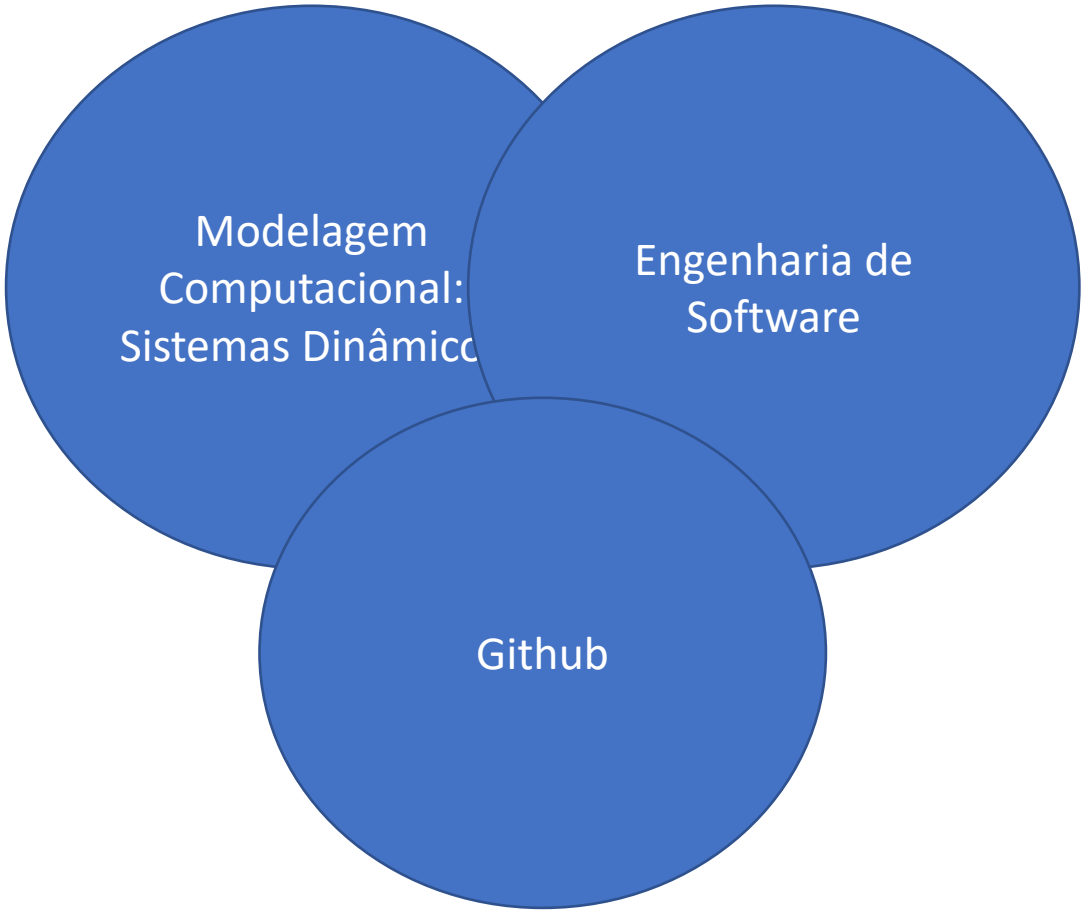
Sistemas Dinâmicos para projetos de software com foco em correção de bugs

Março 2021

Orientando: Giseldo

Orientador: Dr. Tárcio e Dr. Antão

Temas


Modelagem Computacional: Sistemas Dinâmicc
Engenharia de Software
Github

Questões de Pesquisa

- Quais os fatores de sucesso dos projetos de software no github?
- Um sistema dinâmico com estes fatores pode auxiliar os stakeholders que utilizam o github?
- O que afeta o tempo de correção de uma issue? A quantidade de pessoas, o tamanho do projeto?

Hipótese

Um sistema dinâmico pode ajudar os stakeholders em engenharia de software a diminuir o tempo de correção de BUGS.

Objetivo

Analisar a relação dinâmica entre o tempo de inclusão de issue e o tempo de sua correção.

Secundário:

Quais os fatores que mais impactam a correção?

Metodologia

- Analisar qualitativamente e quantitativamente os projetos no github
- Mapear as variáveis de impacto
- Disponibilizar o modelo
- Testar o Modelo

Referencial

The Promises and Perils of Mining GitHub

Eirini Kalliamvakou
University of Victoria
Canada
ikaliam@uvic.ca

Georgios Gousios
Delft University of Technology
Canada
G.Gousios@tudelft.nl

Kelly Blincoe
University of Victoria
Canada
kblincoe@acm.org

Leif Singer
University of Victoria
Canada
lsinger@uvic.ca

Daniel M. German*
University of Victoria
Canada
dmg@uvic.ca

Daniela Damian
University of Victoria
Canada
danielad@cs.uvic.ca

ABSTRACT

With over 10 million `git` repositories, GitHub is becoming one of the most important source of software artifacts on the Internet. Researchers are starting to mine the information stored in GitHub's event logs, trying to understand how its users employ the site to collaborate on software. However, so far there have been no studies describing the quality and properties of the data available from GitHub. We document the results of an empirical study aimed at understanding the characteristics of the repositories in GitHub and how users take advantage of GitHub's main features—namely commits, pull requests, and issues. Our results indicate that, while GitHub is a rich source of data on software development, mining GitHub for research purposes should take various potential perils into consideration. We show, for example, that the majority of the projects are personal and inactive; that GitHub is also being used for free storage and as a Web hosting service; and that almost 40% of all pull requests do not appear as merged, even though they were. We provide a set of recommendations for software engineering researchers on how to approach the data in GitHub.

1. INTRODUCTION

GitHub is a collaborative code hosting site built on top of the `git` version control system. GitHub introduced a "fork & pull" model in which developers create their own copy of a repository and submit a pull request when they want the project maintainer to pull their changes into the main branch. In addition to code hosting, collaborative code review, and integrated issue tracking, GitHub has integrated social features. Users are able to subscribe to information by "watching" projects and "following" users, resulting in a feed of information on those projects and users of interest. Users also have profiles that can be populated with identifying information and contain their recent activity within the site.

With over 10.6 million repositories[1] hosted as of January 2014, GitHub is currently the largest code hosting site in the world. Its popularity, the integrated social features, and the availability of metadata through an accessible API have made GitHub very attractive for software engineering researchers. Existing research has been both qualitative [4, 7, 16, 17, 19] and quantitative [10, 24, 25, 26]. Qualitative studies have focused on how developers use GitHub's social features to form impressions and draw conclusions on other developers' and

Tempo e fatores que afetam a correção de BUGS seria possível coletar quantitativamente?

Referencial

ELSEVIER

Information and Software Technology

Volume 93, January 2018, Pages 130-146

Are you smelling it? Investigating how similar developers detect code smells

Mário Hozano [a, b], Alessandro Garcia [c], Baldoino Fonseca [d], Evandro Costa [d]

Show more

+ Add to Mendeley Share Cite

https://doi.org/10.1016/j.infsof.2017.09.002

Abstract

Context

A code smell indicates a poor implementation choice that often worsens software quality. Thus, code smell detection is an elementary technique to identify refactoring opportunities in software systems. Unfortunately, there is limited knowledge on how similar two or more developers detect smells in code. In particular, few studies have investigated if developers agree or disagree when recognizing a smell and which factors can influence on such (dis)agreement.

Objective

We perform a broader study to investigate how similar the developers detect code smells. We also analyze whether certain factors related to the developers' profiles concerning background and experience may influence such (dis)agreement. Moreover, we analyze if the heuristics adopted by developers on detecting code smells may influence on their (dis)agreement.

Method

We conducted an empirical study with 75 developers who evaluated instances of 15 different code smell types. For each smell type, we analyzed the agreement among the developers and we assessed the influence of 6 different factors on the developers' evaluations. Altogether more than 2700 evaluations were collected, resulting in substantial quantitative and qualitative analyses.

Results

The results indicate that the developers presented a low agreement on detecting all 15 smell types analyzed in our study. The results also suggest that factors related to background and experience did not have a consistent influence on the agreement among the developers. On the other hand, the results show that the agreement was consistently influenced by specific heuristics employed by developers.

Conclusions

Our findings reveal that the developers detect code smells in significantly different ways. As a consequence, these findings introduce some questions concerning the results of previous studies that did not consider the different perceptions of developers on detecting code smells. Moreover, our findings shed light towards improving state-of-the-art techniques for accurate, customized detection of code smells.

Resultados Esperados

- Auxiliar os stakeholders com o sistema dinâmico mapeado na área de engenharia de software e correção de Bugs

Referências

- Tese “Capturando a Dinâmica da Gestão da Terceirização de Tecnologia da Informação para o Apoio a Decisões: Um estudo de caso em organizações públicas” Bezerra, 2015.

- Livro “Software Process Dynamic” Madachy 2007