Mimir - Sistema de árbitro online para estudos de compreensão de código

Eidson Jacques Almeida de Sá

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brasil

ejas@cin.ufpe.br

Abstract. *Code comprehension is the activity in which software engineers try to understand the behavior of a software system using its source code as the primary reference. [Hermans 2021] estimates that half of a program's development time is dedicated to reading, so it's important that this activity be done most efficiently. Students and professionals should receive training in efficient code reading, but there are no efficient systems to do that. Developers commonly use online judge systems to improve their programming abilities, but the existent market tools only support code writing, not code understanding. There is no efficient system that allows test creation where the goal is not to write an algorithm, but to evaluate the user comprehension of one or more snippets of code. Studies about code comprehension either need an implementation of a system to conduct the test, increasing its scope and complicating its implementation, or uses an improvised tool, that impairs the experiment's progress because of the lack of specific features. This article shows the creation of Mimir, an online judge system for the conduction of code comprehension tests, implemented from requirements based on a set of studies. To verify its behavior, two case studies were conducted based on previous experiments, [Gopstein et al. 2017] and [Langhout and Aniche 2021], with students of the Informatic Center at the Federal University of Pernambuco, which showed the system's efficacy to support studies, either to academic or educational goals.*

Resumo. *Compreensão de código é a atividade na qual engenheiros de software buscam o entendimento do comportamento de um sistema de software utilizando seu codígo fonte como referência primária. [Hermans 2021] estima que metade do tempo de desenvolvimento de um programa é dedicado a ler, portanto, é importante que essa atividade seja feita da maneira mais eficiente possível. Estudantes e profissionais da área devem receber treinamento na leitura eficiente de código, mas não há plataformas eficientes para isso. Desenvolvedores frequentemente utilizam sistemas de árbitro online para aprimoramento de habilidades de programação, mas as ferramentas existentes no mercado só dão apoio à construção de programas e não à leitura de código. Não existe uma plataforma eficiente que permita a criação de testes onde o objetivo não é escrever um algoritmo, e sim avaliar o entendimento do usuário sobre um ou mais trechos de código. Estudos envolvendo compreensão de código ou precisam da implementação de um sistema para conduzir o teste, aumentando seu escopo e dificultando sua implementação, ou utilizam um sistema improvisado, que prejudica o andamento do experimento devido a falta de funcionalidades*

específicas. Este artigo mostra a criação do Mimir, um sistema de árbitro online para condução de testes de compreensão de código, implementado a partir de requisitos baseados em um conjunto de estudos da área. A fim de verificar seu funcionamento, foram conduzidos dois estudos de caso baseados em experimentos anteriores, [Gopstein et al. 2017] e [Langhout and Aniche 2021], com estudantes do Centro de Informática da Universidade Federal de Pernambuco, que mostraram a eficácia da plataforma em amparar estudos do tipo, seja para fins acadêmicos ou educacionais.

1. Introdução

Sistemas de computação podem ser desenvolvidos de formas completamente diferentes e produzir o mesmo resultado. Trechos de código de tamanhos distintos e utilizando diversos tipos de ferramentas são capazes de gerar softwares com as mesmas funcionalidades, porém, não necessariamente são igualmente legíveis para os seus desenvolvedores. Ler um código não é apenas a maneira principal de melhorar habilidades de programação, mas também é uma maneira de coletar informações essenciais para completar um objetivo técnico na engenharia de software [Bourque and Fairley 2014].

Compreensão de código é definido como a atividade onde engenheiros de software buscam o entendimento do comportamento de um sistema de software utilizando seu codígo fonte como referência primária [Marvin Zelkowitz 2002] e pode se dar de diversas maneiras. A mais tradicional se trata da leitura de trechos de código, mas outras ferramentas também podem ser utilizadas, como diagramas, vídeos e documentações.

É utilizada frequentemente no contexto de *code reviews*, onde desenvolvedores de um time analisam o código de outros desenvolvedores a fim de descobrir problemas de performance e aumentar a qualidade do código. O repositório do *Visual Studio Code* [https://github.com/microsoft/vscode], um dos maiores editores de código do mercado, por exemplo, tem seu código aberto e possui mais de 15000 *pull requests*, onde usuários precisam ter seus códigos revisados antes de aprovados.

Tal atividade tem sido alvo de diversos estudos na área de computação. A busca pelo termo *code comprehension* retorna 3.630 resultados no *Google Scholar* e a *International Conference on Program Comprehension* [Serebrenik and Sharma 2021], o principal fórum científico da área de compreensão de programas, teve sua 29ª edição no ano de 2021. Grande parte dos estudos da área envolvem *surveys* ou formulários de diversas formas para analisar a legibilidade de um código. Aproximadamente 74% deles pedem que o usuário provenha alguma informação sobre o código: explicar o funcionamento do código; responder perguntas sobre características do código; lembrar parte do código [Oliveira et al. 2020].

Além disso, formulários do tipo também são utilizados para fins acadêmicos, para estudos ou avaliações de cursos que demandam a compreensão de código dos alunos. Todavia, esses estudos são feitos em ferramentas improvisadas, como o *Google Forms*, que possuem limitações específicas para esses casos de uso, como a impossibilidade de verificar se o usuário saiu da tela com o teste aberto, o não armazenamento do tempo de duração do teste e falta de definição de questões similares que deveriam ter a maior distância possível, mesmo com uma ordenação aleatoria.

A fim de ter uma melhor opção de sistema de formulários para testes de compreensão de código, foi implementado uma plataforma com funcionalidades específicas para experimentos do tipo. Este trabalho discute os fundamentos e motivações que serviram de base para a criação do Mimir, um sistema de árbitro online criado para condução de testes de compreensão de código. São mostrados os requisitos iniciais e o funcionamento do sistema. Além disso, é exibida a condução de dois estudoe de caso construído a partir de experimentos anteriores, demonstrando a usabilidade do sistema, que mesmo tendo um viés mais tecnológico, serve como amparo para estudos de pesquisa. Também serão exibidos futuros trabalhos que podem dar continuidade a este projeto e um apanhado geral do seu desenvolvimento.

2. Fundamentos

Um sistema de árbitro online é uma plataforma na internet completamente automatizada que verifica um código provido de um usuário em tempo real [Wasik et al. 2018]. As questões apresentam uma descrição, um modelo de entrada e um modelo de saída, e o usuário deve prover um algoritmo que dada qualquer entrada nesse estilo, retorne a saída esperada.

Possuem geralmente um Oráculo, capaz de mostrar a saída esperada dada determinada entrada, e exemplos de entrada/ saída para auxiliarem o usuário. Sistemas do tipo são utilizados para uma avaliação confiável do código fonte de um algoritmo, que são compilados e testados em um ambiente homogêneo. Seu objetivo geral é uma avaliação segura, confiável, contínua e baseada em nuvem de algoritmos submetidos por usuários [Wasik et al. 2018]. Todavia, esses sistemas tem seu escopo fechado e não permitem sua utilização para outros tipos de avaliações que não se tratem da construção de um novo trecho de código.

Existem três tipos de dificuldades que ocorrem ao ler um código [Hermans 2021]: Falta de conhecimento, falta de informação e falta de poder de processamento. Essas dificuldades acontecem devido as diferenças de conceitos e funcionalidades utilizadas, e assim, trechos de códigos que fornecem uma mesma saída podem ter diferenter percepções de diferentes usuários. Para testes que envolvem compreensão de código, as questões precisam do trecho do código no enunciado para colher determinado tipo de dado, como os estudos a seguir:

Em *Understanding Misunderstandings in Source Code* [Gopstein et al. 2017], foi proposto um estudo para identificar pequenos trechos (átomos) de código que causam confusão no usuário que está lendo. Para isso, foi conduzido um teste onde usuários liam trechos de código com mesmas saídas (espaçados entre outros trechos de código), mas de forma obfuscada e transformada, permitindo analisar se determinado átomo de confusão dificultaria a leitura do usuário.

[Ajami et al. 2017] conduziram um experimento com profissionais da área para verificar quais estruturas de código são consideradas mais difíceis, ou seja, demoram mais para ser entendidas e/ou produzem mais erros. Em *Recursion vs. iteration: An empirical study of comprehension* [Benander et al. 1996], foi feito um estudo com estudantes de ciência da computação para analisar se códigos com mesma finalidade, mas utilizando recursão ou iteração, teriam percepções diferentes sobre sua saída.

Em *The impact of identifier style on effort and comprehension*

[Binkley et al. 2013], foi conduzido um estudo para analisar o impacto de estilos diferentes de identificadores na velocidade e na acurácia da compreensão de código. [Miara et al. 1983] promoveu um teste para verificar se identações diferentes prejudicariam a compreensão de usuários sobre um mesmo trecho de código.

Todos esses estudos tiveram como objetivo analisar a compreensão de usuários sobre trechos de código sob diferentes perspectivas, e todos eles possuíam trechos de código como parte do enunciado e perguntas objetivas sobre seu funcionamento para coleta das respostas. Esse formato é padrão em quase todos os experimentos de compreensão de código e é utilizado de diversas formas.

3. Motivações

Para condução desses estudos, foram utilizadas diversas ferramentas de formulários, sejam elas existentes no mercado, criadas exclusivamente para cada estudo, ou até mesmo ferramentas para exibição de trechos de código que são utilizadas em entrevistas presenciais.

A falta de uma ferramenta que sirva para condução desses estudos aumenta a complexidade dos mesmos, já que, além da condução do estudo, também entra no escopo a criação de uma ferramenta específica para coleta de respostas do usuário com a junção de diversas funcionalidades.

Existem ferramentas de formulários no mercado, como o *Google Forms* e o *SurveyMonkey*, mas são plataformas desenvolvidas para perguntas e exercícios básicos e acabam por não possuírem todas as ferramentas necessárias para um estudo mais específico sobre compreensão de código.

Nessas plataformas, não há personalização de formatação e de linguagem, sendo utilizadas, geralmente, capturas de tela com trechos de código, o que dificulta a criação e a manutenção dessas questões. Além disso, dados importantes como duração por questão e verificação se o usuário saiu do formulário não são possíveis em ambas as plataformas.

Ferramentas mais complexas e específicas que são bastante utilizadas também seriam necessárias, como a definição de questões similares (como as obfuscadas e as transformadas em [Gopstein et al. 2017]) que necessitam de certa distância durante o formulário para não impactar nas respostas dos usuários.

São plataformas que são utilizadas de forma improvisada, e a falta de uma ferramenta que cumpra com os requisitos específicos para a condução de estudos do tipo acrescenta uma complexidade a mais para estes projetos. O Mimir surge como uma proposta de plataforma que auxilie esses estudos, visando fortificar a área de compreensão de código.

4. Requisitos

Para elicitação dos requisitos do projeto, foi conduzida uma reunião com o *stakeholder* Fernando Castor, professor que já conduziu diversos experimentos do tipo, de aproximadamente 1 hora no dia 25 de Julho de 2022. As perguntas já haviam sido formuladas após reuniões anteriores, e com essa reunião as dúvidas restantes foram tiradas para que o projeto pudesse ser implementado.

O formato de acesso definido foi **Autenticação com o Google**, e que apenas os condutores de testes / criadores de questões precisariam ser autenticados. Usuários que respondem os testes deveriam ser capazes de fazer **Submissões sem autenticação**

A **Criação de questões** deveria ser feita pelos usuários de maneira personalizada com título, um ou mais trechos de código, pergunta, resposta esperada e se será utilizado algum tipo de destaque nos trechos de código, de acordo com a linguagem de programação utilizada. Além disso, essas questões podem ser **Privadas** ou não, o que define se os outros usuários do sistema serão capazes de ver determinada questão.

Além disso, a **Criação de testes** deveria ser feita de maneira personalizada com título, página de instruções, questões demográficas opcionais e a escolha das questões a serem utilizadas. A **Ordem das questões** deve ser randomizada quando forem ser respondidas, mas o usuário, ao criar o teste, pode definir questões similares que deverão ter a maior distância possível em qualquer ordem sorteada.

Na **Tela Inicial**, as questões e os testes devem aparecerão para os usuários, e deverão ter telas únicas onde os usuários poderão ver os seus dados e serem capazes de editá-los. Na **Visualização de Teste**, o usuário poderá coletar o link que será utilizado para respondê-lo, e **Exportar as Respostas** obtidas até então em um arquivo .csv.

Para **Submeter Respostas**, os usuários deverão acessar um link obtido pelo condutor do teste e assim terão acesso ao formulário. A tela inicial mostrará a página de instruções, seguida das questões demográficas em uma tela e as questões definidas pelo condutor, uma por vez. Após todas as questões serem respondidas, o usuário que está respondendo deve ver uma tela final e a submissão será enviada para o sistema.

5. Sistema

A tela de autenticação do sistema é uma tela branca com a logo do Mimir e um botão simples para login com o Google, como pode ser visto na Figura 1.

Figura 1. Tela de autenticação

Ao efetuar o login, o usuário encontra uma tela com 2 tabelas: Questões e testes, como mostrado na Figura 2. Essas tabelas estão paginadas em ordem alfabética e, ao clicar nas setas de alguma das opções, abrirá a tela da questão ou teste. Na tabela de questões só aparecem as questões criadas pelo usuário e as questões que não são privadas.

Na Figura 3 está a tela com o formulário que aparecerá ao clicar no botão de criar questão. O usuário deve digitar título, um ou mais trechos de código, pergunta, resposta esperada e se o trecho deverá estar destacado de acordo com a linguagem. As linguagens suportadas atualmente são C, Java e Python, e também há a opção de não destacar o trecho.

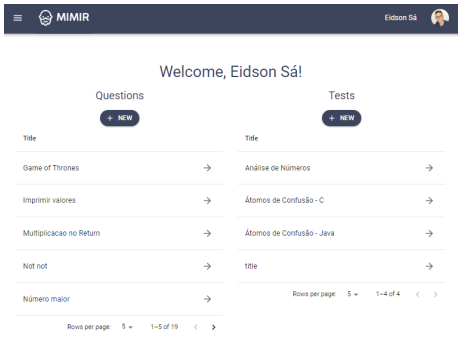

Figura 2. Tela inicial

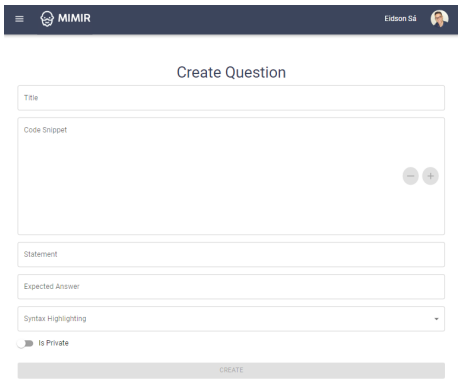

Figura 3. Tela de Criação de Questões

A tela com os dados de uma questão é aberta após ela ser criada ou ao clicar na sua respectiva seta em alguma das tabelas de questões. Essa tela possui os dados da questão e um botão para editá-la. Um exemplo de uma questão chamada *Game of Thrones* com 2 trechos de código aparece na Figura 4.

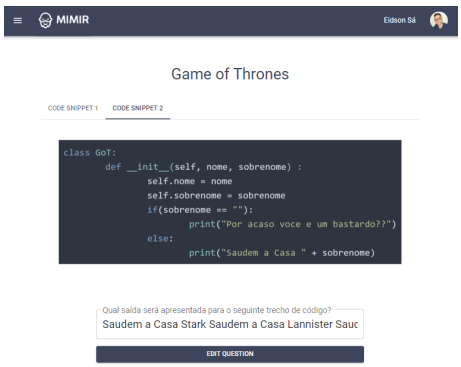

Figura 4. Tela específica de uma Questão

O formulário de criar testes, presente na Figura 5, precisa de um título e uma página de instruções, que já possui um texto padrão. O usuário pode colocar várias questões demográficas, que aparecem antes do teste para o usuário, e selecionar as

questões do sistema que ele deseja utilizar. Também pode selecionar quais questões funcionam como pares e devem ser espaçadas da melhor maneira possível.

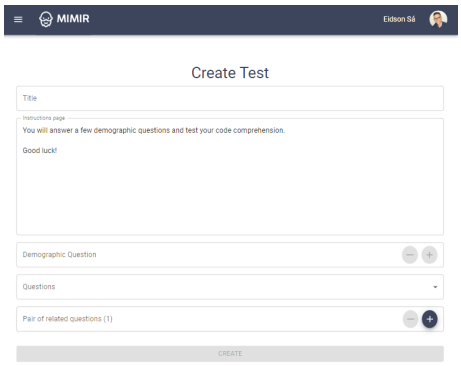

Figura 5. Tela de Criação de Testes

Na figura 6 está a tela com os dados de teste possui uma tabela com as questões escolhidas, mostra a página de instruções e as questões demográficas. Também mostra um botão que copia o link que deve ser utilizado para responder o teste, um botão para editar o teste, e se já houver alguma resposta para o teste em questão, um botão que exporta as submissões em .csv.

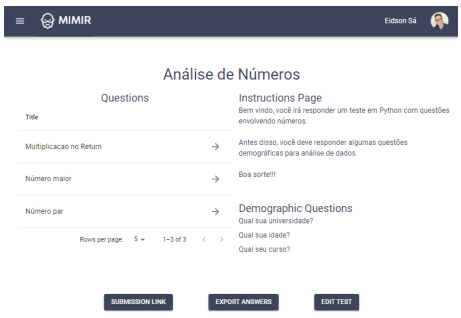

Figura 6. Tela específica de um Teste

Para responder algum teste, a tela inicial é a tela com as intruções definidas pelo criador do teste (Figura 7). A segunda tela são as questões demográficas (Figura 8) e as seguintes são as questões em ordem aleatória e seguindo o algoritmo de separação entre questões similares, como exemplificado na Figura 9.

6. Estudos de caso

Para validação da plataforma, foram conduzidos 2 testes no Mimir, a fim de verificar se o sistema seria capaz de ser utilizado em experimentos que envolvem testes de compreensão de código, como os mostrados na seção de Fundamentos.

O primeiro foi um experimento baseado em [Gopstein et al. 2017] e está na Figura 10. Foram utilizadas 6 questões retiradas do experimento original, utilizando as versões obfuscadas e transformadas dos átomos *Implicite Predicate*, *Infix Operator Precedence* e *Post-Increment*. Essas questões foram marcadas como questões similares e apareciam sempre com o maior espacamento possível.

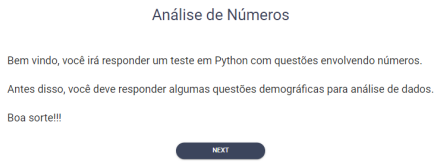

Figura 7. Tela de página de instruções

Figura 8. Tela de questões demográficas

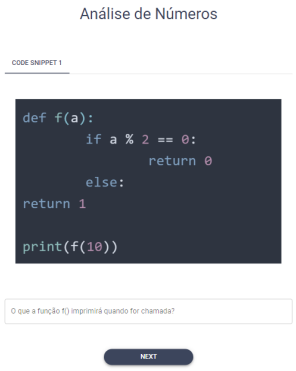

Figura 9. Tela com uma questão

O teste foi enviado para estudantes de Engenharia da Computação da Universidade Federal de Pernambuco que responderam e enviaram suas submissões. A plataforma funcionou corretamente em todas as respostas, e os dados puderam ser exportados para um .csv como planejado.

Na Figura 11 está o segundo teste, que foi feito baseado em [Langhout and Aniche 2021], tendo o objetivo de achar átomos de confusão em Java. Também foram utilizadas 6 questões retiradas do experimento original, utilizando as versões obfuscadas e transformadas dos átomos *Change of Literal Encoding*, *Type Conversion* e *Repurposed Variables*. Essas questões foram marcadas como questões similares e apareciam sempre com o maior espacamento possível.

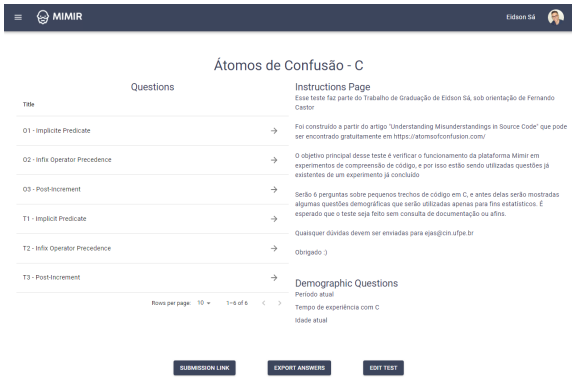

Figura 10. Tela do teste de Átomos de Confusão em C

O teste foi enviado para estudantes de Ciência da Computação da Universidade Federal de Pernambuco que responderam e enviaram suas submissões. A plataforma também funcionou corretamente em todos os casos, e os dados das submissões puderam ser exportados para um .csv.

Figura 11. Tela do teste de Átomos de Confusão em Java

Ao final dos experimentos, feedbacks foram coletados com os usuários respondentes por meio de entrevistas. O único feedback coletado acerca da plataforma foi sobre as questões demograficas, segundo as quais usuários disseram que questões do tipo funcionam melhor com alternativas prontas, para definir tempo de experiência, período e idade de maneira menos abstrata.

7. Conclusão

Na mitologia nórdica, Mimir é conhecido por ser deus mais sábio de Asgard, devido ao seu poço, *Mímisburnnr*, onde bebe de sua água todos os dias para obter mais sabedoria. Assim surge a plataforma Mimir, construída para ajudar usuários a conduzirem estudos de compreensão de código e auxiliar a academia a obter melhores resultados no desenvolvimento de software.

O Mimir foi feito de maneira escalável e facilmente expansível, levando em conta que trabalhos diferentes na área precisam de funcionalidades diferentes, e a ferramenta precisa ser facilmente adaptável e capaz de receber melhorias. Para isso, a ferramenta está

disponível em código aberto [https://github.com/eidsonsa/mimir/] Há diversos trabalhos futuros mapeados para aprimorar a plataforma:

Autenticação para respondentes Usuários que não pretendem criar questões ou testes, mas que os respondem com frequência e desejam guardar um histórico de suas submissões.

Autenticação com outras plataformas Utilização de outros sistemas de autenticação além do Google, como o Facebook, Github e autenticação própria com email e senha.

Tipos diferentes de questões Implementação de mais tipos de questões, como questões de múltipla escolha ou que demandam a escrita de algum trecho de código

Mais linguagens de programação Atualmente, o Mimir só suporta destaque de códigos em C, Java e Python, o que pode ser aumentado no futuro

Exportação em outros tipos O Mimir exporta as submissões em csv, mas pode vir a exportar também em outros tipos, como pdf e parqueet.

Adaptação de Javascript para Typescript O projeto tem seu código feito em Javascript, e pode ser alterado para Typescript afim de melhorar sua legibilidade

Versão Mobile O sistema pode ser utilizado em dispositivos móveis, mas precisa de uma versão que funcione de maneira melhor nesses sistemas

Melhora de UI Com a maior utilização do sistema, feedbacks devem ser colhidos para atualizações no design da plataforma, de forma que a aplicação fique mais prática e intuitiva para os usuários

O objetivo do desenvolvimento do Mimir de criar uma plataforma que permitisse a realização de testes envolvendo compreensão de código, seja para auxiliar atividades educacionais ou para fortificar a área acadêmica, foi alcançado e funcionou como esperado. A utilização do sistema foi feita de maneira simples e usuários conseguem criar questões e testes de maneira bastante intuitiva. Os dados conseguiram ser exportados para um arquivo .csv em um formato capaz de fazer análises, como mostrado na Tabela 1.

Período	Experiência	Idade	O2	O1	T2	T1	elapsedTime	exitScreen
5º	1,5 anos	21	true	false	true	false	610.098	true
4	1 ano e meio	20	false	true	true	false	135.556	false
5°	2 anos	20	true	false	false	false	270.203	true

Tabela 1. Exemplo de arquivo .csv gerado pelo Mimir

Os experimentos conduzidos na seção anterior foram completados de maneira eficiente e sem dificuldades. A plataforma se mostrou pronta para auxiliar estudos do tipo e permite ao usuário condutor personalizá-lo e coletar dados de maneira fácil, apenas compartilhando o link com as pessoas que irão respondê-la. Sem a necessidade de criação de plataformas específicas ou a utilização de sistemas improvisados,espera-se que os estudos posteriores na área de compreensão de código sejam feitos de maneira mais simples e rápida, reduzindo uma parte essencial do seu escopo.

A plataforma também foi feita de maneira que pode ser utilizada para fins educacionais. Com o salvamento do tempo decorrido e sabendo se o respondente saiu da tela, professores podem utilizar o Mimir para conduzir provas, servindo como um facilitador para exercícios de cursos que demandam a leitura de determinado trecho de código.

A ferramenta cumpre os requisitos estabelecidos e de maneira escalável, tendo como objetivo servir como um auxílio para a comunidade, que ainda pode melhorar o projeto e adicionar novas funcionalidades para ele, devido a sua distribuição em código aberto. É esperado que o Mimir fique cada vez mais completo e ajude a melhorar a comunidade de computação cada vez mais.

Referências

Ajami, S., Woodbridge, Y., and Feitelson, D. G. (2017). Syntax, predicates, idioms - what really affects code complexity? In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 66–76.

Benander, A. C., Benander, B. A., and Pu, H. (1996). Recursion vs. iteration: An empirical study of comprehension. *Journal of Systems and Software*, 32(1):73–82.

Binkley, D., Davis, M., Lawrie, D., Maletic, J. I., Morrell, C., and Sharif, B. (2013). The impact of identifier style on effort and comprehension. *Empirical Softw. Engg.*, 18(2):219–276.

Bourque, P. and Fairley, R. E., editors (2014). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition.

Gopstein, D., Iannacone, J., Yan, Y., DeLong, L., Zhuang, Y., Yeh, M. K.-C., and Cappos, J. (2017). Understanding misunderstandings in source code. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, page 129–139, New York, NY, USA. Association for Computing Machinery.

Hermans, F. (2021). *The programmer's brain: What every programmer needs to know about cognition*. Manning publications Co.

Langhout, C. and Aniche, M. F. (2021). Atoms of confusion in java. *CoRR*, abs/2103.05424.

Marvin Zelkowitz, P. (2002). *Advances in Computers*. Number v. 56 in Advances in Computers. Elsevier Science.

Miara, R. J., Musselman, J. A., Navarro, J. A., and Shneiderman, B. (1983). Program indentation and comprehensibility. *Commun. ACM*, 26(11):861–867.

Oliveira, D., Bruno, R., Madeiral, F., and Castor, F. (2020). Evaluating code readability and legibility: An examination of human-centric studies. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 348–359.

Serebrenik, A. and Sharma, A., editors (2021). *29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021*. IEEE.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2018). A survey on online judge systems and their applications. *ACM Comput. Surv.*, 51(1).