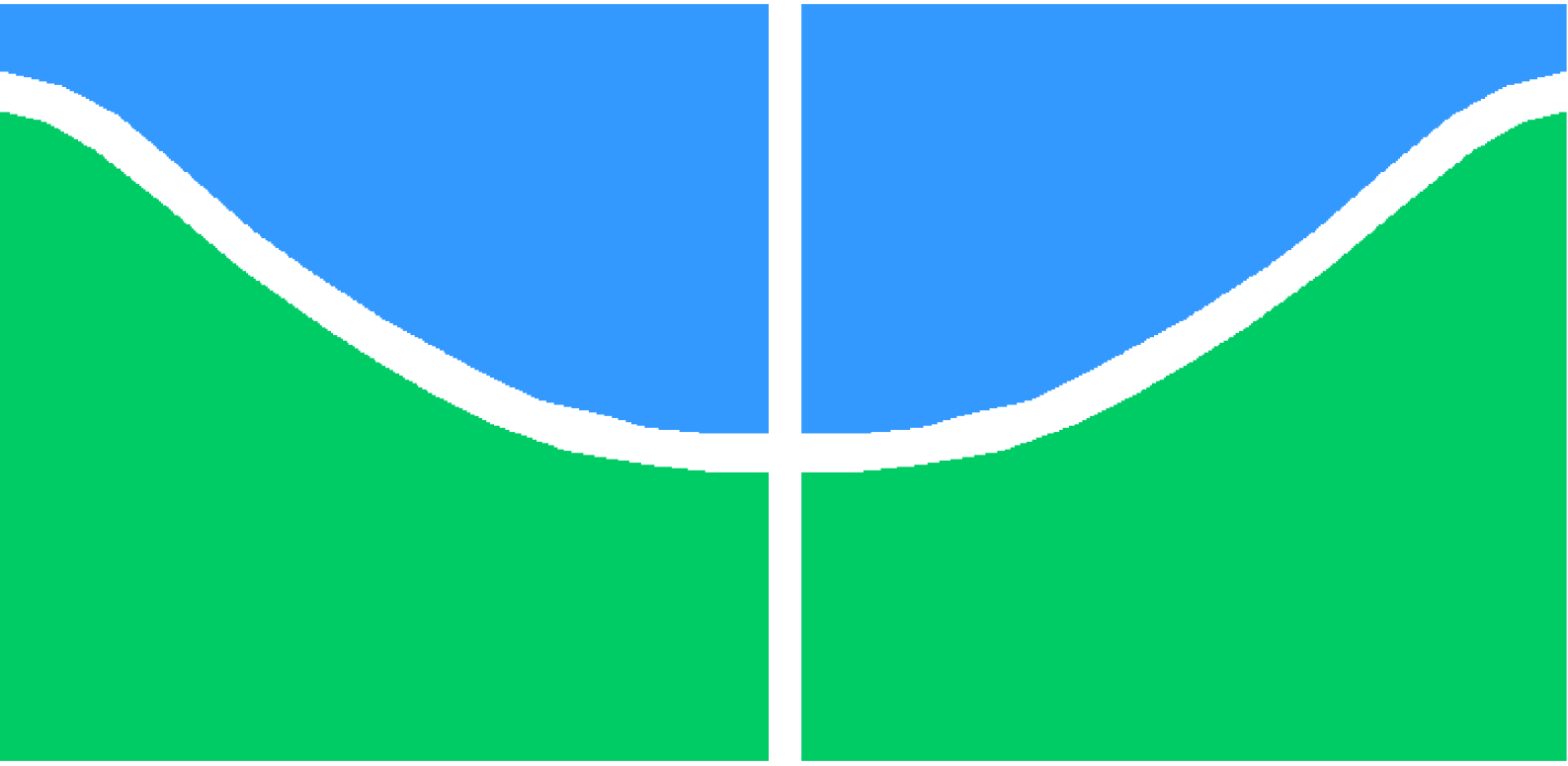

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Aplicação de Software Analytics e o Algoritmo de Ranqueamento de Páginas Para Apoiar Decisões de Planejamento de Sprint

Autor: Alessandro Caetano Beltrão

Orientador: (Prof. Msc. Hilmer Rodrigues Neri)

Brasília, DF

2016

Alessandro Caetano Beltrão

Aplicação de Software Analytics e o Algoritmo de Ranqueamento de Páginas Para Apoiar Decisões de Planejamento de Sprint

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: (Prof. Msc. Hilmer Rodrigues Neri)

Coorientador: (Prof. Dr. Paulo Roberto Miranda Meirelles)

Brasília, DF

2016

Alessandro Caetano Beltrão

Aplicação de Software Analytics e o Algoritmo de Ranqueamento de Páginas Para Apoiar Decisões de Planejamento de Sprint/ Alessandro Caetano Beltrão. – Brasília, DF, 2016-

58 p. : il. (algumas color.) ; 30 cm.

Orientador: (Prof. Msc. Hilmer Rodrigues Neri)

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Software Analytics. 2. Page Ranking. I. (Prof. Msc. Hilmer Rodrigues Neri). II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Aplicação de Software Analytics e o Algoritmo de Ranqueamento de Páginas Para Apoiar Decisões de Planejamento de Sprint

CDU 02:141:005.6

Alessandro Caetano Beltrão

Aplicação de Software Analytics e o Algoritmo de Ranqueamento de Páginas Para Apoiar Decisões de Planejamento de Sprint

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 23 de junho de 2016:

(Prof. Msc. Hilmer Rodrigues Neri)
Orientador

(Profa. Dra. Carla Silva Rocha Aguiar)
Convidado 1

(Prof. Msc. Renato Coral Sampaio)
Convidado 2

Brasília, DF
2016

Agradecimentos

Agradeço ao meu orientador, Hilmer Rodrigues Neri, e coorientador, Paulo Roberto Miranda Meirelles, pela confiança, pelo conhecimento transimitido durante o curso, as aulas e as interações no ambiente do LAPPIS, por me motivarem a ser um aluno melhor e pela convivência. Agradeço também a todos meus amigos do LAPPIS, pela experiência compartilhada durante todos os anos de projeto, por me motivarem a ser um profissional cada vez melhor e pelas risadas e momentos de descontração.

Agradeço ao meu pai, Mario Luiz Beltrão, e minha mãe, Diva Luiza Caetano, pela paciência, motivação, carinho e por permitirem que eu cursasse este curso me apoiando de todas as formas possíveis, mesmo em momentos de dificuldade. Agradeço ao meu irmão, Matheus Caetano Beltrão, por me motivar, pelas risadas e por me ajudar a passar por momentos de dificuldade com alegria.

Agradeço a Rayane Alves Ferreira por todas as alegrias e momentos que me proporciona, por suportar a distância entre nós e estar ao meu lado momentos de estresse, desequilíbrio e tristeza, tornando assim meus dias mais belos e incríveis. À ela, todo meu amor.

Agradeço também aos meus amigos Marcos Alexandre Torres, Rafael Fogaça, Pedro Henrique Martins Faria, Marcos Paulo, Rodrigo Alves, Idelmino Ramos Neto, Guilherme Aguiar da Silva, Layara Karla, Laís Paula Cenci, Caik Hott, Janini Hott, Valéria Cortes, Alexandre Artur, Marco Aurélio e Samara Anjos. À eles, toda a minha gratidão.

"A mente não deve ser modificada pelo tempo e pelo lugar.
A mente é o seu próprio lugar, e dentro de si.
Pode fazer um inferno do céu, e do céu um inferno."
(Paraíso Perdido - John Milton)

Resumo

A utilização de metodologias ágeis para o desenvolvimento de software vem sendo uma crescente nos últimos anos, com isso, trazendo a tona as dificuldades intrínsecas desta atividade que também estão presentes na metodologia ágil. Uma das dificuldades que se destacam e é uma constante em todos os processos, independente de metodologia, é a dificuldade de planejar entregas, ou releases. Para reduzir o impacto das incertezas dentro de projetos, principalmente os de grande complexidade e níveis de abstração, foram propostas pela literatura diversos métodos e tecnicas, entre elas a utilização de análise de dados para tomar decisões e planejar o futuro. Este trabalho propõe da utilização de *Software Analytics*, para a anáĺise de relevancia de *issues* de repositórios de código aberto com a utilização de algoritmos de *Page Ranking* e centralidade de redes para apoiar a tomada de decisões de planejamento de sprint e a priorização de atividades.

Palavras-chaves: Planejamento de Release. Software Analytics. Ranqueamento de Páginas, Priorização de Atividades.

Abstract

The use of agile methodologies for software development has been increasing in recent years, thereby bringing out the intrinsic difficulties this activity which are also present in the agile methodology. One of difficulties that stand out and is a constant in all processes, independent methodology is the difficulty of planning deliveries, or releases. To reduce the impact of uncertainties in projects, especially those of great complexity and abstraction levels, have been proposed in the literature various methods and techniques, including the use of data analysis to make decisions and plan the future. This paper proposes the use of *Software Analytics* for the relevance analysis of *issues* on open source repositories with the use of *Page Ranking* algorithms and network centrality to support sprint planning decisions and prioritizing activities.

Key-words: Release Planning, Software Analytics, Page Ranking, Prioritizing Activities.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

SPB	Software Público Brasileiro
MPOG	Ministério do Planejamento, Orçamento e Gestão

Sumário

1 Introdução

1.1 Contexto

Os métodos de desenvolvimento ágeis surgiram como uma alternativa aos métodos de desenvolvimento chamados de tradicionais. Ao invés de planejar, analisar e projetar para um futuro em médio e longo prazo, a proposta dos métodos ágeis é que essas atividades, por exemplo, sejam realizadas continuamente em ciclos de produção[1] curtos, de 2 a 4 semanas, durante todo o ciclo de vida do projeto. Para representar essa nova forma de olhar para o desenvolvimento de software, quatro valores foram propostos e enfatizados por um grupo de experientes desenvolvedores que se tornaram signatários do conhecido manifesto ágil (BECK et al., 2001):

- **Indivíduos e interação** entre eles mais que processos e ferramentas.
- **Software em funcionamento** mais que documentação abrangente.
- **Colaboração com o cliente** mais que negociação de contratos.
- **Responder a mudanças** mais que seguir um plano.

No ciclo de desenvolvimento ágil o cliente direciona as *releases* ao priorizar os requisitos, representados pelas histórias de usuário, que possuem maior valor para o negócio. A cada iteração, o cliente escolhe um conjunto de histórias que devem ser implementadas, respeitando a capacidade produtiva do time. A partir disso, são identificadas as atividades necessárias para que uma determinada história seja desenvolvida e estas são atribuídas aos programadores. São atividades de diferentes áreas da engenharia de software, como por exemplo: projetar, codificar, testar, medir, implantar, gerenciar configurações do software. Ao final da *release*, uma versão de produto de software é entregue ao cliente e o ciclo se inicia novamente.

O planejamento de *release* é uma atividade essencial realizada no início de cada novo ciclo de produção. Nesse momento são definidas as funcionalidades que serão implementadas para próxima versão de liberação do produto, além dos demais recursos do projeto que deverão ser alocados. Mesmo com o alto nível de interação entre as pessoas, característico em projetos de desenvolvimento ágil, o planejamento de release é uma tarefa difícil tanto computacionalmente quanto cognitivamente, pois diversos tipos de incertezas tornam difícil a definição e estruturação do problema a ser atacado(NGO-THE; RUHE, 2008).

[1] Neste trabalho os termos ciclo de produção, iteração e *sprint* são utilizados como sinômino

Desta forma, é fundamental para os times de desenvolvimento ágil que o planejamento de *release* possua um alto nível de confiabilidade(MCDAID et al., 2006).

A disponibilidade de dados confiáveis, na frequência necessária, permite que engenheiros e gerentes tomem decisões que podem ser fundamentais para permitir que o processo de desenvolvimento de software tenha sucesso e que ao final seja entregue um produto de qualidade (CZERWONKA et al., 2013). Processos de desenvolvimento de software adaptativos são constantemente afetados por mudanças nas condições de negócio. O antigo modo de operação reativo precisa ser substituído por decisões em tempo real e proativas baseadas em informações atualizadas e compreensivas(BIRD; MENZIES; ZIMMERMAN, 2015). Na Microsoft, por exemplo, vários times utilizam dados coletados para melhorar processos como:

- Avaliação de riscos e mudança em ferramentas de análise de impacto.
- Otimização da estrutura de *branchs* dos repositórios de código.
- Análise sócio-técnica de dados.
- Busca customizada de bugs e logs.
- Acompanhamento de tendências e relatórios do status de desenvolvimento.

Os métodos atuais de planejamento de *release* utilizam diversas fontes de informação para aprimorar as decisões e diminuir incertezas, estas fontes podem incluir: *stakeholders*, dados coletados de desenvolvedores, informações de *bugtrackers*, entre outras. Os principais defeitos da abordagem tradicional de planejamento de *release* estão relacionados a incapacidade de lidar com as mudanças constantes que estão acontecendo durante a evolução do projeto. A literatura sugere várias técnicas e métodos para reduzir as incertezas deste processo, (BIRD; MENZIES; ZIMMERMAN, 2015), por exemplo, sugere a utilização de *Software Analytics* como caminho para auxiliar na tomada de decisões para o ciclo de desenvolvimento.

Neste trabalho serão investigadas técnicas e ferramentas de mineração em repositórios de software para recuperar e analisar dados que possam apoiar a tomada de decisão sobre na atividade de planejamento de *releases*.

1.2 Problema

O planejamento de *release* é conhecido por ser um problema cognitivamente e computacionalmente difícil (NGO-THE; RUHE, 2008). Esta atividade, dependendo do tamanho e complexidade do projeto, pode incluir centenas de novas *features* que devem ser entregues nas próximas *releases*. Durante o planejamento de *release*, a grande quantidade

de incertezas torna difícil a formulação dos problemas a serem atacados, além disto, as decisões neste processo são tomadas por seres humanos, que utilizam de modelos, objetivos específicos do projeto, comunicação e muitas vezes a intuição resolver os problemas. Para a solução deste problema, alguns métodos são propostos na literatura, como por exemplo o *EVOLVE+*, uma abordagem sistemática proposta por (NGO-THE; RUHE, 2008), que buscar guiar e auxiliar a tomada de decisão por parte dos integrantes de um projeto. Vista a relevância da fase de planejamento de *sprint* em um projeto e a grande quantidade de dificuldades encontradas neste processo, foi formulado então o seguinte problema, que motiva o desenvolvimento desta pesquisa.

Dificuldade na tomada de decisões de priorização de atividades em Release Plannings

1.3 Objetivos

O objetivo geral deste trabalho consiste na realização de um estudo de caso onde será avaliada a relação entre relevância das *issues* de um projeto e a priorização de atividades em um processo de planejamento de *release*. Como o trabalho foi dividido em duas fases, os objetivos específicos da primeira fase foram definidos como:

- Definir técnicas e ferramentas de mineração e recuperação de de dados de software que deverão ser utilizadas.
- Propor uma abordagem de verificação da correlação entre a relevância das *issues* de um repositório e o planejamento de *sprint*.

1.4 Questão de Pesquisa

Diferentemente de outros campos da ciência e engenharia que possuem paradigmas de pesquisa bem desenvolvidos, a engenharia de software ainda não desenvolveu um método ou modelo de guia para estes tipos de estudos. Por este motivo, é comum ver críticas relatórios de pesquisas de software que fogem dos paradigmas tradicionais(SHAW, 2003).A questão de pesquisa é a parte mais importante de um estudo, ela deve guiar toda a metodologia de pesquisa. Um estudo deve fazer perguntas que que identifiquem e definam o escopo das atividades de pesquisa(KITCHENHAM; CHARTERS, 2007a).

Com base no problema identificado, foi possível derivar uma questão de pesquisa que deve ser respondida ao final deste trabalho.

Como podemos verificar a correlação entre issues consideradas relevantes e as funcionalidades planejadas para uma release?

1.4.1 Questões Secundárias

A partir desta questão inicial, é possível definir questões secundárias para permitir a investigação de métodos que possibilitem a solucionar questão primária. Como o cálculo de relevância pode ser realizado de formas diferentes é necessário escolher o melhor método para a realização desta tarefa, além disto métodos de análise devem ser definidos para que a melhor interpretação dos dados seja feita. As questões secundárias foram definidas como as seguintes abaixo:

- Qual é a melhor maneira de calcular a relevância das *issues* de um projeto?
- Como podemos determinar relação entre as relevância das *issues* de um projeto e o planejamento de *sprint* para apoiar a tomada de decisão/priorização?

1.5 Organização do Trabalho

Este trabalho foi dividido em cinco capítulos, são eles:

- **Capítulo 1 - Introdução:** Neste capitulo encontra-se a introdução do trabalho que está dividida nos seguintes tópicos: contexto do trabalho, problema, objetivos e questão de pesquisa.
- **Capítulo 2 - Referencial Teórico:** Neste capitulo são apresentados os conceitos teóricos que fundamentaram este estudo, quais algoritmos serão utilizados e como eles serão aplicados para a execução do exemplo de uso.
- **Capítulo 3 - Metodologia:** Neste capitulo apresenta-se, primeiramente, a metodologia de pesquisa utilizada e, posteriormente, são apresentadas as tecnologias utilizadas para tornar possível a execução deste estudo.
- **Capítulo 4 - Ranqueamento de *Issues* no SPB:** Neste capitulo descreveu-se como o ranqueamento de *issues* foi realizado no projeto escolhido, posteriormente, são apresentados os resultados deste ranqueamento.
- **Capítulo 5 - Conclusão e Próximos Passos:** Além das considerações finais relacionadas ao exemplo de uso realizada nesta primeira parte do trabalho, descreveu-se os objetivos planejados para a segunda parte do trabalho.

2 Referencial Teórico

2.1 Software Analytics

Durante muito tempo a disponibilidade de dados em projetos de software para análise foi um problema. Hoje em dia, com o auxílio da internet, dos projetos de software livre e das características pervasivas e ubíquas da computação, o volume de dados a serem analisados se tornou um problema. Processar e analisar esses dados manualmente se tornou inviável(BIRD; MENZIES; ZIMMERMAN, 2015). Atualmente, por exemplo, pesquisas mostraram que o *Mozilla Firefox* teve 1.3 milhões de relatos de defeitos e outras plataformas como *Sourcefoge.net* e o *Github* hospedam 430.000 e 38 milhões de projetos, respectivamente(BUSE; ZIMMERMANN, 2012).

Na literatura não existe um consenso da definição de *Software Analytics*, o livro (BIRD; MENZIES; ZIMMERMAN, 2015) por exemplo, define *Software Analytics* como: "A análise de dados de software para gerentes e engenheiros de software, com o objetivo de capacitar indivíduos e times de desenvolvimento, a ganhar e difundir conhecimento a partir de seus dados para tomar melhores decisões". Essa atividade de análise ajuda a usuários, desenvolvedores e gerentes a responderem questões de introspecção, por exemplo, 'O por que e como um determinado evento ocorreu'. A utilização de *Software Analytics* permite esse tipo de introspecção pois ao invés de propor e considerar a análise de métricas e dados independentemente, ela propõe diferentes tipos de análise em diferentes camadas de documentos, de forma a filtrar, resumir, modelar e realizar experimentações que permitam um melhor entendimento do que está acontecendo dentro do projeto(BUSE; ZIMMERMANN, 2012).

Hoje é comum empresas como Google, Facebook e Microsoft aplicarem métodos de análise de dados diariamente em seus projetos e produtos. Além disso, o número de conferências interessadas nos assuntos de mineração e análise de artefatos de software cresceu, destacando-se duas, a*Mining Software Repositories* (MSR) e a *PROMISE Conference on Repeatable Experimentes in Software Engineering*. Cada uma possui um foco diferente, sendo a MSR preocupada com a coleta dos dados, enquanto que a PROMISE com a eficácia e repetibilidade da análise de dados.

Diversos autores propõem soluções que utilizam técnicas de *Software Analytics* para ajudar na tomada de decisões em projetos de software. Czerwonka, por exemplo, propôs a plataforma de análise *CODEMINE* depois de observar *inputs*, *outputs* e dados de ferramentas de diversos times na Microsoft. O *CODEMINE* fornece informações de diversos artefatos de software, entre eles: *milestones*, código fonte, *bugtrac-*

kers e outros o que permite que novas pesquisas sejam desenvolvidas neste contexto (ABDELLATIF; CAPRETZ; HO, 2015). Já Baysal, propôs a utilização de um *dashboard* como uma ferramenta complementar ao *Bugzilla* para os times de desenvolvimento da Mozilla. Desta forma, os desenvolvedores podiam ter informações mais detalhadas a respeito de *patches*, comentários e *bugs* reportados, auxiliando na tomada de decisões no dia-a-dia (ABDELLATIF; CAPRETZ; HO, 2015).

2.1.1 Projetos de Software Livre e Software Analytics

A análise de dados obtidos de ferramentas de versionamento de código, como o Git, Bazaar, Subversion ou CVS pode trazer informações importantes a respeito da evolução de um projeto de software, por exemplo: o nível de engajamento dos desenvolvedores; o número de defeitos; a qualidade interna do produto; resultados de execução de testes, entre outros(BIRD; MENZIES; ZIMMERMAN, 2015).

O Github é o maior repositório de software do mundo. Além dos serviços de gerenciamento de código padrões presentes em um sistema baseado em git(*push, commit, pull, clone, fork*), o Github também fornece uma maneira dos desenvolvedores de um projeto interagirem através de *features* de uma rede social. Por exemplo, um desenvolvedor que aprecia um projeto pode marca-lo com uma estrela, de forma que as estrelas de um projeto são um indicador de sua popularidade. Alguns dos projetos mais populares hoje no Github são: JQUERY/JQUERY que é uma biblioteca de *scripting* para HTML, TORVALDS/Linux o projeto do *kernel* do Linux, RAILS/RAILS um framework para a linguagem Ruby e o DOCKER/DOCKER que é um motor de contêineres para aplicações (BORGES; HORA; VALENTE, 2016).

Devida a alta popularidade do Github, ele é largamente usado para a realização de pesquisas e mineração de dados. Por exemplo, Zho et al. mostra que a adoção de pastas com nomes padrão (*doc, test, examples*) em projetos pode ter um impacto na popularidade do projeto. Já em outro estudo Aggarwal et al. mostra que projetos com uma alta popularidade atraem mais pessoas que queiram contribuir com a *Wiki* e documentação (BORGES; HORA; VALENTE, 2016).

Para que fosse possível a realização destes estudos foi necessário que os repositórios fossem abertos, e que fosse utilizada a API do Github. A API fornecida pelo Github é um ponto de acesso comum por onde é possível extrair informações de repositórios públicos, essas informações podem estar divididas em dois grandes grupos:

- **Metadados:** Os metadados fornecidos são informações associadas a cada *commit* ou *issue*, são: criador(a), data, mensagem do *commit* ou *issue*, a *branch*, o repositório, e o escopo do *commit* ou *issue*. Além disso, na mensagem do *commit* podem haver referencias a outros desenvolvedores, defeitos ou outros *commits* e *issues*.

- **Snapshots:** O código fonte do projeto, que pode ser obtido em diferentes fases do desenvolvimento, já que a ferramente permite que um usuário avance ou retroceda dentro dos *commits*. Desta forma, é possível analisar o código de um projeto referente em vários períodos diferentes.

Abaixo, na Figura 1, podemos ver o digrama de classes da API fornecida pelo Github, com base nesta imagem, é possível ver que as *issues* e informações dos *commiters* são exportadas a partir de um ponto comum de acesso.

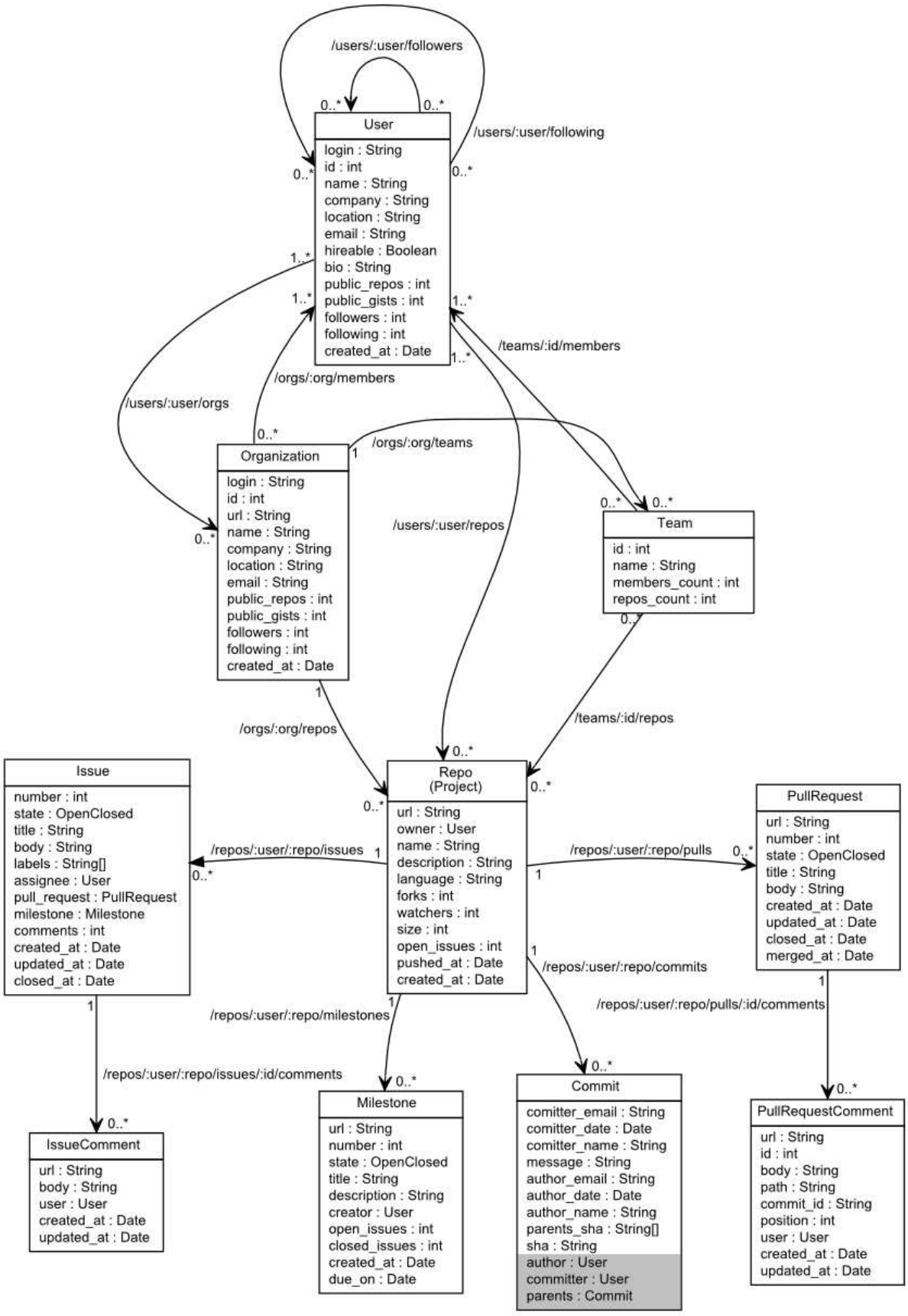

Figura 1 – Diagrama do Esquema do Github

2.2 Centralidade de Redes e o Ranqueamento de Páginas

Centralidade é um dos tópicos mais estudados na teoria dos grafos, sendo um tópico de grande importância em estudos de redes sociais (NEWMAN, 2010). Ela determina o grau de importância de um vértice dentre todos os outros dentro de uma rede. Na Figura 2 é ilustrado um exemplo de centralidade. O conceito de centralidade de redes já foi aplicado a diversos contextos, dentre eles: investigar a influencia de redes inter organizacionais, estudos de relevância, vantagens em redes de troca, competência em organizações formais, oportunidades de emprego e diversos outros campos do mercado e ciência(BORGATTI; EVERETT, 2006).

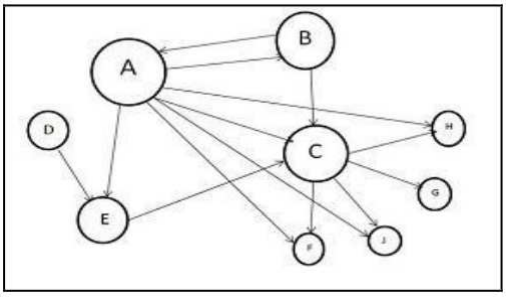

Figura 2 – Exemplo de Relevância de um Nó em uma Rede, extraída de (MUPPIDI; KORAGANJI, 2016).

Diversos processos comumente encontrados no dia-a-dia são caracterizados como um processo de fluxo, como por exemplo o envio de e-mails, uma rede social acessada por milhares de usuários e a distribuição de dinheiro e informação entre pessoas. Esses processos diferem em dimensão e tipo, mas ainda assim podem ser comparados e exemplificados. Considerando que os processos de troca de e-mails, troca de informações, uma doença infecciosa que está sendo transmitida em uma população, a distribuição de pacotes e a troca de moedas representam um fluxo, então podemos descrevê-los sob a perspectiva da Teoria dos Grafos(BORGATTI, 2005). Por exemplo:

- Dinheiro: Considere uma moeda, ou nota de um dólar, que se move pela economia e muda de mãos a cada transação. Uma nota é indivisível e só pode estar em um lugar em um determinado período de tempo. Na perspectiva da teoria dos grafos, a movimentação de uma nota em uma rede pode ser feita como um *passeio (Walk)*, sendo, desta forma, representado como um processo de Markov (BORGATTI, 2005).

- Fofoca: Imagine uma informação privada que passa por uma rede de empregados de uma empresa. Diferentemente de uma moeda, essa mesma informação pode estar em diversos locais diferentes em um determinado período de tempo, se replicando a cada pessoa que passa a ter acesso a informação, mas geralmente não atravessa o mesmo vértice mais de uma vez, apesar de poder visitar o mesmo nó mais de uma

vez. A movimentação desta informação em grafo pode ser representada como uma trilha[1] (BORGATTI, 2005).

- E-mail: Um vírus, ou spam de e-mail, que envia mensagens a todos os contatos de uma pessoa simultaneamente pode ser considerado um processo de fluxo. Um e-mail pode ser representado como uma trilha da mesma forma que é realizado com uma informação, já que compartilha propriedades semelhantes, como existir em diversos locais ao mesmo tempo, e geralmente, os e-mails não passam pelo mesmo vértice, ou caminho, duas vezes (BORGATTI, 2005).

- Infecções: O caso de uma infecção em que o hospedeiro ficou imune. A infecção vai passar por duplicação de pessoa a pessoa, mas nunca vai voltar a infectar aqueles que já se tornaram imunes (BORGATTI, 2005).

- Pacotes: Um pacote de rede tem a característica de possuir um remetente e um destinatário. Normalmente é preferível que o pacote percorra o menor caminho possível até chegar ao seu destino. Dessa forma, a movimentação do pacote deve seguir o caminho geodésico[2] pela rede de roteadores (BORGATTI, 2005).

A partir destas ideias discutidas a respeito de centralidade, pretende-se modelar as *issues* de um repositório de código como um processo de fluxo. Uma *issue* em repositório é uma informação única que pode ser associada a outras *issues*, pessoas ou trechos específicos de código. Sendo assim, as informações contidas em *issue* se mantem a mesma, porém com a característica de poder influenciar o planejamento e o desenvolvimento do projeto.

Ao longo dos anos, diferentes métodos de medida de centralidade foram criados, entre eles: Centralidade de Grau, Proximidade *(Closeness)*, Intermediação *(Betweenness)*, Autovetor *(Eigen Vector)*, Informação, Intermediação de Fluxo, *Rush Index*, Medidas de Influencia de Katz, Hubbell, Hoede e Taylor (BORGATTI; EVERETT, 2006). Estes métodos diferem nas suposições iniciais que fazem para alcançar o objetivo de determinar o nó com maior importância, por exemplo, o algoritmo de Freeman de proximidade e intermediação conta apenas os caminhos geodésicos entre os nós, assumindo que qualquer fluxo que percorra uma rede, siga sempre pelo caminho mais curto. Outros algoritmos como a centralidade de autovetor de Bonacich conta travessias, o que assume que as trajetórias podem ser sinuosas, atravessando nós e arestas repetidas vezes até chegar ao destino (BONACICH, 1987).

A partir de todos estes métodos discutidos por (BORGATTI; EVERETT, 2006) e (BONACICH, 1987), foi escolhido um algoritmo para ser aplicado no exemplo de

[1] Em um grafo, uma trilha(*trail*) é um passeio (*Walk*) que não possui arestas repetidas

[2] Um caminho geodésico em um grafo é composto pelas arestas que representam o menor caminho entre dois vértices.

uso realizado nesta primeira etapa da monografia. O algoritmo escolhido foi o *Page Ranking*, ou algoritmo de ranqueamento de paginas, a escolha deste algoritmo deu-se ao fato de ser um dos algoritmos mais utilizados no calculo de relevância e ser largamente estudado (BRIN; PAGE, 2012), o que inclui pesquisas como a realizada por (MUPPIDI; KORAGANJI, 2016) que realiza uma comparação entre os algoritmos de ranqueamento contemporâneos. A próxima seção deste capítulo apresenta a fundamentação teórica a respeito do algoritmo de ranqueamento de páginas

2.2.1 Ranqueamento de Páginas

O algoritmo de ranqueamento de páginas foi criado por Larry Page e Sergey em 1996. Ele permite calcular a importância relativa de páginas da web e tem aplicações em mecanismos de busca, estimativas de trafego e navegação na web. O algoritmo foi criado com o intuito de prover uma solução de busca de informações na web a partir da estrutura dos links, usada no hipertexto da web. (PAGE et al., 1999).

A premissa do algoritmo de ranqueamento de páginas é que cada página na web possui um número de *links* de saída e de entrada, e que páginas com um grande número de *links* são mais importantes do que as com poucos *links*. Além disso, o algoritmo também leva em consideração a relevância dos *links* de entrada de uma página, ou seja, se uma página da web possui um *link* de entrada que possui uma alta relevância, essa página tende a ser mais importante que outra que possua vários *links* mas que vieram de páginas menos relevantes.

A função de ranqueamento de páginas é uma função recursiva definida na seguinte forma:

Seja x uma página da web. Então:

- $PR(x)$ *Page rank* da página x
- $PR(y)$ *Page rank* da página y
- $L(x)$ é o conjunto de páginas que possuem link para x
- $C(y)$ é o grau de saída de y
- α é a probabilidade de um pulo aleatório
- N é o número total de web-sites

$$PR(x) := \alpha \left(\frac{1}{N}\right) + (1 - \alpha) \sum_{y \in L(x)} \frac{PR(y)}{C(y)}$$

A função de ranqueamento de páginas não ranqueia um *web-site* por completo, mas páginas individuais de cada *web-site*. O *page rank* de uma página y que possui *links* para x também não influencia o *page rank* de x uniformemente. No algoritmo de ranqueamento de páginas o *page rank* de uma página y é balanceado com base no grau de saída [3] $C(y)$ na página y, de forma que quanto maior o número de *links* de saída da página y menos a página x vai se beneficiar disto. Depois deste passo, é realizado o somatório do *page rank* das páginas y, desta forma, cada link de entrada para a página x aumenta seu *page rank* (PAGE et al., 1999).

Depois deste primeiro passo, a soma do *page rank* $PR(y)$ é multiplicada por um fator de amortecimento α. Este fator representa a probabilidade de um pulo aleatório em uma determinada página, ou seja, a probabilidade de que uma pessoa que esteja surfando na *web*, e esteja em uma pagina y clique em qualquer um dos *links* desta página. A probabilidade de que uma pessoa clique em um dos *links* de uma página é puramente baseada no número de *links* da página, e a probabilidade que ela chegue em uma página específica é a soma das probabilidades de cada *link* que a pessoa teve que seguir para chegar a ela. Por se tratar de uma probabilidade, o α assume valores entre 0 e 1, porém para muitos casos é sugerido o uso de 0.85 (PAGE et al., 1999).

Na Figura 3 é apresentado um exemplo como a relevância de uma página influencia as páginas seguintes.

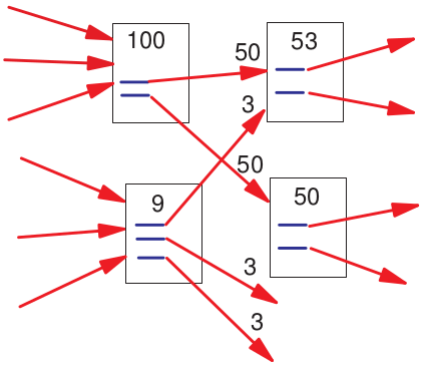

Figura 3 – Simplificação do Cálculo de Ranqueamento de Páginas, extraída de (PAGE et al., 1999).

A Figura 3 ilustra como o ranqueamento de páginas se divide através das paginas e como ele contribui para o ranque das páginas seguintes. Na Figura 3 é possível ver como uma página que possui um *page rank* acumulado de 100 distribui seu ranque igualmente para as paginas seguintes, de forma que cada um dos seus dois *links* receba 50 de ranque. Já em umá página que possui 9 de ranque acumulado, distribui igualmente 3 de ranque para cada um dos seus 3 *links* subsequentes. Na Figura 4 é ilustrado como um estado estável

[3] O grau de saída de um vértice em um grafo direcionado é igual ao número de caudas (*tails*) incidentes ao vértice.

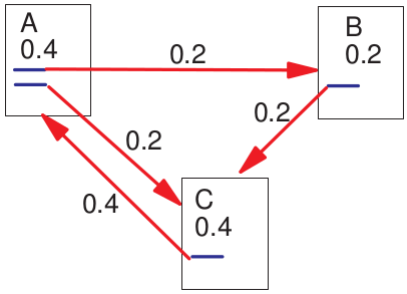

Figura 4 – Simplificação do Cálculo de Ranqueamento de Páginas, extraída de (PAGE et al., 1999).

pode ser alcançado depois que o algoritmo é executado em um conjunto de páginas, de forma que o ranque de uma página seja distribuído igualmente para as páginas posteriores a ela.

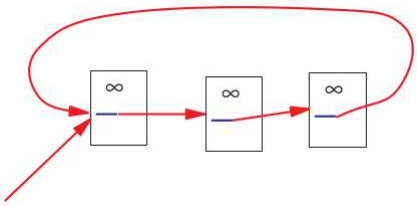

Figura 5 – *Loop* de Ranqueamento, extraída de (PAGE et al., 1999).

Um dos problemas dessa função de ranqueamento, é que caso duas páginas da web que apontem uma para outra, e uma terceira aponte para qualquer uma destas, o *loop* vai acumular ranque porém, nunca vai distribuir isso para nenhuma outra página seguinte. Na ilustração apresentada na Figura 5, possível observar esse fenômeno.

2.3 Considerações Finais do Capítulo

Neste capítulo foi apresentado o referencial teórico a respeito dos tópicos de *Software Analytics*, Centralidade e o algoritmo *Page Ranking*. No próximo capítulo é apresentada a metodologia utilizada neste estudo, passando primeiramente pela definição da população de estudo, depois as técnicas de intervenção que devem ser aplicadas e, finalmente, como a análise dos dados deve ser feita ao final desta primeira etapa da monografia.

3 Metodologia

A utilização de experimentação é relativamente recente na área de Engenharia de Software. Um dos trabalhos seminais nessa área foi proposto por Kitchenham e Charters (2007b), e teve o objetivo de estabelecer um guia para se realizar revisões sistemáticas da literatura. Nesta monografia foram utilizados alguns elementos da estrutura do protocolo proposto por Kitchenham e Charters (2007b) de forma a sistematizar a busca por estudos que orientassem esse trabalho. Um dos elementos previstos no protocolo, e que foi utilizado nesta monografia, é a composição de uma *string* de busca em bases digitais, originalmente proposto na área de medicina, o PICO. (PAI et al., 2004) Trata-se de uma abstração proposta para agrupar conjuntos de termos e auxiliar a composição da *string* de busca a ser submetida às bases. Segundo a adaptação proposta, o PICO procura organizar:

- *Population* (População): A população que o estudo será aplicado, no contexto de engenharia de software pode ser: uma determinada função (Desenvolvedores, Gerentes, Usuários), categoria de engenheiro de software (Iniciante, Junior, Senior), uma área de aplicação, ou um setor da industria.
- *Intervention* (Intervenção): É a metodologia, ferramenta, tecnologia ou procedimento aplicado no estudo.
- *Comparison* (Comparação): É a metodologia, ferramenta, tecnologia ou procedimento com o qual a intervenção está sendo comparada.
- *Outcome* (Resultado): Resultados da aplicação da intervenção na população.

Considerando os objetivos e as questões de pesquisa apresentadas, na introdução desta monografia, a estrutura do PICO para este trabalho foi assim definida:

- **População:** Repositórios de software-livre.
- **Intervention (Intervenção):** Busca, mineração e análise de dados com utilização do algoritmo de *Page Ranking* em *issues* de repositórios de código.
- **Comparação:** Não se aplica a este estudo pois o exemplo de uso construído não visa a comparação com alguma técnica já conhecida de priorização de atividades.
- **Resultado:** Determinar a eficiência da utilização de algumas das técnicas de *Software Analytics* de forma a apoiar decisões de planejamento de *release*.

3.1 Definição da População

A população deste estudo foi definida como repositórios de software-livre, mais especificamente aqueles repositórios hospedados na plataforma Github. Como foi discutido na Seção 2.1.1 esta plataforma, apesar de não ser livre, é a mais popular do mundo e hoje hospeda cerca de 38 milhões de projetos de software. Além disto, outros estudos como o realizado por Borges, Hora e Valente (2016) mostram como essa plataforma pode ser utilizada para a busca e mineração de informação.

Além da popularidade dominante do Github em relação a outras plataformas baseadas em Git, outro dos fatores que direcionaram a escolha desta ferramenta como população é a existência da API também discutida na Seção 2.1.1. Através desta solução é possível extrair informações diversas dos repositórios no Github, entre elas podemos citar: *commits*, contribuidores, comentários, trechos de código, *wiki*, *milestones* e *issues*. Esta API também torna possível a extração de dados do repositório como número de estrelas concedidas por outros desenvolvedores, número de *forks* e qual a linguagem em que o projeto está sendo desenvolvido.

Como a API do Github pode fornecer informações de *issues*, *milestones* e da *wiki* de um repositório, e esta monografia deve se limitar a analisar estas informações, ela deve ser utilizada para isto. Uma observação que é necessário fazer é que a API tem a capacidade de retornar no máximo 100 eventos por requisição, ou página, e possui um limite e 400 páginas, desta forma deverão ser analisados repositório que possuam um máximo de 40000 *issues*, sendo que cada *issue* deve possuir um número máximo de 40000 comentários.

3.2 Método de Intervenção

A intervenção é a busca, mineração e análise das *issues* dos repositórios. Para a realização desta tarefa foi selecionada a linguagem de programação Python.O Python é uma linguagem de programação dinâmica, interpretada, de alto nível e de uso geral criada por Guido Van Rossum. Esta linguagem possui estruturas de dados de alto nível, é dinamicamente tipada e é muito atrativa para o desenvolvimento rápido de aplicações e scripts (PYTHON, 2016).

O Python foi escolhido como linguagem principal para o desenvolvimento da solução proposta pela quantidade de bibliotecas e ferramentas disponíveis para trabalhar com requisições web, busca e análise textual e criação, análise e visualização de grafos. A utilização destas ferramentas tem o objetivo de facilitar as análises realizadas neste estudo de forma a reduzir o trabalho de implementação de algoritmos que já são extremamente usados e estudados. Dentre as bibliotecas pesquisadas, as principais e que foram utilizadas

neste trabalho foram:

- **Jellyfish:** Biblioteca Python para a realização de correspondência aproximada de textos de palavras, podendo realizar a comparação entre distancia de palavras com utilização de algoritmos como os de: Levenshtein, Jaro-Winkler e Hamming.
- **NetworkX:** Pacote Python para a criação, manipulação e estudo de estruturas, dinâmicas e funções de redes complexas. Pode ser utilizada para geração de grafos randômicos e sintéticos e possui diversos algoritmos para análise de redes como o algoritmo de ranqueamento de páginas e o Hits.
- **NLTK:** Abreviação para *Natural Languagem Toolkit*, é uma plataforma para a construção de programas Python que trabalhem com dados compostos por linguagem humana. Possui diversas ferramentas para processamento de texto em linguagens naturais, oferecendo suporte para o português e o inglês.

É importante citar que todas essas bibliotecas e ferramentas utilizadas são livres, de forma que o código fonte de cada uma delas pode ser baixado, executado e alterado para a utilização neste estudo.

Utilizando estas ferramentas foi criado o *script* descrito no apêndice A. Este *script* realiza a função de buscar no repositório definido as *issues* e os comentários referentes a elas. Depois da execução deste primeiro passo, é realizada uma análise textual para determinar se uma *issue* foi mencionada nos comentários de outra. Desta forma, foi gerado um grafo direcionado onde para cada marcação que uma *issue* possui era adicionada uma aresta para ela. Assim que o grafo foi gerado, utilizou-se o algoritmo de ranqueamento de páginas para determinar quais vértices, representados pelas *issues*, são os mais revelantes.

3.3 Analise dos Resultados

Nesta primeira etapa do trabalho a analise dos resultados foi realizada manualmente, porém espera-se que para a etapa seguinte seja possível realizar esta analise de forma automatizada utilizando a biblioteca *Jellyfish* apresentada na Seção 3.2.

Para que fosse possível analisar os dados gerados pela intervenção foi feita a comparação manual da relevância das *issues* analisadas e o planejamento de *release* presente na *Wiki* do projeto escolhido. Isso foi possível pois o projeto do Software Público Brasileiro, ou SPB, possui a característica de ter organizado todo o planejamento de suas duas últimas *releases*, ou seja a *release* 4 e 5, inteiramente dentro do repositório de código.

Sendo assim, foi possível alinhar o texto das *issues* mais relevantes com as *features* consideradas estratégicas para estas *releases* e desta forma determinar se a intervenção teve resultados satisfatórios.

3.4 Considerações Finais do Capitulo

Neste capitulo foram apresentadas as principais estruturas, técnicas e ferramentas utilizados para a elaboração desta monografia. Primeiramente foi apresentada a metodologia de pesquisa e a identificação da população de pesquisa, o método de intervenção e de análise de resultados. Logo após cada um destes pontos é discutido, de forma a justificar a escolha dos repositórios do Github como população, a utilização de busca, mineração e análise dos dados dos repositórios utilizando o Python, e como a análise dos resultados foi feita nesta primeira etapa do trabalho. No próximo capitulo será apresentado como estas técnicas foram utilizadas para que fosse possível executar o exemplo de uso que faz parte desta monografia e alguns resultados colhidos a partir deste exemplo.

4 Ranqueamento de *Issues* no SPB

4.1 O Portal do Software Público Brasileiro

O portal do Software Público Brasileiro (SPB) é um espaço virtual, criado em 12 de abril de 2007, que agrega um conjunto de softwares desenvolvidos nas áreas de saúde, educação, saneamento, gestão de TI, Geoprocessamento, entre outras. Os serviços do SPB são acessados não só no Brasil, mas também por outros países, como por exemplo: Uruguai, Argentina, Portugal, Venezuela, Chile e Paraguai. A politica de compartilhamento de software resulta em uma gestão mais racionalizada de recursos e gastos em Tecnologia da Informação por parte da Administração Pública Federal. Isso possibilita também, a ampliação de parcerias e reforço da politica de software livre no setor público (SPB, 2015).

O novo Portal do Software Público Brasileiro é composto de:

- Lista de discussão (Mailman).
- Plataforma de redes sociais com blog, e-Portifólios, RSS, discussão temática, agenda de eventos, galeria de imagens, e demais funcionalidades de um CMS (Noosfero).
- Sistema de controle de versão, repositório de código-fonte e ambiente desenvolvimento colaborativo (GitLab).
- Autenticação única, busca e integração de ferramentas a fim de tornar a navegação intuitiva e transparente entre as diversas ferramentas que compõem o novo Portal (Colab).

Este ambiente possui uma arquitetura de sete máquinas com funções distintas que são fundamentais para que estes serviços sejam fornecidos. Um destes ambientes, chamado de *reverse proxy*, ou proxy reverso, trata todas as requisições HTTP e as direciona para uma segunda máquina onde as requisições são distribuidas para os ambientes que fornecem o serviço solicitado (DOCS, 2014). A Figura 6 mostra como estes ambientes se comunicam e como são distribuídos os serviços em cada uma das máquinas.

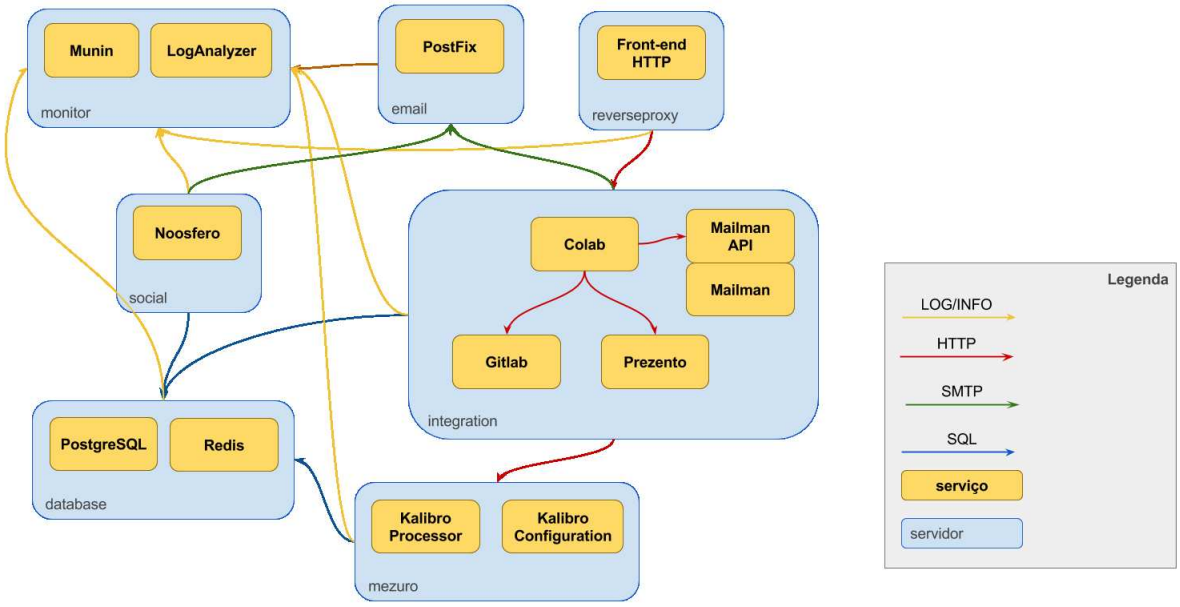

Figura 6 – Diagrama da Arquitetura do SPB, extraída de (DOCS, 2014)

- **Reverse Proxy:** Proxy reverso.
- **Integration:** Segundo proxy reverso, repositorio git, listas de email e *backend* final de e-mail.
- **Email:** Relay de e-mail.
- **Social:** Servidor da rede social Noosfero.
- **Database:** Servidor de banco de dados PostgreSQL
- **Mezuro:** Servidor de análise de código Mezuro.
- **Monitor:** Monitoramento e informações dos outros serviços.

4.2 Organização do Trabalho No SPB

O Software Público Brasileiro foi desenvolvido com base no *Scrum*. O *Scrum* é um método para gerenciar o desenvolvimento e manutenção de produtos, criado em meados da década de 1990 por (SCHWABER; SUTHERLAND, 2001). O *Scrum* é flexível o suficiente para permitir que outros processos, métodos ou práticas possam ser empregados conjuntamente com ele.

Como métodos de desenvolvimento do SPB, foram utilizadas adaptações do *Scrum* e do XP, além de práticas de *DevOps*, mantendo contudo os valores prescritos no manifesto ágil. Sendo assim, o projeto foi dividido em três níveis, o nível operacional, o nível tático e

o nível estratégico. Dentro do nível estratégico são definidas as metas estratégicas de cada *release*, o período de duração, e as épicas de negocio. Dentro do nível tático é onde existe a interação entre os responsáveis pelo projeto na UnB e os representantes do MPOG, neste nível que são definidos quais as *features* que deverão ser implementadas na próxima *release* e a priorização das mesmas. A medida que desce-se o nível a partir das metas estratégicas e do planejamento tático e nos aproximamos do nível operacional, o grau de granularidade aumenta até chegar a micro atividades, ou seja ao nível de tarefas diárias executadas pelos desenvolvedores do projeto. Podemos ver essa divisão ocorrendo nas Figuras 7 e 8 abaixo, que mostram como essa quebra é feita e como isso acontece na *Wiki* do projeto.

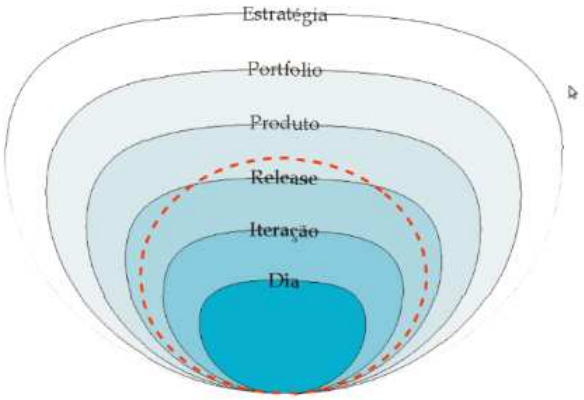

Figura 7 – Diagrama Dos Níveis de Projeto do SPB, extraído de *SPB - RIO Info*

Metas Estratégicas

- Sustentação do novo SPB pelo MP
- Melhorias na usabilidade e da visão do usuário na integração das ferramentas
- Interação do novo SPB com outras plataformas
- Acompanhar o processo de entrada e evolução da qualidade do projetos SPB

Épicas de Negócio

1. Integração dos perfis de usuários

- Definir sistema de widgets do colab

2. Evolução da busca global integrado com o núcleo do Portal

- [x] Estudo e definição das informações em ordem de prioridade/relevância
- [x] Organização dos conteúdos listados na busca
- [x] Design da Busca Global
- [x] Conteúdos das Listas (já contemplado)
- [x] Conteúdos do Noosfero (objetivo na R5)
- [x] Flexibilizar os blocos de busca para os plugins
- [x] Generalizar os filtros no core do colab
- [x] Desenvolver nova interface da página de busca
- Aplicação do design visual
  - [x]Formatar resultados da busca global (e caixa de filtros)

Figura 8 – Wiki do SPB com as Métas de Estratégicas e Épicas de Negócio

Já dentro do nível operacional existem as histórias de usuário e as atividades. Dentro do SPB, as histórias de usuário são tratadas como *milestones* e as atividades como *issues* do Gitlab. Dentro de uma *sprint* cada time escolhe uma ou mais histórias, do

backlog da *release*, e realiza a divisão de suas atividades entre seus membros. A utilização da ferramenta Gitlab permite que as atividades possam ser comentadas e acompanhadas pelos usuários, facilitando, desta forma, o acompanhamento do projeto tanto por parte do Ministério do Planejamento Orçamento e Gestão e dos desenvolvedores. As imagens 9 e 10 abaixo mostram a estrutura de uma história e de uma atividade.

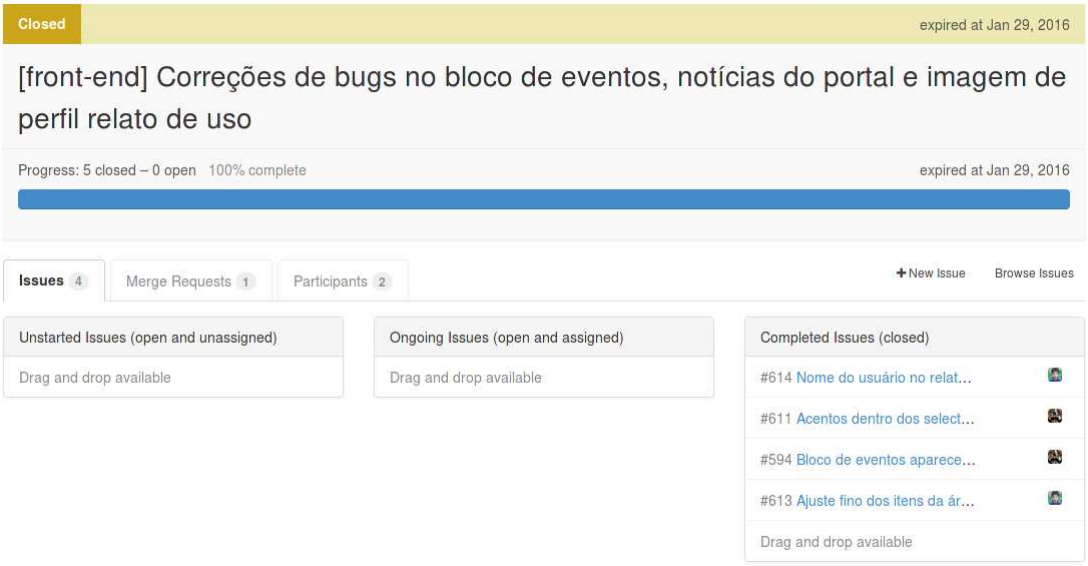

Figura 9 – Estrutura de um *Milestone* no SPB

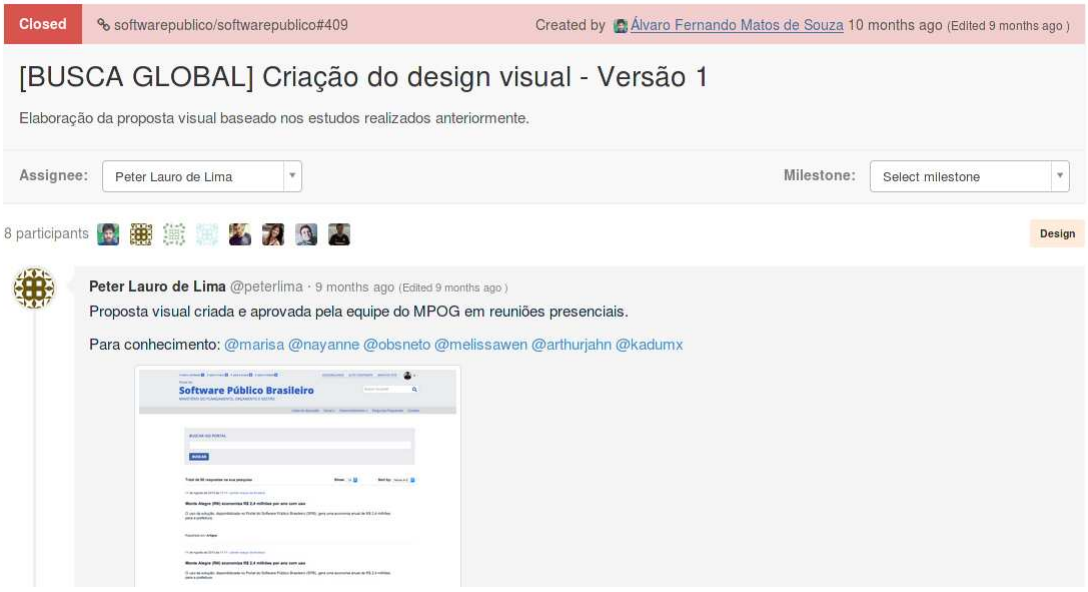

Figura 10 – Estrutura de uma *Issue* no SPB

A escolha do SPB como um caso piloto foi estratégica, visto que é um projeto de domínio público e utilizando o Gitlab, é possível exportar informações de desenvolvedores, atividades, histórias e épicas, além disto é possível extrair informações especificas relacionadas ao código fonte do projeto. Sem este tipo de acesso a informação seria praticamente

impossível a execução deste experimento. Outro fator determinante é a forma com o que o SPB concentra o planejamento de *release* dentro da repositório, o permite com que a relação entre as atividades seja evidenciada mais facilmente.

4.3 Ranqueamento de Páginas no SPB

Essa sessão apresenta como o algoritmo de ranqueamento de páginas descrito no apêndice A foi aplicado ao Software Público Brasileiro de forma a extrair a relevância das *issues* das *releases* 4 e 5. Como foi dita na sessão 4.2, a escolha deste projeto para o estudo de caso foi estratégica de forma a facilitar a execução deste experimento, já que o SPB possui estrutura e condições favoráveis a isso.

4.3.1 Vértices e Arestas

Como foi visto na Seção 2.2.1 a utilização do algoritmo de ranqueamento de páginas depende que inicialmente exista um grafo direcionado.[1]

Para que um grafo do SPB fosse gerado, foi utilizada a API do Gitlab juntamente com um *script Python* para que todas as *issues*, *milestones* e comentários fosse exportado em formato JSON, isto pode ser visto nos métodos *get_repo_issues* e *get_repo_milestones* do apêndice A. Como foi dito, o algoritmo de ranqueamento de páginas precisa de um dígrafo para ser executado, uma das adaptações deste *script* é que em alguns casos os grafos gerados não eram direcionados, e por isto, estes grafos tiveram que ser convertidos em grafos direcionados substituindo cada um dos arcos ou arestas por um arco de entrada e um de saída, este processo pode ser visto nas Figuras 11 e 12.

Figura 11 – *Issues* antes da conversão dos arcos

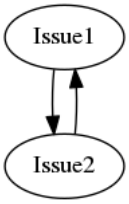

Figura 12 – *Issues* depois da conversão dos arcos

[1] Um grafo direcionado, ou dígrafo, D consiste de um conjunto finito de elementos não vazios $V(D)$ chamados de vértices e um conjunto finito $A(D)$ de pares ordenados de vértices distintos chamados arcos. Desta forma, $V(D)$ é chamado de conjunto de vértices e $A(D)$ o conjunto de arcos de D (BANG-JENSEN; GUTIN, 2008).

4.3.1.1 Issues

Como o foco do estudo é analisar a relevância das *issues*, o dígrafo foi gerado com base nelas com o método *gen_raph* no apêndice A. Dentro do grafo, cada *issue* foi tratada como um vértice, e as arestas eram formadas a medida que uma *issue* era marcada, através de comentários ou eventos, dentro de outra. A Figura 13 exemplifica esse processo.

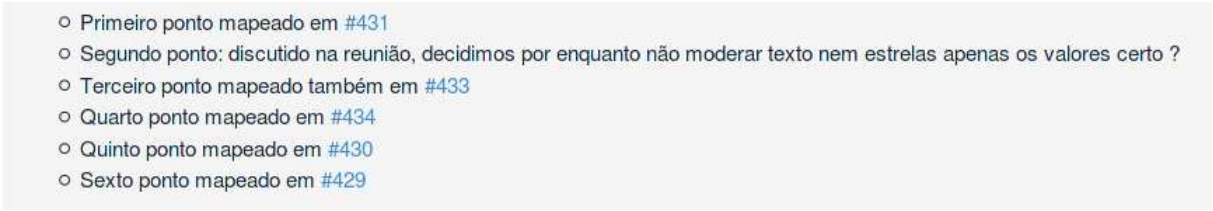

Figura 13 – Exemplo de comentário com marcação de *issue*

Para a realização deste processo foram analisados os comentários de todas as 879 *issues* no repositório do SPB. O dígrafo gerado por esse processo, foi então utilizado para a aplicação do algoritmo de ranqueamento de páginas, a Figura 14 demonstra como esse grafo se apresenta.

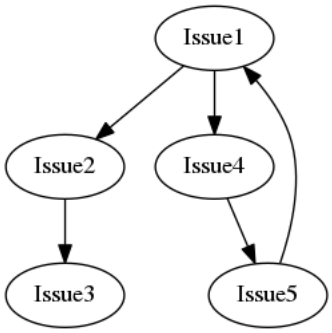

Figura 14 – Demonstração de grafo das *issues*

4.3.1.2 *Milestones*

Dentro do projeto SPB, um *milestone* é uma história com um conjunto de *issues* associadas. Para que fosse possível uma maior aproximação com o ranqueamento de páginas considerou-se que todas as *issues* dentro de um *milestones* estão interligadas, de forma que elas formem um grafo completo[2], essa abstração pode ser vista na Figura 15.

[2] Um grafo G é considerado como completo se cada par de vértices distintos em G são adjacentes (BANG-JENSEN; GUTIN, 2008).

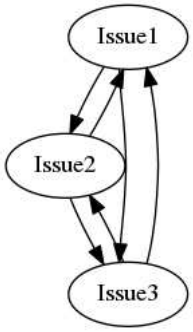

Figura 15 – Demonstração de grafo das *issues* em um mesmo *milestones*

4.4 Resultados

Utilizando-se das abstrações descritas na Seção 4.3 foi possível gerar um grafo representando as *issues* do SPB, esta execução pode ser vista no método *run_pagerank* disponível no apêndice A. Este grafo apresenta as *issues* mais altamente conectadas no centro e as menos conectadas nas bordas. Na Figura 16, é possível perceber que a grande maioria das *issues* tem poucas conexões, ou seja, possuem poucos comentários de marcação, além disto, em alguns casos estas *issues* não estavam associadas a nenhum *milestone*. Esse comportamento pode denotar um baixo índice de comunicação entre a equipe de desenvolvimento e os *stakeholders* no inicio do projeto.

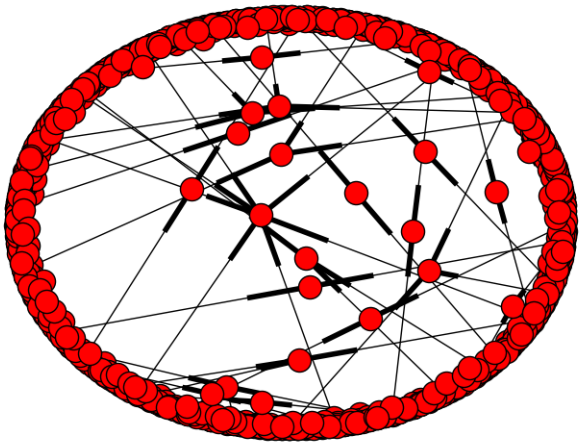

Figura 16 – Grafo das *issues* do SPB

A partir do grafo representado na Figura 16, foi executado o algoritmo de ranqueamento de páginas, o algoritmo utilizado foi o *Page Ranking*, discutido na Seção **??**. Para permitir a execução do algoritmo, foram utilizadas as abstrações para *issues* e *milestones* discutidas nas seções 4.3.1.1 e 4.3.1.2. Após a execução, o algoritmo exporta as *issues* associadas a um valor numérico, de forma que, quanto maior o valor associado a uma

issue, maior o seu ranque relativo. No SPB, as dez *issues* com o maior ranque relativo estão apresentadas na tabela 4.4.

Issues	**Pagerank**
Moderação do valor dos recursos economizados do relato de uso	0.0059332219490603475
Configurar o *NGINX* para servir os dados do *syslog*	0.003453026773943251
Exibir mensagem de erro próxima ao campo instituição no relato de uso	0.003453026773943251
Subir o Gitlab 8.5 usando o pacote feito.	0.003453026773943251
Melhora da busca global	0.003453026773943251
Tela branca em Editar blocos laterais de comunidade	0.003453026773943251
Importar as notícias wiki do portal	0.0030396609114237347
Cadastro de usuário quebrado (username: boscojr)	0.002626295048904218
Adicionar e-mail do usuário na Tela de processar solicitação de Entrar na comunidade.	0.002626295048904218
Retirar pergunta por SISP na criação de uma nova instituição?	0.002626295048904218

Tabela 1 – As dez *issues* melhor ranqueadas do Software Publico Brasileiro

Com base nas *issues* melhores ranqueadas, foi possível perceber que alguns termos aparecem com maior frequência, são eles: **Relato de Uso**, **Busca Global**, **Gitlab** e **SISP**. Outras das *issues* que aparecem com o ranque alto, estão ligadas a atividades de manutenção, sejam elas: correção de *bugs*, melhorias de *front-end* e infraestrutura. Desta forma, aparentemente, as *features* de implementação do relato de uso, da busca global, atualização do Gitlab e algumas das correções de *bugs* foram consideradas de maior importância do que aquelas que apresentaram grau de relevância menor.

Através destes dados foi possível realizar uma comparação manual com o planejamento das *releases* 4 e 5 do SPB. Essa comparação foi feita através da comparação dos termos das *issues* mais bem ranqueadas com o *backlog* presente na *Wiki*. Para essas *releases* algumas da *features* prioritárias são: a implementação do relato de uso de software, evolução da busca global e melhorias gerais, termos comuns aos de muitas das *issues* com ranque alto. Já o termo Gitlab aparece com uma alta relevância devido ao grande esforço dedicado a atualização da plataforma durante a fase de finalização do projeto, mas que devido a finalização antecipada da parceria UnB com o MPOG a nova plataforma foi disponibilizada porém não pôde ser integrada. As Figuras 17 e 18 mostram como essas épicas de negocio estão presentes na *Wiki* do projeto e como foram organizadas.

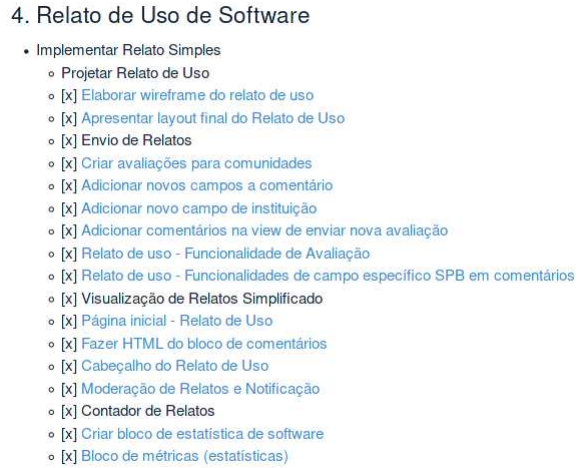

Figura 17 – Épica de Negocio de Relato de Uso

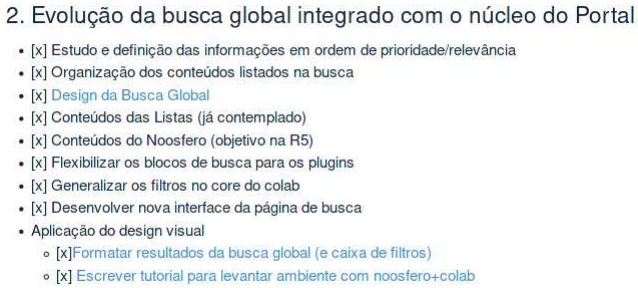

Figura 18 – Épica de Negocio da Melhoria da Busca Global

Outro grande conjunto de *issues* com alto ranque estão ligadas a épica de negocio de melhorias gerais. Indicando que durante parte do desenvolvimento houveram muitas interações entre os *product owners* do MPOG e a equipe de desenvolvedores da UnB e os *seniors* do projeto.

No SPB, as *issues* bem ranqueadas possuem uma correlação de termos com as *features* que demandaram grande parte do esforço do projeto nas *releases* 4 e 5. Desta forma, caso o projeto ainda estivesse em atividade possivelmente as próximas *sprints* poderiam utilizar da aplicação deste ranqueamento para criar um possível *roadmap* e assim validar a solução proposta.

4.5 Considerações Finais do Capítulo

Neste capítulo foram apresentadas as técnicas utilizadas para a busca, mineração e análise utilizadas para o ranqueamento das *issues* do Software Público Brasileiro. Primeiramente foi apresentado o projeto escolhido para o exemplo de uso, depois foram apresentadas os métodos utilizados para a modelagem das *issues* em um grafo direcionado e em seguida foi explicado como o algoritmo de ranqueamento de páginas foi aplicado,

gerando assim resultados preliminares para este trabalho. No próximo capítulo são apresentadas as conclusões preliminares, os trabalhos futuros para a segunda etapa deste trabalho e um cronograma de atividades.

5 Considerações Preliminares e Trabalhos Futuros

Neste trabalho, foi apresentado o projeto de pesquisa a ser realizado e os conceitos fundamentais que o rodeiam, como o problema a resolvido, a questão de pesquisa e a estruturação do exemplo de uso. A primeira etapa deste trabalho apresentou uma revisão bibliográfica a respeito de técnicas e ferramentas utilizadas com *Software Analytics*, aplicações do algoritmo de *Page Ranking*. Além disto, a revisão bibliográfica realizada englobou a atividade de planejamento de *release*, de forma a elucidar como esta atividade é afetada pelas incertezas de um projeto e buscar metodologias propostas por autores diversos em como mitigar este problema.

Os conceitos levantados na revisão bibliográfica foram então utilizados para a implementação de um exemplo de uso, onde foram aplicadas algumas das técnicas de *Software Analytics* e o algoritmo de ranqueamento de páginas com o intuito de relacionar a relevância das *issues* do projeto com o planejamento de *release*. Para a realização desta tarefa foram utilizadas as tecnologias apresentadas na Seção 3 e foi escolhido um projeto de software livre que faz parte da população proposta neste estudo, o projeto utilizado para o exemplo de uso foi o Software Público Brasileiro, e nele, foram analisadas 879 *issues* presentes no repositório do projeto. Utilizando o ranque gerado, foi realizada uma comparação manual das *issues* com a priorização das *features* presentes no repositório do SPB.

A segunda etapa deste trabalho será responsável por prosseguir com o levantamento bibliográfico necessário, bem como estruturar um estudo de caso de forma a validar as proposições feitas nesta primeira etapa. Nesta segunda parte também deverá ser evoluído o método de relacionamento das *issues* e *features* de um repositório, de forma a automatizar como essas informações estão sendo relacionadas. Com isto, será possível validar a solução proposta e com isto responder a questão de pesquisa definida nesta primeira etapa.

5.1 Cronograma

Esta seção contem o cronograma de atividades para o TCC 2. As atividades foram criadas com o objetivo de cumprir com os objetivos propostos neste trabalho e estão passiveis de mudanças nas datas caso algum atraso ou dificuldade seja encontrada durante a realização das atividades.

Atividade	Data
Pesquisa de referencial teórico a respeito dos tópicos de *Big Data*	10/08 a 20/08
Definição de ferramentas de análise e *Big Data*	20/08 a 10/09
Seleção dos repositórios do Github para serem analisados	10/09 a 25/09
Alterações no texto para adequação a um estudo de caso	25/09 a 10/10
Implementação da infraestrutura computacional para a realização do estudo de caso	10/10 a 20/10
Execução do estudo de caso em ambiente de *Big Data*	20/10 a 01/11
Escrita dos Resultados e Considerações Finais do TCC 2	01/11 a 20/11

Tabela 2 – Cronograma Para o TCC 2

Referências

ABDELLATIF, T. M.; CAPRETZ, L. F.; HO, D. Software analytics to software practice: A systematic literature review. In: *2015 IEEE/ACM 1st International Workshop on Big Data Software Engineering*. [S.l.]: IEEE Press Piscataway,USA, New Jersey, 2015. p. 30–36. Citado 2 vezes nas páginas 25 e 26.

BANG-JENSEN, J.; GUTIN, G. Z. *Digraphs: Theory, Algorithms and Applications*. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2008. ISBN 1848009976, 9781848009974. Citado 2 vezes nas páginas 43 e 44.

BECK, K. et al. *Manifesto for Agile Software Development*. Ward Cunningham, 2001. Disponível em: <http://www.agilemanifesto.org/>. Citado na página 21.

BIRD, C.; MENZIES, T.; ZIMMERMAN, T. *The Art and Science Of Analyzing Software Data*. 1 edition. ed. [S.l.]: Morgan Kaufmann, 2015. 672 p. Citado 3 vezes nas páginas 22, 25 e 26.

BONACICH, P. Power and Centrality: A Family of Measures. *American Journal of Sociology*, The University of Chicago Press, v. 92, n. 5, p. 1170–1182, 1987. ISSN 00029602. Disponível em: <http://dx.doi.org/10.2307/2780000>. Citado na página 30.

BORGATTI, S. P. Centrality and network flow. *Social Networks*, v. 27, n. 1, p. 55–71, January 2005. Disponível em: <http://dx.doi.org/10.1016/j.socnet.2004.11.008>. Citado 2 vezes nas páginas 29 e 30.

BORGATTI, S. P.; EVERETT, M. G. A graph-theoretic perspective on centrality. *Social Networks*, v. 28, n. 4, p. 466 – 484, 2006. ISSN 0378-8733. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0378873305000833>. Citado 2 vezes nas páginas 29 e 30.

BORGES, H.; HORA, A. C.; VALENTE, M. T. Predicting the popularity of github repositories. *CoRR*, abs/1607.04342, 2016. Disponível em: <http://arxiv.org/abs/1607.04342>. Citado 2 vezes nas páginas 26 e 36.

BRIN, S.; PAGE, L. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, v. 56, n. 18, p. 3825–3833, 2012. Cited By 43. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84870444130&partnerID=40&md5=e04b61a53166573e4ed640409f3ce27c>. Citado na página 31.

BUSE, R. P. L.; ZIMMERMANN, T. Information needs for software development analytics. In: . University of Michigan: IEEE Press Piscataway, USA, New Jersey, 2012. p. 987–996. ISBN 978-1-4673-1067-3. Citado na página 25.

CZERWONKA, J. et al. Codemine: Building a software development data analytics platform at microsoft. *IEEE Software*, v. 30, n. 4, p. 64–71, July 2013. ISSN 0740-7459. Citado na página 22.

DOCS, S. *SPB Docs*. 2014. Disponível em: <https://softwarepublico.gov.br/doc/arquitetura.html>. Citado 3 vezes nas páginas 13, 39 e 40.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Disponível em: <http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf>. Citado na página 23.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Citado na página 35.

MCDAID, K. et al. Managing uncertainty in agile release planning. In: . [s.n.], 2006. p. 138–143. Cited By 8. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-44149118496&partnerID=40&md5=f2e6f9b72bd79cd165080a770c50ad80>. Citado na página 22.

MUPPIDI, S.; KORAGANJI, V. Survey of contemporary ranking algorithms. *International Journal of Applied Engineering Research*, v. 11, n. 1, p. 322–325, 2016. Cited By 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84959373281&partnerID=40&md5=2384caa578fad0ff936b305a5b79160c>. Citado 3 vezes nas páginas 13, 29 e 31.

NEWMAN, M. *Networks: An Introduction*. New York, NY, USA: Oxford University Press, Inc., 2010. ISBN 0199206651, 9780199206650. Citado na página 29.

NGO-THE, A.; RUHE, G. A systematic approach for solving the wicked problem of software release planning. *Soft Computing*, v. 12, n. 1, p. 95–108, 2008. Cited By 42. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-34548040975&partnerID=40&md5=2cb40dfbfc5063672251c76c3977940c>. Citado 3 vezes nas páginas 21, 22 e 23.

PAGE, L. et al. *The PageRank Citation Ranking: Bringing Order to the Web*. [S.l.], 1999. Previous number = SIDL-WP-1999-0120. Disponível em: <http://ilpubs.stanford.edu:8090/422/>. Citado 4 vezes nas páginas 13, 31, 32 e 33.

PAI, M. et al. Systematic reviews and meta-analyses: an illustrated, step-by-step guide. madhupai@berkeley.edu, v. 17, n. 2, p. 89–95+, 2004. ISSN 0970-258X. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/15141602>. Citado na página 35.

PYTHON. *About Python*. 2016. Disponível em: <https://www.python.org/doc/essays/blurb/>. Citado na página 36.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide*. 2001. Citado na página 40.

SHAW, M. Writing good software engineering research papers: Minitutorial. In: *Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003. (ICSE '03), p. 726–736. ISBN 0-7695-1877-X. Disponível em: <http://dl.acm.org/citation.cfm?id=776816.776925>. Citado na página 23.

SPB. *Sobre o Portal*. 2015. Disponível em: <https://softwarepublico.gov.br/social/spb/sobre-o-portal>. Citado na página 39.

Apêndices

APÊNDICE A – *Script* Para Mineração e Análise das *Issues* do SPB

```
#!/usr/bin/env python3
import os
import json
import nltk
import requests
from requests.packages.urllib3.exceptions import InsecureRequestWarning
import networkx as nx
import matplotlib.pyplot as plt
from collections import OrderedDict

#Disabling insecure warning because of spb certificate problems
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

'''
usage ./page_ranking
'''
def get_repo_issues():
    """
    Make HTTP GET request on Software Publico Gitlab and recover all issues
    on SoftwarePublico/SoftwarePublico repo
    """
    issues_json = []
    if not os.path.isfile('./issues.json'):
        for page_number in range(1, 9):
            issues_json += requests.get('https://www.softwarepublico.gov.br/gitlab/api/v3/'
                                        +'projects/30/issues?page=' + str(page_number) +
                                        '&per_page=100&private_token=pUvEiwMsSgJ2JU7DGiaf',
                                        verify=False).json()
        with open('issues.json', 'w') as outfile:
            json.dump(issues_json, outfile)
    else:
        with open('issues.json', 'r') as infile:
            issues_json = json.load(infile)
    return issues_json

def get_repo_milestones():
```

```
37      """
38      Make HTTP GET request on Software Publico Gitlab and recover all milestones
39      on SoftwarePublico/SoftwarePublico repo
40      """
41      milestones_json = []
42      if not os.path.isfile('./milestones.json'):
43          for page_number in range(1, 2):
44              milestones_json += requests.get('https://www.softwarepublico.gov.br/gitlab/api/v3/'
45                                              +'projects/30/milestones?page=' + str(page_number) +
46                                              '&per_page=100&private_token=pUvEiwMsSgJ2JU7DGiaf',
47                                              verify=False).json()
48          with open('milestones.json', 'w') as outfile:
49              json.dump(milestones_json, outfile)
50      else:
51          with open('milestones.json', 'r') as infile:
52              milestones_json = json.load(infile)
53      return milestones_json
54
55
56  def get_issues_comments(issue_id):
57      """
58      Make HTTP GET request on Software Publico Gitlab and recover all commentaries
59      from one issue
60      """
61      comments_json = []
62      if not os.path.exists('./issues_comments'):
63          os.makedirs('./issues_comments')
64
65      if not os.path.isfile('./issues_comments/issue_'+str(issue_id)+'_comments.json'):
66          comments_json = requests.get('https://www.softwarepublico.gov.br/gitlab/api/v3/'
67                                       +'projects/30/issues/'+str(issue_id)+
68                                       '/notes?&per_page=100&private_token=pUvEiwMsSgJ2JU7DGiaf',
69                                       verify=False).json()
70          with open('./issues_comments/issue_'+str(issue_id)+'_comments.json', 'w') as outfile:
71              json.dump(comments_json, outfile)
72      else:
73          with open('./issues_comments/issue_'+str(issue_id)+'_comments.json', 'r') as infile:
```

```
74              comments_json = json.load(infile)
75      return comments_json
76
77
78  def gen_milestone_dict():
79      """
80      Generates a valid dict with milestone id as key and issues linked as
    value
81      """
82      milestones_dict = {}
83      issues = get_repo_issues()
84      milestones = get_repo_milestones()
85      for milestone in milestones:
86          milestones_dict[milestone['iid']] = []
87          for issue in issues:
88              try:
89                  if issue['milestone']['iid'] == milestone['iid']:
90                      milestones_dict[milestone['iid']].append(issue['iid'])
91              except TypeError:
92                  continue
93      return milestones_dict
94
95  def gen_issue_dict():
96      """
97      Generates a valid dict with issue id as key and issues linked as value
98      """
99      issues = get_repo_issues()
100     #print(issues[1]['milestone']['iid'])
101     issues_dict = {}
102     for issue in issues:
103         issues_dict[issue['iid']] = []
104         comments = get_issues_comments(issue['id'])
105         for comment in comments:
106             comment_tokens = nltk.word_tokenize(comment['body'])
107             for index, word in enumerate(comment_tokens):
108                 if index < (len(comment_tokens) - 1):
109                     if word == '#' and comment_tokens[index+1][:-1].
    isdigit():
110                         if comment_tokens[index+1][-1] == '_':
111                             issues_dict[issue['iid']].append(
    comment_tokens[index+1][:-1])
112     return issues_dict
113
114 def gen_graph():
115     """
116     Generates networkx valid DAG based on the issues dictionary
117     """
```

```
118     graph = nx.DiGraph()
119     issues_dict = gen_issue_dict()
120    # milestones_dict = gen_milestone_dict()
121     for key in issues_dict:
122         graph.add_node(key)
123     for key, value in issues_dict.items():
124         if isinstance(value, list):
125             for node in value:
126                 graph.add_edge(key, node)
127         else:
128             graph.add_edge(key, value)
129 
130    # for key, value in milestones_dict.items():
131    #     graph.add_node(key)
132    #         for item1 in value:
133    #            for item2 in value:
134    #               graph.add_edge(item1, item2)
135     return graph
136 
137 def run_pagerank():
138     """
139     Runs pagerank on all SoftwarePublico/SoftwarePublico issues
140     """
141     graph = gen_graph()
142     page_rank = nx.pagerank(graph)
143     ranking = []
144     for key, value in page_rank.items():
145         ranking.append(value)
146 
147     od = OrderedDict(sorted(page_rank.items(), key=lambda t: t[1]))
148     print(od)
149     nx.draw_spring(graph)
150     plt.show(block=False)
151     plt.savefig("Graph.png", format="PNG")
152     return 0
153 
154 run_pagerank()
```

Listing A.1 – Algoritmo para mineração e análise das *issues* do SPB