

Projeto Integrador Transdisciplinar em Ciência da Computação II



Conteudista: Prof. Me. Artur Marques

Revisão Textual: Laís Otero Fugaitti

☰ Material Teórico

🔗 Material Complementar

DESAFIO

☰ Situação-Problema 1

☰ Situação-Problema 2

☰ Situação-Problema 3

☰ Problema em Foco

ATIVIDADE

☰ Atividade de Entrega

Material Teórico

Olá, estudante!

Vamos iniciar a disciplina abordando os conceitos necessários para que você possa realizar a atividade através das situações-problema mais à frente.

i **Atenção, estudante! Aqui, reforçamos o acesso ao conteúdo *online* para que você assista à videoaula. Será muito importante para o entendimento do conteúdo.**

Começamos nossa jornada de desafios e projetos na disciplina anterior, Projeto Integrador Transdisciplinar (PIT) em Ciência da Computação I, com o planejamento, ou seja, a definição do MVP (produto mínimo viável). Ele é importante pois seu foco está em permitir que possamos pensar em algo e rapidamente realizá-lo/produzi-lo. Para isso utilizamos metodologias e técnicas que, quando combinadas, permitem a visualização e prototipagem, possibilitando assim que, se errarmos ou não gostarmos do que estamos

construindo, possamos aplicar modificações rapidamente sem gastar muito dinheiro.

Uma vez conseguindo obter o que queremos com o mínimo de funcionalidades possível para tornar a solução funcional, chamamos de MVP. Vamos definir isso? É uma técnica de desenvolvimento em que um novo produto é introduzido no mercado com características básicas mas suficientes para chamar a atenção dos consumidores, então é lançado no mercado somente após obter opinião suficiente dos usuários iniciais do produto. Na verdade esses usuários serão algo como uma amostra estatística na qual você testará sua hipótese (seu produto ou solução).

Portanto, se você refletir a esse respeito, estamos tratando da versão básica do produto ou sistema que desejamos lançar no mercado. Então, quando apresentamos a versão básica aos consumidores, desejamos avaliar a resposta deles. Isso ajuda a tornar o produto muito melhor.


Pense no MVP como uma técnica que provê três recursos distintos:

- O produto/sistema terá recursos suficientes para os usuários comprarem;
- O mecanismo de *feedback* no qual os usuários poderão enviar seus comentários. Os usuários ajudam a fazer as mudanças desejadas no produto, gerando economia de recursos humanos e materiais para a empresa em última instância e possibilitando que o time de desenvolvimento foque no que é essencial, ou seja, o que o cliente quer e não o que imaginamos que ele quer;
- Mostrar os futuros benefícios para os clientes que adotarem o produto no MVP, assim fazendo com que “topem” continuar usando e esperando pelas novidades vindouras (*roadmap* de *features*).

O objetivo principal do MVP é sempre minimizar o tempo e esforço desperdiçados testando como o mercado reage à sua ideia antes de construir o produto completo. Veja exemplos de como os MVPs podem ajudar:

- Valide as hipóteses de ideias de produtos com dados da vida real;
- Reduza o tempo de colocação no mercado de novos lançamentos de recursos;
- Agregue valor aos seus primeiros usuários rapidamente. O MVP é o caminho mais curto que agrega maior valor aos seus primeiros clientes e, ao mesmo tempo, gera aprendizado para você;
- Teste seu produto/ajuste de mercado antes de construir um produto completo;
- Colete dados viáveis sobre o comportamento do usuário para moldar futuras iniciativas de produtos e estratégia de entrada no mercado. Os MVPs podem ser usados como um componente principal em seu processo de priorização de produtos para ajudá-lo a tomar decisões baseadas em dados;
- Aumente a base de usuários de pré-lançamento;
- **Elimine o desperdício:** economize dinheiro e tempo que, de outra forma, seria gasto em ideias infrutíferas.

O processo consiste nas seguintes etapas:

- 1 Desenhe um mapa do seu ecossistema;
 - 2 Defina a proposta de valor para cada parte interessada;
 - 3 Posicione um MVP final;
- 

4

Identifique o maior risco;

5

Crie seu caminho de valor (PRODUCTBOARD, 2021, p. 3-5.).

Claro que, para chegar até o MVP, você teve que experimentar, construir, aprender e reconstruir até que os ajustes apontassem para algo viável e funcional. Entretanto, a jornada não terminou. Após isso, você teve que construir alguns documentos de escopo e especificações (o que? e como?) para documentação, entendimento e compartilhamento quando necessário.

Portanto, desenvolvemos uma visão do produto/*software* e o utilizamos para comunicar o que ele deve fazer e o que não deve fazer. É importante que você coloque "no papel" o que tem em mente, porém, jamais esqueça de comparar o que você pretende com os concorrentes (por mais original que seja sua ideia, sempre há partes já desenvolvidas de uma solução que podem abreviar em muito a curva de desenvolvimento) e mostrar suas vantagens. Uma seção muito importante da visão do produto listou os tipos de usuários (funções) e como o sistema irá agregar valor a eles e tornar seu trabalho melhor. Caso você não tenha pensado nisso, considere voltar ao seu documento e revisar incrementando essas informações, seu produto/*software* só terá a ganhar.

Junto com a visão, fizemos um documento de requisitos (aqui não importa se esse documento foi feito usando metodologias ágeis ou sequenciais). Ele deve conter requisitos funcionais e não funcionais. Às vezes, eles são separados em casos de uso para requisitos funcionais e especificações suplementares para requisitos não funcionais. Por que estou escrevendo isso? O motivo é simples, houve um interlúdio entre o PIT I e o PIT II. Esse tempo é bem-vindo, pois nos permite refletir, aprender coisas novas e dominar técnicas. Tudo isso nos traz novas ideias e com elas a necessidade de revisitar os documentos elaborados e atualizá-los, ajustar seu rumo, corrigir ou melhorar conceitos para uma construção acurada.

Por isso, quando falamos em ter uma ideia de desenvolvimento partindo das nossas orientações iniciais e dos limites no material didático do PIT I, levamos em consideração a visão do produto, que foi construída usando *storytelling*, *canvas* ou *design thinking*), pois são os melhores instrumentos hoje para lastrear uma ideia em estágios iniciais e verificar, além de sua validade, seus conceitos. É mandatório e geralmente é preparado no estágio inicial do projeto, no qual apenas especificações de alto nível são definidas e têm como objetivo principal obter um "de acordo" com o cliente.

Quando o projeto é iniciado, especialmente os grandes, dois documentos são preparados por analistas de negócios, o documento "*as is*" (ou "como é") e o "*to be*" (como será). Aqui estou demonstrando como um projeto nasce a partir de uma ideia, apesar de que mesmo que um produto/*software* preexistente já esteja estabelecido, ele pode sofrer atualizações, mudança ou interrupção. Portanto serve com certeza para o seu caso.

As is descreve o ambiente atual e o fluxo de trabalho no domínio do negócio (algumas coisas são feitas manualmente, por exemplo, usando papéis ou, no caso da área de saúde, exames diagnósticos dependem do especialista médico ou do citologista para saber se uma célula é ou não cancerosa). *To be* descreve como o cliente deseja que as "coisas" sejam (como um *software* será usado em vez da papelada, ou ainda como um algoritmo de visão computacional e *machine learning* diagnosticará câncer a partir das imagens de exames que receberá).

O *as is* e o *to be* enriquecem em muito a visão do projeto e de seu produto/serviço, pois, a partir deles, o analista começa a fazer os casos de uso e as especificações suplementares, que às vezes não são captadas logo de início. O importante aqui é que você perceba que pode combinar ou remover documentos conforme desejar ou precisar para que se adequem ao seu projeto, porém devem fazer sentido e devem manter a completude da ideia.

Projetos menores não precisam de tantos documentos e não envolvem muitas funções. Grandes projetos precisam dessa separação porque diferentes documentos falam para pessoas diferentes. Principalmente se você deseja se comunicar com altos cargos gerenciais ou até mesmo o "*C-level*", não deve fornecer a eles documentos com conteúdo que eles não tenham interesse de ler (cada documento deve ser construído com a linguagem adequada, não adianta colocar linguagem técnica para um membro do conselho ou um vice-presidente de finanças, não porque eles não saibam, é que não há tempo para isso, eles confiam em você e esperam que você esteja maduro tecnicamente para fazer jus ao cargo que ocupará no futuro).


Mas, estamos nesse nosso PIT II descrevendo a fase de desenvolvimento e temos muito a fazer, certo?! Vamos descrever um pouco sobre isso daqui por diante.

Creio que haja uma ansiedade grande por parte da maioria quando falamos em codificação, afinal, quando falamos em escrever programas em uma linguagem de programação, como Java, PHP, *Python*, ou *front-end* de aplicações utilizando HTML, CSS ou *JavaScript*, muitos mudam seus

semblantes, e essa jornada gratificante e enriquecedora acaba se tornando um verdadeiro tormento ou pesadelo para tantos. Porém precisamos sair desse mito e mudar nossa perspectiva, nosso *mindset*. É claro que estudantes de ciências da computação são apaixonados pelo que fazem e isso envolve a codificação, porém, devemos levar em conta muitas vezes a falta de experiência natural de quem está começando, por isso vamos quebrar esse paradigma iniciando com possibilidades de realização por meio do emprego do *no-code* e do *low-code*.

Trata-se de um movimento importante no qual, mais que permitir que usuários tenham empoderamento para aplicações baseadas em *web*, que desenvolvedores possam ter maior produtividade e *gamers* criem seus modais e desenvolvam seus conceitos antes de partirem para a programação pesada. Para soluções simples e elegantes, *no-code* pode ser a solução.

No-code para *web* trata de *JavaScript*, *HTML5* e *CSS*, mas não exige conhecimento profundo inicial num primeiro momento, o que se torna a solução ideal para você que está desenvolvendo um projeto. Claro, não é uma panaceia para todos os projetos que serão desenvolvidos por vocês nesta disciplina, mas trata-se de um facilitador pelo tempo que é limitado e exigirá foco e dedicação. Vejamos agora alguns conceitos.



“O desenvolvimento sem código é um tipo de desenvolvimento para a *web* que permite que não programadores e programadores criem *softwares* usando uma *interface* gráfica com o usuário, em vez de escrever código. O movimento sem código repousa sobre a crença fundamental de que a tecnologia deve permitir e facilitar a criação, não ser uma barreira à entrada. Muito do que fazemos em nosso dia a dia é alimentado por código. Estejamos verificando nossas contas bancárias, curtindo fotos de amigos nas redes sociais ou procurando novas roupas em nossos *sites* de comércio eletrônico favoritos, a programação é o que torna todas essas ações possíveis.

O que antes era um espaço que apenas desenvolvedores e especialistas em programação podiam navegar, agora está aberto a todos. O movimento sem código

removeu o obstáculo de ter que conhecer linguagens de programação, permitindo que qualquer pessoa traga suas ideias à luz.

Quando dizemos *no-code* é simplesmente uma camada de abstração sobre o código. Ou seja, ele pega os fundamentos do código e os traduz em soluções simples de arrastar e soltar, permitindo que os criadores criem aplicativos e *sites* modernos visualmente. Há dezenas de ferramentas ou IDEs que oferecem plataformas de desenvolvimento sem código, disponibilizando todas as funcionalidades do HTML5, CSS e *Javascript*, mas você não precisa conhecer nenhuma dessas linguagens de programação para começar a construir.

As plataformas de desenvolvimento *web* sem código percorreram um longo caminho desde os editores *WYSIWYG* – *What you see is what you get* (o que você vê é o que você pega) do passado. Onde esses *designs* geravam transitáveis para a época, esses *sites* eram simples, oferecendo uma experiência unilateral para o usuário. Em seguida, surgiram os construtores de *sites* mais dinâmicos que as plataformas originais sem código não podiam fazer, ou seja, construir *sites* cheios de interações, animações dinâmicas e outros elementos visuais sofisticados.

Existem muitos casos de uso para o não código. Não se limita apenas a construir *sites*. Ele pode ser usado para construir aplicativos móveis, aplicativos da *web*, aplicativos de voz, ferramentas internas, integrações e para automação de tarefas. Sem conhecer uma única linha de código, é possível construir *chatbots* com *Voiceflow*, conectar vários aplicativos e construir fluxos de trabalho automatizados com *Zapier* e utilizar *Shopify* para executar lojas de comércio eletrônico."

- **WEBFLOW, 2021, p. 1-2**

Interessante e útil, não é mesmo? Não estou tentando criar um atalho aqui, mas lhe apresentar uma maneira moderna e atual de trabalhar com desenvolvimento! Sim, trabalhamos dessa forma nas empresas modernas e atualizadas tecnologicamente, que devem enfrentar uma concorrência feroz e que o fator tempo representa mais que dinheiro, mas sobrevivência.

Importante termos consciência de que não sabemos mais quem são nossos concorrentes. De fato, com a digitalização das empresas e do poder computacional hoje existente, um simples programador em algum canto obscuro do planeta pode estar fazendo a diferença na cura de doenças, criando acessibilidade e inclusão através de seus aplicativos e serviços, criando empresas cujo valor de mercado ultrapassam o produto interno bruto (PIB) de muitas nações. Então, quanto mais tecnologia você tiver ao seu alcance e mais souber sobre elas, mais ajuda terá para fazer a diferença.

As plataformas sem código oferecem uma camada de abstração sobre o código. Ou seja, elas pegam os fundamentos do código e os traduzem em uma solução simples de arrastar e soltar – permitindo que os criadores construam aplicativos e sites modernos visualmente.

Os protótipos iniciais de um produto digital geralmente não precisam de nada perto do investimento em engenharia, como no estágio de lançamento. Já pensou em rever seus protótipos do PIT I usando *no-code*?! Este é o momento.

É claro que isso não vai fazer com que você não programe, na verdade você programará o que é a parte importante, a inteligência de processamento de seu projeto, o que o diferenciará dos demais. Portanto:

- Ferramentas sem código geram código!
- Ferramentas sem código são executadas por “pilhas” de código!
- Ferramentas sem código interagirão com seu código!

Elas exigirão programadores, engenheiros de sistemas e cientistas da computação, ou seja, pessoas que podem pensar em abstrações e, acima de tudo, pessoas que sabem como unir as ferramentas

certas da maneira certa para produzir valor.

Aqui vão algumas plataformas para você acessar e pesquisar para suas construções:

- [Appy Pie](#): construtor de aplicativos móveis sem código, construtor de *sites*, *chatbot* e *workflow* de automação nos seguintes *links*:
 - [App builder](#);
 - [Website builder](#);
 - [Chatbot](#);
 - [Connect](#).
- [Voiceflow](#);
- [Airtable](#);
- [Ninox](#);
- [VINYL](#).

Mas e se você for uma pessoa que gosta de jogos de computador e seu projeto se enveredou por esse caminho, saiba que não é nenhum segredo que fazer um jogo de computador geralmente requer um conhecimento sólido de programação. Mas há também a possibilidade de se utilizar *no-code* e *low-code* para fazer jogos.

Inclusive, trata-se de um ramo muito promissor e que a indústria está captando cada vez mais pessoas. As opções de arrastar e soltar e outros recursos fáceis permitem que qualquer pessoa que nunca tenha programado na vida faça um jogo. Apesar da simplicidade, ainda é possível fazer bons jogos, e as habilidades que você adquiriu durante sua jornada pelo curso promoverão a programação complementar para encerrar seu jogo de forma competente. Vejamos algumas

opções que você possui para acelerar seu desenvolvimento. Todos os *links* vêm com vídeos tutoriais e cursos para que você desenvolva com segurança. Agora é o momento de investir pesado em sua carreira.

- [Game Maker](#);
- [Unity](#);
- [Godot](#);
- [CryEngine](#);
- [Evergine](#);
- [GDevelop](#);

E em nosso desafio de projeto há também os que preferiram criar suas soluções baseadas em *internet* das coisas (IoT) com uma plataforma de *hardware* baseada em *Arduino* ou *Raspberry Pi*. Para esses temos algumas dicas de como conseguir prototipar ou acelerar o desenvolvimento usando *no-code* também. Isso inclui projetos de análise de dados e aprendizagem de máquina.

- [Mendix](#); Abordagem visual orientada por modelo da plataforma do aplicativo IoT permite que desenvolvedores profissionais e menos técnicos consumam serviços IoT das melhores plataformas IoT da classe, incluindo AWS, IBM Watson, *Microsoft Azure* e KPN *LoRa*. Por meio dessa estratégia agnóstica de plataforma, o *Mendix* permite que as organizações criem experiências ricas em dispositivos conectados para transformar suas operações, produtos e modelos de negócios;
- [ThingSpeak](#); Trata-se de uma solução grátis baseada na linguagem MATLAB. Permite agregar, analisar e visualizar fluxos de dados ao vivo. Os dispositivos IoT enviam seus dados ao vivo diretamente para o *ThingSpeak*. A partir daí, você cria visualizações instantâneas e pode enviar alertas usando serviços da *web*. Como uma plataforma de

nuvem, *ThingSpeak* torna possível construir projetos de IoT sem a necessidade de desenvolver seu próprio *software*. Além disso, você não precisa configurar um servidor;

- ***Blynk***: Trata-se de uma plataforma IoT independente de *hardware* que vem com gerenciamento de dispositivos, análise de dados e funcionalidades de aprendizado de máquina, permitindo que você se conecte a qualquer dispositivo. Além disso possui um construtor de aplicativo móvel que permite construir aplicativos IoT arrastando e soltando. Isso vai facilitar bastante a construção de seu projeto;
- ***Record Evolution***: É um serviço em nuvem criado para facilitar o desenvolvimento de soluções de inteligência artificial (IA) para dispositivos de ponta de IoT. A plataforma agnóstica de *hardware* ponta a ponta para IoT e AI vem com ferramentas que cobrem todo o ciclo de desenvolvimento de IoT. Isso inclui gerenciamento de dispositivos, desenvolvimento de aplicativos em um IDE baseado em nuvem com implantação instantânea, além de um conjunto de ferramentas de armazenamento de dados. Pode-se iniciar com a coleta de dados de dispositivos, transformação e modelagem até análises avançadas e aprendizado de máquina.

Para os que já possuem uma massa de dados prévia e quer apenas descobrir padrões ou submeter dados para visualização, exploração ou uso de algum algoritmo de ciência de dados, eu recomendo fortemente que você considere utilizar o *RapidMiner*. Ele é útil para preparação de dados, aprendizagem de máquina e operacionalização de modelos.

Sites

Rapidminer

Clique no botão para conferir o conteúdo.



ACESSE

Rapidminer – Educational License: Acesse a versão para estudantes para usar durante seu projeto

Clique no botão para conferir o conteúdo.

ACESSE

Para darmos início ao desenvolvimento, é importante que tenhamos planejada a gerência de configuração da nossa solução. Para Pressman (2011) a gerência de configuração de *software* é um conjunto de atividades destinadas a controlar a mudança, identificando os produtos de trabalho que são passíveis de mudança, estabelecendo relações entre eles, definindo mecanismos para gerenciar diferentes versões desses produtos de trabalho, controlando as mudanças impostas e auditando e relatando as mudanças feitas.

Portanto é a arte de identificar, organizar e controlar modificações no *software* que está sendo construído por uma equipe de desenvolvimento. Além disso, ela maximiza a produtividade e minimiza erros.

É necessário estabelecer e manter a integridade dos produtos do projeto de *software* ao longo do seu ciclo de vida. As atividades necessárias para realizar isso incluem identificar itens e unidades de configuração, controlar sistematicamente as mudanças e manter a integridade e a rastreabilidade da configuração ao longo do ciclo de vida do *software*. Então, ao identificarmos os itens que precisam ser configurados, devemos lembrar que todos os artefatos do projeto são candidatos a: documentos, modelos gráficos, protótipos, código e qualquer entrega interna ou externa que pode sofrer alteração.

Quando programamos temos muitas vezes o hábito ruim de sobrescrever alterações sobre alterações, código sobre código, o que com certeza dificulta quando precisamos encontrar *bugs* ou pior, quando o código para de funcionar, não conseguimos retornar a uma versão anterior que o permita funcionar novamente. E isso estressa em demasia, como você já deve ter se deparado em algum momento de sua jornada. Para isso eu indico fortemente que você, durante sua codificação,

caso não opte por utilizar ferramentas *no-code*, que utilize um controlador de versionamento, porque ele:

- Melhora a produtividade da equipe e permite a colaboração;
- Melhora a comunicação da equipe com uma solução confiável;
- Reduz erros e conflitos de desenvolvimento;
- Melhora a satisfação do cliente com versões de *software* confiáveis.

Mesmo que nesse desafio você tenha que desenvolver sozinho sua solução, é mandatório utilizar controle de versionamento. Há vários *softwares* que podem fazer isso com maestria, os mais populares mundialmente são *Subversion* (também conhecido como *SVN*), *Git*, *Mercurial* e *Bazar*. Vou focar no SVN e no conhecido *Git*. Ele depende de um repositório central.

Site

TortoiseSVN – Windows

Clique no botão para conferir o conteúdo.

ACESSE

Apache Subversion – Linux

Clique no botão para conferir o conteúdo.

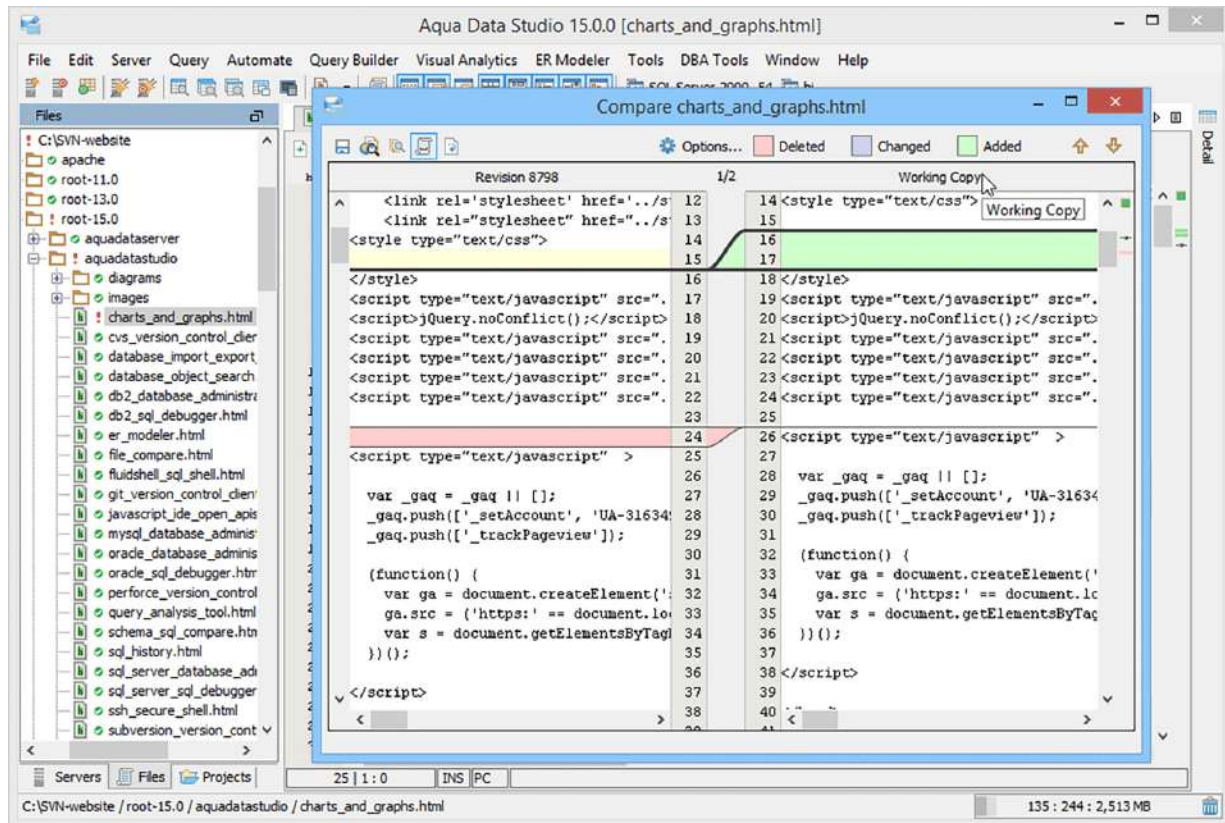


Figura 1 – Interface Subversion Aqua Data Studio

Fonte: Reprodução

Por outro lado, o *Git*, que está inserido no popular *GitHub*, permite colaboração, revisão de código e gerenciamento de código tanto para *open source* quanto projetos privados. O *Git* se tornou o método preferido para muitos desenvolvedores de *software* de código aberto compartilharem e colaborarem em projetos. Porém, não é bem documentado e muda rapidamente, de modo que acompanhar todos os seus recursos pode ser um desafio, mesmo com os vídeos e cursos que tem por aí.

Site

Git for Windows

Clique no botão para conferir o conteúdo.

ACESSE

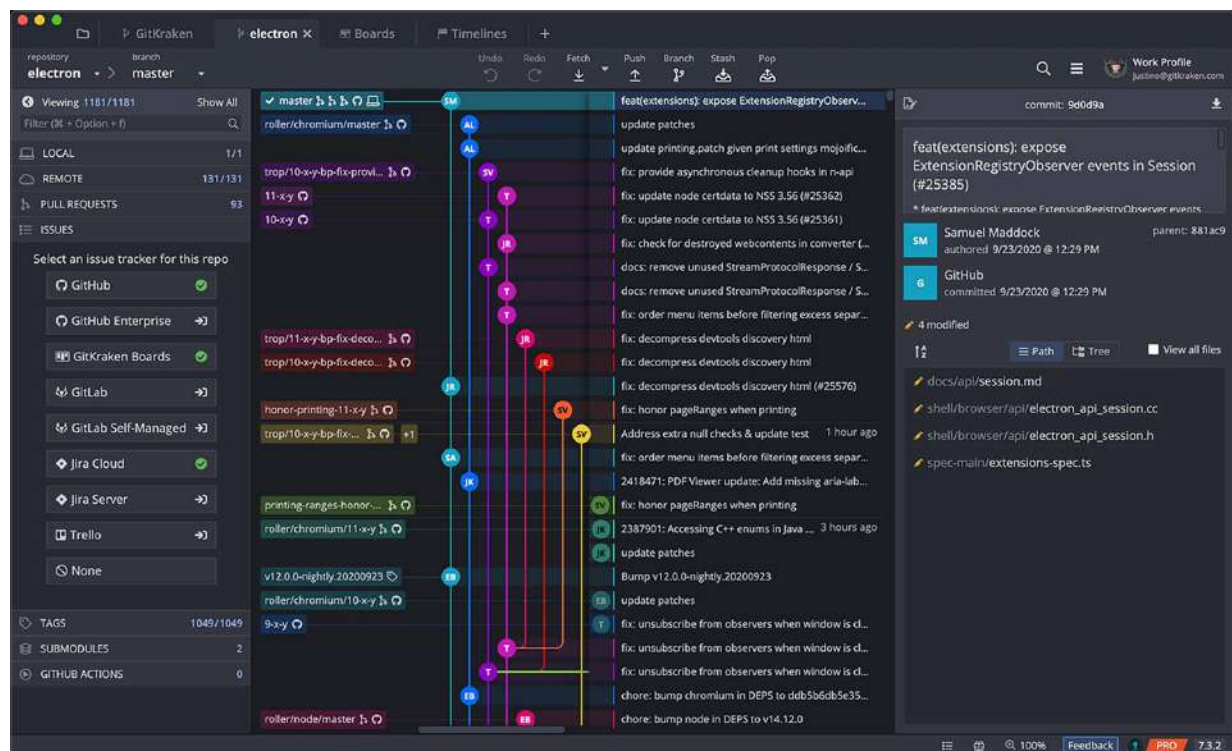


Figura 2 – Interface Git

Fonte: Reprodução

Por fim, um conceito importantíssimo no desenvolvimento de um projeto, e que ocorre durante seu desenvolvimento, é o teste, ou seja, verificação e validação da solução. O teste é a técnica tradicional utilizada para determinar e garantir a qualidade dos produtos. Trata-se, em linhas gerais, da definição ou descrição de um produto de qualidade e os métodos de teste utilizados para garantir que ela seja bem estabelecida. O sucesso desses métodos depende da definição do que constitui um produto de qualidade, da determinação de propriedades mensuráveis que refletem a qualidade, da derivação de critérios significativos de teste baseados nas quantidades mensuráveis e da formulação de testes adequados para garantir essa qualidade.

Todavia, isso não é necessariamente aplicado ao *software* ou a certas soluções que utilizam abstrações matemáticas e algoritmos para sua construção. As características de alto nível do *software* de qualidade são confiabilidade, testabilidade, usabilidade, eficiência, capacidade de carga e manutenção. Na prática, a eficiência muitas vezes acaba por estar em conflito com outros atributos, por exemplo, manutenção e testabilidade.

Outros termos que causam confusão entre os desenvolvedores são as diferenças entre verificação e validação. Embora a distinção possa parecer trivial, os dois cumprem objetivos muito distintos.

A verificação do desenvolvimento refere-se à verificação do aplicativo que ainda está em desenvolvimento, para garantir que está de acordo com essas especificações. Essas verificações podem ser algo tão simples quanto ler as especificações e compará-las com a lógica do código para garantir que

estejam alinhadas. O processo de verificação incluirá atividades como revisões de código, orientações, inspeções, mas poucos, se houver, testes reais.

Enquanto a verificação ocorre enquanto o produto ainda está em desenvolvimento, a validação é realizada após a conclusão de um determinado módulo, ou mesmo a conclusão de toda a aplicação. A validação se concentra em garantir que as partes interessadas obtenham o produto que desejam. No esforço de validação não importa como você chegou lá, apenas que você chegou e que tudo está conforme o esperado.

Isso infelizmente não aflige com dúvidas apenas os testes, há dúvidas também em outra fase importante que trata da validação dos requisitos. Muitos estudantes perguntam como validar os requisitos quando utilizamos metodologia ágil e se é a mesma coisa com metodologias sequenciais ou tradicionais. De forma simples, vou descrever aqui que a validação de requisitos, independente de qual metodologia utilize, trata de garantir que os requisitos tenham alcançado os objetivos do negócio, atendam às necessidades de quaisquer partes interessadas relevantes e sejam claramente compreendidos pelos desenvolvedores.

Então, a validação é uma etapa crítica para encontrar requisitos ausentes e garantir que eles tenham uma variedade de características importantes:

- Descrever corretamente a necessidade do usuário final;
- Ter apenas um significado exato;

- Poder ser modificada conforme necessário;
- Documentar seus atributos e garantir que eles são realmente o que os clientes precisam;
- Vincular facilmente a esses requisitos os códigos e testes.

Portanto, entenda que qualquer forma de validação em *software* (de requisitos ou de código) não está focada no caminho que você percorreu para chegar ao destino, mas sim, se você atingiu o alvo.

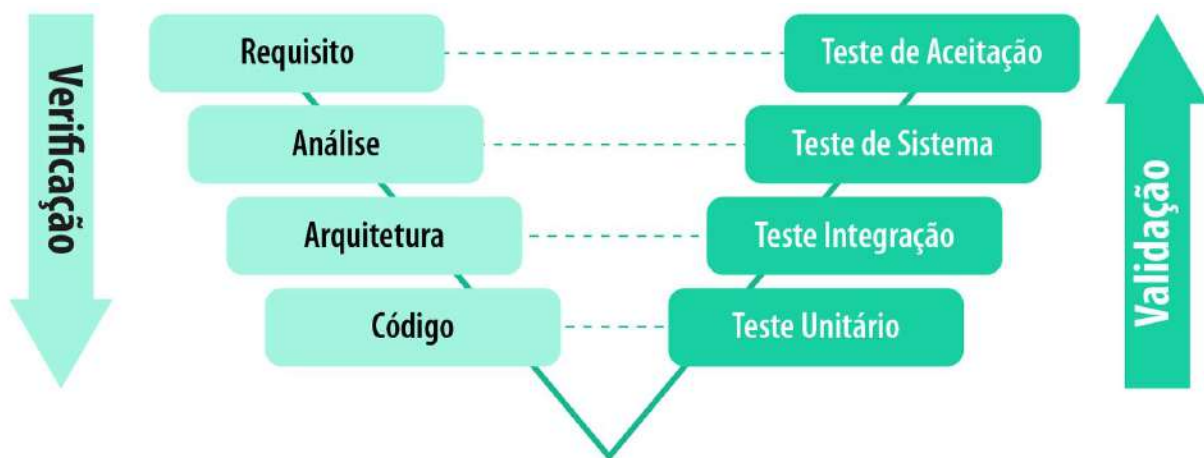


Figura 3 – Exemplo de relação entre o desenvolvimento e os testes com o Modelo V

Fonte: Adaptado de COSTA

Para lhe ajudar nos testes unitários temos o seguinte:

Sites

JUnity

Clique no botão para conferir o conteúdo.

ACESSE

PHP Unity

Clique no botão para conferir o conteúdo.

ACESSE

PyTest

Clique no botão para conferir o conteúdo.

ACESSE

Pytest – Documentation

Clique no botão para conferir o conteúdo.

ACESSE

Pytest – Tutorial

Clique no botão para conferir o conteúdo.

ACESSE

Por fim, é necessário que você escolha uma arquitetura para sua solução caso seja baseada em *software*. Para essa finalidade recomendo que você utilize o *Model View Controller (MVC)*, por motivos de facilidade de entendimento e de facilidade na construção. O MVC é um padrão de *design* de *software* previsível, que pode ser usado em muitas estruturas com muitas linguagens de programação, e isso inclui todas as que usei algumas linhas acima para que você pudesse ter ferramentas para executar testes.

O padrão de *design* MVC serve para separar a camada de apresentação da lógica de negócios. É tudo muito elegante e de entendimento rápido, já que há apenas três elementos principais a se levar em consideração:

- **Model (ou modelo):** que serve para armazenar e gerenciar dados, ou seja, um banco de dados;
- **View (ou visualização):** trata-se da interface gráfica do usuário ou GUI. Aqui trata-se de uma representação visual dos dados: gráfico, diagrama, tabela, formulário etc. A visualização contém todas as funcionalidades que interagem diretamente com o usuário e isso pode incluir o clicar em um botão ou em um evento de entrada de dados ou funcionalidade;
- **Control (ou controlador):** é onde fica o "cérebro" do aplicativo. O controlador conecta o modelo e a visualização. Serve para converter as entradas da visualização em demandas para recuperar ou atualizar dados no modelo. O controlador recebe a entrada da visualização, usa a lógica para traduzir a entrada em uma demanda para o modelo (regras de sistemas, regras de negócios etc.), o modelo pega os dados e o controlador passa os dados do modelo de volta para a visualização para o usuário ver. Isso permite que não haja exposição de dados do banco além do que foi pedido e impede que haja acessos não autorizados, quando bem construído.

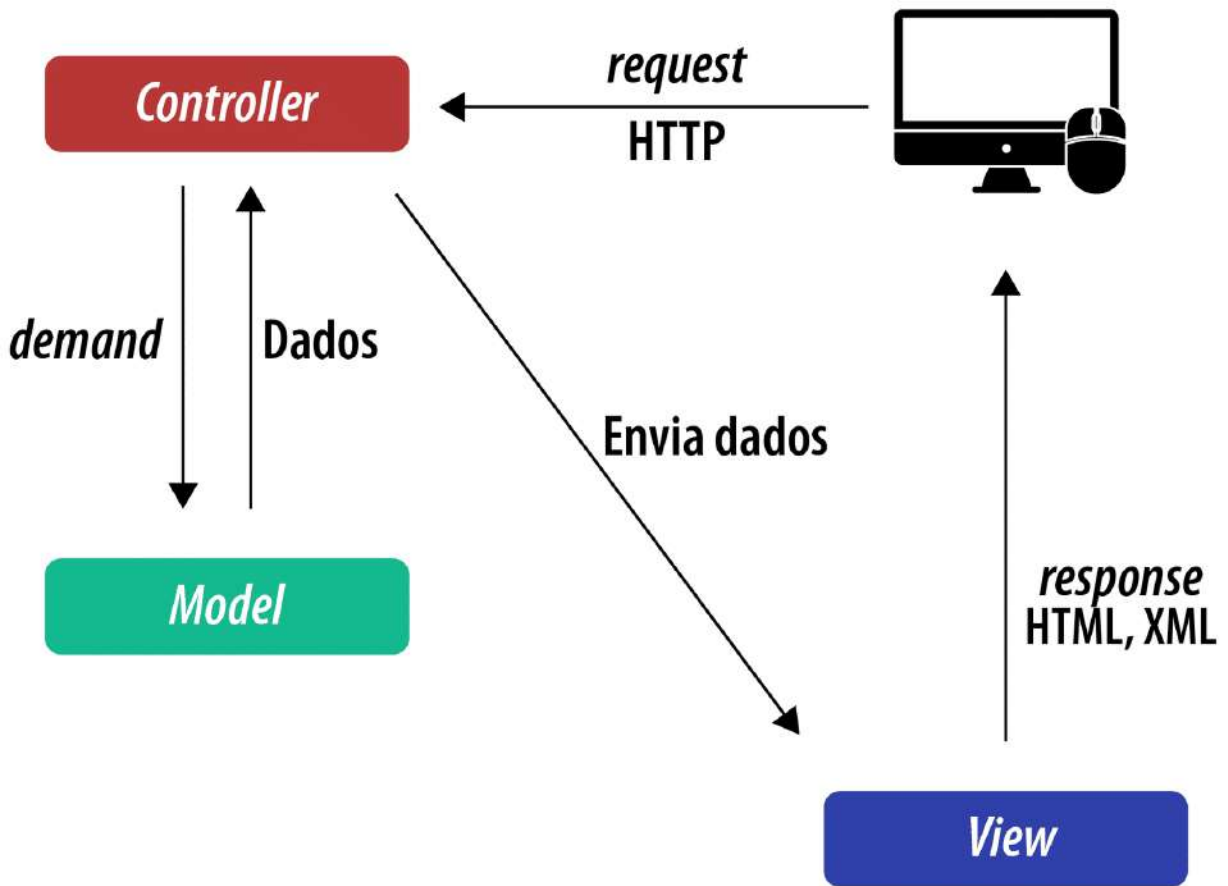


Figura 4 – Esquema de funcionamento do MVC

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta disciplina:

VÍDEOS

Curso de Controle de Versão com *Subversion*

Uso do controle de versão com *Subversion* na gerência de configuração.

Clique no botão para conferir o vídeo indicado.

ASSISTA

Curso de *Git* e *GitHub*: Grátis, Prático e Sem Usar Comandos no Terminal

Aprender o passo a passo como instalar e utilizar o *Git* e o *GitHub* para controle de versionamento nos seus programas.

Clique no botão para conferir o vídeo indicado.

ASSISTA

Criando um CRUD FÁCIL usando o *Framework Spring MVC*

Nessa aula, você aprenderá como utilizar o *framework Spring MVC* e o *Spring Data JPA* com o *Spring Boot* para criar um CRUD de forma aplicada no Java.

Clique no botão para conferir o vídeo indicado.

ASSISTA

Como criar um aplicativo de restaurante com Appy Pie?

Passo a passo para criar seu próprio aplicativo em *Appy Pie*.

Clique no botão para conferir o vídeo indicado.

ASSISTA

Situação-Problema 1

Caro(a), estudante.

Agora, vamos compreender o cenário que será abordado na primeira situação-problema da disciplina.

Atente-se à situação profissional que você precisará entender para poder realizar a atividade.

Você deverá revisar seu projeto e fazer os ajustes necessários para que ele atenda aos requisitos de construção. Trata-se de uma oportunidade para que você possa refletir melhor sobre o que e como escreveu no PIT I como planejamento e realizar um ciclo completo de melhoria contínua.

Situação-Problema 2

Vamos compreender o cenário que será abordado na segunda situação-problema da disciplina. Atente-se à situação profissional que você precisará entender para poder realizar a atividade.

Está na hora de você pôr a mão na massa.

Selecione a arquitetura, *design pattern*, melhor configuração e linguagem de sua preferência para desenvolver a codificação completa, ou seja, o *software* e/ou *hardware*, ou ainda, colocar o *software* embarcado no dispositivo que projetou.

Lembre-se de que, conforme explicamos no material teórico, como facilitador, você pode optar por desenvolver por meio de ferramentas *low-code*; porém pode fazer da forma tradicional se se sentir mais confortável.

Situação-Problema 3

Por fim, vamos compreender o último cenário, abordado na terceira situação-problema da disciplina. Atente-se à situação profissional que você precisará entender para poder realizar a atividade.

Uma vez desenvolvido o *software*/artefato, é importante fazermos todos os testes de funcionamento e situações imprevisíveis que somente os usuários podem gerar. Pedimos que ofereça a solução desenvolvida por você para 5 (cinco) colegas testarem e gerarem um relatório a que chamaremos de laudo. Você juntará esses laudos, realizará correções e melhorias sugeridas, negará o que não for possível e nos entregará esses relatórios, suas respostas e mudanças que foram feitas.

Problema em Foco

Aqui vão algumas orientações para o desenvolvimento de seu projeto.

Situação 1

Você deverá revisitar todos os documentos criados no seu PIT I e revisá-los com a intenção de realizar um ciclo de melhoria contínua, deixando a documentação de planejamento mais robusta e mais clara. Você deverá utilizar fortemente as suas competências de atenção aos detalhes, abstração e *feedback* de seus colegas, pedindo por parte deles a crítica da solução que você pretende implementar. As contribuições deles serão de imensa valia ao seu projeto, assim como as suas para com os deles. O mais importante é como você reage às críticas e as aceita como forma de melhorar como cientista da computação e também como pessoa. Lembre-se, não escolhemos com quem trabalhamos, portanto é importante desenvolver empatia (coloque-se no lugar do outro). Uma vez revisado os documentos e escolhida as melhores sugestões de seus colegas e, é claro, as suas próprias sugestões, compile tudo num único volume, ele será seu guia daqui em diante nos próximos desafios.

Situação 2

Aqui o foco é construção, portanto esperamos muita codificação de sua parte para que o código, artefato ou análise de dados possam ser desenvolvidos e apresentados. Lembre-se de que você tem liberdade de escolha de:

- Linguagem;
- Banco de dados;

- Hospedagem;
- Plataforma;
- Modo de codificação (tradicional ou *low-code*).

O que se espera de você como indivíduo inserido na área da ciência da computação é forte colaboração de sua parte para com seus colegas e vice-versa. Isso não significa em hipótese alguma que o projeto deve ser feito em grupo. Não é!

Porém, a troca de ideias e iniciativas de aprendizagem que caracterizam essa profissão para o resto de nossos dias deve ser levada adiante para que todos possamos sair maiores e melhores do que quando iniciamos. Portanto, não se acanhe em ajudar seu colega caso esteja em dificuldade, ambos só têm a ganhar.

Esperamos que sejam entregues o código, artefato ou análise, todos funcionando, neste desafio (códigos fontes – *front end* e *back end*, bibliotecas, manual de uso, vídeo gravado demonstrando o funcionamento – pelo menos cinco minutos de duração narrado por você). Caso haja um produto além do *software*, ele deverá estar documentado e deverá ser enviado em conjunto, conforme orientação de seu tutor.

Situação 3

Nessa situação, o desafio é a realização de testes feitos pelos clientes, que, no seu caso, são seus colegas julgando o que você desenvolveu. É importante que nesse momento você já tenha escolhido quais serão seus dublês de usuários entre seus colegas e forneça-lhes o endereço para que possam testar sua solução como só um usuário saiba fazer.

Talvez você pergunte: "Professor, mas se eu desenvolver um robô, por exemplo?". Bom... Nesse caso, é simples. Você escolhe colegas que estejam próximos ao lugar que você mora e os convida a testar ou leve até eles para que seja testado. O importante aqui é desenvolver a capacidade de aceitação da mudança, reconhecimento do erro como forma de melhorar continuamente, trabalho em equipe e, por fim, resiliência necessária para continuar e perseverar para atingir o resultado.

É importante que você entregue as cinco opiniões/testes sobre seu *software*/produto como sendo a prova do sucesso nesse desafio (um arquivo PDF) e um vídeo demonstrando que as mudanças foram adotadas. Vídeo de até cinco minutos. Importante adotar padrões, construa um *template* para que seus colegas possam laudar a qualidade. Ele precisa ter no mínimo:

- Nome de quem testou;
- Data do teste;
- O que testou e funcionou (descrição da funcionalidade);
- O que testou e não funcionou (descrição da funcionalidade e o que deve ser corrigido);
- Funcionalidade não testada ou por que faltou ou a funcionalidade existe no projeto mas não foi realizada (descrever).

É importante que você não esconda essas falhas de maneira alguma, elas servem para seu aprendizado e para deixar na "mesma página" a sua visão em comparação com a do cliente. O que vale é ter os documentos dos laudos e o filme de que você trabalhou para fazer as melhorias.

Atividade de Entrega

Muito bem, estudante.

Agora que você já leu todas as situações-problema, você pode fazer o *download* [deste arquivo](#) para realizar a atividade de entrega.


Caso prefira, o arquivo também se encontra no Ambiente Virtual de Aprendizagem.

Referências

PRESSMAN, R. S. **Engenharia de Software**: uma abordagem profissional. 7. ed. Porto Alegre: Bookman, 2011.

PRODUCTBOARD. **Produto mínimo viável (MVP)**. 2021. Disponível em: <<https://www.productboard.com/glossary/minimum-viable-product-mvp>>. Acesso em: 28/12/2021.

WEBFLOW. **O que é desenvolvimento sem código?**. 2021. Disponível em: <<https://webflow.com/no-code>>. Acesso em: 28/12/2021.

 Muito bem, estudante! Você concluiu o material de estudos! Agora, volte ao Ambiente Virtual de Aprendizagem para realizar a Atividade.