

Projeto Integrador Transdisciplinar em Engenharia de Software II



Conteudista: Prof. Me. Artur Marques

Revisão Textual: Prof.^a M.^a Thassiana Reis Félix

☰ 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.

Introdução

Apesar de a área de programação ser considerada fundamental para o desenvolvimento profissional na Engenharia de *Software*, há quem ainda possui pouca experiência no assunto. Assim, por acreditarmos que o contato com este Projeto Integrador gere valor para tanto para a vida pessoal quanto profissional de seus usuários, neste material, você encontrará tanto uma retomada de conteúdo, focado em certos princípios utilizados na Engenharia da Computação, quanto a apresentação de novos, capazes de auxiliar até mesmo aqueles com pouca experiência na

manipulação e consecução de artefatos de *software*. Para isso, daremos foco em itens importantes na produção de uma solução, como, por exemplo, UX – experiência do usuário, projeto físico do banco de dados, codificação de *front-end* e padrões de *design* para o *back-end* e, claro, teste de *software*, além de outras coisas.

O desenvolvimento de *networking* durante o curso e, claro, durante toda sua jornada profissional, é fundamental para um projeto de vida bem-sucedido. Desse modo, é importante que você, estudante, desenvolva-o durante a execução do projeto. Para que isso aconteça, você deve conversar com seus colegas, fazer novas amizades, trocar experiências, fazer e permitir receber sugestões. Lembre-se: as coisas ficam melhores quando há empatia e colaboração.

Para este primeiro momento, que compreende a retomada de conteúdo, uma atividade fundamental é revisar o projeto proposto na fase de planejamento, realizado no Projeto Integrador Transdisciplinar em Engenharia de *Software* I, pois trata-se de uma oportunidade para recordar o que foi planejado, revalidar a ideia, entender ou aprofundar os estudos. Afinal, estudante, de lá para cá, muita coisa em sua cabeça deve ter mudado, incluindo a aprendizagem de disciplinas novas e técnicas que podem ter acrescentado valor a este seu trabalho. Também é possível que, a partir de reflexões, você tenha mudado de ideia, porque agora vê, de forma mais clara, os desafios que se apresentam e, por isso, quer revisá-los. Aproveite para fazer isso o mais cedo possível. Aliás, o momento é agora!

Vamos começar a retomada de conteúdos com o banco de dados. Um banco de dados tem informações que são representativas do mundo real. Quando modelamos aspectos desse mundo, descobrimos que precisamos representar algo, seja ele físico, um fato ou um evento que acontece.

Dentro desse banco, acontece a modelagem de dados de um *software* – ou seja, o processo de criar uma visão conceitual das informações que o banco contém ou deveria conter, garantindo, assim, a integridade dos dados. Como resultado desse processo, os objetos de dados, as entidades para as quais as informações devem ser armazenadas e as associações, regras ou restrições que regem as informações criam formas e entram no banco de dados. Assim, um modelo de dados é essencial para fornecer clareza e consistência nos metadados – ou seja, aos dados sobre outros dados – e nas definições que compõem um banco de dados, o que contribui para aumentar a qualidade da informação.

Também podemos garantir a qualidade das informações armazenadas em um banco de dados impondo regras para que apenas dados válidos entrem nas tabelas. Para fazer isso, ao projetar o modelo de dados, é necessário definir o domínio de valor para cada campo, diferenciando os que devem ter valores daqueles que podem ser deixados em branco. A qualidade de informação pode, ainda, ser aprimorada pela imposição de restrições que garantem a integridade referencial e mantêm a cardinalidade pretendida nas relações entre as entidades. Essas restrições podem ser derivadas apenas de um modelo de dados adequado. Assim, o desenvolvimento de *software* pode garantir um maior entendimento do sistema a ser desenvolvido se as atividades de modelagem de dados forem realizadas paralelamente ao levantamento de requisitos mais completos e mais corretos.

Como etapa posterior a modelagem de requisitos, iniciamos a construção dos artefatos de estruturais e comportamentais da *Unified Modeling Language (UML)* – Linguagem de Modelagem Universal.

Apesar dela possuir vários diagramas, usaremos o Diagrama de para criarmos as entidades que serão usadas para modelar qualquer “coisa” (coleções de objetos com as mesmas características intrínsecas como – funcionários, produtos, pedidos...) na empresa que devem ser representadas em um banco de dados. Mas, quais aspectos são relevantes em uma classe UML?

- **Nome:** o nome da classe é quase sempre um substantivo, pois representa uma coisa ou fato (alunos, carros, livros, cursos etc.);
- **Descrição:** o primeiro passo na modelagem de uma classe é descrevê-la em linguagem natural. Isso nos ajuda a saber exatamente o que essa classe significa no negócio;
- **Atributos:** cada classe é definida, exclusivamente, por seu conjunto de atributos, também chamados de propriedades. Cada atributo é uma informação que caracteriza cada membro dessa classe no banco de dados. Juntos, eles fornecem a estrutura para nossos objetos ou tabelas de banco de dados. Na UML, identificamos apenas atributos descritivos, ou seja, aqueles que realmente fornecem informações do mundo real relevantes para o negócio sobre os objetos que estão sendo modelados.

Um objeto é a instância de uma classe. Os métodos podem ser invocados em um objeto para inspecionar e manipular suas propriedades. Ao contrário dos sistemas orientados a objetos, os

bancos de dados relacionais tradicionais armazenam informações como registros de dados, não como objetos.

Na construção de um banco de dados relacional, cada classe é primeiro traduzida ou, se preferir, mapeada em um esquema de modelo relacional. Esse esquema é identificado pelo nome da classe e lista todos os atributos no diagrama de classes.

No modelo relacional, um esquema de relação é definido como um conjunto de atributos, juntamente com uma regra de atribuição, que associa cada atributo a um conjunto de valores válidos que podem ser atribuídos a ele. Esses valores são chamados de domínio do atributo. Lembre-se que um domínio é muito mais do que um tipo de dados.

Tudo que escrevemos está atrelado à teoria dos conjuntos. Assim, é importante reconhecer que definir esquemas ou domínios como conjuntos de elementos nos diz muito mais sobre eles, a partir das propriedades matemáticas dos conjuntos. Portanto:

- Um conjunto não pode conter elementos duplicados;
- Os elementos de um conjunto não são ordenados;
- Dado um conjunto, regras podem ser desenvolvidas para determinar quando um elemento pode ser incluído ou excluído dele;
- **Podemos definir subconjuntos:** por exemplo, podemos exibir apenas um conjunto selecionado de atributos de um esquema ou podemos limitar o domínio de um atributo a um intervalo específico de valores;
- **Os subconjuntos podem ser manipulados com os operadores usuais de conjunto:** união, interseção entre outros. Por exemplo, temos dois conjuntos de clientes, um pertencente a uma empresa, outro a uma segunda empresa. Se resolvermos realizar a interseção de ambos, teremos, provavelmente, uma região comum, ou seja, há clientes da empresa 1 que também são clientes da empresa 2.

Quando construímos o banco de dados (projeto físico), cada esquema de relação se torna a estrutura de uma tabela. A sintaxe para criar a tabela inclui um tipo de dado para cada atributo, que é necessário para o banco de dados, porém um tipo de dado não é o mesmo que o domínio do atributo.

A seguir, apresento um quadro comparativo sobre Entidades de relacionamento *versus* Diagramas de classes, assim você consegue, em caso de necessidade, consultar algo rápido e direto.

Quadro 1 – Comparação entre DER e Diagrama de Classes (semelhanças e diferenças)

Diagramas de Entidade de Relacionamento	Diagramas de Classe em UML
Entidades que representam objetos ou tabelas em banco de dados relacional	Classes são equivalentes a uma entidade no mundo relacional
Atributos de entidades, incluindo tipo de dados	Atributos de uma classe têm o mesmo significado em um DER, incluindo o tipo de dados
–	Métodos associados a uma classe específica, representando seu comportamento, ou seja, no mundo relacional, seriam procedimentos (<i>procedures</i>) armazenados

Diagramas de Entidade de Relacionamento	Diagramas de Classe em UML
Relacionamentos entre entidades/objetos ou chaves estrangeiras em um banco de dados	<p>Relacionamentos são aqui agrupados em duas categorias:</p> <ul style="list-style-type: none"> • Relacionamentos entre objetos: nesse caso, instâncias de Classes diferenciados em Dependência, Associação, Agregação e Composição são equivalentes a tipos de relacionamentos em um DER; • Relações entre classes de dois tipos: Generalização/Herança e Realização/Implementação. Este último não tem equivalente no mundo relacional.

Vídeo

Designing Databases from UML Class Models

Vale o investimento e tempo para assistir a este vídeo muito interessante sobre como transformar diagramas de classe em banco de dados.

Como o vídeo está em inglês, apesar de ser a língua de tecnologia de informação no mundo, você poderá, por conveniência, ativar as legendas

do *Youtube* e depois mudar o idioma para Português do Brasil. Isso dará mais conforto para você acompanhar o vídeo.

Clique no botão para conferir o vídeo indicado.

ASSISTA

Mas se desconsiderarmos a modelagem de dados, que riscos incorremos?

- **Redundância desnecessária:** como não há um modelo para ver claramente os objetos de dados, diferentes versões dos mesmos objetos aparecerão com informações diferentes;
- **Aplicativos lentos:** a ausência de um modelo de dados dificulta as tarefas de otimização, o que reduz a capacidade de resposta dos aplicativos;
- **Incapacidade de atender aos padrões de qualidade:** se não houver modelo de dados, seu banco de dados não será documentado, o que é obrigatório em cenários como migrações de banco de dados;
- **Qualidade de *software* ruim:** os requisitos de desenvolvimento de *software* serão ruins e os usuários não terão os aplicativos de que precisam ou desejam;
- **Custos de desenvolvimento mais altos:** teremos que decidir quem pagará pelos custos extras de desenvolvimento e manutenção, além de quem vai explicar quando os prazos não forem cumpridos.


Outro aspecto relevante que temos que recordar diz respeito à camada de apresentação do sistema ou IHC – Interface Humano Computador. Uma boa interface torna mais fácil para os usuários dizer ao computador o que eles querem fazer. Assim, o computador pode solicitar informações dos usuários

e apresentar informações compreensíveis. Nesse sentido, a comunicação clara entre o usuário e o computador é a premissa de trabalho de um bom *design* de interface do usuário. A seguir, trago alguns itens importantes para você, estudante, levar em consideração quando for fazer a IHC de sua aplicação de *software*.

A sua interface:

- **Deve ser clara:** a clareza ajuda a evitar erros do usuário, torna as informações importantes óbvias e contribui para facilitar o aprendizado e o uso;
- **Deve ser consistente:** uma interface consistente permite que os usuários apliquem o conhecimento já aprendido a novas tarefas;
- **Deve ser simples:** *designs* simples são fáceis de aprender e usar e dão à interface uma aparência consistente. Um bom *design* requer equilíbrio entre maximizar a funcionalidade e manter a simplicidade por meio da divulgação progressiva de informações;
- **Deve ser controlada pelo usuário:** o usuário, não o computador, inicia e controla todas as ações;
- **Deve ser direta:** os usuários devem ver a relação visível de causa e efeito entre as ações que realizam e os objetos na tela. Isso os permite sentir que estão no comando das atividades do computador;
- **Deve ser empática e recuperável/reversível:** os usuários cometem erros. Portanto, é importante que as ações realizadas sejam reversíveis. Uma boa interface facilita a exploração e o aprendizado por tentativa e erro;
- **Deve dar uma resposta:** é importante manter o usuário informado e fornecer *feedback* imediato. Além disso, certifique-se de que o *feedback* seja apropriado para a tarefa. Lembre-se: boa mensagem, bom tratamento;
- **Deve ser esteticamente agradável:** cada elemento visual que aparece na tela, potencialmente, compete pela atenção do usuário. Assim, proporcionar um ambiente agradável para trabalhar e que contribua para a compreensão do usuário sobre as informações apresentadas é fundamental.

Há um excerto importante e pertinente às atividades que serão realizadas por você. O exponho a seguir:



"O *design* da interface do usuário (UI) concentra-se em antecipar o que os usuários podem precisar fazer e garantir que a interface tenha elementos fáceis de acessar, entender e usar para facilitar essas ações. A interface do usuário reúne conceitos de *design* de interação, *design* visual e arquitetura da informação. Escolhendo elementos de interface: Os usuários se familiarizaram com os elementos da interface que atuam de uma determinada maneira, portanto, tente ser consistente e previsível em suas escolhas e no *layout*. Isso ajudará na conclusão da tarefa, eficiência e satisfação. Os elementos de interface incluem, mas não estão limitados a:

- **Controles de entrada:** botões, campos de texto, caixas de seleção, botões de opção, listas suspensas, caixas de listagem, alternâncias, campo de data;
- **Componentes de navegação:** trilha de navegação, controle deslizante, campo de pesquisa, paginação, controle deslizante, *tags*, ícones;
- **Componentes informativos:** dicas de ferramentas, ícones, barra de progresso, notificações, caixas de mensagens, janelas modais;
- **Recipientes.**

Há momentos em que vários elementos podem ser apropriados para exibir conteúdo. Quando isso acontece, é importante considerar os *trade-offs*. Por exemplo, às vezes, os elementos que podem ajudar a economizar espaço sobrecarregam mais o usuário mentalmente, forçando-o a adivinhar o que está dentro do menu suspenso ou qual pode ser o elemento.

Práticas recomendadas para projetar uma interface

Tudo decorre de conhecer seus usuários, incluindo a compreensão de seus objetivos, habilidades, preferências e tendências. Depois de conhecer seu usuário, certifique-se de considerar o seguinte ao projetar sua interface:

- **Mantenha a interface simples:** As melhores interfaces são quase invisíveis para o usuário. Eles evitam elementos desnecessários e são claros na linguagem que usam nos rótulos e nas mensagens;
- **Crie consistência e use elementos de interface do usuário comuns:** Ao usar elementos comuns em sua interface do usuário, os usuários se sentem mais à vontade e podem fazer as coisas mais rapidamente. Também é importante criar padrões de linguagem, *layout* e *design* em todo o *site* para ajudar a facilitar a eficiência. Uma vez que um usuário aprende como fazer algo, ele deve ser capaz de transferir essa habilidade para outras partes do *site*;
- **Seja objetivo no *layout* da página:** Considere as relações espaciais entre os itens na página e estruture a página com base na importância. A colocação cuidadosa de itens pode ajudar a chamar a atenção para as informações mais importantes e pode ajudar na digitalização e legibilidade;
- **Use cor e textura estrategicamente:** Você pode direcionar a atenção ou redirecionar a atenção para os itens usando cor, luz, contraste e textura a seu favor;
- **Use a tipografia para criar hierarquia e clareza:** Considere cuidadosamente como você usa o tipo de letra. Diferentes tamanhos, fontes e disposição do texto para ajudar a aumentar a legibilidade, a legibilidade e a legibilidade;
- **Certifique-se de que o sistema comunique o que está acontecendo:** Sempre informe seus usuários sobre localização, ações, mudanças de estado ou erros. O uso de vários elementos de interface do usuário para comunicar o *status* e, se necessário, as próximas etapas podem reduzir a frustração do usuário;
- **Pense nos padrões:** Ao pensar cuidadosamente e antecipar os objetivos que as pessoas trazem para o seu *site*, você pode criar padrões que reduzem a carga do usuário. Isso se torna particularmente importante quando se trata de *design* de formulários, onde você pode ter a oportunidade de ter alguns campos pré-escolhidos ou preenchidos.

Vídeos

Para conhecer melhor os aspectos ideais numa IHC, selecionei dois vídeos muito interessantes para que você possa incrementar a UX de seus clientes, deixando a experiência mais atraente e dentro de boas práticas. Veja-os a seguir.

Princípios de Norman, o Criador do Termo “*User Experience*”

Clique no botão para conferir o vídeo indicado.

ASSISTA

O que é Arquitetura de Informação?


Clique no botão para conferir o vídeo indicado.

ASSISTA

Outro aspecto importante que você deve ter em mente é que precisará programar e, muitas vezes, o tempo é curto. Portanto, quero, aqui, claro, permitir que você programe, de forma clássica, tanto as IHCs de sua aplicação, utilizando HTML, CSS e *JavaScript*, por exemplo, mas também o *back-end*, usando Java, PHP ou *Python*, se for o caso. Apesar disso, não há como deixar de indicar aceleradores de produtividade num novo e emergente grupo de ferramentas chamadas de *low-code* e *no-code*.

Codificar gera estresse na maioria das pessoas com pouca prática ou que não se dão muito bem quando precisam codificar algo. Mas não é o “fim da linha”. Lembre-se que você está nessa jornada para enfrentar desafios e somente dessa forma ela poderá ser gratificante. Saímos desses processos sempre maiores do que quando entramos, com mais conhecimento e experiência principalmente. Portanto, precisamos mudar a nossa perspectiva, nosso *mindset*. Por isso, vamos quebrar esse paradigma iniciando com possibilidades de realização de aplicativos por meio do emprego do *no-code* e do *low-code*.

Esse grupo de ferramentas trata-se de um movimento importante, pois mais do que permitir que usuários tenham empoderamento para aplicações baseadas em *web*, ao pensar em soluções simples e elegantes, o *no-code* pode ser a solução. O *no-code* para *web* trata de *JavaScript*, HTML5 e CSS, mas não exige, em um primeiro momento, um conhecimento profundo inicial, o que o faz torná-lo a solução ideal para você que está desenvolvendo um projeto. Veja alguns conceitos.



“O desenvolvimento sem código é um tipo de desenvolvimento para a *Web* que permite que não programadores e programadores criem *software* 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."

Assim, o *no-code* não se trata de um atalho, mas de 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 do que dinheiro, representa sobrevivência.

Com a digitalização das empresas e do poder computacional existente atualmente, um simples programador, em algum canto obscuro do planeta, pode estar fazendo a diferença ou na cura de doenças, ou criando acessibilidade e inclusão por meio de seus aplicativos e serviços, ou criando empresas cujo valor de mercado ultrapassa o PIB – Produto Interno Bruto – de muitas nações. Como exemplo, podemos citar a *Apple*, a *Tesla*, o *Facebook*, o *Google*, a *Microsoft*, a *Amazon*, a *Alibaba* entre outras. Assim, quanto mais tecnologia você tiver ao seu alcance e mais souber sobre ela, 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 – o que permite 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. Portanto, você já pensou em rever seus protótipos do Projeto Integrador Transdisciplinar I usando *no-code*?!

É claro que optar pelo *no-code* não fará 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. Assim, as plataformas sem código exigem programadores, ou seja, pessoas que podem pensar em abstrações e, acima de tudo, que sabem como unir as ferramentas certas de maneira correta para produzir valor.

Sites

A seguir, deixo algumas plataformas para você acessar e pesquisar para construir a sua:

Appy Pie – App Builder

Clique no botão para conferir o conteúdo.

ACESSE

Appy Pie – Website Builder

Clique no botão para conferir o conteúdo.

ACESSE

Appy Pie – Chatbot Builder

Clique no botão para conferir o conteúdo.

ACESSE

Appy Pie – Connect

Clique no botão para conferir o conteúdo.

ACESSE

Voiceflow

Clique no botão para conferir o conteúdo.

ACESSE

Airtable

Clique no botão para conferir o conteúdo.

ACESSE

Ninox

Clique no botão para conferir o conteúdo.

ACESSE

VINYL

Clique no botão para conferir o conteúdo.

ACESSE

Quando estamos desenvolvendo uma aplicação, é importante que ela seja testada, afinal, o teste poderá determinar e garantir a qualidade do produto.

O sucesso desse teste depende de aspectos como a definição do que constitui um produto de qualidade, a determinação de propriedades mensuráveis que refletem a qualidade, a derivação de critérios significativos de teste baseados nas quantidades mensuráveis e a 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

Vídeo

Webinar Requisitos não Funcionais

Vamos revisar os requisitos de qualidade não funcionais. Aqui, você encontrará um conteúdo muito bom.

Clique no botão para conferir o vídeo indicado.

ASSISTA

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, como o próprio nome sugere, refere-se à verificação do aplicativo que ainda está em desenvolvimento e acontece para garantir que o *software* se encontra de acordo com as 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 com o produto ainda 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.

Assim, o esforço de validação não importa com o 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: a 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, descrevo aqui que a validação de requisitos, independente da metodologia utilizada, 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. Logo, a validação é uma etapa crítica para encontrar requisitos ausentes e garantir que eles tenham uma variedade de características importantes como:

- 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.

Por fim, entenda, estudante, que qualquer forma de validação em *software* (de requisitos ou de código) não está focada no caminho percorrido para chegar ao destino, mas sim, se o alvo foi atingido.

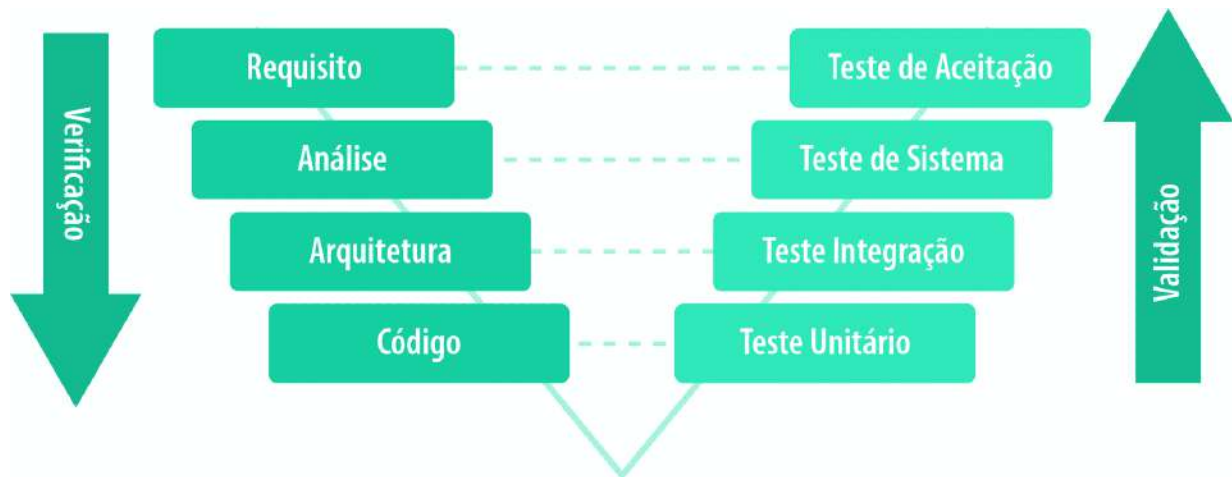


Figura 1 – Exemplo de relação entre o desenvolvimento e os testes com o Modelo V

Sites

Para te ajudar nos testes unitários, temos o seguinte:

JUnity

Clique no botão para conferir o conteúdo.

ACESSE

PhpUnity

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

Tutorial de pytest para iniciantes

Clique no botão para conferir o conteúdo.

ACESSE

Por fim, recomendo que você, estudante, utilize uma construção em camadas com um padrão *MVC – Model – View – Controller*. Apesar de ser, atualmente, primitivo em certos aspectos, com esse padrão você poderá aprender, de forma clara, a importância de adoção. Afinal, o padrão por trás de cada tela que usamos é o MVC, inventado quando não havia *Web* e as arquiteturas de *softwares* eram, na melhor das hipóteses, clientes conversando diretamente com um único banco de dados em redes primitivas. Pois é, a situação mudou muito de lá para cá, não é mesmo?!

Porém não custa lembrar que hoje o MVC é, para todos os efeitos, a pedra angular de qualquer arquitetura de aplicativo, pois, em sua abordagem, separa as principais preocupações comuns para o código de aplicativo organizado e gerenciável. A base de um aplicativo MVC bem projetado é um modelo de dados sólido.

- **Modelo:** o modelo de dados representa as informações principais que seu aplicativo usa para acessar e manipular. Por isso, é considerado o centro da sua aplicação. Enquanto o visualizador e o controlador servem para conectar o usuário com o modelo de dados de forma amigável, o modelo encapsula as preocupações de armazenamento e validação;
- **View:** os visualizadores devem monitorar o modelo de dados, pronto para responder às mudanças. Isso permite que os controladores evitem acoplamentos desnecessários ao visualizador. O controlador deve atualizar o modelo e, em seguida, o visualizador observará e responderá a essa alteração;
- **Controlador:** o código do controlador atua como uma ligação entre o modelo e a visualização, recebendo a entrada do usuário e decidindo o que fazer com ela. Por ter essas funções, é considerado o cérebro do aplicativo.

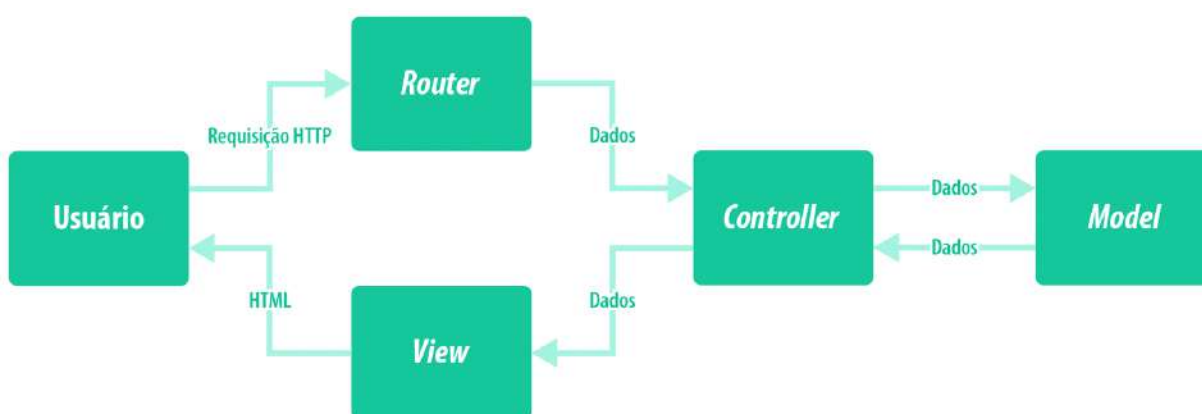


Figura 2 – Exemplo de MVC em aplicação web

Há, inclusive, estudante, um repositório no *Github*. Oriento que você o estude para entender melhor o funcionamento de um aplicativo. Nesse repositório, há a codificação funcional e a documentada.

Leitura

EJS e Sequelize Juntos, no Padrão MVC!

Clique no botão para conferir o conteúdo.

ACESSE

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta disciplina:

VÍDEOS

TDD NA PRÁTICA COM JAVA USANDO @MOCKBEAN

O desenvolvimento orientado por testes é de grande valia no desenvolvimento ágil. Pode ser utilizado junto com XP, em uma gestão de projeto de *software* SCRUM. No vídeo, são explicadas técnicas de como desenvolver testes, conforme o exigido em desenvolvimento na prática de forma introdutória.

Clique no botão para conferir o vídeo indicado.

ASSISTA

CRIANDO UM CRUD FÁCIL USANDO O FRAMEWORK SPRING MVC

No vídeo, você encontrará algo além da criação de um CRUD, usando o *Spring MVC Spring JPA* e *Spring Boot*; você descobrirá a forma correta e possível de utilização de projetos *Java Maven* externos e reaproveitamento de modelos já criados, provando, dessa forma, como é possível programar em Java de modo correto e reaproveitar o código Java.

Clique no botão para conferir o vídeo indicado.

ASSISTA

CONHEÇA O *SPRING MVC*, O *FRAMEWORK WEB* MAIS POPULAR PARA JAVA

Com este vídeo, você aprenderá sobre como funciona o fluxo de uma requisição atendida pelo *Spring MVC*, e ainda, quais as principais vantagens deste *framework web*.

Clique no botão para conferir o vídeo indicado.

ASSISTA

LEITURA

JUNIT TUTORIAL

O artigo aborda um assunto vital para programadores e analistas que trabalham sobre um paradigma ágil, como o *Extreme Programming*. De forma sucinta, o material apresenta tópicos como: Um pouco de XP, Como programar guiado a testes? Teste Unitário (O quê? Por quê? Quando? Quem? Como?), *JUnit* (O quê? Por quê? Quando? Quem? Como?), *JUnit* (Planejamento e arquitetura das classes), *JUnit* (Funcionamento e Análise do resultado), Implementado testes em *JUnit* usando o Eclipse e outros métodos e técnicas complementares.

<https://bitly/3kb3Bek>

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.

Atualizar Plano de Projeto e Desenvolver/Testar uma Solução Funcional de *Software*

Situação 1

Como descrito no corpo teórico e na apresentação desta Unidade, é importante aproveitar esse tempo para revisar o planejamento.

O *KaiZen* (melhoria contínua) é de fundamental importância para a criação de produtos de *software* melhores e mais adequados. Para podermos potencializar a solução planejada no Projeto Integrador Transdisciplinar de Engenharia de *Software* I, embarcar mais detalhes e corrigir conceitos quando aplicável, pedimos que você, estudante:

- Verifique e adeque seu escopo/ideia e a prepare para a execução;

- Verifique e adeque os elementos de modelagens da solução feitos em UML (diagramas de classe, casos de uso, sequência, atividades etc.), veja onde podemos colocar melhorias e alterações necessárias visando acurácia da modelagem;
- Verifique e adeque os elementos da IHC (Interface Humano Computador) para poder refletir o que você pretende desenvolver em nível de telas, mensagens de erro, mapas conceituais, além de revisar o próprio protótipo incrementando-o;
- Faça ou revise os projetos relacionados aos dados a partir do diagrama de classes de persistência (projeto conceitual, projeto lógico normalizado);
- Execute o projeto físico do banco de dados que servirá a sua aplicação (*web* ou *mobile*), documente-o e crie o dicionário de dados.

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.

Situação 2

Nesta situação-desafio, você irá desenvolver o que planejou. Para poder auxiliá-lo(a), oferecemos algumas ferramentas livres na documentação de conteúdo do Projeto Integrador Transdisciplinar II, para que você acelere, caso queira, afinal se temos possibilidades de uso de *no-code* ou *low-code*, como ferramentas para ajudar na produtividade do *front*, vamos utilizar. Claro, você pode realizar a atividade de forma tradicional: a escolha e o tempo são seus. Aqui, esperamos as seguintes entregas:

- Crie um usuário no *GIT* para que sua experiência possa ser revisada e vista pelos tutores e professores. Claro, ela servirá de projeto realizado para que selecionadores e outros possam certificar sua experiência e ficar competitivo na concorrência por vagas. Sugerimos que você continue publicando nesse repositório após a entrega do Projeto Integrador Transdisciplinar II. Será seu CV permanente;
-

Site

Github

Clique no botão para conferir o conteúdo.

ACESSE

- Realizar a construção do código para a interface do usuário baseada em HTML, CSS, *Java Script*, conforme a necessidade de seu projeto para o *front-end*;
- Desenvolver o código de processamento de dados, ou seja, o *back-end* dessa aplicação utilizando o *design pattern*, indicado na documentação desta disciplina, (é indicado MVC);
- Utilizar uma das seguintes linguagens sugeridas: Java, PHP, C#, .NET ou *Python*, o que você estiver melhor familiarizado(a);
- Apresentar *front-end* e *back-end* testados (por exemplo, *JUnit*, *PUnit*, *PHPUnit* etc.) e funcionais, com erros tratados e completos.

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.

Situação 3

Agora, estaremos concentrados em correções, testes de integração, e validação da solução utilizando pares. Além disso, precisamos fechar a documentação de todo o projeto para uma entrega perfeita. Assim, na situação 3, esperamos as seguintes entregas:

- Realizar e documentar os testes (verificação e validação) mostrando que o comportamento esperado foi realizado;
- Oferecer a sua solução para cinco colegas de curso ou profissionais da área para que possam testá-la. Para isso, você, estudante, deve criar um formulário para que possa ser descritos os *bugs*, sugestões de melhoria e colocar *snapshots* das telas com erros para facilitar sua correção e melhoria;
- Coletar as sugestões dadas, realizar correções e melhorias sugeridas (quando pertinente) e incorporá-las na documentação do projeto;
- Manter tudo documentado, criando estruturas de diretório no *GIT* para todas as situações de desafio além, é claro, da codificação;

- Apresentar, em um vídeo gravado de até cinco minutos, sua solução em funcionamento.

Problema em Foco

Orientações para o Projeto:

Situação 1

- Revisitar e revisar todos os documentos criados por você em seu Projeto Integrador Transdisciplinar de Sistemas de Informação I;
- Atentar-se e ter a intenção de realizar um ciclo de melhoria contínua, deixando a documentação de planejamento mais robusta e mais clara;
- Utilizar competências de atenção aos detalhes, abstração e *feedback* de seus colegas. Para isso, você, estudante, deve pedir a eles a crítica para a solução que você deve implementar. Assim, o poder da colaboração e empatia deve ser utilizado nesta atividade.

Uma vez revisado os documentos, escolhidas as melhores sugestões de seus colegas e, claro, as suas próprias sugestões, compile e utilize o *GIT* para manter a documentação atualizada e num lugar só, isso inclui o código. Lembre-se de criar uma estrutura de banco de dados com um SGBD que seja de fácil manutenção e popular. Nada de usar SGBD obscuros: isso não é boa prática de mercado.

Situação 2

Fazer a codificação do *front-end*, do *back-end* e da aplicação *web* ou *mobile*, conforme sugerido em seu planejamento. Bem, esperamos muita codificação de sua parte para que o código, ou artefato, ou análise de dados possam ser desenvolvidos e apresentados.

Queremos que você produza códigos fontes *front-end* e *back-end*, bibliotecas, manual de uso, vídeo gravado demonstrando o funcionamento – narrado por você com, pelo menos, cinco minutos de duração. Concentre tudo no *GIT*, passe o *link* de acesso para o(a) Tutor(a) ou deposite o material no local que ele(a) pedir, mas mantenha o *GIT* em qualquer situação.

Lembre-se: para ajudar no *front-end*, ferramentas *no-code* e/ou *low-code* podem auxiliar muito na economia e em um melhor aproveitamento do tempo no projeto.

Aproveite para revisar a interface do usuário, o *design* e outros atributos. Verifique se mudará algo, aproveite para fazer essas modificações quando necessário. Deixe a documentação sempre em ordem. Você a depositará junto com o restante do projeto, conforme orientações do(a) professor(a)-tutor(a), no momento oportuno.

Situação 3

Na situação 3, o desafio é a realização de testes feitos pelos clientes – no caso, os colegas que julgaram o que você desenvolveu. É importante que, nesse momento, você já tenha escolhido, entre seus colegas, quais serão seus **dublês** de usuários e lhes forneça o endereço para que eles possam testar sua solução como só um usuário sabe fazer.

O importante aqui é desenvolver a capacidade de aceitação da mudança, o reconhecimento do erro como forma de melhorar continuamente, o trabalho em equipe e, por fim, a resiliência necessária para continuar a perseverar para atingir o resultado.

É importante que você entregue, em um arquivo PDF, as cinco opiniões/testes sobre seu *software*/produto, além de um vídeo que explicita quais mudanças foram adotadas. Esses materiais consistirão como a prova de seu sucesso, estudante, neste desafio. Lembre-se de construir um *template* para que seus colegas possam laudar a qualidade da criação. Esse documento 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:** por que faltou ou se a funcionalidade existe no projeto, mas não foi realizada (descrever).

Sinta-se à vontade para criar sobre isso.

Tudo que você precisa entregar está descrito no próprio desafio para evitar dúvidas. É importante que você não esconda essas falhas de maneira alguma, afinal, elas servem para seu aprendizado.

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

USABILITY.GOV. *User interface design basics*. 30/09/2020. Disponível em: <<https://www.usability.gov/what-and-why/user-interface-design.html>>. Acesso em: 02/02/2022.

WEBFLOW. *What is no-code development?*. c2022. Disponível em: <<https://webflow.com/no-code>>. Acesso em: 02/02/2022.

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