Geração de Informações Gerenciais para Sistemas de Controle de Versão: Uma prova de conceito utilizado o GitHub

Gabriel Silva Barbosa, Evaldo de Oliveira da Silva

Curso de Bacharelado em Sistemas de Informação – Centro de Ensino Superior de Juiz de Fora (CESJF) – Campus Academia
36016-000 – Juiz de Fora – MG– Brasil

gabri.barbosa.silva@gmail.com, evaldo.oliveira@gmail.com

Abstract. *This article presents the processing of managerial data information stored by the GitHub tool, recognized as a Version Control System widely used in software maintenance management processes. The paper also discusses techniques and technologies to permit extract, transform, and load data for managerial information extracted from GitHub. Finally, the paper approaches the data model generated from the extraction and transformation process. Based on this data model it was possible to generate a dashboard with management information using the Microsoft Power BI tool, considered a solution applied for data analysis from metrics and interactive graphics.*

Resumo. *Este artigo apresenta a geração de informações gerenciais de dados armazenados pela ferramenta GitHub, reconhecida como um Sistema de Controle de Versão amplamente utilizada nos processos de gestão de manutenção de software. O trabalho também aborda técnicas e tecnologias para permitir a extração, transformação e carga de dados para informações gerenciais extraídas do GitHub. Finalmente, o trabalho aborda o modelo de dados gerado a partir do processo de extração e transformação. A partir deste modelo de dados foi possível a geração de um painel com informações gerenciais utilizando a ferramenta Microsoft Power BI, considerada uma solução aplicada para análise de dados a partir de métricas e gráficos interativos.*

1. Introdução

A engenharia de software tem por objetivo apoiar o desenvolvimento profissional de software. Ela inclui técnicas que apoiam todos os aspectos de produção de software (SOMMERVILLE, 2011). No mundo atual, usar ferramentas que apoiam a engenharia de software é fundamental para se ter um processo de desenvolvimento estável, rastreável e controlável. Essas ferramentas carregam consigo dados que podem ser analisados como forma de extração de métricas e na geração de informações para tomada de decisão.

No contexto de gestão de processos de software não é diferente. Em se tratando da manutenção de software, que é uma atividade reconhecida pela complexidade do controle das modificações (ou transações de modificações), e que necessita de aplicações para o registro das transações que possam responder às seguintes questões: “Quando?”, “Quem?”, “Onde?”, “Por que?” e “Como?”. Ou seja, na execução dos processos de software também existe a necessidade do uso de ferramentas específicas para o registro das atividades (SOMMERVILLE, 2011).

Existem ferramentas de software utilizadas para apoiar a manutenção de software nas áreas de versionamento, controle de mudanças e integração contínua, como por exemplo o Git, Mantis BugTracker e Jenkins respectivamente. Estas ferramentas possuem bases de dados próprias que podem ser úteis para aplicação de métodos para extração de informações úteis para melhoria da gestão de software. Tais informações

podem ser melhor exploradas, a partir dos modelos de dados implementados pelas próprias soluções adotadas para registro das etapas de manutenção de software.

Neste contexto, este trabalho apresenta a aplicação de métodos para extração de dados a fim de gerar informações para gestão da manutenção de software. O artigo apresenta como proposta a utilização de técnicas para extração, transformação de dados, definido na área de inteligência de negócios (ou *Business Intelligence*), para gerar informação gerencial de transações armazenadas no GitHub (GITHUB, 2018). Tais técnicas são definidas no artigo para modelagem e carga de dados gerenciais.

Finalmente, este trabalho pretende contribuir para a gestão da manutenção de software apresentando as técnicas e tecnologias para geração de informações gerenciais a partir do armazenamento de transações de versionamento de repositórios.

O artigo está definido da seguinte forma. A seção 2 descreve os conceitos de Sistemas de Controle de Versão e suas operações básicas, modelagem dimensional de para geração de informações gerenciais e a técnicas de extração, transformação e carga de dados. A seção 3 descreve como os dados do GitHub podem ser extraídos para posterior análise de informações. A seção 4 um exemplo de painel com informações gerenciais extraídas a partir do processo apresentado na Seção 3. Finalizando este artigo, temos a seção 5 contendo as considerações finais e trabalhos futuros.

2. Referencial Teórico

Esta seção descreve os principais conceitos que estão relacionados para fundamentação teórica deste trabalho. Os conceitos discutidos nesta seção tratam do processo de gerenciamento de riscos e detalha suas etapas.

2.1. Sistemas de Controle de Versão

O gerenciamento de versão corresponde a atividades e processos para o controle da evolução de diferentes de componentes ou itens de configuração utilizados durante a manutençãode software (SOMMERVILLE et al., 2011). O controle de versão deve ser apoiado por ferramentas que automatizem e garantam a manutenção das aplicações. Dentre as ferramentas mais conhecidas para o gerenciamento de versão está o GIT. Este tem propriedades bastante interessantes que o levam a ser popular. O GIT considera que os dados são como um conjunto de snapshots (captura de algo em um determinado instante, como em uma foto) de um mini-sistema de arquivos (CHACON E STRAUB, 2014), então o git armazena referências sobre os snapshots. Assim, o sistema de controle de versão não armazena arquivos redundantes, apenas as referências de alterações sobre um arquivo principal. Além disso o GIT tem quase todas as suas operações armazenadas localmente, isso garante que problemas devido à latência de comunicação de uma rede não afetem o desempenho. Tudo no Git tem seu checksum (valor para verificação de integridade) calculado antes que seja armazenado e então passa a ser referenciado pelo checksum. Isso significa que é impossível mudar o conteúdo de qualquer arquivo ou diretório sem que o Git tenha conhecimento (CHACON E STRAUB, 2014). Por fim o sistema de controle de versão permite reversão das alterações, o que confere tranquilidade a equipe de desenvolvimento no tocante a mudanças que comprometam o projeto.

O GitHub é um serviço WEB para armazenamento e colaboração de código que utiliza o GIT para controle de versão. Sendo o maior host único para repositórios Git, é o ponto central de colaboração de milhões de desenvolvedores e projetos. Muitos projetos de código aberto o usam para hospedagem do Git, rastreamento de problemas, revisão de código e outras coisas (CHACON E STRAUB, 2014). A plataforma também funciona como uma rede social para programadores permitindo interação e comunicação.

A maior parte do processo de desenvolvimento de software é uma atividade executada por equipes (SOMMERVILLE et al., 2011), para garantir que diferentes membros da equipe possam trabalhar em conjunto sob o mesmo componente, o sistema de controle de versão adota o seguinte procedimento: os programadores devem efetuar o check-out do código e trabalhar livremente sobre ele em seu espaço privado de trabalho, em seguida eles realizam o check-in para o repositório. Os demais programadores são notificados das alterações naquele componente e devem fazer o check-out sempre que iniciarem seus trabalhos. Informações como data da alteração, autor da mudança e comentários associados são gravados pelo sistema. Este procedimento soluciona o problema de versionamento de código entre desenvolvedores proposto por Ian Sommerville: O desenvolvedor "Alice" está fazendo algumas mudanças em um sistema, o que envolve a mudança dos components A, B e C. Ao mesmo tempo, o desenvolvedor "Bob" está trabalhando em mudanças e estas requerem mudanças nos componentes X,C e Z. Portanto, os desenvolvedores "Alice" e "Bob", estão mudando o componente C (SOMMERVILLE et al., 2011). A Figura 1 apresenta um exemplo deste tipo de gerenciamento com base na referência apresentada.

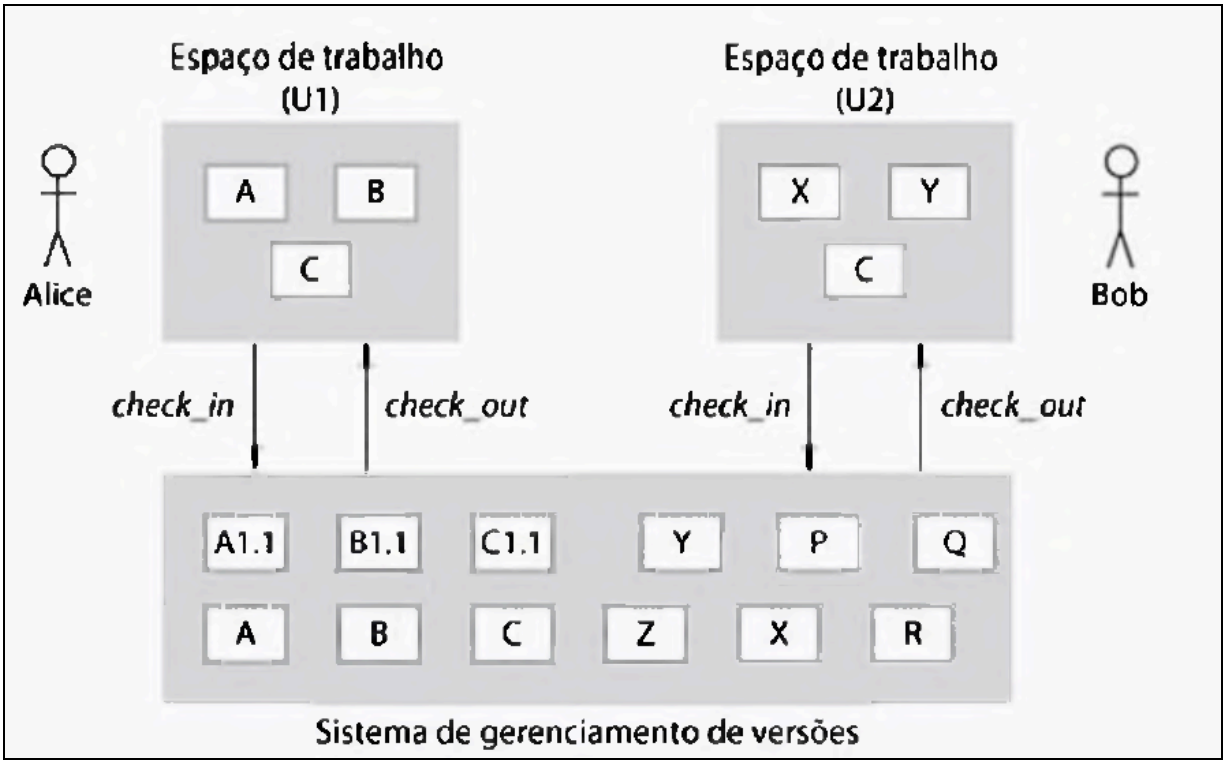

Figura 1. Exemplo de Check-in e Check-out a partir de um repositório de versões. Fonte: SOMMERVILLE (2011).

2.2. Modelagem Dimensional

Os dados mantidos por Sistemas de Informação (SI) dentro das organizações, são considerados fatos que podem ser gravados e que possuem um significado implícito.

Quando os dados corporativos são mantidos e acessados com frequência, torna-se claro sua importância no processo de tomada de decisão, sendo necessário obter o controle de como os dados são gerados e por qual sistema de informação. Porém, a questão de "propriedade" dos dados envolve um aspecto mais amplo, abrangendo uma interpretação mais ampla desta característica. Trata-se de identificar quem pode criar, alterar ou eliminar determinado dado; ou seja, com base no processo de negócio automatizado por um SI, qual subunidade administrativa, setor ou grupo de pessoas,

possui prerrogativas que lhe permitem alterar ou aprovar alterações do estado dos registros, como parte das respectivas funções.

A definição de tais capacidades decorre da natureza dos processos e da estrutura funcional existente na organização e pertence ao domínio da alta administração decidir sobre como os dados devem ser mantidos. Os dados são representações de fatos da organização e do negócio, e a sua manutenção só deve ser determinada pelos usuários que conhecem os processos estabelecidos. O resultado do entendimento de regras de negócios e o processamento destas regras por meio de SIs, resulta na geração de um conjunto de dados que devem estar organizados em aplicações que permitam recuperá-los posteriormente para diversos fins (KIMBALL e ROSS, 2011).

Em uma organização que possui diversas aplicações funcionando em setores distintos, os dados corporativos estão armazenados em tabelas de dados podendo formar um grande banco de dados. Estas tabelas de banco de dados são mantidas por softwares chamados Sistemas Gerenciadores de Banco de Dados (ou SGBDs). Os setores de tecnologia das organizações devem estar cientes da existência e capacidade dos SGBDs e quais destas tecnologias devem atender melhor a organização (ELMASRI e NAVATHE, 2010).

Existem vários SGBDs disponíveis no mercado, podendo ser de uso livre ou proprietário. Alguns exemplos mais populares são: MySQL, SQLServer, Oracle ou IBM DB2. Porém, o propósito deste curso não é aprofundar nos tipos de SGBDs ou até mesmo nas técnicas de como os dados devem ser criados e mantidos.

Assim, o objetivo da discussão sobre banco de dados é permitir que o entendimento de como os dados corporativos podem estar organizados e como tais dados podem ser úteis para geração de dados úteis para tomada de decisão.

A organização de dados corporativos a partir de modelos dimensionais permite prover informações para tomada de decisões a partir de sistemas gerenciais. Tais sistemas devem possuir ferramentas OLAP (do inglês, *Online AnalyticalProcessing*). Essas ferramentas são capazes de navegar pelos dados de um DataWarehouse (DW), possuindo uma estrutura adequada tanto para a realização de pesquisas como para a apresentação de informações (KIMBALL e ROSS, 2011).

A funcionalidade de soluções OLAP é inicialmente caracterizada pela análise dinâmica e multidimensional dos dados consolidados de uma organização permitindo que as atividades do usuário final sejam tanto analíticas quanto navegacionais. Portanto, a tecnologia fornecida pelas ferramentas OLAP permite aos analistas de negócios, gerentes e executivos analisar e visualizar dados corporativos de forma rápida e interativa (KIMBALL e ROSS, 2011).

De acordo com Kimball e Ross (2011) em um modelo dimensional, as tabelas de fatos expressam os diversos relacionamentos entre as dimensões. Por outro lado, as tabelas de dados de dimensões "qualificam" os fatos, com campos descritivos. As dimensões apresentam-se em consultas qualificadas e são definidas "por dimensão", no um contexto de uma base de dados dimensional para uma concessionária, teríamos as dimensões: vendas "por semana", "por marca", "por loja" servindo de base para os agrupamentos de dados.

O DTR abaixo (Figura 2) apresenta um exemplo de modelo dimensional para a situação anteriomente apresentada, o caso de uma concessionária, onde a tabela fato é representada pela "Tabela Fato de Vendas" que armazena a quantidade e média de vendas por meio das dimensões "Modelo do Veículo", "Marca do Veículo", "Tempo" e "Vendedor". Todas as tabelas se relacionam com a Tabela Fato de Venda, que por sua vez armazena dados redundantes das chaves destas. Deste modo podemos gerar uma View e ter informações relevantes como, por exemplo: Quantidade de vendas sumarizada por vendedor e marca de veículo. E isso pode ser associado a tabela de tempo a fim de encontrar essas quantidades por bimestre, trimestre, etc. A tabela de tempo neste contexto possui chaves que remetem a uma data completa (dia/mês/ano),

sendo assim, somente as chaves já trazem tudo que precisamos. Por meio da tabela fato os relatórios de podem ser gerados por meio de uma ferramenta de BI.

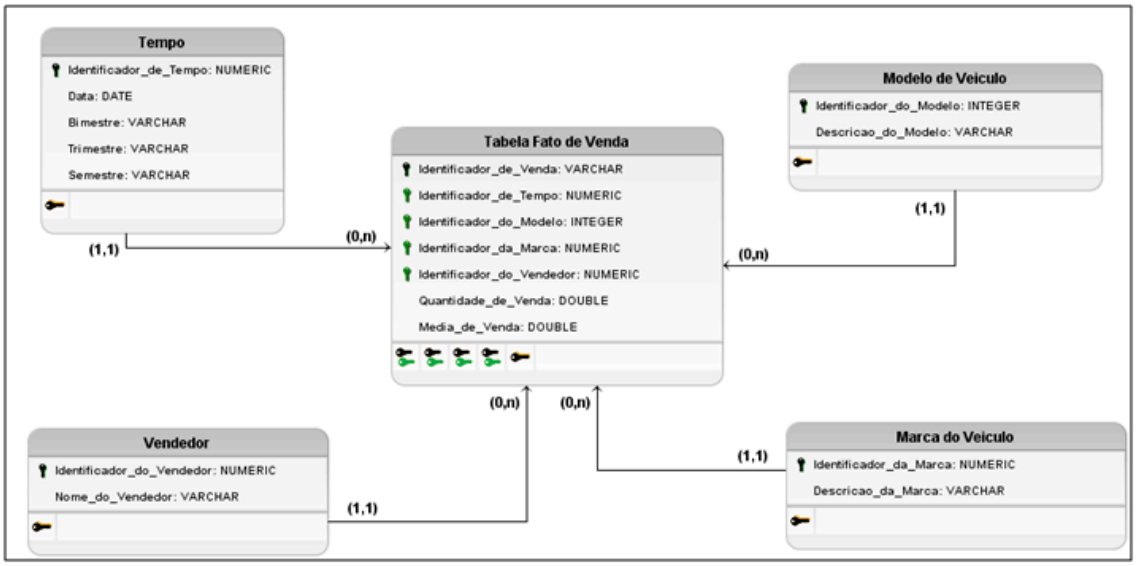

Figura 2. Exemplo de modelo dimensional. Fonte: Elaborado pelo autor (2018).

2.3. Processo de ETL

De acordo com Kimball e Ross (2011), as aplicações para *Business Intellignce* (ou BI) podem ser construídas a partir de um conjunto de técnicas e ferramentas que permitam a organização e análise das informações para o suporte a tomada de decisão, tendo como base a transformação de dados em informações úteis para a gestão de negócios e equipes de trabalho.

Em aplicações de BI utiliza-se técnicas para a aquisição de dados históricos referente aos KPIs (indicadores de performance) estabelecidos para análise e tomada de decisão, possibilitando a geração de um DW. A técnica amplamente utilizada é chamada de ETL (*Extraction, Transform and Load*), que significa extração, transformação e carregamento (KIMBALL e ROSS, 2011).

O ETL é um processo que envolve as seguintes tarefas: a) Extração de dados de sistemas transacionais ou de arquivamento de origem, que são a principal fonte de dados para o DW, neste ponto é preciso ter ferramentas que auxiliem no processo, já que é inviável uma extração manual; b) transformação dos dados, envolve limpeza, filtragem, validação, aplicação de cálculos e regras de negócios. Os dados devem ser armazenados de forma padronizada e de preferência, estruturados para devidos tratamentos; e c) carga de dados, ocorre em uma base de dados DW ou qualquer outro banco de dados ou aplicativo que armazene dados a serem consultados por *dashboards* (KIMBALL e ROSS, 2011). Abaixo segue uma representação do esquema de ETL (Figura 3).

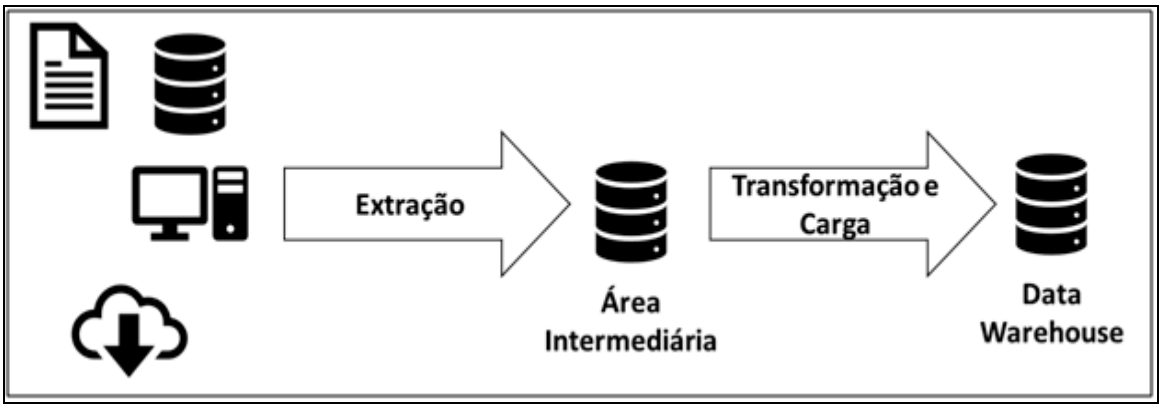

Figura 3. Representação do esquema de ETL. Fonte: Elaborado pelo autor (2018).

O processo de ETL pode ser feito sobre ferramentas utilizadas comumente no dia-a-dia de desenvolvedores e gerentes. É preciso descobrir como esses dados são disponibilizados e a melhor forma de extraí-los.

3. Extração e Transformação de dados utilizando o sistema GitHub

Esta seção descreve as tecnologias e processos utilizados para realizar o processo de ETL para obtenção de informações gerenciais do controle de versionamento de software, tendo como base de dados as informações extraídas do sistema GitHub discutido na seção 2.1. Nesta seção também será detalhado tecnicamente como os dados podem ser extraídos da plataforma do GitHub por meio de bibliotecas e métodos de acesso.

3.1. Serviço REST para extração de dados

A REST(*RepresentationalStateTransfer*) API (*ApplicationProgamming Interface*) v3, é um serviço fornecido pelo GitHub que permite o acesso dos programadores e gerentes a sua base de dados (GITHUB, 2018). Contendo um vasto leque de dados, ela entrega ao usuário da plataforma referências pertinentes sobre repositórios, conta, commits e toda a gama de funcionalidades dispostas pela ferramenta.

A API pode se acessada na WEB de forma simplificada passando parâmetros em sua URL (*UniformResourceLocator*) padrão, por exemplo: https://api.github.com/repos/Nome de Usuário/Nome do Repositório, com estes parâmetros, obtemos informações sobre um repositório específico. Neste cenário, omitindo a variável Nome do Repositório, a API retornará dados sobre todos os repositórios associados a conta. No caso de alterarmos na URL a constante repos para users teremos elementos associados a conta. E assim, parâmetros são adicionados ou modificados, afim de se obter as informações desejadas.

O retorno é em Json(*JavaScriptObjectNotation*) (Figura4). Se trata de uma notação que para seres humanos é de fácil compreensão, e para máquinas, fácil de interpretar e gerar. Tipado, ele oferece suporte ao paradigma orientado a objetos (JSON, 2018).

```
{
    id: 125863553,
    name: "WEB-APP",
    full_name: "gajfmg/WEB-APP",
  - owner: {
        login: "gajfmg",
        id: 18491844,
        avatar_url: "https://avatars0.githubusercontent.com/u/18491844?v=4",
        gravatar_id: "",
        url: "https://api.github.com/users/gajfmg",
        html_url: "https://github.com/gajfmg",
        followers_url: "https://api.github.com/users/gajfmg/followers",
        following_url: "https://api.github.com/users/gajfmg/following{/other_user}",
        gists_url: "https://api.github.com/users/gajfmg/gists{/gist_id}",
        starred_url: "https://api.github.com/users/gajfmg/starred{/owner}{/repo}",
        subscriptions_url: "https://api.github.com/users/gajfmg/subscriptions",
        organizations_url: "https://api.github.com/users/gajfmg/orgs",
        repos_url: "https://api.github.com/users/gajfmg/repos",
        events_url: "https://api.github.com/users/gajfmg/events{/privacy}",
        received_events_url: "https://api.github.com/users/gajfmg/received_events",
        type: "User",
        site_admin: false
    },
```


Figura 4. Exemplo de retorno Json através da API. Fonte: Elaborado pelo autor (2018).

Por se tratar de uma ferramenta aberta, o GitHub possui inúmeros repositórios abertos que podem ser acessados de forma livre. Para esse trabalho, foram escolhidos de forma aleatória projetos abertos dentro do escopo de data. Deste modo, podemos ter um número maior de dados a serem trabalhados, gerando um relatório mais abrangente.

3.2. Construção do Extrator

Com o Json fornecido pelo GitHub à disposição, é possível fazer bom uso de uma de suas propriedades. Por ser tipado, este conjunto ordenado de dados pode ser transformado em um objeto. Para tal, escolhida à linguagem de programação Java. Escolhida devido a familiaridade do autor e por dispor de bibliotecas que tornam a leitura e parsing de um Json simplificado, abstraindo boa parte do processo.

Para obtermos uma String populada com Json oriundo de uma API WEB, é utilizado o método StringBuilder (ORACLE, 2018) passando como parâmetro a URL da API. Este método permite criar e manipular dinamicamente uma String Java. Neste ponto, é feito uma requisição HTTP (*HyperTextTransferProtocolSecure*) esperando a leitura da página, e assim, temos a String construída e populada com o Json. Como se pode demonstrar no da Figura 5.

```
private StringBuilder recuperarJsonPorUrl(String url) throws IOException {
        HttpClient client;
        client = HttpClientBuilder.create().build();

        HttpGet request = new HttpGet(url);
        HttpResponse response = client.execute(request);

        BufferedReader rd  = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));

        String line = "";
        StringBuilder sb = new StringBuilder();
        while((line = rd.readLine())!= null) {
                sb.append(line + "\n");
        }
        rd.close();

        return sb;
}
```

Figura 5. Armazenamento de dados em uma String com Json. Fonte: Elaborado pelo autor (2018).

O Projeto Gson, é uma biblioteca desenvolvida pelo Google que tem, dentre outras, a capacidade de gerar um objeto Java a partir de uma string Json (GOOGLE, 2018). E isso pode ser feito com poucas linhas de código. No contexto de extração das informações principais de um projeto no GitHub, foram criadas as classes que representam informações sobre projeto e autor, bem como populada uma lista com os demais dados a serem extraídos. No intuito de chegar até a página da API que contém as informações que desejamos é utilizado a parametrização da URL, como descrito no Capítulo 3. Demonstrado no código da Figura 6.

```
private final String login_github ="gajfmg";
private final String nome_projeto ="WEB-APP";
private final String URL_PADRAO = "https://api.github.com/repos/"+login_github+"/"+nome_projeto;
private final String URL_COMMITS = "/commits";
```

Figura 6. Navegação na API. Fonte: Elaborado pelo autor (2018).

Instanciada a classe Gson, é chamado o método fromJson, (GOOGLE, 2018) passando como parâmetros o método recuperarJsonPorUrl que tem como retorno a string Json, como segundo parâmetro, definimos a criação de um objeto Java. Com isso, temos objeto Autor, objeto Projeto e uma Lista com outras informações pertinentes como descrito no código abaixo (Figura 7). A partir deste momento, o Json cumpre o seu objetivo e passamos a lidar apenas com Java.

```
private void inicializarObjetos() throws IOException {
        Type collectionType = new TypeToken<List<Push>>(){}.getType();
        Type objectType = new TypeToken<Owner>(){}.getType();
        Type type  = new TypeToken<Project>(){}.getType();
        Gson gson = new Gson();
        lstPush = (List<Push>) gson.fromJson(recuperarJsonPorUrl(URL_PADRAO + URL_COMMITS).toString(), collectionType);
        owner = gson.fromJson(recuperarJsonPorUrl(URL_PADRAO).toString(), objectType);
        owner = gson.fromJson(recuperarJsonPorUrl(owner.getOwner().getUrl()).toString(), objectType);
        project = gson.fromJson(recuperarJsonPorUrl(URL_PADRAO).toString(), type);
}
```

Figura 7. Inicialização de um Objeto java usando o Gson. Fonte: autor (2018).

3.3. Modelagem dimensional dos dados extraídos do GitHub

Em 2006, a Amazon lançou oficialmente a sua plataforma de serviços de computação em nuvem. A plataforma possui diversos serviços atendendo as mais diversas demandas, seja para projetos acadêmicos e com licenças gratuitas até grandes projetos corporativos de alto orçamento (AMAZON, 2018). A AWS (Amazon Web Services) oferece um arsenal poderoso contendo serviços de processamento, armazenamento, redes, banco de dados e muitos outros.

O serviço utilizado no contexto deste projeto é o de banco de dados. O RDS (*Relational Database Service*) serve ao propósito de termos a base de dados em nuvem, para que possam ser gerenciadas pelas aplicações desenvolvidas de qualquer lugar a qualquer momento (AMAZON, 2018). Criada a instância do RDS na AWS o modelo dimensional pode finalmente ser concebido.

Para armazenamento dos dados extraídos e transformados nas Seções 3.1 e 3.2. foi elaborado o modelo dimensional discutido a seguir.

A tabela `t_dmsao_tempo` armazena datas completas. A tabela `t_repo_proj` guarda os identificadores e nomes dos repositórios a serem monitorados. O mesmo acontece com `t_desenv` e `t_grnte_proj`, contudo, estas são responsáveispor desenvolvedores e gerentes respectivamente. A tabela `t_repo_proj_grnte` se relaciona com os repositórios e gerentes cadastrados armazenando somente suas chaves. Deste modo, podemos determinar qual gerente está associado a cada repositório. Por fim, a tabela fato t_trans_repo_proj guarda as chaves de todas as outras tabelas.

O SGBD escolhido foi o MySql, escolhido devido à familiaridade do autor e da compatibilidade com o RDS. A Figura 8 apresenta o modelo dimensional utilizado nesse trabalho e para a carga de dados.

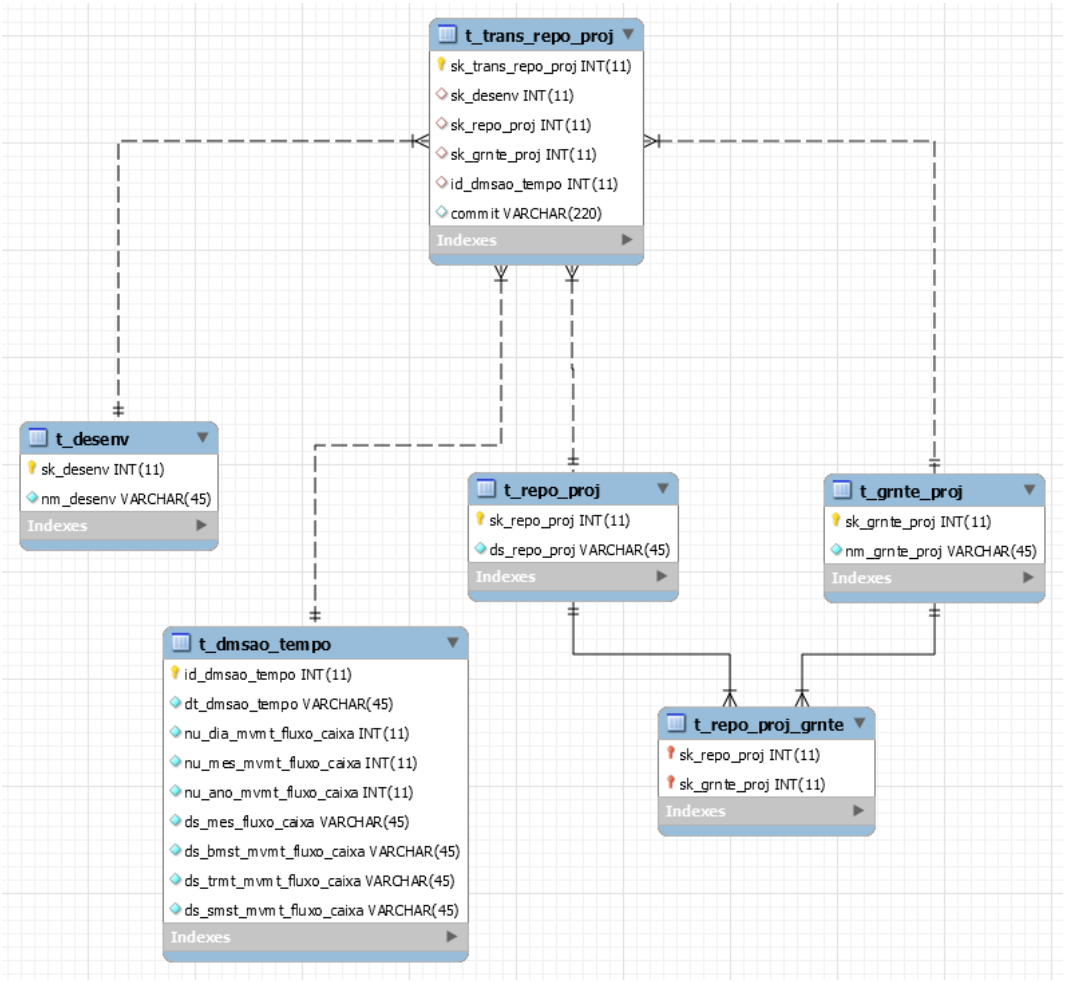

Figura 8. DTR. Fonte: Elaborado pelo autor (2018).

3.4. Carga dos dados no modelo Dimensional

Para apoiar a carga de dados, alguns cadastros foram desenvolvidos para servirem como base para alimentar o modelo dimensional. Estes cadastros foram desenvolvidos utilizando a linguagem PHP a fim de gerar uma aplicação WEB.

O objetivo principal desta aplicação é de facilitar o controle do gerente sobre os projetos monitorados, permitindo que o processo de transformação referencie o projeto e desenvolvedor cadastrado na plataforma do GitHub com o respectivo projeto e desenvolvedor a ser armazenado no modelo dimensional. A Figura 9 apresenta a interface desta aplicação de apoio a este trabalho.

O cadastro de controle deve ser feito informando o nome completo do programador juntamente com seu código pessoal. Neste ponto é necessário que o nome informado seja rigorosamente o mesmo do nome do programador ou gerente associado a base de dados do GitHub. O código pessoal é de âmbito administrativo, respeitando o padrão adotado pela organização. O mesmo acontece com o nome e número identificador do projeto. É importante salientar que o GitHub fornece identificadores para gerentes, desenvolvedores e projetos. Esses identificadores podem perfeitamente ser o padrão a ser adotado, todavia neste contexto a escolha é aberta.

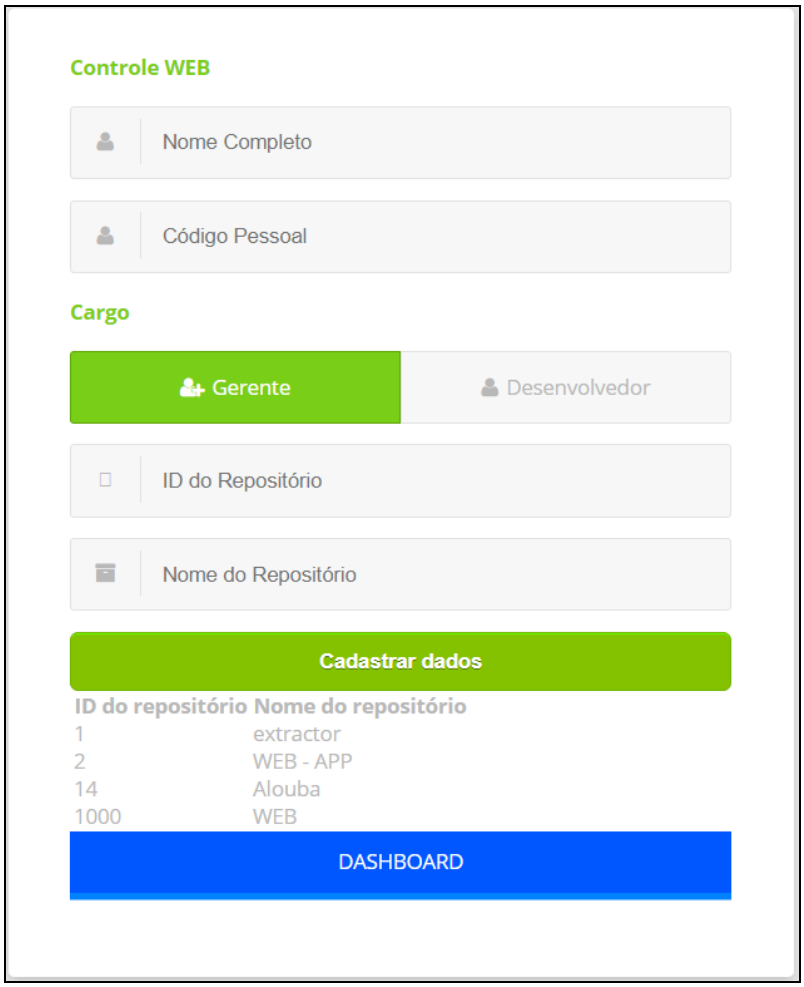

Figura 9. Interface da Aplicação WEB. Fonte: Elaborado pelo autor (2018).

De caráter técnico, a aplicação utiliza-se do módulo PDO (*PHP Data Objects*). Este, é capaz de prover de modo simplificado e leve o acesso e manipulação dos dados as tabelas do banco de dados (PHP, 2018). Respeitando os relacionamentos descritos no Capítulo 3.2, a aplicação popula as tabelas t_grnte_proj, t_desenv, t_repo_proj e t_repo_proj_grnte. A Tabela t_dmsao_tempo é populada previamente de forma manual.

A última tabela a ser populada é a t_trans_repo_proj, que neste contexto é a tabela Fato. Contendo as chaves das demais tabelas como descrito no Capítulo 3.2 ela é alimentada pelo extrator. Com os dados do GitHub extraídos e as tabelas preenchidas, é possível cruzar essas informações afim de gerar chaves devidamente validadas que alimentarão a tabela Fato. Tendo em mãos a string populada, extraímos o nome (para projeto,gerente e desenvolvedor) e data (para referências sobre o tempo a comparação é feita por meio do dia, mês e ano) (Figura 10).

```java
private Integer extrairIdGerentePorNome(Owner owner) throws SQLException {
    return gerenteDao.recuperarIdGerentePorNome(owner.getName());
}

private Integer extrairIdProjetoPorNome(Project project) throws SQLException {
    return projetoDao.recuperarIdProjetoPorNome(project.getName());
}

private Integer extrairIdDmsaoTempo(Push push) throws SQLException {
    Date dataCommit = push.getCommit().getAuthor().getDate();
    Calendar c = Calendar.getInstance();
    c.setTime(dataCommit);
    Integer dia = c.get(Calendar.DAY_OF_MONTH);
    Integer mes = c.get(Calendar.MONTH);
    Integer ano = c.get(Calendar.YEAR);

    return dimensaoTempoDao.recuperarIdDataPorDiaMesAno(dia, mes + 1, ano);
}
```

Figura 10. Extração das chaves. Fonte: Elaborado pelo autor (2018).

Agora é feita a busca, caso exista, da chave correspondente nas tabelas. A Figura 11 demonstra o exemplo para a chave correspondente ao projeto. O comentário de um commit é capturado e gravado sem verificação.

```java
public class ProjetoDao {
    Connection connection = null;
    private final String RECUPERAR_ID_PROJETO_POR_NOME = "SELECT sk_repo_proj FROM t_repo_proj WHERE ds_repo_proj = '";


    public ProjetoDao () {
        connection = new ConnectionFactory().getConnection();
    }

    public Integer recuperarIdProjetoPorNome (String nome) throws SQLException {
        Statement ps = connection.createStatement();
        ResultSet rs;

        rs = ps.executeQuery(RECUPERAR_ID_PROJETO_POR_NOME + nome + "'");

        Integer id = null;

        while (rs.next()) {
            id = rs.getInt("sk_repo_proj");
        }
        ps.close();
        return id;
    }
}
```

Figura 11. Extração das chaves. Fonte:Elaborado pelo autor (2018).

No extrator é utilizado um VO (*Value Object*) que armazena os dados. O VO representa um auxílio na manipulação destes dados, deste modo os acessamos de um único lugar (Figura 12).

```
public class TransRepoProjVO {

    private int id;
    private int idDesenvolvedor;
    private int idRepoProj;
    private int idGerenteProj;
    private int idDmsaoTempo;
    private String commit;

    /**
     * Este atributo é auto incremento.
     */
    public int getId() {
        return id;
    }

   /**
    * Este atributo é recuperado da tabela TAL.
    */
   public int getIdDesenvolvedor() {
       return idDesenvolvedor;
   }

   public void setIdDesenvolvedor(int idDesenvolvedor) {
       this.idDesenvolvedor = idDesenvolvedor;
   }

   public int getIdRepoProj() {
       return idRepoProj;
   }

   public void setIdRepoProj(int idRepoProj) {
       this.idRepoProj = idRepoProj;
   }

   public int getIdGerenteProj() {
       return idGerenteProj;
   }

   public void setIdGerenteProj(int idGerenteProj) {
       this.idGerenteProj = idGerenteProj;
   }

   public int getIdDmsaoTempo() {
       return idDmsaoTempo;
   }

   public void setIdDmsaoTempo(int idDmsaoTempo) {
       this.idDmsaoTempo = idDmsaoTempo;
   }

   public String getCommit() {
       return commit;
   }

   public void setCommit(String commit) {
       this.commit = commit;
   }
```

Figura 12. VO. Fonte: Elaborado pelo autor (2018).

Neste ponto feito um acesso em loop ao VO e obtêm-se as chaves válidas representando gerente, desenvolvedor, projeto, data, além do comentário extraído de forma direta, tudo pelo método extrairDados (Figura 13).

```
public void extrairDados() throws SQLException, IOException {
    inicializarObjetos();
    for(Push push : lstPush){
        if(push != null){
            if (extrairIdAuthorPorNome(push) != null) {
                TransRepoProjVO vo = new TransRepoProjVO();
                vo.setIdDesenvolvedor(extrairIdAuthorPorNome(push));
                vo.setCommit(push.getCommit().getMessage());
                vo.setIdDmsaoTempo(extrairIdDmsaoTempo(push));
                vo.setIdGerenteProj(extrairIdGerentePorNome(owner));
                vo.setIdRepoProj(extrairIdProjetoPorNome(project));
                list.add(vo);
            }
        }
```

Figura 13. Chaves validadas. Fonte: Elaborado pelo autor (2018).

A carga das chaves capturadas na tabela fato é por meio do método salvarDados que acessa o VO e executa a query de inserção (Figura 14).

```
public class ExtractorDao {
    Connection connection = null;
    private final String INCLUIR_DADOS_EXTRAIDOS =
            "INSERT INTO t_trans_repo_proj (sk_desenv, sk_repo_proj, sk_grnte_proj, id_dmsao_tempo, commit ) VALUES (?,?,?,?,?)";


    public ExtractorDao () {
        connection = new ConnectionFactory().getConnection();
    }

    public void salvarDados (TransRepoProjVO vo) {

        PreparedStatement ps;
        try {
            ps = (PreparedStatement) connection.prepareStatement(INCLUIR_DADOS_EXTRAIDOS);
            ResultSet rs;
            ps.setInt(1, vo.getIdDesenvolvedor());
            ps.setInt(2, vo.getIdRepoProj());
            ps.setInt(3, vo.getIdGerenteProj());
            ps.setInt(4, vo.getIdDmsaoTempo());
            ps.setString(5, vo.getCommit());
            ps.executeUpdate();
        } catch (SQLException ex) {
            Logger.getLogger(ExtractorDao.class.getName()).log(Level.SEVERE, null, ex);
        }


    }
}
```

Figura 14. Carga de dados na tabela Fato. Fonte: Elaborado pelo autor (2018).

Com a base de dados dimensional populada, é finalizado o processo de ETL. Agora, é preciso fazer a análise destes dados para que possamos gerar relatórios gerenciais.

4. Geração de Painéis para acompanhamento da produtividade no desenvolvimento de software

Esta seção descreve como os dados gerados a partir do processo de ETL abordado na Seção 3 pode ser apresentado em relatórios gerenciais utilizados ferramentas de *dashboard*. A ferramenta utilizada neste trabalho para geração de dados gerenciais é a Microsoft Power BI.

De acordo com a Microsoft (2018) o Power BI contempla um conjunto de ferramentas de software utilizadas para a análise de negócios e apoio a decisões. Os chamados *dashboards* (ou painéis) construídos por meio das soluções do Power BI,

fornecem uma visão de “360 graus” para os usuários corporativos sendo possível criar métricas em uma solução única, e por meio destas soluções os dados são atualizados em tempo real e disponíveis em diversos dispositivos. Os usuários podem explorar os dados através de funcionalidades intuitivas que facilitam a geração de informações para análise de negócios. Além disso, a criação de um painel é uma tarefa descomplicada, suportada por conexões e diversos SGBDs. Os dados e relatórios podem acessar os painéis em qualquer lugar, e que se atualizam automaticamente com qualquer alteração em seus dados.

O *dashboard* apresentado abaixo na Figura 15 mostra um exemplo de métrica para a quantidade de *commits* realizados por mês, onde podemos ver diversos projetos monitorados e seus desenvolvedores. Neste caso por exemplo, pode-se ver que o mês de março possui o maior número de *commits*. Esses indicativos são significativos, pois podemos perceber de forma bastante visual quais meses são de fato mais produtivos, e quais meses merecem atenção do gerente visando aumentar essa produtividade. Metas devem ser estabelecidas. Estes gráficos devem estar ao alcance dos membros da equipe de modo a terem total compreensão do andamento do projeto.

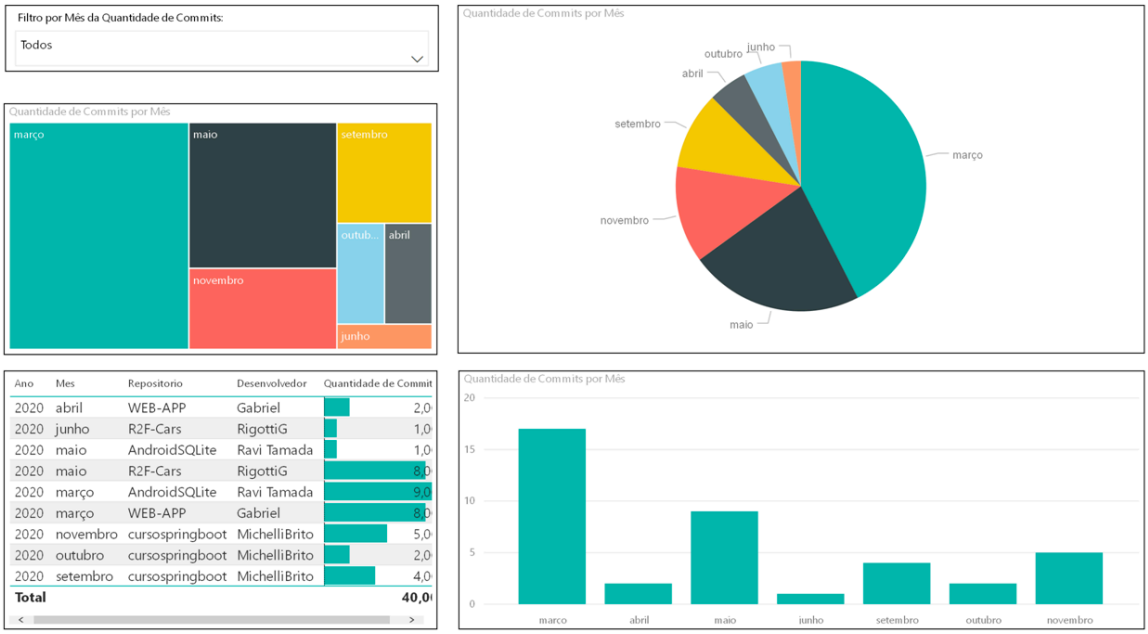

Figura 15. Dashboard. Fonte: Elaborado pelo autor (2018).

É possível gerar muitos outros gráficos com as informações obtidas da API do GitHub. Sendo assim, pode-se ter métricas personalizadas já que trabalhamos de forma direta com a base de dados da plataforma. Um exemplo seria poder determinar um padrão de comentários (número de caracteres, letras maiúsculas, minúsculas e etc) e determinar de forma automatizada se os desenvolvedores estão, ou não, o seguindo. Isso pode ser feito em conjunto com o número de commits detalhado na figura acima. Deste modo temos informações sobre quais projetos estão mais ativos e seguem melhor um padrão estabelecido. Isso foge do escopo de funcionalidades fornecidas pelo GitHub. De todo modo, as análises que podem ser feitas envolvem o tipo de métrica a ser processada.

5. Considerações Finais

Gerar informações gerenciais a partir de uma massa de dados é uma necessidade constante dos gerentes de projeto de software. A tomada de decisões em uma

organização passa pela análise destes dados. As ferramentas desenvolvidas e propostas apoiam o processo de ETL sob uma plataforma já bastante popular no universo de desenvolvimento de software, o GitHub. Trabalhar nativamente com a database do GitHub permite a criação de métricas próprias adaptadas a realidade de cada organização. Com este trabalho, temos a possibilidade de integrar os dados extraídos do ambiente do GitHub para promover uma análise que vai além do que o próprio GitHub oferece. Tornando possível mensurar com cada vez mais exatidão a evolução de um projeto, desenvolvedor e gerente ao longo do tempo

Mesmo com as limitações técnicas que possui hoje o extrator, com as informações obtidas através dele é possível promover o uso de novas tecnologias para apoiar a geração de informações gerenciais. O uso de uma base de dados multidimensional torna possível a análise de diversas informações que serão usadas para geração de *dashboards* interativos, portáteis e personalizáveis. Os *dashboards* podem deixar cada vez mais o gerente a par de tudo o que ocorre no projeto que administra a qualquer hora e de qualquer lugar.

Como proposta de trabalhos futuros, pretende-se aplicar os resultados deste trabalho para gerar informações para outros processos para manutenção de software, tais como controle de mudanças, integração contínua e entrega contínua.

Referências

AMAZON. **Cloud solutions**. Disponível em: <https://aws.amazon.com/pt/solutions/?nc2=h_ql_s&awsm=ql-2>. Acesso em: 09 jun. 2018.

DEVELOPER GITHUB. **Restapi v3**. Disponível em: <https://developer.github.com/v3/?>. Acesso em: 08 jun. 2018.

ELMASRI, Ramez; NAVATHE, Shamkant. Fundamentals ofdatabase systems. Addison-Wesley PublishingCompany, 2010.

GITHUB. **Gsonuserguide.** Disponível em: <https://github.com/google/gson/blob/master/userguide.md>. Acesso em: 08 jun. 2018.

JSON. **Json**. Disponível em: <https://www.json.org/json-pt.html>. Acesso em: 08 jun. 2018.

KIMBALL, Ralph; ROSS, Margy. **The data warehouse toolkit: the complete guideto dimensional modeling**. John Wiley& Sons, 2011.

MICROSOFT. **O que é o PowerBI?** Disponível em https://powerbi.microsoft.com/pt-br/what-is-power-bi/. Acesso em 18 de Abr de 2018 (a).

MICROSOFT. **Microsoft recognized as a leader in analytics and BI platforms for 11 years**. Disponível em: https://info.microsoft.com/ww-landing-gartner-bi-analytics-mq-2018.html. Acesso 10 de Jun de 2018 (b).

ORACLE. **Java documentation**. Disponível em: <https://docs.oracle.com/javase/tutorial/java/data/buffers.html>. Acesso em: 09 jun. 2018.

PHP. **Intropdo**. Disponível em: <http://php.net/manual/pt_br/intro.pdo.php>. Acesso em: 08 jun. 2018.

SOMMERVILLE, Ian. **Engenharia de software**. 9 ed. São Paulo: Pearson, 2011. p.481-484.

CHACON, Scott; STRAUB, Ben. **Pro git**: Everything you need to know about GIT. 2 ed. [S.L.]: Apress, 2014. 516 p.