

Projeto de *Software* com Métodos Ágeis



Cruzeiro do Sul Virtual
Educação a distância

Material Teórico



Parte C

Responsável pelo Conteúdo:

Prof. Me. Artur Marques

Revisão Textual:

Prof.^a Esp. Kelciane da Rocha Campos



- **Introdução;**
- **Design e Arquitetura da Solução;**
- **Priorização de Sprints;**
- **Criação do Sprint Backlog;**
- **Desenvolvimento de Artefatos de Sistema Baseados em UML.**



OBJETIVO DE APRENDIZADO

- Desenvolver na prática os artefatos UML de um projeto ágil.



Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:



Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

Introdução

Já vimos como o Trello é útil para conformarmos o arranjo de nossas histórias do usuário, criar *backbones* de histórias compondo a jornada do usuário, arranjar os cartões de tarefas/atividades dos desenvolvedores e obviamente arranjar as *sprints* para priorização e puxar os *cards* para a própria *sprint*.

Além disso, é um facilitador de comunicação para o time ágil e um organizador visual para vermos *cards* novos acrescentados para a *sprint*, *bugs* e outros tipos de *cards* necessários para o sistema. Sim, dá para fazer porque o Trello permite utilizar sistemas de cores.

Lembre-se: o Trello é gratuito, mas há versões pagas com recursos poderosos para um time profissional em uma empresa, assim como podemos utilizar o Trello e o *Jira Service Desk* para parte de *workflow* ágil e integrar com os repositórios de código coletivo num *bit bucket/github* ou ampliar a colaboração com o *Confluence*. Portanto, você tem uma suíte ágil completa para seus projetos.

Você pode acessar e fazer uma autocapacitação, sua carreira irá agradecer muito. Então, vamos lá; siga os *links* dos treinamentos:



Tutorial Trello – Como usar o Trello do ZERO. Disponível em: https://youtu.be/0Uqf_DLbAUU

Design e Arquitetura da Solução

Arquitetura e *design* de *software* são o esqueleto de um sistema. Eles definem como o sistema deve se comportar em termos de diferentes requisitos funcionais e não funcionais. Atualmente, não existe uma especificação clara das atividades e processos de *design* de arquitetura de *software* em ambientes ágeis.

Além disso, Arquitetura Ágil é um conjunto de valores, práticas e colaborações que apoiam o projeto e a arquitetura ativas e evolutivas de um sistema. Essa abordagem adota normalmente a mentalidade DevOps, permitindo que a arquitetura de um sistema evolua continuamente ao longo do tempo, ao mesmo tempo em que oferece suporte às necessidades dos usuários atuais. Não há consenso sobre isso, apenas diversos caminhos; você, com a prática, precisará escolher a sua abordagem ou forma de resolver problemas.

Conforme os escritos de Mihaylov (2015, p. 1), “nem todas as empresas têm uma função designada de arquiteto de softwares; muitas empresas delegam essas responsabilidades a seus desenvolvedores sênior”.

Um arquiteto de *software* tradicional tem quatro recursos principais:

- **Concentre-se no panorama geral:** um arquiteto de *software* deve considerar como o sistema ficaria meses no futuro (às vezes até anos). Além disso, ele deve considerar todos os outros sistemas (por exemplo,

quaisquer 3 ou 4 sistemas partidários e armazenamentos de dados) envolvidos e como a comunicação entre eles vai acontecer;

- **Orientado para conformidade:** um arquiteto de software deve considerar possíveis problemas de conformidade. Podem ser normas legislativas, licenças, padrões ou outros, mas ele deve garantir que o sistema futuro atenderá a esses critérios importantes;
- **Produz projetos:** uma entrega importante é uma coleção de documentos e diagramas que descrevem a arquitetura de diferentes perspectivas. A equipe de desenvolvimento usa essas entregas para começar a construir o sistema;
- **Pouca experiência prática:** o objetivo do arquiteto de software é produzir os documentos finais para a equipe de desenvolvimento. Embora o arquiteto de software geralmente tenha experiência como desenvolvedor, ele raramente está envolvido no processo de desenvolvimento – ele é o mentor dos desenvolvedores para construir o sistema projetado. Em algum momento, ele pode até mesmo mudar para outro projeto e deixar os desenvolvedores por conta própria. (MIHAYLOV, 2015, p. 2)

Para converter a função tradicional do arquiteto de software em uma que se encaixe no mundo ágil, temos que dar uma olhada em algumas variantes possíveis, e colocar o arquiteto de software diretamente na equipe de desenvolvimento é uma delas.

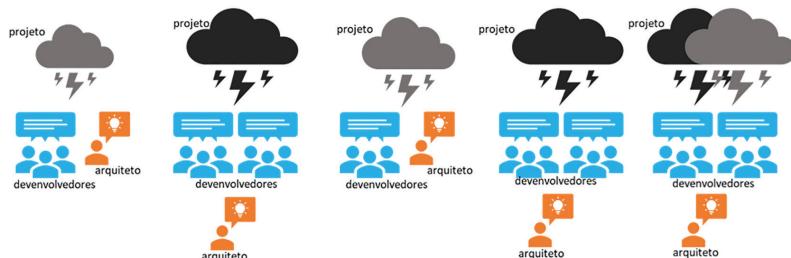


Figura 1 – Cada equipe de desenvolvimento tem um arquiteto de software

Métodos ágeis e arquitetura de software não são incompatíveis, mas complementares

A arquitetura Agile oferece suporte às práticas de desenvolvimento Agile por meio de colaboração, design emergente, arquitetura intencional e simplicidade de design, assim como as práticas de desenvolvimento e arquitetura permitem o design para testabilidade, implementação e liberação, apoiados por prototipagem rápida, modelagem de domínio e inovação descentralizada.

Conforme os escritos anteriores nesse texto, podemos perceber que o arquiteto de software ágil recebe responsabilidades um pouco diferentes do tradicional:

- **Equilíbrios entre o panorama geral e o agora:** um arquiteto de software ágil precisa pensar sobre o que está acontecendo durante o desenvolvimento e alinhá-lo com o panorama geral do sistema;
- **Experiência prática:** um arquiteto de software ágil também é desenvolvedor e trabalha na implementação do sistema. Isso fornece feedback em primeira mão sobre as decisões arquitetônicas tomadas;

- **Produz protótipos para tomar decisões informadas:** quando uma decisão técnica importante precisa ser feita, um protótipo rápido pode revelar se essa decisão é viável e como ela afetaria o sistema existente. Além disso, a comunicação com toda a equipe de desenvolvimento é essencial, pois é um esforço colaborativo, ao invés de uma atividade de um único homem;
- **Foco na sustentabilidade:** é extremamente importante que as decisões arquitetônicas levem a uma arquitetura de software sustentável – que irá apoiar o projeto em longo prazo. Uma parte essencial disso é a responsabilidade pessoal e empatia. O arquiteto de software ágil faz parte da equipe de desenvolvimento, portanto, ele obtém *feedback* em primeira mão por meio de suas decisões, conforme descrito acima. Isso aumenta o nível de responsabilidade pessoal em contraste com a abordagem em cascata, onde as decisões são transferidas para a equipe de desenvolvimento implementar.

A metodologia ágil coloca o foco na colaboração e, ao separar os arquitetos de *software* dos desenvolvedores, você torna a colaboração difícil. Por isso você deverá colocar o arquiteto de *software* presente em cada equipe de desenvolvimento, pois a comunicação entre os membros da equipe melhora. (MIHAYLOV, 2015, p. 4)

Então, vamos manter tudo simples; se você criou um artefato visual para descrever um certo aspecto do sistema, não há necessidade de criar um documento de texto extenso descrevendo a mesma coisa com palavras, a menos que você queira adicionar detalhes que você não pode expressar visualmente.

Não tenha um segundo de dúvida sobre o que usar para gerar isso – sim, o mundo utiliza em ágil UML; é um padrão, todos sabem disso, as universidades o ensinam, as empresas o utilizam e, portanto, é a ferramenta para unir todos na organização.

Se você possui dificuldade em utilizar isso, bem, melhor focar nesse reforço. É o dia a dia, além da codificação, é claro.

Além disso, claro, você pode usar quadro branco, notas *Post-It*, documentos de texto, *wikis* e o que mais sua imaginação, tempo e recursos puderem proporcionar.

Assim, fica entendido que para nós, a arquitetura de *software* é uma forma de pensar sobre sistemas de computação – por exemplo, sua configuração e *design*.

Praticamente, o que estamos fazendo até aqui em todas as unidades é a abordagem de aspectos arquitetônicos de um *software* e ele deverá cumprir os seguintes objetivos:

- **descrever uma mudança necessária nos componentes de uma arquitetura:** Isso pode significar adicionar novos componentes, remover os desatualizados, substituir ou melhorar os componentes ou mudar a maneira como são organizados e como funcionam juntos;
- **incluir o raciocínio ou motivações por trás da mudança:** Por que isso precisa mudar? Deve explicar por que os componentes existentes são inadequados, limitantes ou restritivos em caso de uma evolução da aplicação;
- descrever as opções disponíveis para arquiteturas futuras que abordem todas as questões;

- explicar os benefícios, valor, riscos, custos, oportunidades, restrições e opções futuras associadas a cada alternativa;
- descrever quaisquer rotas alternativas para fechar as lacunas e ir da arquitetura atual para a arquitetura de destino.

Quando definimos o formato, levamos em consideração 3 importantes pilares associados:

- **Elementos:** Existem três classes de elementos de *software*, a saber, elementos de processamento, elementos de dados e elementos de conexão. Os elementos de processamento são aqueles componentes que pegam alguns dados e aplicam transformações neles, e podem gerar dados novos ou atualizados. Os elementos de dados são aqueles que contêm as informações a serem utilizadas, transformadas e manipuladas. Os elementos de conexão unem a descrição arquitetônica ao fornecer *links* de comunicação entre outros componentes. Os próprios elementos de conexão podem ser elementos de processamento ou de dados, por exemplo, chamadas de procedimento, dados compartilhados ou mensagens;
- **Formulários:** a forma arquitetônica consiste em propriedades e relacionamentos ponderados. A definição implica que cada componente da arquitetura seria caracterizado por algumas restrições, geralmente decididas pelo arquiteto, e algum tipo de relacionamento com um ou mais outros componentes. As propriedades definem as restrições nos elementos do *software* no grau desejado pelo arquiteto;
- **Racional:** a lógica explica as diferentes decisões e escolhas arquitetônicas; por exemplo, porque um determinado estilo, elemento ou forma arquitetônica foi escolhido. A lógica está ligada aos requisitos, visões arquitetônicas e partes interessadas. Provavelmente, todas as escolhas são regidas por quais são os requisitos. Existem muitos componentes externos diferentes que têm interesse no sistema e esperam coisas diferentes do mesmo sistema. Portanto, temos que considerar as diferentes demandas externas e expectativas que afetam e influenciam a arquitetura e sua evolução. (MEKNI; BUDDHAVARAPU; CHINTHAPATLA; GANGULA, p. 5)

Isso significa que devemos pensar em criar um documento básico para um documento abrangente de arquitetura de *software* baseada em visualização usando os esquemas de organização padrão; além disso, devemos decidir quais visões arquiteturais devem ser produzidas, dado o escopo da arquitetura de *software* com relação aos recursos disponíveis. Este esboço deverá chegar ao time de desenvolvimento, ao dono do produto e, claro, às partes interessadas, onde você minuciosamente colocará os benefícios que a arquitetura possui. Esse negócio de fazer arquitetura de *software* de maneira solitária, sozinho, é o caminho para o erro e fracasso. Ágil é forte porque é um esforço coletivo consciente que nos leva para a construção de *software* de alto valor.

A metodologia de arquitetura de *software* proposta em ambientes ágeis permite que a arquitetura e o *design* do *software* suportem os recursos sem potencialmente criar retrabalho imprevisto por desestabilizar a refatoração.

A entrega da funcionalidade planejada é mais previsível quando a arquitetura para os novos recursos já está em vigor. Isso requer olhar para frente no processo de planejamento e investir em arquitetura, incluindo o trabalho de *design* na iteração atual, que oferecerá suporte aos recursos futuros e às necessidades do cliente. O refinamento arquitetônico não está completo.

O processo de refinamento intencionalmente não está completo devido a um futuro incerto com as mudanças nas orientações da tecnologia e na engenharia de requisitos.

Não entendeu?!

Vou deixar claro, a arquitetura em um projeto ágil é uma adaptação e descoberta contínua. Decerto que no início das primeiras *sprints* haverá muita adaptação e mudança, mas com o tempo a previsibilidade e a facilidade de desenvolvimento associada a um ritmo cada vez mais veloz e certo demonstrarão que a arquitetura é funcional.

As fases de engenharia de requisitos ou arquitetura de *software* não ocorrem apenas uma vez, mas sim são continuamente distribuídas ao longo do processo de desenvolvimento. Quando houver um primeiro conjunto de requisitos (geralmente incompleto) disponível, o arquiteto prossegue para o projeto arquitetônico.

Por fim, Mekni, Buddhavarapu, Chinthapatla e Gangula (2018, p. 11) escrevem que: “ser capaz de alternar rapidamente entre o problema a ser resolvido (os requisitos) e sua solução (a arquitetura) pode ajudar a distinguir mais claramente os dois e evitar misturar problema e solução já na fase de engenharia de requisitos”.

Priorização de *Sprints*

A priorização é um dos aspectos mais importantes de qualquer forma de trabalho de desenvolvimento porque escolher a coisa certa a fazer permite maximizar o valor entregue em um *Sprint*. Ter um entendimento comum das prioridades capacita uma equipe a se mover em uma direção uniforme visando a um objetivo comum.

Para ajudar, pense nesses itens quando for priorizar:

- preparar continuamente o seu *backlog* do produto (o *backlog* não é uma lista fechada, se fosse não seria um processo ágil, teríamos uma metodologia tradicional);
- aplicar os aprendizados da revisão *Sprint* (o que acontecer numa revisão de *sprint* deve servir como lição aprendida e crescimento do time, incorpore e mantenha o foco, pois errar uma vez é aprendizado, errar duas ou mais é escolha);
- ter um acordo de “divisão de trabalho” em vigor (eu chamo isso de “pacto de sangue” com a equipe; é algo do tipo “faremos isso e seremos fiéis e responsáveis quanto a isso”).

Tenha uma lista de perguntas que o time use para gerar valor e ajudar. Normalmente, o dono do produto já trará a lista priorizada do que quer e irá discutir com o time para executá-la. Momentos de impasse surgirão, aprenda a negociar e a ser empático(a). Às vezes, muitas inclusive, o dono do produto coloca um monte de coisas, mas não é “forçar a barra”, afinal ele também é pressionado por resultado como você. Sem teorias da conspiração, ok?

Algumas perguntas legais para fazer:

- quais itens fornecem o maior valor para o cliente?
- quais itens fornecem mais benefícios para o negócio?
- qual é a entrega de valor mais próxima/rápida de fazer/obter?
- quais itens são os mais arriscados/desafiadores?
- quais itens resultam no maior custo se não forem feitos agora?
- quais são as dependências entre os itens?
- quais itens contribuem mais para o nosso objetivo dessa *Sprint*?

Ou seja, a lista não se esgota nisso apenas, cada um sabe onde sente mais dor, certo?

Eu recomendo fortemente tanto ao dono do produto quanto ao time de desenvolvimento priorizarem por itens que fornecerão maior valor para o cliente. Por outro lado, há quem prefira por valor comercial, que nesse caso é a soma cumulativa de coisas que afetam a saúde de uma empresa. Isso envolve muitas coisas, como valor para o acionista e valor para o cliente.

Aliás, sempre façam isso!

Criação do *Sprint Backlog*

Você conhece a “regra de Pareto”? Em outras palavras: 80% dos benefícios da empresa ou do projeto vêm de 20% do tempo despendido pela equipe. Em um contexto de desenvolvimento de produtos Ágil, a mesma regra seria algo como 80% do valor do produto vem de 20% dos itens do *backlog* do produto.

Quando vemos a aplicação, devemos priorizar daquele *backlog* do produto imenso que desenvolvemos em etapas anteriores o que será desenvolvido seguindo os princípios e explanações que dei até aqui.

Para priorizar histórias de usuários, poderíamos usar alguns critérios de priorização:

- valor que os usuários atribuem à visão do produto;
- urgência;
- restrições de tempo;
- complexidade técnica;
- preferências das partes interessadas.

Claro, você pode utilizar outros, ou sua empresa exigir que sejam outros. Mantendo o bom senso, Ágil aceita!

Lidamos com o problema de priorização em dois níveis:

- **Nível do produto:** determinar quais recursos do produto poderiam contribuir melhor para atingir os objetivos principais do projeto;

- **Nível de tarefas:** definir quais partes do trabalho, as histórias de usuário ou as tarefas que devem ser realizadas e em que ordem, durante o processo de desenvolvimento do produto de *software*.

A priorização de requisitos em todas as metodologias de desenvolvimento de *software* é considerada uma parte vital do projeto, mas é especialmente crítica no desenvolvimento de *software Ágil*.

De qualquer forma, como em projetos Ágeis tudo é muito lúdico e coletivo, você pode criar um jogo parecido com o *Planning Poker*, criar categorias de prioridades e jogar cartas acumulando os pontos e categorias ou, por exemplo, utilizar a técnica *MoSCoW* utilizada por analistas de negócios. É assim:

Uma lista de requisitos ou histórias de usuário deve ser categorizada nos seguintes quatro grupos:

- **M: Must/Deve.** Descreve um requisito que deve ser satisfeito na solução final para que a solução seja considerada um sucesso;
- **S: Should/Deveria.** Representa um item de alta prioridade que deve ser incluído na solução, se possível. Frequentemente, esse é um requisito crítico, mas que pode ser satisfeito de outras maneiras, se for absolutamente necessário;
- **C: Could/Poderia.** Descreve um requisito que é considerado desejável, mas não necessário. Isso será incluído se o tempo e os recursos permitirem;
- **W: Won't/Não vou.** Representa um requisito que as partes interessadas concordaram que não será implementado em uma determinada versão, mas pode ser considerado no futuro.

Agora vamos pôr a mão na massa. O *sprint backlog* é uma lista de tarefas identificadas pela equipe *Scrum* a serem concluídas durante o *sprint*. Durante a reunião de planejamento do *sprint*, a equipe seleciona alguns itens do *backlog* do produto, **geralmente na forma de histórias de usuário, e identifica as tarefas necessárias para concluir cada história de usuário**. A maioria das equipes também **estima quantas horas cada tarefa levará para ser concluída por alguém da equipe**.

É fundamental que a equipe selecione os itens e o tamanho da lista de pendências do *sprint*. Por serem as pessoas que se comprometem a completar as tarefas, devem ser as pessoas que escolherão com o que se comprometerão durante o *Sprint*.

O *sprint backlog* é comumente mantido como uma planilha, mas também é possível usar o *TRELLO* ou qualquer um de vários produtos de *software* projetados especificamente para *Scrum* ou Ágil.

Tabela 1 – Exemplo simples de *Sprint Backlog* usando planilha

História do Usuário	Tarefas	Dia 1	Dia 2	Dia 3	Dia 4	Dia 5
Como membro, posso ler perfis de outros membros para encontrar alguém até hoje.	Codifique o...	8	4	8	0	
	Projete o...	16	12	10	4	
	Encontre-se com Mary sobre...	8	16	16	11	
	Projete a IU	12	6	0	0	
	Teste automação...	4	4	1	0	
	Codifique o outro...	8	8	8	8	

História do Usuário	Tarefas	Dia 1	Dia 2	Dia 3	Dia 4	Dia 5
Como membro, posso atualizar minhas informações de faturamento.	Atualizar teste de segurança	6	6	4	0	
	Projete uma solução para...	12	6	0	0	
	Escreva um plano de teste	8	8	4	0	
	Automatizar teste...	12	12	10	6	
	Codifique o...	8	8	8	4	

Fonte: MOUNTAIN GOAT

O *Sprint Backlog* requer que cada membro da Equipe de Desenvolvimento, junto com o *Scrum Master*, estime o quanto eles podem controlar durante cada sessão do *Sprint*. Em média, a maioria dos *Sprints* tende a durar duas semanas, mas esse número pode variar com base no tamanho da equipe e nas capacidades de produção. Certifique-se de escolher uma duração de *Sprint* que funcione para sua equipe, de forma que não tenham pressa para terminar as tarefas.

Então, para a nossa *sprint backlog*, temos, por exemplo (utilizando o Trello):

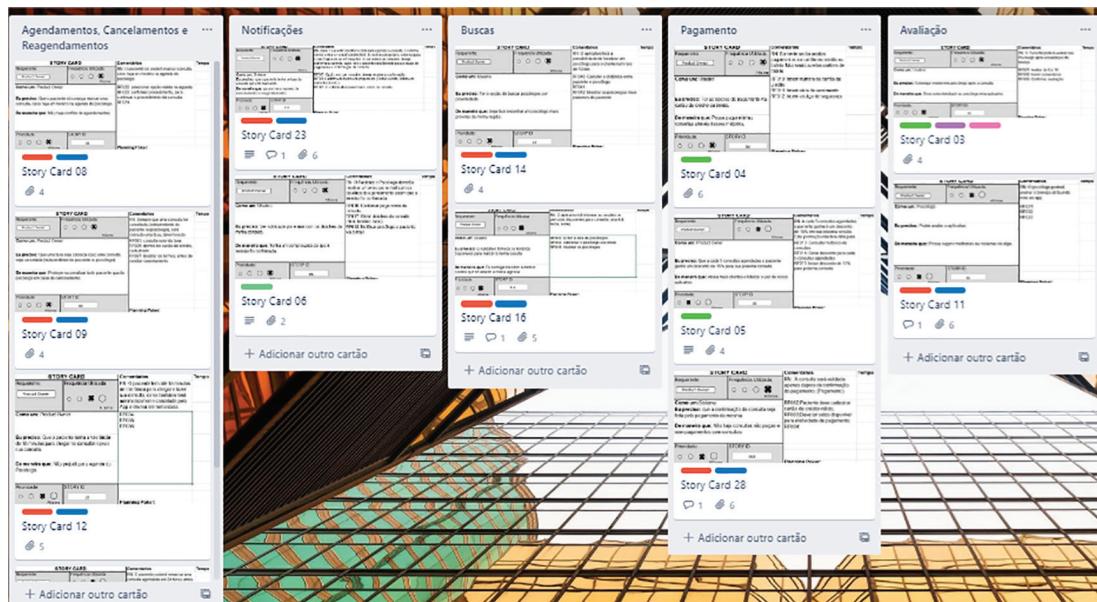


Figura 2 – *Sprint Backlog* de projeto

Fonte: Acervo do conteudista

Você pode ver as seleções das *sprints* aqui nomeadas verticalmente por temas ou funcionalidades, você pode escolher o nome que quiser usando o Trello. O importante é que você veja o arranjo.

Como você percebeu, na figura acima, por exemplo: as *Sprints* com nome de Agendamentos, Cancelamentos e Reagendamentos são Cartões que possuem várias tarefas e artefatos em seus interiores, os quais devem correr nas *sprints*; depois vemos uma outra *sprint* chamada de Notificações, que possui cartões com tarefas e artefatos em seu interior. Assim sucessivamente. Utilizamos código de cores para compor graus de dificuldade e prioridade.

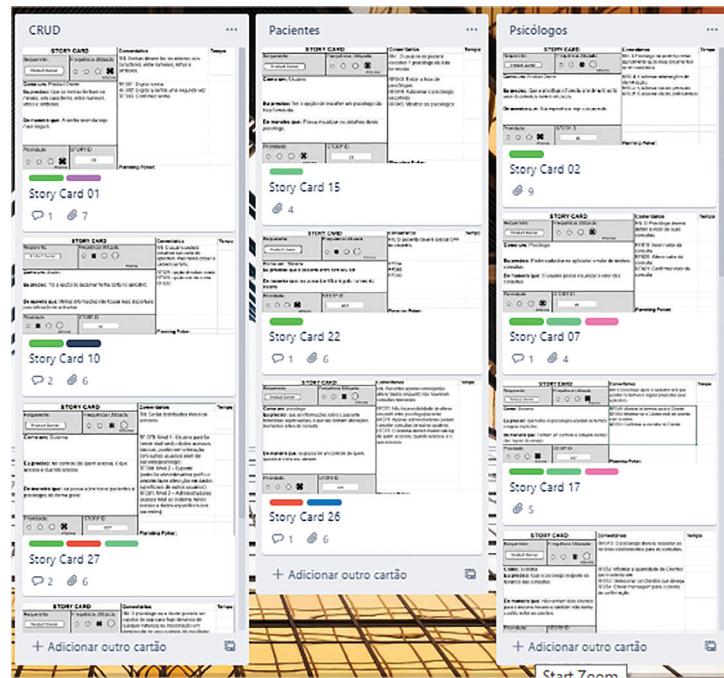


Figura 3 – Complemento das *Sprints* de desenvolvimento do projeto

Fonte: Acervo do conteudista

Desenvolvimento de Artefatos de Sistema Baseados em UML

Diagramas de Classe

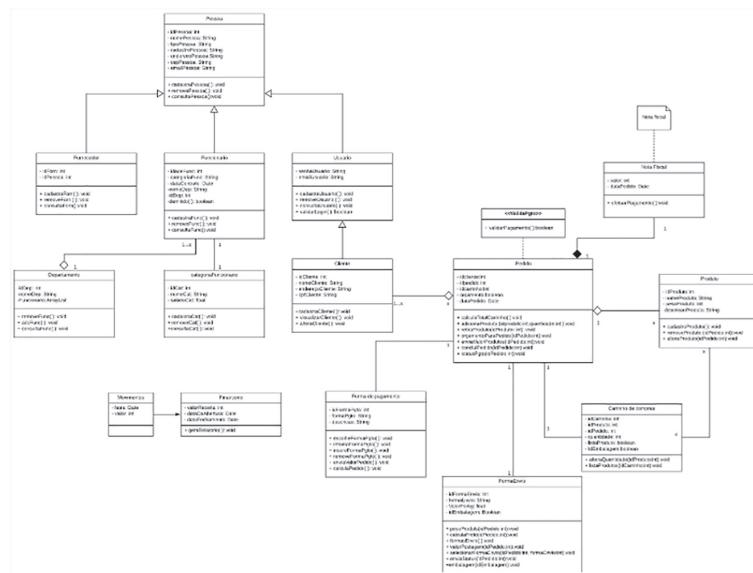


Figura 4 – Diagrama de classe geral do sistema

Fonte: Acervo do conteudista

Vamos ver em partes que se transformem em algo visível para que você possa praticar, já que os fundamentos já foram passados.

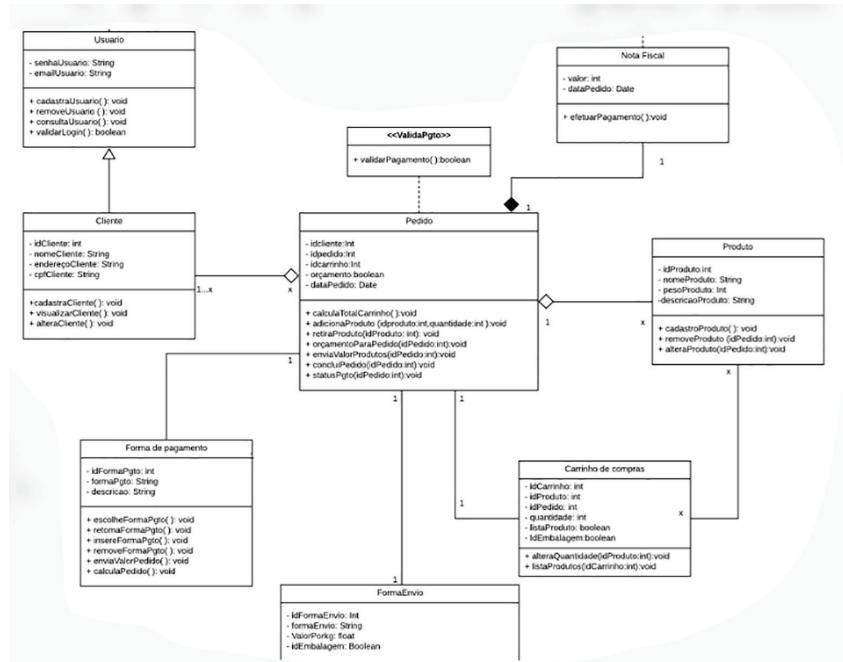


Figura 5 – Visão da entidade central para as transações de onde toda a modelagem iniciou (Pedidos)

Fonte: Acervo do conteudista

Diagrama de Caso de Uso

Vamos dar uma geral na nossa aplicação.

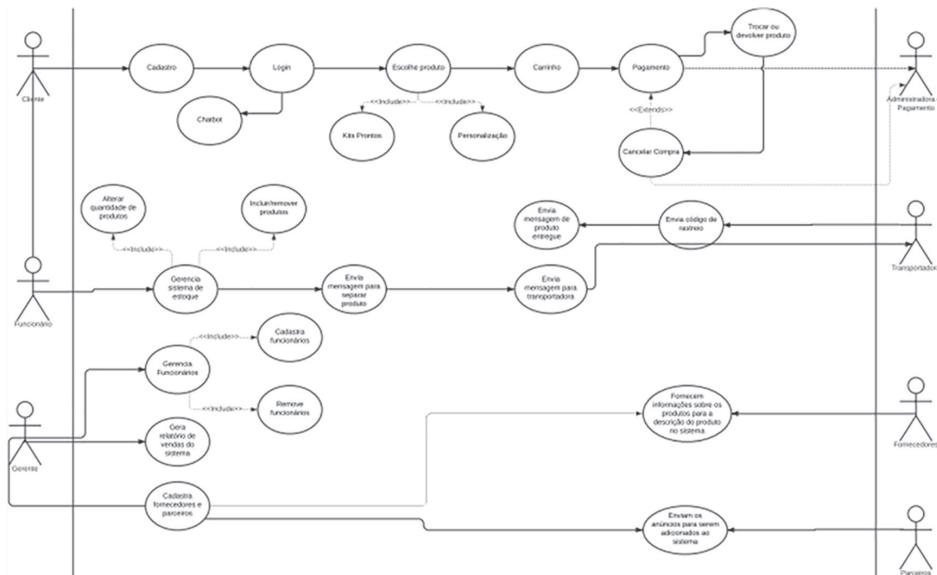


Figura 6 – Visão dos casos de uso gerais utilizados pela aplicação, bem como os atores envolvidos

Fonte: Acervo do conteudista

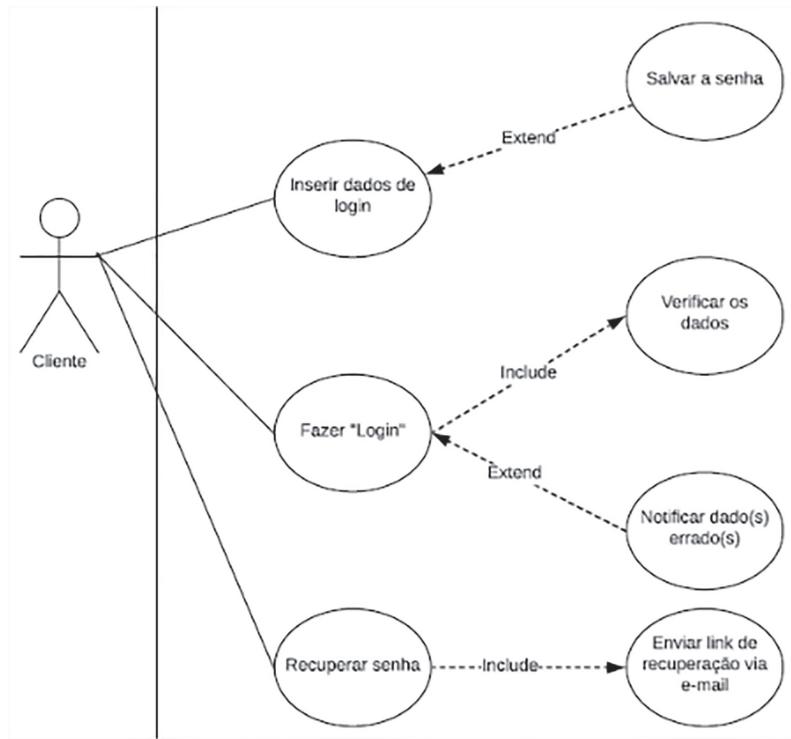


Figura 7 – Caso de uso em um nível maior de detalhamento do projeto

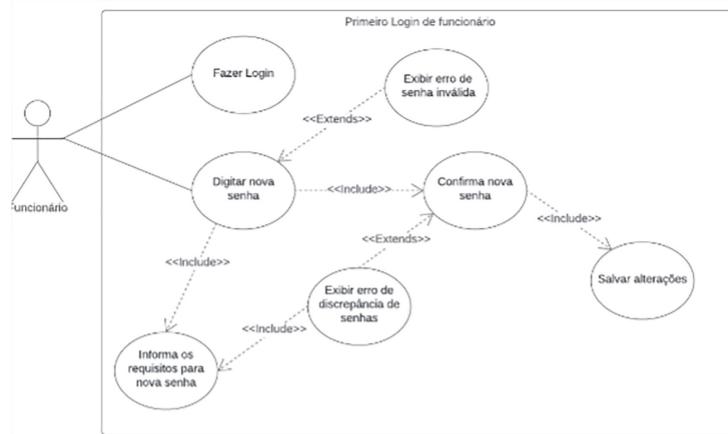
Fonte: Acervo do conteudista



Figura 8 – Caso de uso Alterar senha

Fonte: Acervo do conteudista

Casos de Uso Expandidos



Primeiro login de funcionário	
Referências:	RF 0040, RF 0041
Descrição geral:	O caso de uso inicia-se quando o funcionário faz seu primeiro login
Atores:	Funcionário.
Pré-condições:	Usuário incluso no sistema.
Garantia de sucesso (Pós-condições):	Senha alterada, funcionário logado.
Requisitos especiais:	
Fluxo básico:	<ol style="list-style-type: none"> 1. Fazer Login; 2. Digitar a nova senha; 3. Informar os requisitos para a nova senha; 4. Confirmar a nova senha; 5. Salvar as alterações.
Fluxo alternativo:	<ol style="list-style-type: none"> 2.1. Exibir erro de senha inválida; 2.2. Informar os requisitos para a nova senha; 2.3. Retornar ao passo 2; 3.1. Exibir erro de discrepância de senhas; 3.2. Retornar ao passo 3.

Figura 9 – Caso de uso expandido para o Primeiro *login* do funcionário. Nível de detalhamento aumenta

Fonte: Acervo do conteudista

Diagrama de Sequência

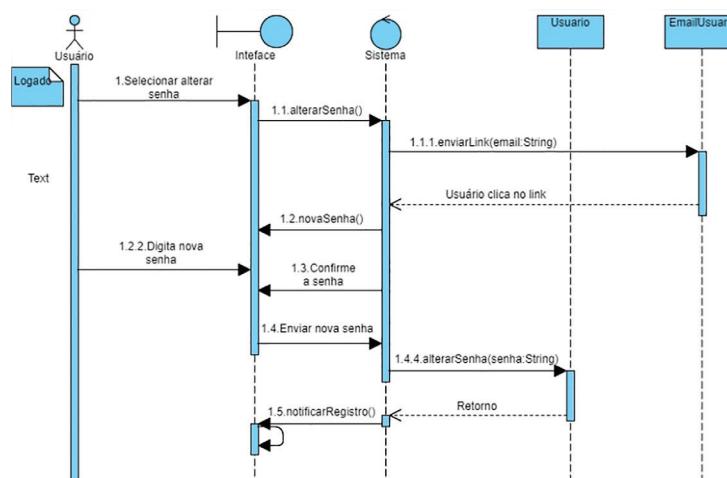


Figura 10 – Diagrama de sequência para o caso de uso Alterar a senha do projeto

Fonte: Acervo do conteudista

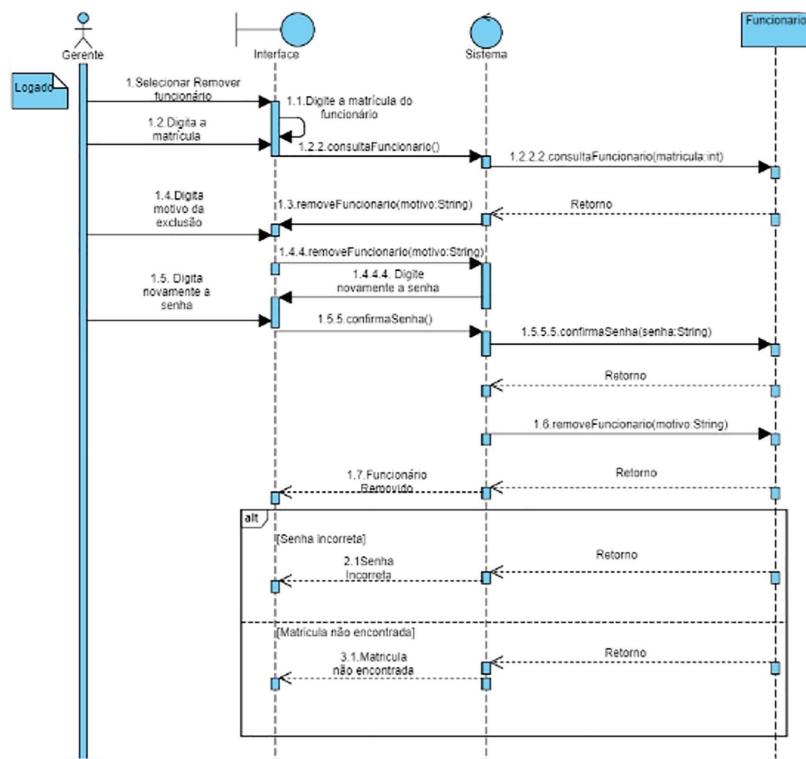


Figura 11 – Diagrama de sequência para o caso de uso Remover funcionário

Fonte: Acervo do conteudista

Claro que além desses artefatos da arquitetura do *software*, é importante você desenvolver o projeto lógico do banco de dados, bem como o projeto físico em um SGBD que você domine, como o MySQL ou MS-SQL ou ainda o *ORACLE*.

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

▶ Vídeos

Metodologia Ágil Scrum – Refinando as planilhas do *Product Backlog* e do *Backlog da Sprint*

<https://youtu.be/k0mbA5Ltg1w>

Micro Refinamento e Planejamento de itens do *Backlog*

<https://youtu.be/Ww04f8mRbIs>

Visão de produto no *Scrum*

<https://youtu.be/vg1S1WYZa6o>

Priorização *MoSCoW*

<https://youtu.be/QIDYcubuxUA>

Análise *MoSCoW*

<https://youtu.be/7XcthAr6JHA>

Referências

- ARLANDY, M. **Software architecture and agile. Are they both really compatible?** 2019. Disponível em: <<https://medium.com/quick-code/software-architecture-and-agile-are-they-both-really-compatible-c1eef0afcbb1>>. Acesso em: 30/08/2020.
- MEKNI, M.; BUDDHAVARAPU, G.; CHINTHAPATLA, S.; GANGULA, M. *Software architectural design in agile environments*. **Journal of Computer and Communications**, v. 6, n. 1, jan. 2018. Disponível em: <<https://www.scirp.org/journal/paperinformation.aspx?paperid=81436#:~:text=Software%20architecture%20and%20design%20is,agile%20environments%20does%20not%20exist>>. Acesso em: 30/08/2020.
- MIHAYLOV, B. **Towards na agile software architecture.** 2015. Disponível em: <<https://www.infoq.com/articles/towards-agile-software-architecture/>>. Acesso em: 30/08/2020.



Cruzeiro do Sul
Educacional