RESSALVA

Atendendo solicitação do(a) autor(a), o texto completo desta dissertação será disponibilizado somente a partir de 25/01/2024.

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências e Tecnologia - Campus de Presidente Prudente

LOBUS-OWL: Modelo de Localização de Defeitos em Código-Fonte Apoiado por Ontologia

ALISSON SOLITTO DA SILVA

FCT-Unesp – Presidente Prudente
Julho/2022

LOBUS-OWL: Modelo de Localização de Defeitos em Código-Fonte Apoiado por Ontologia

ALISSON SOLITTO DA SILVA

Orientador: *Dr. Rogério Eduardo Garcia*

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, da Universidade Estadual Paulista (UNESP) - Campus de Presidente Prudente, como requisito para obtenção do título de Mestre em Ciência da Computação.
Área de Concentração: Computação Aplicada
Linha de Pesquisa: Sistemas de Informação

FCT-Unesp – Presidente Prudente
Julho/2022

S586l

Silva, Alisson Solitto da
LOBUS-OWL : Modelo de Localização de Defeitos em Código-Fonte Apoiado por Ontologia / Alisson Solitto da Silva. -- Presidente Prudente, 2022
99 p. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências e Tecnologia, Presidente Prudente
Orientador: Rogério Eduardo Garcia

1. Localização de Defeitos. 2. Ontologia. 3. Manutenção Corretiva. 4. Engenharia de Software. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências e Tecnologia, Presidente Prudente. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Presidente Prudente

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: LOBUS-OWL: Modelo de Localização de Defeitos em Código-Fonte Apoiado por Ontologia

AUTOR: ALISSON SOLITTO DA SILVA
ORIENTADOR: ROGÉRIO EDUARDO GARCIA

Aprovado como parte das exigências para obtenção do Título de Mestre em Ciência da Computação, área: Computação Aplicada pela Comissão Examinadora:

Prof. Dr. ROGÉRIO EDUARDO GARCIA (Participaçao Virtual)
Departamento de Matemática e Computação / UNESP/Câmpus de Presidente Prudente

Prof. Dr. LEONARDO CASTRO BOTEGA (Participaçao Virtual)
Programa de Pós-Graduação em Ciência da Informação / Unesp, Faculdade de Filosofia e Ciências, Marília

Prof. Dr. MARCELO MEDEIROS ELER (Participaçao Virtual)
Escola de Artes, Ciências e Humanidades / Universidade de São Paulo

Presidente Prudente, 25 de julho de 2022

Este trabalho é dedicado a Deus, a minha família e a minha esposa. A conclusão deste trabalho e de mais uma etapa acadêmica se deu graças a todo o esforço e dedicação dos meus pais com a minha educação desde cedo.

Agradecimentos

Agradeço primeiramente a **Deus**, por sempre ter iluminado o meu caminho e por sempre me conceder saúde para viver e seguir com os meus objetivos.

A minha família e a minha esposa, por todo apoio, carinho e compreensão.

Ao orientador e coorientador do presente trabalho, que confiaram em mim e me incentivaram para o desenvolvimento deste.

A todos os professores do curso pela dedicação e comprometimento ao transmitir conhecimento durante as disciplinas.

A todos os amigos e professores do curso de Ciência da Computação (2014-2018) e aos meus amigos do Departamento de Sistemas de Informação (2017-2020) do UNIVEM que me apoiaram a seguir nessa jornada.

Agradeço a todos os professores ao longo de toda a minha vida, que direta ou indiretamente contribuíram para minha formação pessoal e profissional. Os professores são essenciais para a evolução da sociedade!

Porque as pessoas que são loucas o suficiente para achar que podem mudar o mundo, são as que o fazem.
Steve Jobs

Resumo

Os relatórios de defeitos ajudam a triagem de inconsistências encontradas durante todo o ciclo de vida do software, por possuírem informações valiosas para a Engenharia de Software. O processo de localização de defeitos busca identificar artefatos de código-fonte possivelmente relacionados ao defeito reportado. Essa tarefa executada de maneira manual é onerosa para os programadores responsáveis pela correção, que geralmente precisam realizar a análise no código-fonte para identificar o artefato defeituoso e, depois, realizar as manutenções necessárias. O processo de localização de defeitos auxilia os programadores delimitando o espaço de busca e identificando arquivos relevantes relacionados ao defeito. Geralmente, as metodologias adotadas para o processo de localização de defeitos utilizam técnicas de aprendizado de máquina com similaridade textual, algoritmos de classificação e agrupamento de arquivos de código-fonte de acordo com os dados extraídos dos relatórios de defeitos. Nessas abordagens de localização de defeitos, os arquivos de código-fonte são considerados como unidades, podendo gerar ruídos quando o arquivo de código-fonte é grande. Neste trabalho é apresentado um modelo de localização de defeitos utilizando o conhecimento semântico do código-fonte com o apoio de ontologias. O desempenho do modelo proposto foi avaliado em seis projetos de software relevantes de código aberto (AutoMapper, MsBuild, EfCore, AspNetCore, MQTTnet e NLog) utilizando as métricas de avaliação *Top N Rank of Files* (TNRF), *Mean Reciprocal Rank* (MRR) e *Mean Average Precision* (MAP). Os resultados mostram que o modelo proposto alcança resultados significativos.

Palavras-chave: Localização de Defeitos, Ontologia, Manutenção Corretiva, Engenharia de Software.

Abstract

Bug reports help triage inconsistencies found throughout the software lifecycle, as these reports contain valuable information for Software Engineering. The bug location process seeks to identify source code artifacts possibly related to the reported bug. This manually performed task is costly for the programmers responsible for the correction, who usually need to perform analysis on the source code to identify the defective artifact and then carry out the necessary maintenance. The bug location process helps programmers by delimiting the search space and identifying relevant files related to the bug report. Generally, the methodologies adopted for the process of localization bugs use machine learning techniques with textual similarity, classification algorithms and grouping of source code files according to the data extracted from bug reports. In these bug location approaches, source code files are considered as single units, and may generate noise when the source code file is large. In this work, a bug location model is presented using semantic domain knowledge of the source code with the support of ontologies. We evaluated the performance of the proposed model in six relevant open source software projects (AutoMapper, MsBuild, EfCore, AspNetCore, MQTTnet and NLog) and carried out the experiments using the evaluation metrics Top N Rank of Files (TNRF), Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP). The results show that the proposed model achieves significant results.

Keywords: Bug Location, Ontology, Corrective Maintenance, Software Engineering.

Lista de Figuras

Lista de Tabelas

Sumário

CAPÍTULO 1

Introdução

1.1 Contextualização, Justificativa e Motivação

Independente da complexidade e domínio de aplicação, o software continua a evoluir ao longo do tempo. Portanto, melhorias contínuas, manutenções corretivas (em consequência de defeitos encontrados nos artefatos) e a necessidade de refatoração do código-fonte fazem parte do ciclo de vida do software.

Os defeitos podem estar presentes desde o início da concepção do software. Por exemplo, na especificação dos requisitos do sistema, na modelagem de diagramas e casos de uso, como também nos estágios finais do processo ou na implantação do sistema (Delamaro et al., 2013).

O processo de manutenção corretiva tem como objetivo localizar e sanar defeitos. Esse processo exige um esforço manual, em que o responsável deve interpretar o relato do defeito e identificar quais entidades do código-fonte devem ser alteradas para reparar o problema (Gujral et al., 2015).

Nas etapas do processo de manutenção corretiva, o processo de localização de defeitos é iniciado após um relato de defeito. A equipe responsável pelo processo de manutenção corretiva inicia um trabalho manual de interpretação do defeito reportado e busca identificar trechos do código-fonte que podem ter originado o problema. Após a identificação do artefato de código-fonte que possui o defeito, um programador é responsável por realizar as manutenções necessárias (Jeong et al., 2009).

Para apoiar as atividades de manutenção corretiva ferramentas denominadas *Bug Tracking System* (BTS) possibilitam reportar as inconsistências encontradas durante todo o ciclo de vida do software (Zhang e Lee, 2011). Com a utilização de BTS é possível armazenar e gerenciar os relatórios de defeitos (*Bugs Report*). O relatório de defeito descreve as situações em que o software

não se comporta como o esperado, contribuindo para a triagem de defeitos (Zibran, 2016).

A representação semântica do código-fonte com o uso de ontologias pode contribuir para a evolução do processo de localização de defeitos. A ontologia fornece uma camada de integração e interoperabilidade, unificando a representação de diferentes domínios em um modelo semântico (Happel e Seedorf, 2006).

A literatura registra poucos modelos que utilizam ontologias em processos relacionados à Engenharia de Software. Além disto, alguns dos modelos não descrevem uma ontologia de domínio rica. Geralmente, são estruturas que representam apenas uma taxionomia, não possuindo conceitos, nem relações, ou sequer a possibilidade de interferência de novos conhecimentos como uma ontologia.

Os trabalhos de Kiefer et al. (2007) e Tran e Le (2014) fazem uso de ontologias para a representação de artefatos do software de modo superficial, ou seja, as ontologias utilizadas representam apenas uma taxionomia de alguns elementos do domínio, sem regras e relações expressivas.

A utilização de ontologia no processo de localização de defeitos contribui para a descoberta de arquivos de código-fonte relacionados com o relato de defeito que não estão explicitamente relacionados. A ontologia permite a formalização da estrutura semântica do código-fonte, possibilitando a compreensão de seus conceitos e relações, além da inferência de novos conhecimentos.

Este trabalho propõe um modelo para a localização de defeitos em código-fonte apoiado por ontologia. O uso de ontologia possibilita a representação semântica do domínio do paradigma de programação orientado a objetos. O modelo contempla a geração de conhecimento semântico do código-fonte, até a identificação e classificação dos artefatos candidatos de código-fonte que estão relacionados com o relatório de defeito. Desta forma, é possível realizar a identificação de artefatos do código-fonte a partir de um domínio conhecido, não havendo a necessidade de utilizar abordagens de aprendizado de máquina com inúmeros dados para treinamento.

1.2 Formulação do Problema

O processo de localização de defeitos busca identificar artefatos de código-fonte candidatos relacionados ao defeito reportado. Quando essa tarefa é executada de maneira manual, torna-se onerosa para os responsáveis, pois é necessário identificar e realizar as manutenções necessárias no artefato.

As metodologias adotadas para o processo de localização de defeitos que têm como objetivo localizar artefatos candidatos a partir de um relatório de defeito geralmente utilizam técnicas de aprendizado de máquina, que consis-

tem em similaridade textual, algoritmos de classificação e agrupamento de arquivos de código-fonte de acordo com os dados extraídos dos relatórios de defeitos [Gharibi et al. (2018), Swe e Oo (2018), Chen et al. (2019), Loyola et al. (2018), Lam et al. (2017) e Rahman e Roy (2018)].

Existem duas principais abordagens no estado da arte para o processo de localização de defeitos que consideram relatos de defeitos para a recuperação da informação (Loyola et al., 2018). A primeira abordagem é baseada em similaridade, que consiste em classificar os relatórios de defeitos e os arquivos de código-fonte, obtendo como resultado os arquivos de código-fonte relacionados com o relatório de defeito de acordo o nível de similaridade entre ambos. A segunda abordagem é baseada em técnicas de aprendizado de máquina, que utilizam um conjunto de artefatos para treinamento (relatórios de defeitos históricos e seus respectivos arquivos de código-fonte defeituosos). Esta também utiliza algoritmos para classificar e agrupar documentos em um rótulo relevante por similaridade textual, resultando em uma função de pontuação para determinar os arquivos de código-fonte candidatos para correção.

Nos projetos de Gharibi et al. (2018) e Swe e Oo (2018) referentes a modelos de localização de defeitos, existem lacunas que serão exploradas nessa proposta. Uma das principais lacunas nos modelos está relacionada aos processos utilizados para obter a relação entre os artefatos de código-fonte e os relatórios de defeitos. Nenhum dos trabalhos busca representar semanticamente a estrutura do código-fonte o que aqui se questiona, uma vez que a utilização de ontologias torna possível a interoperabilidade da representação de código-fonte em diferentes linguagens. Além disso, a adição da semântica pode resultar na recuperação da informação relacionada as descrições dos relatórios de defeitos e na desambiguação de entidades identificadas nos relatos de defeitos.

As metodologias de localização de defeitos atuais consideram apenas a análise estática dos arquivos de código-fonte, não observando sua estrutura arquitetônica e semântica.

As abordagens com métodos estáticos que tratam os relatórios de defeitos como uma coleção de palavras não consideram o contexto e não exploram totalmente as informações semânticas contidas nos relatórios de defeitos

Desse modo, levanta-se o problema de como considerar a estrutura arquitetônica e semântica do código-fonte para apoiar o processo de localização de defeitos.

1.3 Objetivos Gerais e Específicos

O presente trabalho propõe um modelo de localização de defeitos com o apoio de ontologia para a representação formal do conhecimento do paradigma de

programação orientado a objetos.

Busca-se realizar a identificação e classificação de artefatos de código-fonte a partir de relatórios de defeitos considerando a estrutura semântica do código-fonte.

A fim de alcançar o objetivo principal, logo uma série de objetivos específicos são definidos:

- Desenvolver um gerador semântico do código-fonte utilizando uma ontologia para modelar o conhecimento do paradigma de programação orientada a objeto.
- Analisar e converter tokens extraídos do código-fonte em conceitos e relações da ontologia do paradigma de programação orientada a objeto.
- Extrair conceitos e instâncias da ontologia para geração de uma base de conhecimento.
- Extrair entidades encontradas no relatório de defeito de acordo com os conceitos do código-fonte.
- Identificar e classificar os artefatos de código-fonte candidatos a manutenção com base no reconhecimento de conceitos extraídos do relatório de defeito.
- Avaliar o modelo de localização de defeitos proposto analisando a eficácia da localização e classificação dos artefatos de código-fonte defeituosos.

1.4 Organização da Monografia

O presente trabalho é composto por 7 capítulos organizados na seguinte estrutura:

- Capítulo 2 fornece a base teórica de conceitos como: Web Semântica, Ontologia, SPARQL e a relação da Web Semântica com a área de Engenharia de Software;
- Capítulo 3 aborda conceitos sobre defeitos de software e sistemas de rastreamentos de defeitos;
- Capítulo 4 são apresentados os trabalhos correlatos e as discussões que envolvem esta proposta;
- Capítulo 5 tem como objetivo apresentar ao leitor a proposta de pesquisa e metodologia para o desenvolvimento do projeto de pesquisa;
- Capítulo 6 detalha o conjunto de dados e as métricas utilizadas para aferir o desempenho do trabalho.
- Capítulo 7 são apresentadas as considerações finais e os trabalhos futuros desta proposta.

CAPÍTULO 7

Considerações Finais

No presente trabalho é apresentado o projeto LOBUS-OWL, um modelo para o processo de localização de defeitos apoiado por ontologia e o reconhecimento de entidades. Com o uso de ontologias é possível descrever formalmente os dados e metadados de do paradigma de programação orientado a objetos, tornando possível a localização de artefatos de código-fonte candidatos a alteração de acordo com os relatórios de defeitos reportados.

O trabalho de Aguiar et al. (2019) apresentado no Capítulo 4 expõe uma ontologia de código-fonte com suporte a linguagens de programação orientadas a objetos. A ontologia proposta por Aguiar et al. (2019) possibilita o desenvolvimento de novas pesquisas que possam explorar a recuperação da informação no domínio do código-fonte de maneira semântica. Até o trabalho de Aguiar et al. (2019) as pesquisas relacionadas ao uso de ontologias de para a representação do código-fonte careciam de uma estrutura rica em regras e relações.

As pesquisas de Gharibi et al. (2018) e Swe e Oo (2018), expostas no Capítulo 4, utilizam algoritmos para identificar a similaridade semântica e a relação entre os relatos de defeitos e os arquivos de código-fonte. Os trabalhos de Gharibi et al. (2018) e Swe e Oo (2018) apresentam lacunas que são exploradas no presente trabalho. Os dois trabalhos de Gharibi et al. (2018) e Swe e Oo (2018) fazem a análise estática do código-fonte, sem considerar as relações entre os arquivos de código-fonte. Com a utilização de ontologias para a representação paradigma de programação orientado a objetos é possível obter a interoperabilidade em diferentes linguagens de programação e fazer novas inferências no modelo semântico.

O conhecimento semântico do código-fonte possui um papel importante para no modelo de localização de defeitos proposto no presente trabalho.

A localização de defeitos com o uso de ontologias possibilita a interoperabilidade, padronização, organização e reuso da informação de domínio do código-fonte da aplicação. A modelagem da estrutura arquitetônica e semântica do código-fonte é importante para obter artefatos que não são identificados no relatório de defeito e a desambiguação de termos analisando o contexto das demais entidades anotadas.

Com o apoio de ontologias no processo de localização de defeitos é possível realizar o reconhecimento de entidades sem a necessidade de algoritmos de aprendizado de máquina profundos. Com a conclusão deste trabalho espera-se contribuir para a evolução do processo de localização de defeitos auxiliando programadores a otimizar o processo de manutenção corretiva, tornando este processo menos oneroso e mais eficiente.

A aplicação desenvolvida contribui diretamente para a evolução da pesquisa de processos de localização de defeitos em código-fonte propondo uma nova solução com a utilização de ontologia para formalização da semântica do código-fonte. A modelagem do domínio semântico com o apoio de ontologia para o processo de localização de defeito desenvolvido neste trabalho fomenta o avanço de novas pesquisas na área utilizando ontologias para a modelagem semântica.

Durante os experimentos foi observado que a falta de uma estrutura para os relatórios de defeitos do GitHub ameaça a viabilidade da localização de defeitos em alguns casos. Os textos dos relatórios de defeitos no GitHub podem conter imagens, trechos de código-fonte, *stack trace* e etc. Os trechos de código-fonte incluídos nos relatórios de defeitos podem gerar falsos positivos, causando uma identificação indevida de artefatos que não estão relacionados com o defeito.

7.1 Principais Contribuições

A seguir apresentam-se as principais contribuições que este trabalho acrescenta para a área de localização de defeitos:

- Arquitetura de um modelo para a localização de defeitos utilizando o conhecimento de domínio semântico do código-fonte com o apoio de ontologias.
- Processo de compilação e análise de projetos em C Sharp para a serialização de triplas RDF.
- Novas entidades e predicados para a ontologia OOC-O (Aguiar et al., 2019).
- Criação de uma base inédita de conhecimento semântico constituída por

seis projetos de software relevantes de código aberto (AutoMapper, MsBuild, EfCore, AspNetCore, MQTTnet e NLog).

- Conjunto de dados com base na decomposição e combinação dos conceitos instanciados na ontologia de código-fonte dos projetos.
- Processo de reconhecimento de entidades de acordo com a decomposição e combinação dos conceitos da base de conhecimento semântica do código-fonte.
- Processo para classificação semântica dos artefatos de acordo com as métricas CMM e DEM adaptadas para o modelo desenvolvido. E a métrica WEM criada para contribuir para a classificação dos artefatos.
- Processo de agrupamento e geração de ontologias por artefatos de código-fonte.
- Cálculo de ranqueamento adaptado para a classificação das ontologias dos artefatos candidatos.
- Arquitetura e projeto para obter dados de *issues* e *pull requests* de maneira automatizada utilizando o conjunto de APIs disponibilizadas pelo GitHub.
- Conjunto de dados de *issues* e *pull requests* de seis projetos de software relevantes de código aberto (AutoMapper, MsBuild, EfCore, AspNetCore, MQTTnet e NLog) disponibilizados no GitHub.

7.2 Trabalhos Futuros

Para trabalhos futuros, o modelo proposto para a localização de defeitos utilizando o reconhecimento de entidades e ontologias podem ser aprimorados utilizando outras entradas de dados para a recuperação da informação. Adicionando como parâmetro ao modelo a análise de defeitos similares torna possível a verificação de quais artefatos foram impactados no histórico de versão do código-fonte. Desse modo, é possível aprimorar a precisão da recuperação da informação.

Outras melhorias para a evolução do desempenho do projeto LOBUS-OWL podem englobar a evolução dos módulos de reconhecimento de entidades e o conhecimento do domínio da aplicação. O modulo de reconhecimento de entidades pode ser evoluído para o uso de um modelo treinado com base nas informações do domínio da aplicação e de relatórios de defeitos históricos do repositório do projeto. A evolução do conhecimento de domínio da aplicação pode ser realizado com a extensão de ontologias de acordo com o domínio

da aplicação, enriquecendo os conceitos e as relações entre as entidades do domínio do código-fonte e da aplicação.

Além disso, um avanço futuro é a disponibilização desse modelo como um *plugin* nos repositórios dos projetos do GitHub. Desta forma, ao ser relatado um novo defeito o *plugin* poderá analisar a descrição do relatório de defeito e apresentar os artefatos de código-fonte candidatos para correção. Com o plugin vinculado no repositório do código-fonte a cada novo *commit* também é possível manter a ontologia atualizada para a realização da análise dos relatórios de defeitos.

Com o apoio de ontologias também torna possível o desenvolvimento de trabalhos para a evolução de temas relacionados a detecção de *code smells*, localização de *design pattern*, reuso de componentes, localização de vulnerabilidades. Além de possibilitar a inferência de novos conhecimento a partir do domínio do código-fonte.

Referências Bibliográficas

AGUIAR, C. Z. D.; ALMEIDA FALBO, R. D.; SOUZA, V. E. S. Ooc-o: A reference ontology on object-oriented code. In: *International Conference on Conceptual Modeling*, Springer, 2019, p. 13–27.

ALANI, H.; BREWSTER, C.; SHADBOLT, N. Ranking ontologies with aktiverank. In: *International Semantic Web Conference*, Springer, 2006, p. 1–15.

ATZENI, M.; ATZORI, M. Codeontology: Rdf-ization of source code. In: *International Semantic Web Conference*, Springer, 2017, p. 20–28.

AUTOMAPPER Automapper. Online; Acessado em 05/02/2022, 2022a. Disponível em https://github.com/AutoMapper/AutoMapper

AUTOMAPPER v11 invalidoperationexception: Stack empty on mapping which used to generate a subquery 3869. Online; Acessado em 29/05/2022, 2022b. Disponível em https://github.com/AutoMapper/AutoMapper/issues/3869

BERGER OLIVIER, V. V. Helios project's bug ontology draft. Online; Acessado em 29/06/2020, 2010. Disponível em http://heliosplatform.sourceforge.net/ontologies/helios_bt.html

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. *Scientific american*, v. 284, n. 5, p. 34–43, 2001.

BREITMAN, K. K. *Web semântica: a internet do futuro*. Grupo Gen-LTC, 2000.

BRICKLEY, D.; GUHA, R. V.; MCBRIDE, B. Rdf schema 1.1. *W3C recommendation*, v. 25, p. 2004–2014, 2014.

BUGZILLA Bugzilla. Online; Acessado em 29/05/2020, 1998. Disponível em https://www.bugzilla.org/

CHEN, D.; LI, B.; ZHOU, C.; ZHU, X. Automatically identifying bug entities and relations for bug analysis. In: *2019 IEEE 1st International Workshop on Intelligent Bug Fixing (IBF)*, IEEE, 2019, p. 39–43.

DAN BRICKLEY, L. M. Foaf vocabulary specification 0.99. Online; Acessado em 01/02/2022, 2014.
Disponível em http://xmlns.com/foaf/spec

DELAMARO, M.; JINO, M.; MALDONADO, J. *Introdução ao teste de software*. Elsevier Brasil, 2013.

DUCHARME, B. *Learning sparql: querying and updating with sparql 1.1*. "O'Reilly Media, Inc.", 2013.

ECLIPSE Bug 561625. Online; Acessado em 29/05/2020, 2020.
Disponível em https://bugs.eclipse.org/bugs/show_bug.cgi?id=561625

EKRAMIFARD, A.; KAHANI, M. Providing a source code security analysis model using semantic web techniques. In: *2015 International Congress on Technology, Communication and Knowledge (ICTCK)*, IEEE, 2015, p. 33–37.

EXPLOSION spacy-models. Online; Acessado em 01/05/2022, 2022.
Disponível em https://github.com/explosion/spacy-models/releases/tag/en_core_web_md-3.3.0

FERNEDA, E. *Recuperação de informação: Análise sobre a contribuição da ciência da computação para a ciência da informação*. Tese de Doutoramento, Universidade de São Paulo, 2003.

GASEVIC, D.; KAVIANI, N.; MILANOVIĆ, M. Ontologies and software engineering. 2009, p. 593–615.

GHARIBI, R.; RASEKH, A. H.; SADREDDINI, M. H.; FAKHRAHMAD, S. M. Leveraging textual properties of bug reports to localize relevant source files. *Information Processing & Management*, v. 54, n. 6, p. 1058–1076, 2018.

GITHUB Docs gihub - about issues. Online; Acessado em 15/02/2022, 2022a.
Disponível em https://docs.github.com/pt/issues/tracking-your-work-with-issues/about-issues

GITHUB Docs gihub - managing labels. Online; Acessado em 15/02/2022, 2022b.
Disponível em https://docs.github.com/pt/issues/using-labels-and-milestones-to-track-work/managing-labels

GLOVE Common crawl. Online; Acessado em 05/02/2022, 2022. Disponível em https://nlp.stanford.edu/projects/glove/

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, v. 43, n. 5-6, p. 907–928, 1995.

GUARINO, N.; OBERLE, D.; STAAB, S. What is an ontology? In: *Handbook on ontologies*, Springer, p. 1–17, 2009.

GUARINO, N.; ET AL. Formal ontology and information systems. In: *Proceedings of FOIS*, 1998, p. 81–97.

GUJRAL, S.; SHARMA, G.; SHARMA, S.; ET AL. Classifying bug severity using dictionary based approach. In: *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, IEEE, 2015, p. 599–602.

HAPPEL, H.-J.; SEEDORF, S. Applications of ontologies in software engineering. In: *Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC*, Citeseer, 2006, p. 5–9.

HESSE, W. Ontologies in the software engineering process. In: *EAI*, 2005, p. 3–16.

IEEE, S. E. T. C. Ieee standard glossary of software engineering terminology. Institute of Electrical and Electronics Engineers, 1983.

ISOTANI, S.; BITTENCOURT, I. I. *Dados abertos conectados: Em busca da web do conhecimento*. Novatec Editora, 2015.

JENA, A. Apache jena fuseki. Online; Acessado em 05/09/2021, 2021. Disponível em https://jena.apache.org/documentation/fuseki2/

JEONG, G.; KIM, S.; ZIMMERMANN, T. Improving bug triage with bug tossing graphs. In: *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, p. 111–120.

KHAN, W.; DAUD, A.; NASIR, J. A.; AMJAD, T. A survey on the state-of-the-art machine learning models in the context of nlp. *Kuwait journal of Science*, v. 43, n. 4, 2016.

KIEFER, C.; BERNSTEIN, A.; TAPPOLET, J. Analyzing software with isparql. In: *Proc. rd International Workshop on Semantic Web Enabled So ware Engineering (SWESE')..(Cit. on p.)*, 2007.

LAM, A. N.; NGUYEN, A. T.; NGUYEN, H. A.; NGUYEN, T. N. Bug localization with combination of deep learning and information retrieval. In: *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, IEEE, 2017, p. 218–229.

LASSILA, O.; SWICK, R. R.; ET AL. Resource description framework (rdf) model and syntax specification. 1998.

LOYOLA, P.; GAJANANAN, K.; SATOH, F. Bug localization by learning to rank and represent bug inducing changes. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, p. 657–665.

MANNING, C.; RAGHAVAN, P.; SCHÜTZE, H. Introduction to information retrieval. *Natural Language Engineering*, v. 16, n. 1, p. 100–103, 2010.

MICROSOFT Roslyn. Online; Acessado em 05/09/2021, 2021.
Disponível em https://github.com/dotnet/roslyn

MICROSOFT Aspnetcore. Online; Acessado em 05/02/2022, 2022a.
Disponível em https://github.com/dotnet/aspnetcore

MICROSOFT Botframework solutions. Online; Acessado em 05/02/2022, 2022b.
Disponível em https://github.com/microsoft/botframework-solutions

MICROSOFT Csharp coding conventions. Online; Acessado em 05/02/2022, 2022c.
Disponível em https://docs.microsoft.com/pt-br/dotnet/csharp/fundamentals/coding-style/coding-conventions#naming-conventions

MICROSOFT Efcore. Online; Acessado em 05/02/2022, 2022d.
Disponível em https://github.com/dotnet/efcore

MICROSOFT Mqttnet. Online; Acessado em 05/02/2022, 2022e.
Disponível em https://github.com/dotnet/MQTTnet

MICROSOFT Msbuild. Online; Acessado em 05/02/2022, 2022f.
Disponível em https://github.com/dotnet/msbuild

MIZOGUCHI, R. Part 3: Advanced course of ontological engineering. *New Generation Computing*, v. 22, n. 2, p. 193–220, 2004.

NLOG Nlog. Online; Acessado em 05/02/2022, 2022.
Disponível em https://github.com/NLog/NLog

OMG, O. M. G. Ontology definition metamodel (odm). Online; Acessado em 29/05/2020, 2014.
Disponível em https://www.omg.org/

PALSHIKAR, G. K. Techniques for named entity recognition: a survey. In: *Bioinformatics: Concepts, Methodologies, Tools, and Applications*, IGI Global, p. 400–426, 2013.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. Glove: Global vectors for word representation. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, p. 1532–1543.

RAHMAN, M. M.; ROY, C. K. Improving ir-based bug localization with context-aware query reformulation. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, p. 621–632.

SEGUNDO, J. E. S.; CONEGLIAN, C. S. Web semântica e ontologias: um estudo sobre construção de axiomas e uso de inferências. *Informação & Informação*, v. 21, n. 2, p. 217–244, 2016.

SEGUNDO, J. E. S.; CONEGLIAN, C. S.; LUCAS, E. R. D. O. Conceitos e tecnologias da web semântica no contexto da colaboração acadêmico-científica: um estudo da plataforma vivo. *Transinformação*, v. 29, n. 3, p. 297–309, 2017.

SONG, H.-J.; JO, B.-C.; PARK, C.-Y.; KIM, J.-D.; KIM, Y.-S. Comparison of named entity recognition methodologies in biomedical documents. *Biomedical engineering online*, v. 17, n. 2, p. 1–14, 2018.

SPACY spacy. Online; Acessado em 05/10/2021, 2021.
Disponível em https://spacy.io/

STORY, H. Baetle. Online; Acessado em 29/06/2020, 2012.
Disponível em https://code.google.com/archive/p/baetle/

STUCKENSCHMIDT, H.; VAN HARMELEN, F. *Information sharing on the semantic web*. Springer Science & Business Media, 2005.

SWE, K. E. E.; OO, H. M. Bug localization approach using source code structure with different structure fields. In: *2018 IEEE 16th International Conference on Software Engineering Research, Management and Applications (SERA)*, IEEE, 2018, p. 159–164.

TRAN, H. M.; LE, S. T. Software bug ontology supporting semantic bug search on peer-to-peer networks. *New Generation Computing*, v. 32, n. 2, p. 145–162, 2014.

UDDIN, J.; GHAZALI, R.; DERIS, M. M.; NASEEM, R.; SHAH, H. A survey on bug prioritization. *Artificial Intelligence Review*, v. 47, n. 2, p. 145–180, 2017.

VOORHEES, E. M. The trec question answering track. *Natural Language Engineering*, v. 7, n. 4, p. 361–378, 2001.

W3C Web ontology language (owl). Online; Acessado em 07/05/2020, 2012.
Disponível em https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

W3C Web ontology language (owl). Online; Acessado em 07/05/2020, 2013. Disponível em https://www.w3.org/OWL/

W3C Semantic web ontology. Online; Acessado em 07/05/2020, 2015. Disponível em https://www.w3.org/standards/semanticweb/ontology

WITTE, R.; ZHANG, Y.; RILLING, J. Empowering software maintainers with semantic web technologies. In: *European Semantic Web Conference*, Springer, 2007, p. 37–52.

ZHANG, T.; LEE, B. A bug rule based technique with feedback for classifying bug reports. In: *2011 IEEE 11th International Conference on Computer and Information Technology*, IEEE, 2011, p. 336–343.

ZIBRAN, M. F. On the effectiveness of labeled latent dirichlet allocation in automatic bug-report categorization. In: *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, IEEE, 2016, p. 713–715.