Vitor Botelho Vaz de Melo

Identificando Linhas de Códigos Comentadas em Repositórios de Software

Proposta de Projeto de Pesquisa e Tecnológico da disciplina de Projeto Orientado em Computação do DCC/UFMG

Orientador:
André Hora - Departamento de Ciência da Computação

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Belo Horizonte

2019/1

1 INTRODUÇÃO

À medida que a Ciência da Computação avança e milhões de linhas de códigos são escritas, diariamente e em diversas linguagens de programação, os desenvolvedores percebem que, para alcançar sucesso a médio e longo prazo em seus projetos, é necessário utilizar-se, cada vez mais, das melhores ferramentas e práticas de programação.

Uma prática de programação muito comum ao se desenvolver softwares é a de ”remover” uma linha ou um bloco de código tornando-os comentários, sendo conhecida como commented-out code (Figura 1.1). Comentar uma linha de código pode ser útil, tanto que os próprios ambientes de programação (IDE's e editores de texto) oferecem atalhos fáceis para isso. Contudo, esta prática se torna um verdadeiro problema quando estes códigos comentados estão inseridos em um sistema grande, complexo e mantido por diversas pessoas.(1).

```
61      h2 = hashlib.sha512(word2.encode('utf-8')).hexdigest()
62
63  #    print('Hash1 [{}]:\n{}\n\nHash2 [{}]:\n{}\n'.format(word1, h1, word2, h2))
64
65  #    # Use list so we can shuffle it
66  #    rez = []
67  #    for a, b in zip(h1, h2):
68  #        rez.append(chr(ord(a) ^ ord(b)))
69
70  #    #print(str(base64.a85encode(''.join(rez).encode())))
71
72  #    # Shuffle three times for good measure
73  #    random.shuffle(rez)
74  #    random.shuffle(rez)
75      random.shuffle(rez)
76
```

Figura 1.1: Commented-out code example

A inserção de código comentado nos arquivos fontes de um sistema pode causar redução da legibilidade, distração e perda de tempo. Martin, em seu livro clássico Clean Code, enfatiza *Few practices are as odious as commenting-out code. Don't do this!*, ele argumenta que outras pessoas não terão coragem de deletar este código por achar que ele pode ser importante (1).

Além disso, quando um mantenedor se depara com códigos comentados ele pode ter uma

série de dúvidas, como *"Por que este código está comentado? Isso é útil em alguma função? Este código está relacionado com determinada mudança?"*, entre outras questões específicas para cada sistema. Este tipo de reflexão pode no mínimo ser uma perda de tempo, ou até mesmo uma distração para introduzir bugs no sistema (1).

Este trabalho tem como principal objetivo fazer um estudo abrangente desta prática em repositórios de código aberto. No entanto, existem poucos estudos científicos abordando esta problemática e não há, entre as ferramentas mais populares, uma solução que identifica automaticamente este tipo de comentário (2). Portanto, na disciplina Projeto Orientado em Computação I, será desenvolvido uma ferramenta apropriada utilizando técnicas de aprendizado de máquina e, a partir disso, na disciplina Projeto Orientado em Computação II, será realizada a análise de diversos sistemas.

2 *REFERENCIAL TEÓRICO*

2.1 Trabalhos Anteriores

GRIJÓ L. e HORA A. exploraram o problema da inserção de código comentado e demostraram interessantes resultados no artigo Minerando Código Comentado (2). No trabalho foi desenvolvido um parser heurístico que identifica *commented-out code* para a linguagem java que, e obteve uma precisão de 83%. Com esta ferramenta foi identificado que alguns sistemas possuem como mediana 4,17% de taxa de commented-out code em relação ao total de comentários, com alguns sistemas chegando a 30%.

Neste trabalho é proposto uma metodologia multi-linguagem baseada em técnicas de aprendizado de máquina. Tem se a ambição de superar a precisão da ferramenta desenvolvida no trabalho anterior, possibilitando a validação dos resultados encontrados e generalizá-lo para diversas linguagens.

2.2 Mineração de Repositórios de Software

Para se entender as práticas e diversas informações sobre um sistema pode-se analisar as diferentes versões do código deste sistema. Dessa forma pode-se identificar em um sistema as boas e más práticas presentes, questões arquiteturais, entre outras questões, com o objetivo de melhorar a manutenibilidade deste sistema. Através da mineração de repositório de software podemos predizer quais seções do código apresentam mais defeitos e maior dificuldade de compreensão; usar a evolução do software para identificar partes do código que são mais importantes para manutenção; entender como diversos desenvolvedores e times afetam a qualidade do código. (3).

2.3 Git e GitHub

Git[1] é uma ferramenta amplamente usada na comunidade de desenvolvimento de software para controle de versão de código. Ela permite que seja avaliado diferentes versões do código de um mesmo sistema (4). O Github[2] é uma plataforma que permite hospedar repositórios de código com o sistema de versionamento Git na nuvem. Nele é possível encontrar diversos repositórios de código aberto que serão as fontes de estudo deste trabalho.

2.4 Algoritmos de Classificação

Os algoritmos de classificação são algoritmos de aprendizado supervisionado, no qual utilizamos uma base de treino em que previamente sabemos a ”classe” de cada objeto para treinar um modelo que pode dizer qual a classe dos objetos dos quais não conhecemos (5).

Podemos modelar o problema de identificar *commented-out code* como sendo um problema de classificação binária. Neste caso, os objetos serão os comentários presentes em códigos de diversas linguagens, e a classe será se aquele comentário é normal representado por 0 ou código comentado representado por 1.

2.4.1 Precisão e Revocação

As taxas de precisão (*precision*) e revocação (*recall*) nos permite avaliar a qualidade do classificador (6). Elas são definidas como:

$$precision = \frac{tp}{tp+fp} \qquad recall = \frac{tp}{tp+fn}$$

- tp = True Positive: Elementos classificados como 1 ou positivos corretamente.
- fp = False Positives: Elementos classificados como 1 ou positivos incorretamente.
- fn = False Negative: Elementos classificados como 0 ou negativos incorretamente.

[1] https://git-scm.com/
[2] https://github.com/

3 METODOLOGIA

3.1 Extração e Preparação dos dados

Os comentários deverão ser extraídos de diferentes repositórios de software. Em sequência, estes comentários deverão ser classificados manualmente como *commented-out code* ou comentário normal a fim de se formar a base de treinamento. Esta base deverá contemplar diferentes tipos de linguagem para que o classificador consiga identificar códigos comentados independente de linguagem. Dessa forma teremos uma base de dados em que o X será igual a uma linha de comentário e o Y igual a 0 se for um comentário normal ou 1 se for um código comentado, que servirão, respectivamente, como input e output para o treinamento do classificador.

3.2 Treinamento de Modelos de Classificação

Figura 3.1: Procedimento básico para definição do modelo de classificação

A figura 3.1 ilustra os passos básicos para encontrar um modelo de classificação [1]. Primeiramente, a partir dos dados coletados e classificados, devemos pré-processar os textos de co-

[1] Exemplo extraído de https://gabrielschade.github.io/2018/01/08/machine-learning-intro.html

mentários para que possam ser *inputs* do modelo. Pretende-se usar uma representação binária, baseada em caracteres, que será melhor definida ao longo do trabalho. Em sequência, será selecionada uma variedade de modelos/algoritmos de classificação, como, por exemplo, redes neurais multi-layer perceptron (7) (5), LSTM (8) , dentre outros. Por fim, será computado os índices de precisão e revocação dos modelos para se definir o melhor modelo.

3.3 Compilação de dados de Repositórios

Com a ferramenta de identificação de códigos comentados preparada, serão analisados diversos repositórios relevantes de softwares de código aberto, encontrados no GitHub. Ao identificar os comentários, será possível calcular a taxa de código comentado em relação ao total de comentários em uma versão do software, e com o auxilio da ferramenta Git será analisado a evolução do software através do código em diferentes versões. Com isso teremos indícios de como os desenvolvedores estão lidando com essa prática, entre diversas outras informações que podem ser exploradas com este tipo de técnica.

4 RESULTADOS ESPERADOS

Espera-se que ao final da disciplina Projeto Orientado em Computação I obtenha-se uma ferramenta robusta e com alto nível de precisão para identificar códigos comentados em diversas linguagens de programação. No final da disciplina Projeto Orientado em Computação II espera-se compreender como a má prática de remover código através de comentário afeta a evolução dos diferentes repositórios de código aberto, respondendo perguntas como: **(1)** Qual a taxa de código comentado média nos repositórios de software? **(2)** Como essa taxa evolui ao longo das versões. **(3)** Ela varia dependendo da linguagem? **(4)** Existe uma correlação entre arquivos mais modificados e a presença de código comentado?, dentre outras questões.

5 ETAPAS E CRONOGRAMAS

Período	Atividade
19/03 a 12/04	Estudo e elaboração da proposta do projeto.
13/04 a 28/04	Criação de uma ferramenta para extração dos comentários em repositórios de software abertos.
29/04 a 19/05	Classificação dos dados extraídos para definição da base treino. Elaboração e Apresentação intermediária.
20/05 a 16/06	Treinamento e avaliação de um modelo de classificação para identificação de *commented-out code*.
17/06 a 05/07	Elaboração do artigo, poster e apresentação final. Submissão do artigo científico com os resultados parciais para o evento VEM (Workshop on Software Visualization, Evolution and Maintenance)

Na disciplina de Projeto Orientado em Computação II será desenvolvido o restante das atividades, sendo elas: seleção e extração dos repositórios a serem avaliados, definição de métricas relativas a *commented-out code*, desenvolvimento de uma ferramenta para computar, salvar e visualizar os resultados encontrados.

Referências Bibliográficas

1 MARTIN, R. C. *Clean Code: a handbook of agile software craftsmanship*. [S.l.: s.n.], 2009.

2 GRIJó, L.; HORA, A. Minerando codigo comentado. In: *6th Brazilian Workshop on Software Visualization, Evolution and Maintenance*. [S.l.]: VEM, 2017. p. 1–7.

3 TORNHILL, A. *Your Code as Crime Scene: Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs*. [S.l.: s.n.], 2015.

4 BIRD, C. et al. The promises and perils of mining git. *Mining Software Repositories, International Workshop on*, v. 0, p. 1–10, 05 2009.

5 DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern Classification*. [S.l.: s.n.], 2015.

6 POWERS, D. M. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. Bioinfo Publications, 2011.

7 ZURADA, J. M. *Introduction to artificial neural systems*. [S.l.]: West publishing company St. Paul, 1992.

8 GERS, F. A.; SCHMIDHUBER, J.; CUMMINS, F. Learning to forget: Continual prediction with lstm. IET, 1999.