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

2019/2

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 3.1). Remover uma linha de código por comentário 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 R., 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).

Na disciplina Projeto Orientado em Computação I foi desenvolvida uma ferramenta apropriada para identificar *commented-out code* utilizando técnicas de aprendizado de máquina. Portanto, na disciplina Projeto Orientado em Computação II será realizado um estudo abrangente desta prática em diversos repositórios de código aberto.

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%.

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

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

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.

3 METODOLOGIA

3.1 Construção de um pipeline de execução

Na disciplina de Projeto Orientado em Computação I foram criadas duas principais ferramentas, A primeira separa um arquivo ”completo” de código em código e comentário. A segunda identifica, dentro dos comentários, o que seria *commented-out code*. Além disso, há bibliotecas para o Git e GitHub que permitem caminhar via software por diversas versões de um repositório.

Portanto será criada uma ferramenta que integra todas as etapas necessárias para a coleta de informações relativas a esta prática em diferentes versões de diversos repositórios de software.

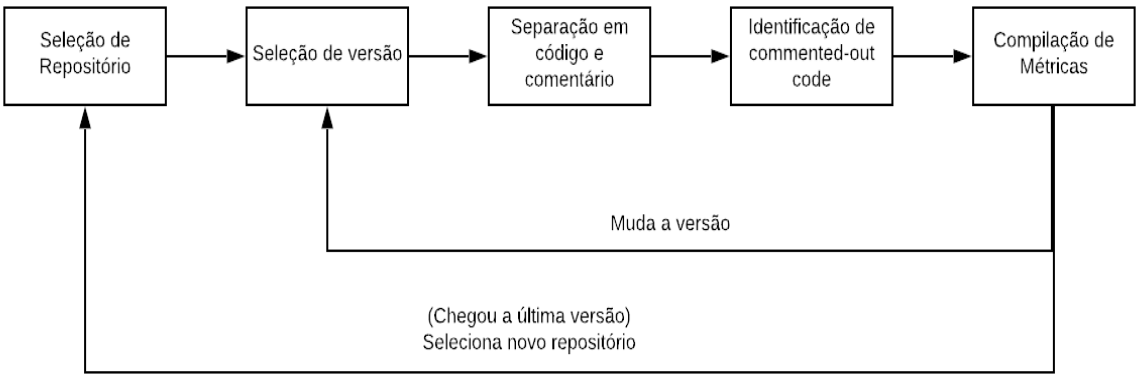

Figura 3.1: Fluxo de execução do algoritmo

3.2 Compilação de dados de Repositórios

Com a ferramenta 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

Ao final da disciplina Projeto Orientado em Computação I foi obtida 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
11/08 a 08/09	Consolidação do POC I e elaboração de proposta para o POC II
09/09 a 29/09	Criação do pipeline de execução.
30/09 a 04/10	Elaboração e Apresentação intermediária.
05/10 a 20/10	Execução do algoritmo em diversos repositórios de software
21/10 a 10/11	Visualização e análise dos dados coletados
11/11 a 05/12	Elaboração do artigo, poster e apresentação final.

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.