UNIVERSIDADE FEDERAL DE ALFENAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**Rafael Alves Menicucci Pinto**

**Pedro Seiti Mazine Kiyuna**

# *Test&Code*: Uma Aplicação Web Gratuita e de Código Aberto para Correção Automatizada de Avaliações de Programação Orientada a Objetos

Alfenas/MG

2020

**Rafael Alves Menicucci Pinto**

**Pedro Seiti Mazine Kiyuna**

# *Test&Code*: Uma Aplicação Web Gratuita e de Código Aberto para Correção Automatizada de Avaliações de Programação Orientada a Objetos

Trabalho apresentado como parte dos requisitos para a disciplina de Trabalho de Conclusão de Curso pelo curso de Ciência da Computação da Universidade Federal de Alfenas - UNIFAL-MG.

Orientador: Rodrigo Martins Pagliares.

Alfenas/MG

2020

**Rafael Alves Menicucci Pinto**

**Pedro Seiti Mazine Kiyuna**

***Test&Code*: Uma Aplicação Web Gratuita e de Código Aberto para Correção Automatizada de Avaliações de Programação Orientada a Objetos**

A Banca examinadora abaixo-assinada, aprova o Trabalho apresentado como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

---

**Prof. Rodrigo Martins Pagliares**
**Universidade Federal de Alfenas**

---

**Prof. Luiz Eduardo da Silva**
**Universidade Federal de Alfenas**

---

**Prof. Flávio Barbieri Gonzaga**
**Universidade Federal de Alfenas**

Alfenas/MG

2020

# RESUMO

A correção automatizada de avaliações visa beneficiar professor e aluno. O professor passa a não ter a necessidade de corrigir cada avaliação. O aluno se beneficia por meio da possibilidade de se obter resultado imediato da sua nota. Embora existam várias ferramentas para correção automatizada de avaliações, grande parte delas, além de não serem gratuitas e de código-aberto, faz correção apenas de questões de múltipla-escolha, mas não faz correção de avaliações com código orientado a objetos. Este trabalho apresenta uma ferramenta *web* gratuita e de código-aberto, nomeada *Test&Code*, para correção automatizada de avaliações de programação orientada a objetos. A ferramenta se baseia em testes de unidade automatizados para a correção dos códigos dos alunos. Este trabalho apresenta uma revisão da literatura sobre ferramentas para correção automatizada de avaliações dos mais diversos tipos (múltipla escolha, completar lacunas, códigos orientado a objetos, etc.). O resultado da revisão da literatura é usado para justificar a proposta de desenvolvimento da *Test&Code*. Desenvolvemos com sucesso uma ferramenta *web* que usa testes automatizados para a correção de avaliações com código orientado a objetos, tendo resultados satisfatórios tanto em testes simples quanto em exemplos mais complexos. A partir dos resultados deste trabalho podemos concluir que, apesar da necessidade da criação de testes automatizados pelo professor para uso da *Test&Code*, acaba sendo uma forma de substituir o gabarito, economizando tempo de correções manuais pelo professor e beneficiando o aluno com a possibilidade de se obter uma resposta imediata de sua nota.

**Palavras-Chave:** correção automatizada, orientação a objetos, construção (*build*) automatizada de *software*, ferramenta *web*.

# ABSTRACT

The automated correction of assessments aims to benefit both teacher and student. The teacher no longer needs to correct each assessment. The student benefits from the possibility of obtaining an immediate result of his grade. Although there are several tools for automated assessments correction, most of them, in addition to not being free and open-source, do only correction of multiple-choice questions, but do not correct assessments with object-oriented code. This work presents a free and open source web tool, called Test&Code, for automated correction of object-oriented programming assessments. The tool is based on automated unit tests to correct student codes. This work presents a literature review on tools for automated correction of assessments of the most diverse types (multiple choice, filling in gaps, object-oriented code, etc.). The result of the literature review is used to justify the proposal for the development of *Test&Code*. We have successfully developed a web tool that uses automated tests to correct assessments with object-oriented code. We obtained satisfactory results both in simple tests and in more complex examples. From the results of this work we can conclude that, despite the need for the creation of automated tests by the teacher for use of *Test&Code*, it ends up being a way to replace the answer key, saving time for manual corrections by the teacher and benefiting the student with the possibility of getting an immediate response to your grade.

**Keywords**: automated correction, objected oriented, automated software build, web tool.

## LISTA DE FIGURAS

FIGURA 1 - Estrutura básica do MVC …………………………………………………….....17

FIGURA 2 - Exemplo do funcionamento MVC na ferramenta *Test&Code* ………………......18

FIGURA 3 - Template da arquitetura via Diagrama de Implantação (*Deployment*) UML......19

FIGURA 4 - Interface gráfica usada para criação de uma conta de *Professor* ………...……23

FIGURA 5 - Interface gráfica usada pelo *Aluno*, pelo *Professor* e pelo *Administrador* para autenticação na *Test&Code* …………………………………………………………..24

FIGURA 6 - Interface gráfica usada pelo *Professor* para criação de disciplinas………..…..24

FIGURA 7 - Interface gráfica usada pelo *Professor* para adição de alunos à disciplina……..25

FIGURA 8 - Projeto compactado com os testes para serem aplicados na avaliação……......25

FIGURA 9 - Interface gráfica usada pelo *Professor* para *upload* do projeto ……...………...26

FIGURA 10 - Interface gráfica da disciplina criada pelo *Professor* ………………..……..26

FIGURA 11 - Interface gráfica usada pelo *Professor* para criação de avaliações.…………...26

FIGURA 12 - Interface gráfica usada pelo *Aluno* para acesso à avaliação …..……………..27

FIGURA 13 - Relação entre o conteúdo típico de uma avaliação disponibilizada pelo *Professor* (à esquerda na figura) e o conteúdo necessário que o aluno deve ter antes da submissão de sua resposta (à direita na figura) ………………………………..…………..28

FIGURA 14 - Interface gráfica para disponibilização da nota do *Aluno* ……………………29

FIGURA 15 - Interface gráfica usada pelo *Professor* para visualização de notas dos alunos………………………………………………………………………………………29

FIGURA 16 - Gráfico disponível para o papel *Administrador* na *Test&Code*.……………....30

FIGURA 17 - Repositório da *Test&Code* no Github (https://bit.ly/2IUwNat) ………......44

FIGURA 18 - Exemplo completo de avaliação enviada pelo *Professor* …………...………45

FIGURA 19 - Diagrama de classes do sistema PDV entregue aos alunos como parte do enunciado da avaliação presencial e manuscrita …………..……………………………...52

## LISTA DE QUADROS

QUADRO 1 - Tipos de questões corrigidas e licenciamento das ferramentas/plataformas analisadas ……………………………………………………………………………………12

QUADRO 2 - Objetivos dos principais papéis identificados para a *Test&Code* …………….14

QUADRO 3 - Principais requisitos funcionais e não-funcionais implementados na *Test&Code*…………………………………………………………………..…………..15

QUADRO 4 - *User Stories*………………...………………………………………………16

QUADRO 5 - Histórias de usuário criadas para o papel *Professor* ……………...…………..41

QUADRO 6 - Histórias de usuário criadas para o papel *Aluno* …………………...……....42

QUADRO 7 - Histórias de usuário criadas para o papel *Administrador* ………..…...............43

## LISTA DE CÓDIGOS

CÓDIGO 1 - Função de criação de avaliação por parte do *Professor* ……………………..…..…...31

CÓDIGO 2 - Função de envio da solução do aluno e cálculo de sua nota…………………...33

CÓDIGO 3 - Classe *endereçotest* …………………………………………………………53

CÓDIGO 4 - Classe *lojatest* ……………………………………………………………….54

## SUMÁRIO


| | | |
|---|---|---|
| **1** | **INTRODUÇÃO**…………………………………………………………..... | 9 |
| **2** | **REVISÃO BIBLIOGRÁFICA**……………………………………………… | 10 |
| **3** | ***TEST&CODE*: PRINCIPAIS INTERESSADOS, REQUISITOS E ARQUITETURA**……………………………………………………….. | 13 |
| 3.1 | VISÃO……………………………………………………………………… | 13 |
| 3.2 | STAKEHOLDERS…………………………………………………………. | 13 |
| 3.3 | REQUISITOS……………………………………………………………… | 14 |
| 3.4 | *USER STORIES*………………………………………………………. | 16 |
| 3.5 | ARQUITETURA DA *TEST&CODE*………………………………………. | 17 |
| **4** | **TECNOLOGIAS**……………………………………………………….. | 20 |
| 4.1 | TECNOLOGIAS USADAS NO *FRONTEND* DA *TEST&CODE*……….. | 20 |
| 4.2 | TECNOLOGIAS USADAS NO *BACKEND* DA *TEST&CODE*…………. | 20 |
| **5** | **EXEMPLO DE USO**……………………………………………………… | 22 |
| 5.1 | INTRODUÇÃO AO EXEMPLO……………………………………….... | 22 |
| 5.2 | EXEMPLO DE USO - CADASTRO E AUTENTICAÇÃO…….………… | 23 |
| 5.3 | EXEMPLO DE USO - *PROFESSOR*………………………………………. | 24 |
| 5.4 | EXEMPLO DE USO - *ALUNO*……………………………………………. | 27 |
| 5.5 | EXEMPLO DE USO - VISUALIZAR NOTAS……………………….…... | 28 |
| 5.6 | EXEMPLO DE USO - *ADMINISTRADOR*……………………………..… | 29 |
| **6** | **DETALHES DA IMPLEMENTAÇÃO**………………………………… | 31 |
| **7** | **DISCUSSÕES**…………………………………………………..…….. | 35 |
| **8** | **CONCLUSÃO E TRABALHOS FUTUROS**……………………………. | 37 |
| | **REFERÊNCIAS**……………………………………………………….... | 39 |
| | | |
| | **APÊNDICES**………………………………………………………….. | 41 |
| | **APÊNDICE A - HISTÓRIAS DE USUÁRIO**……………………………. | 41 |
| | **APÊNDICE B - REPOSITÓRIO DA *TEST&CODE***…………………..... | 44 |
| | **APÊNDICE C - EXEMPLO DE AVALIAÇÃO ENVIADA PELO PROFESSOR**........................................................................................ | 45 |
| | | |
| | **ANEXOS**………………………………………………………………… | 46 |
| | **ANEXO A - ENUNCIADO E DIAGRAMA DE CLASSES DO SISTEMA PDV USADO COMO AVALIAÇÃO MANUSCRITA**………..……….. | 46 |
| | **ANEXO B - EXEMPLOS DE GABARITOS DO SISTEMA PDV UTILIZADO PARA VALIDAR A FERRAMENTA *TEST&CODE***….. | 53 |

# 1 INTRODUÇÃO

A correção automatizada de avaliações de códigos orientados a objetos, sendo elas qualquer atividade avaliativa, como por exemplo provas e trabalhos, é um recurso que exclui a necessidade da correção manual de cada avaliação, possibilitando a geração de resultados imediatos, ou seja, tem o intuito de beneficiar tanto o professor quanto o aluno.

Encontramos na literatura muitas ferramentas relacionadas à correção automatizada de avaliações como Moodle (Moodle, 2002), Google Classroom (Google Classroom, 2014), *moJEC* (Lückemeyer, 2017), *CodeRunner* (Lobb, Harlow 2016), *MestreGR* (MestreGR, 2010), *Prova Fácil* (Prova Fácil, 2012), *Merrit* (Merrit, 2008), SAC (SAC, 2008), Coursera (Coursera, 2012), edX (edX, 2012), Udemy (Udemy, 2010), Alura (Alura, 2013), Linkedin Learning (Linkedin Learning, 2002) e Udacity (Udacity, 2011). Porém, a maioria, além de serem aplicações pagas e não serem de código-aberto, fazem correção somente de questões de múltipla-escolha, não apresentando a funcionalidade de correção de códigos orientado a objetos.

Além disso também tivemos outra motivação com a pandemia da COVID-19, que surgiu no ano da confecção desta monografia, fazendo com que as aulas fossem lecionadas a distância, se encaixando com o escopo deste trabalho.

Este trabalho apresenta uma ferramenta de *software* para a *web,* gratuita e de código-aberto, nomeada *Test&Code*, para correção automatizada de avaliações de programação orientada a objetos. A ferramenta se baseia em testes de unidade automatizados, criados pelo professor, para a correção dos códigos dos alunos, ainda que o professor tenha que codificar os testes para serem utilizados como gabarito nas avaliações

O restante deste trabalho está organizado da seguinte maneira: no Capítulo 2 apresentamos a revisão bibliográfica. O Capítulo 3 apresenta os principais requisitos da *Test&Code* juntamente com sua arquitetura. No Capítulo 4 apresentamos as tecnologias que foram utilizadas para a criação da *Test&Code*. O Capítulo 5 mostra um exemplo de uso. O Capítulo 6 apresenta detalhes de implementação dos principais requisitos da *Test&Code*. O Capítulo 7 apresenta as discussões sobre este trabalho. Conclusões e trabalhos futuros são apresentados no Capítulo 8.

# 2 REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta uma revisão da literatura sobre ferramentas para correção automatizada de avaliações.

Nossa experiência possibilita identificar que em um curso relacionado a área da computação é gerada uma quantidade enorme de linhas de código por semestre, por todas turmas e discentes, o que dificulta o trabalho de correção do professor e faz com que o discente não tenha um *feedback* imediato com os resultados das suas avaliações. Segundo Auffarth et. al. (SAC, 2008) "Muitos dos estudantes notam a importância do resultado em tempo real para o aprendizado".

Durante a revisão da literatura, encontramos algumas ferramentas para correção automatizada de avaliações: Moodle, Google Classroom, *moJEC*, *CodeRunner*, *MestreGR*, *Prova Fácil*, *Merrit* e SAC. Além disso, existem MOOCS (Massive Open Online Courses) contendo cursos *online* gratuitos e pagos, como o Coursera e edX bem como outras plataformas de ensino tais como Udemy, Alura, Linkedin Learning e Udacity.

Ambos o Moodle e o Google Classroom são sistemas de gerenciamento de conteúdo, podendo se criar disciplinas bem como também avaliações com questões objetivas e dissertativas simples.

O *moJEC* e o *CodeRunner* são módulos de extensão gratuitos para o *Moodle* que corrigem automaticamente códigos de programação. O moJEC se especializa em códigos *Java*, utilizando-se de uma classe de teste com *JUnit* (JUnit, 1998), desta maneira é a ferramenta que mais se assemelha a este trabalho. No *CodeRunner* é necessário que o professor declare quais as entradas e as saídas esperadas para as questões da avaliação, sem a necessidade de uso de um *framework* de testes de unidade como o *JUnit*, possibilitando os testes em qualquer linguagem de programação.

O *MestreGR*, *Prova Fácil* e *Merrit* são sistemas pagos que possibilitam o gerenciamento de avaliações. Em todos os sistemas de *software* analisados, observamos o suporte à correção de questões objetivas e questões dissertativas. Porém, somente o MestreGR oferece suporte para correções de redações. Em todos os sistemas o aluno tem retorno imediato de seu desempenho por meio de dados compilados e gráficos.

Coursera é uma plataforma de cursos *online* que utiliza-se de um sistema de correção

automatizada de questões objetivas e de questões contendo códigos. O sistema é utilizado por diversos cursos oferecidos na plataforma. A correção automatizada de códigos no Coursera pode ser obtida de forma simples e com retorno imediato da nota para o aluno ou de maneira complexa com retorno em questão de minutos. Correções simples são baseadas apenas no comparativo entre a saída produzida pelo código do aluno com a saída esperada pelo professor (gabarito). Correções mais complexas analisam de forma automatizada, por exemplo, legibilidade, consumo de memória e desempenho do código entregue pelo aluno.

O SAC é uma plataforma para testes e validação automatizada de códigos orientado a objetos em que os professores especificam e definem os exercícios e os alunos enviam suas soluções, recebendo retorno imediato sobre a validade do código e estatísticas sobre cada exercício realizado.

O edX, Udemy, Alura, LinkedIn Learning e Udacity são plataformas que disponibilizam cursos *online* das mais diversas áreas, em que é possível a criação de avaliações. Em todas elas é possível criar questões objetivas, porém, apenas em algumas, por exemplo, Udemy e Alura, é permitido questões dissertativas ou questões que envolvam correção de código.

O Quadro 2.1 apresenta um resumo das ferramentas/plataformas analisadas juntamente com os tipos de questões suportadas e licenciamento, sendo que todas elas permitem a criação de avaliações pelos professores. Os resultados apresentados para as ferramentas/plataformas marcadas com um (*) são com base no melhor dos esforços a partir da nossa experiência de uso das ferramentas/plataformas, tendo em vista que não encontramos na literatura sustentação para as conclusões apresentadas no quadro.

| | Questões Objetivas | Questões Dissertativas | Correção Códigos | Código Aberto | Gratuita |
|---|---|---|---|---|---|
| **Google Classroom** | X | X | | | X |
| **Moodle** | X | X | | X | X |
| **MoJEC** | | | X | X | X |
| **CodeRunner** | | | X | X | X |
| **MestreGR** | X | X | | | |
| **Prova Fácil** | X | X | | | |
| **Merrit** | X | X | | | |
| **SAC** | | | X | | |
| **Coursera*** | X | X | X | | |
| **edX*** | X | | | | |
| **Udemy*** | X | X | X | | |
| **Alura*** | X | X | | | |
| **Udacity*** | X | | X | | |
| **Linkedin Learning** | X | | | | |

**Quadro 2.1 - Tipos de questões corrigidas e licenciamento das ferramentas/plataformas analisadas.**
**Fonte: Quadro criado pelos autores.**

# 3 *TEST&CODE*: PRINCIPAIS INTERESSADOS, REQUISITOS E ARQUITETURA

Neste capítulo apresentamos a visão, os principais interessados (*stakeholders*), os requisitos e a arquitetura utilizada para desenvolvimento da *Test&Code*.

## 3.1 VISÃO

No início deste trabalho, descrevemos textualmente a Visão (LARMAN, 2007) que comunica em termos sucintos as principais características da *Test&Code* no intuito de direcionarmos os esforços de desenvolvimento e delimitarmos o escopo deste trabalho:

*"Para professores e alunos que realizam avaliações com códigos orientado a objetos, a Test&Code é uma ferramenta em software que realiza a correção automática de avaliações de programação orientada a objetos, gerando resultados imediatos para o aluno e reduzindo o trabalho do professor. Diferentemente da maioria de ferramentas em software disponíveis no mercado, a Test&Code é gratuita de código aberto."*

## 3.2 STAKEHOLDERS

Os *stakeholders* são pessoas ou organizações que possuem interesses no sucesso do desenvolvimento de um produto ou serviço, seja na área de *software* ou não. Um *papel* é uma função que determinado *stakeholder* desempenha em uma ferramenta de *software* no sentido de realizar seu objetivos de negócio. O Quadro 3.2 apresenta os principais papéis e seus respectivos objetivos com relação à *Test&Code*.

| Papel | Objetivos |
| --- | --- |
| *Professor* | Seu principal objetivo é avaliar o desempenho dos discentes por meio da aplicação de avaliações e divulgação das notas. Outros objetivos incluem criar disciplinas, adicionar discentes à(s) disciplina(s), criar avaliações, etc. |
| *Aluno* | Os principais objetivos do *aluno* são realizar a avaliação proposta pelo professor e visualizar suas notas. |
| *Administrador* | Seus objetivos incluem visualizar gráficos de utilização (número de professores usando a *Test&Code*, número de disciplinas cadastradas, etc) e conceder permissões de administrador para outros usuários. |

**Quadro 3.2 Objetivos dos principais papéis identificados para a *Test&Code*.**
**Fonte: Quadro criado pelos autores.**

## 3.3 REQUISITOS

Requisitos são as descrições das funções e restrições que o produto a ser desenvolvido deve possuir e normalmente são classificados como Requisitos Funcionais e Requisitos Não-Funcionais (SOMMERVILLE, 2011). Requisitos funcionais são descrições das funcionalidades de *software* do ponto de vista de um usuário desempenhando um papel durante seu uso. Requisitos não-funcionais são relacionados às propriedades do sistema e abordam aspectos de qualidade que quando ignorados podem comprometer a consistência e usabilidade do software.

O Quadro 3.3 apresenta os principais Requisitos Funcionais (RF) e Não-Funcionais (NRF) da *Test&Code*. Cada requisito possui um identificador único. Observe que os requisitos são descrições detalhadas dos objetivos presentes no Quadro 3.2.

Para o papel de *Professor* os requisitos funcionais incluem a criação de disciplinas e avaliações, adição de discentes em suas disciplinas e também a visualização das notas de seus discentes para cada avaliação criada (RF-01).

| Id | Descrição |
|---|---|
| RF-01 | Para o Professor:<br>- Criar disciplinas e avaliações.<br>- Adicionar alunos em suas disciplinas.<br>- Visualizar notas dos discentes para cada avaliação aplicada. |
| RF-02 | Para o Aluno:<br>- Fazer a cópia da avaliação em sua máquina local (*download)*.<br>- Submeter a solução da avaliação (*upload*).<br>- Visualizar suas notas nas disciplinas. |
| RF-03 | Para o Administrador:<br>- Visualizar gráficos com informações de uso da *Test&Code.*<br>- Conceder permissões de administrador a outros professores. |
| RF-04 | - Oferecer recursos auxiliares ao usuário, como emails automáticos e mensagens de erro ou sucesso. |
| NRF-01 | - Oferecer segurança de informações, com controle de acesso às páginas de diferentes papéis. |
| NRF-02 | - Oferecer rapidez e confiabilidade na disponibilização das notas ao Aluno e Professor. |

**Quadro 3.3 - Principais requisitos funcionais e não-funcionais implementados na *Test&Code*.**
**Fonte: Quadro criado pelos autores.**

Para papel de *aluno* os requisitos funcionais incluem poder fazer *download* da avaliação, bem como fazer *upload* da sua resposta e poder visualizar suas notas nas disciplinas (RF-02).

E por último os requisitos funcionais para o papel de *Administrador* inclui a visualização de gráficos com informações da ferramenta e a possibilidade de dar permissões de *Administrador* a outros *Professores* (RF-03).

Ao realizar um requisito funcional na ferramenta, o usuário recebe uma mensagem, podendo ela ser de sucesso, confirmando a ação, ou de erro, dando novas informações sobre o ocorrido (RF-04).

A ferramenta oferece controle do acesso dos usuários, restringindo o acesso à páginas que não possui permissão. Por exemplo, na tentativa de acesso de uma página do professor pelo aluno, ele é redirecionado à sua página inicial (NRF-01).

Além disso a ferramenta também oferece rapidez e confiabilidade na disponibilização das notas, sendo calculado a partir de testes automatizados na avaliação, e por isso as notas são geradas instantaneamente (NRF-02).

## 3.4 *USER STORIES*

Os RFs apresentados no Quadro 3.3 são sentenças em alto nível de abstração. Um maior detalhamento torna-se necessário. Técnicas como Casos de Uso e *User Stories* são amplamente utilizadas na prática. Pelo fato do desenvolvimento da ferramenta ter sido feito em iterações curtas e com possibilidade de mudanças durante o projeto, optamos por usar *User stories* neste trabalho.

No decorrer deste documento apresentamos exemplos com base nas Histórias de Usuários apresentadas no Quadro 3.4 (vide Apêndice A para uma lista completa de Histórias de Usuários implementadas na *Test&Code*). Conforme pode ser visto no Quadro 3.4, a História de Usuário US-0001 está associada com o RF-01 descrito no Quadro 3.3. A História de Usuário US-0002 está associada com o RF-02. Já a História de Usuário US-0003 está associada com o RF-03 no referido quadro.

| ID | Descrição | Critério de aceitação |
|---|---|---|
| US-0001 | Enquanto *Professor* quero ver a nota de todos alunos em uma avaliação específica para poder visualizar o desempenho dos alunos. | - Após selecionar a disciplina, o *Professor* pode escolher a opção *notas*. Esta opção possui todos os alunos da matéria com suas respectivas notas<br>- Uma opção para fazer o *download* das respectivas respostas deve estar disponível para o *Professor*. |
| US-0002 | Enquanto *Aluno* eu quero criar uma conta na *Test&Code* para poder realizar as avaliações e obter a nota em tempo real. | - O *Aluno* possui o seguintes atributos: *nome*, *email*, *matrícula* e *senha*.<br>- Para evitar o cadastro de senhas erradas, deve-se solicitar ao *Aluno* confirmar a senha (digitando novamente).<br>- Uma vez cadastrado, testar o acesso com sucesso usando as credenciais criadas (*usuário*, *senha*)<br>- Uma vez cadastrado, testar o acesso com insucesso usando credenciais erradas (combinação de *usuário*/*senha* errada)<br>- Uma mensagem deverá ser informada para o *Aluno* em caso de acesso com sucesso ou insucesso durante autenticação. |
| US-0003 | Enquanto *Administrador* eu quero ter informações sobre a quantidade de usuários (*alunos* e *professores*) que utilizam a ferramenta. | O *Administrador* tem acesso a uma tela com gráficos para observar a quantidade de alunos, professores e avaliações que foram criadas em cada mês. |

**Quadro 3.4 - *User Stories*.**
**Fonte: Quadro criado pelos autores.**

## 3.5 ARQUITETURA DA *TEST&CODE*

A *Test&Code* é uma aplicação *web* que faz uso da arquitetura Model-View-Controller (MVC). MVC separa a aplicação em 3 componentes: *model* (modelo), *view* (visão) e *controller* (*controlador*), como mostra a Figura 3.1.

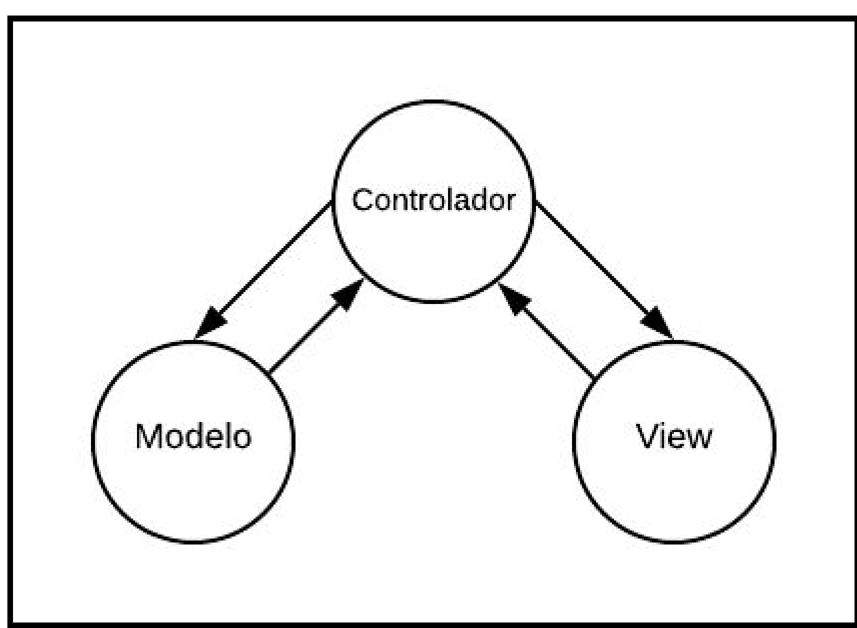


**Figura 3.1 - Estrutura básica do MVC.**

A *View* é responsável pela aparência da página *web* e corresponde ao *front-end* da aplicação, englobando tudo que está disposto aos olhos do usuário, como tabelas, formulários, menus, botões, etc.

O *Controlador* é responsável pela coleta dos dados oriundos de uma requisição do usuário, utilizando o *Modelo* como objeto que encapsula os dados. O *Controlador* pode delegar para outros componentes ou assumir a responsabilidade de implementar a lógica de negócios da aplicação a partir do processamento dos dados presentes no *Modelo*. Após processamento, o *Controlador* redireciona a saída do processamento para uma *View* adequada, que por sua vez é disponibilizada ao usuário em seu navegador *web*. *Controlador* e *Modelo* representam o *backend* da *Test&Code*.

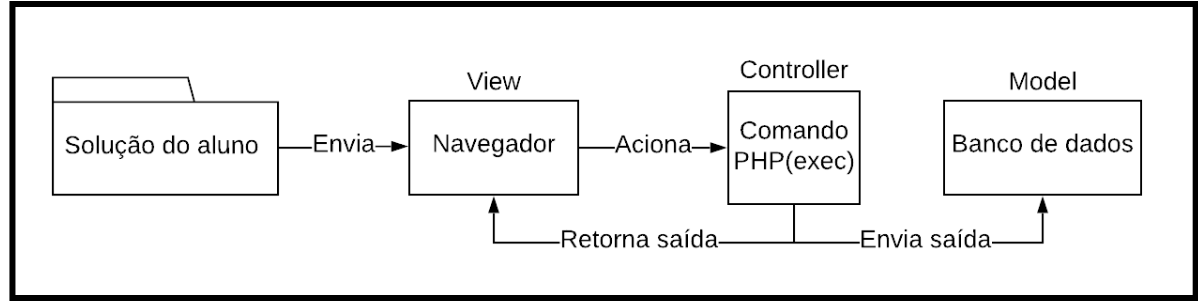


**Figura 3.2 - Exemplo do funcionamento MVC na ferramenta *Test&Code*.**

Como exibido na Figura 3.2, o *Aluno* por meio da *View* renderizada no navegador *web* envia a solução da avaliação criada para processamento pelo *back-end* da *Test&Code*. O *Controlador* recebe a solução e executa comandos PHP, que no caso da *Test&Code*, tem a função de enviar comandos Maven (Apache Maven, 2002) pelo terminal do servidor para execução dos testes necessários para correção automatizada da solução. O *Controlador* usa a saída produzida pelo Maven para calcular as notas dos alunos, enviar esses resultados para o banco de dados e também enviar os resultados para a próxima *View* a qual irá exibir as notas para o aluno.

A aplicação *web Test&Code*, implementada usando MVC, é implantada em uma arquitetura cliente/servidor, conforme ilustrado pelo diagrama de implantação UML (*Deployment diagram*) (LARMAN, 2007) na Figura 3.3.

Implantamos a *Test&Code* em um uma máquina com sistema operacional GNU/Linux(Ubuntu), servidor *web* Artisan, e MariaDB como Sistema Gerenciador de Banco de Dados. No componente *Test&Code*, subcomponente do componente Artisan na Figura 3.3, observa-se a arquitetura MVC citada anteriormente, contendo a *View*, o *Controller* e o *Model*, sendo a *View* e o *Controller* seguidos da palavra *Model* (em azul na figura). Dessa forma o diagrama da Figura 3.3 pode ser instanciado para várias classes diferentes que atuam como *Model*. Por exemplo, suponha a classe de modelo *Aluno*. Para esta classe, o componente *Test&Code* na figura passa a ser constituído de três subcomponentes (*Aluno*, *ControllerAluno* e *ViewAluno*). Raciocínio semelhante pode ser feito para outras classes de modelo (*Professor*, *Disciplina*, *Nota*, etc).

A conexão entre o servidor em que a *Test&Code* é implantado (cubo à esquerda na Figura 3.3) e a máquina do usuário (cubo à direita na Figura 3.3) é feita via protocolo HTTP (*HyperText Transfer Protocol*) a partir de um conjunto de requisições/respostas HTTP. Os

componentes se comunicam via troca de mensagens transportadas sobre o protocolo HTTP (linhas pontilhadas na figura). Além disso, como visualizado na figura, existe uma conexão entre os componentes *Model* da ferramenta e o banco de dados, responsável pela manipulação dos dados (leitura e escrita).

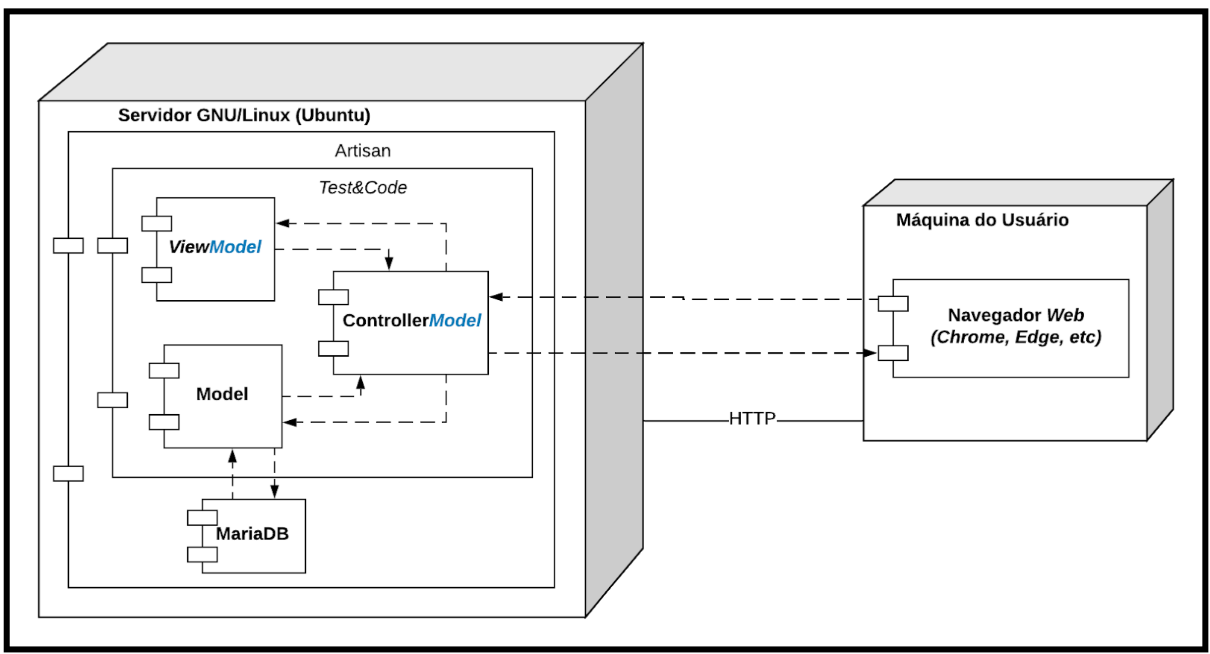


**Figura 3.3 Template da arquitetura via Diagrama de Implantação (*Deployment*) UML.**

# 4 TECNOLOGIAS

Apresentamos neste capítulo as tecnologias cliente (*frontend*) e servidor (*backend*) usadas na criação da *Test&Code*.

## 4.1 TECNOLOGIAS USADAS NO *FRONTEND* DA *TEST&CODE*

Usamos na implementação do *frontend* da *Test&Code* as tecnologias HTML5 (W3C, 2008), CSS3 (W3C, 1996), Bootstrap 4.3.1 (BOOTSTRAP, 2011), ferramentas do Google Charts (GOOGLE CHARTS, 2007) e Material Icons (MATERIAL ICONS, 2014).

HTML (*HyperText Markup Language*) é uma linguagem de marcação utilizada na construção de documentos *web*, que são interpretados por navegadores *web*. CSS (Cascading Style Sheets) é uma linguagem declarativa usada para estilizar (cores, fontes, espaçamento, *layout*, etc.) um documento *web*. HTML e CSS são normalmente usadas em conjunto.

Bootstrap é um *framework web* com código-fonte aberto para desenvolvimento de componentes responsivos que se adaptam para diferentes tipos de dispositivos e resoluções para aplicações *web*.

O Google Charts e o Material Icons são ferramentas de estilização de documentos *web* que permitem a criação de gráficos e utilização de ícones, respectivamente.

## 4.2 TECNOLOGIAS USADAS NO *BACKEND* DA *TEST&CODE*

O *backend* da *Test&Code* baseia-se na linguagem PHP 7.4+ (PHP, 2001), por meio do *framework* Laravel 7+ (Laravel, 2011), com a funcionalidade de acesso em um Sistema Gerenciador de Banco de Dados Relacional (SGBDR). Na *Test&Code* usamos o *Artisan*, que fornece um servidor de aplicações padrão do Laravel (invocado pelo comando *artisan serve*), e o MariaDB (MARIADB, 2009) como SGBDR.

Conforme mencionado anteriormente, o Laravel é um *framework* PHP, livre, de código-aberto, para o desenvolvimento de sistemas *web* que utilizam o padrão MVC (*Modelo*, *Visão*, *Controlador*).

*Artisan* é uma interface em linha de comando que fornece um conjunto de comandos que facilita a construção de seu aplicativo. Como por exemplo o comando *artisan serve* citado anteriormente.

A construção (*build*) automatizada de *software* prescreve a transformação de código fonte em um artefato que pode ser executado de forma independente deste código. Algumas ferramentas ajudam nesse processo de construção gerenciando, por exemplo, as dependências de *software* necessárias em tempo de compilação e execução, além de auxiliarem a realização testes de forma automatizada.

Usamos a ferramenta de *build* de *software* chamada *Apache Maven* para auxiliar a execução dos testes e correção automatizada das avaliações. Maven utiliza um arquivo XML (Extensible Markup Language) de nome *pom.xml* para descrever o projeto de *software* sendo construído. No arquivo *pom.xml* estão declarados os *plug-ins*, funcionalidades a serem executadas pelo Maven, como compilação e testes. Dependências de componentes de *software* necessários em tempo de compilação e execução também são declaradas no arquivo *pom.xml*. O JUnit é um exemplo de dependência que permite a confecção de testes automatizados de unidade. Os testes com JUnit são executados por um *plug-in* Maven.

# 5 EXEMPLO DE USO

Neste capítulo apresentamos exemplos de uso da *Test&Code* para cada um dos papéis identificados: *Aluno*, *Professor* e *Administrador*.

## 5.1 INTRODUÇÃO AO EXEMPLO

Além da utilização de testes simples para a validação da ferramenta, usamos um exemplo contendo regras de negócio para um sistema de Ponto de Venda (PDV), similar ao usado em caixas de supermercados, no intuito de demonstrar a aplicabilidade da *Test&Code*. Este exemplo é usado por Silva e Siqueira (SILVA e SIQUEIRA, 2013) e discutido em detalhes em Larman (LARMAN, 2007).

Além disso, este exemplo já foi usado como avaliação em sala de aula, de forma manuscrita (sem auxílio de computador), na disciplina *Programação Orientada a Objetos* em um curso de bacharelado em Ciência da Computação de uma universidade pública brasileira. A avaliação impressa é constituída de um enunciado contendo as questões a serem avaliadas (Veja Anexo A para maiores detalhes). O enunciado da avaliação contém questões cobrindo aspectos de implementação orientada a objetos tais como classes, objetos, atributos, associações, construtores e métodos. No enunciado, trechos de código são implementados parcialmente (esqueleto de código), com lacunas a serem completadas pelo discente. Nesta avaliação em particular, um diagrama de classes foi fornecido ao discente como referência na implementação. Além do diagrama de classes, o professor também pode incluir no enunciado da avaliação, diagramas de sequência, diagramas de pacotes, dentre outros diagramas UML que auxiliem o aluno no entendimento do código entregue junto com a avaliação. Nesta avaliação em particular, o professor optou por entregar apenas um diagrama de classes.

As seções a seguir ilustram o processo, usando a *Test&Code*, de criação desta avaliação exemplo pelo *Professor*, a confecção da avaliação pelo *Aluno* e a geração de gráficos de monitoramento pelo *Administrador*.

## 5.2 EXEMPLO DE USO - CADASTRO E AUTENTICAÇÃO

Para a utilização da *Test&Code*, deve-se primeiramente realizar o cadastro de uma conta. Tanto o cadastro do *Aluno* quanto do *Professor* são feitos na mesma página, se diferenciando pelo fato do formulário do *Aluno* apresentar um campo adicional de matrícula. A Figura 5.1 apresenta o formulário usado para cadastro do *Professor*.

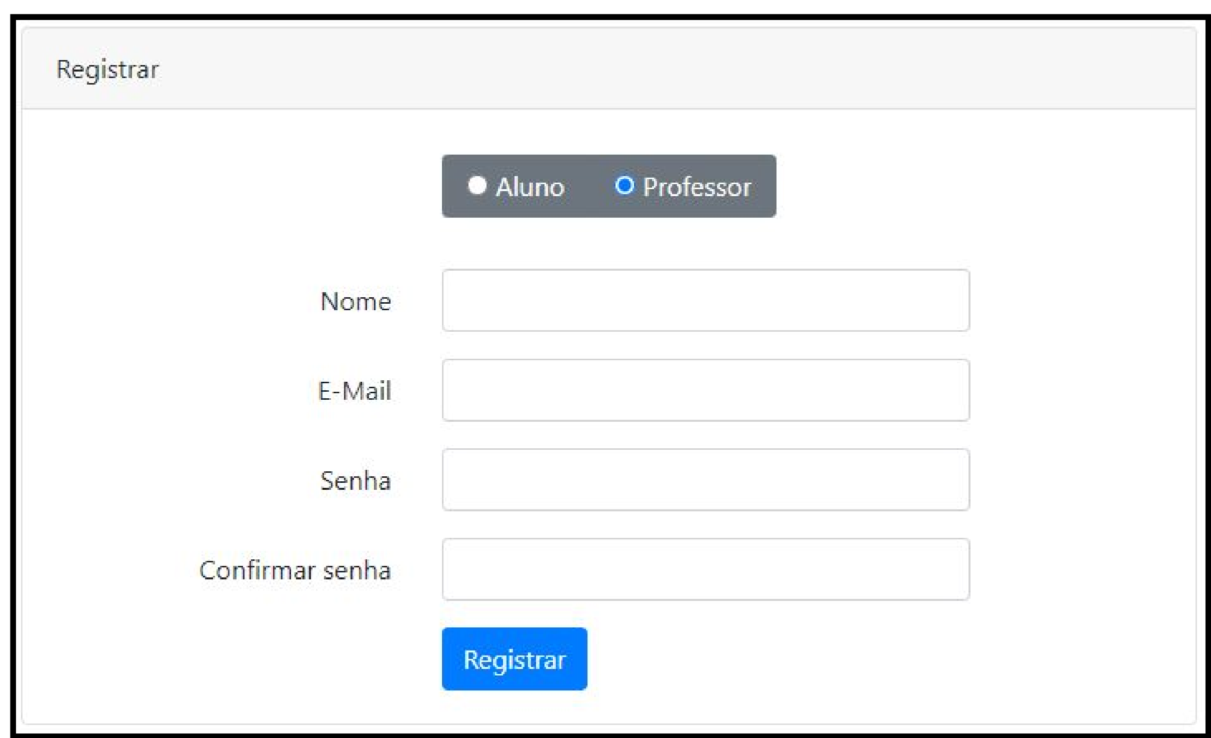


**Figura 5.1 - Interface gráfica usada para criação de uma conta de *Professor*.**

Para o *Administrador*, não é necessário o cadastramento, pois se trata de um papel adicional atribuído a um *Professor*, previamente registrado. Inicialmente um *Professor* no papel de *Administrador* é criado e armazenado no banco de dados no momento da implantação da *Test&Code* em ambiente de produção. Este usuário *Administrador* pode conceder permissão de *Administrador* aos outros usuários a partir da busca de seus nomes.

O usuário (*Professor*, *Aluno* ou *Administrador*) pode fazer o *login* utilizando-se do seu *email* e *senha*, como mostra a Figura 5.2.

**Figura 5.2 - Interface gráfica usada pelo *Aluno*, pelo *Professor* e pelo *Administrador* para autenticação na *Test&Code*.**

## 5.3 EXEMPLO DE USO - *PROFESSOR*

Uma vez autenticado na *Test&Code*, o *Professor* deve criar sua disciplina a partir do menu lateral, conforme ilustra a Figura 5.3.

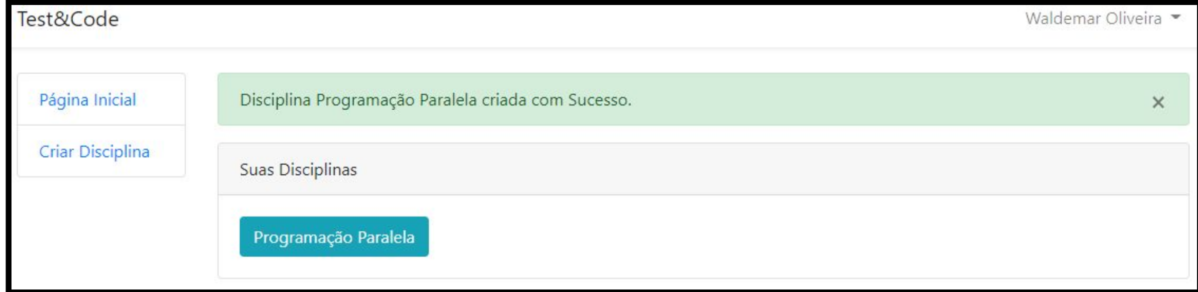


**Figura 5.3 - Interface gráfica usada pelo *Professor* para criação de disciplinas.**

Na sequência, o *Professor* deve adicionar seus alunos à disciplina recém-criada selecionando a opção *Adicionar* exibida na Figura 5.4.

Alunos Buscados

| Nome | Matricula | Email | Adicionar |
|---|---|---|---|
| **Daniel Santos** | 2016.1.08.026 | daniel@hotmail.com | + |

Alunos Adicionados à Disciplina

| Nome | Matricula | Email |
|---|---|---|
| **Pedro Silva** | 2016.1.08.013 | pedro@hotmail.com |
| **Rafael Ferreira** | 2016.1.08.016 | rafael@hotmail.com |

**Figura 5.4 - Interface gráfica usada pelo *Professor* para adição de alunos à disciplina.**

Uma vez que uma disciplina esteja criada e alunos matriculados, o *Professor* deve incluir a avaliação na *Test&Code*. Isso é feito por meio do *upload* do projeto *Maven* completo, totalmente funcional (contendo todos os métodos implementados, juntamente com todos os testes JUnit). Os testes de unidade só são acessados pelo *Professor*, pois fornecem o gabarito da avaliação prática. No nosso exemplo, fazemos o *upload* do projeto Maven, compactado no formato *zip*, contendo o sistema PDV (Figura 5.5), por meio do botão *Fazer Upload do Projeto*, na página da disciplina conforme exibido na Figura 5.6.

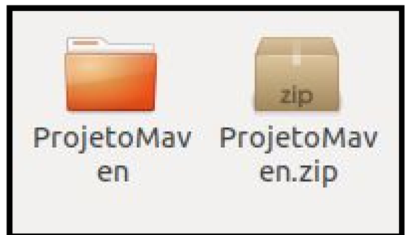


**Figura 5.5 - Projeto compactado com os testes para serem aplicados na avaliação.**

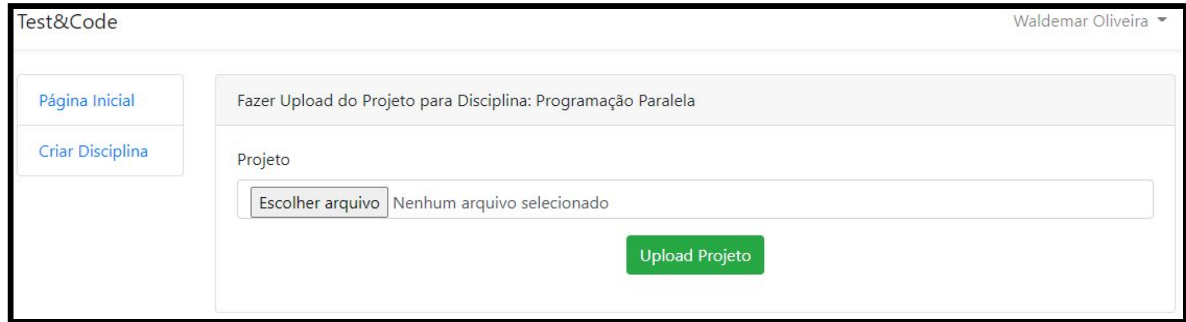


**Figura 5.6 - Interface gráfica usada pelo *Professor* para *upload* do projeto.**

Após isto, o *Professor* deve então selecionar a opção *Criar Avaliação* (Figura 5.7) e escolher o projeto que fez o *upload* anteriormente, para então fazer o *upload* da avaliação (Figura 5.8), contendo o enunciado, o diagrama de classes (opcional) e o esqueleto do código que serve como uma base para que os alunos continuem a implementação do projeto.

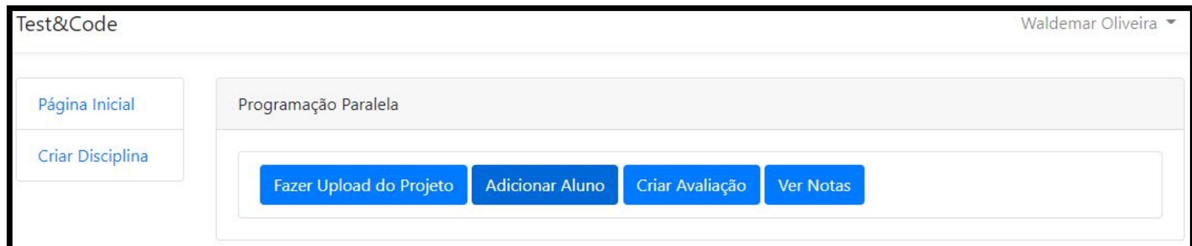


**Figura 5.7 - Interface gráfica da disciplina criada pelo *Professor*.**

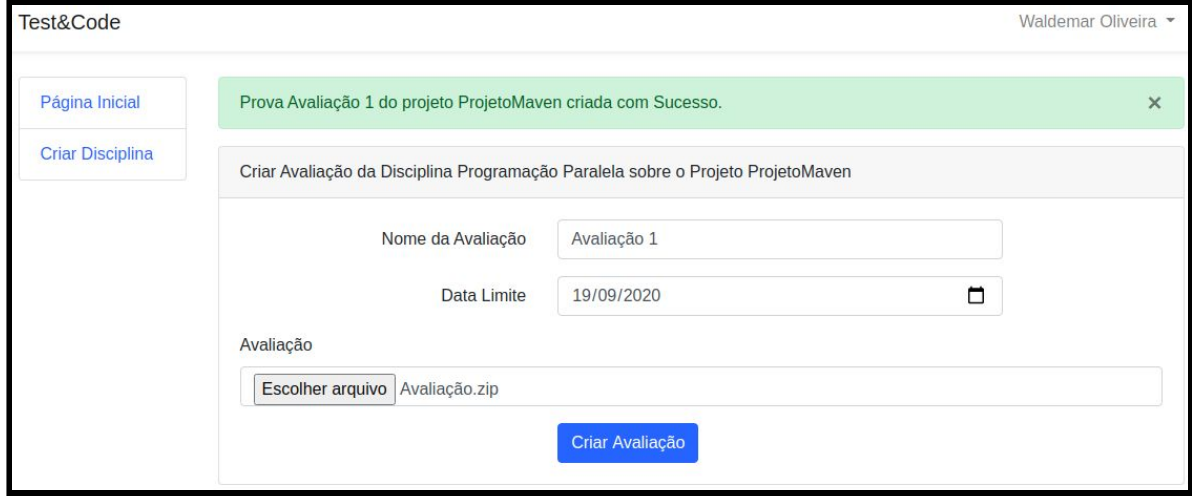


**Figura 5.8 - Interface gráfica usada pelo *Professor* para criação de avaliações.**

## 5.4 EXEMPLO DE USO - *ALUNO*

Após a criação da avaliação pelo *Professor*, o *Aluno* recebe um email, avisando sobre a disponibilidade da avaliação, e então deve entrar na página da disciplina para acesso à avaliação. Isto pode se feito por meio da opção "Baixar" (Veja Figura 5.9).

Provas e Trabalhos da disciplina Programação Paralela

| Nome | Data Limite | Baixar | Enviar Resposta |
|---|---|---|---|
| **Avaliação 1** | 2020-09-19 | | |

**Figura 5.9 - Interface gráfica usada pelo *aluno* para acesso à avaliação.**

Após "baixar" (*download*), o *Aluno* tem em sua máquina local todo o conteúdo necessário para realização da avaliação. Este conteúdo pode ser composto de código fonte e enunciado ou composto somente do projeto *Maven* com o esqueleto do código fonte a ser completado pelo *Aluno*. Neste caso, detalhes do enunciado da avaliação devem ser discutidos com o professor(a).

De maneira mais comum, o conteúdo da avaliação na máquina local do *Aluno* é constituído do esqueleto de código fonte em conjunto com o enunciado da avaliação e uma ou mais imagens com diagramas UML que normalmente são citados no enunciado. Veja Apêndice C para exemplo de uma avaliação contendo todos estes artefatos (esqueleto de código, enunciado em pdf, diagrama UML, etc).

Após o *Aluno* finalizar a avaliação em sua máquina local, ele deve fazer o *upload* da solução. Para isso, o *Aluno* deve compactar (a *Test&Code* suporta arquivos usando o padrão *zip* - extensão *.zip*) a pasta que foi disponibilizada para realização da avaliação (Figura 5.10). O arquivo compactado deve conter somente a pasta do projeto, excluindo qualquer outro documento que estava incluído anteriormente como enunciado textual ou imagens (e.g. arquivos PDF e de imagens *.png*).

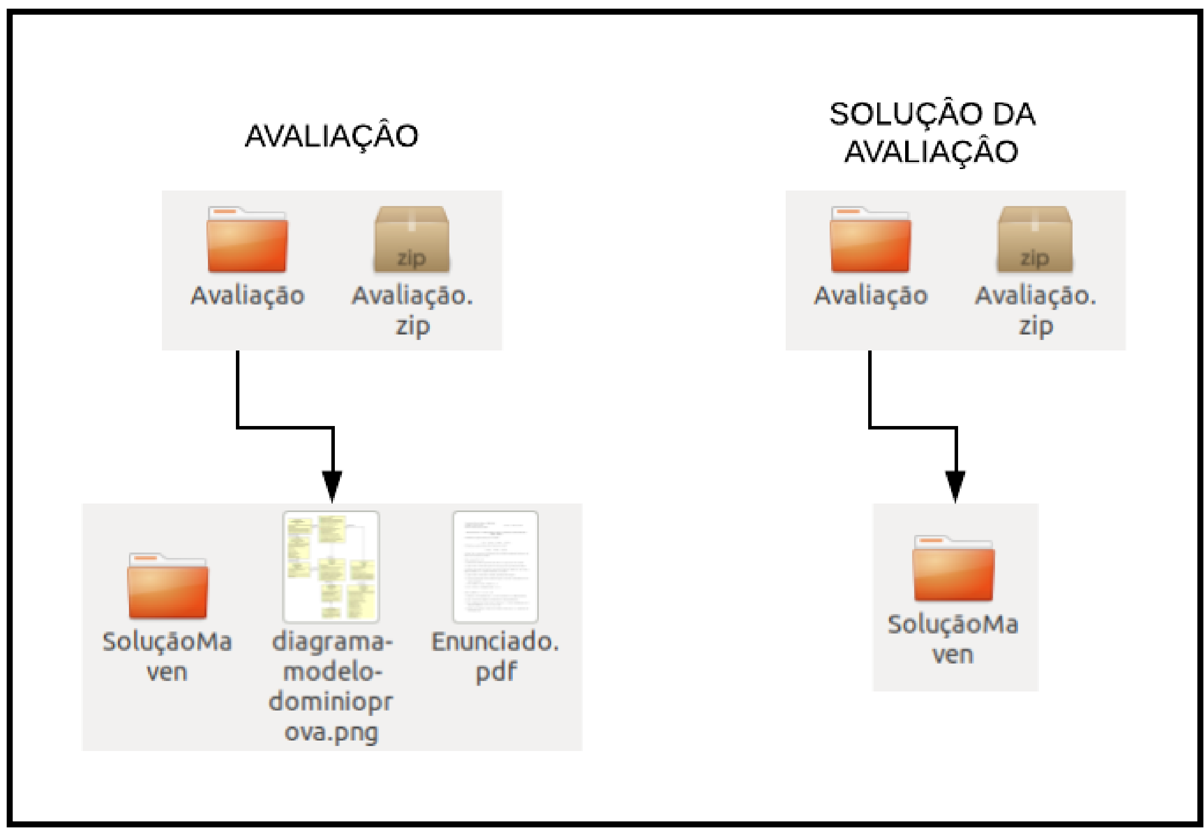


**Figura 5.10 Relação entre o conteúdo típico de uma avaliação disponibilizada pelo *Professor* (à esquerda na figura) e o conteúdo necessário que o *Aluno* deve ter antes da submissão de sua resposta (à direita na figura).**

## 5.5 EXEMPLO DE USO - VISUALIZAR NOTAS

Logo após envio da avaliação, o *Aluno* é redirecionado automaticamente para a página *web* contendo o resultado, conforme exibido na Figura 5.11. O *Aluno* também pode acessar a página com resultados, sem ser redirecionado, acessando itens do menu lateral semelhante ao do *Professor*.

Suas notas

Programação Paralela 2020/2

| Nome | Nota |
|---|---|
| Avaliação 1 | 8.7 |

**Figura 5.11 - Interface gráfica para disponibilização da nota do *Aluno*.**

Para visualização de notas, o *Professor* deve selecionar a disciplina e, na sequência, a opção *Ver Notas* (vide Figura 5.7). A Figura 5.12 apresenta as notas dos *Alunos*.

Notas dos Alunos da Disciplina:Programação Paralela

| Nome | Matricula | Nota |
|---|---|---|
| Rafael Ferreira | 2016.1.08.016 | 8.7 |
| Pedro Silva | 2016.1.08.013 | 10 |
| Daniel Santos | 2016.1.08.026 | 10 |
| João Oliveira | 2016.1.08.011 | 8.7 |
| Thiago Souza | 2016.1.08.015 | 10 |

**Figura 5.12 - Interface gráfica usada pelo *Professor* para visualização de notas dos *Alunos*.**

## 5.6 EXEMPLO DE USO - *ADMINISTRADOR*

Como o *Administrador* é um papel adicional, ele apresenta as funcionalidades de um *Professor*. Um *Administrador* tem autorização para acessar a gráficos de uso da *Test&Code.* Os gráficos contém informações de quantos alunos, professores e avaliações foram criadas em cada mês. Além de autorização de acesso a gráficos, o *Administrador* também tem acesso a funcionalidade de concessão de permissões, podendo conceder permissão de *Administrador*

para outros *Professores* da *Test&Code*.

A Figura 5.13 exibe um gráfico típico, disponível para o papel de *Administrador* da *Test&Code*.

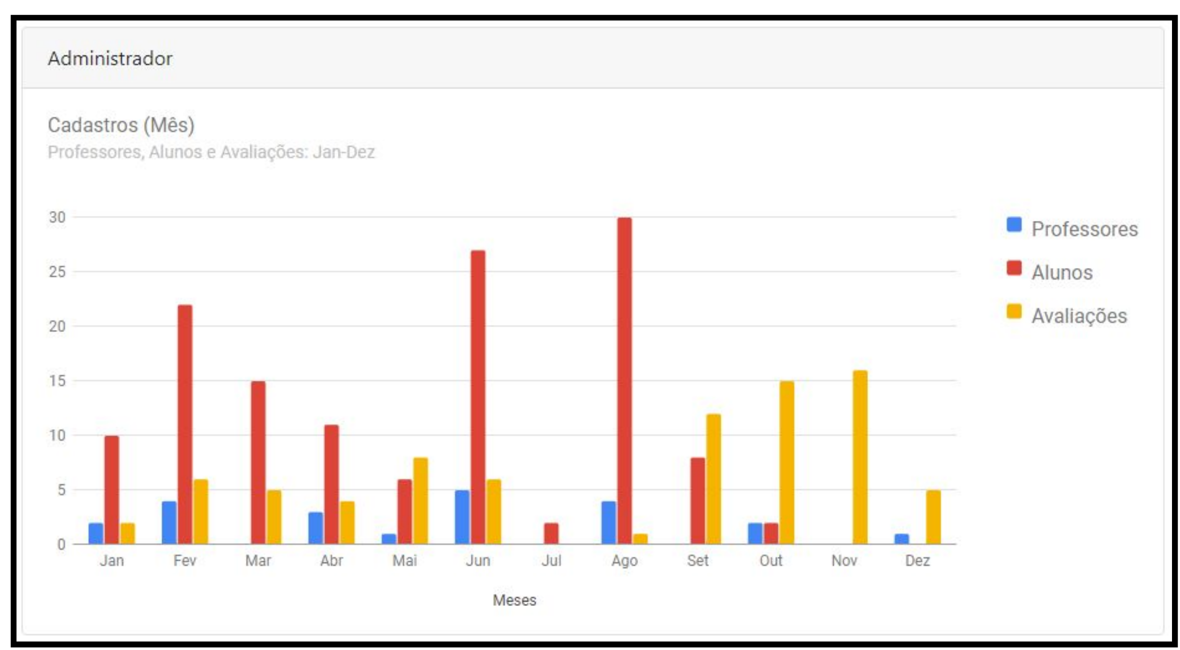


**Figura 5.13. Gráfico disponível para o papel *Administrador* na *Test&Code*.**

# 6 DETALHES DA IMPLEMENTAÇÃO

Neste capítulo demonstramos detalhes da implementação da ferramenta *Test&Code*. Focamos nos RFs principais para o *Professor* e *Aluno*. Em ambos os códigos palavras reservadas foram destacadas em negrito.

Seguindo a *User Story US-0005* para o papel *Professor* disponibilizada no Quadro A.1 do Apêndice A que diz "*Enquanto professor quero criar uma avaliação para os alunos...*" e envolve o requisito RF-01 do Quadro 3.3, o método *criarProva()* da classe *CriarProvaController* do Código 6.1 faz a validação da avaliação enviada pelo *Professor* (Linhas 03 a 05), verificando se a extensão do arquivo enviado é zip.

Nas Linhas 07 a 09, o projeto compactado enviado pelo *Professor* recebe um nome único, composto pelo intervalo de tempo em segundos entre o horário atual e a data de 1 de Janeiro de 1970 às 00:00:00 (Era Unix), seguido do nome do arquivo. Na Linha 12 e 13 essa avaliação é armazenada no sistema de diretórios de arquivos da disciplina.

Da Linha 20 até a Linha 26, a avaliação é adicionada ao banco de dados da *Test&Code*, com todas as informações necessárias buscadas nas linhas anteriores.

Por último, utilizamos a classe *Session* (Linha 28 a 30) do *framework* Laravel, após o envio da avaliação, para envio de uma mensagem de sucesso ao *Professor* que, posteriormente, será mostrada junto a *view* que é retornada nas Linhas 32 e 33.

```
01 public function criarProva(Request $avaliacaoCompactada,
02                                $disciplina, $nomeProjeto){
03 $this->validate($avaliacaoCompactada, [
04     'featured' => 'required|mimes:zip'
05 ]);
06
07 $featured = $avaliacaoCompactada->featured;
08 $featured_new_name =
09 time().$featured->getClientOriginalName();
10 $featuredsemzip = pathinfo($featured_new_name,
11                                PATHINFO_FILENAME);
12 $featured->move('uploads/'.$disciplina.'/provas/'.
```

```
13                 $featuredsemzip.'/',$featured_new_name);
14
15  $idProjeto = DB::table('projetos')
16              ->select('projetos.*')
17              ->where('projetos.nomeProjeto','=',$nomeProjeto)
18              ->first();
19
20  $prova = Prova::create([
21     'featured' => 'uploads/'.$disciplina.'/provas/'.
22      $featuredsemzip .'/'. $featured_new_name,
23     'idProjeto' => $idProjeto->id,
24     'nomeProva' => $avaliacaoCompactada->nome,
25     'dataLimite' => $avaliacaoCompactada->dataLimite,
26  ]);
27
28  Session::flash('status','Prova'.$avaliacaoCompactada->nome.
29                 'do projeto '. $nomeProjeto.' criada com
30                 Sucesso.');
31
32  return view('/professor/criarProva',
33               compact('disciplina','nomeProjeto'));
```

**Código 6.1 - Função de criação de avaliação por parte do *Professor*.**

Na Figura 6.2, Linhas 01 a 12, é feita a preparação para que os testes sejam executados, ocorrendo basicamente o acréscimo dos testes do projeto do *Professor* à pasta do projeto do *Aluno*, bem como o arquivo `pom.xml`.

Nas linhas 13 a 15 o teste é executado, a partir do comando Maven (*mvn test*), gerando uma saída que é utilizada para cálculo da nota. Essa saída é observada nas próximas linhas, pelo laço de repetição *foreach* da Linha 19, que guarda as informações em variáveis para serem utilizadas no cálculo final da nota.

Este cálculo da nota é feito na Linha 33, caso tudo tenha corrido conforme o esperado. Caso exista algum problema no envio, a *Test&Code* apresenta uma mensagem de erro ao *Aluno* e sua solução é removida do sistema de pastas para que possa ser feito um novo envio,

como pode ser observado nas Linhas 37 a 40.

```
01 exec("ls uploads/$disciplina/respostas/".
02      auth()->user()->nome.
03      "/$nomeProva/$nomeResposta",$out3);
04 $nomeProjeto = $out3[0];
05 exec("cp -r $projeto->featured/src/test
06      uploads/$disciplina/respostas/".
07      auth()->user()->nome.
08      "/$nomeProva/$nomeResposta/$nomeProjeto/src/");
09 exec("cp -r $projeto->featured/pom.xml
10      uploads/$disciplina/respostas/".
11      auth()->user()->nome.
12      "/$nomeProva/$nomeResposta/$nomeProjeto");
13 exec("mvn test -f uploads/$disciplina/respostas/".
14      auth()->user()->nome.
15      "/$nomeProva/$nomeResposta/$nomeProjeto",$out2);
16
17 $total=0;
18 $search = 'Tests run';
19 foreach($out2 as $linha){
20    if(strstr($linha, $search)){
21       $arrayRespostas = explode(',',$linha);
22       foreach($arrayRespostas as $resposta){
23          if(strstr($resposta, 'Tests run')){
24             $total=preg_replace("/[^0-9]/", "", $resposta);
25          }
26          if(strstr($resposta, 'Failures')){
27             $erros=preg_replace("/[^0-9]/", "", $resposta);
28          }
29       }
30    }
31 }
32 if($total!=0){
33    $nota = 10-((10/$total)*$erros);
34 }else{
```

```
35     exec("rm -r -d uploads/$disciplina/respostas/".
36          auth()->user()->nome."/$nomeProva");
37     Session::flash('erro', 'Houve um erro no envio da sua
38                    prova,por favor verifique se seu envio
39                    está conforme o necessário.');
40     return redirect()->back();
41  }
42
43  $resposta = Resposta::create([
44     'featured' => 'uploads/'.$disciplina.'/respostas/'.
45                   auth()->user()->nome .'/'. $nomeProva.'/',
46     'idProva' => $prova->id,
47     'idAluno' => auth()->user()->id,
48     'nota' => $nota,
49  ]);
50
51  Session::flash('status', 'Prova enviada com sucesso.');
```

**Código 6.2 - Função de envio da solução do aluno e cálculo de sua nota.**

# 7 DISCUSSÕES

A *Test&Code* funciona em torno de 3 papéis: *Administrador*, *Professor* e *Aluno.* Implementamos funcionalidades específicas de cada papel.

As funções mais simples estão na mão do *Administrador*, onde é possível tornar outros usuários administradores (concessão de privilégio) e visualizar gráficos sobre o cadastro de alunos, professores e avaliações no *software* ao longo do tempo.

A *Test&Code* disponibiliza para o *Professor* as funcionalidades relacionadas à disciplinas, como a sua criação e adição de alunos nas mesmas. Após a criação da disciplina o *Professor* pode fazer *upload* de seus projetos com as classes de teste para que possam ser utilizados posteriormente na correção das avaliações. Para se criar a avaliação é necessário escolher um dos projetos já enviados anteriormente e depois fazer o *upload* do projeto contendo o esqueleto do código fonte para que o aluno possa fazer o *download* e realizar a avaliação.

O *Test&Code* envia um *email* ao *Aluno* após ser adicionado em uma disciplina e também após a criação da avaliação pelo *Professor*. Dessa forma, o *Aluno* pode acessar a página da disciplina, fazer o *download* da avaliação, realizar o que for pedido pelo *Professor* e fazer o *upload* da solução.

Por último, após o envio da solução, o *Aluno* possui acesso imediato ao seu resultado. Um *Aluno* não tem acesso à nota de outro *Aluno*. O *Professor* tem acesso às notas de todos alunos para uma determinada disciplina.

Para o cálculo das notas, todas as classes são testadas e as implementações corretas dos métodos possuem o mesmo peso na nota final. Esta é uma limitação atual da *Test&Code*.

Caso haja algum problema com o envio da solução, como por exemplo algum sistema de arquivos incorreto, a *Test*&Code aciona uma mensagem de erro para o *Aluno*, que deve então verificar se está no padrão pedido e reenviar a solução.

A fim de testar a viabilidade da ferramenta utilizamos o sistema PDV discutido no Capítulo 5. Trata-se de um exemplo de certa complexidade (Vide Anexo A para o enunciado completo do exemplo), que já foi utilizado em uma avaliação da disciplina Programação Orientada a Objetos em um curso de Bacharelado em Ciência da Computação de uma Universidade Federal no Brasil. Os resultados da aplicação tradicional foram de que 40% dos

alunos tiveram notas abaixo de 6, representando um desempenho abaixo da média e que corroboram nossa hipótese de que o exemplo que utilizamos para testes da *Test&Code* possui relativa complexidade, tanto em nível de *design* de objetos, quanto em nível de algoritmos cobrados no enunciado.

Levando em conta o exemplo descrito no parágrafo anterior, com a utilização de um sistema com certa complexidade na criação de avaliações na ferramenta *Test&Code*, podemos observar que existem certas dificuldades para o professor na criação dos testes que serão utilizados como gabarito nas avaliações. Dificuldades essas geradas pela necessidade do entendimento de dublês de teste, como por exemplo, *mocks*, *spies*, *stub*, dentre outros (Kaczanowski, 2019).

Apesar da complexidade do exemplo usado para teste da *Test&Code*, reconhecemos que mais testes com outros exemplos de avaliações precisam ser feitos.

Após pesquisa sobre plataformas MOOCS, citadas no Capítulo 2, encontramos poucas informações sobre a forma de correção de códigos utilizadas por algumas delas, talvez pelo fato de serem de código fechado, dificultando a comparação com a *Test&Code*.

# 8 CONCLUSÃO E TRABALHOS FUTUROS

O desenvolvimento do presente trabalho possibilitou concluirmos que a correção automatizada de avaliações, além de gerar notas imediatamente, contribuindo com os alunos, também diminui o trabalho de correção do *Professor*.

A partir da utilização de projetos *Maven*, é possível executar um maior número de casos de testes para um projeto, possibilitando ao discente enviar um código mais complexo com várias classes para a correção. A possibilidade de testar códigos com relativa complexidade (em número de classes e com diversos alternativas de *design*) faz da *Test&Code* uma opção viável como *software* corretor automático de código orientado a objetos.

Para trabalhos futuros, nós sugerimos a implementação de suporte à avaliações de questões de múltipla escolha, preenchimento de lacunas, bem como avaliações de códigos estruturados, ampliando assim o uso da ferramenta. Além disso, indicamos o suporte ao *Gradle* (GRADLE, 2009), ferramenta de *build* automatizado de *software*, similar ao *Maven*. Este suporte tem o potencial de dar flexibilidade ao professor na escolha de sua ferramenta de *build* predileta, além de aumentar a variedade de projetos a serem aceitos pela *Test&Code*.

Outra implementação possível para manter o NRF-01 do Quadro 3.3 do Capítulo 3 sobre segurança da ferramenta, seria outras possibilidades de se fazer *Login,* como por exemplo com a utilização de contas de terceiros(Google, GitHub, etc).

Com relação a criação das avaliações destacamos como opção a possibilidade de agendamento para a abertura e disponibilização da avaliação para os alunos.

Além disso, também sugerimos serem implementadas alternativas para restringir o envio de soluções. Para isto, já existe o facilitador da *data limite*, escolhida na criação da avaliação.

Observando os requisitos sugeridos nos dois parágrafos anteriores, uma outra implementação para o aluno seria a opção de não receber mais emails automáticos da ferramenta.

Duas limitações conhecidas discutidas no capítulo anterior também podem ser consideradas como trabalhos futuros, são elas a falta da possibilidade do *Professor* poder adicionar pesos diferentes para testes e a criação de respostas inteligentes informando erros

específicos, já que atualmente somente é informado que um erro ocorreu.

Outro trabalho futuro sugerido é a possibilidade do professor fazer o download das soluções enviadas pelos alunos, para assim ter acesso a implementação dos métodos e não somente aos resultados de suas saídas, além da disponibilização de outras métricas relacionadas às notas dos alunos para o professor, como média, desvio padrão, entre outras.

E por último, apesar de utilizarmos um exemplo de relativa complexidade para a validação da ferramenta, novos testes devem ser feitos, a fim de validar a *Test&Code* em uma maior variedade de situações.

# REFERÊNCIAS


APACHE. MAVEN, 2002. Disponível em: <https://bit.ly/3mctcTz>. Acesso em: 17 mai. 2020.

AUFFARTH, B.; LÓPEZ-SÁNCHEZ, M.; CAMPOS I MIRALLES, J.; PUIG, A. System for Automated Assistance in Correction of Programming Exercises (SAC)⋆. Computer aided assessment for programming courses, University of Barcelona, Barcelona, 2008. Disponível em: <https://bit.ly/3kHhv5F >. Acesso em: 17 mai. 2020.

COURSERA, 2012. Disponível em: <https://bit.ly/32TORIy >. Acesso em: 18 mai. 2020.

GOOGLE. GOOGLE CHARTS, 2007. Disponível em: <https://bit.ly/3kGd9vw>. Acesso em: 17 mai. 2020.

LARMAN, Craig; Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Desenvolvimento Iterativo. 3. ed.: Bookman, 2007.

LOBB, Richard; HARLOW, Jenny. Coderunner: A Tool for Assessing Computer Programming Skills. Moodle question-type plug-in, The University of Canterbury, Christchurch, 2016. Disponível em: <https://coderunner.org.nz/>. Acesso em: 17 mai. 2020.

MERRIT. MERRIT, 2008. Disponível em: <https://meritt.com.br/>. Acesso em: 17 mai. 2020.

MESTREGR. MESTREGR, 2010. Disponível em: <https://www.mestregr.com.br/>. Acessado em: 17 mai. 2020.

LÜCKEMEYER G. MOJEC, 2017. Disponível em: <https://bit.ly/3m2E1r1>. Acesso em: 17 mai. 2020.

PROVA FÁCIL. PROVA FÁCIL, 2012. Disponível em: <https://www.provafacilnaweb.com.br/>. Acesso em: 17 mai. 2020.

SILVA, Douglas Miranda da; SIQUEIRA, Renan Domingues. Frameworks para automação de testes. 2013. Trabalho de Conclusão de Curso - Instituto de Ciências Exatas da Universidade Federal de Alfenas, Alfenas, 2013.

MOODLE. MOODLE, 2002. Disponível em: <https://moodle.org/?lang=pt_br>. Acesso em: 24 jun. 2020.

Test&Code, 2020. Disponível em: <https://bit.ly/2IUwNat>.

LARAVEL. LARAVEL, 2011. Disponível em: <https://laravel.com/docs/7.x>. Acesso em: 27 jun. 2020.

BOOTSTRAP. BOOTSTRAP, 2011. Disponível em: <https://getbootstrap.com/>. Acesso em: 27 jun. 2020.

PHP. PHP, 2001. Disponível em: <https://www.php.net/>. Acesso em: 27 jun. 2020.

MARIADB Foundation. MARIADB, 2009. Disponível em:<https://mariadb.org/>. Acesso em: 27 jun. 2020.

GOOGLE. MATERIAL ICONS, 2014. Disponível em: <https://bit.ly/3m2E6uP>. Acesso em: 27 jun. 2020.

JUNIT. JUNIT 1998, Disponível em: <http://junit.org/>. Acesso em: 29 jun. 2020.

GRADLE. GRADLE, 2009. Disponível em: <https://gradle.org/releases/>. Acesso em:01 jul. 2020.

CARVALHO, A. M. B. R. Requisitos de Software. Campinas: Unicamp, 2013. Disponível em: <https://bit.ly/3nyIX7h >. Acesso em: 07 jul. 2020.

HARVARD, MIT. EDX, 2012. Fundada por Harvard e MIT. Disponível em: <https://www.edx.org/>. Acesso em:11 jul. 2020.

CAELUM. ALURA, 2013. Disponível em: <https://www.alura.com.br/>. Acesso em: 11 jul. 2020.

UDEMY. UDEMY, 2010. Disponível em: <https://www.udemy.com>. Acesso em: 11 jul. 2020.

UDACITY. UDACITY, 2011. Disponível em:<https://www.udacity.com/>. Acesso em: 11 jul. 2020.

W3C, WHATWG. HTML5, 2008. Disponível em: <https://bit.ly/2Hx18eo>. Acesso em: 11 jul. 2020.

W3C. CSS, 1996. Disponível em:<https://www.w3.org/TR/css-2018/>. Acesso em: 11 jul. 2020.

SOMMERVILLE, I; Tradução Ivan Bosnic e Kalinka G. de O. Gonçalves; revisão técnica Kechi Hirama. Engenharia de Software. 9ª edição. São Paulo : Pearson Prentice Hall, 15 de jun. de 2011.

LINKEDIN LEARNING. LINKEDIN LEARNING, 2002. Disponível em: <https://www.linkedin.com/learning/me>. Acesso em: 24 ago. 2020.

GOOGLE CLASSROOM. GOOGLE CLASSROOM, 2014. Disponível em: <https://classroom.google.com/>. Acesso em: 01 nov. 2020.

KACZANOWSKI, Tomek; Practical Unit Testing with JUnit and Mockito. Kaczanowski, 2019.

# APÊNDICES

## APÊNDICE A - HISTÓRIAS DE USUÁRIO

Os requisitos funcionais da *Test&Code* foram identificados e documentados usando *User Stories* (Veja Quadros A.1, A.2 e A.3). Todas elas foram implementadas com parte deste trabalho.

| ID | História de usuário | Critério de aceitação |
| --- | --- | --- |
| **US-0001** | Enquanto *Professor* eu quero criar uma conta para poder aplicar avaliações de programação orientada a objetos e obter uma correção automatizada. | - O *Professor* possui o seguintes atributos: *nome*, *email* e *senha*.<br>- Para evitar o cadastro de senhas erradas, deve-se solicitar ao *Professor* confirmar a *senha* (digitando novamente).<br>- Uma vez cadastrado, testar o acesso com sucesso usando as credenciais criadas (*usuário*, *senha*).<br>- Uma vez cadastrado, testar o acesso com insucesso usando credenciais erradas (combinação de *usuário*/*senha* errados).<br>- Uma mensagem deverá ser informada para o *Professor* em caso acesso com sucesso ou insucesso durante autenticação. |
| **US-0002** | Enquanto *Professor* eu quero criar uma disciplina para que eu possa adicionar meus alunos e criar avaliações. | - Uma disciplina possui os seguinte atributos: *nome*, *ano/período*.<br>- Somente professores podem criar disciplinas.<br>- Na página da Disciplina o professor poderá adicionar alunos.<br>- Uma mensagem de sucesso ou insucesso na tentativa de criação de uma disciplina deve ser exibida. |
| **US-0003** | Enquanto *Professor* eu quero adicionar alunos em uma disciplina para poder criar avaliações da disciplina. | - O *Professor* fornece o nome do aluno e seleciona a opção *buscar*. Uma lista com todos alunos de mesmo nome (dados do aluno são gerados na US - 0001 do Quadro A.2) . O *Professor* seleciona o aluno a ser adicionado.<br>- Após adicionar o aluno, uma mensagem de aluno adicionado com sucesso deverá ser exibida e um *email* deverá ser enviado ao aluno.<br>- Caso não seja encontrado um aluno com o nome especificado, uma mensagem "aluno não encontrado" deve ser exibida. |
| **US-0004** | Enquanto *Professor* eu quero fazer *upload* de um projeto para poder criar uma avaliação. | - O *Professor* poderá escolher o projeto da sua máquina que deseja fazer o *upload*.<br>- O projeto deve estar compactado e deve conter tanto o código funcional como as classes de teste.<br>- Após o envio uma mensagem com os resultados dos testes deve ser mostrada, indicando os erros caso existam. |

| | | |
|---|---|---|
| **US-0005** | Enquanto *Professor* quero criar uma avaliação para os alunos de forma a poder avaliá-los na disciplina. | - O *Professor* deve dar um nome para a avaliação, uma data limite para a entrega e escolher um dos projetos já enviados para então enviar um arquivo `.zip` que pode conter os seguintes arquivos:<br>- Um arquivo no formato `pdf` contendo o enunciado da avaliação (com ou sem um diagrama de classes) para o aluno implementar.<br>- Um projeto que não contenha as classes de teste e que possua métodos em branco ou parcialmente implementados para finalização do código pelo aluno, relacionado ao projeto escolhido na US - 0004 deste quadro.<br>- Todos os alunos de uma disciplina deverão receber um *email* após a criação da avaliação pelo professor.<br>- O *email* contém informações sobre a avaliação e disciplina. |
| **US-0006** | Enquanto *Professor* quero ver a nota de todos alunos em uma avaliação específica para poder visualizar o desempenho dos alunos. | - Após selecionar a disciplina, o Professor pode escolher a opção *notas*. Esta opção possui todos os alunos da matéria com suas respectivas notas. |

**Quadro A.1 - Histórias de usuário criadas para o papel *Professor*.**
**Fonte: Quadro criado pelos autores.**

| ID | História de usuário | Critério de aceitação |
|---|---|---|
| **US-0001** | Enquanto *Aluno* eu quero criar uma conta para poder realizar as avaliações e obter a nota em tempo real. | - O *Aluno* possui o seguintes atributos: *nome*, *email*, *matrícula* e *senha*.<br>- Para evitar o cadastro de senhas erradas, deve-se solicitar ao *Aluno* confirmar a senha (digitando novamente).<br>- Uma vez cadastrado, testar o acesso com sucesso usando as credenciais criadas (*usuário*, *senha*).<br>- Uma vez cadastrado, testar o acesso com insucesso usando credenciais erradas (combinação de *usuário*/*senha* errada).<br>- Uma mensagem deverá ser informada para o *Aluno* em caso acesso com sucesso ou insucesso durante autenticação. |
| **US-0002** | Enquanto *Aluno* eu quero acessar uma avaliação disponível na *Test&Code* para implementar a resposta. | - Escolhida a disciplina, todas as avaliações da mesma devem ser mostradas.<br>- Para cada avaliação, será exibida uma opção para que possa ser feito o *download* pelo aluno, junto com com as informações desta (nome e data limite de entrega). |
| **US-0003** | Enquanto *Aluno* eu quero fazer o *upload* da avaliação finalizada para ver minha nota. | - O *Aluno* deve enviar a resposta, antes da data limite, acessando uma opção no mesmo local que fez *download* da avaliação da disciplina.<br>- A resposta do aluno deve estar um arquivo compactado (.zip) com as implementações pedidas.<br>- Fica a critério do *Professor* aceitar ou não a resposta, caso seja enviado após a data limite.<br>- A nota do *Aluno* deve ser exibida após correção automatizada. |

**Quadro A.2 - Histórias de usuário criadas para o papel *Aluno*.**
**Fonte: Quadro criado pelos autores.**

| ID | História de usuário | Critério de aceitação |
|---|---|---|
| **US-0001** | Enquanto *Administrador* eu quero conceder e revogar permissões de outros usuários. | - O *Administrador*, a partir do nome do usuário (*Professor*), pode conceder permissões de administrador bem como revogá-las. |
| **US-0002** | Enquanto *Administrador* eu quero ter informações sobre a quantidade de usuários (*Aluno* e *Professor*) que utilizam a ferramenta. | O *Administrador* tem acesso a uma tela com gráficos para observar a quantidade de alunos, professores e avaliações que foram criadas em cada mês. |

**Quadro A.3 - Histórias de usuário criadas para o papel *Administrador*.**
**Fonte: Quadro criado pelos autores.**

# APÊNDICE B - REPOSITÓRIO DA *TEST&CODE*

Os detalhes para instalação e configuração da *Test&Code*, bem como o código fonte, podem ser encontrados no repositório do GitHub (Figura B.1).

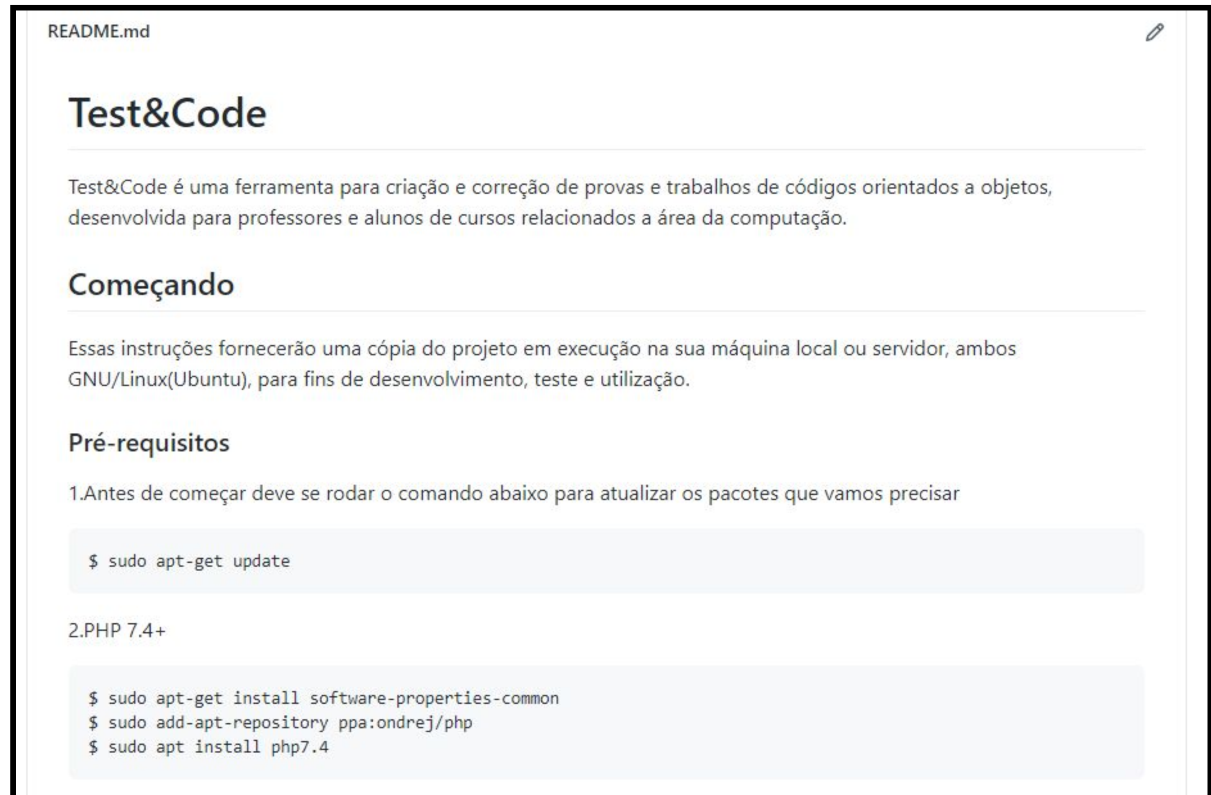


**Figura B.1 - Repositório da *Test&Code* no Github (**https://bit.ly/2IUwNat**).**

# APÊNDICE C - EXEMPLO DE AVALIAÇÃO ENVIADA PELO PROFESSOR

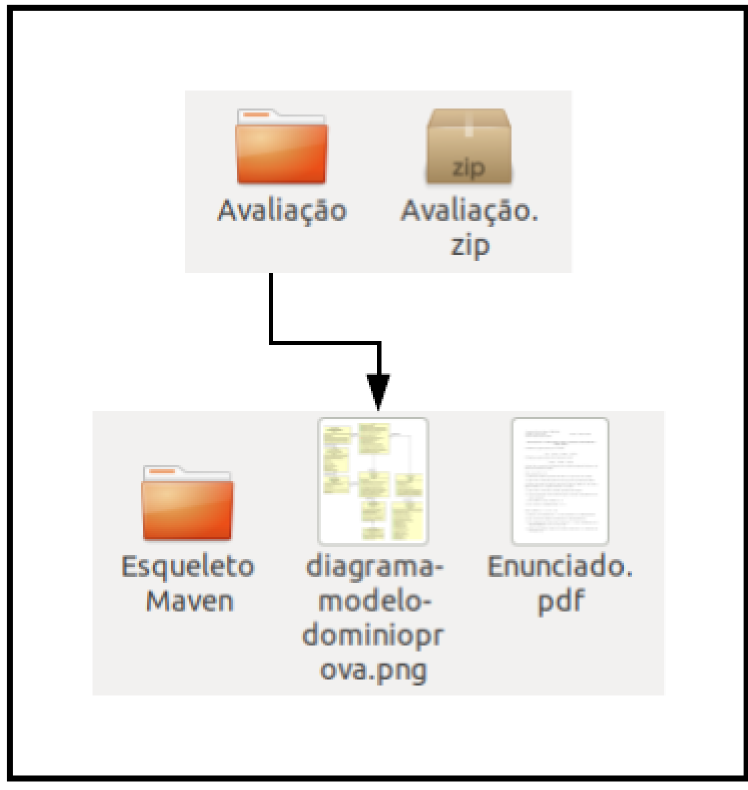


**Figura C.1 - Exemplo completo de avaliação enviada pelo *professor*.**

# ANEXOS

## ANEXO A - ENUNCIADO E DIAGRAMA DE CLASSES DO SISTEMA PDV USADOS COMO AVALIAÇÃO MANUSCRITA

Neste anexo está o enunciado usado em uma avaliação manuscrita e presencial no curso de Ciência da Computação na Universidade Federal de Alfenas e uma figura do diagrama de classes do sistema PDV que foi utilizado como complemento da avaliação.

| Curso: Bacharelado em Ciência da Computação | Período**:** 3º |
|---|---|
| Disciplina: Programação Orientada a Objetos | Valor:10 pontos (peso 0.35) |
| Professor: Rodrigo Martins Pagliares | Data: 05/06/2013 |
| Aluno(a): | |

**1. O Diagrama de classes no final da prova foi parcialmente implementado. Cabe a você terminar a implementação nos espaços indicados de acordo com o enunciado das questões presentes no FINAL da prova.**

**Observações**

- Classes completas ou trechos de uma classe não listados, mas presentes no diagrama de classes, são considerados implementados de forma correta e poderão ser criados e usados caso seja necessário.
- Os espaços em branco no decorrer do código servem para vocês terem noção do tamanho do gabarito em linhas. Todavia, todas as respostas devem estar na folha pautada entregue no início da avaliação.
- As **instruções import** foram omitidas para maior clareza.

⟶ associação (has a)

- - - - -▷ Implementação de iinterface

——▷ (is a) Herança

*Itálico* (Classe ou método abstrato)

```
package dominio;
public class Registradora {
   private String id;
   private Venda[] vendas = new Venda[10];
   private int contadorVendas;
   private CatalogoProdutos catalogo = new CatalogoProdutos();
   ....
   ....
  public void fazerPagamento(double quantiaFornecida, int operadora,
                                   int quantidadeParcelas, int tipoJuros) {
     Venda corrente = getVendaCorrente();
     ____________________________________________________ // Questão 2.1

   }

 public Venda getVendaCorrente() {
      Venda venda = null;
      _____________________________      // Questão 2.2
      return venda;
   }
```

---

```
package dominio;

public class Venda {
   private ItemVenda[] itensVenda;
   private boolean estaCompleta;
   private Pagamento pagamento;
   ...
   ...
  public void fazerPagamento(double quantiaFornecida, int operadora, int
                                   quantidadeParcelas, int tipoCalculadora) {
    ________________________________________________________ // Questão 2.3
  }
```

---

```
public class PagamentoCartao __________________________________{    // Questão 2.4
   private int operadora;
   private int quantidadeParcelas;
   private CalculadoraFinanceira calculadora;

   public PagamentoCartao(double quantia, int operadora, int quantidadeParcelas, int
                              tipoCalculadora) {
      __________________        // Questão 2.5
      this.operadora = operadora;
      this.quantidadeParcelas = quantidadeParcelas;
     ________________________________________ // Questão 2.6
     ________________________________________
```

```
      ____________________________________________
      ____________________________________________
      ___________________________________________
    }

    public double simularParcelas(double quantia, int quantidadeParcelas) {
      float juros = consultarTaxaJuros();
      double montanteComJuros = __________________________________ // Questão 2.7
      return montanteComJuros / quantidadeParcelas;
    }

     @Override
     public float consultarTaxaJuros() {
        __________________________________  // Questão 2.8
        __________________________________
        __________________________________
        __________________________________
        __________________________________
        __________________________________
        __________________________________
        __________________________________
        __________________________________
        __________________________________
    }

    @Override
    public String toString() {
      return "Tipo de pagamento...: Cartão de Crédito\n" +
           super.toString() + "\n"
           + "Operadora.........: " + Operadora.getOperadoraFormattedAsString(operadora)+"\n"
           + "Quantidade de parcelas....: " + quantidadeParcelas + "\n"
           + "Valor de cada parcela...: " + ____________________________//Questão 2.9
    }
}
```

--------------------------------------------------------------------------------------------------------

```
public __________ IJuros {          //Questão 2.10
    ______________________________

}
```

---

```
public class CatalogoProdutos {
    private DescricaoProduto[] descricoesProdutos;
    private int contadorDescricoesProdutos;

    CatalogoProdutos() {
      descricoesProdutos = new DescricaoProduto[10];
      DescricaoProduto d1 = new DescricaoProduto("01", 3.75, "Chocolate Talento");
      DescricaoProduto d2 = new DescricaoProduto("02", 1.50, "Chiclete Trident");
      DescricaoProduto d3 = new DescricaoProduto("03", 2.50, "Lata de Coca-cola");
      DescricaoProduto d4 = new DescricaoProduto("04", 2.00, "Agua Mineral Caxambu");
      DescricaoProduto d5 = new DescricaoProduto("05", 5.99, "Cerveja Corona extra");

      DescricaoProduto d6 = new DescricaoProduto("06", 2.50, "Biscoito cream cracker");
      DescricaoProduto d7 = new DescricaoProduto("07", 4.50, "Leite condensado");
      DescricaoProduto d8 = new DescricaoProduto("08", 18.00, "Cafe Prima Qualitat");
      DescricaoProduto d9 = new DescricaoProduto("09", 2.00, "Danete");
      DescricaoProduto d10 = new DescricaoProduto("10", 1.00, "Bombril");

      descricoesProdutos[contadorDescricoesProdutos] = d1;
      ____________________________________________________  //Questão 2.11
      ...
      ...
      ...
    }

    public DescricaoProduto getDescricaoProduto(String id)___________ {  //Questão 2.12
        ____________________________________________________  //Questão 2.13
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
        ____________________________________________________
    }
}
```

---

```
public class TestePDV {

  public static void main(String[] args) {

    Endereco endereco = new Endereco("Rua X", "", 5, "Alfenas", "Aeroporto", "MG",
                                     "37130-000");
    Loja loja = new Loja("Supermercado Preço Bão", endereco);

    Registradora registradora = loja.getRegistradora("R01");
    CatalogoProdutos catalogo = registradora.getCatalogo();

    _________________  //Questão 2.14
```

```
        registradora.criarNovaVenda();
        registradora.entrarItem("01", 3);
        registradora.entrarItem("02", 2);
        registradora.entrarItem("03", 1);

      ___________________________________________ //Questão 2.14
      ___________________________________________ //Questão 2.14
      ___________________________________________ //Questão 2.14

      registradora.finalizarVenda();
      double totalVenda = registradora.getVendaCorrente().calcularTotalVenda();
      registradora.fazerPagamento(totalVenda, _________, 1, __________);  //Questão 2.15

       //Pagamento com cartão não gera troco
       gerarRecibo(registradora, 0.0);

   }

   public _______ void gerarRecibo(Registradora registradora, double troco) { //Questão 2.16
      Venda venda = registradora.getVendaCorrente();
      System.out.println("");
      System.out.println("---------------------- Supermercado Preço Bão -----------------------");
      System.out.println("                          Registradora : " + registradora.getId());
      System.out.println("\t\t\t\tCUPOM FISCAL");
      System.out.println(venda);
      System.out.println("Troco................: R$ " + troco);
   }
}
```

**Questão 2.1** – Qual trecho de código deverá ser inserido aqui?

**Questão 2.2** - Termine a implementação deste método obtendo a venda corrente armazenada no vetor vendas

**Questão 2.3** – O pagamento adequado deverá ser criado nesta linha de código

**Questão 2.4 -** De acordo com o diagrama de classes, qual código deverá ser inserido aqui?

**Questão 2.5 –** Qual código deverá ser inserido aqui?

**Questão 2.6 –** Se o argumento **tipoCalculadora** for de juros simples, um objeto do tipo **CalculadoraJurosSimples** deve ser criado e associado corretamente à variável de instância correspondente. Caso o objeto **tipoCalculadora** seja de juros compostos, ,um objeto do tipo **CalculadoraJurosCompostos** deve ser criado e associado corretamente à variável de instância correspondente.

**Questão 2.7** – Use a calculadora da questão 2.6 para calcular o montante com juros.

**Questão 2.8 -** Implemente este método com a seguinte lógica: para uma parcela, sem juros; para 2 parcelas, taxa de juros de 2,5%; para 3 parcelas, taxa de juros de 5.0%.

**Questão 2.9 -** O método simular parcelas para obter o valor de cada parcela deverá ser invocado neste trecho

**Questão 2.10 - IJuros** é uma interface com apenas um método

**Questão 2.11 –** Qual código deverá ser inserido aqui?

**Questão 2.12 -** Este método pode gerar uma exceção **DescricaoProdutoInexistente**

**Questão 2.13 -** Implemente este método

**Questão 2.14 -** O método **entrarItem** pode gerar uma exceção do tipo **DescricaoProdutoInexistente** que deverá ser tratada aqui nestas lacunas.

**Questão 2.15 -** O pagamento é feito com cartão de crédito passando como argumento a operadora **AMERICAN** e usando uma calculadora de juros simples.

**Questão 2.16 -** Complete esta lacuna. Por que seu preenchimento é necessário?

Boa Prova !
Rodrigo Martins Pagliares

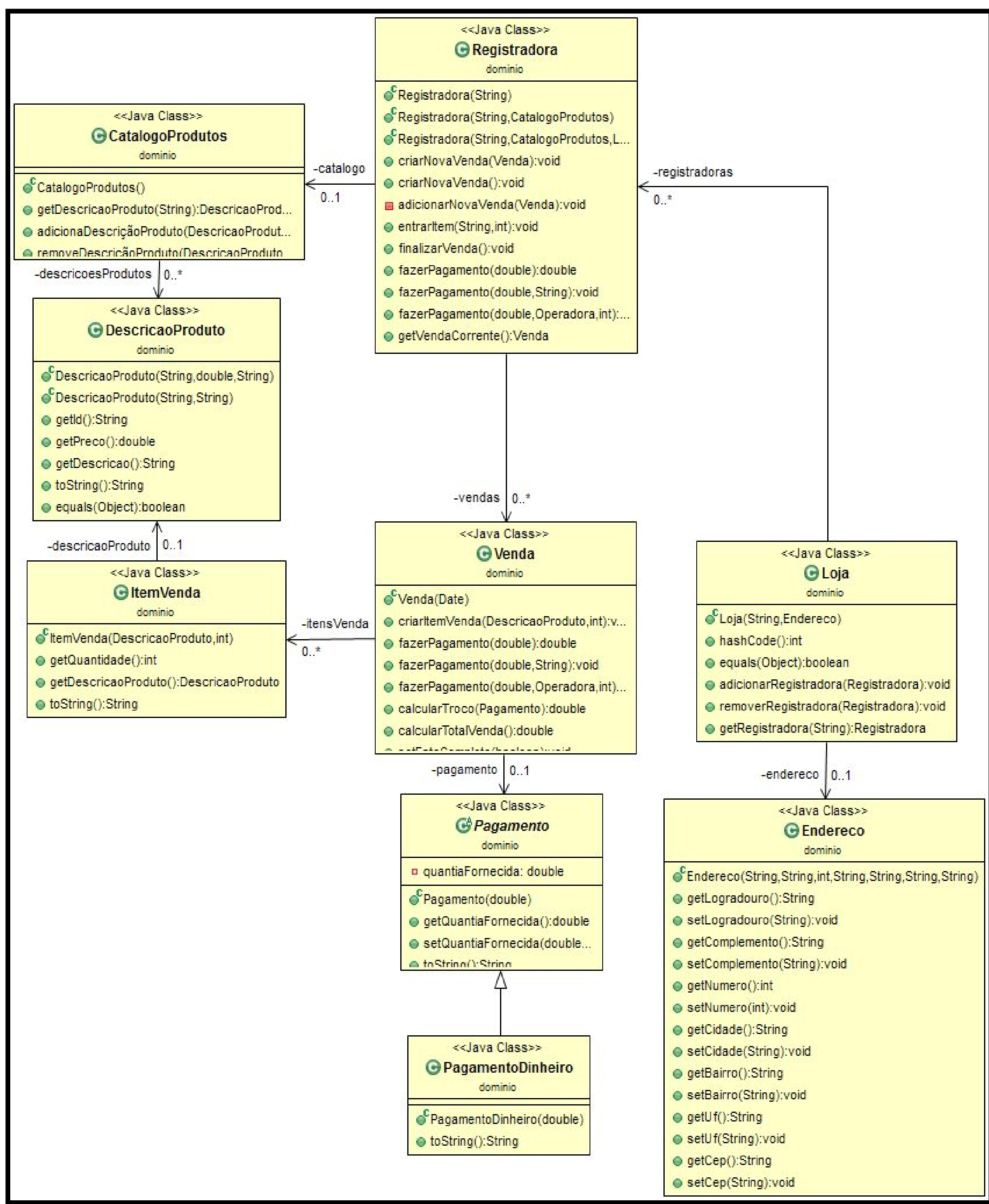


**Figura A.1 - Diagrama de classes do sistema PDV entregue aos alunos como parte do enunciado da avaliação presencial e manuscrita.**

## ANEXO B - EXEMPLOS DE GABARITO DO SISTEMA PDV UTILIZADO PARA VALIDAR A FERRAMENTA *TEST&CODE*

Neste anexo apresentamos dois exemplos de testes do sistema PDV utilizados na validação da *Test&Code*.

Abaixo vemos um exemplo de testes em uma classe de domínio (Código B.1), onde nas Linhas 10 a 17 é feito o preparo da classe *Endereco* com as informações esperadas (gabarito) e posteriormente por meio de chamadas ao método `assertEquals()` (Linhas 23 a 29), verifica-se se os dois parâmetros passados são iguais, sendo um o que deveria ser o correto e outro gerado a partir de uma função da classe de domínio correspondente ao código do aluno.

```
01  package testesunitarios;
02
03  import org.testng.annotations.Test;
04  import.testng.annotations.BeforeMethod;
05  import static org.testng.Assert.*;
06  import dominio.*;
07
08  public class EnderecoTest {
09
10     private Endereco endereco;
11
12     @BeforeMethod
13     public void setUp() {
14        endereco = new Endereco("Rua Gabriel Monteiro da
15        Silva", "Universidade", 700, "Alfenas", "Centro",
16        "MG", "37130000");
17      }
18
19     @Test
20     public void construtorDeveSetarAsInformacoesDeEndereco(){
```

```
21        assertEquals(endereco.getLogradouro(),
22                     "Rua Gabriel Monteiro da Silva");
23        assertEquals(endereco.getComplemento(),
24                     "Universidade");
25        assertEquals(endereco.getNumero(), 700);
26        assertEquals(endereco.getCidade(), "Alfenas");
27        assertEquals(endereco.getBairro(), "Centro");
28        assertEquals(endereco.getUf(), "MG");
29        assertEquals(endereco.getCep(), "37130000");
30    }
31 }
```

**Código B.1 - Classe EnderecoTest**

Para este outro exemplo na Figura B.2, observa-se a utilização de *mocks* nas Linhas 18, 23 e 34, com auxílio do framework *Mockito*, que cria um objeto simulado de forma controlada para que possa ser feito testes de uma classe de domínio em conjunto com este objeto.

```
01 package testescomdubles;
02
03 import org.testng.annotations.Test;
04 import org.testng.annotations.BeforeMethod;
05 import static org.testng.Assert.*;
06 import static org.mockito.Mockito.*;
07
08 import dominio.*;
09
10 //Classe Ok
11 public class LojaTest {
12
13    Loja loja;
14
15    @BeforeMethod
16    public void setUp() {
```

```
17            //Criando um Dummy de Endereço
18         loja = new Loja("C&A", mock(Endereco.class));
19      }
20
21      @Test()
22      public void deveVerificarSeARegistradoraFoiCriada() {
23         Registradora registradora = mock(Registradora.class);
24         when(registradora.getId()).thenReturn("06");
25
26         loja.adicionarRegistradora(registradora);
27
28         assertTrue(registradora.equals(
29                    loja.getRegistradora("06")));
30      }
31
32      @Test()
33      public void deveVerificarSeARegistradoraFoiRemovida() {
34         Registradora registradora = mock(Registradora.class);
35         when(registradora.getId()).thenReturn("06");
36
37         loja.adicionarRegistradora(registradora);
38         loja.removerRegistradora(registradora);
39
40         assertNull(loja.getRegistradora("06"));
41      }
42   }
```

**Código B.2 - Classe LojaTest**