**UNIVERSIDADE FEDERAL DO CEARÁ**

**CAMPUS DE QUIXADÁ**

**CURSO DE SISTEMAS DE INFORMAÇÃO**

**ALBENOR ARAÚJO FILHO**

# UM FRAMEWORK PARA AVALIAÇÃO AUTOMÁTICA DE PRÁTICAS DE PROGRAMAÇÃO BASEADO EM TESTES DE SOFTWARE

**QUIXADÁ**

**2016**

**ALBENOR ARAÚJO FILHO**

# UM FRAMEWORK PARA AVALIAÇÃO AUTOMÁTICA DE PRÁTICAS DE PROGRAMAÇÃO BASEADO EM TESTES DE SOFTWARE

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.
Área de concentração: Computação
Orientador Prof. Regis Pires Magalhães

**QUIXADÁ**

**2016**

**ALBENOR ARAÚJO FILHO**

# UM FRAMEWORK PARA AVALIAÇÃO AUTOMÁTICA DE PRÁTICAS DE PROGRAMAÇÃO BASEADO EM TESTES DE SOFTWARE

Trabalho de Conclusão de Curso submetido a Coordenação do Curso de Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel. Área de concentração: computação

Aprovado em ___/ Julho/ 2016

BANCA EXAMINADORA

_____

Prof. MSc. Regis Pires Magalhães (Orientador)
Universidade Federal do Ceará – UFC

_____

Prof. MSc. Lívia Almada Cruz
Universidade Federal do Ceará – UFC

_____

Prof. MSc. Victor Aguiar Evangelista de Farias
Universidade Federal do Ceará – UFC

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

A687f Araújo Filho, Albenor.
Um framework para avaliação automática de práticas de programação baseado em testes de software / Albenor Araújo Filho. – 2016.
56 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2016.
Orientação: Prof. Me. Regis Pires Magalhães.

1. Desenvolvimento de software. 2. Teste e avaliação de software. 3. Frameworks. I. Título.
CDD 005

---

À meu pai, Albenor.

À minha mãe, Gorete.

Aos meus amigos.

Aos meus professores.

## AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado saúde e força para superar todas as dificuldades, por todas as oportunidades que me foram dadas e por todas as experiências que pude viver nessa Universidade.

Agradeço a toda minha família, em especial aos meus pais Albenor e Gorete, e minha irmã Daniela, que com muito carinho e apoio, não mediram esforços para que eu chegasse até essa etapa de minha vida.

Agradeço a todos os meus amigos de Piquet Carneiro e amigos do CSF que sempre se importaram com a minha vida acadêmica, e em todas as dificuldades me deram estímulo para seguir em frente.

Agradeço aos grandes amigos que fiz nessa Universidade, com os quais vivi muitos momentos importantes e inesquecíveis da minha vida. Em especial, Sávio, Ricardo, Rafael, Wanrly, Junior, Alex, Alexsandro, Adeilson, Yago, Caio, William, Claudio, Cintia, Tercio, Klyssia, Bruno, Anderson, Danrley e Guilherme.

Agradeço em especial aos amigos Junior Leonel, Daniel Filho, Alysson Gomes, Alexsandro Oliveira e Alex Oliveira por terem me dado muito suporte e incentivo para a realização desse trabalho.

Agradeço a todo o grupo PET-SI e aos professores tutores Davi Romero e Lucas Ismaily pelos grandes ensinamentos que pude adquirir participando desse grupo.

Agradeço a todos os professores do campus UFC Quixadá pelo excelente trabalho que exercem, nos dando ensinamentos impagáveis.

Agradeço ao meu orientador, professor Regis Pires Magalhães, que sempre acreditou na minha capacidade e me deu o estímulo e apoio necessários para continuar.

“If you can't fly then run, if you can't run then walk, if you can't walk then crawl, but whatever you do you have to keep moving forward.” (Martin Luther King Jr.)

**Resumo**

Um dos principais problemas para o desenvolvimento dos alunos nos cursos de computação é a dificuldade de aprendizado em programação no decorrer da disciplina. Isso pode ser relacionado a falta de prática, considerando que por demandar de um tempo muito grande para corrigir os trabalhos de cada aluno, os professores acabem não tendo condições de dar um rápido *feedback* para eles. Como tentativa de incentivar a intensificação de atividades práticas de programação e rápido *feedback*, este trabalho busca o desenvolvimento de um *framework* que automatize o processo de correção de trabalhos de programação dos alunos utilizando casos de teste desenvolvidos pelo professor para avaliá-los. Dessa forma, o aluno poderá ter um *feedback* automático sobre seu trabalho e o professor não precisará empregar muito tempo para isso.

Palavras chave: Desenvolvimento de software. Teste e avaliação de software. Frameworks.

**ABSTRACT**

One of the main problems related to the development of skills of computer science students is the difficulty in learning programming languages in introductory courses. This may be related to the lack of programming assignments in class, since the evaluation of such assignments demands much time, which leads professors to exclude these assignments from their curriculum. In an attempt to alleviate such problem, this project main goal is to develop a *framework* that automates the evaluation process of programming assignments by using test cases developed by the professors. Thus, the students can receive fast *feedback* about their assignments and the professors do not need to spend much time evaluating them.

Keywords: Software development. Software testing and evaluation. Frameworks

## LISTA DE ILUSTRAÇÕES

Figura 1- Estrutura do JUnit ..... 14
Figura 2 - Procedimentos metodológicos ..... 22
Figura 3 - Diagrama de Caso de uso ..... 24
Figura 4 - Diagrama de classes ..... 24
Figura 5 - Primeiro fluxo de execução do framework ..... 26
Figura 6 - Exemplo de método de teste para ser criado ..... 28
Figura 7 - Novo fluxo de execução do framework ..... 28
Figura 8 - Diagrama de classes do framework ..... 30
Figura 9 - Entradas e saídas do framework ..... 31
Figura 10 - Fluxo de interação do professor com o ambiente web ..... 32
Figura 11 - Fluxo de interação do aluno com o ambiente ..... 33
Figura 12 - Página de criação de teste do professor ..... 34
Figura 13 - Pagina inicial do aluno ..... 34
Figura 14 - Página de submissão do aluno ..... 35
Figura 15 - Página de resultado após submissão ..... 35
Figura 16 – Hierarquia de diretórios ..... 36
Figura 17 - Questionário feito aos alunos ..... 37

**SUMÁRIO**


1 INTRODUÇÃO ..... 10

2 FUNDAMENTAÇÃO TEÓRICA ..... 12
2.1 Teste de Software ..... 12
2.2 Ferramentas de Teste ..... 13
2.2.1 JUnit ..... 14
2.3 Avaliação Automática ..... 14
2.3.1 Carregamento dinâmico de classes Java com *classloaders* ..... 15
2.3.2 Reflexão ..... 16
2.3.3 Framework de avaliação ..... 17

3 TRABALHOS RELACIONADOS ..... 18
3.1 Ferramenta MOJO ..... 18
3.2 Testador automático e método de avaliação de programas em Java ..... 19
3.3 Ferramenta ProgTest ..... 20

FERRAMENTA MOJO ..... 21

TESTADOR AUTOMÁTICO E MÉTODO DE AVALIAÇÃO DE PROGRAMAS EM JAVA ..... 21

FERRAMENTA PROGTEST ..... 21

4 PROCEDIMENTOS METODOLÓGICOS ..... 22
4.1 Avaliação dos trabalhos relacionados ..... 22
4.2 Etapa de documentação e projeto ..... 23
4.3 Etapa de implementação ..... 25
4.3.1 Primeira estrutura desenvolvida. ..... 25
4.3.2 Nova estrutura do framework ..... 27
4.3.3 Padrões de desenvolvimento de casos de teste para avaliação ..... 29
4.3.4 Políticas de desenvolvimento de classes pelo aluno ..... 30
4.3.5 Ambiente web para execução do framework de avaliação desenvolvido ..... 31

5 AVALIAÇÃO DO FRAMEWORK ..... 37

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS ..... 40

REFERÊNCIAS ..... 42

APÊNDICE A ..... 43

APÊNDICE B ..... 51

APÊNDICE C ..... 56

# 1 INTRODUÇÃO

A programação constitui uma base essencial para as disciplinas dos cursos de computação. Ela constitui uma maneira de organizar o pensamento de forma coerente para expressar instruções para o computador. Segundo Forbellone e Eberspacher (2005), ela permite escolher caminhos para resolver problemas conhecidos, aplicando a lógica para criar algoritmos que podem ser implementados usando uma linguagem de programação. A computação visa a busca de soluções para problemas, o que torna inquestionável a importância da programação nesse ramo. Na maioria dos cursos de computação são trabalhados fundamentos de programação logo no primeiro semestre para introduzir conceitos que são necessários em várias disciplinas no decorrer do curso.

É comum ver nos primeiros semestres alunos com dificuldades em aprender a programar. A dificuldade em aprender os conceitos de programação refletem em um grande índice de reprovação, problemas com matérias que usam programação como base, e desistência do curso. Segundo Moreira e Favero (2009), a dificuldade em aprender programação pode ser consequência de vários fatores como por exemplo, base matemática fraca, má interpretação de problemas e não entendimento do assunto. Essa dificuldade também pode estar relacionada a pouca realização de atividades práticas ou mesmo a dificuldade de prover um *feedback* rápido para as atividades que são realizadas em sala de aula. Um *feedback* rápido é importante para que os alunos possam saber onde estão errando e já trabalhar em uma correção. Isso estimula o melhor aprendizado do aluno baseado na prática.

Para que as atividades práticas de programação sejam melhor aproveitadas nas disciplinas, faz-se necessário que essas atividades sejam avaliadas. Para isso, demanda-se um grande acompanhamento do professor, o que muitas vezes é insuficiente devido ao tempo reduzido que ele tem para gerenciar todas as disciplinas que leciona, o grande número de alunos e a grande quantidade de trabalhos para análise. Além disso, hoje em dia ainda há disciplinas de programação onde as atividades práticas utilizam o tradicional “papel e caneta”, tornando o processo de avaliação muito mais demorado. Assim, a longa espera do aluno por um *feedback* sobre avaliações e exercícios, por exemplo, ou para que sejam apresentadas as suas notas, pode acabar influenciando para desmotivação e até desistência do curso.

Com base nesses problemas, vários trabalhos foram desenvolvidos para incentivar a prática nas disciplinas e diminuir o esforço do professor como, por exemplo, a ferramenta MOJO (CHAVES et al., 2013) que integra o ambiente virtual de aprendizagem Moodle a juízes online para prover um ambiente para elaboração, submissão e avaliação de atividades de programação. Outro bom exemplo é o ambiente ProgTest (CORTE, 2006), um sistema de submissão e avaliação de práticas de programação e casos de teste, desenvolvido para auxiliar as disciplinas de fundamentos de programação e teste de software. Ambas as ferramentas têm como objetivo automatizar o processo de avaliação de práticas de programação, propondo amenizar a insuficiência no envolvimento do professor nessas práticas e despertar nos alunos uma maior familiaridade com a programação através delas.

Este trabalho propõe o desenvolvimento de um *framework* para apoiar as disciplinas dos cursos de computação no que diz respeito à avaliação de práticas de programação. A proposta é que esse *framework* utilize casos de teste desenvolvidos pelo professor para avaliar automaticamente as atividades de programação na linguagem Java que são submetidas pelos alunos. A intenção é dar um suporte maior às disciplinas de programação, auxiliando o professor a controlar melhor as atividades práticas e proporcionando um *feedback* mais rápido aos alunos. Embora já existam ferramentas que podem ser utilizadas com esse fim, o desenvolvimento deste trabalho teve como estímulo a experiência prática no desenvolvimento de um mecanismo que pudesse trazer benefícios para alunos e professores utilizando os conhecimentos obtidos no decorrer da graduação. O trabalho aqui descrito teve como metodologia selecionar com base nas ferramentas existentes, quais funcionalidades seriam mais adequadas para serem implementadas, levantar requisitos para o desenvolvimento desse *framework*, como também também requisitos de uma futura plataforma web que possa utiliza-lo, e implementar o *framework* para que pudesse ser utilizado em exercícios práticos de programação na linguagem Java, como listas de exercícios ou até mesmo provas práticas.

O trabalho foi dividido em seis capítulos. No capítulo 2 é apresentada a fundamentação teórica, onde são descritos os conceitos que foram necessários para o desenvolvimento do trabalho. O capítulo 3 apresenta uma breve descrição e comparativo sobre alguns trabalhos relacionados. Os procedimentos metodológicos para o desenvolvimento do trabalho são descritos em detalhes no capítulo 4. O capítulo 5 descreve uma avaliação feita e o capítulo 6 descreve algumas considerações finais e sugestões de trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão descritos os principais conceitos relacionados a este trabalho, e como cada conceito contribui com o desenvolvimento do mesmo. Estão descritos os conceitos de Teste de Software, Ferramentas de Teste, e Avaliação Automática.

## 2.1 Teste de Software

O processo de desenvolvimento de software envolve uma série de atividades nas quais, apesar das técnicas, métodos e ferramentas empregados, erros no produto ainda podem ocorrer. A atividade de teste consiste em uma análise dinâmica do produto. Essa é uma atividade relevante para a identificação e eliminação de erros que persistem. (MALDONADO et al, 2004). Pressman (2011) define teste de software como um conjunto de atividades que podem ser planejadas com antecedência e executadas sistematicamente.

Segundo Rocha (2005), o teste de software envolve quatro etapas que devem ser executadas durante o desenvolvimento do software: planejamento de testes, projeto de casos de teste, execução de testes e coleta dos resultados, além de avaliação dos resultados coletados. Essas etapas devem realizar-se em três fases de teste: teste de unidade, teste de integração e teste de sistema.

O teste de unidade testa cada módulo do software individualmente, buscando erros de lógica e verificando se cada módulo está funcionando adequadamente. O teste de integração é uma técnica para integrar os módulos componentes da estrutura de software, visando identificar erros de interface entre eles. O teste de sistema verifica se todos os elementos do sistema combinam-se adequadamente e se o desempenho global é atingido na perspectiva dos requisitos do usuário. (PRESSMAN, 2005).

Para este trabalho, o propósito de usar teste de software teve foco no processo de avaliação automática de atividades de programação, utilizando os testes para avaliar as atividades submetidas pelos alunos.

O teste de software tem como propósito verificar a existência de falhas. Tentar testar todos os possíveis valores de entrada seria muito exaustivo e demandaria de muito trabalho, tornando essa tarefa quase impossível em determinados casos. Portanto, para obter testes eficientes, deve-se procurar criar conjuntos de casos de teste que possam cobrir grande parte de possíveis defeitos no software que será testado.

Corte (2006) afirma que, para que a atividade de teste possa ser conduzida de forma sistemática e teoricamente fundamentada, faz-se necessária a aplicação de critérios de teste.

Os critérios de teste podem ser estabelecidos através de algumas técnicas: estrutural, baseada em erros e funcional.

Na técnica de teste estrutural, os requisitos de teste derivam-se a partir dos aspectos de implementação do software, já na técnica baseada em erros, os requisitos de teste são definidos baseados em erros comuns que podem ocorrer durante o processo de desenvolvimento de software. Neste trabalho, a técnica que pode ser utilizada é a técnica de teste funcional.

A técnica de teste funcional, também denominada de caixa preta, aborda o desenvolvimento de testes através da especificação do software. Com base no que está descrito é que são formulados os critérios de teste. O objetivo, segundo Rocha (2005), é encontrar distinções entre o comportamento atual do sistema e o que está descrito em sua especificação. Pressman (2005) descreve que o teste funcional procura encontrar erros relacionados a funções incorretas ou omitidas; erros de interface; erros de estrutura de dados ou de acesso a dados externos; erros de comportamento e desempenho e, por fim, erros de iniciação e término. Tomando como base que o professor desenvolverá seus testes segundo o que especificar ao aluno para implementar, essa é a técnica que mais se enquadraria com o *framework* aqui proposto.

## 2.2 Ferramentas de Teste

A qualidade e a produtividade da atividade de teste são dependentes do critério de teste utilizado e da existência de uma ferramenta que o suporte. Sem a existência de uma ferramenta, a aplicação de um critério torna-se uma atividade propensa a erros e limitada a programas muito simples. (DOMINGUES, 2002).

Com o crescimento da complexidade de software ao longo do tempo, o uso de ferramentas que automatizem o processo de teste de software se torna mais do que conveniente e justificável. O processo de teste pode se tornar muito mais trabalhoso e suscetível a erro, sem a utilização dessas ferramentas. O uso de ferramentas de teste automatiza o processo, tornando-o muito mais rápido e preciso. Assim como afirma Rocha (2005), a automatização permite verificação rápida e eficiente das correções de defeitos, agiliza o processo de depuração e permite a captura e análise dos resultados de teste de forma consistente. Para realizar a avaliação automática dos trabalhos submetidos pelos alunos, propõe-se para esse trabalho que os professores construam casos de teste para cada atividade que elaborar. A intenção é que o *framework* possa trabalhar também com casos de teste

compatíveis com o JUnit. A idéia é que o professor, tendo conhecimento de testes de software, possa utilizar esse conhecimento para testar da forma que achar mais adequada as práticas de programação dos alunos. Se o professor tiver conhecimentos sobre a ferramenta JUnit que possam melhorar a qualidade de seus testes, o *framework* suportará a execução dos mesmos. Caso não possua experiência em JUnit, o professor pode fazer um caso de teste normal, sem utilização do JUnit para isso.

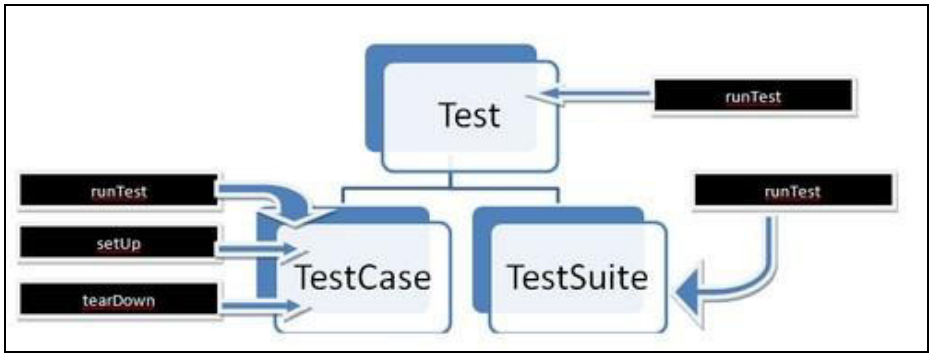
### 2.2.1 JUnit

O JUnit é um *framework open-source* utilizado para facilitar o desenvolvimento de códigos em Java, verificando se os resultados gerados pelos métodos são os esperados. Caso não sejam, o JUnit exibe os possíveis erros que estão ocorrendo nos métodos. Essa verificação é chamada de teste unitário, ou de unidade. (SILVA e Jorge, 2008).

O teste de unidade testa métodos do código, garantindo uma melhor qualidade do produto no processo de desenvolvimento. No caso da linguagem Java, esse teste é feito pelo JUnit em cada método separadamente.

A API do JUnit é organizada como mostra a Figura 1. Existe uma classe 'Test' que contém um método runTest que tem a função de realizar testes particulares. A classe TestCase, que testa os resultados de um método, e a classe TestSuite que permite o agrupamento de múltiplos casos de teste em uma coleção, bem como a execução sequencial de todos eles a partir de uma única chamada.

**Figura 1- Estrutura do JUnit**




Fonte: Portal JUnit Wikidot

## 2.3 Avaliação Automática

Vários trabalhos que têm como objetivo auxiliar o aprendizado de programação, abordam o desenvolvimento de ferramentas que fazem avaliação automática de atividades de programação. A avaliação automática tem como objetivo verificar se determinada solução

está correta ou não, permitindo uma avaliação precisa de uma grande quantidade de atividades a partir de critérios estabelecidos. Uma grande vantagem disso é que os professores podem inserir mais atividades práticas de programação nas disciplinas de computação, pois não precisarão se preocupar tanto com o tempo e o esforço necessários para corrigir uma grande parcela de atividades. Embora ainda haja um certo esforço para preparar os testes, ele é pouco se comparado ao esforço necessário para avaliar muitos trabalhos sem uma ferramenta que possa automatizar esse processo.

Em cada caso, essa avaliação pode se basear em critérios diferentes. No caso da ferramenta MOJO (CHAVES et al., 2013), a avaliação automática das atividades de programação é feita utilizando juízes online que analisam se o código compila e se está retornando o que é esperado. A ferramenta ProgTest (Corte, 2006) utiliza para a avaliação automática, comparações de resultados da execução dos códigos do professor e dos códigos dos alunos e usa uma ferramenta de testes para fazer testes de cobertura nos casos de teste submetidos. O avaliador de Nunes e Lisbôa (2004) avalia com testes estruturais e de estado, se os códigos estão sendo implementados corretamente. Todos os trabalhos analisados realizam a avaliação utilizando seus critérios e atribuem uma nota de acordo com os pesos dados a cada critério. O trabalho que aqui é proposto utiliza, para a avaliação automática de cada atividade, casos de testes desenvolvidos pelos professores. São atribuídos valores para cada caso de teste que dão composição a uma pontuação ao final do processo de avaliação. Dessa forma, o aluno terá um rápido *feedback* e o professor terá menos esforço, especialmente se a quantidade de atividades for relativamente grande para justificar o esforço para criação dos testes. Acoplado a uma plataforma web, o aluno pode submeter o seu trabalho, e automaticamente visualizar sua nota ao fim da submissão.

Para construir esse *framework* de avaliação automática, foi necessário entender como trabalhar com classes que não estão no projeto corrente, afinal, classes de caso de teste feitas pelo professor e classes de implementação feitas pelos alunos serão a entrada do *framework*, não pertencendo ainda ao projeto corrente. Com isso, foram necessários os estudos de alguns conceitos para fazer o carregamento dinâmico de classes e para chamar métodos de classes recém carregadas. Foram estudados os conceitos de *classloaders* e reflexão, que serão descritos a seguir.

### 2.3.1 Carregamento dinâmico de classes Java com *classloaders*

Quando estamos escrevendo um código em Java, ao fazermos uso de classes externas ao projeto, essas classes são carregadas por algum mecanismo. Esse carregamento, utilizando

*IDEs* como o Eclipse[1], por exemplo, se torna fácil pois pode acontecer automaticamente quando adicionamos o nome do pacote ao código escrito ou, manualmente, se utilizarmos as funcionalidades da IDE para adicionar as classes ao projeto. Porém, podem haver casos, como no trabalho aqui apresentado, em que o carregamento dessas classes precise ser dinâmico, ou seja, em tempo de execução. Para isso foi necessário entender como classes são carregadas em tempo de execução.

Na linguagem Java, o mecanismo responsável por fazer o carregamento de classes se chama *classloader*. *Classloader* é uma classe que carrega outras classes. Sendo uma classe, o classloader também precisa ser carregado e isso é feito através de código nativo do Java. Tudo em Java é carregado através de *classloaders* específicos. Por padrão, o *classloader* carrega todas as classes que estiverem presentes no *classpath*.

O *Classloader* permite carregar classes diferentes com o mesmo nome de forma distinta. Para isso basta que as classes pertençam a instâncias de *classloader* diferentes. Além disso, podemos criar *classloaders* customizados para se adaptar melhor às necessidades da aplicação. Para este trabalho, foi utilizado um *classloader* customizado que herda de *URLClassLoader*. Com ele é possível adicionar um arquivo de extensão *.JAR* inteiro dentro do *classpath*. As classes são adicionadas pela URL e guardadas pelo *URLClassloader* com o caminho e o nome do *classloader* utilizado. Dessa forma, as classes carregadas não entram em conflito. Ao final do carregamento, é possível fechar o *classloader* para que as classes não utilizadas sejam coletadas pelo *garbaje colector* da linguagem Java.

### 2.3.2 Reflexão

Java é uma linguagem compilada e de forte tipagem. Quando se desenvolve em Java, tudo tem que estar bem definido. Os atributos e métodos da aplicação são bem conhecidos por todo o escopo do projeto corrente. As chamadas de método, instanciação de objetos e as classes importadas são escritas antes da compilação. Mas existem situações, como nos trabalhos relacionados aqui citados, onde precisamos ter características dinâmicas para que classes e métodos sejam carregados após o desenvolvimento, em tempo de execução. A *IDE* Eclipse, por exemplo, permite a instalação de plug-ins ao software já desenvolvido dela. O software não precisa ser refatorado para que o plug-in possa funcionar. Ele adiciona as funcionalidades desse plug-in de forma dinâmica.

Sabendo que o carregamento de classes Java em tempo de execução é possível, é necessário saber como invocar métodos dessas classes sem que se saiba o nome dos mesmos.

[1] https://eclipse.org/

Existem diversos trabalhos que realizam testes nos métodos dos trabalhos submetidos, mas a aplicação, no momento do desenvolvimento, não tem essa informação. É possível invocar métodos de classes dinamicamente carregadas ou até mesmo criar instâncias dessas classes. Para isso podemos usar reflexão. A reflexão é um recurso que trabalha com a execução de métodos mesmo que a aplicação não tenha conhecimento dos nomes dos mesmos. Para isso basta que as classes estejam no *classpath.* A linguagem Java tem um pacote chamado *java.lang.reflect* que permite a criação de chamadas dinâmicas a classes sem a necessidade de conhecê-las quando estamos escrevendo nosso código. Assim, aplicações como as que são aqui descritas são capazes de trabalhar com funcionalidades que só serão descobertas em tempo de execução. Algumas funcionalidades que a reflexão possibilita são:

- Listar atributos presentes em uma classe e obter seus valores em objetos;
- Instanciar classes de forma dinâmica;
- Invocar métodos dinamicamente por meio de uma *string* com o nome do método;

Como benefícios do uso de reflexão, podemos citar o ganho de produtividade. Se seguirmos pelo ponto de vista de que não há necessidade de reescrever o código toda vez que for necessário adicionar uma nova funcionalidade. Podemos citar também a ideia de se ter um software extensível, onde funcionalidades podem ser "acopladas" ao software já desenvolvido.

O *framework* desenvolvido neste trabalho usa reflexão para fazer a chamada e instanciação de classes de forma dinâmica. Casos de teste e implementação dos alunos, após carregados para o *classpath*, podem ser instanciados e seus métodos podem ser invocados a partir de métodos do pacote *java.lang.reflect*, como o *Method.invoke()*, *Method.getDeclaredMethod().*

### 2.3.3 Framework de avaliação

Segundo Jaques (2016), um *framework* é um conjunto de códigos abstratos e/ou genéricos, geralmente classes, desenvolvidos em alguma linguagem de programação, que relacionam-se entre si para disponibilizar funcionalidades específicas ao desenvolvedor do software. O trabalho aqui desenvolvido trabalha o desenvolvimento de um mecanismo que usa de funcionalidades de outras bibliotecas para prover avaliação automática de código Java, por meio do carregamento dinâmico de classes de teste em conjunto com as classes a serem testadas para que sejam executadas. Portanto, utilizamos a denominação *framework* para o mecanismo criado. O *framework* criado se assemelha a um *framework* de testes, porém se difere por contabilizar uma pontuação que será obtida pelos métodos de teste.

# 3 TRABALHOS RELACIONADOS

Existem vários trabalhos que foram desenvolvidos em busca de alternativas para facilitar o aprendizado de programação. Dentre esses trabalhos, muitos adotam como alternativa para o aprendizado o desenvolvimento de ferramentas que dão subsídios para permitir uma maior aplicação de atividades práticas nas disciplinas de programação. A seguir serão descritos três trabalhos que adotam o desenvolvimento de ferramentas como alternativa para aprendizado, bem como as semelhanças e diferenças com o trabalho aqui proposto.

## 3.1 Ferramenta MOJO

Analisando as dificuldades encontradas no aprendizado de programação e, também, na dificuldade no gerenciamento das atividades práticas, Chaves et. al. (2013) propuseram um ambiente para automatizar a elaboração, submissão e avaliação das atividades práticas propostas pelo professor para desenvolvimento do aluno nas linguagens de programação. Trata-se da ferramenta MOJO, um mecanismo que integra o ambiente virtual de aprendizagem Moodle[2] com juízes online. Essa integração tem como justificativa obter as funcionalidades de ambos os sistemas para prover um ambiente coeso e completo para auxiliar professores e alunos. O Moodle é responsável por fornecer a interface e o conjunto de atividades para gerenciar o acompanhamento das atividades de programação. E os juízes online, que são sistemas comumente utilizados em maratonas de programação, são responsáveis por fazer a avaliação das atividades de programação que são submetidas no sistema.

Os juízes online que são integrados ao Moodle fazem uma avaliação automática dos trabalhos submetidos. Os juízes online integrados foram o Servidor URI Online Judge[3], Servidor SPOJ[4], e o servidor UVa Online Judge[5]. Eles recebem o código fonte que é enviado pelo aluno e posteriormente compilam e executam esse código. Durante a execução, os juízes online utilizam dados formatados como a entrada do programa, processam esses dados e realizam uma comparação dos resultados obtidos com os resultados que são esperados na execução. Assim como no trabalho de Chaves et al. (2013), o trabalho que aqui está sendo

[2] https://moodle.org/
[3] http://status.urionlinejudge.com.br/
[4] http://www.spoj.com/
[5] https://uva.onlinejudge.org/

proposto faz uma avaliação automática dos trabalhos submetidos, semelhante à avaliação feita pelos juízes online.

Diferente do que o trabalho de Chaves et. al. (2013) aborda, este trabalho não fez um mecanismo para integração de diferentes ambientes. Foi desenvolvido um *framework* que faz avaliação automática de trabalhos de programação através de casos de teste desenvolvidos pelo próprio professor.

## 3.2 Testador automático e método de avaliação de programas em Java

Segundo Nunes e Lisbôa (2004), para ter uma melhor contribuição no aprendizado de fundamentos de programação cada tópico visto em sala de aula deveria ter uma aula prática correspondente na qual o aluno pudesse testar aquilo que viu. O ideal seria que o professor tivesse um bom acompanhamento das atividades práticas e avaliasse os exercícios dados. Entretanto, a maioria das atividades práticas como listas de exercícios, podem não ser corrigidas pelo professor, e, por isso, muitos alunos costumam não fazê-las. Muitos professores não cobram a entrega de listas de exercício, desde que avaliar as resoluções das listas de cada aluno se torne uma tarefa bastante dispendiosa em termos de tempo.

Semelhante ao trabalho de Nunes e Lisbôa (2004), o trabalho aqui proposto é o desenvolvimento de um mecanismo para auxiliar o aprendizado de programação, avaliando automaticamente códigos em Java através de testes. A ferramenta de Nunes e Lisbôa (2004) realiza correção funcional, verificando a coerência de construtores e métodos e correção de estado, verificando a coerência de campos públicos e privados. São geradas classes de teste que têm a função de interceptar a criação e invocação de métodos da classe a ser testada, verificando a existência de erros. Durante os testes, são verificados três tipos de erro: erro de compilação, ocorrência de exceções e retorno de método incorreto. Após a execução de todos os testes nos problemas dos alunos, a ferramenta gera um relatório com as notas parciais e finais de todos os alunos.

A ferramenta de Nunes e Lisbôa (2004) não oferece submissão de trabalhos, diferente do que está sendo aqui proposto. Em Nunes e Lisbôa (2004) o professor é quem se encarrega de recolher todos os trabalhos dos alunos, adicioná-los em um pacote para realizar os testes e, depois de tudo isso, é que ele acionará a execução de testes em todos os programas, de todos os alunos. A ferramenta não trabalha a necessidade de *feedback* automático para os alunos. O trabalho aqui proposto é um *framework* onde os alunos possam submeter seus trabalhos e a correção seja acionada logo após a submissão.

## 3.3 Ferramenta ProgTest

O trabalho de Corte (2006) aborda a importância no ensino de testes de software em conjunto com fundamentos de programação. Assim como a maioria dos trabalhos semelhantes estudados, esse trabalho foi desenvolvido levando em consideração a preocupação que se tem com a insuficiência no aprendizado dos alunos em programação. Segundo a autora, a introdução de conceitos de teste de software em conjunto com fundamentos de programação tem sido uma iniciativa para amenizar os problemas com ensino de programação, pois pode auxiliar a desenvolver a compreensão e a análise dos estudantes, facilitando uma melhor compreensão do comportamento de seus programas. A ideia principal do trabalho de Corte (2006) é fornecer mecanismos para que o ensino de fundamentos de programação seja integrado com teste de software.

Assim como este trabalho, Corte (2006) aborda o desenvolvimento de um ambiente para submissão e avaliação automática de trabalhos práticos de programação, o ambiente ProgTest. Trata-se de um ambiente baseado na Web que avalia tanto os programas quanto os casos de teste que são fornecidos pelos alunos. O ambiente foi desenvolvido para aceitar programas em Java. Os casos de teste que são submetidos pelos alunos devem ser compatíveis com o *framework* de testes JUnit para que sejam aceitos pela ferramenta de teste que o ambiente ProgTest integrou, que é a ferramenta JaBUTi. A avaliação automática de cada programa se dá por meio de execuções trocadas de teste estrutural e teste funcional nos programas submetidos. Para cada trabalho pedido, o professor deixa no sistema, uma implementação de referência, o programa "oráculo" e um conjunto de casos de teste que são utilizados para testar essa implementação de referência. O sistema executa o programa referência do professor com os casos de testes fornecidos por ele e, para cada trabalho submetido, o sistema executa o programa do aluno com os casos de teste do aluno, o programa do aluno com os casos de teste do professor, e o programa do professor com os casos de teste do aluno. A ferramenta de teste JaBUTi realiza teste de cobertura em cada execução, fornecendo os índices de cobertura nos critérios de teste implementados, obtendo assim os resultados do teste estrutural dos trabalhos. Os resultados dos testes funcionais são obtidos na execução dos casos de teste do JUnit, comparando-se a saída obtida com a saída esperada.

Diferente do trabalho de Corte (2006), no trabalho aqui proposto não há necessidade de que o aluno envie casos de teste para as suas atividades de programação. Com esse

*framework*, o professor é quem desenvolverá os casos de teste que farão a avaliação automática das atividades que serão pedidas.

Quadro 1 – Comparativo sobre trabalhos relacionados

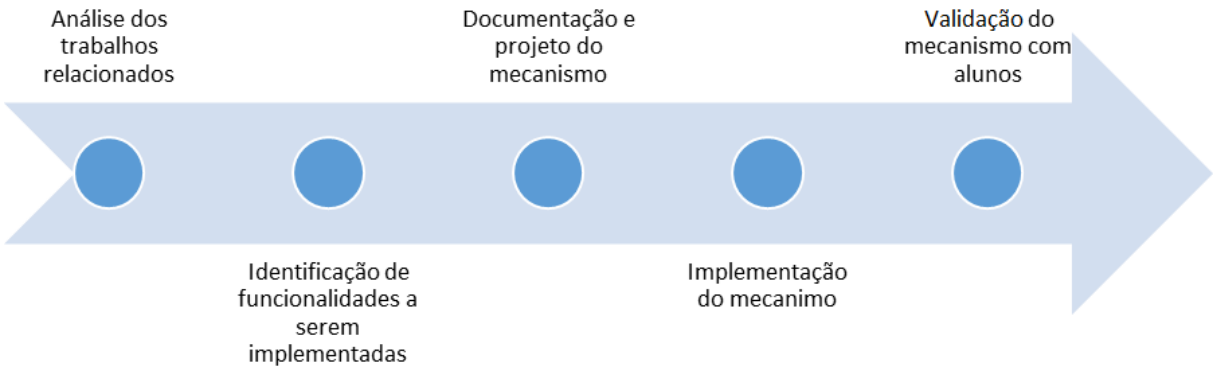
| FERRAMENTA MOJO | TESTADOR AUTOMÁTICO E MÉTODO DE AVALIAÇÃO DE PROGRAMAS EM JAVA | FERRAMENTA PROGTEST |
|---|---|---|
| Integra diferentes ambientes para realizar avaliação automática. | Usa testes de correção funcional e correção de estado para realizar avaliação automática. | Usa casos de teste e implementações de trabalhos em combinação para fazer avaliação automática. |
| Trabalha com submissão de trabalhos | Trabalhos precisam ser importados manualmente | Trabalha com submissão de trabalhos |
| Integrado a um ambiente web | Não possui ambiente web integrado | Integrado a um ambiente web |
| Trabalha com diferentes linguagens | Trabalha apenas com linguagem Java | Trabalha apenas com linguagem Java |

Fonte: Desenvolvido pelo autor

# 4 PROCEDIMENTOS METODOLÓGICOS

A Figura 02 ilustra quais atividades foram definidas para o desenvolvimento deste trabalho. Conforme analisado nos trabalhos relacionados, a preocupação com a dificuldade no aprendizado em programação leva ao desenvolvimento de ferramentas que possam tornar as correções de atividades práticas de programação mais frequentes nas disciplinas dos cursos de computação. O *framework* desenvolvido neste trabalho visa exatamente isso: dar apoio ao processo de avaliação de atividades de programação para que o professor possa incluir mais atividades práticas em suas disciplinas sem que tenha que empregar um grande tempo para corrigir todas elas. A ideia foi de permitir que os alunos possam submeter suas atividades de programação, como listas de exercício, trabalhos e até provas práticas. O *framework* fará uma avaliação automática dessas atividades através de testes de software que serão desenvolvidos e submetidos pelos professores usando o *framework*.

**Figura 2 - Procedimentos metodológicos**




Fonte: Elaborada pelo autor

## 4.1 Avaliação dos trabalhos relacionados

A primeira etapa visou uma análise dos trabalhos relacionados para que pudessem ser definidas as principais funcionalidades a serem implementadas no *framework*. Nessa análise, foi questionada a viabilidade deste trabalho pelo fato de já existirem mecanismos semelhantes ao que é aqui proposto. No entanto, a justificativa para a execução desse trabalho se dá também pelo estímulo em criar um mecanismo útil e que possa ser utilizado para

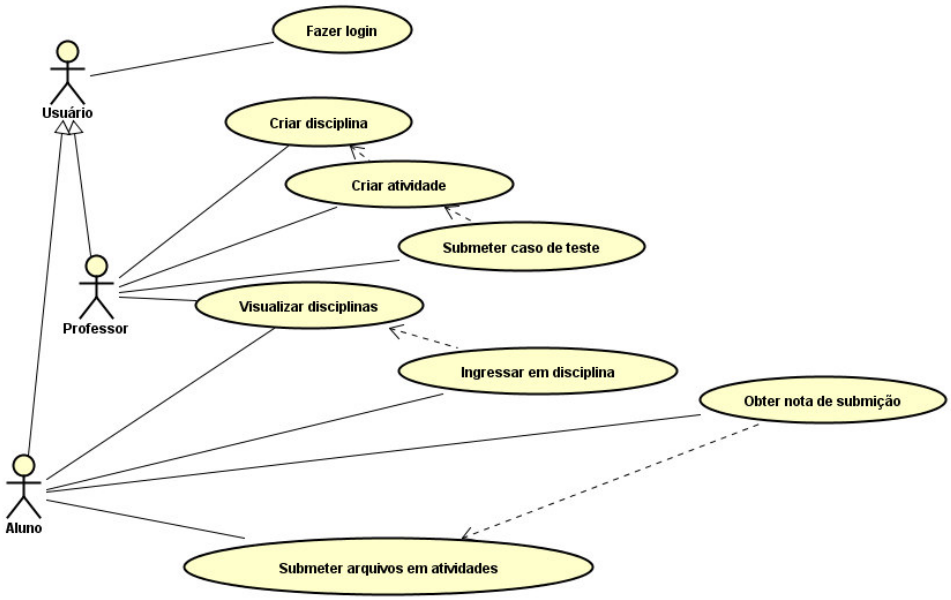
benefício dos professores e alunos de computação. Portanto, foi considerado que o desenvolvimento desse *framework* seria viável pois, além de ter o objetivo de ser útil para as disciplinas de programação, o estudo para o desenvolvimento desse *framework* tem como motivação proporcionar ao autor a oportunidade de pôr em prática os conhecimentos adquiridos na graduação para construção de algo útil para professores e alunos. Após a análise sobre os trabalhos relacionados, levando em consideração o tempo para conclusão e esforço empregados para o desenvolvimento, foram escolhidas as funcionalidades que seriam desenvolvidas. Foi definido que seria criado um *framework* que permita a submissão de um caso de teste elaborado pelo professor, para determinado trabalho que for pedido por ele, e que permita também que após submetido este caso de teste para o trabalho pedido, os alunos submetam as suas implementações para aquele trabalho. Após a submissão do trabalho do aluno, o *framework* irá executar o caso de teste do professor com a submissão do aluno e gerar um resultado que será automaticamente mostrado para o aluno. De início, o *framework* dá suporte apenas à linguagem Java.

## 4.2 Etapa de documentação e projeto

Estando definida qual seria a principal finalidade do *framework*, foi iniciada a etapa de documentação. Tendo o *framework* funcionando, a pretensão é de que se tenha uma forma de que a submissão e avaliação seja feita por cada usuário em sua própria máquina. Sendo assim, pretende-se que o framework seja acoplado em uma plataforma web e que futuramente seja disponibilizado online para que seja utilizado por professores e alunos. Tendo isso em vista, foi construído um documento de requisitos para a construção de uma plataforma web que executará esse *framework*. O Apêndice A mostra o documento que descreve o levantamento de requisitos.

Com os requisitos levantados, foram construídos também os diagramas de caso de uso (Figura 3) e de classes (Figura 4) para que servissem de base para a implementação desse ambiente. Além disso, foram feitos protótipos das telas que deveriam ser implementadas. O Apêndice B mostra um documento com os protótipos.

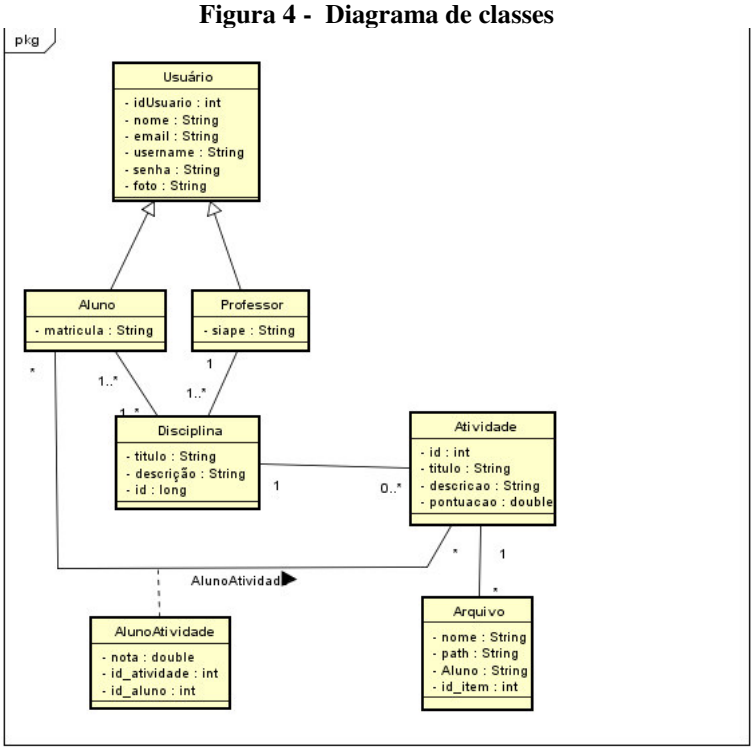
**Figura 3 - Diagrama de Caso de uso**




Fonte: Elaborada pelo autor

Os diagramas de caso de uso e de classes foram desenvolvidos visando a implementação de um ambiente web para execução do *framework*.

**Figura 4 - Diagrama de classes**




Fonte: Elaborada pelo autor

Apesar de o foco desse trabalho ter sido o desenvolvimento do *framework* de avaliação, foi visto como necessário a construção de uma interface com o usuário para que o mesmo pudesse ser avaliado. Sendo assim, a implementação da plataforma foi também iniciada juntamente com a implementação do *framework.*

## 4.3 Etapa de implementação

O texto a seguir descreve o processo de desenvolvimento do *framework* para avaliação automática, bem como o início do desenvolvimento de uma aplicação web para exemplificar a utilização do *framework* em um ambiente interativo com o usuário.

### 4.3.1 Primeira estrutura desenvolvida.

Tendo a documentação formulada, iniciou-se a fase de implementação do *framework*. A primeira ideia se deu em criar um projeto Java e colocar nos diretórios da máquina local dois arquivos com extensão Java, um que simularia a implementação do aluno e outro que seria o caso de teste simulando o arquivo do professor. O projeto Java teria o objetivo de, através dos caminhos dos arquivos, executar o caso de teste do professor com a implementação do aluno e gerar uma nota.

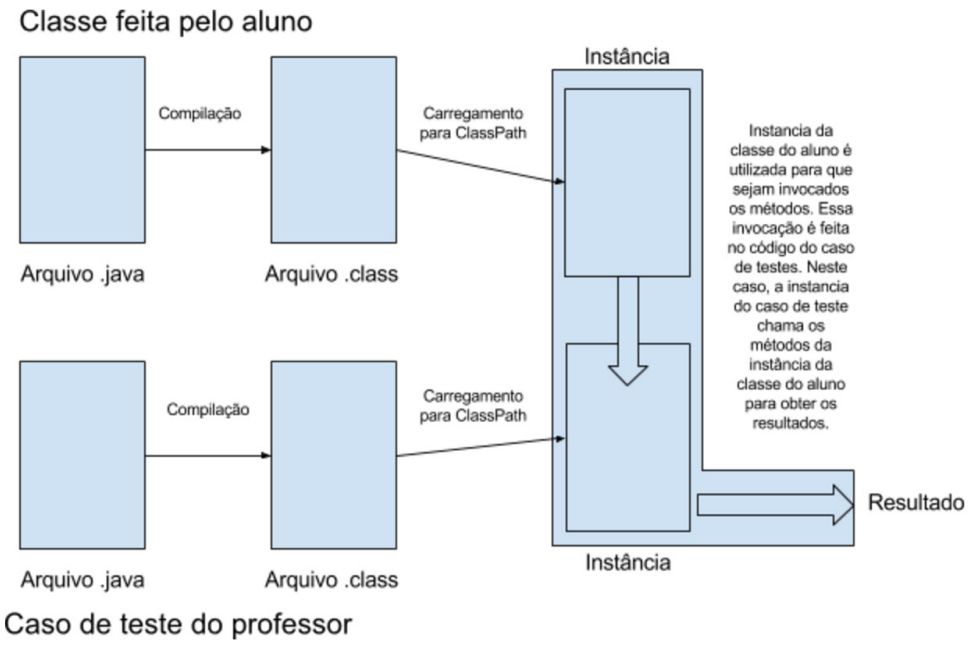
De início, a primeira dificuldade foi em encontrar uma forma de carregar esses arquivos Java vindo de localizações diferentes da localização do projeto corrente. Através da *IDE* Eclipse, é possível importar classes que estão no mesmo projeto e, até mesmo, em projetos diferentes usando a própria *IDE* para fazer isso de forma manual. Mas o que era preciso era uma forma dinâmica de importar esses arquivos, de modo que, após a submissão da implementação do aluno, essas classes fossem importadas para o projeto corrente. O projeto não tem conhecimento sobre quais classes seriam importadas, então não seria possível instanciar tais classes para realizar testes em seus métodos. Através de pesquisas sobre como fazer um carregamento dinâmico de classes foi visto que existe uma classe Java chamada *ClassLoader,* capaz de carregar classes Java dinamicamente para o *classpath* do projeto. O *ClassLoader* utiliza classes já compiladas, então surgiu a necessidade de compilar as classes java antes de usar o *classloader*.

Outra dificuldade encontrada foi em saber como desenvolver testes para classes que não existem no projeto. O professor deveria trabalhar com uma classe que só seria importada para seu projeto após a submissão do aluno. Após algumas pesquisas foi visto que a

linguagem Java possui um recurso chamado reflexão que permite criar chamadas em tempo de execução, sem a necessidade de conhecer as classes e os objetos envolvidos quando estiver escrevendo o código.

Tendo esses conhecimentos, um primeiro mecanismo foi criado. A estrutura funcionava da seguinte forma: Foi construída uma classe chamada ExternalClassBuilder que era responsável por pegar o caminho e o nome de um arquivo de extensão Java, compilar esse arquivo e carregar esse arquivo Java no *classpath*. O retorno do método era uma instância da classe recém compilada. A partir da instância da classe, era possível testar seus métodos através de métodos de reflexão do Java como o *getDeclaredMethod*. Dessa forma, um caso de teste foi desenvolvido para testar uma implementação em Java. Através de seus caminhos na máquina local, o projeto compilava e carregava as duas classes Java e, em seguida, o projeto executava uma instância do caso de teste que fazia uso de uma instância da classe de implementação para dar um resultado.

**Figura 5 - Primeiro fluxo de execução do framework**




Fonte: Elaborada pelo autor

Foi visto que essa estrutura era inviável, devido ao esforço que deveria ser empregado pelo professor para criar os testes. Desenvolvido dessa forma, o *framework* exigiria que o professor tivesse que trabalhar com técnicas de reflexão dentro dos testes. Porém, a criação desse *framework* visa facilitar o processo de correção de atividades, então a

criação de casos de teste pelo professor não deve se tornar algo ainda mais complexo. Além disso, o *framework* desenvolvido dessa forma permitiria que apenas um arquivo Java isolado fosse testado por vez.

### 4.3.2 Nova estrutura do framework

A proposta é de se ter uma forma mais flexível de fazer com que o *framework* funcione. O propósito foi que não fosse necessário que o professor tivesse conhecimentos de reflexão e que fosse possível também testar mais de um arquivo no mesmo projeto, permitindo que o aluno envie um arquivo de extensão JAR (Java ARchive) de um projeto inteiro, contendo possivelmente várias classes ao invés de apenas uma isolada. Da mesma forma, professores devem enviar arquivos JAR com seus testes, podendo conter vários testes em um único *JAR*.

Tendo em mente que essa seria a melhor forma de trabalhar o desenvolvimento do *framework*, uma ideia diferente foi posta em prática. Foram criados três projetos diferentes, um que simulava o *framework*, um que simulava o teste do professor, e um que simulava a implementação do aluno. Ficou definido que apenas o projeto do *framework* de avaliação deveria usar reflexão, pois o projeto do professor deveria se preocupar apenas em realizar os testes. Contudo, o professor deverá criar os casos de teste pensando em uma classe que ainda não existe em seu projeto, mas poderá fazer isso sem o uso de reflexão. De toda forma, o professor não teria como instanciar essa classe. O aluno deveria se preocupar em escrever os métodos com o nome exato que seria utilizado no caso de teste do professor para que o teste não desse erros desnecessários. Pensando nisso, como forma de facilitar esse processo para o professor e para o aluno, foi definido que o professor desenvolverá junto com os casos de teste, uma ou mais interfaces que deverão ser implementada pelos alunos em cada classe que for desenvolvida. A interface deve conter a assinatura dos métodos que o professor pedir para os alunos desenvolverem. Dessa forma, o professor pode criar seus casos de teste recebendo por parâmetro uma instância de *object* e, dentro do método, fazer um *cast* para a interface desenvolvida. E os alunos não precisariam se preocupar com o nome dos métodos, pois já iriam utilizar os métodos que estão nomeados na interface.

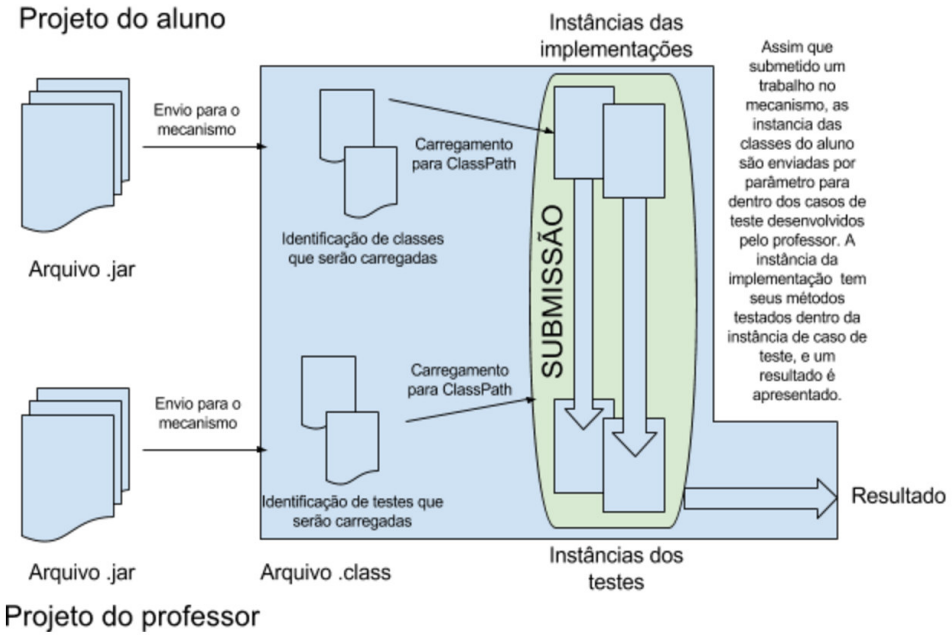
**Figura 6 - Exemplo de método de teste para ser criado**

```
public double testSoma(Object object) {
    Calculadora calc = (Calculadora)object;
    int retornoEsperado = 10;
    int retornoFeito = calc.soma(5, 5);
    double result;
    if(retornoEsperado==retornoFeito){
        result = 1;
    }else{
        result = 0;
    }
    return result;
}
```

Fonte: Elaborada pelo autor

A figura 6 mostra um exemplo simples de como deverá ser um método de teste. No exemplo, o teste é feito para avaliar um método de soma de uma classe que implementa uma calculadora. O método recebe um objeto do tipo *Object* e faz o *cast* para uma interface Calculadora, que deverá ser a mesma utilizada tanto pelo aluno como pelo professor. Ao final do teste, o retorno deverá ser a quantidade de pontos que aquele teste irá valer em nota, dependendo se a condição for satisfatória ou não.

**Figura 7 - Novo fluxo de execução do framework**



Fonte: Elaborada pelo autor

Toda a estrutura foi desenvolvida com o intuito de trazer mais facilidade para o professor e para o aluno, mas nem tudo pode ser resolvido com o uso de reflexão e *classloaders*. Sendo assim, alguns padrões de desenvolvimento tiveram que ser definidos para serem usados pelos alunos e professores. Embora grande tenha sido o esforço para tentar reduzir ao máximo a utilização de padrões de desenvolvimento, alguns se tornaram realmente necessários para a execução do *framework*. O emprego de tais padrões, no entanto, não depende de muito esforço. O pequeno esforço em desenvolver o código de acordo com os padrões do *framework* se torna muito pequeno em comparação com o grande esforço que seria para corrigir o trabalho de todos os alunos um a um. Os padrões de desenvolvimento são descritas a seguir.

### 4.3.3 Padrões de desenvolvimento de casos de teste para avaliação

- O professor deverá desenvolver uma interface com a assinatura dos métodos para cada classe que deverá ser escrita pelo aluno;
- O professor deverá criar uma classe de caso de teste em Java para cada classe de implementação que seus alunos deverão desenvolver. Se o professor quiser testar 5 classes do projeto do aluno, por exemplo, deverá desenvolver 5 casos de teste, um para cada classe do aluno;
- O professor deverá informar ao aluno, no enunciado do exercício, o nome exato que as classes de implementação que serão desenvolvidas deverão ter.
- O professor deverá informar aos alunos, no enunciado do exercício, o nome dos pacotes que deverão ser criados.
- No ato da entrada do projeto de casos de teste, o professor deverá informar o nome de cada caso de teste desenvolvido, seguido do nome da classe que será desenvolvida pelo aluno. O nome das classes devem ser especificados com o seguinte formato: *meupacote.minhaclasse* (Ver figura 12).
- Dentro dos casos de teste, o professor deverá receber, para cada método de teste que criar, uma instancia de *object*. Dentro do método criado, ele poderá criar uma instância com base na interface e fazer um *cast* de *object* para desenvolver o teste a partir daí;
- Cada método do caso de testes terá um valor em pontos para ser somado para a nota. Sendo assim cada método deverá ter um retorno do tipo *double*, e ao final da execução de cada método, o retorno deverá ser o valor que será dado em pontos para aquela questão. Sugere-se que o professor atribua o valor dentro de condições que validem a

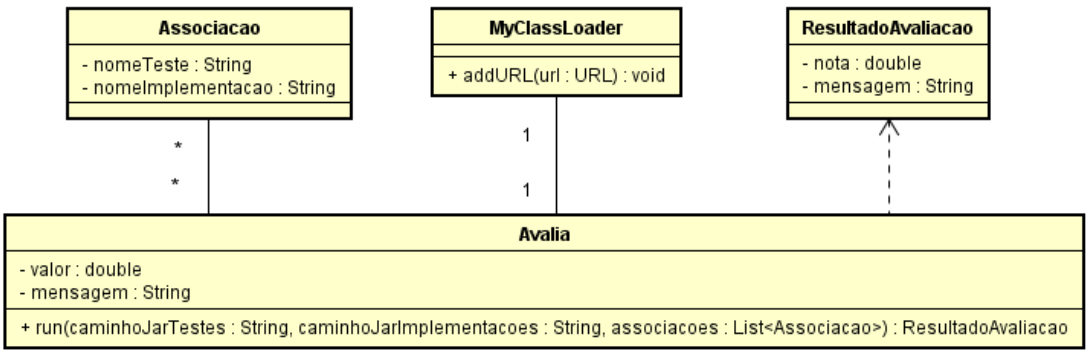
função pedida, retornando o valor total atribuído se a função executar conforme o esperado, e 0 caso não funcione corretamente;

### 4.3.4 Políticas de desenvolvimento de classes pelo aluno

- O aluno deverá utilizar as interfaces disponibilizadas pelo professor para serem implementadas em cada classe que criar;
- O aluno deverá colocar as classes que implementar dentro do pacote que for especificado pelo professor. Essa especificação ocorre no enunciado do trabalho;
- As interfaces disponibilizadas pelo professor também deverão ser colocadas dentro do pacote que o professor informar.

O Apêndice C mostra um documento para servir de instrução para professores sobre como os testes devem ser criados. Seguindo esses padrões, o *framework* estará pronto para executar da forma esperada. Os arquivos de extensão *JAR* do professor e do aluno serão devidamente carregados e executados, resultando em uma avaliação automática no momento em que o *framework* for posto em execução. O *framework* foi desenvolvido de forma desacoplada, visando ser facilmente acoplado em aplicações que desejam fazer uso de avaliação automática por casos de teste. A figura 8 mostra como foi organizada a estrutura:

**Figura 8 - Diagrama de classes do framework**




Fonte: Elaborada pelo autor

Para que funcione adequadamente e resulte a saída esperada, o *framework* precisa ser executado recebendo as entradas corretas. As entradas para a execução do *framework* são:

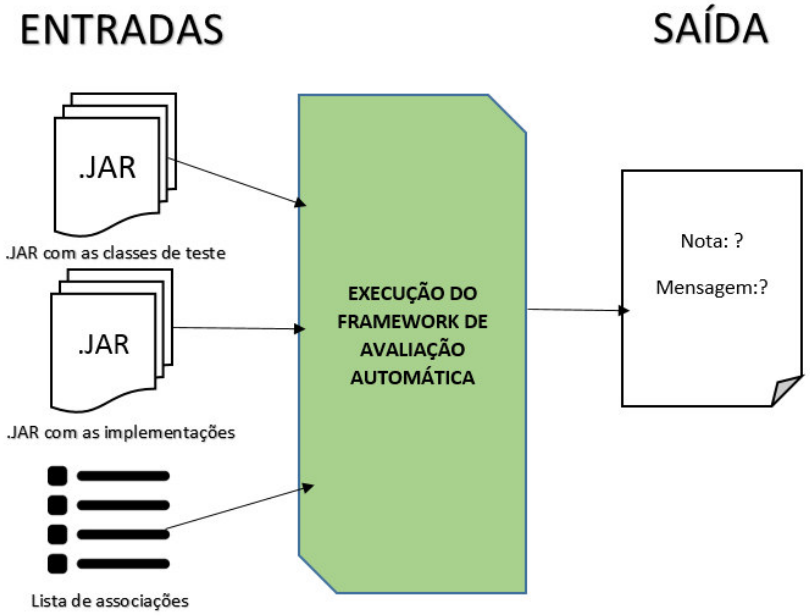
- Caminho do arquivo de extensão .JAR que contém os testes;
- Caminho do arquivo de extensão .JAR que contém as implementações a serem

testadas;

- Lista de associações descriminando qual classe de teste testará qual classe de implementação. Essa lista é preenchida pelo professor;

A figura 9 mostra uma representação gráfica das entradas e saídas da execução do *framework* desenvolvido.

**Figura 9 - Entradas e saídas do framework**




Fonte: Elaborada pelo autor

### 4.3.5 Ambiente web para execução do framework de avaliação desenvolvido

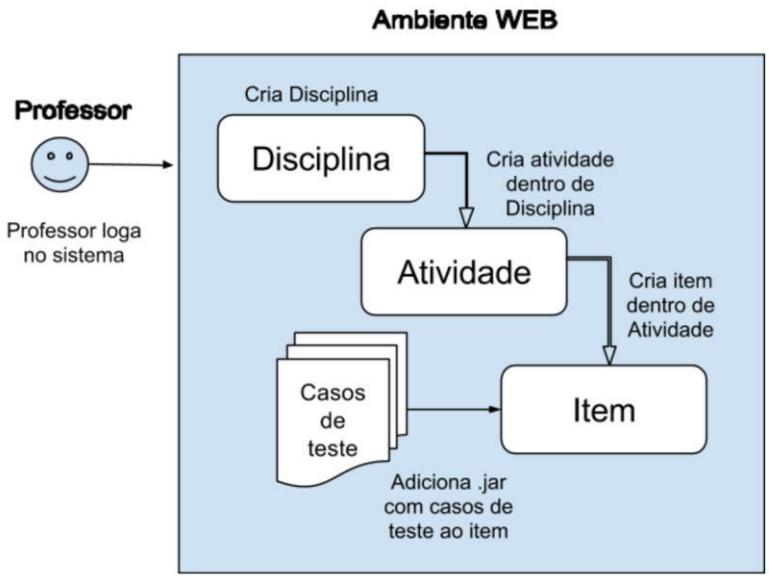
O foco principal do trabalho aqui descrito foi o desenvolvimento de um *framework* que recebendo, como entradas, os casos de teste do professor, uma lista de associações descriminando os testes associados as classes implementadas e as implementações do aluno, fosse capaz de gerar como saída uma nota para avaliar trabalhos práticos de programação. Porém, para execução desse *framework*, é interessante que se tenha uma forma de que cada aluno possa executar a submissão de seu projeto em sua própria máquina. Para que o *framework* fosse melhor avaliado, uma interface com usuário era necessária. Assim, considerando que a intenção é que futuramente esse *framework* fique disponível online, o desenvolvimento dessa interface foi feito dando início a uma aplicação web. O desenvolvimento de uma aplicação web demanda bastante tempo para a escrita de código e

também para a configuração do ambiente. Portanto, o foco dessa etapa foi o desenvolvimento da estrutura básica necessária para que o *framework* de avaliação fosse executado.

Uma parcela considerável de tempo foi tomada para que o projeto fosse configurado adequadamente. Foi configurado um projeto Maven[6] com Spring MVC[7], conectado ao banco de dados PostgresSql[8] utilizando *framework* Hibernate[9] e Apache Tomcat [10]como servidor local. Foram desenvolvidas classes modelo que são responsáveis pelos dados da aplicação, regras de negócio, lógica e funções; as classes de controladores, que responsáveis por fazer a mediação de entrada de dados, convertendo-os em comandos para o modelo. Foram desenvolvidas também as classes de serviço e repositório, responsáveis por fazer chamadas e persistência de dados no banco.

Como forma de tornar a execução desse *framework* mais agradável ao usuário, foram feitas melhorias no design das páginas utilizando técnicas de *css* juntamente com o *framework front-end bootstrap*[11]. O desenvolvimento teve como finalidade desenvolver tudo que fosse necessário para tornar possível a execução do *framework* pelo professor e pelo aluno. As figuras a seguir mostram os processos definidos como necessários para execução.

**Figura 10 - Fluxo de interação do professor com o ambiente web**

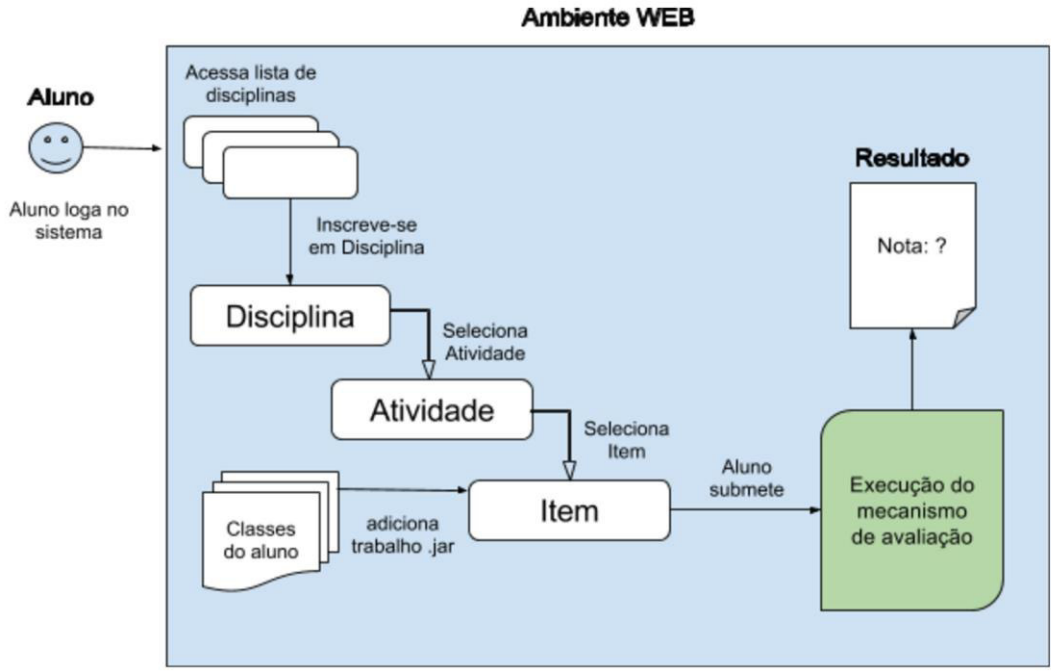



Fonte: Elaborada pelo autor

[6] https://maven.apache.org/
[7] https://spring.io/
[8] https://www.postgresql.org/
[9] http://hibernate.org/
[10] http://tomcat.apache.org/
[11] http://getbootstrap.com/

A figura 10 mostra o fluxo do professor no ambiente web para execução do *framework*. O professor poderá criar disciplinas, dentro de cada disciplina ele poderá cadastrar atividades e dentro das atividades ele pode cadastrar itens que servirão como *containers* para receber as submissões dos alunos e guardar os casos de teste do professor para aquela submissão. No momento em que o professor cria um item, ele submete os casos de teste na criação. Após esse processo, o aluno estará pronto para acessar a disciplina e enviar seu trabalho. A figura a seguir mostra o fluxo de interação do aluno.

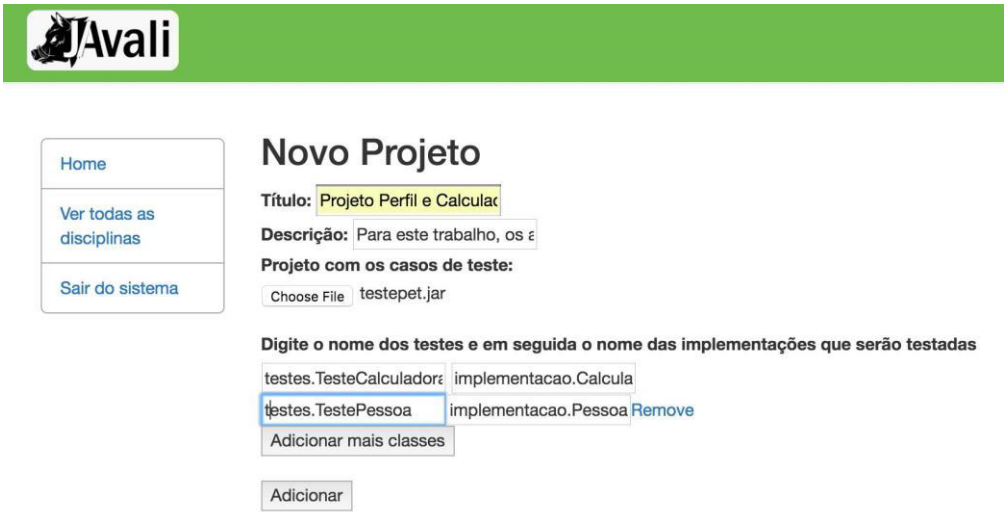
**Figura 11 - Fluxo de interação do aluno com o ambiente**




Fonte: Elaborada pelo autor

A figura 11 ilustra o fluxo de interação do aluno dentro da plataforma web para execução do *framework* de avaliação. O aluno poderá se inscrever nas disciplina cadastradas, e navegar dentro dos conteúdos para submeter seu projeto. Ao submeter o projeto, a plataforma web retornará uma página informando a nota, e caso tenha algum erro, informa em qual método o erro foi encontrado. Os fluxos esperados, descritos nas figuras 10 e 11, foram desenvolvidos para permitir que alunos e professores possam executar o *framework*, resultando numa versão inicial, que foi desenvolvida, de uma plataforma web para execução da avaliação automática de práticas de programação. A plataforma foi nomeada com o nome de Javali, numa tentativa criativa de mesclar as palavras “Java” e “Avalia”.

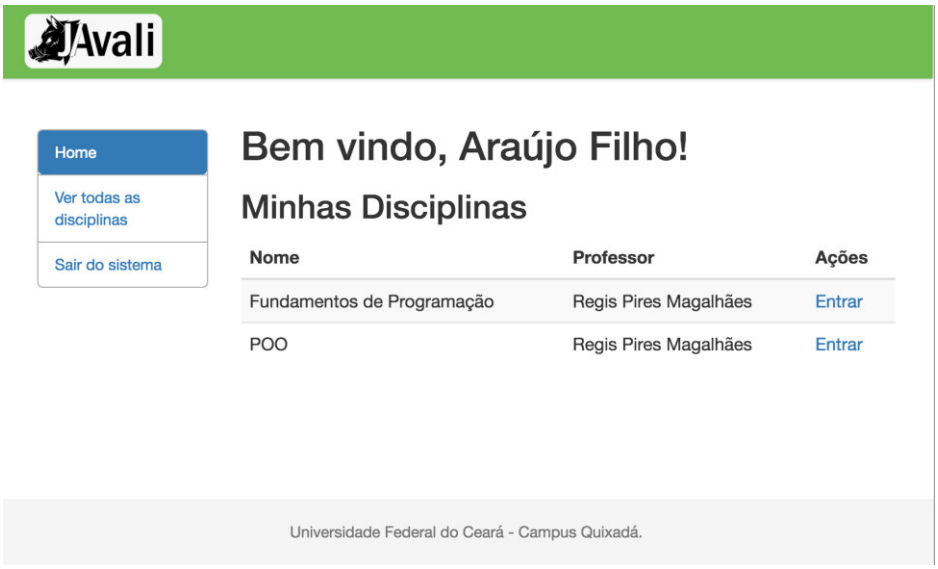
**Figura 12 - Página de criação de teste do professor**




Fonte: Elaborada pelo autor

A figura 12 mostra como deve ser cadastrado o teste pelo professor dentro de uma atividade. O professor deve adicionar um título, uma descrição que serve como enunciado da atividade, adicionar seu arquivo JAR com os casos de teste, e listar o nome de todas as classes de teste, seguidas dos nomes das respectivas classes que serão testadas. O professor deverá listar o nome das classes no seguinte formato: *meupacote.minhaclass.*

**Figura 13 - Pagina inicial do aluno**




Fonte: Elaborada pelo autor

A figura 13 mostra a página inicial de um usuário do tipo aluno. Nela é mostrada a listagem das disciplinas em que o aluno está inscrito. Navegando dentro da disciplina, o aluno encontrará atividades cadastradas onde poderá fazer a submissão de seu projeto. A figura 14 mostra a página de submissão de projeto para as atividades.

**Figura 14 - Página de submissão do aluno**

Fonte: Elaborada pelo autor

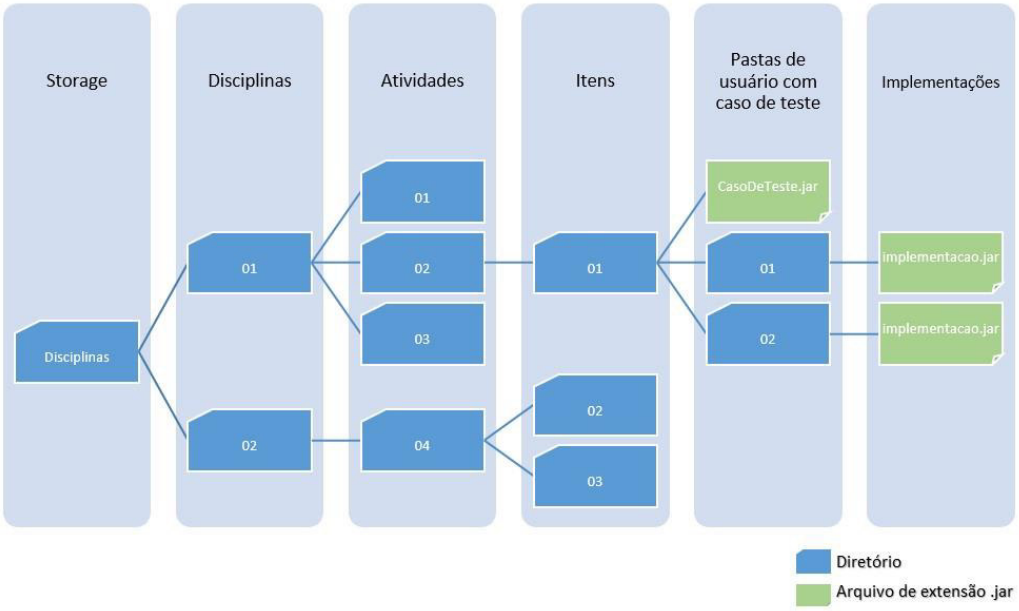
Após a submissão, uma página mostra ao aluno a pontuação que foi obtida com o projeto enviado, e uma mensagem informando o nome do métodos que acusaram erros, caso tenham. A figura 15 mostra a página de resultado.

**Figura 15 - Página de resultado após submissão**

Fonte: Elaborada pelo autor

Além da persistência de dados relacionados aos modelos no banco de dados, uma hierarquia de diretórios é criada para armazenar os casos de teste e implementações relativos a cada submissão. Cada disciplina, atividade e item tem um diretório criado no momento de sua criação. Esse diretório tem como nome o id do modelo criado. No momento da submissão de um trabalho pelo aluno, um diretório com o id do mesmo é criado dentro do diretório do item em que ele submeteu. Assim, a plataforma web pode fazer uso do caso de teste que já se encontra dentro do diretório do item com a implementação que estará dentro da pasta criada com o id do aluno. A figura 16 ilustra como é organizada a hierarquia de diretórios.

**Figura 16 – Hierarquia de diretórios**



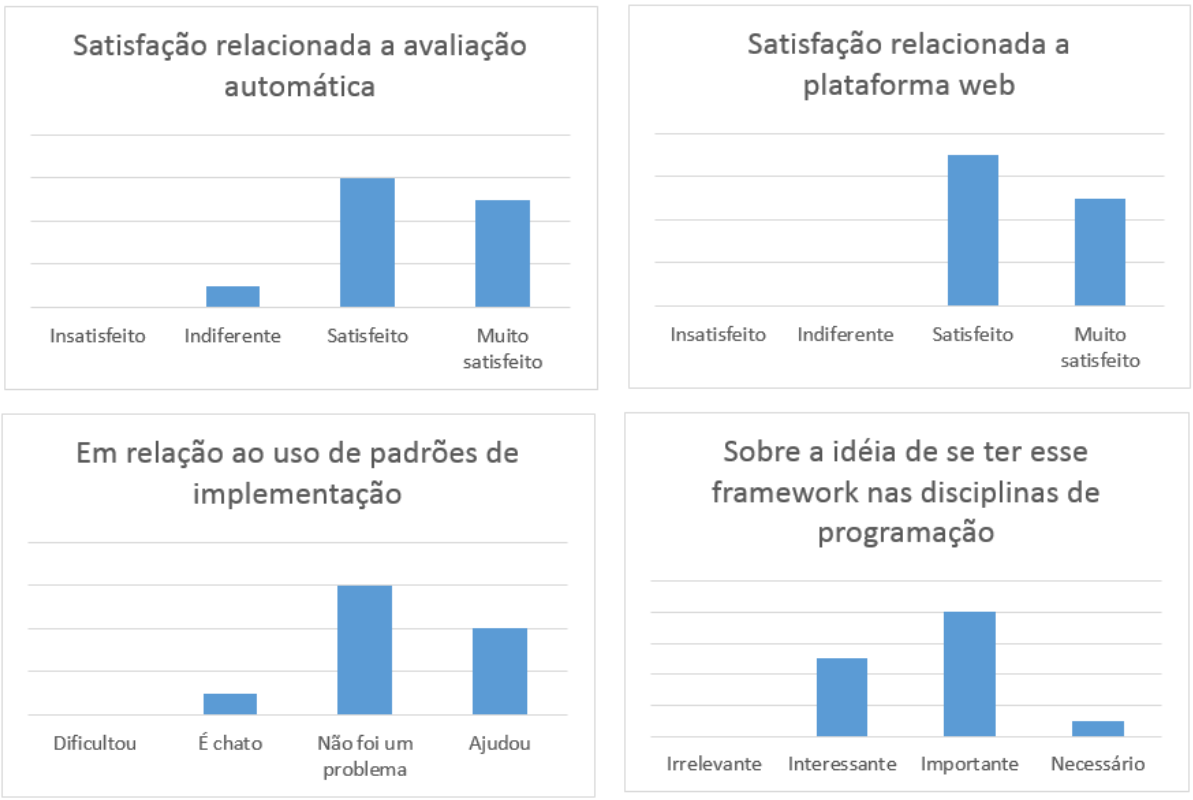

Fonte: Elaborada pelo autor

# 5 AVALIAÇÃO DO FRAMEWORK

Como forma de avaliar o *framework* desenvolvido, foi realizado uma método qualitativo de avaliação do *framework*. Para isso, a utilização do *framework* foi feita dentro da plataforma web que foi desenvolvida para que o usuário pudesse ter uma interação melhor com o *framework* de avaliação. A metodologia utilizada foi a de grupo de foco. Esse método dispensa a necessidade de um grande número de pessoas, pois analisa aspectos estatísticos de qualidade. Essa metodologia permite que os participantes discutam sobre o assunto que está sendo abordado e deem opiniões que possam ou não se basear na opinião dos outros participantes. A metodologia consiste em reunir grupos pequenos, onde tenha um intermediador para iniciar uma discussão para avaliar conceitos e identificar problemas.

A avaliação foi feita por 11 alunos do campus da UFC de Quixadá. Para início, foi simulada uma situação de utilização da plataforma web com o *framework* acoplado em sala de aula, onde foi solicitado aos 11 que fizessem uma atividade para ser avaliada pelo *framework* através da plataforma web. Foi pedido aos alunos que escrevessem duas classes, uma classe de calculadora, com quatro métodos relacionados as operações básicas da matemática, e uma classe de pessoa, onde era pedido que fosse recebido uma idade com o tipo inteiro e retornasse uma mensagem informando se a pessoa tinha idade permitida ou não para ingerir álcool. Foi orientado aos alunos quais os pacotes onde eles deveriam colocar suas classes. As interfaces para implementação de cada classe foram disponibilizadas através de um link. Após implementadas as classes, os alunos exportaram seus projetos em um arquivo de extensão *.JAR*. Foi pedido que eles acessassem a plataforma web através de um link, que ficou disponível em um endereço local na rede, para que eles realizassem a submissão da atividade em suas próprias máquinas. Previamente, já haviam sido adicionados à plataforma web os casos de teste responsáveis por fazer a avaliação das atividades. Ao submeter os arquivos no devido local, os alunos visualizaram a nota que o *framework* deu para suas atividades e onde ocorreram erros. Após todos terem realizado o processo de submissão da atividade, os alunos responderam individualmente a um pequeno questionário contendo algumas perguntas sobre a o processo avaliativo de seus trabalhos. A figura 17 mostra os resultados obtidos através desse questionário.

**Figura 17 - Questionário feito aos alunos**




Fonte: Elaborada pelo autor

Após o questionário, foram colocados pontos para serem discutidos pelo grupo como um todo, de forma a incentivar a discussão e aprimoramento de ideias baseado na opinião coletiva. A primeira questão levantada foi se os alunos sentiram falta de algo que achavam essencial na avaliação e não viram quando submeteram o trabalho. De início, a grande maioria teve dificuldades para discutir sobre isso, demonstrando não ter encontrado algo que realmente estivesse faltando. Porém, alguns relataram que seria interessante que o *framework* informasse não só os métodos que deram erro, mas também o motivo pelo qual o método estaria errado. Os alunos se demonstraram satisfeitos em poder executar o *framework* de avaliação através da plataforma web. Outra questão colocada para discussão foi pontos de melhoria para serem colocados como trabalhos futuros. Nesse momento, os alunos tiveram mais facilidade em discutir. Alguns alunos falaram que seria interessante que a plataforma web estivesse disponível online para que o *framework* pudesse ser utilizado através dela, e que o histórico das notas ficasse disponível para eles. Outros alunos sugeriram que, também relacionado a plataforma web, o caminho percorrido até chegar a página de submissão fosse menor, o que foi rebatido por outro grupo de alunos que disse que esse fluxo era necessário para a organização da disciplina. Alguns alunos sugeriram que a ferramenta pudesse também

ser utilizada para suportar código de outras linguagens, e que seria muito interessante se fosse usada com uma metodologia de *gameficação* durante o decorrer da disciplina.

# 6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O trabalho aqui descrito, após o estudo de aplicações existentes relacionadas ao incentivo da intensificação de práticas de programação, bem como correção das mesmas, se propôs a obter conhecimentos necessários e desenvolver um *framework* com esse objetivo, permitindo submissão de implementações e testes em Java, e através de reflexão e *classloaders* fazer a execução combinada deles para gerar um resultado. Tudo isso como uma forma de trazer mais uma opção para correção rápida de práticas de programação, a fim de que possibilite um *feedback* rápido para os alunos, tornando possível a cobrança por mais atividades práticas de programação com correção.

Para atingir esses objetivos, o trabalho se dividiu em algumas etapas, que constaram na identificação e análise das ferramentas semelhantes já existentes, eleger com base nessas ferramentas as melhores funcionalidades para serem implementadas de acordo com o nosso cenário, projetar o desenvolvimento de um ambiente web para execução do *framework*, desenvolver a implementação do *framework* e de boa parte do ambiente web para execução do mesmo, e uma avaliação do *framework* por alunos de computação do campus de Quixadá;

O trabalho aqui desenvolvido poderá servir de base para o desenvolvimento de outros. Como trabalhos futuros propõe-se:

- Finalização da plataforma web e disponibilização como um serviço em um endereço web, guardando todos os dados devidamente. A plataforma ainda não está armazenando as notas do usuário no banco ou disponibilizando uma lista com as notas do aluno para o professor. Também não está ainda permitindo que as informações sejam editadas no banco de dados.
- Um estudo para reduzir os padrões de implementação que foram definidos. Como exemplo, remover a necessidade de o caso de teste precisar receber um *object* como parâmetro ao invés de uma instância da interface que o professor já possui. Isso ocorre porque quando o *framework* envia a instância da implementação do aluno para dentro da instância de teste, ele chama o método que testa por reflexão usando a função *method.invoke().* Como o método de teste recebe a instância da implementação do aluno como parâmetro, o framework precisa mandar por parâmetro o mesmo tipo que o método de teste recebe. Por exemplo, *method.invoke(instanciaDaImplementacao.class).* Fazendo dessa forma, a função não manda exatamente o tipo da instância da implementação do aluno para dentro da

instância de teste. Ela manda um tipo *"Class",* gerando assim conflito com o tipo de parâmetro que seria recebido no teste.

- Tentar remover a necessidade de o professor precisar preencher uma lista de associações entre as classes que testam e as implementações que devem ser testadas. O *framework* desenvolvido não tem como identificar qual teste testará qual implementação, então por isso é necessário utilizar esse padrão. Porém, pode ser pensado em utilizar algo como anotações a serem colocadas nas classes do aluno discriminando o nome do teste que deveria ser usado, ou mesmo utilizar o JUnit para usar uma notação que identifique todos os testes e os execute.
- Um estudo para tentar aprimorar o resultado de falha mostrado ao aluno. O *framework* está atualmente mostrando apenas o nome dos métodos que estão com defeito. Porém, seria muito interessante que detalhes sobre essa falha fossem mostradas ao aluno. Por exemplo, quais foram as condições utilizadas para testar o método e quais resultados foram obtidos na utilização do método da classe do aluno.
- Um estudo para incorporar *gameficação* na utilização desse framework dentro das disciplinas, utilizando a plataforma web Javali.
- Um estudo para tornar possível que a plataforma web javali utilize outros mecanismos e passe a aceitar submissão e avaliação de práticas de programação de linguagens diferentes de Java.

Os código estão abertos no GitHub para quem desejar fazer uso deles.

Link para o código do framework:
https://github.com/albenorfilho/JavaliFrameworkCore
Link para o código da plataforma web Javali:
https://github.com/albenorfilho/javaliproject

## REFERÊNCIAS

CHAVES, et al. **Uma Ferramenta para Auxiliar na Elaboração, Submissão, e Correção de Atividades Práticas em Disciplinas de Programação**. Congresso da Sociedade Brasileira de Computação: DesafIE! – II Workshop de Desafios da Computação Aplicada a Educação, Mossoró – RN, 2013a.

CHAVES, et al. **MOJO: Uma Ferramenta para Auxiliar o Professor em Disciplinas de Programação**. ESUD 2013 – X Congresso Brasileiro de Ensino Superior a Distância, Belém – PA, Junho, 2013b.

CORTE, Camila Kozlowski Della. **Ensino integrado de fundamentos de programação e teste de software.** São Carlos – SP, Maio, 2006.

DOMINGUES, André Luís dos Santos. **Avaliação de Critérios e Ferramentas de Teste para Programas OO**. São Carlos – SP, Setembro, 2002.

FORBELLONE, André Luiz Villar; EBERSPACHER, Henri Frederico. **Lógica para Programação**. 3ª Edição. São Paulo – SP: Editora Prentice Hall, 2005.

JAQUES, Rafael. **O que é um Framework? Para que serve?** 2016. Disponível em: <http://www.phpit.com.br/artigos/o-que-e-um-framework.phpit>. Acesso em: 25 jun. 2016.

MALDONADO et al. **Introdução ao Teste de Software.** São Carlos – SP: Notas didáticas do ICMC, Abril, 2004.

MOREIRA, Meireille P.; FAVEIRO, Eloi L. . **Um ambiente para Ensino de Programação com Feedback Automático de Exercícios**. XXIV Congresso da Sociedade Brasileira de Computação, Bento Gonsalves – RS, Workshop em Educação em Computação, 2009.

NUNES, Ingrid Oliveira de; LISBÔA, Maria Lúcia Blanck**. Testador Automático e Método de Avaliação de Programas em Java**. Congresso Sul Brasileiro de Computação, Porto Alegre – RS, Outubro, 2004.

PRESSMAN, Roger S. **Engenharia de Software: Uma abordagem Profissional.** 7ª Edição. Porto Alegre – RS: ABDR, 2011.

ROCHA, André Dantas. **Uma ferramenta baseada em aspectos para apoio ao teste funcional de programas Java**. São Carlos – SP, janeiro, 2005.

SILVA, Alexandre Menezes; JORGE, Eduardo Manuel de Freitas. **JUnit – Teste de Unidade**. Disponível em < http://junit.wikidot.com/>. Acesso em 02 Abr. 2014

# APÊNDICE A - DOCUMENTO DE REQUISITOS PARA A PLATAFORMA WEB DE AVALIAÇÃO

## 1. Introdução

### 1.1. Propósito do documento

O objetivo desse documento é descrever os requisitos da plataforma de avaliação Javali, bem como servir de guia para o desenvolvimento.

### 1.2. Escopo do Produto

A plataforma web tem como objetivo auxiliar no aprendizado de programação, realizando avaliação automática de atividades de programação utilizando casos de teste do JUnit.

### 1.3. Definições e Abreviações

Abreviações:

RF: requisito funcional;
RNF: requisito não funcional.

### 1.4. Visão Geral do documento

Este documento apresenta uma descrição geral do sistema e, logo em seguida, descreve suas funcionalidades especificando as entradas e saídas para todos os requisitos funcionais. Faz também uma descrição sucinta dos requisitos não funcionais contidos neste sistema.

## 2. Descrição Geral

A plataforma web JAVALI avaliará trabalhos de programação submetidos pelos alunos cadastrados no sistema. Qualquer pessoa terá permissão para criar um usuário. Usuários poderão editar, remover e visualizar suas informações e deverão se cadastrar em uma turma cadastrada no sistema. Apenas o administrador poderá cadastrar um professor. O professor tem o privilégio de criar, remover, editar e visualizar turmas e atividades no sistema.

### 2.1. Perspectiva do Produto

A plataforma web estará disponível online para ser utilizada por professores e alunos das disciplinas de programação.

### 2.2. Funções do Produto

Gerenciamento de Usuários: cadastrar, visualizar, modificar e excluir usuário do sistema.
Gerenciamento de professores: cadastrar, visualizar, modificar, e excluir professores do sistema.
Gerenciamento de turmas: cadastrar, visualizar, modificar e excluir turmas do sistema.
Gerenciamento de atividades: cadastrar, visualizar, modificar e excluir atividades no sistema.
Gerenciamento de trabalhos: cadastrar, visualizar, modificar e excluir trabalhos no sistema.
Submissão de trabalhos: Enviar trabalhos para avaliação automática.
Avaliação automática: Avaliar trabalhos com base nos critérios do professor e atribuir notas.
Geração de relatório: Gerar visualização das notas dos alunos de cada turma, podendo ser exportado para PDF, XLS ou outros formatos.

### 2.3. Restrições Gerais

O sistema não permitirá que pessoas não cadastradas acessem as informações de turmas.

## 3. Requisitos

### 3.1. Requisitos Funcionais

RF. 1: Cadastro de Usuário.

**Descrição:** Qualquer pessoa poderá se cadastrar no sistema, e entrar nos grupos de disciplina disponíveis.
**Entrada:** Nome de usuário, E-mail e senha.
**Processo:** O aluno acessa o sistema, realiza o cadastro que será salvo no banco de dados e seleciona uma turma para entrar.
**Saída:** Mensagem de confirmação bem sucedido do cadastro caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF. 2: Modificação de Cadastro de Usuário

**Descrição:** O usuário entra com o campo onde ele deseja modificar e o modifica.
**Entrada:** Campo desejado e o novo dado.
**Processo:** Atualização do banco de dados.
**Saída:** Mensagem de confirmação bem sucedido da modificação do cadastro caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 3: Exclusão do Cadastro de Usuário

**Descrição**: Os usuários poderão excluir sua conta do sistema.
**Entrada:** Usuário logado no sistema.
**Processo:** O usuário seleciona uma opção para excluir sua conta.
**Saída:** Mensagem de confirmação da exclusão do cadastro caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 4: Visualização de dados do usuário

**Descrição:** Os usuários poderão visualizar as informações do seu perfil.
**Entrada:** Usuário logado no sistema.
**Processo:** O usuário seleciona uma opção para visualizar suas informações.
**Saída:** Tela com as informações do usuário.

RF 5: Cadastro de perfil de professor

**Descrição:** O administrador do sistema cadastrará professores que se interessarem em gerenciar turmas de disciplinas dentro do sistema.
**Entrada:** Nome de usuário, Email e senha.
**Processo:** O administrador realiza o cadastro do professor, que terá a permissão para criar turmas dentro do sistema.
**Saída:** Mensagem de confirmação bem sucedido o cadastro caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 6: Editar de perfil de professor

**Descrição:** O professor poderá alterar suas informações
**Entrada:** Campo que deseja alterar, e novo dado.
**Processo:** Atualização do banco de dados.

**Saída:** Mensagem de confirmação bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 7: Exclusão de perfil de professor

**Descrição:** O professor poderá excluir seu perfil do sistema.
**Entrada:** Professor logado no sistema.
**Processo:** O professor selecionará uma opção de "Excluir conta", e será removida do banco de dados.
**Saída:** Mensagem de confirmação bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 8: Visualizar dados do professor

**Descrição:** O professor poderá visualizar seus dados
**Entrada:** Usuário cadastrado no sistema.
**Processo:** O professor seleciona uma opção para visualizar suas informações.
**Saída:** Tela com as informações do professor.

RF 9: Cadastrar turmas

**Descrição:** O professor poderá criar turmas dentro do sistema.
**Entrada:** Nome do grupo, descrição;
**Processo:** O professor preenche dados sobre a nova turma, que são salvas no banco de dados.
**Saída:** Mensagem de confirmação bem sucedido o cadastro caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 10: Editar turmas

**Descrição:** O professor poderá editar informações sobre as turmas que cadastrou.
**Entrada:** Campo que deseja alterar, dados novos;
**Processo:** Alteração do banco de dados.
**Saída:** Mensagem de confirmação bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 11: Visualizar turmas

**Descrição:** Qualquer usuário do sistema poderá ver as turmas cadastradas
**Entrada:** Usuário logado no sistema;
**Processo:** As turmas estarão disponíveis na página inicial do sistema.
**Saída:** Informações sobre as turmas.

RF 12: Excluir turmas

**Descrição:** O professor poderá excluir as turmas que cadastrou.
**Entrada:** Professor logado no sistema
**Processo:** O professor seleciona a opção remover turma e o sistema remove do banco de dados.
**Saída:** Mensagem de confirmação bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro

RF 13: Cadastrar atividade

**Descrição:** O professor poderá cadastrar atividades a serem cobradas em suas turmas. A atividade ficará guardada em um banco de atividades do qual o professor poderá fazer uso outras vezes.
**Entrada:** Título da atividade, descrição da atividade, caso de teste da atividade;
**Processo:** Cadastro da atividade no banco de dados.
**Saída:** Mensagem de bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 14: Editar atividade

**Descrição:** O professor poderá editar informações sobre atividades cadastradas.
**Entrada:** Campo que deseja alterar, dados novos;
**Processo:** Alterar banco de dados.
**Saída:** Mensagem de confirmação bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro

RF 15: Remover atividade

**Descrição:** O professor poderá remover atividades.
**Entrada:** Professor logado no sistema;
**Processo:** Opção remover atividade, e exclusão da atividade do banco de dados.

**Saída:** Mensagem de confirmação bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro

RF 16: Visualizar atividade

**Descrição:** O professor poderá ver as atividades cadastradas para ele.
**Entrada:** Professor logado no sistema;
**Processo:** Selecionar a atividade desejada e mostrar tela contendo informações sobre a atividade.
**Saída:** Tela contendo informações sobre a atividade.

RF 17: Cadastrar trabalho

**Descrição:** O professor poderá cadastrar trabalhos a serem pedidos para os alunos nas turmas que tiver cadastrado. O professor poderá adicionar para o trabalho, alguma atividade que já esteja cadastrada ou criar uma nova.
**Entrada:** Título do trabalho, descrição do trabalho, período de submissão, atividade;
**Processo:** Cadastro de trabalho adicionado a turma.
**Saída:** Mensagem de bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 18: Editar trabalho

**Descrição:** O professor editar trabalhos que cadastrou com novas informações.
**Entrada:** Campos que deseja alterar, novos dados
**Processo:** Alterar banco de dados
**Saída:** Mensagem de bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 19: Remover trabalho

**Descrição:** O professor poderá remover os trabalhos que cadastrou.
**Entrada:** Professor logado no sistema
**Processo:** Opção de remover trabalho, e exclusão do trabalho no banco.
**Saída:** Mensagem de bem sucedido caso tenha sido efetuado com sucesso, senão, mensagem de erro.

RF 20: Visualizar trabalho

**Descrição:** Qualquer usuário dentro da turma poderá ver os trabalhos cadastradas para ela.
**Entrada:** Usuário logado no sistema;
**Processo:** Selecionar o trabalho desejado e mostrar tela contendo informações sobre o trabalho.
**Saída:** Tela contendo informações sobre o trabalho.

RF 21: Submeter trabalho

**Descrição:** O aluno poderá submeter seu trabalho dentro das atividades abertas na turma em que está cadastrado.
**Entrada:** Aluno logado no sistema, trabalho de programação;
**Processo:** O aluno envia seu trabalho para submissão.
**Saída:** Tela informando a nota atribuída a atividade do aluno.

RF 22: Avaliação automática de trabalhos.

**Descrição:** O sistema avaliará automaticamente os trabalhos dos alunos no momento da submissão.
**Entrada:** Trabalho do aluno, casos de teste do professor;
**Processo:** Sistema aplica casos de teste do professor na atividade do aluno, contabiliza os pontos levando em consideração os pesos atribuídos a cada teste, e gera uma nota.
**Saída:** Nota do trabalho do aluno.

RF 23: Gerar relatório com notas dos alunos

**Descrição:** O professor poderá gerar um relatório com as notas da turma para cada atividade adicionada.
**Entrada:** Professor logado no sistema;
**Processo:** Opção de gerar relatório de notas na atividade selecionada.
**Saída:** Pdf contendo relação com as notas dos alunos da turma.

## 3.2. Requisitos Não Funcionais

RF01: Linguagem e plataforma de programação

Aplicação Web desenvolvida em Java que será executada em servidor de aplicação JEE.

RF02: Layout

A plataforma web utilizará o *framework* front-end Bootstrap.

RNF03: Carregamento rápido

Usar AJAX para tornar as páginas mais interativas.

RNF04: Segurança

A plataforma web deverá guardar apenas o hash da senha para segurança do usuário.

RNF05: Acessibilidade

A plataforma web deverá funcionar, pelo menos, nos navegadores mais utilizados (Firefox, Chrome, Internet Explorer, Safari, Opera).

# APÊNDICE B - PROTÓTIPOS DA PLATAFORMA WEB

Imagem 1 – Página inicial do ambiente

Imagem 2– Pagina inicial de aluno

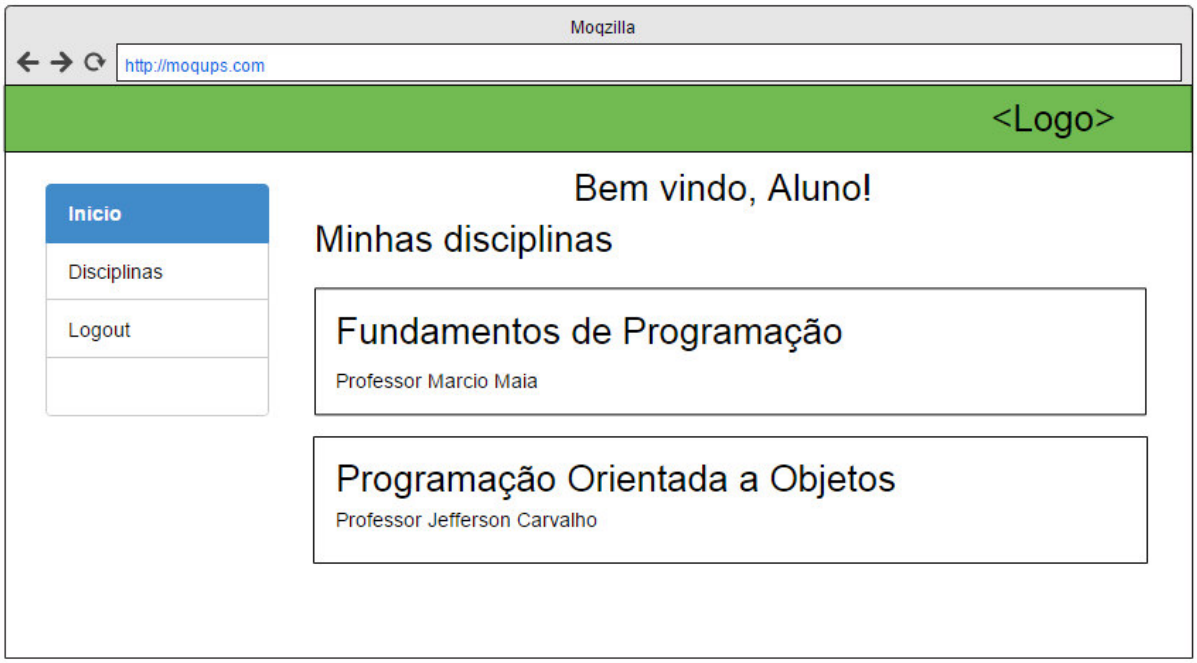


Imagem 3 – Pagina inicial da disciplina – Lado do aluno

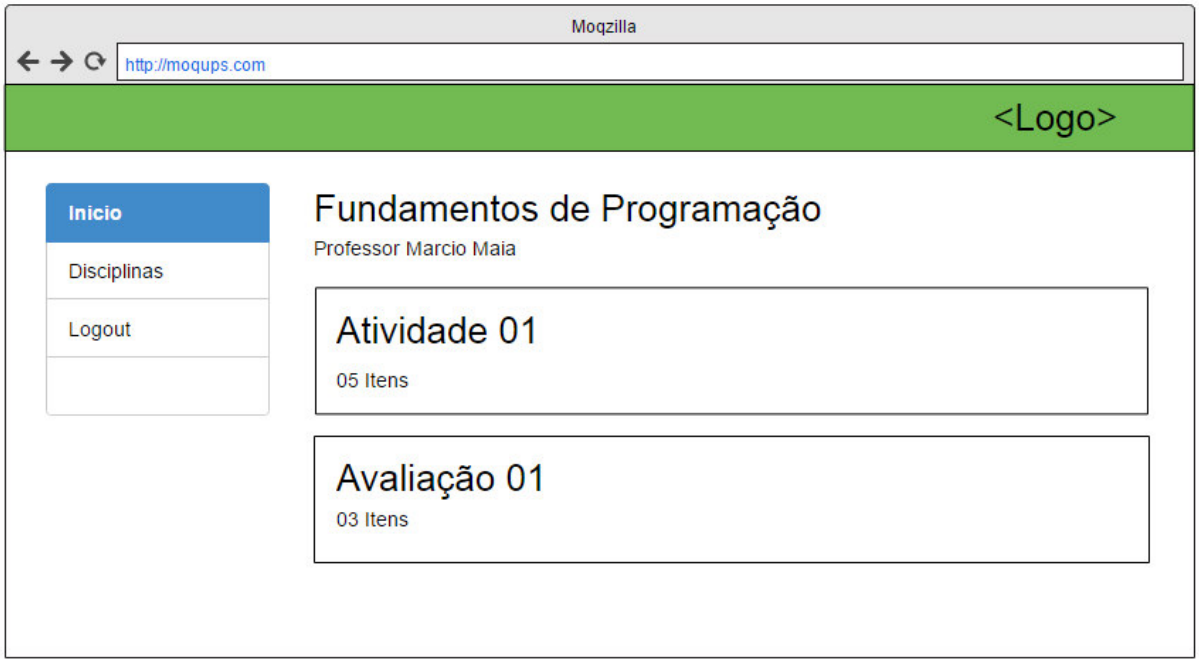



Imagem 4 – Pagina inicial de atividade

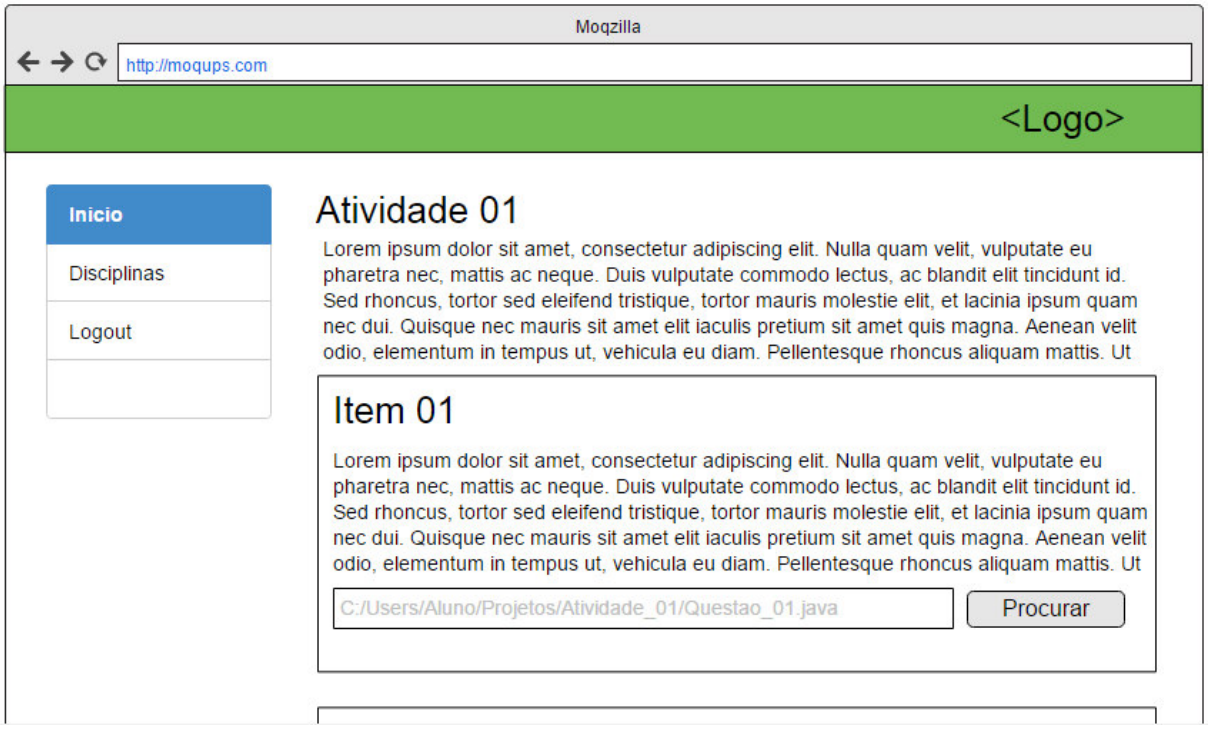


Imagem 5– Pagina para listar disciplinas

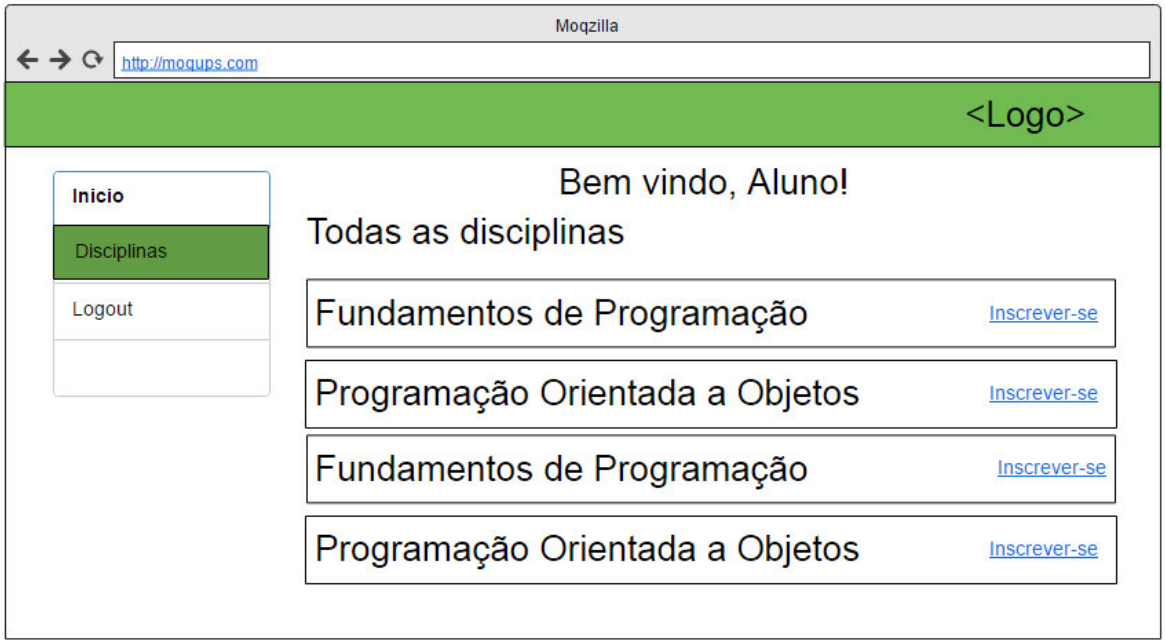



Imagem 6 – Pagina inicial do professor

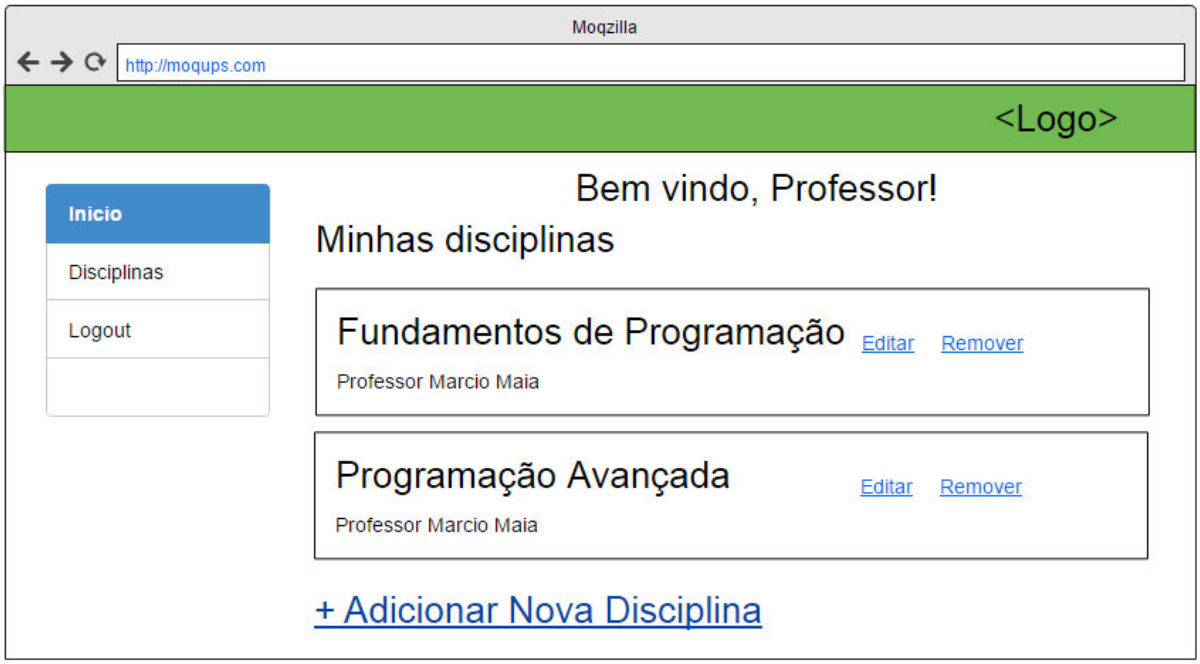


Imagem 7 – Pagina inicial de disciplina – Lado do professor

Imagem 8 – Pagina inicial de atividade – Lado do professor

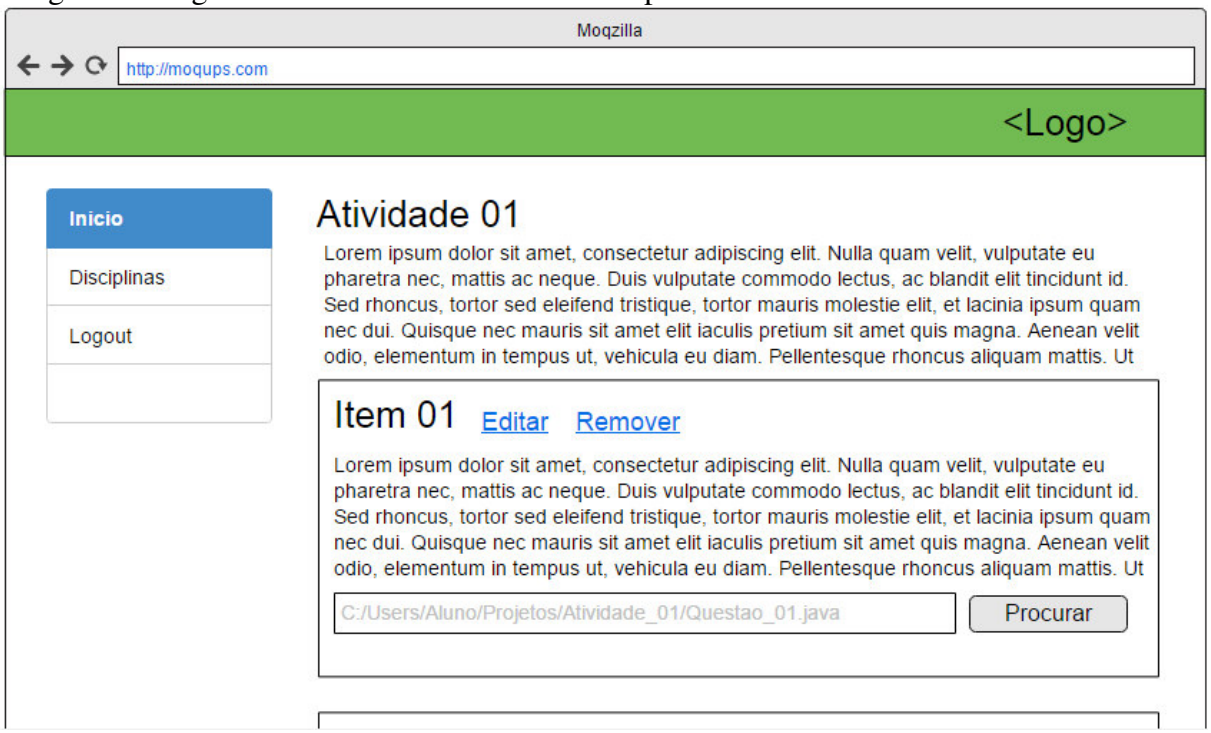


Imagem 9 – Pagina para cadastrar nova disciplina

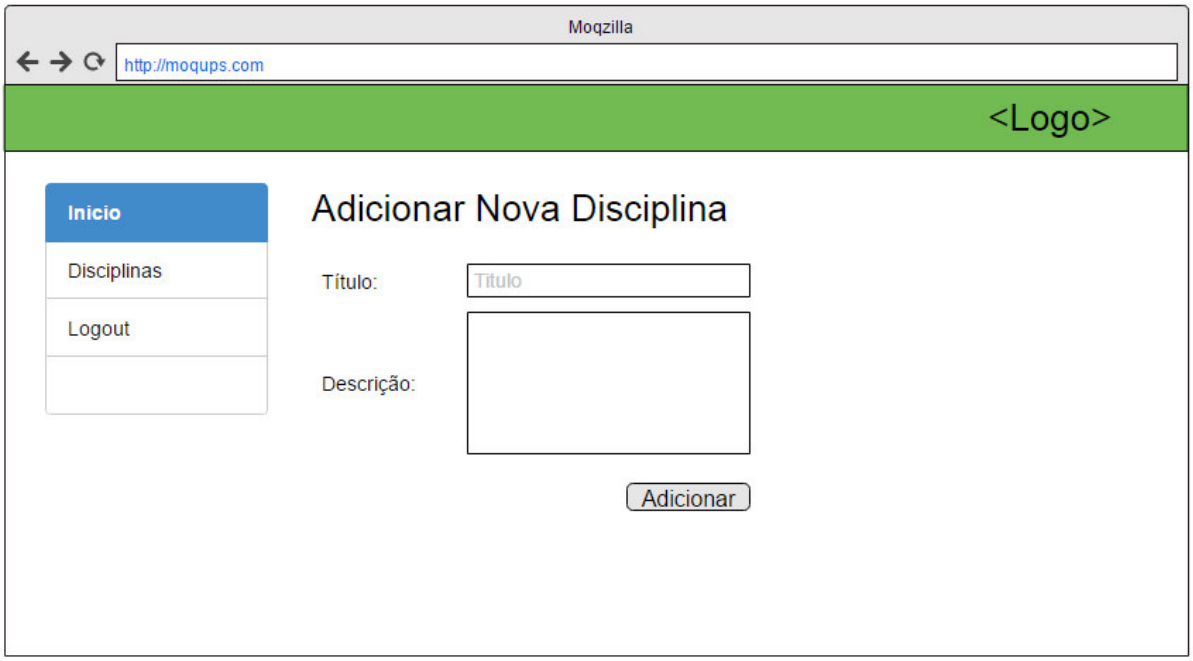



Imagem 10 – Pagina inicial para administrador

# APÊNDICE C - PADRÕES DE DESENVOLVIMENTO
# GUIA PARA O PROFESSOR

Algumas padrões devem ser seguidos para que o *framework* de avaliação funcione corretamente. A seguir serão descritas as políticas e alguns exemplos.

1) Uma interface deverá ser disponibilizada pelo professor. O caso de teste usará essa interface para dar *cast* numa instância de *object*, já que a classe a ser testada não está ainda no *classpath*, impossibilitando de ser instanciada em tempo de desenvolvimento (Veja um exemplo disto na figura do item 6, na linha 8).

2) Para cada classe que que for pedida ao aluno para implementar, deverá ser criado um caso de teste para testa-la. Não será possível testar mais de uma classe dentro do mesmo caso de teste. Exemplo: Se for pedido ao aluno que crie uma classe que implemente uma calculadora e outra que implemente um conversor de moedas, deverão ser escritos duas classes de caso de teste: TesteCalculadora e TesteConversor, por exemplo.

3) É importante que o professor informe ao aluno que as classes desenvolvidas deverão ter um nome específico. Por exemplo: Se for pedido ao aluno que crie uma classe que implemente uma calculadora, deve ser especificado que a classe deverá ter o nome "CalculadoraImpl", por exemplo.

4) No ato da submissão do caso de teste, o professor deve informar o nome da classe pedida ao aluno e o nome do caso de teste que a testa. Exemplo: Se for pedido ao aluno que implemente uma calculadora, no momento de submissão do caso de teste deverá ser especificado nos campos corretos que TesteCalculadora é para testar CalculadoraImpl, por exemplo.

5) O caso de teste deverá receber por parâmetro uma variável do tipo Object. Isso porque, em tempo de desenvolvimento, o projeto ainda não conhece as classes corretas para instanciar. Além disso, é importante que se dê um *cast* usando uma variável do tipo da interface que está no pacote *javali.interfaces*. Veja exemplo de como deverá ser uma

função para testar a soma de uma calculadora:

```
1  package javaliproject.testes;
2
3  import javaliproject.interfaces.Calculadora;
4
5  public class TesteCalculadora {
6
7      public double testSoma(Object calculadora) {
8          Calculadora calc = (Calculadora)calculadora;
9          int retornoEsperado = 10;
10         int retornoFeito = calc.soma(5, 5);
11
12         double result;
13         if(retornoEsperado==retornoFeito){
14             System.out.println("O método soma está correto");
15             result = 1;
16         }else{
17             System.out.println("Método soma com defeito");
18             result = 0;
19         }
20
21         return result;
22
23     }
```

6) O retorno dos métodos deverá ser do tipo double. Ao final de cada método, o valor retornado deverá ser 0 ou o valor total que será atribuído para aquele método. Por exemplo, se o aluno tiver no total de suas classes 4 métodos para serem testados, cada método valerá 2,5 pontos. Então 2,5 deverá ser o valor retornado para a função caso o método passe no teste. Isso é necessário para que ao final da execução de todos os métodos, a nota seja gerada. Veja um exemplo na imagem do item anterior.

7) É importante informar ao aluno que ele precisa implementar a mesma interface que está no projeto do professor. Por exemplo, se no teste o professor usar a interface Calculadora do pacote *javaliproject.interfaces*, o aluno deverá escrever sua classe implementando essa mesma interface com o mesmo pacote.

8) É importante deixar claro para o aluno que ele deverá implementar suas classes dentro de um pacote que será especificado no enunciado da atividade.