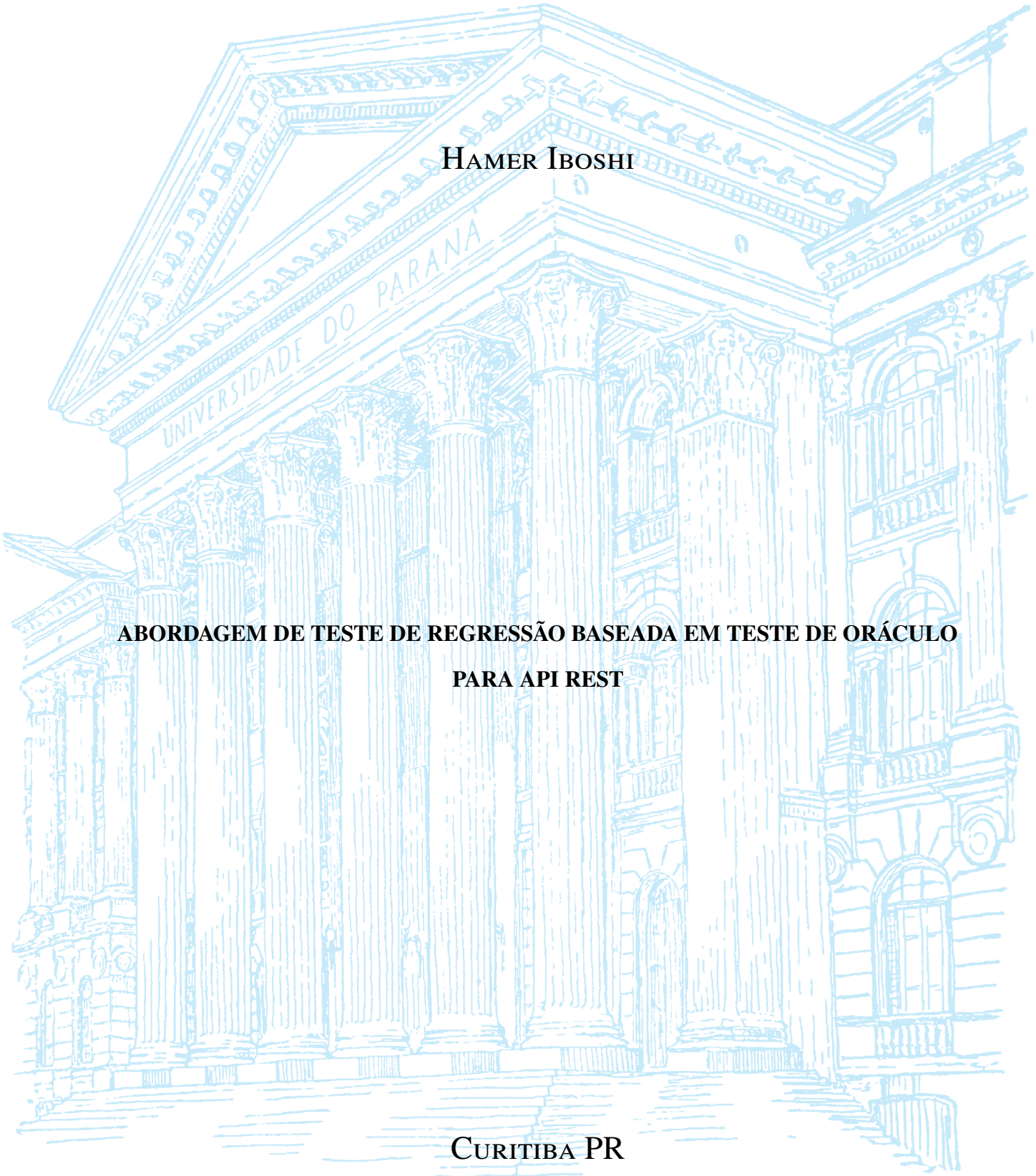
Universidade Federal do Paraná

Hamer Iboshi



ABORDAGEM DE TESTE DE REGRESSÃO BASEADA EM TESTE DE ORÁCULO PARA API REST

Curitiba PR
2019

Hamer Iboshi

ABORDAGEM DE TESTE DE REGRESSÃO BASEADA EM TESTE DE ORÁCULO PARA API REST

Trabalho apresentado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Marcos Didonet Del Fabro.

Curitiba PR
2019

Agradecimentos

Agradeço primeiramente a Deus que durante a minha caminhada na universidade se fez mais do que presente principalmente nos momentos mais difíceis.

Aos meus amigos do Dinf que sempre me ajudaram muito, estudando, fazendo trabalhos juntos e principalmente pela amizade tornando esses anos mais divertidos e especiais. Em especial o Fernando Erd e o Jedian Brambilla que auxiliaram a revisar esse trabalho.

Aos amigos do Jesus na UFPR por serem uma verdadeira família aqui em Curitiba, vocês me inspiram e me ensinaram de verdade a buscar e me parecer mais com Jesus, ainda assim continua sendo impossível tentar descrever em poucas palavras tudo que vivi e aprendi com eles.

A minha família que mesmo de longe me apoiou em todos os momentos, em especial aos meus avós Fátima e Maurílio por assumirem papel de Pai e Mãe durante a maior parte da minha vida, minha mãe Andreia e meu Pai Luciano que mesmo não estando perto sempre me suportaram de inúmeras formas. Amo muito vocês!

Por último mas não menos importante ao meu orientador Didonet, por sua orientação, não só durante esse trabalho mas em todo período que fui bolsista do C3SL no LDE. Agradeço também aos meus companheiros de equipe no C3SL.

A todos eles que contribuíram direta ou indiretamente na minha vida, minha eterna gratidão!

Resumo

Um dos desafios ao desenvolver um sistema OLAP (Processamento Analítico Online) é garantir que dados já validados não sejam afetados pelo desenvolvimento de novas funcionalidades ou pela inserção e processos que precisam ser aplicados nos dados. O objetivo deste trabalho é identificar quando este tipo de problema ocorre no intuito de evitar que mudanças indesejadas ou efeitos colaterais afetem funcionalidades e dados já validados. A abordagem proposta neste trabalho visa resolver este problema utilizando testes de regressão, detalhando a implementação da abordagem, as características do sistema, como a abordagem é inserida na arquitetura desse sistema e demonstrando como foi a sua aplicação no Laboratório de Dados Educacionais voltado a dados educacionais abertos do Brasil.

Palavras-chave: teste de software, teste de regressão, API REST, OLAP, teste de oráculo.

Abstract

A challenge for developers of an OLAP system is to guarantee that data already validated don't be affected by the development of new features or insertion and process applied to data. The goal of this work is to identify when this kind of problem occurs to prevent undesired changes or side effects that affect features or validated data. The proposed approach of this work aims to solve this problem using regression tests, detailing the implementation, characteristics of the system, how the approach is inserting at the architecture of this system and showing how was applied at Educacional Data Lab (LDE) a system of Brazil educational open data.

Keywords: software test, regression test, API REST, data warehouse, oracle test.

Lista de Figuras

Lista de acrônimos

DINF	Departamento de Informática
UFPR	Universidade Federal do Paraná
OLAP	Processamento Analítico Online
API	Interface de Programação de Aplicação
REST	Transferência Representacional de Estado
C3SL	Centro de Computação Científica e Software Livre
LDE	Laboratorio de Dados Educacionais
SIMCAQ	Simulador de Custo-Aluno Qualidade
PNAD	Pesquisa Nacional por Amostra de Domicílios
IBGE	Instituto Brasileiro de Geografia e Estatística
INEP	Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira

Sumário

1 Introdução

A motivação principal deste trabalho surgiu partindo do contexto onde havia um fluxo de trabalho onde o setor de pesquisa de políticas educacionais da UFPR enviava para equipe de desenvolvimento fichas técnicas sobre como os indicadores educacionais deveriam ser desenvolvidos, com a fonte dos dados, as variáveis utilizadas, como seria calculado, entre outras coisas. Depois a equipe de desenvolvimento do C3SL utilizava as fichas técnicas para implementação dos indicadores no LDE, isso engloba a inserção e transformação dos dados, a implementação das regras de negócio e cálculo do indicador, e na parte final o desenvolvimento da interface que é onde os usuários visualizam os dados disponibilizados pela plataforma. Após esse processo as pessoas do setor de pesquisa em políticas públicas faz a validação dos dados, fazendo os cálculos dos indicadores com os dados direto da fonte de dados original e depois comparando com os dados fornecidos pelos indicadores da plataforma. O maior problema acontece depois durante a evolução do sistema eles começaram a notar que os resultados apresentavam divergência a medida que o LDE era desenvolvido, então surgiu um processo de re-validação, na qual todo processo de validação era refeito. O principal problema é que esse processo detectava o problema meses depois que uma alteração ou mudança era realizada, ou seja, quando os erros eram reportados os desenvolvedores estavam envolvidos com outras atividades diferentes. Isso tudo implica que um esforço grande era necessário para investigar quais mudanças ocasionaram os efeitos colaterais nos resultados já validados, para depois resolver os problemas e o próprio processo de revalidação levava dias para ser feito, ressaltando que esse problema acontecia com frequência.

Um dos desafios de um software[1] OLAP (Processamento Analítico Online) em desenvolvimento que possui um fluxo de dados que passa por transformações é garantir que as funcionalidades já concluídas e validadas não sofram efeitos colaterais, fazendo com que resultados até então corretos sejam afetados por alterações posteriores. Essas alterações podem ser realizadas durante os processos que influenciam no fluxo dos dados, como a extração, transformação e carregamento, ou durante a implementação das regras de negócio na Interface de Programação de Aplicação (API).

Nesse contexto o objetivo deste trabalho é propor uma abordagem para realização de testes de regressão, que é definido por Pressman (2010) como uma reexecução de um certo conjunto de testes que já foi realizado para assegurar que mudanças não propaguem efeitos colaterais indesejados. Possibilitando a partir desta abordagem que falhas e efeitos colaterais sejam identificados a medida que mudanças são realizadas em um software OLAP.

Este trabalho está organizado da seguinte forma. No Capítulo 2 os fundamentos de testes de software, data warehouse utilizando modelo OLAP e API serão explicados para o entendimento da abordagem. O Capítulo 3 discorre sobre trabalhos relacionados, em especial será abordado

[1] De acordo com Pressman (2010) software são: (1) instruções (programas de computador) que quando executados produzem funcionalidades, funções, e performance; (2) estruturas de dados que possibilitam os programas manipular informações adequadamente, e (3) informação descritiva em dois aspectos cópia física e formulários virtual que descrevem as operações e como se utiliza os programas.

sobre um artigo elaborado por Li e Offutt (2017), onde são definidos conceitos base e traz uma breve comparação com o estado da arte no tema de testes de oráculo. No Capítulo 4 é apresentado detalhes sobre a proposta, explicando mais a fundo sobre a arquitetura, os componentes do sistema e como a solução está presente nesse contexto, detalhes da implementação e finalizando a seção com aplicação da solução como teste automatizado em um projeto do Centro de Computação Científica e Software Livre (C3SL) que é o Laboratório de Dados Educacionais (LDE). Capítulo 5 discorre sobre as contribuições da abordagem e o que poderia ser feito para trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo será possível compreender conceitos, fundamentos e alguns desafios que englobam a abordagem proposta, ainda que o principal tema da abordagem seja os testes de software a arquitetura do software reflete diretamente na necessidade de tipos específicos de teste, que cobrem desde processos de extração, transformação e inserção no banco de dados até as saídas geradas pelo sistema.

2.1 Teste de Software

Toda está seção está embasada no autor Pressman (2010), explicando os principais tópicos de teste de software no escopo deste trabalho. O autor divide testes de software em dois grandes objetivos, verificação e validação. Verificação assegura que o software implementa corretamente uma função específica. A validação tem um conjunto diferente de tarefas onde assegura que o software foi desenvolvido de acordo com os requisitos. Uma perspectiva macro de estratégias de teste de software e do processo de software pode ser visualizada com o exemplo da espiral na Figura 2.1.

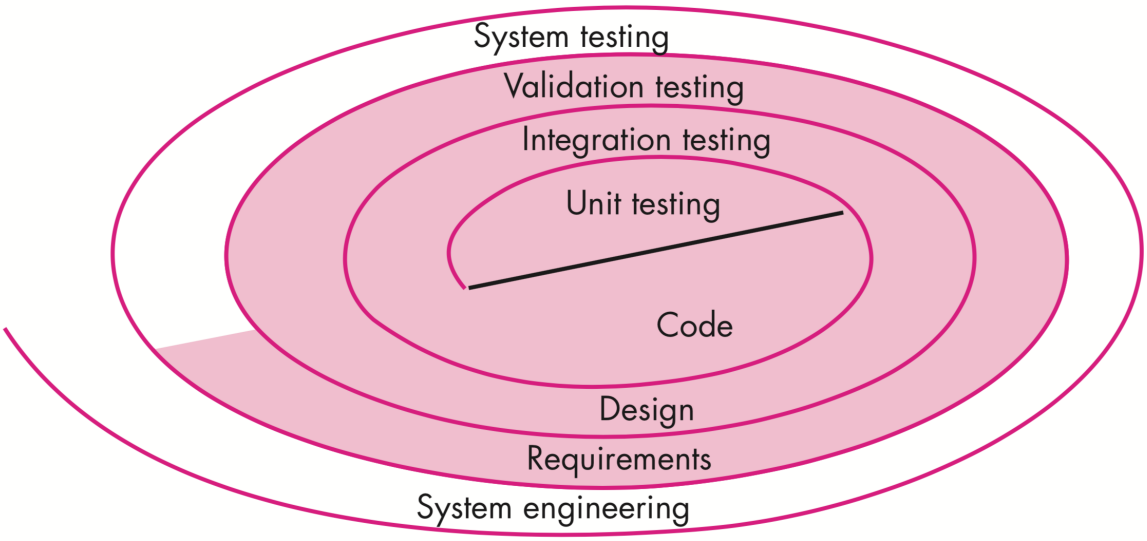


Figura 2.1: Estratégias de Teste

Fonte: Pressman (2010)

Iniciando no escopo mais interno da espiral e avançando no sentido horário os Testes de Unidade ou Unitários (*Unit testing*) é concentrado em cada unidade (por exemplo: componentes,

classes, ou objetos da aplicação Web) do código fonte do software implementado, com objetivo de assegurar uma cobertura completa e máxima detecção de erro no código. Seguindo o fluxo o Teste de Integração (*Integration testing*) onde o foco está no projeto e na construção da arquitetura do software, por exemplo, nesta fase há uma preocupação com a entrada, saída e principalmente com a cobertura sobre os principais caminhos de controle do software. Avançando há o teste de validação (*Validation testing*), como já comentado anteriormente os requisitos estabelecidos durante a modelagem dos requisitos são validados com o software que foi construído, fornece uma garantia final no sentido dos requisitos informacionais, funcionais, comportamentais e de performance do sistema. Então na fase mais externa da espiral o Teste de Sistema (*System testing*) onde o comportamento do sistema e outros elementos do sistema são testados como um todo. Portanto para realizar teste de software é necessário passar por cada fase da parte interna e assim expandindo o escopo para as outras fases.

É importante ressaltar que o autor explica as estratégias de software que são convencionais no desenvolvimento de software, ou seja, que são utilizadas por equipes de desenvolvimento. Nesta mesma linha há várias abordagens para implementação de uma estratégia, a mais utilizadas pelas equipes é uma visão onde os testes são realizados incrementalmente, começando individualmente pelas unidades, depois testando a integração entre os componentes do software, resultando em testes que exercitam o sistema que está sendo construído. Nas subseções abaixo nos aprofundaremos em algumas estratégias.

2.1.1 Testes de Unidade

Testes de unidade (ou testes unitários) focam especificamente na menor unidade do projeto de software, componente ou módulo. Utilizando descrições do nível de componente do projeto como guia para testar caminhos de controle no intuito de descobrir erros no escopo do módulo. A unidade foca na lógica de processamento interno e estruturas de dados do escopo do componente.

A interface do módulo é testada para assegurar que a informação siga o fluxo para saída de acordo com a unidade do programa que está sendo testado, as estruturas de dados locais são testadas para assegurar que os dados armazenados temporariamente se mantém íntegros durante todos os passos da execução. Todos os caminhos de controle independentes são testados para assegurar que os módulos são executados pelo menos uma vez. Condições de escopo são testadas para assegurar que o módulo opera de acordo com o escopo estabelecido para limitar ou restringir o processamento. Por último todos os caminhos de tratamento de erro são testados, alguns tipos de erros que deveriam ser testados quando avaliamos tratamentos de erros são:

1. descrição do erro é incompreensível;
2. erro notificado não corresponde com o erro encontrado;
3. condição do erro causa intervenção do sistema antes do tratamento do erro;
4. condição de exceção processada é incorreta;
5. descrição do erro não fornece informação suficiente para auxiliar na localização da causa do erro.

Normalmente os testes unitários são considerados complementares a cada etapa de codificação e cada caso de teste deve ser acompanhado de um conjunto de resultados esperados. Uma visão geral do ambiente de testes de unidade pode ser visto abaixo na Figura 2.2. Os

componentes não são independentes e por isso softwares como o *Driver* e *Stub* são essenciais para o ambiente de testes unitários. O driver é semelhante a um "programa principal"que recebe os dados dos casos de teste, que originam da interface (*Interface*), estruturas de dados locais (*Local data structures*), condições de fronteira (*Boundary conditions*), caminhos independentes (*Independent paths*), caminhos de tratamentos de erros (*Error-handling paths*) e passam esses dados para os componentes serem testados e exibe os resultados. Os stubs servem para substituir módulos que são acionados pelo componente por interfaces que realizam uma manipulação mínima de dados, exibindo a verificação da entrada e retornando o controle para o componente que está sendo testado.

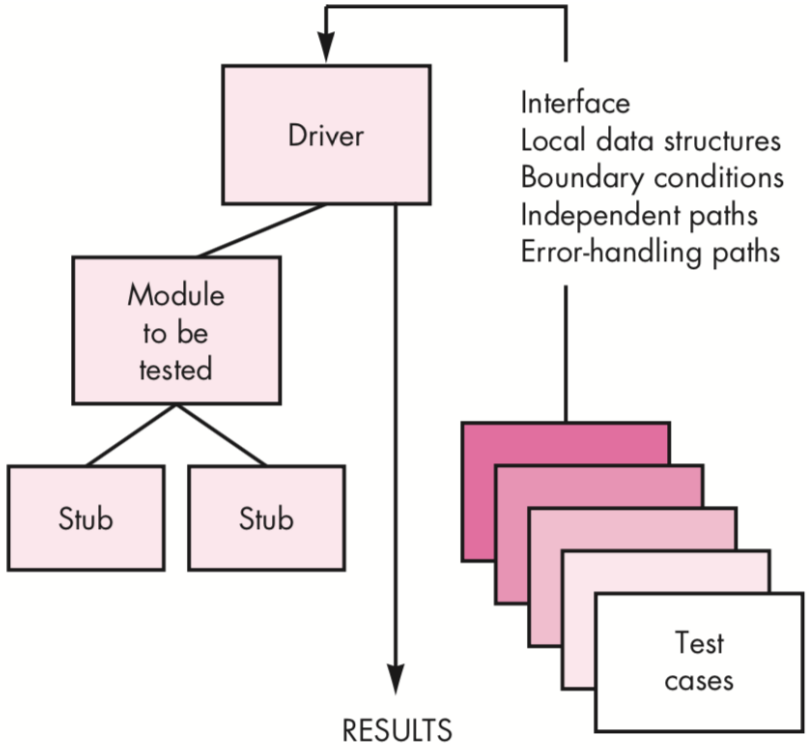


Figura 2.2: Ambiente dos testes de unidade

Fonte: Pressman (2010)

2.1.2 Testes de Integração

Neste momento poderia ser questionado: O que garante que os módulos funcionam quando são colocados para funcionarem juntos? mesmo que todos os módulos do software são testados com testes de unidade.

O autor levanta um questionamento semelhante para pontuar que muitas coisas podem dar errado ao colocar os módulos para funcionarem juntos, por exemplo, dados podem ser perdidos, um componente pode ter um efeito indesejado em outro, subfunções combinadas não cumprem sua função global desejada, imprecisões individuais dos módulos que são permitidas propagam uma imprecisão em níveis não aceitáveis, estruturas de dados globais podem apresentar problemas, entre outros problemas.

Os testes de integração são uma técnica sistemática da construção da arquitetura de software enquanto ao mesmo tempo se cria testes para cobrir erros associados à interface, com o

intuito de utilizar os componentes dos testes de unidade e desenvolver a estrutura do software que deve ser guiada pela arquitetura.

O autor ressalta que há uma tendência catastrófica de realizar uma integração não incremental, que segundo ele é a abordagem "Big Bang", na qual todos os componentes são combinados antes e depois o software é testado por completo resultando em uma catástrofe, onde um conjunto de erros é encontrado de uma vez só. Neste contexto a correção é difícil, pois isolar a causa do problema se torna complicado, e ainda uma vez que erros forem corrigidos podem surgir novos erros e um processo aparentemente interminável continua. Pressman (2010) ressalta que o processo de desenvolvimento ideal realiza uma integração incremental, onde o software é construído e testado em pequenos incrementos e os erros são mais fáceis de serem isolados e corrigidos, desta forma as interfaces também se tornam mais fáceis de serem testadas por completo.

2.1.3 Testes de Regressão

De acordo com Pressman (2010) toda vez que um novo módulo é adicionado como parte dos testes de integração significa que o software muda, novas fluxos de dados, novas operações de entrada e saída podem surgir, e novos controles lógicos do software surgem. Essas alterações podem causar problemas em funções que antes funcionavam perfeitamente, neste contexto de testes de integração o teste de regressão **é uma reexecução de algum conjunto de testes que já foram realizados anteriormente para assegurar que as mudanças não propaguem efeitos colaterais indesejados**. Os testes de regressão em um contexto mais amplo serão bem sucedidos se os resultados descobrirem erros, e os erros devem ser corrigidos. Toda vez que o software é corrigido algo da sua configuração, como o programa, documentação, ou os dados que suportam o software é alterado. Desta forma os testes de regressão contribuem auxiliando que mudanças não gerem comportamentos indesejados ou erros adicionais.

O autor complementa descrevendo que um dos tipos de teste regressão que pode ser conduzidos "manualmente"envolve a execução de um conjunto de testes para todos os casos de testes ou a utilização de ferramentas automáticas de captura/reprodução (*capture/playback tools*). As ferramentas de captura/reprodução permitem que o engenheiro de software **capture os casos de teste e resultados para uma reprodução e comparação posterior**. O conjunto de testes de regressão possuí três tipos diferentes de classes de casos de teste:

- Uma amostra representativa dos testes que vai executar todas as funções do software.
- Testes adicionais que focam nas funções do software que são mais prováveis de serem afetadas por mudanças.
- Testes que focam nos componentes do software que podem ser afetados por mudanças.

Pressman (2010) aconselha que os testes de regressão sejam executados todas as vezes que mudanças majoritárias (ou significativas) forem feitas no software, incluindo mudanças relacionadas a integração de novos componentes. Com o avanço do desenvolvimento do software a quantidade de casos dos testes de regressão podem crescer bastante, portanto o conjunto dos testes de regressão devem ser arquitetados para incluir somente os testes que são direcionados a uma ou mais classes de erros para cada função majoritária, pois torna-se impraticável e ineficiente reexecutar todos os testes para cada função do software a cada mudança.

2.2 Data Warehouse

A literatura sobre *data warehouse* realiza a tradução do termo para armazém ou depósito de dados, mas em geral optam por manter o termo em inglês por ser popular e utilizado com frequência tanto no meio acadêmico como no mercado, por conta disso foi escolhido manter o termo data warehouse neste trabalho.

Elmasri e Navathe (2011) trazem algumas definições concisas sobre o assunto:

1. Um banco de dados como uma coleção de dados relacionados;
2. Um sistema de banco de dados como um banco de dados e um software de banco de dados juntos;
3. Os bancos de dados tradicionais são transacionais (relacionais, orientados a objeto, em rede ou hierárquicos);
4. Um data warehouse também é uma coleção de informações, bem como um sistema de suporte;
5. Os data warehouses têm a característica distintiva de servir principalmente para aplicações de **apoio à decisão** e **são otimizados para recuperação de dados**, e não para processamento de transação de rotina.

Os sistemas de apoio à decisão são descritos por Date (2004) como sistemas que ajudam na análise de informações do negócio e sua meta é auxiliar na administração e a definir tendências, apontar problemas e tomar decisões inteligentes. O banco de dados é principalmente apenas para leitura (*read-only*). As atualizações são limitadas, em geral, as operações periódicas são de carga ou renovação onde essas operações são dominadas por operações inserção (*INSERT*). As operações remoção (*DELETE*) podem ser feitas ocasionalmente. Operações atualização nos dados já inseridos (*UPDATE*) quase nunca são realizadas, apenas em alguns casos a atualização é realizada em algumas tabelas de trabalho auxiliares, mas processos normais de apoio à decisão praticados atualmente quase nunca atualizam o banco de dados de apoio à decisão propriamente dito.

O autor ainda comenta sobre algumas características adicionais:

- As colunas costumam ser usadas em combinação;
- Geralmente a integridade não é uma preocupação (supõe-se que os dados estejam corretos quando são carregados pela primeira vez e não são atualizados);
- As chaves frequentemente incluem um componente temporal;
- O banco de dados costuma ser grande (principalmente levando em consideração a inserção periódica de dados);
- O banco de dados costuma estar fortemente indexado;
- O banco de dados envolve frequentemente vários tipos de redundância controlada.

De acordo com Elmasri e Navathe (2011) um data warehouse com frequência é um depósito de dados integrados de múltiplas fontes, processados para armazenamento em um modelo multidimensional. Diferente da maioria dos bancos de dados transacionais, um data warehouse costumam apoiar a análise de **série temporal** e **tendência**, ambas exigindo mais

dados históricos do que geralmente é mantido nos bancos de dados transacionais. Os bancos de dados transacionais costumam ser voláteis que implica que as informações podem mudar constantemente, diferente de um data warehouse onde as informações mudam com muito menos frequência e podem ser consideradas não de tempo real com atualizações periódicas.

Elmasri e Navathe (2011) define características que diferenciam um data warehouse:

1. Visão conceitual multidimensional.
2. Dimensionalidade genérica.
3. Dimensões e níveis de agregações ilimitados.
4. Operações irrestritas entre dimensões.
5. Tratamento dinâmico de matriz esparsa.
6. Arquitetura cliente-servidor.
7. Suporte para múltiplos usuários.
8. Acessibilidade.
9. Transparência.

Podemos ver acima as características de um data warehouse, agora será abordado algumas definições realizadas por Kimball e Ross (2013) sobre os componentes de um data warehouse e suas funções especificas de uma perspectiva mais voltada pra área de negócios. Na figura 2.3 é ilustrado como esses componentes se organizam, há quatro componentes distintos e separados a serem considerados da perspectiva de um ambiente data warehouse o Sistema de origem operacional (*operational source systems*), Área de armazenamento temporário (*data staging area*), Área de apresentação dos dados (*Data presentation area*), Ferramentas de acesso aos dados (*Data access tools*).

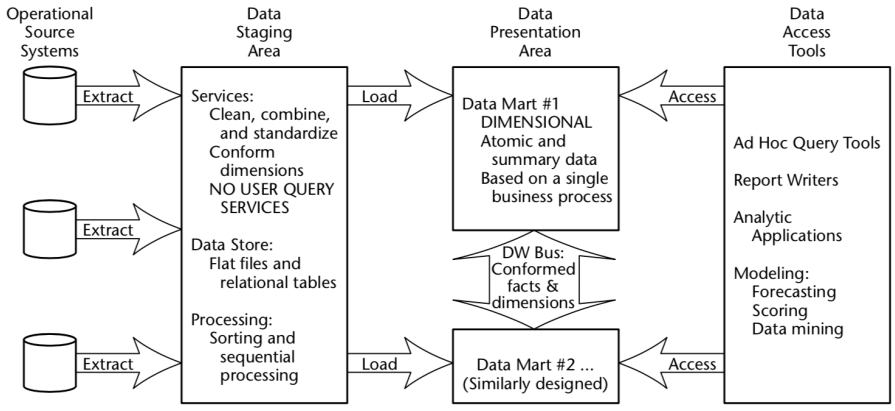


Figura 2.3: Elementos básicos de um data warehouse

Fonte: Kimball e Ross (2013)

2.2.1 Sistemas de origem operacional

Os sistemas de origem operacional são repensáveis por registrar as transações de um negócio e suas prioridades são desempenho e disponibilidade. Consultas neste tipo de sistema são limitadas e normalmente este tipo de sistema mantém poucos dados históricos e quando esses bancos tem um apoio de um data warehouse eles são aliviados da responsabilidade de representar o passado (no contexto de séries históricas).

Uma observação importante pontuada por Elmasri e Navathe (2011) é que este tipo de componente pode ser representado por diversas fontes de dados heterogêneas, por exemplo, outros bancos de dados (como nas definições já dadas acima) e outras entradas de dados, como no contexto do data warehouse utilizado na seção de experimentos onde as fontes dos dados são bases de dados abertos.

2.2.2 Área de armazenamento temporário

A *data staging area* pode ser traduzida como área de armazenamento temporário, área de transição de dados ou ainda área intermediária de armazenamento e é todo processo entre os sistemas de origem operacional (fontes de dados) e área de apresentação de dados (o data warehouse em si) que é normalmente referenciado como ETL[1] (Extração, Transformação e Carregamento).

O processo de **Extração** é o primeiro passo e significa ler e entender a fonte dos dados para copiar os dados necessários para a área de armazenamento temporário para as manipulações seguintes.

Uma vez que os dados são extraídos é iniciado o processo de **Transformação** dos dados, onde há inúmeras transformações que podem ser realizadas, como a limpeza dos dados que engloba correção de erros ortográficos, falta de alguns elementos, identificação, tratamento de dados inválidos, duplicados ou inconsistentes), combinação de múltiplas fontes de dados, estabelecimento de relações de entre chaves e as tabelas, padronização dos tipos de dados, entre outros.

O processo de **Carregamento** é o último passo que parte da inserção de todos os dados que foram extraídos e transformados no banco de dados. Pode englobar também questões relacionadas a agregação[2] dos dados para otimização das consultas.

2.2.3 Área de apresentação dos dados

A área de apresentação dos dados é onde os dados são organizados, armazenados, e estão disponíveis para qualquer consulta direta para usuários ou aplicações analíticas. Neste ponto os autores Kimball e Ross (2013) tem uma opinião forte onde pontuam que todos os dados são apresentados, armazenados e acessados em **esquemas dimensionais**, eles complementam que a industria concluiu que modelagem dimensional é a técnica mais viável para entregar dados a usuários de data warehouse (Na Subseção 2.2.5 será abordado sobre este conceito).

[1] *Extract, Transform and Load*

[2] Um exemplo de agregação é trocar um campo que possui nomes de estados do Brasil em uma determinadas tabelas para um valor numérico referente a cada estado, depois criar uma tabela auxiliar que possuí o nome dos estados e seus respectivos identificadores numéricos.

2.2.4 Ferramentas de acesso aos dados

Um dos componentes finais e mais relevantes são as ferramentas de acesso aos dados que é um termo utilizado para uma variedade de capacidades que proporcionam para usuários uma área de apresentação para tomadores de decisões analíticas e por definição todas ferramentas de acesso consultam os dados da área de apresentação dos dados. Essas ferramentas podem ser simples como uma ferramenta de consultas *ad hoc*[3] ou complexas como aplicações de mineração de dados ou para modelagem de aplicação.

2.2.5 Modelagem dimensional

Elmasri e Navathe (2011) explica que modelos multidimensionais tiram proveito de relacionamentos inerentes nos dados para preencher os dados em matrizes multidimensionais, que são chamadas de **cubos de dados** (em casos onde essas matrizes tem mias de três dimensões podem ser chamadas de *bi-percubos*). Ele complementa que dados que atendem a essa formação dimensional o seu desempenho em consultas nas matrizes multidimensionais pode ser muito melhor do que em modelos relacionais.

Um exemplo de uma matriz multidimensional pode ser visto na Figura 2.4[4], onde há duas dimensões representando o número de matriculas de alunos da educação básica no Brasil, nas linhas estão representadas as regiões do Brasil, e nas colunas os anos disponíveis.

NÚMERO DE MATRÍCULAS Baixar

Educação Básica

Número de Matrículas por Região - Brasil, 2013 a 2018

Região	2013	2014	2015	2016	2017	2018
Centro-Oeste	3.638.417	3.654.528	3.644.924	3.643.646	3.639.987	3.670.932
Nordeste	14.968.836	14.806.714	14.405.392	14.325.245	14.338.627	14.213.442
Norte	5.144.488	5.131.557	5.071.784	5.030.223	5.010.901	4.992.490
Sudeste	19.806.604	19.705.590	19.236.902	19.350.189	19.144.341	19.074.940
Sul	6.484.103	6.472.982	6.437.510	6.468.176	6.474.237	6.504.063
Total	50.042.448	49.771.371	48.796.512	48.817.479	48.608.093	48.455.867

Fonte: Elaborado pelo Laboratório de Dados Educacionais a partir dos microdados do Censo Escolar/INEP 2013 - 2018

Figura 2.4: Número de Matrículas por Região e Ano

2.3 OLAP

Um dos problemas resolvidos por um Sistema OLAP segundo Silberschatz et al. (2006) está relacionado aos bancos de dados terem um grande volume de dados e por conta disso é necessário a utilização de ferramentas que permitam a derivação desses dados transformando em

[3]Significa destinado a uma finalidade especifica, neste contexto para cada situação ou finalidade uma consulta diferente é criada.

[4]Retirado de: <https://dadoseducacionais.c3sl.ufpr.br/#/indicadores/matriculas>

informações que os humanos possam entender e utilizar. Silberschatz et al. (2006) também traz uma definição sobre o assunto:

> Um sistema de processamento analítico on-line, ou OLAP, **é um sistema interativo que permite que um analista veja diferentes resumos dos dados multidimensionais**. A palavra *on-line* **indica que um analista precisa ser capaz de solicitar novos resumos e obter respostas on-line**, dentro de alguns segundos. e não deve ser forçado a esperar muito tempo para ver o resultado de uma consulta.

Silberschatz et al. (2006) ainda complementa explicando que com um sistema OLAP o analista de dados pode examinar diferentes tabulações cruzadas sobre os mesmos dados, selecionando os atributos na tabulação cruzada. Um exemplo de tabulação cruzada é o da Figura 2.4, onde foi selecionado como dimensão o Ano e a Região, outras dimensões poderiam ser selecionadas como na 2.5[4] com os campos Região e Sexo, por exemplo, com base nessa nova tabulação cruzada é possível identificar que em todos as regiões em 2018 a quantidade de alunos do sexo masculino foi maior do que a do sexo feminino. Silberschatz et al. (2006) define essa operação de mudar as dimensões utilizadas em uma tabulação cruzada de **pivotar**.

NÚMERO DE MATRÍCULAS Baixar

Educação Básica

Número de Matrículas por Região e Sexo - Brasil, 2018

Região	Masculino	Feminino	Total
Centro-Oeste	1.865.351	1.805.581	3.670.932
Nordeste	7.227.114	6.986.328	14.213.442
Norte	2.545.128	2.447.362	4.992.490
Sudeste	9.676.220	9.398.720	19.074.940
Sul	3.319.050	3.185.013	6.504.063
Total	24.632.863	23.823.004	48.455.867

Fonte: Elaborado pelo Laboratório de Dados Educacionais a partir dos microdados do Censo Escolar/INEP 2018

Figura 2.5: Número de Matrículas por Região e Sexo em 2018

O autor comenta sobre mais uma funcionalidade de um OLAP que é chamada de fatia ou fatiamento (*slicing*), onde essa operação pode ser imaginada como uma fatia (*slice*) de um cubo de dados (quando os valores para múltiplas dimensões são fixos essa operação pode ser chamada de *dicing*(corte em cubos), um exemplo dessa operação pode ser visto na Figura 2.6[4] na qual o valor feminino foi selecionado para dimensão sexo.

NÚMERO DE MATRÍCULAS

Baixar

Educação Básica

Número de Matrículas, sexo (feminino) por Região e Sexo - Brasil, 2018

Região	Feminino	Total
Centro-Oeste	1.805.581	1.805.581
Nordeste	6.986.328	6.986.328
Norte	2.447.362	2.447.362
Sudeste	9.398.720	9.398.720
Sul	3.185.013	3.185.013
Total	23.823.004	23.823.004

Fonte: Elaborado pelo Laboratório de Dados Educacionais a partir dos microdados do Censo Escolar/INEP 2018

Figura 2.6: Número de Matrículas por Região para o Sexo feminino em 2018

2.4 API REST

Introduzindo o assunto Mark (2012) define Serviços WEB[5] (*Web Services*) são Servidores WEB construídos com o propósito de suportar as necessidades de um site ou qualquer outra aplicação.

De arcordo com Fielding (2000) REST (Transferência Representacional de Estado ou *Representational State Transfer*) é um estilo de arquitetura para sistemas distribuídos de hipermídia, que descrevem princípios de engenharia de software guiando o REST e as restrições de interações escolhidas para manter esses princípios, que são contrastados com outros estilos arquiteturais. REST é um estilo híbrido derivado de vários outros estilos arquiteturais baseados em rede[6] (Metodologia de Classificação, Fluxo de Dados, Estilos de Replicação, Estilos de Hierarquia, entre outros). No início de seu trabalho Fielding (2000) define um Estilo Arquitetural como um conjunto coordenado de restrições arquiteturais que restringe as funções/recursos dos elementos da arquitetura e os permitem se relacionar entre os elementos de qualquer arquitetura que esteja em conformidade com esse estilo.

Mark (2012) traz uma definição de API RESTful na qual ter o estilo de arquitetura API REST faz um serviço Web ser "RESTful"e ele também traz uma visão concisa sobre essas restrições do estilo arquitetural REST:

- Cliente-Servidor (*Client-Server*)

  Representa a separação das responsabilidades que é o tema principal das restrições Cliente-Servidor Web. A Web é um sistema baseado em Cliente-Servidor, na qual os clientes e servidores tem funções diferentes, que devem ser implementadas e implantadas independentemente, utilizando qualquer linguagem ou tecnologia, desde que estejam em conformidade com a interface uniforme da Web.

- Sem estado (*Stateless*)

  A restrição "Sem Estado"direciona que o servidor não é obrigado a memorizar o estado do cliente de suas aplicações. Como resultado, cada cliente precisa incluir toda informação contextual necessária para cada interação com o servidor Web. Servidores Web precisam que os clientes gerenciem a complexidade da comunicação do estado de sua própria aplicação, para que o servidor Web consiga atender um número grande de clientes, está troca é a chave que contribui para que o estilo arquitetural Web seja escalável.

- Cache[7]

  A "Cache"é uma restrição muito importante pois ela instrui o servidor Web a declarar o que tem *cacheability* (o que pode ser guardado na cache) para cada resposta de dados. Guardar os dados das respostas na Cache auxilia a reduzir a latência percebida pelo cliente, no geral aumenta a disponibilidade e confiabilidade de uma aplicação, e controla o carregamento de um servidor Web. Em linhas gerais a Cache reduz o custo geral da Web.

[5] O termo “World Wide Web” (Rede de Alcance Mundial) é popularmente conhecida como “Web”, na qual as referências dessa seção se referem de acordo com o autor Mark (2012)

[6] Os estilos arquiteturais baseados em rede são explicados por Fielding (2000) em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/net_arch_styles.htm

[7] Em linhas gerais a cache é uma área de armazenamento temporário onde os dados e objetos utilizados (ou que é esperado que eles sejam utilizados no futuro) são guardados para otimizar acessos futuros.Kimball e Ross (2013)

- Sistema em camadas (*Layered System*)

  A restrição de sistema em camada habilita os intermediários baseados em rede, geralmente esses intermediários vão interceptar as comunicações para propósitos específicos. Intermediários baseados em rede são frequentemente utilizados para reforçar a segurança, resposta da cache, e balancear o carregamento.

- Interface Uniforme (*Uniform Interface*)

  As interações entre os componentes Web significativos seus clientes, servidores e intermediários baseados em rede dependem da uniformidade das suas interfaces. Se algum dos componentes não segue os padrões estabelecidos, então a comunicação do sistema quebra.

Mark (2012) define Programas cliente utilizam Interface de Programação de Aplicação (API) para se comunicarem com Serviços WEB. De uma maneira mais genérica o autor comenta que uma API disponibiliza um conjunto de dados e funções para facilitar a interação entre programas de computador e os permitem trocar informações, ou seja uma Web API é a interface para um Serviço Web, na qual ela escuta e responde diretamente as requisições dos clientes.

Um assunto essencial é a forma que acessamos os recursos, Mark (2012) comenta que APIs REST utilizam Identificadores Uniformes de Recurso (URIs) para endereçar os recursos. Atualmente as projeções de URI possuem uma comunicação claro e acessam um modelo de recurso de APIs como:

- http://api.soccer.restapi.org/leagues
- http://api.soccer.restapi.org/leagues/seattle
- http://api.soccer.restapi.org/leagues/seattle/teams

Neste exemplo simples é possível identificar facilmente o conteúdo que está sendo acessado, por exemplo, no primeiro item que o conteúdo é referente as ligas de futebol, no segundo fico mais especifico sobre as ligas de futebol de Seattle, depois sobre os times de futebol presentes na liga de Seattle. Com isso é possível perceber o conceito de hierarquia no modelo dos recursos que são separados por barra (’/’), na qual os recursos ficam mais nítidos para desenvolvedores da parte do cliente que forem utilizar os dados de uma API. O autor traz algumas definições sobre as regras utilizadas para projeção de URIs, uma das regras importantes é a consulta (*query*), onde as consultas são identificadas pelo carácter de interrogação (’?’) e logo após ele vem os parâmetros a serem consultados, no exemplo abaixo o primeiro item retorna os usuários e é intuitivo que o conteúdo a ser retornado no segundo item será os usuários que possuem "role"igual a "admin":

- http://api.restapi.org/users
- http://api.restapi.org/users?role=admin

3 Trabalhos Relacionados

Um dos desafios iniciais deste trabalho foi encontrar outras abordagens semelhantes. Não foi encontrado nenhum trabalho voltado ao tema central que é testes de regressão, o trabalho mais semelhante voltado a área de testes foi o de **Estratégias de testes de oráculo para testes baseados em modelo** dos autores Li e Offutt (2017), que será o único abordado neste capítulo.

É importante ressaltar que o trabalho dos autores traz uma análise de estudos anteriores semelhantes com diferentes tipos de estratégias de teste de oráculo comparado todos os estudos entre si e com a própria abordagem deles. Entre os estudos anteriores que foram abordados por eles referente às estratégias somente um é voltado a aplicações web. Os autores comentam o problema do comparador de teste de oráculo para aplicações Web é como determinar de maneira automática se a aplicação Web tem saídas corretas dado um caso de teste e os resultados esperadas. No trabalho que eles analisaram a estratégia de teste aplicações web onde os comparadores checam os documentos HTML, conteúdos e *tags*.

Na Seção de experimento dos autores há 5 aplicações web que foram utilizadas nas comparações, entretanto 4 dessas aplicações que foram retiradas do mesmo livro *Ammann and Offutt's book* são muito diferentes da abordagem e dos assuntos tratados neste trabalho. A última aplicação é um *Web Service* que processa grandes quantidades de dados utilizando Hadoop, mas por ser um projeto/produto proprietário os autores não puderem entrar em muitos detalhes de como a aplicação funciona ou fornecer a base do código.

Uma das contribuições dos autores foi juntar definições fundamentais de teste de outros autores e adaptá-las para alinhar os termos tratados nos vários trabalhos e projetos analisados por eles. A seguir há algumas definições que auxiliam no entendimento da abordagem deste trabalho que está presente no Capítulo 4:

> Definição 1 (Entradas de teste): As entradas necessárias para completar alguma execução do sistema em teste. Neste contexto, entradas de teste são sequencias de chamadas de método para o sistema em teste, incluindo objetos, parâmetros e recursos necessários.
>
> Definição 2 (Resultados esperados): Os resultados que serão produzidos quando as entradas de teste forem executadas se o programa satisfazer o comportamento desejado.
>
> Definição 3 (Oráculo de teste): Um oráculo de teste fornece os resultados esperados para alguns estados do programa como especificado pela estratégia oráculo de teste. **O oráculo de teste determina se um teste passa comparando o resultado esperado com os resultados atuais**. Oráculos de teste consistem geralmente na asserção que compara os resultados esperados com os resultados atuais.
>
> Definição 4 (Teste): Um teste consistem de entradas de teste e oráculos teste.

Definição 6 (Falha): Uma falha é um comportamento externo ou incorreto com respeito aos requisitos ou outras descrições a respeito do comportamento esperado. Quando um teste executa, um declaração que houve uma falha deve ser alcançada. Sobre as circunstâncias certas, um teste que alcança uma falha deve causar uma falha no software que os testadores podem observar.

Definição 7 (Testes unitários): Testes unitários acessam o software projetando e executando testes para os métodos em suas classes.

Definição 8 (Testes do sistema): Testes do sistema acessam o software projetando e executando testes que capturam o comportamento do sistema. Na indústria desenvolvedores frequentemente escrevem testes unitários e testadores escrevem testes de sistema. Além disso, a distinção entre teste de integração, testes de sistema e testes de aceitação do usuário frequentemente não é clara.

Definição 9 (Requisitos de teste): Requisitos de teste são elementos específicos dos artefatos de software que devem ser satisfeitos ou cobertos.

4 Teste de regressão para API REST baseado em teste de oráculo

Neste capítulo há uma descrição detalhando a abordagem utilizada para solucionar um problema real, nas seções a seguir será explicado sobre o problema solucionado, a arquitetura na qual a solução foi inserida, detalhes sobre a implementação, e utilização da solução no LDE.

4.1 Problema

Durante o desenvolvimento do projeto LDE após ter implementado e validado um conjunto de funcionalidades e indicadores[1] surgiu um problema na qual os dados e resultados desses indicadores já validados anteriormente começaram a apresentar erros, que foram provocados por mudanças (implementações novas, atualizações, etc.) e a medida que o projeto avançava esse tipo de problema voltava a se repetir.

4.2 Arquitetura e processos

A arquitetura da perspectiva do fluxo dos dados onde a abordagem proposta foi utilizada está na Figura 4.1, que traz uma perspectiva mais ampla de como os componentes se comunicam e o papel de cada um para que o sistema como um todo cumpra seu objetivo. Ao longo desta seção será explicado como alguns dos conceitos vistos nos capítulos anteriores são utilizados, algumas características, entre outros detalhes.

4.2.1 Fontes de dados e extração

A fonte dos dados utilizados no LDE estão listados abaixo:

- Microdados do Censo Escolar/INEP[2]
- Microdados do Censo de Educação Superior/INEP
- Pesquisa Nacional por Amostra de Domicílios Contínua (PNAD Contínua)
- IBGE
- Entre outros.

[1] Um exemplo de indicador educacional é o número de matrículas e pode ser visto um exemplo na Figura 2.4

[2] Os microdados fornecidos pelo INEP podem ser encontrados em: <http://portal.inep.gov.br/microdados>

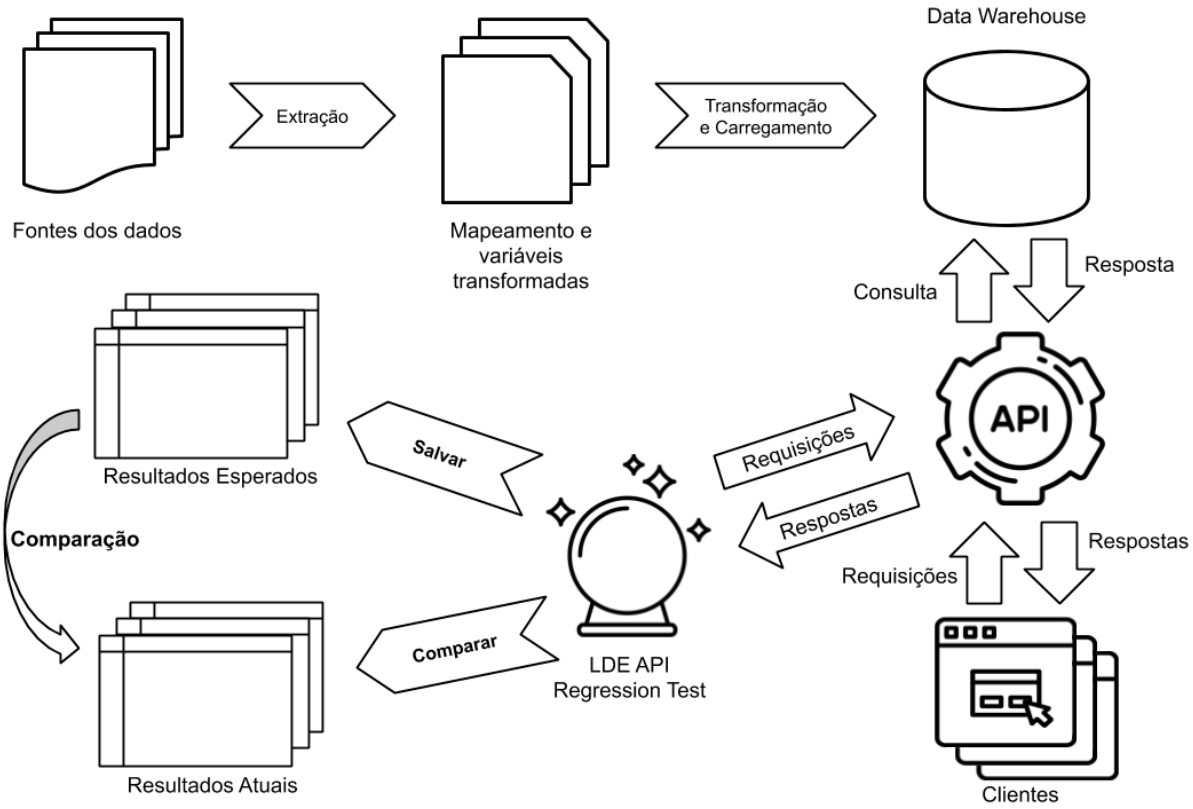


Figura 4.1: Arquitetura do LDE

Fonte: Figura produzida pelo autor deste trabalho

Cada fonte de dados tem suas próprias características, dicionários e principalmente uma periodicidade diferente, da mesma forma o cuidado ao extrair os dados muda de uma fonte de dados para outra. No caso do censo escolar há dados com informações sobre escolas, docentes, matriculas, turmas, entre outras informações. A sua periodicidade de lançamento de novas informações é de 1 ano, ou seja, a cada um ano o processo *ETL* precisa ser realizado, isto implica não só que mais um ano precisará ser extraído e inserido, mas que muitas vezes esses dados precisam ser mapeados novamente pois alguns campos podem ser adicionados, removidos, renomeados e mudanças de tipo.

Um outro exemplo são os dados da PNAD contínua possui dados relacionados a força de trabalho e desenvolvimento socioeconômico no Brasil e está sendo atualizada trimestralmente.

4.2.2 Mapeamento e transformação

Para cada tabela há um arquivo de mapeamento, que possuí para cada variável uma descrição, tipos para cada campo, nomes e transformações para cada ano disponível. Na Figura 4.2 é possível ver o exemplo de mapeamento da tabela de matrículas, na qual cada linha indica uma variável (ou uma coluna no banco de dados). É possível notar uma parte da evolução histórica dos dados dessa tabela nas colunas referentes aos anos de 2013 à 2017.

Na Figura 4.3 é possível notar uma transformação simples para o ano de 2013 onde o campo *'TP_SEXO'* do arquivo fonte nessa ano não tinha os valores *'1'* e *'2'* mas *'m'*, *'M'*, *'f'* ou *'F'*, então foi necessário aplicar essa transformação ao inserir os dados para que essa variável ficasse coerente com os outros anos.

matricula

Var.Lab	Rot.Padrão	Novo Rótulo	Colu	Nome Banco	Tipo de Dado	2013	2014	2015	2016	2017
ANO	NU_ANO_CENSO	Ano do Censo	0	ano_censo	INT	ANO_CENSO	ANO_CENSO	NU_ANO_CENSO	NU_ANO_CENSO	NU_ANO_CENSO
CEBMA002N0	ID_MATRICULA	Código único da matrícula	0	id	INT	PK_COD_MATRICULA	PK_COD_MATRICULA	ID_MATRICULA	ID_MATRICULA	ID_MATRICULA
CEBMA003N0	CO_PESSOA_FISIC	Código do aluno (ID_INEP)	0	cod_aluno	BIGINT	FK_COD_ALUNO	FK_COD_ALUNO	CO_PESSOA_FISI	CO_PESSOA_FISIC	CO_PESSOA_FISIC
CEBMA004N0	NU_DIA	Data de nascimento do alur	0	nasc_dia	TINYINT	NU_DIA	NU_DIA	NU_DIA	NU_DIA	NU_DIA
CEBMA005N0	NU_MES	Data de nascimento do alur	0	nasc_mes	TINYINT	NU_MES	NU_MES	NU_MES	NU_MES	NU_MES
CEBMA006N0	NU_ANO	Data de nascimento do alur	0	nasc_ano	SMALLINT	NU_ANO	NU_ANO	NU_ANO	NU_ANO	NU_ANO
CEBMA007N0	NU_IDADE_REFERE	Idade do aluno no mês de r	0	idade_referencia	TINYINT	NUM_IDADE_REFEREN	NUM_IDADE_REFEREN	NU_IDADE_REFEF	NU_IDADE_REFERI	NU_IDADE_REFER
CEBMA008N0	NU_IDADE	Idade calculada pelo ano de	0	idade	TINYINT	NUM_IDADE	NUM_IDADE	NU_IDADE	NU_IDADE	NU_IDADE
CEBMA010N0	NU_DUR_ATIV_COM	Tempo de permanência (em	0	tempo_mesma_re	INT	NU_DUR_ATIV_COMP_	NU_DUR_ATIV_COMP_I	NU_DUR_ATIV_CO	NU_DUR_ATIV_COI	NU_DUR_ATIV_CO
CEBMA011N0	NU_DUR_ATIV_COM	Tempo de permanência (em	0	tempo_outras_rec	INT	NU_DUR_ATIV_COMP_	NU_DUR_ATIV_COMP_(	NU_DUR_ATIV_CO	NU_DUR_ATIV_COI	NU_DUR_ATIV_CO
CEBMA012N0	NU_DUR_AEE_MES	Tempo de permanência (em	0	tempo_aee_mesn	INT		NUM_DUR_AEE_MESM	NU_DUR_AEE_ME	NU_DUR_AEE_MES	NU_DUR_AEE_ME
CEBMA013N0	NU_DUR_AEE_OUT	Tempo de permanência (em	0	tempo_aee_outra	INT		NUM_DUR_AEE_OUTRA	NU_DUR_AEE_OU	NU_DUR_AEE_OUT	NU_DUR_AEE_OU
CEBMA014N0	TP_SEXO	Sexo	0	sexo	TINYINT	~CASE WHEN "TP_SEX	TP_SEXO	TP_SEXO	TP_SEXO	TP_SEXO
CEBMA015N0	TP_COR_RACA	Cor/raça	0	cor_raca_id	TINYINT	TP_COR_RACA	TP_COR_RACA	TP_COR_RACA	TP_COR_RACA	TP_COR_RACA
CEBMA016N0	TP_NACIONALIDAD	Nacionalidade	0	nacionalidade	TINYINT	TP_NACIONALIDADE	TP_NACIONALIDADE	TP_NACIONALIDA	TP_NACIONALIDAI	TP_NACIONALIDA
CEBMA017N0	CO_PAIS_ORIGEM	Código do país de origem	0	cod_pais_origem	INT	FK_COD_PAIS_ORIGEM	FK_COD_PAIS_ORIGEM	CO_PAIS_ORIGEM	CO_PAIS_ORIGEM	CO_PAIS_ORIGEM

Figura 4.2: Exemplo de mapeamento das variáveis da tabela de matrícula

Fonte: Figura produzida pelo autor deste trabalho

sexo	TINYINT	~CASE WHEN "TP_SEXO"='m' OR "TP_SEXO"='M' THEN 1 when "TP_SEXO"='f' OR "TP_SEXO"='F' THEN 2 END

Figura 4.3: Transformação variável sexo

Fonte: Figura produzida pelo autor deste trabalho

4.2.3 Data warehouse e carregamento

Todo o processo de carregamento dos dados é realizado utilizando a ferramenta Hotmapper que utiliza os arquivos de mapeamento para fazer o carregamento dos dados no Data warehouse. Segundo Henrique Varella Ehrenfried (2019) o Hotmapper é um programa de interface de linha de comandos (*CLI*) que com alguns comandos cria e atualiza tabelas, inserindo e atualizando os dados, utilizando um simples arquivo de definição de mapeamento interpretado pelo programa CLI, armazenando o esquema e os dados mapeados para diferentes anos.

É importante lembrar que o Data Warehouse do LDE possuí as principais características descritas na fundamentação sobre o assunto, onde a tecnologia utilizada é o MonetDB[3] que é um banco de dados colunar, é importante ressaltar que a série histórica em quase todas as tabelas é controlado pela coluna "ano_censo", da mesma forma como comentado anteriormente a periodicidade de lançamento dos dados varia dependendo da fonte, as mais comuns são trimestrais e anuais, isso implica que a periodicidade de inserção dos dados segue a mesma linha, relembrando que são características tipicas de um data warehouse na qual o banco de dados é utilizado majoritariamente para leitura e operações de inserção, atualização e remoção são ocasionais.

4.2.4 API REST

O fluxo da API REST é ilustrado na Figura 4.4, que representa um fluxo simplificado pois explica com foco no funcionamento e não entra em detalhes de arquitetura, cache, entre outras coisas. Iniciando pela requisição (*request*), passando pelo construtor da consulta (*Query Builder*) que monta a consulta a ser feita no banco de dados, passando para o processo de consulta (*Query*) em si que utiliza a consulta montada na fase anterior e realiza a consulta no Data warehouse, depois no processo de transformação do resultado (*Result Transformation*) a

[3]Mais detalhes podem ser encontrados em: <https://www.monetdb.org/Home>

API recebe a resposta do data warehouse e realiza as operações necessárias para enviar os dados, por exemplo, transformar os dados no formato desejado, desta forma o processo chega ao fim onde o cliente que fez a requisição recebe a resposta.

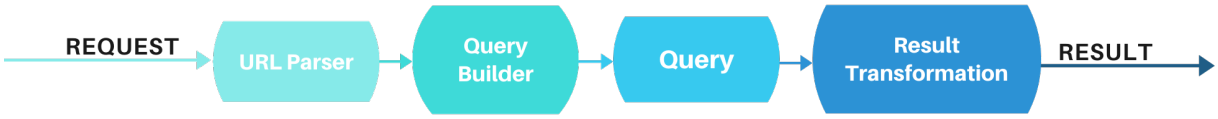


Figura 4.4: Fluxo Simplificado da API REST

Fonte: Figura produzida pelo autor deste trabalho

A Figura 4.5 é um exemplo de requisição onde a primeira informação necessária é qual rota está sendo acessada, neste caso é a rota número de matrículas da educação básica (*enrollment*). O segundo dado é relacionado a quais dimensões estão vão ser utilizadas para o agrupamento dos dados. O terceiro é o campo filtro (*filter*), neste caso os filtros utilizados foram o de ano mínimo e ano máximo, ou seja, os dados retornado serão filtrados e retornaram dados deste intervalo de anos. Vale ressaltar que por conta do filtro ser um intervalo e não um ano especifico o ano é utilizado como uma dimensão. O último parâmetro desta requisição é o formato (*format*) escolhido JSON.

https://simcaq.c3sl.ufpr.br/api/v1/enrollment?
dims=*region*&filter=*min_year:2013*,*max_year:2018*&*format=json*

Figura 4.5: URL da Requisição

Fonte: Figura produzida pelo autor deste trabalho

A Consulta no formato *SQL* construída pela API pode ser visualizada na Figura 4.6, na qual é destacado no seleção *SELECT* os campos relacionados as dimensões escolhidas na requisição (*ano_censo e regiao*), da mesma forma o filtro de 2013 a 2018, o agrupamento e ordenação pelas dimensões.

A API responde em dois formatos JSON e CSV, abaixo é possível visualizar um exemplo da consulta[4] da Figura 4.5, sendo este o mesmo exemplo da consulta no cliente na Figura 2.4.

4.2.5 Cliente

O Cliente não influencia na abordagem mas faz parte da arquitetura, de maneira que através dele o objetivo de um sistema OLAP é alcançado, onde humanos possam entender, utilizar e analisar os dados online. Abaixo há duas referências para os clientes que utilizam a API REST do LDE.

Site do Laboratório de Dados Educacionais (LDE): <`https://dadoseducacionais.c3sl.ufpr.br`>

Site do SIMCAQ (Simulador de Custo-Aluno Qualidade): <`https://simcaq.c3sl.ufpr.br`>

[4]Disponível em: `https://simcaq.c3sl.ufpr.br/api/v1/enrollment?dims=region&filter=min_year:2013,max_year:2018&format=json`

```
SELECT
    COUNT(*) AS "total",
    matricula.ano_censo AS "year",
    regiao.nome AS "region_name"
FROM matricula
INNER JOIN
    regiao ON (matricula.regiao_id=regiao.id)
WHERE
    (matricula.ano_censo >= 2013 )
    AND (matricula.ano_censo <= 2018 )
GROUP BY
    matricula.ano_censo, regiao.nome
ORDER BY
    matricula.ano_censo ASC, regiao.nome ASC
```

Figura 4.6: Query construída pela API

Fonte: Figura produzida pelo autor deste trabalho


```
{
  "result": [
    {
      "total": 3638417,
      "year": 2013,
      "region_name": "Centro-Oeste"
    },
    {
      "total": 14968836,
      "year": 2013,
      "region_name": "Nordeste"
    },
    ...
    {
      "total": 6504063,
      "year": 2018,
      "region_name": "Sul"
    }
  ]
}
```


Figura 4.7: Resposta da API no formato JSON

Fonte: Figura produzida pelo autor deste trabalho

4.3 Implementação

É importante destacar que como foi definido na seção de testes de software por Pressman (2010) os testes de regressão são testes de integração e por conta disso suas definições dão uma direção mais clara pra compreender a abordagem.

Os testes de integração visa a construção da arquitetura de software enquanto ao mesmo tempo se cria testes para cobrir erros relacionados a interface, e da mesma forma o desenvolvimento da estrutura de software e da mesma forma os testes dessas estruturas devem ser guiado pela arquitetura, como a arquitetura do sistema da abordagem foi visto na seção anterior.

A nome escolhido para abordagem é *LDE API Regression Test*, a implementação possui duas funções principais que são ilustradas na Figura 4.1.

Inicialmente há alguns requisitos para utilização da abordagem, ter instalado Python3.5[5] que é a linguagem de programação escolhida, *virtualenv*[6] que permite a utilização de ambientes Python isolados, e o *pip*[7] que é o gerenciador de pacotes padrão para Python. O endereço detalhado do repositório e documentação com instruções de instalação pode ser encontrado no final desta seção.

Para configuração inicial é necessário preencher o arquivo de configuração *settings.py*, onde o **BASE_URL** é URL da API a ser testada, o *BASE_ROUTE_LIST* é um vetor que contém a lista das rotas a serem testadas.

A **função de Salvar** (*save*) é responsável por fazer as requisições nas rotas da API que foram definidas no *settings.py* e salvar as requisições, para facilitar as comparações o formato escolhido para salvar as requisições foi CSV. É importante ressaltar que as rotas presentes na configuração devem ter sido validadas, desta forma as rotas salvas se tornam os resultados esperados. Um exemplo de CSV salvo para rota *enrollment* pode ser visto na Figura 4.8. Vale ressaltar que a definição de entrada de teste presente em Li e Offutt (2017) também fundamenta este formato na qual as entradas de teste são sequencias de chamadas de método para o sistema em teste, incluindo objetos, parâmetros e recursos necessários.

Está função possuí um objetivo essencial nos testes de regressão que é definido por Pressman (2010) na seção de testes de regressão como a captura dos casos de teste e resultados para uma reprodução e comparação posterior.

```
1 python manage.py save
```

Há a possibilidade de salvar uma rota específica, desta forma quando houver atualizações (por exemplo inserção de novos dados) e essas alterações forem validades é possível atualizar somente a rota que foi alterada, fazendo a requisição direta para está rota.

```
1 python manage.py save --route enrollment
```

Voltando na definição de Oráculo de teste do autor Li e Offutt (2017) que diz que um oráculo de teste determina se um teste passa comparando o resultado esperado (gerados pelo método de salvar) com os resultados atuais. Os oráculos de teste consistem geralmente na **asserção** que compara os resultados esperados com os resultados atuais. A **função de comparação** é responsável por refazer ou reproduzir as requisições nas rotas da API e comparar as respostas atuais com os resultados esperados já salvos pela função de salvar, comparando os

[5]Disponível em: <https://www.python.org/

[6]Disponível em: https://virtualenv.pypa.io/en/latest/installation/

[7]Disponível em: <https://pip.pypa.io/en/stable/installing/>

```
,total,year,region_name
0,3638417,2013,Centro-Oeste
1,14968836,2013,Nordeste
2,5144488,2013,Norte
3,19806604,2013,Sudeste
4,6484103,2013,Sul
5,3654528,2014,Centro-Oeste
6,14806714,2014,Nordeste
7,5131557,2014,Norte
8,19705590,2014,Sudeste
9,6472982,2014,Sul
10,3644924,2015,Centro-Oeste
11,14405392,2015,Nordeste
12,5071784,2015,Norte
13,19236902,2015,Sudeste
14,6437510,2015,Sul
15,3643646,2016,Centro-Oeste
16,14325245,2016,Nordeste
17,5030223,2016,Norte
18,19350189,2016,Sudeste
19,6468176,2016,Sul
20,3639987,2017,Centro-Oeste
21,14338627,2017,Nordeste
22,5010901,2017,Norte
23,19144341,2017,Sudeste
24,6474237,2017,Sul
25,3670932,2018,Centro-Oeste
26,14213442,2018,Nordeste
27,4992490,2018,Norte
28,19074940,2018,Sudeste
29,6504063,2018,Sul
```

Figura 4.8: Resposta da API no formato CSV

Fonte: Figura produzida pelo autor deste trabalho

arquivos e fazendo uma asserção para identificar se há diferenças e retornar-las. Está função exerce um papel central dos testes de regressão definidos por Pressman (2010) onde no contexto de testes de integração o teste de regressão é uma reexecução de algum conjunto de testes que já foram realizados anteriormente para assegurar que as mudanças não propagem efeitos colaterais indesejados.

```
python manage.py compare
```

A resposta da execução do teste é uma lista com todas as rotas, as rotas que passarem no teste, ou seja, a sua versão salva é a mesma que a versão atual aparecem "OK"ao lado, as que falharam aparece "FAIL"e caso alguma rota venha a falhar o programa retorna erro. Uma opção ao executar o comando *compare* é utilizar o *--verbose"* e a diferença dos arquivos que falharam é exibida.

Quando uma rota falha significa que a resposta dessa rota não está de acordo com os resultados esperados e alguma mudanças pode ter propagado efeitos colaterais indesejados no sistema.

O comando abaixo pode ser executado para comparar rotas especificas e a opção *verbose* mostra as diferenças caso a rota venha a falhar.

```
python manage.py compare --route enrollment --verbose
```

Repositório e documentação completa do *LDE API Regression Test*: <https://github.com/C3SL/lde-api-regression-test>

4.4 Experimentos

Um dos pontos importantes abordados é que para inserir as rotas nos testes de regressão os dados precisam ter sido validados, isso significa que no fluxo de trabalho atual há um processo realizado por uma das partes interessadas (do projeto LDE) que possuí o domínio do negócio (de dados abertos educacionais), na qual os resultados esperados são calculados com base nos dados fonte originais e a partir disso o resultado mostrado nos clientes são verificados com esses cálculos realizados pelos responsáveis e se os resultados estiverem corretos pode ser considerado que está valido.

No contexto do LDE, a abordagem foi utilizada para fazer parte da Integração Continua no Gitlab (*Gitlab-CI*)[8] que já possuía testes unitários. A ideia principal do Gitlab CI é a cada pedaço de código que vai sendo adicionado em uma aplicação hospedada em um repositório no Git (ferramenta utilizada para versionamento de código) o *Pipeline* é responsável por rodar uma série de *scripts* para construir um ambiente do zero, testar e validar as mudanças de código antes dessas mudanças serem realizadas no ramo principal de código.

Nos exemplos reais que veremos a seguir é importante ressaltar que o *Pipeline* é um ambiente configurado para execução da API REST e especificamente para o *LDE API Regression Test*. Um exemplo de comandos executados pelo CI no *Pipeline* para execução do teste:

```
1  npm install --global gulp gulp-cli babel babel-cli babel-core ...
2  npm install
3  gulp build && gulp &
4  curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
5  python get-pip.py
6  sleep 60
7  git clone https://gitlab.c3sl.ufpr.br/simcaq/lde-api-regression-test.git
8  cd lde-api-regression-test
9  pip install -r requirements.txt
10 python manage.py compare --verbose
```

Na Figura 4.9 um exemplo onde todos os testes passaram e não foi detectado nenhuma falha no código a ser submetido, o teste completo com configuração de ambiente executando todos os comandos acima dura 4 minutos e 48 segundos, e a execução encerra com sucesso.

No exemplo da Figura 4.10 foi detectado falha em 3 rotas e a diferença do resultado esperado para o resultado atual é exibido no teste para cada uma das rotas, e a execução retorna com falha.

No fluxo de desenvolvimento que já estava sendo utilizado pela equipe do LDE a cada alteração realizada a ser acrescentada no ramo de desenvolvimento os testes unitários são executados, nas Figuras que vimos nessa Seção isso é ilustrado pelo outro *Pipeline* "run_tests".

Um ponto a ser destacado é que está abordagem possibilita um **processo de desenvolvimento ideal realizando uma integração incremental** comentado por Pressman (2010) na subseção de testes de integração, na qual o software é construído e testado em pequenos incrementos e os erros são mais fáceis de serem isolados e corrigidos, tornando as interfaces (como a API) mais fáceis de serem testadas por completo. Isso pode ser expressado pela forma

[8]Mais detalhes sobre o funcionamento do Gitlab CI pode ser encontrado aqui: https://docs.gitlab.com/ee/ci/

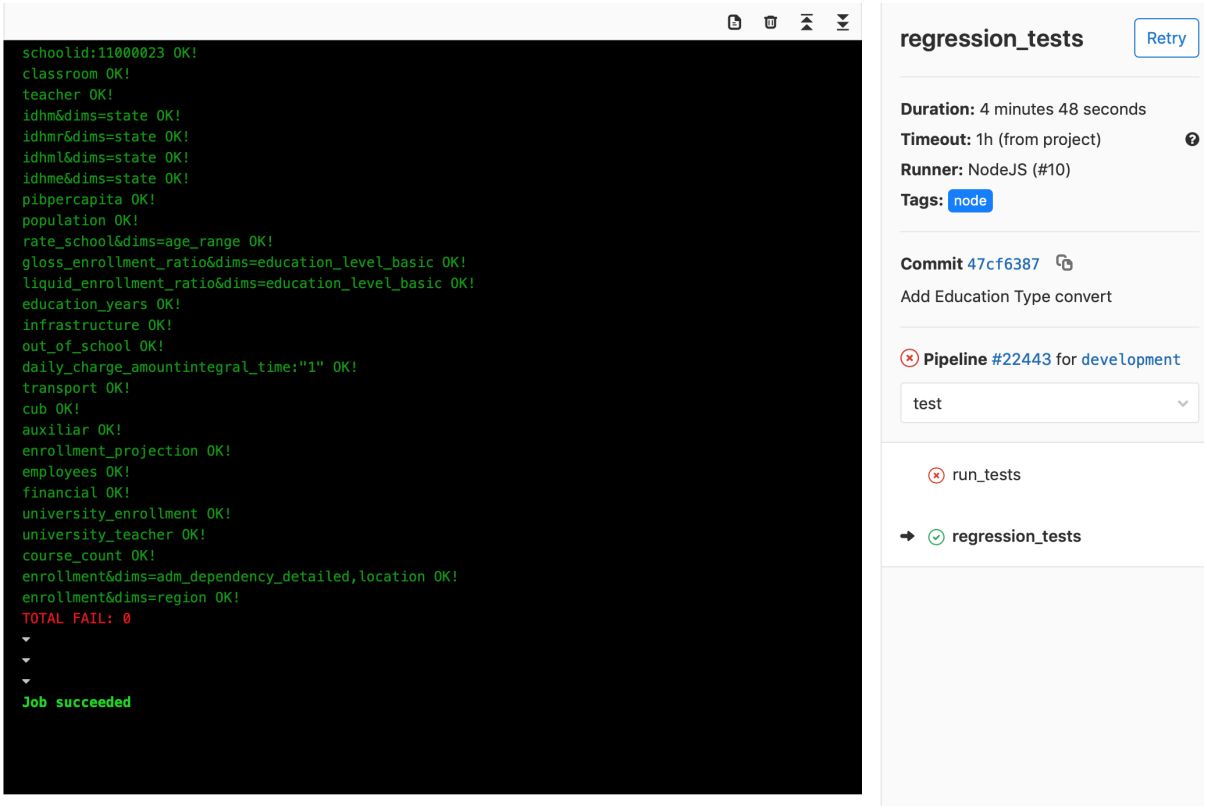


Figura 4.9: Exemplo de teste com nenhuma falha detectada do Regression Test no GitlabCI do LDE

Fonte: Figura produzida pelo autor deste trabalho

em que as funções de salvar e comparar funcionam, onde a medida que novas rotas são criadas e validadas elas podem ser incluídas no teste de regressão incrementalmente.

Embasado todo o contexto de como as partes do sistema se comunicam e um pouco sobre como é o fluxo de teste durante o desenvolvimento, um pergunta importante a ser respondida é: que tipo de falhas ou erros o teste de regressão desta abordagem detecta?

Relembrando que o Pressman (2010) comenta que os testes de regressão em um contexto mais amplo vão ser bem sucedidos se os resultados descobrirem erros, e a partir disso os erros devem ser corrigidos, e todas as vezes que o software é corrigido algo da sua configuração, como o programa, documentação, ou os dados que suportam o software, o software é alterado. Nesta mesma linha **os testes de regressão contribuem auxiliando para que mudanças não gerem comportamentos indesejados ou erros adicionais**.

As falhas são definidas por Li e Offutt (2017) como comportamentos externos ou incorretos relacionados aos requisitos ou outras descrições a respeito do comportamento esperado.

O primeiro tipo de falha está diretamente relacionado com alguns problemas que Pressman (2010) comenta que podem ocorrer, como componentes que podem ter um efeito indesejado em outro, subfunções combinadas que não cumprem sua função global desejada, imprecisões individuais dos módulos que são permitidas propagam uma imprecisão em níveis não aceitáveis quando combinados, estruturas de dados globais podem apresentar problemas, entre outras coisas. Esses tipos de erros estão relacionado as regras de negócio implementadas na API REST que são abstraídas como componentes na literatura. Na qual se alguma alteração nova provoca uma mudança indesejada ou efeitos colaterais nos resultados que já foram validados então há algo errado nesta alteração e ela deve ser analisada e não poderia ser incluída no ambiente de

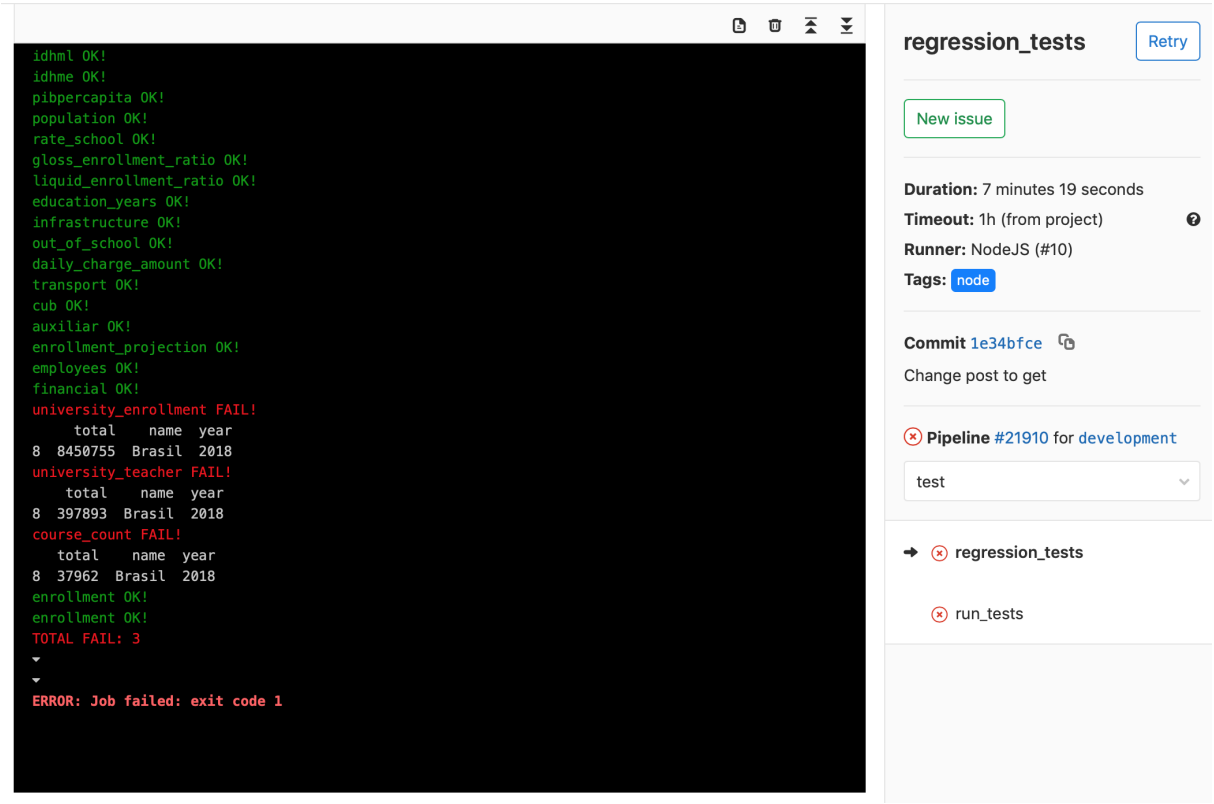


Figura 4.10: Exemplo de teste com detecção de falhas do Regression Test no GitlabCI do LDE

Fonte: Figura produzida pelo autor deste trabalho

desenvolvimento. Pois consequentemente se a alteração for aplicada, ela provavelmente em um momento posterior será incluída no ambiente de produção que é onde os usuários finais acessam o sistema, representando que esses usuários acessaram dados incorretos na qual foi originado pelas mudanças indesejadas.

O segundo tipo de falha está relacionado a problemas que Pressman (2010) define como **os dados que suportam o software**, que está associado com a detecção de falhas nos dados que significa que algo relacionado ao processo de extração, transformação e carregamento ocasionou as mudança indesejada ou efeitos colaterais nos resultados que já foram validados e da mesma forma deve ser investigado de acordo com as rotas que apresentaram esses erros.

Repositório da API REST do LDE: <https://gitlab.c3sl.ufpr.br/simcaq/simcaq-node>

5 Conclusão

Este trabalho apresentou um oráculo de testes com testes de regressão fundamentando testes de software, teste de oráculo e todas as partes que englobam a arquitetura da API RESTful utilizada no experimento. O oráculo permitiu verificar quando mudanças provocam comportamentos indesejados ou falhas em rotas já validadas, com aplicação no LDE.

Um dos relatos mais interessantes foi o de uma das partes interessadas pelo projeto que era responsável por fazer a validação dos dados, onde esse processo de re-validação dos dados precisava ser feito com alguma periodicidade para garantir que os dados continuavam corretos mesmo depois da evolução e avanço do desenvolvimento do sistema, gastando horas e até dias pra conseguir fazer esse processo de re-validação dos dados através dos Clientes (Site do LDE e do SIMCAQ). Atualmente a abordagem proposta nesse trabalho realiza este processo, como foi possível ver na seção anterior as rotas principais da API são re-validadas em poucos minutos.

Alguns pontos que poderiam ser melhorados na abordagem diretamente na implementação são a possibilidade de utilizar outros formatos além de CSV para salvar e comparar as respostas da API, contabilizar o tempo que cada rota leva para responder uma requisição e mais voltado a desempenho paralelizar o processo de salvar e comparar para que as rotas de uma API pudessem ser testadas paralelamente.

Como principal utilidade da abordagem acontece durante o período de desenvolvimento, vale ressaltar que seria interessante como trabalho futuro mensurar anteriormente a utilização da abordagem o quanto é gasto de esforço pra detectar mudanças indesejadas e efeitos colaterais e até mesmo quanto tempo leva pra essa detecção. Este questionamento poderia ser respondido de forma precisa mensurando o esforço e tempo gasto, possibilitando equipes de desenvolvimento em sistemas com características semelhantes a mensurarem a efetividade para detectar esse tipo de mudança indesejada e avaliar se o tempo de resposta a esse tipo de problema pode melhorar consideravelmente adotando o teste de oráculo proposto neste trabalho.

As contribuições relevantes para trabalhos futuros estão relacionadas a análise das detecções das falhas em projetos em andamento com essas características, levando em consideração o tipo de erro (implementação das regras de negócio, banco de dados, entre outros), a frequência e como a detecção de erros provocados por mudanças indesejadas em dados já validados influencia no desenvolvimento de um sistema. Como não foram encontrados outros trabalhos semelhantes um possível trabalho futuro seria a utilização de testes de regressão em sistemas com características diferentes, sejam características diferentes na arquitetura ou no tipo dos dados utilizados pelo sistema.

Um outro trabalho interessante que pode ir um pouco na linha do trabalho relacionado de teste de oráculo é testar a escalabilidade dos testes em sistemas mais complexos ou que possuem uma quantidade de rotas maior, para entender o quanto é efetivo testar todas as rotas ou só as principais rotas na detecção de erros.

Referências

Date, C. J. (2004). *Introdução a sistemas de bancos de dados*. Campus.

Elmasri, R. e Navathe, S. B. (2011). *Sistemas de banco de dados*. Pearson Addison Wesley.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*.

Henrique Varella Ehrenfried, Rudolf Eckelberg, H. I. E. T. D. W. M. D. D. F. (2019). Hotmapper: Historical open data table mapper. *EDBT 2019*, páginas 550–553.

Kimball, R. e Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley Publishing, 3rd edition.

Li, N. e Offutt, J. (2017). Test oracle strategies for model-based testing. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 43(4):372–395.

Mark, M. (2012). *REST API design rulebook*. OReilly.

Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, USA, 7 edition.

Silberschatz, A., Korth, H. F., Sudarshan, S., Souza, L. F. P. d. e Vieira, D. (2006). *Sistema de banco de dados*. Elsevier.