

# Índice

Índice	1
Abertura dos alunos	1
Tema: Comunicação não violenta - Marília & Carina	1
Desenvolvimento de software	2
1. Ágil	2
2. SCRUM	3
3. Terminal	7
4. Git	8
4. Atividade Complementar	9
Dúvidas	9
• Modelo em cascata?	9
• O que é cenário BDD?	10
• O que é Sprint?	10
• O que é PO?	11

**Assuntos: Introdução, Terminal, GIT, SCRUM.**

## Abertura dos alunos

Tema: Comunicação não violenta - Marília & Carina

- Livro Comunicação Não Violenta para dinâmicas de feedback e convívio social  
<[https://edisciplinas.usp.br/pluginfile.php/7390544/mod\\_resource/content/1/comunicacao-nao-violenta-marshall-b-rosenberg.pdf](https://edisciplinas.usp.br/pluginfile.php/7390544/mod_resource/content/1/comunicacao-nao-violenta-marshall-b-rosenberg.pdf)>

Quatro componentes:

- Observação (apresentar fatos ao invés de opiniões)
- Sentimento
- Necessidades
- Pedidos

Além de comunicar de forma gentil, também é parte do conceito a necessidade de que as ideias sejam transmitidas de forma clara e com ações palpáveis pelo ouvinte.

---

# Desenvolvimento de software

## 1. Ágil

Vídeo passado para assistirmos: What is Agile? <  What is Agile? >

Os princípios do método ágil são baseados em **4 valores**:

- Indivíduos e interações mais que processos e ferramentas
- Software funcionando mais que documentação abrangente
- Colaboração com o cliente mais que negociação de contratos
- Responder a mudanças mais que seguir um plano

Os **12 princípios** são os seguintes:

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. A contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade: a arte de maximizar a quantidade de trabalho não realizado--é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.

12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Esses princípios orientam as equipes ágeis na condução de projetos de software. Eles enfatizam a importância de:

- Foco no cliente e no valor entregue
- Adaptação às mudanças
- Comunicação e colaboração
- Excelência técnica
- Sustentabilidade

As metodologias ágeis, como Scrum, Kanban e Extreme Programming, são baseadas nesses princípios.

Cascata: (Anoto depois de ver a gravação da aula gente)

Ágil: rápido alinhamento, feedback constante, ajustes frequentes.

---

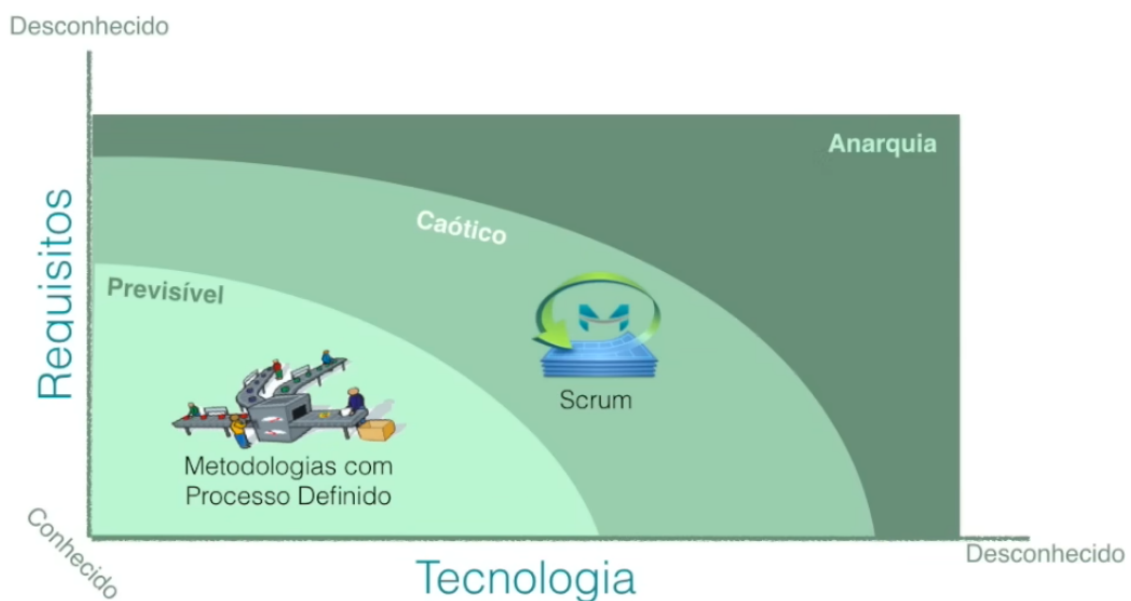
## 2. SCRUM

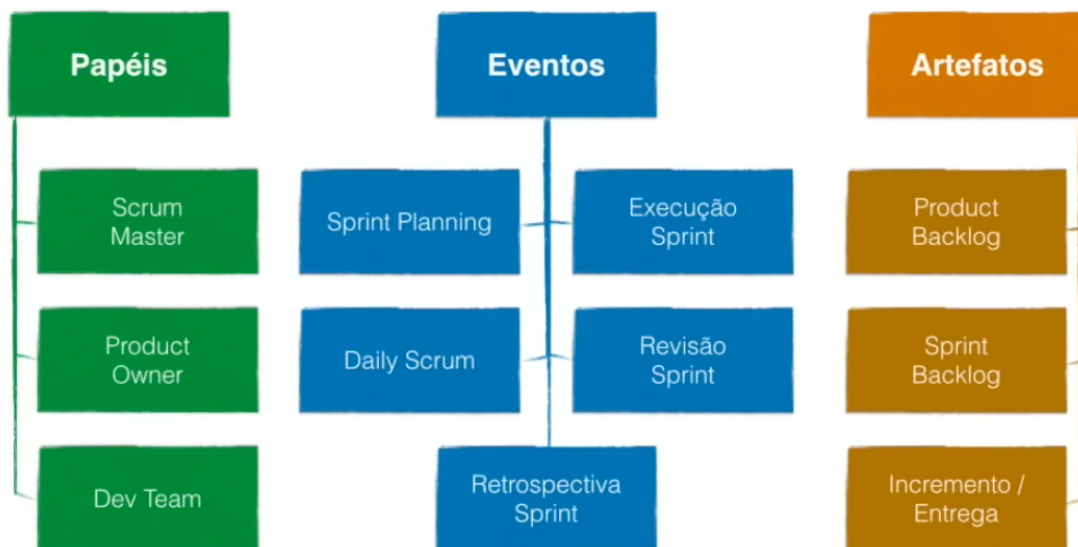
Vídeo 'Scrum - Aprenda Scrum em 9 minutos: [Scrum - Aprenda Scrum em 9 minutos](#)  
recomendação alunos:

artigo sobre scrum: <https://www.treasy.com.br/blog/scrum/>

slide explicativo sobre: [https://pt.slideshare.net/serge\\_rehem/scrum-em-15-minutos](https://pt.slideshare.net/serge_rehem/scrum-em-15-minutos)

Melhor aplicado quando não se conhece exatamente os processos necessários para se atingir o objetivo desejado e a tecnologia a ser utilizada.





#### Pilares:

- Transparência (Transparência dos processos, requisitos de entrega e status)
- Inspeção (Inspeção constante dos feitos)
- Adaptação (Tanto processo, quanto produto, são adaptáveis a mudanças)

#### Participantes:

**Product Owner:** ponto central com poderes de liderança. decide a ordem e como os recursos serão utilizados. responsável pela comunicação com a equipe

**Scrum Master:** ajuda os envolvidos a entender o scrum e criar sua própria metodologia dentro do scrum

**Dev Team:** pessoas que vão atuar na parte operacional. é o time que decide quais práticas serão utilizadas e como chegar, na parte técnica, ao objetivo definido pelo Product Owner

**Visão do produto:** Product owner fornece macro planejamento

**Product Backlog:** Product owner e Scrum master determinam as funcionalidades necessárias, As prioridades são determinadas pela sua importância.

**Sprint:** Pequenas subdivisões das funcionalidades do product backlog, possuem datas de entrega individuais.

**Sprint Backlog:** Planilha de divisão dos sprints, com tempos pré definidos e fixos de entrega (geralmente de 2 a 4 semanas).

**Planejamento Sprint:** Reunião de time que define quantas funcionalidades podem ser concluídas no tempo de um sprint.

**Execução Sprint:** Se montam as funcionalidades determinadas. É esperado que ao fim de cada sprint se tenha um pedaço funcional do produto.

#### Daily - Reunião diária de acompanhamento (15 min):

- ☐ O que fez ontem?
- ☐ O que vai fazer hoje?
- ☐ Tem algum impedimento para realizar as tarefas escolhidas para o dia?

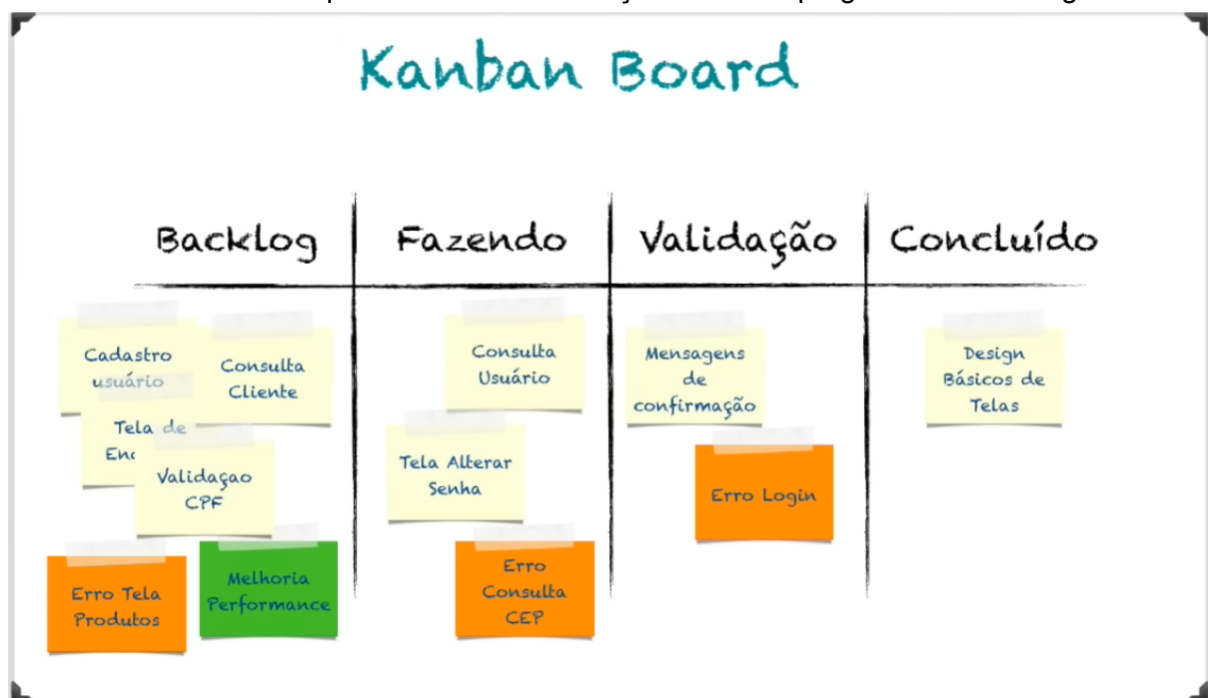
**Revisão Sprint:** Ao fim de um sprint, se avalia o sprint e verifica-se se são necessárias mudanças, e se ele está de acordo com as expectativas do projeto, ou se ocorrerá uma adaptação do Product Backlog.

**Retrospectiva Sprint:** Verificação das necessidades de adaptação nos processos, e o que pode ser melhorado ou modificado.

**Burndown chart:** Um gráfico de determinação de progresso de sprint, ele determina quantas histórias devem ser entregues por dia em um sprint, e demonstra visualmente o progresso.



**Kanban Board:** Tabela que dá uma demonstração visual do progresso do backlog.



## Montando o Sprint Backlog:



No To-Do sabemos quais tarefas temos que fazer para o desenvolvimento do software. Cada um do time pega uma tarefa, coloca seu nome e passa para o In-Progress

Vídeo: 10 DICAS PARA ESCREVER HISTÓRIAS DE USUÁRIO

1. Evitar o uso da palavra usuário e utilizar a construção de personas
2. Não se apegue as formas clássicas
3. Criar e moldar os echos
4. Criar histórias independentes
5. Listar cenários de PPT(e sempre em cenários de negócios)
6. Detalhar a história
7. Avaliar as 7 dimensões de um produto de software
8. A história de usuário é o meio e não o fim
9. Histórias de usuários são lembretes do que vai fazer no futuro
10. Cuidado com o design empírico x design participativo (fazer o design participativo com o cliente para que ele participe e colabore junto com o time)

- 
1. criação de personas
  2. conforme o processo avança, definir as próximas etapas
  3. sempre lembrar do acrônimo INVEST: Independente (não se basear em histórias anteriores); negociável (tudo pode ser modificado a qualquer momento); Valorosa (valor para o cliente)
  4. listar critérios de aceitos pelo cliente + cenários BDD (??)

- a. Cenários de Desenvolvimento Orientado a Comportamento (BDD). “Sabendo que [x] faz [y], então [z]”.



- b.
- Desenvolvimento Orientado a Comportamento (BDD) é uma abordagem que consiste em definir o comportamento de um recurso através de exemplos em texto.
  - Focar em descrever cenários de negócio, e não cenários de interação puramente técnica
    - Exemplo de cenário de negócio: (??)
    - Exemplo de interação puramente técnica: “Sabendo que [a pessoa] [pressiona o botão], então [o botão deve ter uma função]”
5. Detalhar a história
6. História de usuário é meio e não fim

---

Story points são subdivisões nas histórias de usuário, que determinam pontos a serem seguidos para o desenvolvimento da história de usuário.

Organização dos story points priorizando as tarefas de menor complexidade/ dificuldade. Métodos de fibonacci e planning poker.

---

### 3. Terminal

Cookbook comandos terminal: <https://www.hostinger.com.br/tutoriais/comandos-linux>

#### Comandos mais comuns:

- **CD nomeDaPasta**  
Serve para navegar por entre arquivos e diretórios (caminhos de pastas).
- **Cat nomeDoArquivo.tipo**  
Serve para criar e relacionar arquivos.  
(exemplo: cat nomedoarquivo1.txt nomedoarquivo2.txt > nomedoarquivo3.txt  
junta os arquivos 1 e 2, em um novo arquivo, o arquivo 3)
- **CP nomeDoArquivo.tipo diretório/de/pasta**  
Serve para copiar arquivos para um diretório específico.
- **MV nomeDoArquivo.tipo diretório/de/pasta**  
Serve para mover e renomear arquivos  
Renomear: MV nomeDoArquivo.tipo nomeNovo.tipo

## 4. Git

Conteúdo repositório GIT e terminal: [https://github.com/geandrol/conteudo\\_git/tree/main](https://github.com/geandrol/conteudo_git/tree/main)

Se configurado da maneira correta, é possível comandar o GIT e GitHub por meio do prompt de comando (ou semelhantes).

Deve-se fazer os comandos git no diretório do projeto!

**Branch** são bifurcações no seu projeto, como se fossem “salas alternativas”, para fazer e testar mudanças, sem alterar de vez o projeto oficial.

### Tutorial básico início:

1. no prompt de comando coloque 'cd pasta/do/projeto' para ir ao diretório do projeto e fazer todas as alterações lá.
2. Copie o caminho do projeto no GitHub.
3. use o comando **git clone linkCopiado**
- 4.


### Comandos mais comuns GIT:

- **git clone linkCopiado**  
Clona as informações do repositório do gitHub, para a pasta atual.
- **git status**  
Mostra os arquivos adicionados recentemente ao projeto.
- **git add nomeDoArquivo ou diretório/de/pasta**  
Insere arquivos ou diretórios para o controle de versão.  
Pode-se usar um “.” no lugar do nome ou diretório, para ele adicionar o diretório atual.
- **git pull origin nomeDaBranch**  
Obtém o repositório a última atualização da ramificação do projeto.  
Geralmente a branch principal se chama “Master” ou “Main”
- **git commit -m “mensagem do commit”**  
Efetiva as alterações feitas na pasta do projeto para serem enviadas como uma versão. Ao fazer isso ele efetiva a versão local como definitiva.  
“Mensagem do commit” é o nome dado a essa versão do projeto.
- **git push origin nomeDaBranch**  
Faz o upload do último commit para a nuvem (geralmente GitHub)
- **git checkout nomeDaBranch**  
Altera a branch atual.
- **git checkout -b nomeNovaBranch**  
Cria uma nova branch, e costuma já acessa-la.
- **git branch -d nomeDaBranch**  
Deleta uma branch.



## 4. Atividade Complementar

Entregar no discord (Canal 'Atividades BSM'):

 SA-SUD1 - Handout - My Action Plan\_vF-TRAD (REV FE).docx

(Crie uma cópia do arquivo para preencher)

## Dúvidas

- Modelo em cascata?

R: É um processo que vai sendo realizado passo a passo, anteriormente ao método Ágil era o mais utilizado nas empresas.

R2:

O modelo em cascata é um modelo de desenvolvimento de software sequencial, no qual o processo é visto como um fluir constante para frente, como uma cascata. As etapas do modelo em cascata são:

- Requisitos: Nesta etapa, são definidos os requisitos do software, incluindo funcionalidades, restrições e objetivos.
- Projeto: Nesta etapa, é criado o projeto do software, incluindo a arquitetura, o design e a implementação.
- Implementação: Nesta etapa, o software é implementado, codificado e testado.
- Testes: Nesta etapa, o software é testado para garantir que atende aos requisitos.
- Manutenção: Nesta etapa, o software é mantido e atualizado.

O modelo em cascata é um modelo simples e fácil de entender. Ele é adequado para projetos de software em que os requisitos são bem definidos e não há muitas mudanças esperadas. No entanto, o modelo em cascata também apresenta algumas desvantagens, como:

- É difícil adaptar o projeto às mudanças de requisitos.
- É difícil identificar problemas no início do projeto.
- O projeto pode ser demorado e caro.

O modelo em cascata é um modelo de desenvolvimento de software clássico que ainda é utilizado em alguns projetos. No entanto, ele tem sido substituído por metodologias ágeis, como Scrum e Kanban, que são mais flexíveis e adaptáveis às mudanças.

Aqui estão algumas vantagens e desvantagens do modelo em cascata:

Vantagens:

- É simples e fácil de entender.
- É adequado para projetos com requisitos bem definidos.
- É um processo controlado e documentado.

Desvantagens:

- É difícil adaptar o projeto às mudanças de requisitos.
- É difícil identificar problemas no início do projeto.
- O projeto pode ser demorado e caro.

## • O que é cenário BDD?

Desenvolvimento Orientado a Comportamento (BDD) é uma abordagem que consiste em definir o comportamento de um recurso através de exemplos em texto.

Cenários de Desenvolvimento Orientado a Comportamento (BDD).

Exemplo: “Sabendo que [x] faz [y], então [z]”.

- a. Cenários de Desenvolvimento Orientado a Comportamento (BDD). “Sabendo que [x] faz [y], então [z]”.



- b.
  - i. Desenvolvimento Orientado a Comportamento (BDD) é uma abordagem que consiste em definir o comportamento de um recurso através de exemplos em texto.
- c. Focar em descrever cenários de negócio, e não cenários de interação puramente técnica
  - i. Exemplo de cenário de negócio: (??)
  - ii. Exemplo de interação puramente técnica: “Sabendo que [a pessoa] [pressiona o botão], então [o botão deve ter uma função]”

- O que é Sprint?

Subdivisões de um product backlog, com uma data de conclusão fixa.

- O que é PO?

Product Owner; Dono do produto.