhos mostram que a *readability* está relacionada aos aspectos que dificultam ou facilitam a compreensão do texto, por exemplo, o número de sentenças e sílabas. As fórmulas para mensurar a *readability* de um texto utilizam esses elementos (KLARE, 1974). Por outro lado, as fórmulas para medir *readability* de um programa utilizam, por exemplo, comprimento da linha (quantidade de caracteres) e comprimento do identificador (BUSE; WEIMER, 2008). Se aplicarmos as percepções obtidas na área de Educação e Psicologia no código-fonte podemos assumir que a *readability* está conectada a quão fácil é compreender o código de um programa.

4.1.3 Definição na Área de Design

Outra área que também apresenta definições sobre *legibility* é a área de design. Weisman (WEISMAN, 1981), por exemplo, afirma que esse conceito pode ser definido como "o grau de capacidade dos usuários de se orientarem no ambiente". O'Neill (O'NEILL, 1991) afirma que a *legibility* arquitetônica "é o grau em que as características projetadas no ambiente ajudam as pessoas a criarem uma imagem mental eficaz ou mapa cognitivo das relações espaciais dentro de uma construção". Herzog et al. (HERZOG; LEVERICH, 2003) colaboram com essa ideia afirmando que "legibilidade se refere a características do ambiente mais amplo que promovem a compreensão, auxiliando na localização e na construção de um mapa cognitivo útil". Esses trabalhos mostram que a *legibility* pode está relacionada com as características visuais de um ambiente.

Outros trabalhos dessa área estudaram aspectos com relação à construção e visualização de textos em monitores. O trabalho de Shendy et al. (SHEEDY et al., 2005), por exemplo, define *legibility* como um "atributo do texto relacionado à sua capacidade de ser devidamente identificado." O trabalho de Strizver (STRIZVER, 2013) afirma que a *legibility* está relacionada com características de design, incluindo altura, formas de caracteres, largura e contraste de traço. Por outro lado, o conceito de *readability* para Shendy et al. (SHEEDY et al., 2005) está relacionado "à forma como o texto é organizado". Embora esses trabalhos estejam bem distantes da área de Engenharia de Software, eles definem *legibility* e *readability* através de conceitos mais claros e distintos.

Para ilustrar a representação da *legibility* e da *readability*, Strizver (STRIZVER, 2013) apresenta alguns exemplos em seu livro. A Figura 7 contém o exemplo de um texto com grau de *legibility* baixo. Nessa imagem, o design tipográfico optou por formas mais elaboradas e expressivas e isso segundo Strizver (STRIZVER, 2013) impacta a facilidade de identificar os elementos no texto. Em seguida, na Figura 8, Strizver (STRIZVER, 2013) apresenta alguns fragmentos de texto em que existe um alto grau de *legibility*. Esses

fragmentos contêm um “design limpo, consistente e descomplicado". Fontes de textos que adotam essas características tendem a conter um alto grau de *legibility* porque elas facilitam a leitura. Com relação a *readability*, Strizver (STRIZVER, 2013) apresenta o exemplo da Figura 9. Nesse exemplo, o autor mostra como a *readability* é afetada pelo tamanho, entrelinhamento, comprimento da linha, alinhamento e espaçamento entre letras e palavras. Embora essas características também estejam ligadas a aspectos visuais, o que é esperado da área de design, elas se relacionam a como o texto está organizado.

Figura 7 – Texto com Baixa Legibility

Fonte: (STRIZVER, 2013)

Os conceitos de *readability* e *legibility* na área de design são semelhantes ao que foi discutido na área da linguística. A *readability* é vista como um aspecto de organização do texto que facilita o entendimento. Por outro lado, a *legibility* é vista como um aspecto do texto ou do ambiente que ajuda a criar mapas mentais, esses aspectos facilitam a identificação pelo leitor. Portanto, é possível concluir que no design os conceitos de *readability* e *legibility* também são complementares.

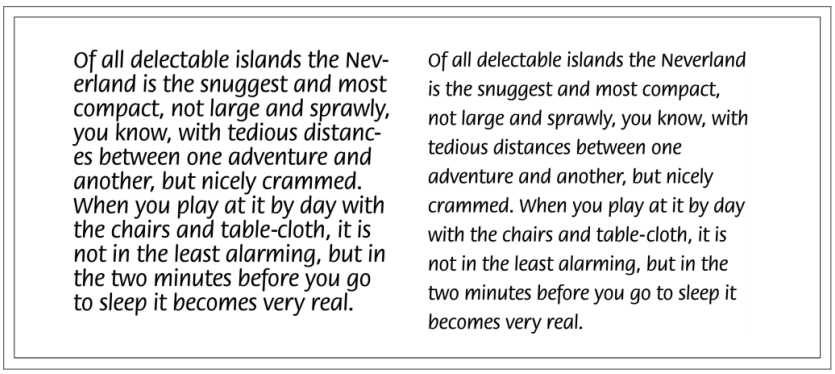
4.1.4 Definição na área de Interação Humano-Computador (IHC)

Na área de interação humano-computador, os termos *readability*, *legibility*, têm significados mais bem definidos. O trabalho de Krebs et al. (KREBS; XING; JR, 2002) afirma que a *readability* pode ser medida pelo “desempenho de leitura dos observadores para o texto exibido”. Outro exemplo é o trabalho de Zuffi (ZUFFI et al., 2009) que define *readability* como “a característica que permite ler facilmente frases do material de estímulo, independentemente de seu significado".

Figura 8 – Texto com Alta Legibility

Fonte: (STRIZVER, 2013)

Figura 9 – Texto com Alta Readability



Fonte: (STRIZVER, 2013)

Enquanto *legibility* para Zuffi (ZUFFI et al., 2009) *"se refere às propriedades visuais de um personagem ou símbolo, determinando a facilidade com que pode ser reconhecido."* Smith (SMITH, 1979) também aborda a temática de *legibility* em seu trabalho que tem por objetivo discutir sobre qual tamanho de letra é ideal para manter a legibilidade. Nesse trabalho o autor afirma que *legibility* está conectada com aspectos visuais como: tamanho da letra, a matriz total ou alfabeto de letras usado, as características detalhadas da forma, o uso de maiúsculas ou minúsculas, a proporção de altura para largura e a largura do traço. Adicionalmente, o trabalho de Greco et al. (GRECO et al., 2008) definiu *legibility* como "a facilidade de leitura de uma fonte (ou seja, de detecção de caracteres)". Esses trabalhos

reforçam a ideia encontrada em outras áreas de que a *legibility* está conectada a aspectos visuais que facilitam a compreensão do texto.

Percebe-se que na área de interação humano-computador, os trabalhos examinados referem-se a *readability* como a facilidade de compreender a linguagem escrita. Por outro lado, *legibility* é entendida como o conjunto de aspectos visuais que impactam a *readability* e influenciam a compreensão. Aumentar o tamanho da letra, por exemplo, pode melhorar a *legibility*. Essa característica é bem acolhida atualmente pelos estudos em IHC e foi confirmada pelo experimento de Smith (SMITH, 1979).

Enquanto Smith (SMITH, 1979) estava preocupado em estudar como o tamanho das letras em um monitor impactava a *legibility*, Zuffi et al. (ZUFFI et al., 2007) investigou o limite na diferença de luminosidade que impacta na *legibility*. Os autores identificam que a luminosidade entre a cor do primeiro plano e a cor do fundo é também um aspecto importante. Texto claro em fundo escuro é mais difícil de ler.

A *readability* e *legibility* são conceitos interligados. Ambos caminham juntos, porém possuem suas próprias características. A *legibility*, por exemplo, é impactada pelo contraste e brilho porque esses elementos influenciam o modo como um símbolo é exibido. Por outro lado, a *readability* é impactada tanto pelos aspectos da baixa *legibility* quanto pelo desempenho de leitura dos indivíduos.

4.1.5 Definição em Engenharia de Software

Na Engenharia de Software, os termos *readability*, *legibility* e *understandability* têm significados sobrepostos. Por exemplo, Coleman e Liau (COLEMAN; LIAU, 1975) em seu trabalho afirma que o termo *readability* está associado à "facilidade de leitura" de uma sentença. Neste trabalho, os autores procuraram criar uma fórmula para fornecer um método econômico para medir a *readability* mecanicamente com um scanner óptico. Na mesma vertente McCallum e Peterson (MCCALLUM; PETERSON, 1982) afirmam que *readability* pode ser *"uma medida de facilidade (ou dificuldade) de ler e entender um trecho de texto"*. Neste trabalho os autores também buscaram desenvolver uma ferramenta para calcular a *readability*. Esses trabalhos mostram que a *readability* é percebida como um aspecto ligado a facilidade de leitura e compreensão.

Outro trabalho que aborda a temática é o de Aggarwal et al. (AGGARWAL; SINGH; CHHABRA, 2002). Os autores afirmam que a *readability* pode ser estimada através da porcentagem de linhas de comentários no código total porque esses são fatores que facilitam a compreensão do código. Os trabalhos de Buse e Weimer (BUSE; WEIMER, 2008) (BUSE; WEIMER, 2010) definem *readibility* como *"um julgamento humano de quão fácil um texto é de entender"*. Nesses dois trabalhos, os autores afirmam que a *readability* é vista como um fator crítico para a qualidade geral do software. O trabalho de Posnett (POSNETT; HINDLE; DEVANBU, 2011) define *readability* como "uma impressão subjetiva que os programadores têm da dificuldade do código à medida que tentam entendê-lo".

Esses conceitos mostram que a *readability* está muito próxima do que é a definição de compreensão.

Adicionalmente, o conceito de *legibility* é definido de forma muito semelhante ao *readability*. Por exemplo, o trabalho de Moonen (MOONEN, 2003) define a *legibility* do software usando os mesmos termos que Lynch (LYNCH, 1984) usou para definir a *legibility* de um espaço físico: *"a facilidade com que suas partes podem ser reconhecidas e organizadas em um padrão coerente"*. Esse trabalho identifica que a melhoria da legibilidade implica em sistemas mais memoráveis e capazes de gerar modelos mentais mais fortes. Esse conjunto de fatores pode tornar o código mais fácil de explorar e manter. A *legibility* é, portanto, um aspecto importante para a exploração de sistemas segundo o autor. Somado a essa ideia, Almeida et al. (ALMEIDA et al., 2003) afirmam que *"a legibility é fundamental para a manutenção do código; se o código-fonte é escrito de forma complexa, entendê-lo exigirá muito mais esforço"*. Ou seja, a *legibility* também é vista como uma característica que facilita a compreensão do código.

Lin e Wu (LIN; WU, 2008) afirmam que *"a compreensibilidade do software determina se um sistema pode ser facilmente compreendido por outros indivíduos ou se os artefatos de um sistema podem ser facilmente compreendidos por outros indivíduos"*. Percebe-se que esse trabalho define o conceito de compreensibilidade do software utilizando o mesmo conceito que outros autores utilizam para definir *readability* e *legibility*. Por outro lado, Xia et al. (Xia et al., 2018) tratam a *comprehension* e *understanding* como sinônimos, expressando que *"a compreensão do programa (também conhecida como, compreensão do código-fonte) é um processo onde os desenvolvedores adquirem ativamente conhecimento sobre um sistema de software explorando e pesquisando artefatos de software e lendo fontes relevantes código e/ou documentação"*. Percebe-se com essas definições que existe uma sobreposição dos conceitos de *readability* e *comprehension* na área de Engenharia de Software.

No contexto da Engenharia de Software, as definições para *readability* muitas vezes estão muito próximas das definições de compreensão de código. O termo *legibility* é geralmente tratado como sinônimo para *readability*. Infelizmente não conseguimos encontrar nenhum trabalho que diferencia os termos *readability* e *legibility* na área de Engenharia de Software. A falta de definições claras pode levar a categorização de problemas e soluções de forma equivocada. Sobretudo, pode negligenciar a relevância de estudos separados. Por exemplo, o trabalho de Fakhoury et al. (FAKHOURY et al., 2019b) estudou anti-padrões linguísticos, em resumo, esse trabalho estudou aspectos ligados a *readability* como nome de identificadores e aspectos ligados a *legibility* como estrutura visual do código, porém não apresenta distinções sobre esses conceitos. Da mesma forma, o trabalho de (SANTOS; GEROSA, 2018) estudou atributos como recuo de código e o uso de importações e também não os distingue. Ambos afirmam que os aspectos estudados estão ligados ao tema de *readability*, porém seguindo a definição encontrada em outras áreas podemos perceber

que os fatores estudados são de naturezas diferentes. Embora os resultados das pesquisas realizadas sobre o tema de legibilidade de código não sejam refutados, estudos mais aprofundados podem também mostrar quais formas são melhores para medir esses fatores em código fonte.

4.1.6 Conclusões

Com base nas diferenças entre os termos *readability* e *legibility* que estão bem estabelecidos em outras áreas, como linguística (DUBAY, 2004), design (STRIZVER, 2013), interação humano-computador (ZUFFI et al., 2007) e educação (TEKFI, 1987), acreditamos que os dois termos devem ter significados claros, distintos, embora relacionados, também na área de Engenharia de Software. No geral, o conceito de *legibility* nas áreas citadas acima está relacionado a aspectos visuais dos elementos de texto e *readability* está relacionado a aspectos estruturais e semânticos do texto.

Usar termos diferentes para representar esses dois conceitos pode ser útil na identificação de problemas específicos e melhorias das práticas e recursos de programação, por exemplo, para pessoas com deficiência visual, técnicas para melhorar *legibility*, como recuo e uso de espaço vertical têm um grande impacto porque podem ajudar na compreensão de um texto ou de um programa. Além disso, para melhorar *readability*, ferramentas de ambiente de desenvolvimento integrado podem fornecer recursos com informações adicionais que ajudem a conectar semanticamente as partes de um programa. O nome de identificadores significativos também podem melhorar o grau de *readability*, uma vez que nomes significativos podem ajudar na aquisição do conhecimento da função do programa (TEASLEY, 1994). Diferenciar os termos *readability* e *legibility* ajudam pesquisadores a entender melhor os impactos e as limitações de seus estudos, técnicas e ferramentas. Se existem conjuntos de problemas diferentes, com características e soluções muitas vezes distintas e que possivelmente afetam stakeholders distintos, faz sentido que também existam termos distintos. Adicionalmente, os nomes que são dados às coisas impactam a forma como são percebidas, como mostra Adam em seu trabalho (ALTER, 2013).

Portanto, é possível afirmar que as características estruturais e semânticas do código-fonte de um programa afetam a capacidade dos desenvolvedores de entendê-lo durante a leitura do código. Por exemplo, construções de linguagens de programação, padrões de codificação e identificadores significativos, afetam sua *readability*. Por outro lado, as características visuais do código-fonte de um programa, que afetam a capacidade dos desenvolvedores de identificar os elementos do código durante a leitura, como quebras de linha, espaçamento, alinhamento, recuo, linhas em branco, capitalização do identificador, impacta sua *legibility*. Além disso, esses dois termos são empregados de acordo com essas definições informais. A Tabela 3 mostra alguns exemplos de trabalhos que se enquadram nas definições propostas.

Tabela 3 – Exemplos de estudos sobre readability e legibility

Referência	**Categoria de Estudo**	**Atributos estudados**
(AJAMI; WOODBRIDGE; FEITELSON, 2019)	Readability	If, for, negação; looping contagem regressiva.
(ARAB, 1992)	Legibility	Como e quanto recuar, onde colocar os pares "begin-end", quando usá-los, onde quebrar linhas, onde inserir linhas em branco, até onde inserir alguns caracteres em branco.
(AVIDAN; FEITELSON, 2017)	Readability	Nomes das variáveis.
(BAUER et al., 2019)	Legibility	Recuo em quatro níveis (0, 2, 4, 8).
(GEFFEN; MAOZ, 2016)	Legibility	Quatro estilos diferentes de ordenação de métodos.

Fonte: Elaborado pelo autor (2020)

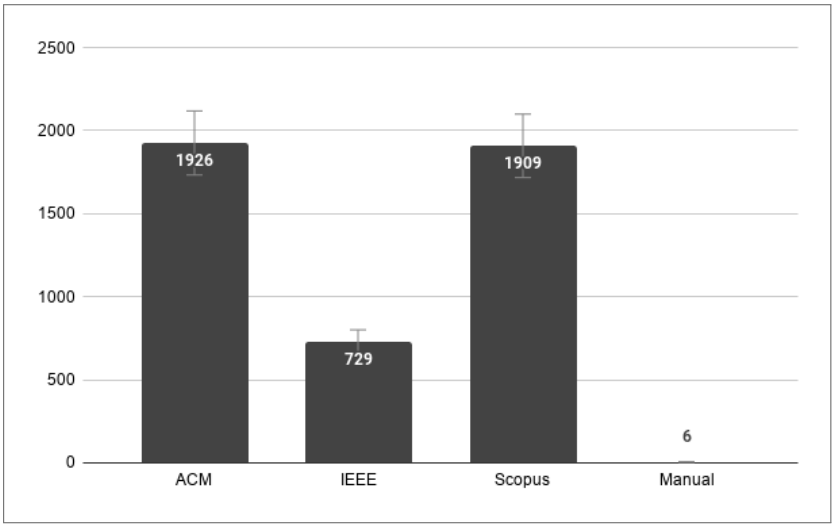
4.2 RQ2. COMO PODEMOS AVALIAR CODE READABILITY E CODE LEGIBILITY?

Os resultados para essa pergunta de pesquisa foram publicados no artigo Evaluating Code Readability and Legibility An Examination of Human centric Studies (OLIVEIRA et al., 2020), na International Conference on Software Maintenance and Evolution (ICSME). Esse trabalho teve como objetivo sintetizar os resultados da revisão sistemática buscando entender como os estudos centrados no ser humano avaliam se certa maneira de escrever código é mais legível ou ilegível do que outra funcionalmente equivalente. Nele é apresentado, quais os tipos de tarefas são realizados por seres humanos nos estudos de readability e legibility, quais habilidades cognitivas são exigidas dos sujeitos e quais variáveis de resposta esses estudos empregam. Contudo, para responder a segunda pergunta de pesquisa dessa dissertação foram utilizados apenas os dados sobre as tarefas realizadas pelos sujeitos e as variáveis de respostas utilizadas nos estudos. Os resultados sobre as habilidades cognitivas associadas a cada tipo de tarefa realizadas nos estudos não estão relacionados diretamente com os objetivos dessa dissertação, portanto não serão apresentados nessa seção. A seguir é apresentado uma análise descritiva dos dados, posteriormente, são apresentadas as tarefas realizadas pelos sujeitos nos 54 estudos e, por último, é discutido quais as variáveis de resposta utilizadas pelos estudos.

4.2.1 Análise Descritiva

A revisão sistemática da literatura desenvolvida neste trabalho teve sua fase de condução da pesquisa efetuada como acordado no protocolo de pesquisa. Esse protocolo foi resumidamente exposto na seção de metodologia (Capítulo 3) e pode ser encontrado em sua totalidade no Apêndice A. Para a etapa de pesquisa dos estudos primários, primeira fase da etapa de condução da revisão sistemática, foram realizadas uma busca manual em eventos relevantes para a pesquisa e uma busca automática em três bases de repositórios de artigos, sendo elas: ACM, IEEE e Scopus. A Figura 10 mostra a quantidade de trabalhos retornados por cada mecanismo de pesquisa. Nesta figura podemos observar que o mecanismo de busca da ACM retornou 1926 trabalhos, IEEE teve um número de 729 trabalhos retornados e o Scopus retornou 1909 trabalhos. Os resultados da busca manual foi de 6 relevantes trabalhos encontrados. Os trabalhos da busca manual não foram utilizados como artigos sementes porque foram encontrados durante a leitura dos artigos incluídos na fase de seleção final. Mais detalhes sobre esses artigos são apresentados na subseção 3.4.4. A lista completa com todos os artigos incluídos na revisão pode ser encontrada no Apêndice B.

Figura 10 – Resultados das estratégias de busca



Fonte: Elaborado pelo autor (2020)

Após o processo de triagem dos trabalhos relevantes para a pesquisa, o total de trabalhos incluídos nesta revisão sistemática foi de 54 trabalhos. Entre esse conjunto de trabalhos, 21 deles tem sua origem do mecanismo de busca da ACM, 8 foram oriundos do mecanismo de busca da IEEE, 20 foram oriundos do mecanismo de busca da Scopus e 5 foram oriundos da busca manual. Esses dados podem ser melhor visualizados na Tabela 4.

Embora o número de artigos retornados pelo mecanismo da ACM tenha sido um dos mais altos, entre os 54 artigos aceitos apenas 38,88% deles foram oriundos do mecanismo ACM. Apenas 37,03% dos artigos foram do mecanismo Scopus. Se somados esses dois

mecanismos de busca retornaram mais da metade dos estudos incluídos nesta pesquisa, cerca de 75,92% dos artigos.

Tabela 4 – Resultados das estratégias de busca

Método de Busca	**Fontes**	**Artigos Retornados**	**Artigos Rejeitados**	**Artigos Incluídos**
Automático	ACM	1,926	1,905	21
	IEEE	729	715	8
	Scopus	1909	1883	20
Manual	Conhecido pelos Pesquisadores	6	1	5

Fonte: Elaborado pelo autor (2020)

Todos esses trabalhos foram analisados no processo de triagem, essa análise teve início com a leitura do título e resumo para aplicação dos critérios de exclusão. Posteriormente, ainda no processo de triagem, os mesmos atributos dos trabalhos foram lidos e os critérios de inclusão foram aplicados. Os critérios de exclusão e inclusão estão descritos no Capítulo 3. Todo esse processo foi o que resultou no número final de trabalhos incluídos. Os critérios de exclusão mais frequentes formam CE1 que está relacionado a artigos que estão fora do escopo primário desta pesquisa e CE5 que corresponde a estudos que são sobre ajudas de compreensão de programa, como visualizações ou outras formas de análise ou ajudas sensoriais (por exemplo, gráficos, execução baseada em traços, resumo de código, mineração de especificações, engenharia reversa).

Dentro do processo de triagem dos trabalhos, optamos por aplicar uma fase de análise da qualidade, essa fase teve como objetivo analisar a metodologia e os resultados dos trabalhos incluídos no estudo. Os resultados obtidos dessa análise mostram que 79,62% dos trabalhos receberam uma pontuação de qualidade dentro das classificações boa ou excelente, onde 44,44% deles tiveram uma pontuação de qualidade considerada excelente. Nenhum artigo foi excluído nessa fase porque o valor mínimo estipulado para exclusão nessa fase foi de 4,5. A Tabela 5 apresenta as pontuações possíveis e a porcentagem dos estudos que receberam as pontuações apresentadas. A lista com a pontuação de cada artigo pode ser encontrada no Apêndice C.

A Figura 11 mostra a distribuição dos artigos, incluídos na revisão sistemática, por ano de publicação. Nesta imagem pode-se perceber o aumento considerável das publicações sobre o tema de legibilidade de código nos últimos anos.

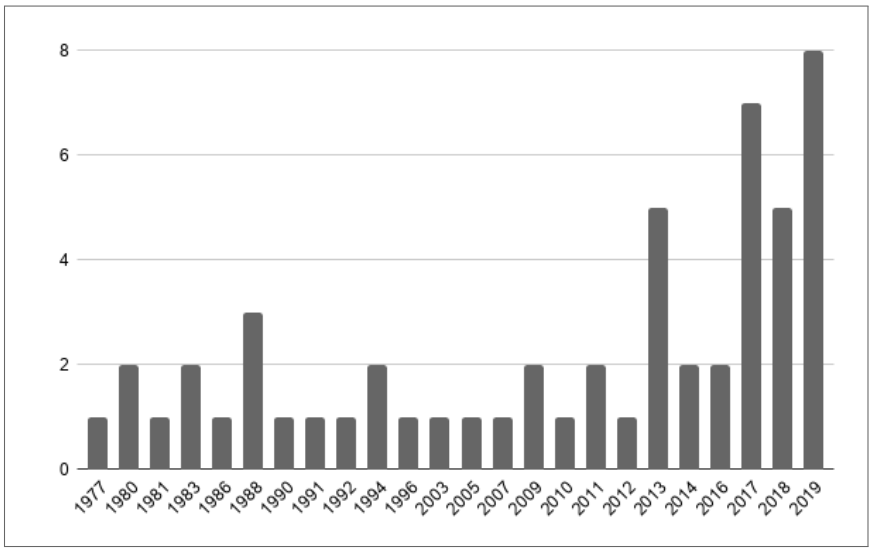
Destes artigos, um total de 14 trabalhos foram oriundos do International Conference on Program Comprehension (ICPC). Outras conferências/revistas que retornaram números consideráveis de trabalhos incluídos foi a Empirical Software Engineering (EMSE) que teve 6 trabalhos incluídos, a International Conference on Software Engineering (ICSE) e a International Journal of Man-Machine Studies (IJMMS) ambos com 3 trabalhos respec-

Tabela 5 – Resultados da Fase de Avaliação da Qualidade.

Classificação	Pontuações possíveis	Porcentagem de trabalhos
Excelente	9	31,48%
	8,5	12,96%
Boa	8,0	25,92%
	7,5	9,25%
Média	7,0	11,11%
	6,5	1,85%
Baixa	6	5,55%
	5	1,85%

Fonte: Elaborado pelo autor (2020)

Figura 11 – Distribuição dos Artigos por Ano de Publicação.



Fonte: Elaborado pelo autor (2020)

tivamente incluídos. A Tabela 6 apresenta uma visualização dos eventos e conferências que tiveram trabalhos incluídos nesta revisão, em ambas visualizações é possível ver a quantidade de trabalhos por cada evento ou revista.

Dentro do conjunto de 54 artigos incluídos, 72,2% utilizam apenas experimento controlado como método de pesquisa, 13,0% usam estudo de caso e 7,4% dos trabalhos utilizaram o método de pesquisa de opinião com experimento. Nesse trabalho são considerados experimentos controlados os estudos que além de declararem o método fazem uma investigação de hipótese onde uma ou mais variáveis independentes são manipuladas para medir seu efeito em uma ou mais variáveis dependentes. Por outro lado, quasi-experimento são considerados estudos em que os sujeitos não são atribuídos aleatoriamente aos tratamentos (EASTERBROOK et al., 2008). A Figura 12 apresenta todos os tipos de estudos encontrados no conjunto de trabalhos. Desse conjunto de artigos apenas 9,25% utilizam mais de um

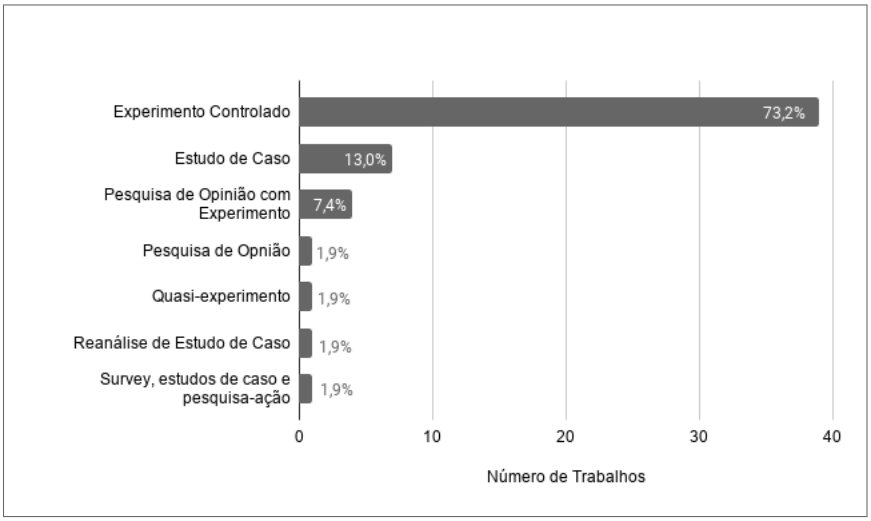
Tabela 6 – Distribuição dos Artigos por Veículo de Publicação.

Veículos de Publicação	**Trabalhos Incluídos**
International Conference on Program Comprehension (ICPC)	14
Empirical Software Engineering (EMSE)	6
Transactions on Software Engineering (TSE)	4
International Conference on Software Engineering (ICSE)	3
International Journal of Man-Machine Studies (IJMMS)	3
Communications of the ACM (CACM)	2
Journal of Software: Evolution and Process (JSEP)	2
Special Interest Group on Programming Languages (SIGPLAN)	2
ACM 1980 annual conference (ACM1980)	1
Australasian User Interface Conference (AUIC)	1
Australasian Computing Education Conference (ACE)	1
Frontiers in Education Conference (FEC)	1
Innovations in Systems and Software Engineering (ISSE)	1
International Conference on Generative Programming: Concepts & Experiences (ICGP)	1
International Conference on Software Maintenance (ICSM)	1
International Journal of Human-Computer Studies (IJHCS)	1
Journal of Systems and Software (JSS)	1
Science of Computer Programming (SCP)	1
Software: Practice and Experience (SPE)	1
Special Interest Group on Computer Science Education (SIGCSE)	1
Special Interest Group on Operating Systems (SIGOPS)	1
Symposium on the Foundations of Software Engineering (SIGSOFT)	1
The Journal of Systems and Software (TJSS)	1
Transactions on Computing Education (TCE)	1
Transactions on Knowledge and Data Engineering (TKDE)	1
Working Conference on Mining Software Repositories (MSR)	1
Total	**54**

Fonte: Elaborado pelo autor (2020)

método de pesquisa para validarem suas hipóteses. Esses trabalhos associaram mais de um dos seguintes métodos: experimento controlado aliado com survey de opinião (BENIAMINI et al., 2017) (FAKHOURY et al., 2019b) (MALAQUIAS et al., 2017) e survey aliado a estudo de caso e pesquisa-ação (MEDEIROS et al., 2019).

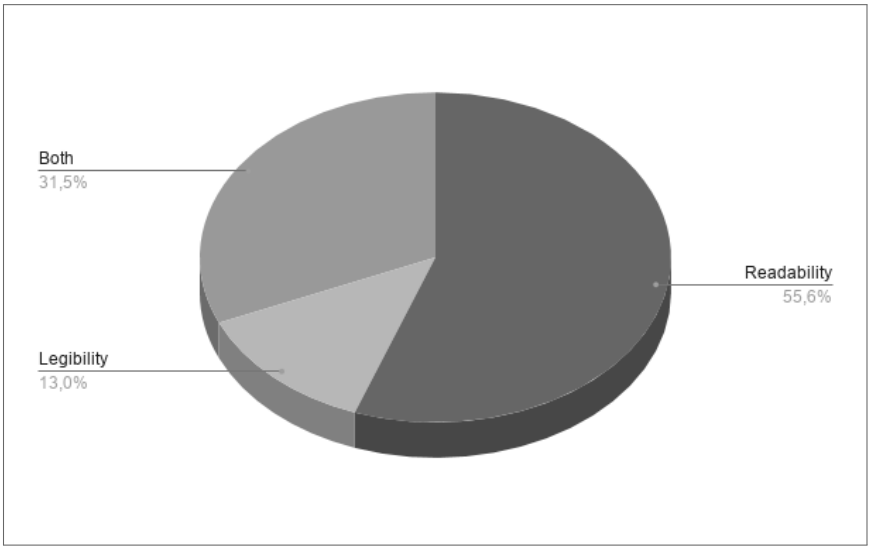
Figura 12 – Porcentagem dos Tipos de Estudos Incluídos.



Fonte: Elaborado pelo autor (2020)

Os contextos de todos esses trabalhos incluídos correspondem diretamente a temática de compreensão de código. Essa pesquisa, portanto, procurou estudar os assuntos de *legibility* e *readability*. Nesse sentido, os trabalhos incluídos procuraram estudar um desses dois temas ou os dois em conjunto. A Figura 13 apresenta as porcentagens de cada temática, mostrando que 55,6% desses estudos são sobre a temática de *readability*, apenas 13% é sobre a temática de *legibility* e 31,5% aborda ambas temáticas.

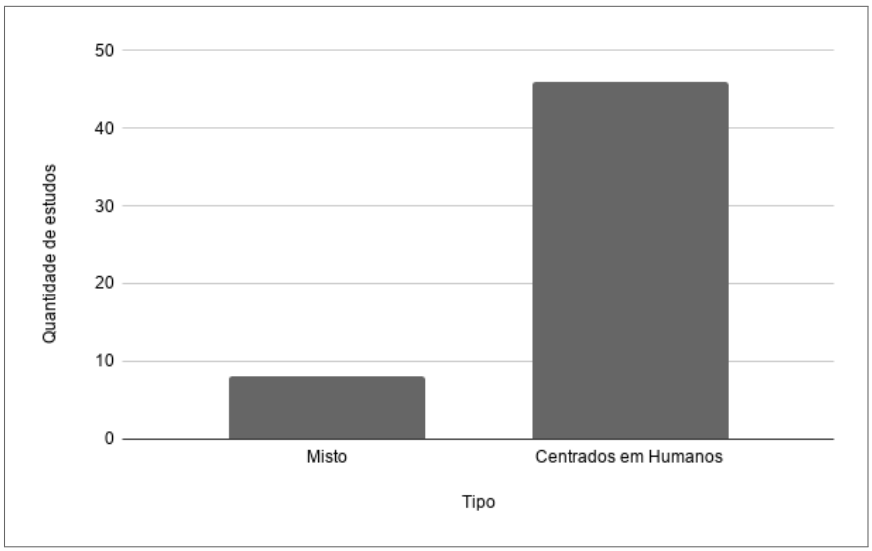
Figura 13 – Porcentagem das Categorias dos Estudos.



Fonte: Elaborado pelo autor (2020)

Ainda, é possível distinguir os trabalhos baseado em como o método científico é conduzido. Por exemplo, 85,18% dos trabalhos solicitam que participantes realizem alguma(s) tarefa(s) e os dados do estudo são extraídos a partir da execução delas. Esse tipo de trabalho foi nomeado como centradas em humanos. Outros trabalhos, no total de 14,81%, associam atividades realizadas por participantes com dados extraídos de código-fonte através de ferramentas. Esses foram chamados de Misto. É importante ressaltar que foram retornados trabalhos que apenas extraíram os dados a partir da execução de ferramentas em código-fonte, foram excluídos por não estarem ligados com os objetivos da revisão.

Figura 14 – Quantidade dos Estudos por Abordagem.



Fonte: Elaborado pelo autor (2020)

Assim, os principais resultados dessa análise descritiva mostram que dos 54 trabalhos incluídos, a maioria dos estudos foi retornado pela ACM e Scopus. Com relação aos trabalhos retornados nesta pesquisa, muitos dos excluídos estavam fora do escopo da pesquisa ou tratavam de assuntos como visualização e análise como ajuda para compreensão de programas. Quanto aos artigos incluídos na revisão, 79,62% receberam uma pontuação de qualidade dentro das classificações boa ou excelente. Entre os últimos anos houve um aumento considerável de estudos nesta área de pesquisa. Entre os locais de publicação os que mais retornaram estudos relevantes foram os International Conference on Program Comprehension, Empirical Software Engineering e International Conference on Software Engineering. Os métodos de pesquisa mais utilizados entre os estudos incluídos nesta revisão foram experimento e experimento controlado. Nesses trabalhos um total de 55,6% são sobre *readability*. Enquanto 85,18%, do conjunto geral, são trabalhos centrados em humanos.

4.2.2 Síntese dos Dados

Tarefas realizadas por seres humanos

A tarefa essencial de compreensão de código é a leitura de código. Por construção, todos os estudos selecionados têm pelo menos uma tarefa de leitura em que o sujeito é obrigado a ler um trecho de código, um conjunto de trechos ou mesmo programas grandes e completos. Uma vez que todos os estudos comparam duas ou mais maneiras de escrever código, os sujeitos geralmente têm várias tarefas de leitura de código. Além disso, os participantes também devem compreender o código. No entanto, existem na ciência diferentes maneiras de medir o desempenho de compreensão de um participante. Os sujeitos podem então ser solicitados a fornecer informações sobre o código, agir de acordo com o código ou fornecer opinião pessoal sobre um trecho de código ou programa. Na maioria dos estudos incluídos na revisão sistemática, mais de um tipo de tarefa foi empregado. A Tabela 7 resume as tarefas identificadas.

Tabela 7 – Tarefas realizadas.

Tipo de tarefa	**Estudos**
Fornecer informações sobre o código (40 estudos)	
Explique o que o código faz	18 estudos: (AVIDAN; FEITELSON, 2017; BENANDER; BENANDER; PU, 1996; BENIAMINI et al., 2017; BINKLEY et al., 2013; BOYSEN; KELLER, 1980; CHAUDHARY; SAHASRABUDDHE, 1980; FAKHOURY et al., 2019b; JBARA; FEITELSON, 2014; JBARA; FEITELSON, 2017; KASTO; WHALLEY, 2013; LAWRIE et al., 2007; LOVE, 1977; MIARA et al., 1983; WIEDENBECK, 1986; WIEDENBECK, 1991; WOODFIELD; DUNSMORE; SHEN, 1981; BLINMAN; COCKBURN, 2005; O'NEAL; EDWARDS, 1994)
Responda perguntas sobre as características do código	27 estudos: (AJAMI; WOODBRIDGE; FEITELSON, 2019), (BAUER et al., 2019), (BOYSEN; KELLER, 1980), (DOLADO et al., 2003), (GEFFEN; MAOZ, 2016), (GOPSTEIN et al., 2017), (ISELIN, 1988), (KASTO; WHALLEY, 2013), (MAćKOWIAK; NAWROCKI; OCHODEK, 2018), (O'NEAL; EDWARDS, 1994), (SCHULZE et al., 2013), (SIEGMUND et al., 2017), (SYKES; TILLMAN; SHNEIDERMAN, 1983), (TEASLEY, 1994), (TROCKMAN et al., 2018), (WIESE; RAFFERTY; FOX, 2019), (WIESE et al., 2019), (WOODFIELD; DUNSMORE; SHEN, 1981), (YEH et al., 2017), (BINKLEY et al., 2009), (BINKLEY et al., 2013), (SCALABRINO et al., 2019), (MIARA et al., 1983), (Tenny, 1988), (Ceccato et al., 2009), (OMAN; COOK, 1988), (OMAN; COOK, 1990)

Continuação da Tabela 7	
Tipo de tarefa	**Estudos**
Lembre-se (parte) do código	7 estudos: (BINKLEY et al., 2009; BINKLEY et al., 2013; LAWRIE et al., 2007; LOVE, 1977; WIEDENBECK, 1986; WIEDENBECK, 1991; CHAUDHARY; SAHASRABUDDHE, 1980)
Agir no código (15 estudos)	
Encontrar e corrigir *bugs* no código	10 estudos: (BENIAMINI et al., 2017; JBARA; FEITELSON, 2014; Kleinschmager et al., 2012; MALAQUIAS et al., 2017; SCANNIELLO; RISI, 2013; SCHANKIN et al., 2018; FAKHOURY et al., 2019b; HOFMEISTER; SIEGMUND; HOLT, 2019; SCHULZE et al., 2013; SIEGMUND et al., 2017)
Modifique o código	8 estudos: (BENIAMINI et al., 2017; Kleinschmager et al., 2012; Ceccato et al., 2009; GEFFEN; MAOZ, 2016; MALAQUIAS et al., 2017; JBARA; FEITELSON, 2014; SCHULZE et al., 2013; WIESE et al., 2019)
Escrever código	3 estudos: (STEFIK; SIEBERT, 2013; WIESE; RAFFERTY; FOX, 2019; WIESE et al., 2019)
Fornecer opinião pessoal (30 estudos)	
Opinião sobre o código (em termos de legibilidade de forma e/ou conteúdo)	23 estudos: (ARNAOUDOVA; PENTA; ANTONIOL, 2016; BUSE; WEIMER, 2010; JBARA; FEITELSON, 2014; JBARA; FEITELSON, 2017; OMAN; COOK, 1988; OMAN; COOK, 1990; O'NEAL; EDWARDS, 1994; POSNETT; HINDLE; DEVANBU, 2011; SCALABRINO et al., 2018; STEFIK; GELLENBECK, 2011; STEFIK; SIEBERT, 2013; ARAB, 1992; BAUER et al., 2019; AVIDAN; FEITELSON, 2017; BENIAMINI et al., 2017; SANTOS; GEROSA, 2018; MEDEIROS et al., 2019; WIESE; RAFFERTY; FOX, 2019; WIESE et al., 2019; Ceccato et al., 2009; MALAQUIAS et al., 2017; SYKES; TILLMAN; SHNEIDERMAN, 1983; WANG; POLLOCK; VIJAY-SHANKER, 2014)
Responda se entendeu o código	4 estudos: (O'NEAL; EDWARDS, 1994; SCALABRINO et al., 2019; SYKES; TILLMAN; SHNEIDERMAN, 1983; TROCKMAN et al., 2018)
Avalie a confiança em sua resposta	3 estudos: (LAWRIE et al., 2007; WIEDENBECK, 1991; YEH et al., 2017)

Continuação da Tabela 7	
Tipo de tarefa	**Estudos**
Avalie a dificuldade da tarefa	7 estudos: (FAKHOURY et al., 2019b; MIARA et al., 1983; BAUER et al., 2019; Ceccato et al., 2009; JBARA; FEITELSON, 2014; JBARA; FEITELSON, 2017; YEH et al., 2017)

Grande parte dos estudos primários, 40 de 54 estudos, exigia que os sujeitos **fornecessem informações sobre o código**. Por exemplo, Benander et al. (BENANDER; BENANDER; PU, 1996) pediram aos sujeitos que explicassem usando texto de forma livre o que um trecho de código faz, logo após ter lido. Blinman et al. (BLINMAN; COCKBURN, 2005) pediu aos sujeitos para escolherem a melhor descrição para um trecho de código entre várias opções. Contudo, explicar o que o código faz não é um método objetivo para avaliar a compreensão porque alguém tem que julgar a resposta. Encontramos 18 estudos que empregaram esse tipo de tarefa.

Os sujeitos de 27 estudos foram convidados a responder a perguntas sobre as características do código. Em alguns estudos, os participantes foram solicitados a prever o comportamento de um código-fonte apenas olhando para ele. Por exemplo, Gopstein et al. (GOPSTEIN et al., 2017) e Ajami e Feitelson (AJAMI; WOODBRIDGE; FEITELSON, 2019) apresentaram aos participantes múltiplos trechos de código e pediram que eles adivinhassem as saídas desses trechos. Dolado et al. (DOLADO et al., 2003) pediram aos sujeitos que respondessem a um conjunto de questões sobre os resultados da expressão, valores finais das variáveis e quantas vezes os loops foram executados.

Em outros estudos, os sujeitos foram questionados sobre questões de nível superior relacionadas ao código-fonte. Por exemplo, Scalabrino et al. (SCALABRINO et al., 2019) perguntaram aos participantes se eles reconhecem um elemento de um domínio específico ou sobre o propósito de usar um componente externo no snippet (por exemplo, APIs JDBC). Da mesma forma, Binkley et al. (BINKLEY et al., 2009) indagaram os sujeitos sobre o tipo de aplicações ou a indústria onde uma linha específica de código pode ser encontrada. Além disso, alguns estudos exigiam que os participantes localizassem elementos de código de interesse. Por exemplo, Binkley et al. (BINKLEY et al., 2013) pediram aos sujeitos que encontrassem identificadores em um snippet, marcando cada linha de código que tinha esse identificador. Ceccato et al. (Ceccato et al., 2009) pediram aos sujeitos para identificar a parte do código que implementa uma funcionalidade específica.

Alguns estudos assumiram que um código fácil de entender também é fácil de memorizar. Portanto, eles tentaram medir o quanto os sujeitos se lembram do código. Por exemplo, Love (LOVE, 1977) pediu aos participantes que memorizassem um programa por três minutos e o reescrevessem com a maior precisão possível nos próximos quatro minutos. Lawrie et al. (LAWRIE et al., 2007) primeiro apresentou um trecho de código para

os sujeitos. Em seguida, em uma segunda etapa, listaram seis identificadores possíveis e os sujeitos tiveram que selecionar aqueles que lembravam que constavam no código. No geral, sete estudos pediram aos sujeitos que se lembrassem do código.

Por outro lado, alguns estudos exigiam que os sujeitos **atuassem no código** (agir no código). Em dez estudos os sujeitos foram solicitados a encontrar e consertar bugs no código. Scanniello et al. (SCANNIELLO; RISI, 2013) pediram aos participantes que o fizessem em dois programas com estilos de identificador diferentes. Em outros oito estudos os sujeitos foram solicitados a modificar o código de um programa de trabalho, ou seja, sem a necessidade de corrigir bugs. Por exemplo, Jbara e Feitelson (JBARA; FEITELSON, 2014) pediram aos participantes que implementassem um novo recurso em um programa visto em uma tarefa anterior. Da mesma forma, Schulze et al. (SCHULZE et al., 2013) solicitaram aos sujeitos para modificar e excluir o código anotado com diretivas específicas do pré-processador, o que também requer a compreensão do respectivo código-fonte.

Em alguns estudos, os sujeitos foram solicitados a escrever códigos a partir de uma descrição. Escrever código em si não é uma tarefa de compreensão, mas pode estar associada a uma tarefa de compreensão. Por exemplo, Wiese et al. (WIESE; RAFFERTY; FOX, 2019) primeiro pediram aos sujeitos que escrevessem uma função que retornaria verdadeiro se a entrada fosse 7 e falso caso contrário, para que eles pudessem identificar qual estilo de código (novato, especialista ou misto) os sujeitos prefeririam ao codificar. Em seguida, eles pediram aos sujeitos que escolhessem a mais legível entre as três versões de uma função, cada uma com um estilo de código diferente. Um dos objetivos deste estudo foi determinar se os sujeitos escrevem código no mesmo estilo que consideram mais legível.

Por último, em 30 estudos os participantes foram convidados a **fornecer opinião pessoal**. Em nove estudos, os sujeitos foram questionados sobre suas preferências pessoais ou intuição, sem qualquer tarefa adicional. Por exemplo, Buse et al. (BUSE; WEIMER, 2010) pediram que avaliassem (de 1 a 5) o quão legível ou legível um trecho de código é. Da mesma forma, Arab (ARAB, 1992) pediu aos sujeitos que classificassem a legibilidade de três esquemas de apresentação do código Pascal em ordem decrescente. No estudo de Santos e Gerosa (SANTOS; GEROSA, 2018), os sujeitos escolheram o mais legível entre dois trechos funcionalmente equivalentes. Em outros estudos, 21 no total, os sujeitos foram questionados sobre sua opinião pessoal enquanto realizavam outras tarefas. Por exemplo, O'Neal et al. (O'NEAL; EDWARDS, 1994) primeiro pediram aos sujeitos para lerem um trecho de código, e então declarar se eles entenderam o trecho e fornecer uma descrição de sua funcionalidade. Da mesma forma, Lawrie et al. (LAWRIE et al., 2007) pediram aos sujeitos que fornecessem uma descrição escrita de forma livre do propósito de uma função e avaliassem sua confiança em sua descrição. Além disso, os sujeitos foram solicitados a avaliar a dificuldade das tarefas de compreensão que eles tiveram que realizar em alguns estudos, por exemplo, no estudo de Fakhoury et al.(FAKHOURY et al., 2019b).

Variáveis de Resposta Utilizadas

Dependendo dos objetivos, metodologia e assuntos de um estudo, as variáveis de resposta variam. Esta seção apresenta os resultados obtidos na análise das variáveis de resposta utilizadas nos estudos selecionados. Como há uma diversidade considerável de variáveis de resposta, nós as organizamos em cinco categorias. A Figura 15 apresenta a frequência de todas as variáveis de resposta. Os números não somam 54, que é o número geral de estudos analisados, pois a maioria dos estudos empregou mais de uma variável de resposta. Na Tabela 8, apresentamos uma síntese detalhada das variáveis de resposta identificadas.

O desempenho dos sujeitos foi medido em alguns estudos em termos de saber se eles eram capazes de fornecer informações corretamente sobre os programas apenas olhando para o código-fonte. As variáveis de resposta podem pertencer à estrutura do código, semântica, uso de algoritmos ou comportamento do programa, por exemplo. Agregamos variáveis de resposta como essas em uma categoria chamada **correção**. Por um lado, a correção pode ser determinada objetivamente. Por exemplo, Bauer et al. (BAUER et al., 2019) mediram a compreensão do código dos sujeitos, pedindo-lhes para preencher um questionário com questões de múltipla escolha referentes à saída do programa. Gopstein et al. (GOPSTEIN et al., 2017) e Ajami e Feitelson (AJAMI; WOODBRIDGE; FEITELSON, 2019) pediram aos sujeitos para prever os resultados da execução de programas curtos. Por outro lado, a correção foi determinada subjetivamente em alguns estudos, onde havia alguma margem para interpretar se os resultados produzidos por um sujeito estavam corretos. Por exemplo, Blinman et al. (BLINMAN; COCKBURN, 2005) avaliaram se a descrição textual dos sujeitos de um programa estava correta. Da mesma forma, Love (LOVE, 1977) pediu aos participantes que escrevessem uma descrição textual de um programa e pontuou essa descrição com base em uma avaliação subjetiva. No geral, as variáveis de resposta de correção foram empregadas por 83,3% dos estudos.

Tabela 8 – Variáveis de resposta e seus estudos correspondentes

Categoria e tipo	**Subtipo**	**Estudos**
Correção (45 estudos)		

Continuação da Tabela 8		
Categoria e tipo	**Subtipo**	**Estudos**
Objetivo	Binário	37 estudos: (AJAMI; WOODBRIDGE; FEITELSON, 2019; BAUER et al., 2019; BINKLEY et al., 2009; BINKLEY et al., 2013; BLINMAN; COCKBURN, 2005; BOYSEN; KELLER, 1980; Ceccato et al., 2009; DOLADO et al., 2003; FAKHOURY et al., 2019b; GEFFEN; MAOZ, 2016; GOPSTEIN et al., 2017; HOFMEISTER; SIEGMUND; HOLT, 2019; Kleinschmager et al., 2012; ISELIN, 1988; JBARA; FEITELSON, 2014; LAWRIE et al., 2007; LOVE, 1977; MAćKOWIAK; NAWROCKI; OCHODEK, 2018; MALAQUIAS et al., 2017; MIARA et al., 1983; OMAN; COOK, 1988; OMAN; COOK, 1990; O'NEAL; EDWARDS, 1994; SCALABRINO et al., 2019; SCANNIELLO; RISI, 2013; SCHANKIN et al., 2018; SIEGMUND et al., 2017; SYKES; TILLMAN; SHNEIDERMAN, 1983; TEASLEY, 1994; Tenny, 1988; TROCKMAN et al., 2018; WIEDENBECK, 1986; WOODFIELD; DUNSMORE; SHEN, 1981; WIEDENBECK, 1991; WIESE; RAFFERTY; FOX, 2019; WIESE et al., 2019; YEH et al., 2017)
Subjetivo	Escala	3 estudos: (SCHULZE et al., 2013) (SCHULZE et al., 2013; STEFIK; SIEBERT, 2013; BENIAMINI et al., 2017)

Continuação da Tabela 8		
Categoria e tipo	**Subtipo**	**Estudos**
	Binário	11 estudos: (AVIDAN; FEITELSON, 2017; BENANDER; BENANDER; PU, 1996; BOYSEN; KELLER, 1980; CHAUDHARY; SAHASRABUDDHE, 1980; FAKHOURY et al., 2019b; KASTO; WHALLEY, 2013; MIARA et al., 1983; WIEDENBECK, 1986; WIEDENBECK, 1991; WIESE; RAFFERTY; FOX, 2019; WIESE et al., 2019)
	Taxa	7 estudos:(BENIAMINI et al., 2017; JBARA; FEITELSON, 2017; LAWRIE et al., 2007; BINKLEY et al., 2013; JBARA; FEITELSON, 2014; LOVE, 1977; WOODFIELD; DUNSMORE; SHEN, 1981)
Opinião (30 estudos)		
Preferência pessoal	Taxa	9 estudos: (ARNAOUDOVA; PENTA; ANTONIOL, 2016; BUSE; WEIMER, 2010; MEDEIROS et al., 2019; OMAN; COOK, 1988; OMAN; COOK, 1990; POSNETT; HINDLE; DEVANBU, 2011; SCALABRINO et al., 2018; STEFIK; GELLENBECK, 2011; STEFIK; SIEBERT, 2013)
	Escolha	6 estudos: (AVIDAN; FEITELSON, 2017; BENIAMINI et al., 2017; SANTOS; GEROSA, 2018; WANG; POLLOCK; VIJAY-SHANKER, 2014; WIESE; RAFFERTY; FOX, 2019; WIESE et al., 2019)
	Classificar	1 study: (ARAB, 1992)
Em compreensibilidade	Binário	2 estudos: (SCALABRINO et al., 2019; TROCKMAN et al., 2018)
	Taxa	2 estudos: (SYKES; TILLMAN; SHNEIDERMAN, 1983; O'NEAL; EDWARDS, 1994)

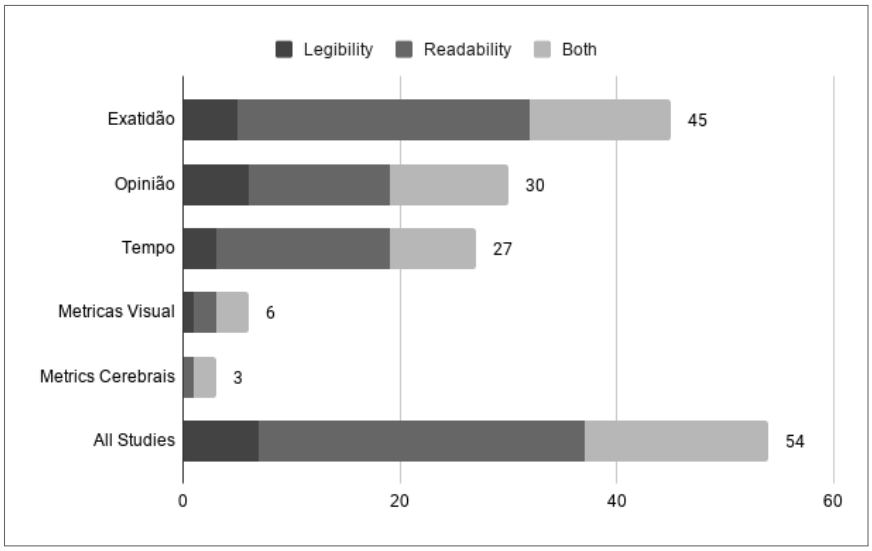
Continuação da Tabela 8		
Categoria e tipo	**Subtipo**	**Estudos**
Opinião profissional	Aceitabilidade	2 estudos: (MEDEIROS et al., 2019; MALAQUIAS et al., 2017)
Avalie a confiança da resposta	Taxa	3 estudos: (LAWRIE et al., 2007; WIEDENBECK, 1991; YEH et al., 2017)
Classifique a dificuldade da tarefa	Classificar	1 study: (BAUER et al., 2019)
	Taxa	6 estudos: (Ceccato et al., 2009; FAKHOURY et al., 2019b; JBARA; FEITELSON, 2017; JBARA; FEITELSON, 2014; MIARA et al., 1983; YEH et al., 2017)
Tempo (27 estudos)		
Tempo para completar a tarefa		20 estudos: (AJAMI; WOODBRIDGE; FEITELSON, 2019; BAUER et al., 2019; BENANDER; BENANDER; PU, 1996; BENIAMINI et al., 2017; BLINMAN; COCKBURN, 2005; Ceccato et al., 2009; DOLADO et al., 2003; HOFMEISTER; SIEGMUND; HOLT, 2019; ISELIN, 1988; JBARA; FEITELSON, 2014; JBARA; FEITELSON, 2017; Kleinschmager et al., 2012; MALAQUIAS et al., 2017; O'NEAL; EDWARDS, 1994; SCANNIELLO; RISI, 2013; SCHANKIN et al., 2018; OMAN; COOK, 1990; OMAN; COOK, 1988; SCHULZE et al., 2013; BINKLEY et al., 2013)
Timpo de leitura de código		5 estudos: (AJAMI; WOODBRIDGE; FEITELSON, 2019; BINKLEY et al., 2013; BOYSEN; KELLER, 1980; FAKHOURY et al., 2019b; SCALABRINO et al., 2019)

Continuação da Tabela 8		
Categoria e tipo	**Subtipo**	**Estudos**
Tempo por questão		4 estudos: (BINKLEY et al., 2009; GEFFEN; MAOZ, 2016; BOYSEN; KELLER, 1980; MAćKOWIAK; NAWROCKI; OCHODEK, 2018)
Número de tentativas		2 estudos: (MALAQUIAS et al., 2017; MAćKOWIAK; NAWROCKI; OCHODEK, 2018)
Métricas Visuais (6 estudos)		
Rastreamento ocular		4 estudos: (BAUER et al., 2019; BINKLEY et al., 2013; JBARA; FEITELSON, 2017; FAKHOURY et al., 2019b)
Letterboxing		2 estudos: (HOFMEISTER; SIEGMUND; HOLT, 2019; SCHANKIN et al., 2018)
Métricas do cérebro (3 estudos)		
fMRI		1 estudo: (SIEGMUND et al., 2017)
fNIRS		1 estudo: (FAKHOURY et al., 2019b)
EEG		1 estudo: (YEH et al., 2017)

A segunda variável de resposta mais utilizada, presente em 30 estudos, é a opinião pessoal dos sujeitos. O que é comum a todos esses estudos é o uso das preferências e intuição dos sujeitos, em vez dos resultados do que eles fazem, para avaliar a *readability* e a *legibility*. Agrupamos essas variáveis de resposta em uma categoria chamada **opinião**. Scalabrino et al. (SCALABRINO et al., 2019), por exemplo, pediram aos sujeitos para declarar se eles entenderam um trecho de código ou não. Santos e Gerosa (SANTOS; GEROSA, 2018) apresentaram pares de trechos funcionalmente equivalentes aos sujeitos e pediram-lhes que escolhessem qual achavam mais *readable* ou *legible*. Stefik e Gellenbeck (STEFIK; GELLENBECK, 2011) solicitaram que os sujeitos classificassem listas de palavras/símbolos associados a construções de programação (por exemplo, condicionais e loops) com base em quão intuitivo eles pensam que são. Lawrie et al. (LAWRIE et al., 2007) pediram aos participantes que avaliassem sua confiança em seu entendimento do código.

O **tempo** que os indivíduos gastam para realizar tarefas foi medido em vários estudos. Essa categoria de variável resposta é a terceira mais utilizada nos estudos analisados, com 27 ocorrências. Há variedade na forma como os estudos medem o tempo. Para Ajami e Feitelson (AJAMI; WOODBRIDGE; FEITELSON, 2019), "o tempo é medido desde a exibição

Figura 15 – Frequência das Variáveis de Resposta.



Fonte: Elaborado pelo autor (2020)

do código até que o sujeito pressione o botão para indicar que terminou". Hofmeister et al. (HOFMEISTER; SIEGMUND; HOLT, 2019) calculou o tempo que os sujeitos gastaram olhando para partes específicas de um programa. Geffen et al. (GEFFEN; MAOZ, 2016) mediram o tempo de resposta para cada pergunta quando os sujeitos respondem a um questionário de múltipla escolha sobre o código. Em vez de medir o tempo diretamente, Malaquias et al. (MALAQUIAS et al., 2017) contaram as tentativas de corrigir erros sintáticos e semânticos no código.

Em alguns estudos, informações sobre o processo da tarefa foram coletadas em vez de seus resultados. Em particular, vários estudos empregaram equipamento especial para rastrear o que os sujeitos veem e empregaram **métricas visuais** como variáveis de resposta. Seis dos estudos analisados empregaram alguma forma de rastreamento ocular. Por exemplo, Blinkley et al. (BINKLEY et al., 2013) calcularam a atenção visual, medida como a quantidade de tempo durante o qual um sujeito está olhando para uma área particular da tela. Bauer et al. (BAUER et al., 2019) calcularam três métricas visuais: duração da fixação, taxa de fixação, ou seja, o número de fixações por segundo e amplitude sacádica, ou seja, o comprimento espacial de uma sacada, ou seja, a transição entre duas fixações. Hofmeister et al. (HOFMEISTER; SIEGMUND; HOLT, 2019) empregaram uma ferramenta de software que limita a visão dos sujeitos do código a algumas linhas de cada vez. Este quadro pode ser deslocado para cima e para baixo usando as teclas de seta para revelar diferentes partes do código. Essa abordagem é chamada de "caixa de correio". Hofmeister et al. (HOFMEISTER; SIEGMUND; HOLT, 2019) chamaram cada quadro de código de uma área de interesse (AOI) e mediram o tempo que os sujeitos gastaram em uma AOI, tempos de leitura de primeira passagem e visitas AOI.

Recentemente, alguns pesquisadores recorreram a ferramentas de monitoramento do

cérebro para entender o que acontece no cérebro dos sujeitos durante a compreensão do programa. No total, três dos estudos analisados empregaram variáveis de resposta com base no monitoramento do cérebro. Siegmund et al. (SIEGMUND et al., 2017) usaram imagem de ressonância magnética funcional (fMRI) para medir a atividade cerebral, detectando mudanças associadas ao fluxo sanguíneo. Fakhoury et al. (FAKHOURY et al., 2019b) empregaram espectroscopia de infravermelho próximo funcional (fNIRS) para medir a atividade cerebral por meio da resposta hemodinâmica dentro das estruturas físicas do cérebro. Yeh et al. (YEH et al., 2017) usou eletroencefalografia (EEG) alavancada para monitorar a atividade elétrica do cérebro. Essas variáveis de resposta foram agrupadas na categoria de **métricas do cérebro**.

Os estudos analisados diferem nas variáveis de resposta que empregaram, dependendo se visavam avaliar a *readability*, a *legibility* ou ambas. Conforme mostrado na Figura 15, os estudos de *readability* alavancaram todas as categorias de variáveis de resposta, enquanto nenhum estudo de *legibility* utilizou as métricas do cérebro. Isso não é surpreendente, considerando o número muito menor de estudos de *legibility*. Além disso, uma alta proporção de estudos de *legibility* (86%) empregou variáveis de resposta de opinião. Este não é o caso de estudos apenas sobre *readability* (43%) ou sobre ambos (65%).

A maioria dos estudos, 38 no total, empregou mais de uma variável para validar suas hipóteses. As variáveis de resposta tempo e acerto são as que mais aparecem juntas (27 estudos), seguidas de opinião e acerto (21 estudos). De maneira diferente, 16 estudos empregaram apenas uma variável de resposta, e correção e opinião foram as variáveis de resposta empregadas isoladamente. Destes, 9 estudos (16,7%) empregaram apenas opinião pessoal. O uso dessa categoria de variável de resposta isoladamente é uma clara ameaça à validade desses estudos. Além disso, nenhum estudo usou o tempo como única variável de resposta. Isso faz sentido, pois, embora o tempo seja frequentemente usado como um proxy para o esforço, não é significativo por si só.

Esses resultados confirmam que existem formas diferentes de medir a *readability* e a *legibility* de um código. Entretanto não é possível afirmar que existem formas corretas e incorretas para avaliar esses aspectos, porém algumas técnicas são utilizadas com mais frequência que outras. Por exemplo, exatidão é utilizada em muito mais estudos comparado a métricas visuais e métricas cerebrais. Embora essas duas últimas métricas demandem equipamentos específicos, seu uso pode ser benéfico principalmente para estudos que investigue a *legibility* porque conseguem capturar com maior precisão o processamento cognitivo dos sujeitos.

4.3 DISCUSSÃO DOS RESULTADOS

A revisão bibliográfica conduzida para responder a primeira pergunta de pesquisa desta dissertação teve 28 trabalhos incluídos. Esses trabalhos mostram que outras áreas como linguística, design, interação humano-computador e educação tem definições muito claras

e concisas sobre o que significa *readability* e *legibility*. Porém, a área de Engenharia de Software ainda sofre com conceitos mal definidos e/ou sobrepostos, acreditamos que *readability* e *legibility* devem ter significados claros, distintos, embora relacionados. Analisando os 28 trabalhos incluídos no referencial bibliográfico, definimos:

> *readability* como as características estruturais e semânticas do código-fonte de um programa que afetam a capacidade dos desenvolvedores de entendê-lo durante a leitura do código e *legibility* como características visuais do código-fonte de um programa, que afetam a capacidade dos desenvolvedores de identificar os elementos do código durante a leitura.

A revisão sistemática conduzida para responder a segunda pergunta de pesquisa desta dissertação teve 54 trabalhos incluídos. A grande maioria dos trabalhos excluídos estavam fora do escopo delimitado para essa pesquisa ou tratavam de assuntos como visualização e análise como ajuda para compreensão de programas. Dos artigos incluídos na revisão, 79,62% receberam uma pontuação de qualidade dentro das classificações boa ou excelente. Os locais de publicação os que mais retornaram estudos relevantes foram os International Conference on Program Comprehension, Empirical Software Engineering e International Conference on Software Engineering. Isso já era esperado uma vez que esses são eventos e periódicos referentes à compreensão de código. Entre os métodos de pesquisa mais utilizados nesses estudos estão: o experimento e o experimento controlado. Nesta revisão, 55,6% dos trabalhos são sobre *readability*.

Para avaliar a *readability* e a *legibility*, os pesquisadores conduzem estudos onde os sujeitos são solicitados a fornecer informações sobre o código, agir sobre o código ou dar sua opinião pessoal. Descobrimos que 16,7% dos estudos apenas pedem aos sujeitos que deem opiniões pessoais. Além disso, embora a maioria dos estudos analisados solicitem aos sujeitos para fornecer informações sobre o código-fonte, as informações variam amplamente por natureza. Sujeitos podem ser questionados para prever a saída de programas, identificar elementos de código, ou explicar a funcionalidade de alto nível.

Existem cinco categorias de variáveis de resposta. A correção é a mais utilizada, sendo empregada em 83,3% dos estudos analisados. Tempo e opinião também são frequentemente empregados (50% e 55,6% dos estudos, respectivamente). Um número significativo de estudos utilizou a variável tempo e/ou opinião associada à correção. Por outro lado, 30% dos estudos empregaram uma única variável de resposta. Os estudos de *readability* e *legibility* + *readability* usaram todas as categorias de variável de resposta, enquanto quase todos os estudos de *legibility* usaram a variável de resposta de opinião. Embora os estudos incluídos nessa revisão sistemática tenham utilizado um conjunto diverso de linguagens de programação, como C++, JavaScript, Java e Python, não é possível afirmar que esses resultados podem ser aplicados em qualquer linguagem de programação e paradigma de programação porque a análise apresentada não foi realizada com esse objetivo. Portanto,

pode-se afirmar apenas que esses resultados foram obtidos utilizando um conjunto de trabalhos que estudaram linguagens de programação diferentes.

5 CONSIDERAÇÕES FINAIS

Este capítulo apresenta as considerações finais do estudo que está dividido nas seguintes seções:

- Limitações e ameaças à validade: é detalhado as ameaças de validade de constructo, validade interna, validade externa e validade de conclusão.
- Trabalhos relacionados: aborda os principais trabalhos relacionados a pesquisa desenvolvida, ou seja, são analisadas revisões sistemáticas da literatura sobre compreensão de código e trabalhos sobre legibilidade de código.
- Conclusões gerais: por fim, essa seção apresenta uma síntese de todos os resultados obtidos nesta pesquisa.
- Trabalhos futuros: apresenta estudos relacionados a esse tema que está em desenvolvimento, assim como outros que podem colaborar com essa pesquisa.

5.1 LIMITAÇÕES E AMEAÇA À VALIDADE

Esse trabalho propõe uma definição para os termos de *readability* e *legibility* em Engenharia de Software e avalia como os trabalhos dessa área estudam esses dois aspectos. Todos os resultados foram oriundos de análises e sínteses de trabalhos primários. É esperado que esses trabalhos tenham sido avaliados quanto a suas ameaças à validade. No entanto, a condução dessa pesquisa tem como herança as ameaças à validade dos estudos incluídos. Além disso, essa pesquisa também apresenta outras ameaças à sua própria condução, tais como validade do constructo, validade interna, validade externa e validade de conclusão.

Validade do construto. A revisão bibliográfica conduzida para responder a primeira pergunta de pesquisa foi construída a partir de estudos primários. Esses estudos foram selecionados através de uma busca *ad hoc* no mecanismo do google scholar e connected paper. A busca no google scholar depende unicamente das palavras chaves utilizadas como parâmetros. Por outro lado, a busca na ferramenta connected paper depende do título do artigo utilizado como parâmetro. No google scholar, o número de trabalhos retornados é muito grande, portanto, foi definido como critério de inclusão para ambas as buscas, o número de citações do trabalho. Acredita-se que o número de citações de um trabalho pode refletir a sua relevância.

A revisão sistemática conduzida neste estudo também foi construída a partir dos estudos primários selecionados decorrentes dos processos de busca e seleção. A busca de estudos depende de uma *string* de busca, que foi definida com base nos artigos do conjunto semente. Escolhemos apenas conferências para pesquisar artigos para compor o

conjunto semente. Isso porque consideramos que um artigo publicado em uma revista é frequentemente uma extensão de um artigo de conferência e, em Ciência da Computação, as pesquisas mais recentes são publicadas em conferências. Além disso, o foco em conferências foi apenas como um meio de construir a *string* de busca. Artigos de periódicos também foram considerados na revisão sistemática.

Foram usados apenas três mecanismos de pesquisa para nossa pesquisa automática. Outros mecanismos, como Springer e Google Scholar, podem retornar resultados diferentes. No entanto, a maioria dos estudos do conjunto semente foi indexada pelo ACM e IEEE, e então usamos o Scopus para expandir nossa pesquisa. Além disso, durante a busca na biblioteca digital da ACM, utilizamos o Guide to the Computing Literature, que recupera recursos de outras editoras, como a Springer. Por fim, evitamos o Google Scholar nesse momento porque ele retornou mais de 17 mil documentos para nossa *string* de pesquisa. A busca automática foi realizada utilizando apenas os títulos e palavras chaves dos trabalhos. O resumo e introdução não foram considerados na busca porque em testes realizados ficou comprovado que a busca se tornaria mais abrangente ao ponto de retornar muitos trabalhos indesejados, ou seja, trabalhos que fazem parte do escopo negativo desse trabalho.

Validade interna. A revisão bibliográfica foi conduzida apenas por um pesquisador. Isso pode representar uma ameaça à validade interna porque a experiência de um único pesquisador pode tornar a busca enviesada. Contudo, foi definida uma metodologia clara e critérios de seleção consistentes. Ao final do processo, outro pesquisador revisou o procedimento metodológico e os resultados obtidos. A revisão sistemática foi conduzida por quatro pesquisadores. Isso também pode representar uma ameaça à sua validade interna, uma vez que cada pesquisador possui um determinado conhecimento e forma de conduzir suas atividades de pesquisa. Porém, cada pesquisador conduzia suas atividades de acordo com o protocolo estabelecido, sendo realizadas discussões periódicas entre todos os pesquisadores.

Outra ameaça à validade é o valor do Kappa de Cohen na etapa de inclusão dos estudos (k = 0,323), que é considerado justo (KITCHENHAM; BUDGEN; BRERETON, 2015). Este valor decorre do uso de três avaliações possíveis ("aceitável", "não aceitável" e "talvez") nessa etapa. Utilizamos "talvez" para evitar a escolha entre "aceitável" e "não aceitável" quando tínhamos dúvidas sobre a inclusão de um artigo - todos os artigos marcados com pelo menos um "talvez" foram discutidos entre todos os pesquisadores envolvidos na revisão. Outros tipos de divergência também foram discutidos entre os pesquisadores. Além disso, alguns estudos primários não relatam em detalhes as tarefas realizadas pelos sujeitos. Por causa disso, podemos ter classificado incorretamente esses estudos ao mapear os estudos para os tipos de tarefas.

Validade externa. A revisão bibliográfica se concentra em estudos que abordam as temáticas de *readability* e *legibility*, considerando aspectos conceituais. Os resultados

encontrados nessa revisão bibliográfica podem não ser representativos, uma vez que foi selecionado uma pequena parcela da população. A revisão sistemática reúne estudos que relatam comparações de formas alternativas de escrever código, considerando aspectos de baixo nível do código. Essas descobertas podem não se aplicar a outros trabalhos que estudam aspectos conceituais e que avaliam a *readability* ou *legibility* do código e que não são estudos empíricos.

Validade da conclusão. De acordo com Kitchenham et al. (KITCHENHAM; BUDGEN; BRERETON, 2015) e Wholin (WOHLIN et al., 2012), a validade da conclusão preocupa-se com a confiabilidade com que podemos tirar conclusões sobre a relação entre um tratamento e os resultados de um estudo empírico. Em uma revisão sistemática, isso se relaciona com a síntese dos dados e quão bem isso apoia as conclusões da revisão. As ameaças do estudo relacionadas a ele são, portanto, apresentadas como validade interna.

5.2 TRABALHOS RELACIONADOS

Os trabalhos relacionados à esta dissertação são revisões bibliográficas e sistemáticas sobre legibilidade de código, assim como estudos primários ou secundários sobre como aferir a compreensão e legibilidade de código em estudos com participantes. A Tabela 9 apresenta a lista dos trabalhos relacionados desse estudo. Nesse contexto, Siegmund (Siegmund, 2016) apresentou uma lista de métodos para avaliar a compreensão, como protocolos de pensar em voz alta, memorização, tarefas de compreensão e técnicas de neuroimagem. Em contraste com nosso trabalho, esta lista não é baseada em uma pesquisa bibliográfica de experimentos de compreensão. Além disso, nosso trabalho apresenta mais métodos, suas multifacetas e tipos de variáveis de resposta.

Existem muitas revisões de literatura sobre estudos empíricos em Engenharia de Software. Por exemplo, Sjøberg et al. (SJØBERG et al., 2005) relataram uma pesquisa que caracterizou quantitativamente os experimentos controlados em SE; e Kampenes et al. (KAMPENES et al., 2009) relataram uma revisão sistemática da literatura para investigar a extensão da randomização e quase-experimentação em Engenharia de Software. Esses estudos consideraram artigos no contexto geral de Engenharia de Software, e não apenas sobre compreensão de código. Caso contrário, Schröter et al. (SCHRÖTER et al., 2017), conduziram uma revisão manual da literatura para explorar estudos empíricos sobre compreensão de programas na Conferência Internacional de Compreensão de Programas (ICPC). Eles descobriram quais partes do entendimento do programa estão sendo investigadas, quais são as terminologias usadas e se a pesquisa relata suas ameaças à validade. Eles não analisaram como esses estudos avaliam a compreensão do código em experimentos.

Outras revisões da literatura estudaram recursos experimentais. Kampenes et al. (KAMPENES et al., 2007) relataram uma revisão sistemática que investigou a prática de reportar o tamanho do efeito, resumindo os tamanhos padronizados dos efeitos detectados nos experimentos. Dybå et al. (DYBå; KAMPENES; SJØBERG, 2006) apresentaram uma

revisão sistemática que realizou uma avaliação quantitativa do poder estatístico da atual pesquisa experimental de Engenharia de Software. Da mesma forma, Siegmund et al. (SIEGMUND; SCHUMANN, 2015) apresentou um catálogo de parâmetros de confusão para experimentos de compreensão baseados em uma pesquisa bibliográfica, incluindo medidas aplicadas e técnicas de controle. Semelhante ao nosso trabalho, esses estudos investigaram o desenho de experimentos de estudos primários, mas na perspectiva de tamanhos de efeito e modelos de inferência de experimentos, enquanto nosso trabalho está interessado nas práticas realizadas pelos sujeitos e seus métodos de avaliação. Além disso, as duas primeiras pesquisas consideram obras além da compreensão do código.

Em resumo, nosso estudo difere dos demais por ter realizado um levantamento bibliográfico sobre legibilidade de código e por ter realizado uma revisão sistemática seguido as etapas definidas por Kitchenham et al.(KITCHENHAM; BUDGEN; BRERETON, 2015). Objetivamos caracterizar de forma ampla o conceito de legibilidade de código, buscando assim entender o que vem sendo produzido em estudos empíricos na área de Engenharia de Software e como os conceitos são definidos em outras áreas em que o assunto é empregado. Este trabalho é parte de uma iniciativa maior onde pretendemos identificar um conjunto ideal de construções de código, expressões idiomáticas e guia de estilos que promovem a legibilidade do código.

Tabela 9 – Lista de Trabalhos Relacionados

Artigos	**Ano**	**Método de pesquisa**	**Protocolo de Pesquisa**	**Domínio**	**Verificar a compreensão/legibilidade de código**	**Sobre formas de avaliação**
(SJøBERG et al., 2005)	2005	Surveys	Sim	Metodologia de pesquisa	Não	Sim
(DYBå; KAMPENES; SJøBERG, 2006)	2006	Systematic review	Sim	Poder estatístico	Não	Sim
(KAMPENES et al., 2007)	2007	Systematic review	Sim	Significância estatística	Não	Sim
(KAMPENES et al., 2009)	2009	Systematic review	Sim	Experimento de campo	Não	Sim
(SIEGMUND; SCHUMANN, 2015)	2015	Literature survey	Não	Compreensão de Programas.	Sim	Sim

Continuação da Tabela 9						
Artigos	**Ano**	**Método de pesquisa**	**Protocolo de Pesquisa**	**Domínio**	**Verificar a compreensão/legibilidade de código**	**Sobre formas de avaliação**
(Siegmund, 2016)	2016	Survey	Não	Compreensão de Programas	Sim	Não
(SCHRÖTER et al., 2017)	2017	Systematic review	Sim	Compreensão de Programas	Sim	Sim

5.3 TRABALHOS FUTUROS

A extensão desse estudo pretende explorar ainda mais os dados coletados através da revisão sistemática. Portanto, pretende-se estudar:

- Quais são os elementos de código que estão sendo investigados em estudos centrados no ser humano que tem como objetivo comparar a *readability* ou *legibility* desses elementos;
- Considerando os sujeitos, as tarefas e as variáveis de resposta em estudos centrados no ser humano, pretende-se investigar quais elementos se mostraram mais *legible* ou *readable*;
- Considerando quais técnicas e quais *ground truth* esses estudos empregam, pretende-se investigar quais recursos são os melhores preditores de readability e legibility do código.

Além desses direcionamentos para extensão do presente estudo outras pesquisas podem ser conduzidas. Por exemplo, um estudo que explore históricos de desenvolvimento de *software* para identificar quais mudanças que um programador aplica para melhorar a legibilidade do código. Outro estudo possível é um experimento controlado com desenvolvedores para validar os resultados obtidos na revisão sistemática.

5.4 CONCLUSÕES

Embora a compreensão de código venha sendo objeto de estudo pela comunidade acadêmica nos últimos anos, pouco tem sido feito sobre quais são os fatores granulares de código que mais impactam o processo de compreensão. Os trabalhos sobre *readability* e *legibility*

também são poucos e em sua grande maioria não analisam esses fatores separadamente. Nesta seção, apresentamos nossas principais contribuições.

Esse trabalho apresentou um levantamento bibliográfico da literatura sobre os termos *readability* e *legibility* na área de educação, linguística, design, interação humano computador e Engenharia de Software. Esse levantamento teve como objetivo caracterizar de forma ampla o conceito de legibilidade de código. Os trabalhos encontrados no levantamento bibliográfico mostram que a *readability* e *legibility* são termos bem definidos em áreas como educação, linguística e design. Esses termos representam aspectos diferentes do texto ou do ambiente, porém ambos estão relacionados. Na Engenharia de Software esses dois termos muitas vezes se sobrepõem. Portanto, foi proposto que *readability*, em Engenharia de Software, corresponde as características estruturais e semânticas do código-fonte e *legibility* as características visuais do código-fonte.

Foi apresentado também uma revisão sistemática da literatura sobre como a *readability* e a *legibility* do código são avaliadas em estudos centrados em humanos que comparam diferentes maneiras de escrever código que são equivalentes. Nosso objetivo com essa revisão foi investigar quais tarefas são realizadas pelos sujeitos e quais variáveis de resposta são empregadas nesses estudos. Conclui-se que os sujeitos podem ser questionados para prever a saída de programas, identificar elementos de código, ou explicar a funcionalidade de alto nível. Além disso, foram encontradas cinco categorias de variáveis de respostas que são frequentemente utilizadas por esses estudos, são elas: exatidão, tempo, opinião, métricas visuais e métricas do cérebro. Este estudo destacou ainda as limitações dos estudos primários:

- 37% deles exerceram uma única habilidade cognitiva;
- 16,7% utilizaram apenas a opinião pessoal como variável de resposta; e
- Poucos estudos avaliaram a *readability* e a *legibility* de forma a simular cenários do mundo real, onde a compreensão do programa é parte de uma tarefa mais complexa, exigindo habilidades cognitivas de nível superior.

Pode-se inferir que as estratégias para avaliar *readability* e *legibility* podem evoluir com o objetivo de mitigar as limitações levantadas. Não existem manuais ou regras definidas para esse tipo de avaliação, assim como não é possível afirmar que existe uma definição das *features* obrigatórias ou opcionais. No entanto, através dos resultados obtidos nesse estudo é possível afirmar que o uso isolado das variáveis de resposta como tempo, corretude e opinião é recorrente em um grupo de trabalhos e que essa recorrência não é benéfica para conclusões sobre o assunto estudado.

Identifica-se, ainda, que embora pouco utilizadas, as métricas visuais e métricas cerebrais, por exemplo rastreamento ocular e FMRI, podem fornecer bons insumos sobre o processo cognitivo presente na compreensão do código. Utilizar essas variáveis de respostas

com variáveis mais comuns como corretude, tempo e opinião podem fornecer informações mais precisas sobre os fatores de *readability* e *legibility* do código. Esperasse, portanto, que novos trabalhos utilizem as definições de *readability* e *legibility* adequadamente, assim como, esperasse que os métodos de avaliação aqui encontrados sejam utilizados e melhorados através de novas pesquisas.

Isto posto, os resultados apresentados nessa dissertação podem auxiliar a condução de novos estudos na área de compreensão de código e legibilidade de código. Para estudos que buscam avaliar *readability* ou *legibility*, é recomendado que sejam utilizadas as definições propostas nessa dissertação porque elas refletem a diferença entre os elementos do código-fonte de um programa. Além disso, a análise sobre as tarefas realizadas e variáveis de resposta utilizadas também podem ser úteis na estruturação de novos ou avaliação de estudos recentes. Pesquisadores podem também utilizar o protocolo construído para condução da revisão sistemática como guia para replicações ou como modelo para novas revisões sistemáticas.

Portanto, com o objetivo de fornecer ainda mais informações que auxiliem pesquisadores na condução de estudos sobre compreensão de código e legibilidade de código, os próximos passos dessa pesquisa pretende realizar uma análise no mesmo conjunto de artigos da revisão sistemática. Essa análise terá como foco levantar as diferentes formas de escrever código e quais delas melhoram ou dificultam a *readability* e a *legibility*.

REFERÊNCIAS

ACM Digital Library. [S.l.], http://dl.acm.org/.

AGGARWAL, K. K.; SINGH, Y.; CHHABRA, J. K. An integrated measure of software maintainability. In: IEEE. *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No. 02CH37318)*. [S.l.], 2002. p. 235–241.

AJAMI, S.; WOODBRIDGE, Y.; FEITELSON, D. G. Syntax, predicates, idioms—what really affects code complexity? *Empirical Software Engineering*, Springer, v. 24, n. 1, p. 287–328, 2019.

ALMEIDA, J. R. de; CAMARGO, J. B.; BASSETO, B. A.; PAZ, S. M. Best practices in code inspection for safety-critical software. *IEEE software*, IEEE, v. 20, n. 3, p. 56–63, 2003.

ALTER, A. The power of names. *The New Yorker*, v. 3, 2013.

ARAB, M. Enhancing Program Comprehension: Formatting and Documenting. *ACM SIGPLAN Notices*, ACM, New York, NY, USA, v. 27, n. 2, p. 37–46, fev. 1992. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/130973.130975>.

ARNAOUDOVA, V.; PENTA, M. D.; ANTONIOL, G. Linguistic Antipatterns: What They Are and How Developers Perceive Them. *Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 21, n. 1, p. 104–158, fev. 2016. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-014-9350-8>.

AVIDAN, E.; FEITELSON, D. G. Effects of Variable Names on Comprehension: An Empirical Study. In: *Proceedings of the 25th International Conference on Program Comprehension (ICPC '17)*. Piscataway, NJ, USA: IEEE Press, 2017. p. 55–65. ISBN 978-1-5386-0535-6. Disponível em: <https://doi.org/10.1109/ICPC.2017.27>.

BAUER, J.; SIEGMUND, J.; PEITEK, N.; HOFMEISTER, J. C.; APEL, S. Indentation: Simply a Matter of Style or Support for Program Comprehension? In: *Proceedings of the 27th International Conference on Program Comprehension (ICPC '19)*. Piscataway, NJ, USA: IEEE Press, 2019. p. 154–164. Disponível em: <https://doi.org/10.1109/ICPC.2019.00033>.

BENANDER, A. C.; BENANDER, B. A.; PU, H. Recursion vs. Iteration: An Empirical Study of Comprehension. *Journal of Systems and Software*, v. 32, n. 1, p. 73–82, 1996. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/0164121295000437>.

BENIAMINI, G.; GINGICHASHVILI, S.; ORBACH, A. K.; FEITELSON, D. G. Meaningful Identifier Names: The Case of Single-Letter Variables. In: *Proceedings of the 25th International Conference on Program Comprehension (ICPC '17)*. Piscataway, NJ, USA: IEEE Press, 2017. p. 45–54. ISBN 978-1-5386-0535-6. Disponível em: <https://doi.org/10.1109/ICPC.2017.18>.

BINKLEY, D.; DAVIS, M.; LAWRIE, D.; MALETIC, J. I.; MORRELL, C.; SHARIF, B. The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 18, n. 2, p. 219–276, abr. 2013. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-012-9201-4>.

BINKLEY, D.; LAWRIE, D.; MAEX, S.; MORRELL, C. Identifier length and limited programmer memory. *Science of Computer Programming*, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 74, n. 7, p. 430–445, maio 2009. ISSN 0167-6423. Disponível em: <http://dx.doi.org/10.1016/j.scico.2009.02.006>.

BLINMAN, S.; COCKBURN, A. Program Comprehension: Investigating the Effects of Naming Style and Documentation. In: *Proceedings of the 6th Australasian User Interface Conference (AUIC '05)*. Newcastle, Australia: ACS, 2005. p. 73–78.

BOSU, A.; CARVER, J. C.; BIRD, C.; ORBECK, J.; CHOCKLEY, C. Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft. *IEEE Transactions on Software Engineering*, IEEE, v. 43, n. 1, p. 56–75, 2016.

BOYSEN, J. P.; KELLER, R. F. Measuring Computer Program Comprehension. In: *Proceedings of the 11th Technical Symposium on Computer Science Education (SIGCSE '80)*. New York, NY, USA: ACM, 1980. p. 92–102. ISBN 0-89791-013-3. Disponível em: <http://doi.acm.org/10.1145/800140.804619>.

BRERETON, P.; KITCHENHAM, B. A.; BUDGEN, D.; TURNER, M.; KHALIL, M. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software*, Elsevier, v. 80, n. 4, p. 571–583, 2007.

BROOKS, R. Towards a theory of the comprehension of computer programs. *International journal of man-machine studies*, Elsevier, v. 18, n. 6, p. 543–554, 1983.

BUSE, R. P.; WEIMER, W. R. A metric for software readability. In: *Proceedings of the 2008 international symposium on Software testing and analysis*. [S.l.: s.n.], 2008. p. 121–130.

BUSE, R. P.; WEIMER, W. R. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, IEEE, v. 36, n. 4, p. 546–558, 2009.

BUSE, R. P. L.; WEIMER, W. R. Learning a Metric for Code Readability. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 36, n. 4, p. 546–558, jul. 2010. ISSN 0098-5589. Disponível em: <https://doi.org/10.1109/TSE.2009.70>.

C++. [S.l.], https://isocpp.github.io/CppCoreGuidelines.

Ceccato, M.; Di Penta, M.; Nagra, J.; Falcarin, P.; Ricca, F.; Torchiano, M.; Tonella, P. The Effectiveness of Source Code Obfuscation: an Experimental Assessment. In: *Proceedings of the IEEE 17th International Conference on Program Comprehension (ICPC '09)*. [S.l.]: IEEE, 2009. p. 178–187. ISSN 1092-8138.

CHAUDHARY, B. D.; SAHASRABUDDHE, H. V. Meaningfulness as a Factor of Program Complexity. In: *Proceedings of the ACM 1980 Annual Conference (ACM '80)*. New York, NY, USA: ACM, 1980. p. 457–466. ISBN 0-89791-028-1. Disponível em: <http://doi.acm.org/10.1145/800176.810001>.

COHEN, J. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, v. 20, n. 1, p. 37–46, 1960. Disponível em: <https://doi.org/10.1177/001316446002000104>.

COLEMAN, M.; LIAU, T. L. A computer readability formula designed for machine scoring. *Journal of Applied Psychology*, American Psychological Association, v. 60, n. 2, p. 283, 1975.

CRESWELL, J. W. Projeto de pesquisa métodos qualitativo, quantitativo e misto. In: *Projeto de pesquisa métodos qualitativo, quantitativo e misto*. [S.l.: s.n.], 2010. p. 296–296.

DAKA, E.; CAMPOS, J.; FRASER, G.; DORN, J.; WEIMER, W. Modeling readability to improve unit tests. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. [S.l.: s.n.], 2015. p. 107–118.

DALE, E.; CHALL, J. S. The concept of readability. *Elementary English*, JSTOR, v. 26, n. 1, p. 19–26, 1949.

DIAS, M.; ORELLANA, D.; VIDAL, S.; MERINO, L.; BERGEL, A. Evaluating a visual approach for understanding javascript source code. In: *Proceedings of the 28th International Conference on Program Comprehension*. [S.l.: s.n.], 2020. p. 128–138.

DOLADO, J. J.; HARMAN, M.; OTERO, M. C.; HU, L. An Empirical Investigation of the Influence of a Type of Side Effects on Program Comprehension. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 29, n. 7, p. 665–670, jul. 2003. ISSN 0098-5589. Disponível em: <https://doi.org/10.1109/TSE.2003.1214329>.

DUBAY, W. H. The principles of readability. *Online Submission*, ERIC, 2004.

DYBå, T.; KAMPENES, V. B.; SJøBERG, D. I. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, v. 48, n. 8, p. 745 – 755, 2006. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584905001333>.

EASTERBROOK, S.; SINGER, J.; STOREY, M.-A.; DAMIAN, D. Selecting empirical methods for software engineering research. In: *Guide to advanced empirical software engineering*. [S.l.]: Springer, 2008. p. 285–311.

FAKHOURY, S.; MA, Y.; ARNAOUDOVA, V.; ADESOPE, O. The effect of poor source code lexicon and readability on developers' cognitive load. In: IEEE. *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. [S.l.], 2018. p. 286–28610.

FAKHOURY, S.; ROY, D.; HASSAN, S. A.; ARNAOUDOVA, V. Improving Source Code Readability: Theory and Practice. In: *Proceedings of the 27th International Conference on Program Comprehension (ICPC '19)*. Piscataway, NJ, USA: IEEE Press, 2019. p. 2–12. Disponível em: <https://doi.org/10.1109/ICPC.2019.00014>.

FAKHOURY, S.; ROY, D.; MA, Y.; ARNAOUDOVA, V.; ADESOPE, O. Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization. *Empirical Software Engineering*, Springer US, p. 1–39, ago. 2019. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-019-09751-4>.

FENG, L.; ELHADAD, N.; HUENERFAUTH, M. Cognitively motivated features for readability assessment. In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*. [S.l.: s.n.], 2009. p. 229–237.

FLESCH, R. A new readability yardstick. *Journal of applied psychology*, American Psychological Association, v. 32, n. 3, p. 221, 1948.

GEFFEN, Y.; MAOZ, S. On method ordering. In: IEEE. *2016 IEEE 24th international conference on program comprehension (ICPC)*. [S.l.], 2016. p. 1–10.

GOPSTEIN, D.; IANNACONE, J.; YAN, Y.; DELONG, L.; ZHUANG, Y.; YEH, M. K.-C.; CAPPOS, J. Understanding Misunderstandings in Source Code. In: *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '17)*. New York, NY, USA: ACM, 2017. p. 129–139. ISBN 978-1-4503-5105-8. Disponível em: <http://doi.acm.org/10.1145/3106237.3106264>.

GOUGH, P. B.; TUNMER, W. E. Decoding, Reading, and Reading Disability. *Remedial and special education*, SAGE Publications Sage CA: Los Angeles, CA, v. 7, n. 1, p. 6–10, 1986.

GRECO, M.; STUCCHI, N.; ZAVAGNO, D.; MARINO, B. On the portability of computer-generated presentations: The effect of text-background color combinations on text legibility. *Human factors*, SAGE Publications Sage CA: Los Angeles, CA, v. 50, n. 5, p. 821–833, 2008.

HERZOG, T. R.; LEVERICH, O. L. Searching for legibility. *Environment and behavior*, Sage Publications, v. 35, n. 4, p. 459–477, 2003.

HOFMEISTER, J. C.; SIEGMUND, J.; HOLT, D. V. Shorter Identifier Names Take Longer to Comprehend. *Empirical Software Engineering*, Kluwer Academic Publishers, Norwell, MA, USA, v. 24, n. 1, p. 417–443, fev. 2019. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-018-9621-x>.

HOOVER, W. A.; GOUGH, P. B. The simple view of reading. *Reading and writing*, Springer, v. 2, n. 2, p. 127–160, 1990.

IEEE Explore. [S.l.], http://ieeexplore.ieee.org/.

ISELIN, E. R. Conditional statements, looping constructs, and program comprehension: an experimental study. *International Journal of Man-Machine Studies*, Academic Press Ltd., London, UK, UK, v. 28, n. 1, p. 45–66, jan. 1988. ISSN 0020-7373. Disponível em: <http://dx.doi.org/10.1016/S0020-7373(88)80052-X>.

JAVASCRIPT. [S.l.], https://google.github.io/styleguide.

JBARA, A.; FEITELSON, D. G. On the Effect of Code Regularity on Comprehension. In: *Proceedings of the 22nd International Conference on Program Comprehension (ICPC '14)*. New York, NY, USA: ACM, 2014. p. 189–200. ISBN 978-1-4503-2879-1. Disponível em: <http://doi.acm.org/10.1145/2597008.2597140>.

JBARA, A.; FEITELSON, D. G. How programmers read regular code: a controlled experiment using eye tracking. *Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 22, n. 3, p. 1440–1477, jun. 2017. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-016-9477-x>.

KAMPENES, V. B.; DYBa, T.; HANNAY, J. E.; SJØBERG, D. I. K. A systematic review of quasi-experiments in software engineering. *Inf. Softw. Technol.*, Butterworth-Heinemann, USA, v. 51, n. 1, p. 71–82, jan. 2009. ISSN 0950-5849. Disponível em: <https://doi.org/10.1016/j.infsof.2008.04.006>.

KAMPENES, V. B.; DYBå, T.; HANNAY, J. E.; SJøBERG, D. I. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, v. 49, n. 11, p. 1073 – 1086, 2007. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584907000195>.

KASTO, N.; WHALLEY, J. Measuring the difficulty of code comprehension tasks using software metrics. In: *Proceedings of the 15th Australasian Computing Education Conference - Volume 136 (ACE '13)*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2013. p. 59–65. ISBN 978-1-921770-21-0. Disponível em: <http://dl.acm.org/citation.cfm?id=2667199.2667206>.

KAUARK, F. d. S.; MANHÃES, F. C.; MEDEIROS, C. H. Metodologia da pesquisa: um guia prático. Via Litterarum, 2010.

KEELE, S. et al. *Guidelines for performing systematic literature reviews in software engineering*. [S.l.], 2007.

KITCHENHAM, B.; BRERETON, O. P.; BUDGEN, D.; TURNER, M.; BAILEY, J.; LINKMAN, S. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, Elsevier, v. 51, n. 1, p. 7–15, 2009.

KITCHENHAM, B. A.; BUDGEN, D.; BRERETON, P. *Evidence-based software engineering and systematic reviews*. [S.l.]: CRC press, 2015. v. 4.

KITCHENHAM, B. A.; DYBA, T.; JORGENSEN, M. Evidence-based software engineering. In: IEEE. *Proceedings. 26th International Conference on Software Engineering*. [S.l.], 2004. p. 273–281.

KLARE, G. R. Assessing readability. *Reading research quarterly*, JSTOR, p. 62–102, 1974.

Kleinschmager, S.; Robbes, R.; Stefik, A.; Hanenberg, S.; Tanter, E. Do Static Type Systems Improve the Maintainability of Software Systems? An Empirical Study. In: *Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC '12)*. [S.l.: s.n.], 2012. p. 153–162.

KREBS, W. K.; XING, J.; JR, A. J. A. A simple tool for predicting the readability of a monitor. In: SAGE PUBLICATIONS SAGE CA: LOS ANGELES, CA. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. [S.l.], 2002. v. 46, n. 17, p. 1659–1663.

LAUGHLIN, G. H. M. Smog grading-a new readability formula. *Journal of reading*, JSTOR, v. 12, n. 8, p. 639–646, 1969.

LAWRIE, D.; MORRELL, C.; FEILD, H.; BINKLEY, D. Effective identifier names for comprehension and memory. *Innovations in Systems and Software Engineering*, v. 3, n. 4, p. 303–318, dez. 2007. ISSN 1614-5054. Disponível em: <https://doi.org/10.1007/s11334-007-0031-2>.

LETOVSKY, S. Cognitive processes in program comprehension. *Journal of Systems and software*, Elsevier, v. 7, n. 4, p. 325–339, 1987.

LIN, J.-C.; WU, K.-C. Evaluation of Software Understandability Based On Fuzzy Matrix. In: *Proceedings of the 2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*. [S.l.: s.n.], 2008. p. 887–892.

LITTMAN, D. C.; PINTO, J.; LETOVSKY, S.; SOLOWAY, E. Mental models and software maintenance. *Journal of Systems and Software*, Elsevier, v. 7, n. 4, p. 341–355, 1987.

LOVE, T. An Experimental Investigation of the Effect of Program Structure on Program Understanding. *ACM SIGOPS Operating Systems Review – Proceedings of an ACM conference on Language design for reliable software*, ACM, New York, NY, USA, v. 11, n. 2, p. 105–113, mar. 1977. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/390018.808317>.

LYNCH, K. Reconsidering the image of the city. In: *Cities of the Mind*. [S.l.]: Springer, 1984. p. 151–161.

MAFRA, S. N.; TRAVASSOS, G. H. Estudos primários e secundários apoiando a busca por evidência em engenharia de software. *Relatório Técnico, RT-ES*, v. 687, n. 06, 2006.

MALAQUIAS, R.; RIBEIRO, M.; BONIFáCIO, R.; MONTEIRO, E.; MEDEIROS, F.; GARCIA, A.; GHEYI, R. The Discipline of Preprocessor-Based Annotations Does #ifdef TAG n't #endif Matter. In: *Proceedings of the 25th International Conference on Program Comprehension (ICPC '17)*. Piscataway, NJ, USA: IEEE Press, 2017. p. 297–307. ISBN 978-1-5386-0535-6. Disponível em: <https://doi.org/10.1109/ICPC.2017.41>.

MARCONI, M. d. A.; LAKATOS, E. M. *Metodologia científica*. [S.l.]: Atlas São Paulo, 2004. v. 4.

MAYRHAUSER, A. von; VANS, A. M. From code understanding needs to reverse engineering tool capabilities. In: IEEE. *Proceedings of 6th International Workshop on Computer-Aided Software Engineering*. [S.l.], 1993. p. 230–239.

MAćKOWIAK, M.; NAWROCKI, J.; OCHODEK, M. On Some End-User Programming Constructs and Their Understandability. *Journal of Systems and Software*, v. 142, p. 206–222, 2018. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121218300633>.

MCCALLUM, D. R.; PETERSON, J. L. Computer-based readability indexes. In: *Proceedings of the ACM'82 Conference*. [S.l.: s.n.], 1982. p. 44–48.

MEDEIROS, F.; LIMA, G.; AMARAL, G.; APEL, S.; KäSTNER, C.; RIBEIRO, M.; GHEYI, R. An investigation of misunderstanding code patterns in C open-source software projects. *Empirical Software Engineering*, Kluwer Academic Publishers, Norwell, MA, USA, v. 24, n. 4, p. 1693–1726, ago. 2019. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-018-9666-x>.

MIARA, R. J.; MUSSELMAN, J. A.; NAVARRO, J. A.; SHNEIDERMAN, B. Program Indentation and Comprehensibility. *Communications of the ACM*, ACM, New York, NY, USA, v. 26, n. 11, p. 861–867, nov. 1983. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/182.358437>.

MOONEN, L. Exploring software systems. In: IEEE. *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* [S.l.], 2003. p. 276–280.

MYNATT, B. T. The effect of semantic complexity on the comprehension of program modules. *International journal of man-machine studies*, Elsevier, v. 21, n. 2, p. 91–103, 1984.

OLIVEIRA, D.; BRUNO, R.; MADEIRAL, F.; CASTOR, F. Evaluating code readability and legibility: An examination of human-centric studies. In: IEEE. *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.], 2020. p. 348–359.

OMAN, P. W.; COOK, C. R. A Paradigm for Programming Style Research. *ACM SIGPLAN Notices*, ACM, New York, NY, USA, v. 23, n. 12, p. 69–78, dez. 1988. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/57669.57675>.

OMAN, P. W.; COOK, C. R. Typographic Style is More than Cosmetic. *Communications of the ACM*, ACM, New York, NY, USA, v. 33, n. 5, p. 506–520, maio 1990. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/78607.78611>.

O'NEAL, M. B.; EDWARDS, W. R. Complexity Measures for Rule-Based Programs. *IEEE Transactions on Knowledge and Data Engineering*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 6, n. 5, p. 669–680, out. 1994. ISSN 1041-4347. Disponível em: <https://doi.org/10.1109/69.317699>.

O'NEILL, M. J. Evaluation of a conceptual model of architectural legibility. *Environment and Behavior*, Sage Publications Sage CA: Thousand Oaks, CA, v. 23, n. 3, p. 259–284, 1991.

PENNINGTON, N. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology*, Elsevier, v. 19, n. 3, p. 295–341, 1987.

POSNETT, D.; HINDLE, A.; DEVANBU, P. A simpler model of software readability. In: *Proceedings of the 8th working conference on mining software repositories*. [S.l.: s.n.], 2011. p. 73–82.

ROEHM, T. Two user perspectives in program comprehension: end users and developer users. In: IEEE. *2015 IEEE 23rd International Conference on Program Comprehension*. [S.l.], 2015. p. 129–139.

SANTOS, D.; SANT'ANNA, C. How does feature dependency affect configurable system comprehensibility? In: IEEE. *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. [S.l.], 2019. p. 19–29.

SANTOS, R. M. a. dos; GEROSA, M. A. Impacts of Coding Practices on Readability. In: *Proceedings of the 26th Conference on Program Comprehension (ICPC '18)*. New York, NY, USA: ACM, 2018. p. 277–285. ISBN 978-1-4503-5714-2. Disponível em: <http://doi.acm.org/10.1145/3196321.3196342>.

SCALABRINO, S.; BAVOTA, G.; VENDOME, C.; LINARES-VÁSQUEZ, M.; POSHYVANYK, D.; OLIVETO, R. Automatically assessing code understandability: How far are we? In: IEEE. *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. [S.l.], 2017. p. 417–427.

SCALABRINO, S.; BAVOTA, G.; VENDOME, C.; LINARES-VáSQUEZ, M.; POSHYVANYK, D.; OLIVETO, R. Automatically Assessing Code Understandability. *IEEE Transactions on Software Engineering*, p. 1–1, 2019. ISSN 2326-3881.

SCALABRINO, S.; LINARES-VáSQUEZ, M.; OLIVETO, R.; POSHYVANYK, D. A Comprehensive Model for Code Readability. *Journal of Software: Evolution and Process*, v. 30, n. 6, p. e1958, 2018. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1958>.

SCANNIELLO, G.; RISI, M. Dealing with Faults in Source Code: Abbreviated vs. Full-Word Identifier Names. In: *Proceedings of the 2013 IEEE International Conference on Software Maintenance (ICSM '13)*. USA: IEEE Computer Society, 2013. p. 190–199. ISBN 9780769549811. Disponível em: <https://doi.org/10.1109/ICSM.2013.30>.

SCHANKIN, A.; BERGER, A.; HOLT, D. V.; HOFMEISTER, J. C.; RIEDEL, T.; BEIGL, M. Descriptive compound identifier names improve source code comprehension. In: IEEE. *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. [S.l.], 2018. p. 31–3109.

SCHRÖTER, I.; KRÜGER, J.; SIEGMUND, J.; LEICH, T. Comprehending studies on program comprehension. In: IEEE. *Proceedings of the IEEE/ACM 25th International Conference on Program Comprehension (ICPC '17)*. [S.l.], 2017. p. 308–311.

SCHULZE, S.; LIEBIG, J.; SIEGMUND, J.; APEL, S. Does the Discipline of Preprocessor Annotations Matter? A Controlled Experiment. In: *Proceedings of the 12th International Conference on Generative Programming: Concepts & Experiences (GPCE '13)*. New York, NY, USA: ACM, 2013. p. 65–74. ISBN 978-1-4503-2373-4. Disponível em: <http://doi.acm.org/10.1145/2517208.2517215>.

SCOPUS. [S.l.], http://www.scopus.com/.

SHARGABI, A. A.; ALJUNID, S. A.; ANNAMALAI, M.; ZIN, A. M. Performing tasks can improve program comprehension mental model of novice developers: An empirical approach. In: *Proceedings of the 28th International Conference on Program Comprehension*. [S.l.: s.n.], 2020. p. 263–273.

SHEEDY, J. E.; SUBBARAM, M. V.; ZIMMERMAN, A. B.; HAYES, J. R. Text legibility and the letter superiority effect. *Human factors*, SAGE Publications Sage CA: Los Angeles, CA, v. 47, n. 4, p. 797–815, 2005.

SHNEIDERMAN, B.; MAYER, R. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer & Information Sciences*, Springer, v. 8, n. 3, p. 219–238, 1979.

Siegmund, J. Program comprehension: Past, present, and future. In: *Proceedings of the IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER '16)*. [S.l.: s.n.], 2016. v. 5, p. 13–20.

SIEGMUND, J.; PEITEK, N.; PARNIN, C.; APEL, S.; HOFMEISTER, J.; KäSTNER, C.; BEGEL, A.; BETHMANN, A.; BRECHMANN, A. Measuring Neural Efficiency of Program Comprehension. In: *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '17)*. New York, NY, USA: ACM, 2017. p. 140–150. ISBN 978-1-4503-5105-8. Disponível em: <http://doi.acm.org/10.1145/3106237.3106268>.

SIEGMUND, J.; SCHUMANN, J. Confounding parameters on program comprehension: a literature survey. *Empirical Software Engineering*, Springer, v. 20, n. 4, p. 1159–1192, 2015.

SJøBERG, D.; HANNAY, J.; HANSEN, O.; KAMPENES, V.; KARAHASANOVIC, A.; LIBORG, N.-K.; REKDAL, A. A survey of controlled experiments in software engineering. *TSE*, v. 31, p. 733– 753, 10 2005.

SMITH, S. L. Letter size and legibility. *Human factors*, SAGE Publications Sage CA: Los Angeles, CA, v. 21, n. 6, p. 661–670, 1979.

SOLOWAY, E.; EHRLICH, K. Empirical studies of programming knowledge. *IEEE Transactions on software engineering*, Ieee, n. 5, p. 595–609, 1984.

STEFIK, A.; GELLENBECK, E. Empirical studies on programming language stimuli. *Software Quality Journal*, Kluwer Academic Publishers, Hingham, MA, USA, v. 19, n. 1, p. 65–99, mar. 2011. ISSN 0963-9314. Disponível em: <http://dx.doi.org/10.1007/s11219-010-9106-7>.

STEFIK, A.; SIEBERT, S. An Empirical Investigation into Programming Language Syntax. *ACM Transactions on Computing Education (TOCE)*, ACM, New York, NY, USA, v. 13, n. 4, p. 19:1–19:40, nov. 2013. ISSN 1946-6226. Disponível em: <http://doi.acm.org/10.1145/2534973>.

STOREY, M.-A. Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, Springer, v. 14, n. 3, p. 187–208, 2006.

STRIZVER, I. *Type Rules: The designer's guide to professional typography*. [S.l.]: John Wiley & Sons, 2013.

SYKES, F.; TILLMAN, R. T.; SHNEIDERMAN, B. The Effect of Scope Delimiters on Program Comprehension. *Software: Practice and Experience*, v. 13, n. 9, p. 817–824, 1983. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380130908>.

TEASLEY, B. E. The effects of naming style and expertise on program comprehension. *International Journal of Human-Computer Studies*, Academic Press, Inc., Duluth, MN, USA, v. 40, n. 5, p. 757–770, maio 1994. ISSN 1071-5819. Disponível em: <http://dx.doi.org/10.1006/ijhc.1994.1036>.

TEKFI, C. Readability formulas: An overview. *Journal of documentation*, MCB UP Ltd, 1987.

Tenny, T. Program Readability: Procedures Versus Comments. *IEEE Transactions on Software Engineering*, v. 14, n. 9, p. 1271–1279, set. 1988. ISSN 2326-3881.

TROCKMAN, A.; CATES, K.; MOZINA, M.; NGUYEN, T.; KäSTNER, C.; VASILESCU, B. "Automatically Assessing Code Understandability" Reanalyzed: Combined Metrics Matter. In: *Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18)*. New York, NY, USA: ACM, 2018. p. 314–318. ISBN 978-1-4503-5716-6. Disponível em: <http://doi.acm.org/10.1145/3196398.3196441>.

WANG, X.; POLLOCK, L.; VIJAY-SHANKER, K. Automatic Segmentation of Method Code into Meaningful Blocks: Design and Evaluation. *Journal of Software: Evolution and Process*, v. 26, n. 1, p. 27–49, 2014. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1581>.

WEISMAN, J. Evaluating architectural legibility: Way-finding in the built environment. *Environment and behavior*, Sage Publications Sage CA: Thousand Oaks, CA, v. 13, n. 2, p. 189–204, 1981.

WIEDENBECK, S. Beacons in computer program comprehension. *International Journal of Man-Machine Studies*, Academic Press Ltd., London, UK, UK, v. 25, n. 6, p. 697–709, dez. 1986. ISSN 0020-7373. Disponível em: <http://dx.doi.org/10.1016/S0020-7373(86)80083-9>.

WIEDENBECK, S. The initial stage of program comprehension. *International Journal of Man-Machine Studies*, Academic Press Ltd., London, UK, UK, v. 35, n. 4, p. 517–540, nov. 1991. ISSN 0020-7373. Disponível em: <http://dx.doi.org/10.1016/S0020-7373(05)80090-2>.

WIESE, E. S.; RAFFERTY, A. N.; FOX, A. Linking Code Readability, Structure, and Comprehension among Novices: It's Complicated. In: *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '19)*. Piscataway, NJ, USA: IEEE Press, 2019. p. 84–94. Disponível em: <https://doi.org/10.1109/ICSE-SEET.2019.00017>.

WIESE, E. S.; RAFFERTY, A. N.; KOPTA, D. M.; ANDERSON, J. M. Replicating Novices' Struggles with Coding Style. In: *Proceedings of the 27th International Conference on Program Comprehension (ICPC '19)*. Piscataway, NJ, USA: IEEE Press, 2019. p. 13–18. Disponível em: <https://doi.org/10.1109/ICPC.2019.00015>.

WOHLIN, C.; AURUM, A. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering*, Springer, v. 20, n. 6, p. 1427–1455, 2015.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012.

WOODFIELD, S. N.; DUNSMORE, H. E.; SHEN, V. Y. The Effect of Modularization and Comments on Program Comprehension. In: *Proceedings of the 5th International Conference on Software Engineering (ICSE '81)*. Piscataway, NJ, USA: IEEE Press,

1981. p. 215–223. ISBN 0-89791-146-6. Disponível em: <http://dl.acm.org/citation.cfm?id=800078.802534>.

Xia, X.; Bao, L.; Lo, D.; Xing, Z.; Hassan, A. E.; Li, S. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering*, v. 44, n. 10, p. 951–976, 2018.

YEH, M. K.-C.; GOPSTEIN, D.; YAN, Y.; ZHUANG, Y. Detecting and Comparing Brain Activity in Short Program Comprehension Using EEG. In: *Proceedings of the 2017 IEEE Frontiers in Education Conference (FIE '17)*. Los Alamitos, CA, USA: IEEE Computer Society, 2017. p. 1–5. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/FIE.2017.8190486>.

ZUFFI, S.; BRAMBILLA, C.; BERETTA, G.; SCALA, P. Human computer interaction: Legibility and contrast. In: IEEE. *14th International Conference on Image Analysis and Processing (ICIAP 2007)*. [S.l.], 2007. p. 241–246.

ZUFFI, S.; BRAMBILLA, C.; BERETTA, G. B.; SCALA, P. Understanding the readability of colored text by crowd-sourcing on the web. *Journal of Color Research and Application*, 2009.

APÊNDICE A – PROTOCOLO DA REVISÃO SISTEMÁTICA

A.1 INTRODUCTION

This systematic review aims to identify and classify the existing concepts in the literature on hard-to-understand code. This review is relevant because understanding code is an important and time-consuming activity in software system development. Code that is difficult to understand can negatively impact several system development-related activities such as testing, code review, and maintenance [1]. Therefore, it is deduced that hard-tounderstand code can directly affect the productivity and overall quality of a software [2]. Thus, studies are needed to find ways to improve code comprehension. This systematic review is part of Reydne Bruno's master dissertation proposal and part of the theoric referential of Delano Oliveira's doctoral thesis. This work addresses a certain level of understanding, at this level we are taking into account the issues of legibility and readability.

A.2 SEARCH STRATEGY

In this section we present the strategy used to search the primary studies of this systematic review. We first present the research questions so that we could delimit the scope of the review and also to guide our search. Then we present our generic search string, followed by the search engines we used as well as how the generic search string was adapted according to the syntax of each search engine.

A.2.1 Research Question

We found several works that seek to clarify the relationship between code readability within the scope of code comprehension. In order to have a specific view and better understand the theme, we defined some research questions.

RQ1: What programming constructs and coding idioms improve code readability in program comprehension tasks?

RQ2: What coding style elements improve code legibility in program comprehension tasks?

RQ3: How do human-centric studies evaluate whether a certain approach is more readable or legible than another one?

RQ3a: What are the tasks performed by human subjects in empirical studies?

RQ3b: What are the dependent variables?

RQ3c: Who are the subjects in the experiments (number of subjects and type (e.g. student, researcher, developer, ...))?

RQ4: In tool-centric studies, how are predicted readability and predicted legibility evaluated?

In RQ2, by "coding style"we mean issues intrinsic to the program source code that affect is appearance and organization but not its behavior nor the solutions it expresses spacing, formatting, organization, naming conventions.

A.2.2 Search String

In the manual search, we will search for papers in the following software engineering-related conferences: ACM/IEEE International Conference on Software Engineering (ICSE), ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), Mining Software Repositories (MSR), IEEE/ACM International Conference on Automated Software Engineering (ASE), International Symposium on Software Testing and Analysis (ISSTA), ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE International Conference on Program Comprehension (ICPC) e International Conference on Software Analysis, Evolution, and Reengineering (SANER), from 2016 to June 2019.

The title and keywords of the seed papers were analyzed, and then we extracted the general terms related to our research questions, e.g. “code comprehension”. We chose general terms as a conservative way to gather as much as papers that could fit in our study; otherwise, we would delimit our set of studies based on specific topics. The resulting terms we gathered composes our generic search string, which returns a document if at least one term is in the title or keywords of the document:

String: Title("code comprehensioncode understandabilitycode understandingcode readabilityprogram comprehension program understandability program understanding program readability programmer experience readability ") OR Keywords (code understandingcode understandingcode understandingcode readabilityprogram comprehensionprogram understandability "program readabilityprogrammer experiencereadability").

In our search string we did not include code legibility because in some preliminary searches we realized that the results returned were not directing the focus of our research questions. We also do not include software readability and code readability because this theme is very broad, thus leaving our main focus.

A.2.3 Search Engine

In this section we present the search engines that we consider for this systematic review, as well as the adaptation of the generic search string according to the syntax of each engine. These search engines were chosen because they are the main search engines for software engineering jobs.

A.2.3.1 ACM

We used the advanced search provided by ACM at http://portal.acm.org with the following string:

acmdlTitle :("code comprehensioncode understandabilitycode understandingcode readabilityprogram comprehensionprogram understandabilityprogram understanding program readabilityprogrammer experiencereadability") OR keywords .author.keyword :("code comprehensioncode understandabilitycode understandingcode readability program comprehensionprogram understandingabilityprogram understandingprogram readabilityprogrammer experiencereadability")

We used the Guide to the Computing Literature, not just what's at the ACM. Therefore, lots of resources from Springer, for example, are included, such as Empirical Software Engineering.

A.2.3.2 IEEE

We used the advanced search provided by IEEE at http: //ieeexplore.ieee.org/Xplore/home.jsp with the following string:

(("Document Title": "code comprehension"OR "code understandability"OR "code understanding"OR "code readability"OR "program comprehension"OR "program understandability"OR "program understanding"OR "program readability "OR"programmer experience "OR"readability "OR ("Author Keywords ":"code comprehension "OR"code understandability "OR"code understanding "OR"code readability "OR"program comprehension "OR"program understandability "OR"program understanding "OR"program readability "OR"programmer experience "OR"readability "))

A.2.3.3 SCOPUS

We used the advanced search provided by SCOPUS at https://www.scopus.com/search/form.uri?disp with the following string:

TITLE ("programmer experience") OR TITLE ("program comprehension") OR TITLE ("code comprehension") TITLE ("code readability") OR TITLE ("code understandability") OR KEY ("code comprehension")) OR KEY ("program comprehension") OR KEY ("program understandability") OR KEY ("readability")

A.2.3.4 Duplicate removal

Duplicates were removed in two steps. First, we ran a script that removed articles whose names were exactly equal. Second, we ordered all the papers by title and manually checked every title manually removing the duplicates.

A.3 SELECTION OF STUDIES

Before applying the inclusion and exclusion criteria, we refined our set of works. This was necessary because we realized that some documents returned by search are not part of the scope of our work. These documents are (i) one page articles; (ii) technical reports; (iii) articles from other areas of study; and (iv) duplicate papers.

A.3.1 Inclusion and Exclusion Criteria

We would have a very large number of works that are not related to our research by using only the procedures defined in the search strategy. Therefore we defined and applied some inclusion and exclusion criteria to discard studies that are not relevant to our research. These criteria should be highlighted in the title, keywords, or abstract of the papers.

A.3.1.1 Inclusion

ID	**Description**
IC1	Scope: The study must be primarily related to the topics of code readability, legibility, understandability, or hard-to-understand code.
IC2	Methodology: The study must be/contain at least one empirical study, such as controlled experiment, quasi-experiment, case study, or survey.
IC3	Comparison: The study must compare alternative programming language constructs, coding idioms, or coding styles. The study may compare those elements directly, e.g., controlled experiments measuring the performance of subjects in code reading tasks, or indirectly, e.g., case studies collecting metrics related to these elements and validating them against some criteria (e.g., expert judgement) to assess readability.

Continuação da Tabela	
ID	**Description**
IC4	Granularity: The study must target fine-grained program elements and low-level/limited-scope programming activities. Not design or comments, but implementation.

A.3.1.2 Exclusion

ID	**Description**
EC1	Out of primary scope: The study is not primarily related to source code comprehension and readability, does not involve any kind of comparison of program characteristics, neither direct or indirect, or is clearly irrelevant to our research questions. For instance, a study focusing on system architecture, high-level design, documentation, or dependencies between system parts (higher-level issues) is excluded.
The paper IS about source code comprehension and readability, BUT ...	
EC2	The study is not a full paper (e.g. master dissertations, doctoral theses, course completion monographs, short papers) or is not written in English. As a rule of thumb, we consider that full papers must be at least 5 pages long.
EC4	The study is about readability metrics without experimental evaluation.
EC5	The study is about program understanding aids, such as visualizations or other forms of analysis or sensory aids (e.g. graphs, trace-based execution, code summarization, specification mining, reverse engineering).
EC6	The study has a focus on accessibility, e.g., study focusing on people with visual impairments or neurodiverse developers (as Autism Spectrum Disorder).

A.3.2 Procedure to Select the Studies

After defining the research question, the strategy used to search for the primary studies, and the inclusion and exclusion criteria, the process to select papers is applied. This

activity is divided into the following steps:

- The search must be performed on each engine (ACM, IEEE, Scopus) according to the defined search string. This will return an X set of papers.
- Due to the large number of X articles returned by the mills, a data cleanup will be performed where duplicate non-article jobs will be removed. This will result in a Z = X-Y set of jobs; Duplicate removal: Duplicates were removed in two steps. First, we ran a script that removed articles whose names were exactly the same. Second, we ordered all the papers by title and manually checked every title manually removing the duplicates.
- The Z papers went through a screening stage, where the title and perhaps summary are analyzed considering the exclusion and inclusion criteria. This occurs as follows:
  - Exclusion criteria must be applied by two or more researchers, each of whom must perform an independent analysis. Each analysis should then be audited in pairs or by a third researcher who is not responsible for that analysis. This is necessary for inconsistencies to be repaired and for consensus to be reached.
  - Inclusion criteria should also be applied by two or more researchers. Each must do their independent analysis, and then each analysis must also be audited in pairs or by another researcher who is not responsible for the analysis.
- The F = Z-W papers that met all the inclusion criteria will be read in full, which will allow for quality assessment and data extraction.

A.4 QUALITY ASSESSMENT

In this section we explain how the papers that have passed in the inclusion and exclusion criteria have been filtered. Thus, we seek to evaluate their quality through what is presented in the methodology and results of each work. This quality assessment step is based on the criteria addressed by the book “Evidencebased software engineering and systematic reviews” by Kitchenhamv, David and Pearl [3]. According to the authors’ recommendation, an evaluation instrument was elaborated considering among the exposed criteria those that best address the research question of this systematic review. The table below presents the cited instrument.

ID	**Questions for quality assessment for study**
Questions about design	
QA1	Are the aims clearly stated?
QA2	Are the independent variables adequately defined?

Continuação da Tabela	
ID	**Questions for quality assessment for study**
QA3	Are the dependent variables adequately defined?
Questions about analysis and rigor	
QA4	Are the data collection methods adequately described?
QA5	Are the study participants or observational units adequately described? For example, SE experience, type (student, practitioner, consultant), nationality, task experience and other relevant variables.
QA6	Were the collected data adequately described (e.g., descriptive statistics)?
QA7	Did the study employ statistical methods to analyze the data?
Questions about conclusion	
QA8	Is there a clear statement of findings, i.e., are all study questions answered?
QA9	Do the researchers explain the threats to the study validity?

During the assessment of each work, it should be highlighted in the paper the sentences that answer each of those questions. At least the following sections should be read: Introduction, Methodology and Results. It is optional to read the full text in order to clarify any doubts on any criteria.

A.5 DATA EXTRACTION

For the data extraction, we use the google form platform combined with google sheets. Through this combination, we are able to organize and optimize the data extraction activity. The table below shows the information extracted by the form and includes questions that answer the research questions. Some questions were also added to obtain standard data such as titles, authors, journal, publication details, etc.

	Question	**Possible answers**
1)	Title	
2)	Authors	
3)	Publication Year	
4)	Publication Venue	

Continuação da Tabela		
	Question	**Possible answers**
5)	What is the topic of the study?	- Readability - Legibility - Both
6)	What exactly is being compared?	Code construct more generally, what are the independent variables? constructs (for vs. while loops, ifs vs. ternary expressions, etc.), Coding style formatting (indentation, spacing, how to lay out the code, naming conventions) coding style (same constructs, different ways of using them, e.g., reassigning to the same variable vs. assigning to different variables)
7)	What code attributes are captured by them (independent variables) ?	Blank lines, number of keywords, code length, number of atoms of confusion or whether the code exhibits or not atoms of confusion.
8)	What were the targeted languages?	Java, C, C++, Haskel, Python, others.
9)	What are the research questions of the study?	In case there are no explicitly stated research questions, we can just write the general goal of the study.
Assessment method		
10)	What is the assessment methodology?	(Quasi-) Experiment, opinion survey, some form of case study.

Continuação da Tabela		
	Question	**Possible answers**
11)	What was the sample of the evaluation?	Students, professionals, software systems.
12)	Human subject/system subject	
13)	What activities did the subjects have to perform? If the sample consists of software systems, what activities involving these systems were conducted?	Time to complete, number of errors, brain activity, pieces of the code that are being inspected/eye tracking, opinions of grades, preexisting readability metrics (e.g., Buse and Weimer's work), etc.
14)	What attributes are being measured (dependent variables)?	(e.g., if it's number of errors, how does the paper assess whether the subject erred? In case it is eye tracking, what metrics/qualitative evaluation is being derived from the eye tracking information?)
15)	Did the paper use statistics? How? What has been tested? What kind of data correction was applied? If multiple tests were applied for the different RQs, please discuss all of them.	
Results		
16)	What did the paper find out? What are the answers to the research questions?	
17)	How the statistics tests infers in the results?	

A.6 SYNTHESIS OF DATA

In this stage of the systematic review, the results of the collected works must be tabulated so that they can be compared and combined (KITCHENHAM, 2015). The data extracted from the works are organized in the google sheets tool that allows us to organize the works according to what we extracted through google form, the platform we use to do the extraction.

According to Kitchenham (2007) the synthesis can be quantitative or qualitative, in the first case the data must be used statistical techniques. In our work, due to the inhomogeneity of the works included, we present a qualitative synthesis. In this type of synthesis, the objective is to integrate results and conclusions in natural language (KITCHENHAM, 2007).

A.7 REFERENCES

[1] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. Confusion detection in code reviews. In 2017 IEEE International Conference on Software Maintenance and Evolution, pages 549–553, Shanghai, China, September 2017.
[2] Eran Avidan and Dror G. Feitelson. Effects of variable names on comprehension: an empirical study. In Proceedings of the 25th International Conference on Comprehension Program, pages 55–65, Buenos Aires, Argentina, May 2017.
[3] Kitchenham, Barbara Ann, David Budgen, and Pearl Brereton. Evidence-based software engineering and systematic reviews. Vol. 4. CRC press, 2015.
[4] Keele, S., 2007. Guidelines for performing systematic literature reviews in software engineering (Vol. 5). Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

APÊNDICE B – LISTA DE ARTIGOS ACEITOS NA REVISÃO SISTEMÁTICA

#	Title	Author	ACM	IEEE	Scopus
1	a comprehensive model for code readability	Scalabrino, S., Linares-Vásquez, M., Oliveto, R., Poshyvanyk, D.			Scopus
2	a paradigm for programming style research	Oman, P.W., Cook, C.R.			Scopus
3	an empirical investigation of the influence of a type of side effects on program comprehension	J. J. Dolado; M. Harman; M. C. Otero; L. Hu	ACM	IEEE	Scopus
4	an experimental investigation of the effect of program structure on program understanding	Love, T.	ACM		Scopus
5	an investigation of misunderstanding code patterns in c opensource software projects	Medeiros, F., Lima, G., Amaral, G., Apel, S., Kästner, C., Ribeiro, M., Gheyi, R.			Scopus
6	automatic segmentation of method code into meaningful blocks design and evaluation	Wang, X., Pollock, L., Vijay-Shanker, K.			Scopus
7	automatically assessing code understandability	S. Scalabrino; G. Bavota; C. Vendome; M. Linares-V?squez; D. Poshyvanyk; R. Oliveto		IEEE	
8	automatically assessing code understandability reanalyzed combined metrics matter	A. Trockman; K. Cates; M. Mozina; T. Nguyen; C. Kästner; B. Vasilescu	ACM	IEEE	Scopus
9	beacons in computer program comprehension	S. Wiedenbeck	ACM		

Continuação da Tabela					
#	**Title**	**Author**	**ACM**	**IEEE**	**Scopus**
10	complexity measures for rulebased programs	M. B. O'Neal and W. R. Edwards	ACM		
11	conditional statements, looping constructs, and program comprehension an experiments study	E. R. Iselin	ACM		
12	descriptive compound identifier names improve source code comprehension	Schankin, A., Berger, A., Holt, D.V., Hofmeister, J.C., Riedel, T., Beigl, M.	ACM		Scopus
13	detecting and comparing brain activity in short program comprehension using eeg	M. K. -. Yeh; D. Gopstein; Y. Yan; Y. Zhuang		IEEE	Scopus
14	does the discipline of preprocessor annotations matter a controlled experiment	Schulze, S., Liebig, J., Siegmund, J., Apel, S.	ACM		Scopus
15	effective identifier names for comprehension and memory	Lawrie, D., Morrell, C., Feild, H., Binkley, D.			Scopus
16	effects of variable names on comprehension an empirical study	E. Avidan; D. G. Feitelson	ACM	IEEE	Scopus
17	empirical studies on programming language stimuli	Stefik, A., Gellenbeck, E.	ACM		Scopus
18	enhancing program comprehension formatting and documenting	Arab, M.	ACM		Scopus
19	how programmers read regular code a controlled experiment using eye tracking	Jbara, A., Feitelson, D.G.	ACM		Scopus
20	identifier length and limited programmer memory	Binkley, D., Lawrie, D., Maex, S., Morrell, C.	ACM		Scopus
21	impacts of coding practices on readability	Dos Santos, R.M., Gerosa, M.A.	ACM		Scopus

Continuação da Tabela					
#	**Title**	**Author**	**ACM**	**IEEE**	**Scopus**
22	indentation simply a matter of style or support for program comprehension	J. Bauer; J. Siegmund; N. Peitek; J. C. Hofmeister; S. Apel	ACM	IEEE	
23	learning a metric for code readability	R. P. L. Buse; W. R. Weimer	ACM	IEEE	Scopus
24	linguistic antipatterns what they are and how developers perceive them	Arnaoudova, V., Di Penta, M., Antoniol, G.			Scopus
25	linking code readability, structure, and comprehension among novices its complicated	E. Wiese; A. Rafferty; A. Fox	ACM	IEEE	
26	meaningful identifier names the case of singleletter variables	G. Beniamini; S. Gingichashvili; A. K. Orbach; D. G. Feitelson	ACM	IEEE	Scopus
27	meaningfulness as a factor of program complexity	Chaudhary, B.D., Sahasrabuddhe, H.V.			Scopus
28	measuring computer program comprehension	Boysen, J.P., Keller, R.F.	ACM		Scopus
29	measuring neural efficiency of program comprehension	Janet Siegmund and Norman Peitek and Chris Parnin and Sven Apel and Johannes Hofmeister and Christian Kästner and Andrew Begel and Anja Bethmann and André Brechmann	ACM		
30	measuring the difficulty of code comprehension tasks using software metrics	Kasto, N., Whalley, J.	ACM		Scopus

Continuação da Tabela					
#	**Title**	**Author**	**ACM**	**IEEE**	**Scopus**
31	measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization	Fakhoury, S., Roy, D., Ma, Y., Arnaoudova, V., Adesope, O.			Scopus
32	on method ordering	Geffen, Y., Maoz, S.			Scopus
33	on some enduser programming constructs and their understandability	Maćkowiak, M., Nawrocki, J., Ochodek, M.			Scopus
34	on the effect of code regularity on comprehension	Ahmad Jbara and Dror G. Feitelson	ACM		
35	program comprehension investigating the effects of naming style and documentation	Scott Blinman and Andy Cockburn	ACM		
36	program indentation and comprehensibility	Miara, R.J., Musselman, J.A., Navarro, J.A., Shneiderman, B.	ACM		Scopus
37	program readability procedures versus comments	T. Tenny	ACM	IEEE	Scopus
38	replicating novices struggles with coding style	E. S. Wiese; A. N. Rafferty; D. M. Kopta; J. M. Anderson	ACM	IEEE	
39	shorter identifier names take longer to comprehend	Hofmeister, J., Siegmund, J., Holt, D.V.	ACM		Scopus
40	syntax, predicates, idioms what really affects code complexity	S. Ajami; Y. Woodbridge; D. G. Feitelson	ACM	IEEE	
41	the discipline of preprocessorbased annotations does #ifdef tag nt #endif matter	Malaquias, R., Ribeiro, M., Bonifacio, R., Monteiro, E., Medeiros, F., Garcia, A., Gheyi, R.	ACM		Scopus

Continuação da Tabela					
#	**Title**	**Author**	**ACM**	**IEEE**	**Scopus**
42	the effect of modularization and comments on program comprehension	Woodfield, S.N., Dunsmore, H.E., Shen, V.Y.	ACM		Scopus
43	the effect of scope delimiters on program comprehension	Sykes, F., Tillman, R.T., Shneiderman, B.			Scopus
44	the effectiveness of source code obfuscation an experimental assessment	M. Ceccato; M. Di Penta; J. Nagra; P. Falcarin; F. Ricca; M. Torchiano; P. Tonella		IEEE	Scopus
45	the effects of naming style and expertise on program comprehension	Teasley, B.E.	ACM		Scopus
46	the impact of identifier style on effort and comprehension	Binkley, D., Davis, M., Lawrie, D., Maletic, J.I., Morrell, C., Sharif, B.	ACM		Scopus
47	the initial stage of program comprehension	Susan Wiedenbeck	ACM		
48	typographic style is more than cosmetic	Oman, P.W., Cook, C.R.			Scopus
49	understanding misunderstandings in source code	Gopstein, D., Iannacone, J., Yan, Y., DeLong, L., Zhuang, Y., Yeh, M.K.-C., Cappos, J.	ACM		Scopus

Manual Papers					
#	**Title**	**Author**	**DOI**		
50	A Simpler Model of Software Readability	Posnett, Daryl and Hindle, Abram and Devanbu, Premkumar	DOI		
51	Recursion vs. iteration: An empirical study of comprehension	Alan C. Benander and Barbara A. Benander and Howard Pu	DOI		

Continuação da Tabela					
#	**Title**	**Author**	**DOI**		
52	Do static type systems improve the maintainability of software systems? An empirical study	S. Kleinschmager and R. Robbes and A. Stefik and S. Hanenberg and E. Tanter	DOI		
53	An Empirical Investigation into Programming Language Syntax	Stefik, Andreas and Siebert, Susanna	DOI		
54	Dealing with faults in source code: Abbreviated vs. full-word names	G. Scanniello and M. Risi	DOI		

APÊNDICE C – RESULTADO DA AVALIAÇÃO DE QUALIDADE

Paper Title	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Total**
a comprehensive model for code readability	1,0	1,0	1,0	1,0	1,0	0,5	1,0	1,0	1,0	**8,5**
A paradigm for programming style research	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	0,0	**8,0**
A Simpler Model of Software Readability	1,0	0,5	0,5	1,0	0,5	1,0	1,0	1,0	1,0	**7,5**
An Empirical Investigation into Programming Language Syntax	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
an empirical investigation of the influence of a type of side effects on program comprehension	1,0	0,5	1,0	1,0	1,0	1,0	1,0	1,0	0,5	**8,0**
an experimental investigation of the effect of program structure on program understanding	1,0	1,0	1,0	1,0	1,0	0,5	1,0	1,0	0,0	**7,5**
an investigation of misunderstanding code patterns in c opensource software projects	1,0	0,5	0,5	1,0	1,0	1,0	1,0	1,0	1,0	**8,0**
automatic segmentation of method code into meaningful blocks design and evaluation	1,0	1,0	1,0	1,0	1,0	1,0	0,0	1,0	1,0	**8,0**
automatically assessing code understandability	1,0	0,5	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**8,5**
automatically assessing code understandability reanalyzed combined metrics matter	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**

Continuação da Tabela										
Paper Title	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Total**
Beacons in computer program comprehension	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	0,5	**8,5**
complexity measures for rulebased programs	1,0	0,5	1,0	1,0	1,0	1,0	1,0	0,5	0,0	**7,0**
conditional statements, looping constructs, and program comprehension an experiments study	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	0,0	**8,0**
Dealing with faults in source code: Abbreviated vs. full-word names	1,0	1,0	1,0	1,0	1,0	0,5	1,0	1,0	1,0	**8,5**
descriptive compound identifier names improve source code comprehension	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
detecting and comparing brain activity in short program comprehension using eeg	1,0	0,5	0,5	1,0	1,0	1,0	1,0	1,0	0,0	**7,0**
Do static type systems improve the maintainability of software systems? An empirical study	1,0	1,0	1,0	0,5	0,5	1,0	1,0	1,0	1,0	**8,0**
does the discipline of preprocessor annotations matter a controlled experiment	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
effective identifier names for comprehension and memory	1,0	0,5	0,5	1,0	1,0	1,0	1,0	1,0	1,0	**8,0**
effects of variable names on comprehension an empirical study	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
empirical studies on programming language stimuli	1,0	0,5	1,0	1,0	1,0	1,0	1,0	1,0	0,5	**8,0**

Continuação da Tabela										
Paper Title	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Total**
enhancing program comprehension formatting and documenting	1,0	1,0	1,0	0,5	1,0	0,0	0,0	0,5	0,0	**5,0**
how programmers read regular code a controlled experiment using eye tracking	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
identifier length and limited programmer memory	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
impacts of coding practices on readability	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
indentation simply a matter of style or support for program comprehension	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
learning a metric for code readability	1,0	0,0	0,0	1,0	1,0	1,0	1,0	1,0	1,0	**7,0**
linguistic antipatterns what they are and how developers perceive them	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
linking code readability, structure, and comprehension among novices its complicated	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
meaningful identifier names the case of singleletter variables	1,0	1,0	1,0	0,5	1,0	0,5	1,0	1,0	1,0	**8,0**
meaningfulness as a factor of program complexity	1,0	0,5	0,5	1,0	1,0	0,5	0,5	1,0	0,0	**6,0**
measuring computer program comprehension	0,5	0,5	0,5	1,0	1,0	1,0	0,5	1,0	0,0	**6,0**
measuring neural efficiency of program comprehension	1,0	0,5	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**8,5**
measuring the difficulty of code comprehension tasks using software metrics	0,5	1,0	1,0	1,0	1,0	1,0	1,0	1,0	0,0	**7,5**

Continuação da Tabela										
Paper Title	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Total**
measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
on method ordering	1,0	1,0	0,5	1,0	1,0	1,0	0,0	1,0	1,0	**7,5**
on some enduser programming constructs and their understandability	1,0	0,5	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**8,5**
on the effect of code regularity on comprehension	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
program comprehension investigating the effects of naming style and documentation	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	0,0	**8,0**
program indentation and comprehensibility	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	0,0	**8,0**
program readability procedures versus comments	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	0,0	**8,0**
Recursion vs. iteration: An empirical study of comprehension	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	0,5	**8,5**
replicating novices struggles with coding style	0,5	0,0	0,5	1,0	1,0	1,0	0,5	1,0	0,5	**6,0**
shorter identifier names take longer to comprehend	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
syntax, predicates, idioms what really affects code complexity	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**
the discipline of preprocessorbased annotations does #ifdef tag nt #endif matter	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**

Continuação da Tabela										
Paper Title	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Total**
the effect of modularization and comments on program comprehension	1,0	1,0	0,5	1,0	1,0	0,0	1,0	1,0	0,5	**7,0**
the effect of scope delimiters on program comprehension	1,0	0,5	0,5	1,0	1,0	1,0	1,0	1,0	0,0	**7,0**
the effectiveness of source code obfuscation an experimental assessment	1,0	0,5	0,5	1,0	1,0	1,0	1,0	1,0	1,0	**8,0**
the effects of naming style and expertise on program comprehension	1,0	0,5	0,5	0,5	1,0	1,0	1,0	1,0	0,0	**6,5**
the impact of identifier style on effort and comprehension	1,0	0,5	0,5	1,0	1,0	1,0	1,0	1,0	1,0	**8,0**
the initial stage of program comprehension	1,0	0,5	0,5	1,0	1,0	1,0	1,0	1,0	0,5	**7,5**
typographic style is more than cosmetic	1,0	0,5	1,0	1,0	0,5	1,0	1,0	1,0	0,0	**7,0**
understanding misunderstandings in source code	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	**9,0**