

# *Laboratório de Fundamentos em TIC*

---

Princípios de Engenharia de Software

Prof. Gabriel Resende Machado



[gabrielmachado@unifeso.edu.br](mailto:gabrielmachado@unifeso.edu.br)



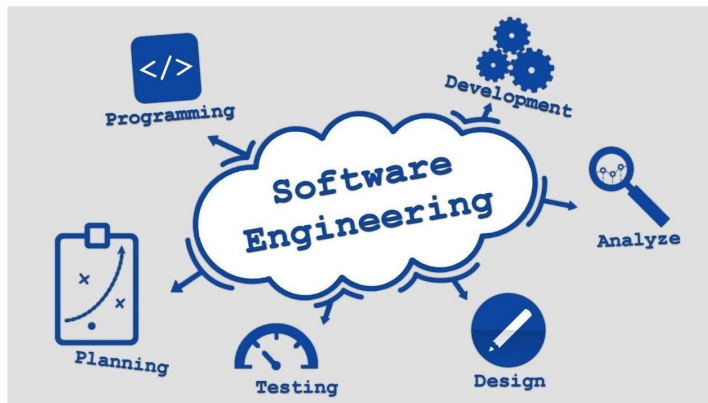
<https://www.linkedin.com/in/machadogabriel>



<https://github.com/UNIFESO-Gabriel/fundamentos-em-tic>

# O que é Engenharia de *Software*?

- Desenvolver um *software* de grande porte não é uma tarefa fácil! Pode ser comparado à construção de casas ou edifícios.
- É necessário o envolvimento tanto do cliente como também de diversos profissionais para que o projeto saia como esperado;
- É fundamental o trabalho em equipe e boa comunicação entre seus membros!
- Para isso, a **Engenharia de *Software*** busca promover e aplicar práticas já consolidadas por profissionais de renome, visando desde o **planejamento e administração, até o desenvolvimento, entrega e manutenibilidade** de um produto de *software*.



# O que é Engenharia de *Software*?

- A Engenharia de *Software* busca enfrentar os desafios da criação de *software* que maneira **eficiente, com alta qualidade, que caiba no orçamento e no cronograma do projeto, e, principalmente, que atenda às necessidades dos usuários finais**;
- A Engenharia de *Software* pode ser dividida em dois ramos que precisam comunicar-se entre si constantemente: **(i) Administrativo e (ii) Técnico**;

Administrativo	Técnico
Levantamento de Requisitos do <i>Software</i>	<i>Design</i> de <i>Software</i> (planejamento estrutural)
<i>Design</i> de <i>Software</i> (planejamento conceitual)	Desenvolvimento de <i>Software</i>
Gestão do Projeto (RH, Financeiro, Tempo)	Testes de <i>Software</i> ( <i>Software Quality Assurance</i> )
Metodologias Ágeis (Scrum, Kanban, XP)	Manutenção de <i>Software</i> ( <i>bugs</i> , manutenções)
	Ferramentas e DevOps ( <i>Git</i> , CI/CD, <i>Docker</i> , IaC)

# Ciclo de Vida do Desenvolvimento de *Software* (SDLC)

- O SDLC (*Software Development Life Cycle*) é um processo estruturado que abrange todas as etapas necessárias para criar, implantar e manter um *software*;
- O SDLC fornece uma estrutura para o desenvolvimento e gestão de projetos de *software*, garantindo que cada etapa seja bem planejada e executada.



# Fases do SLDC

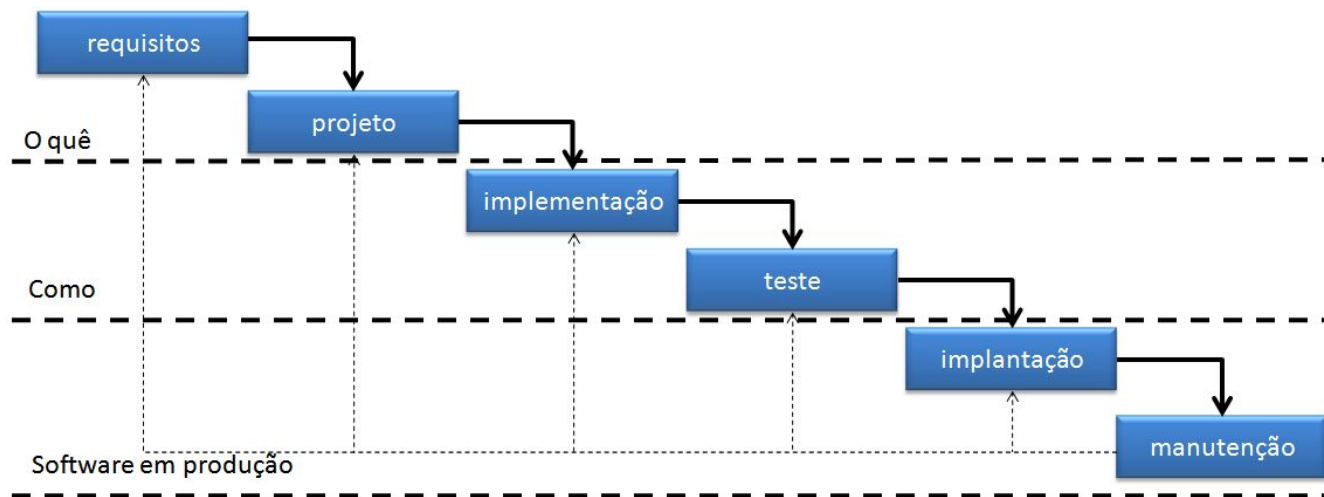
Ciclo	Objetivo	Atividades
Planejamento e Análise de Requisitos	Entender e documentar as necessidades dos usuários e <i>stakeholders</i>	Reuniões com <i>stakeholders</i> , levantamento de requisitos, análise de viabilidade, elaboração de especificações de requisitos
<i>Design</i>	Definir a arquitetura do sistema e detalhar como o <i>software</i> atenderá aos requisitos	Criação de diagramas de arquitetura e de banco de dados, <i>design</i> de interfaces de usuário, especificação de componentes e módulos
Desenvolvimento	Codificar o <i>software</i> conforme o design especificado	Programação, criação de <i>scripts</i> de banco de dados, desenvolvimento de interfaces de usuário, integração de sistemas
Testes	Verificar e validar que o <i>software</i> funciona conforme o esperado e atende aos requisitos	Testes unitários, testes de integração, testes de sistema, testes de aceitação, correção de <i>bugs</i>
Implantação	Colocar o <i>software</i> em produção, tornando-o disponível para os usuários finais	Preparação do ambiente de produção, migração de dados, instalação do <i>software</i> , treinamento de usuários, lançamento do <i>software</i>
Manutenção	Corrigir problemas, melhorar o desempenho e adicionar novas funcionalidades conforme necessário	Correção de <i>bugs</i> , atualizações de segurança, melhorias de desempenho, adição de novos recursos, suporte técnico

# Modelos de SDLC

- Um modelo de SDLC é uma estrutura ou metodologia que define as etapas e processos envolvidos no ciclo de desenvolvimento de *software*;
- Esses modelos fornecem uma abordagem sistemática e disciplinada para planejar, criar, testar e implementar *software*, garantindo que cada fase do processo seja executada de maneira eficiente e controlada;
- Objetivos de um modelo de SDLC:
  - **Planejamento e Organização:** garantir que o desenvolvimento do *software* seja bem planejado e organizado;
  - **Controle de Qualidade:** assegurar que o *software* produzido atenda aos requisitos e padrões de qualidade;
  - **Gerenciamento de Riscos:** identificar e mitigar riscos ao longo do processo de desenvolvimento;
  - **Eficiência e Produtividade:** melhorar a eficiência e a produtividade das equipes de desenvolvimento;
  - **Entrega Pontual:** ajudar a garantir que os projetos de software sejam concluídos no prazo e dentro do orçamento.

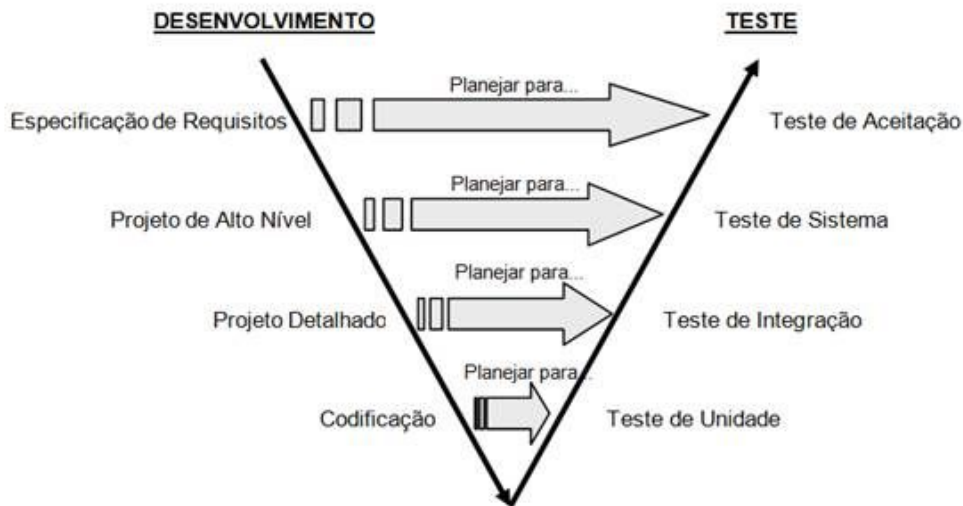
# Modelos de SDLC - Cascata

- Modelo linear e sequencial, onde cada fase do ciclo de vida do *software* deve ser concluída antes da próxima começar;
- **Vantagens:** (i) simples e fácil de entender e gerenciar; (ii) fases bem definidas e distintas;
- **Desvantagens:** (i) inflexível para mudanças de requisitos tardias; (ii) requisitos e funcionalidades não desenvolvidas para todo o processo; (iii) problemas são frequentemente descobertos apenas na fase de teste.



# Modelos de SDLC - Modelo V (V-Model)

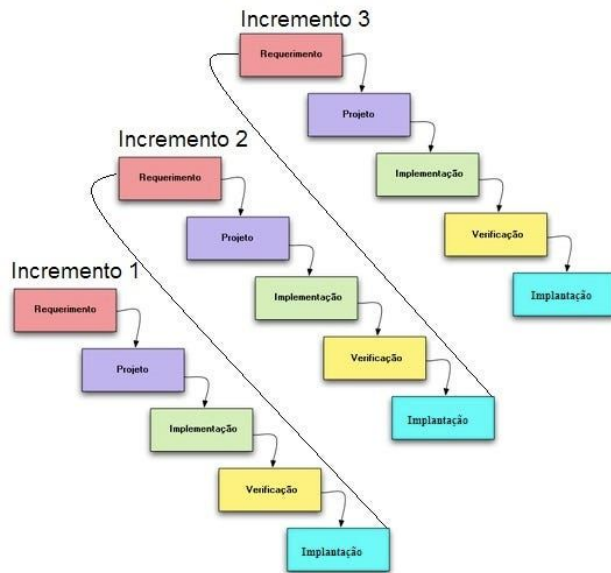
- Extensão do modelo cascata, onde cada fase de desenvolvimento tem uma fase correspondente de testes;
- **Vantagens:** (i) estrutura clara para testes em todas as fases; (ii) rastreabilidade entre fases de desenvolvimento e testes
- **Desvantagens:** (i) ainda é sequencial e pode ser inflexível; (ii) requer boa documentação e planejamento.





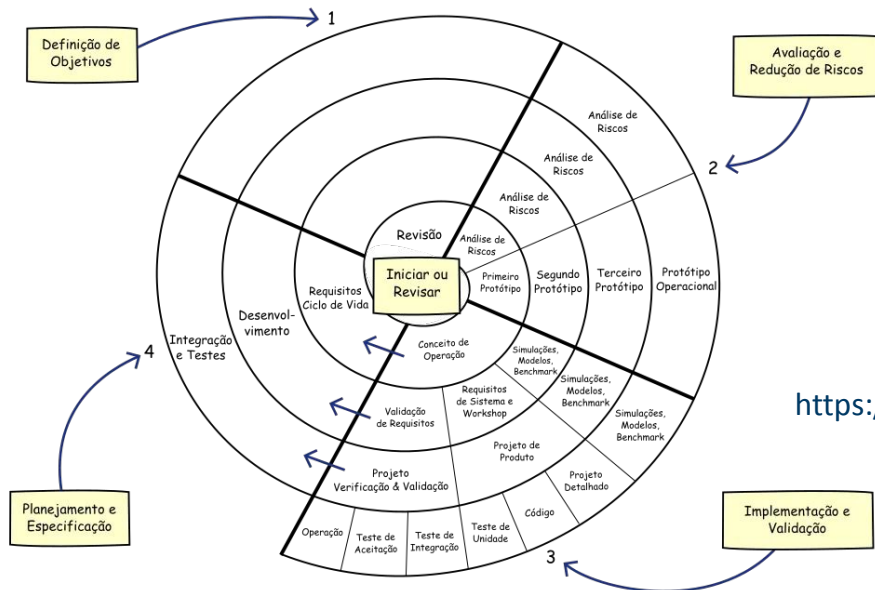
# Modelos de SDLC - Iterativo Incremental

- Desenvolve o software em pequenas partes (iterações), permitindo *feedback* e ajustes ao longo do desenvolvimento;
- Cada iteração inclui planejamento, *design*, desenvolvimento, teste e *feedback* de um incremento do sistema.
- **Vantagens:** (i) Flexível e adaptável a mudanças; (ii) *feedback* constante e melhorias contínuas;
- **Desvantagens:** (i) requer boa gestão e comunicação contínua; (ii) pode ser difícil de gerenciar se não for bem estruturado; (iii) pode consumir recursos rapidamente se não for controlado.



# Modelos de SDLC - Modelo Espiral

- O modelo espiral passa continuamente pelas fases de planejamento, projeto, construção e teste, com melhorias graduais a cada passagem;
- É um processo evolucionário, ou seja, adequado para *softwares* que necessitam de inúmeros refinamentos ao longo do seu desenvolvimento;
- **Vantagens:** (i) foco na gestão de riscos; (ii) maior flexibilidade e a permissão de melhorias contínuas;
- **Desvantagens:** (i) pode ser complexo e caro de implementar; (ii) requer experiência em gestão de riscos.



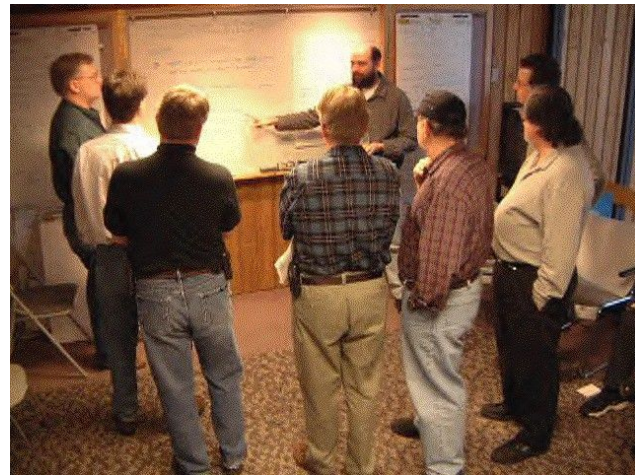
Disponível em  
<https://tinyurl.com/2cc92mfk>.

# Modelos de SDLC - Metodologias Ágeis

- Ágil não tem a ver apenas com rapidez, mas também com **flexibilidade** e **integração**;
- Metodologias ágeis **são adaptáveis a mudanças repentinas no planejamento**, como prioridades, tarefas e características do projeto conforme necessário;
  - O projeto é todo dividido em etapas menores, para que seja mais fácil aplicar mudanças sem comprometer a qualidade;
- Todos os métodos ágeis são guiados pelo **Manifesto para Desenvolvimento Ágil de *Software***, um compilado de valores e princípios baseados na gestão de projetos de *software* bem-sucedidos;
  - Conta com 17 assinaturas de nomes renomados na área de Engenharia de *Software*, como Kent Beck e Martin Fowler. Está disponível em <https://agilemanifesto.org/iso/ptbr/manifesto.html>;
- **O Manifesto Ágil é baseado em quatro valores principais:**
  - Os indivíduos e suas interações acima de procedimentos e ferramentas;
  - O funcionamento do *software* acima de documentação abrangente;
  - A colaboração com o cliente acima da negociação e contrato;
  - A capacidade de resposta às mudanças acima de um plano pré-estabelecido.

# Modelos de SDLC - Princípios das Metodologias Ágeis

- Os signatários do Manifesto Ágil elencaram doze princípios que deixam as intenções das metodologias ágeis mais claras:
  - 1) Priorizar a satisfação do cliente com entregas contínuas e antecipadas;
  - 2) Acolher mudanças nos requisitos para vantagem competitiva;
  - 3) Entregar *software* funcional frequentemente, em intervalos curtos;
  - 4) Colaboração diária entre negócios e desenvolvedores;
  - 5) Projetos com indivíduos motivados, com suporte e confiança;
  - 6) Comunicação eficiente por meio de conversas face a face;
  - 7) Medir o progresso pelo *software* funcional;
  - 8) Promover o desenvolvimento de *software* auto-sustentável;
  - 9) Foco na excelência técnica e bom *design*;
  - 10) Simplicidade, maximizando o trabalho não realizado;
  - 11) Arquiteturas e designs emergem de equipes auto-gerenciáveis;
  - 12) Reflexão regular para melhorar a eficácia.

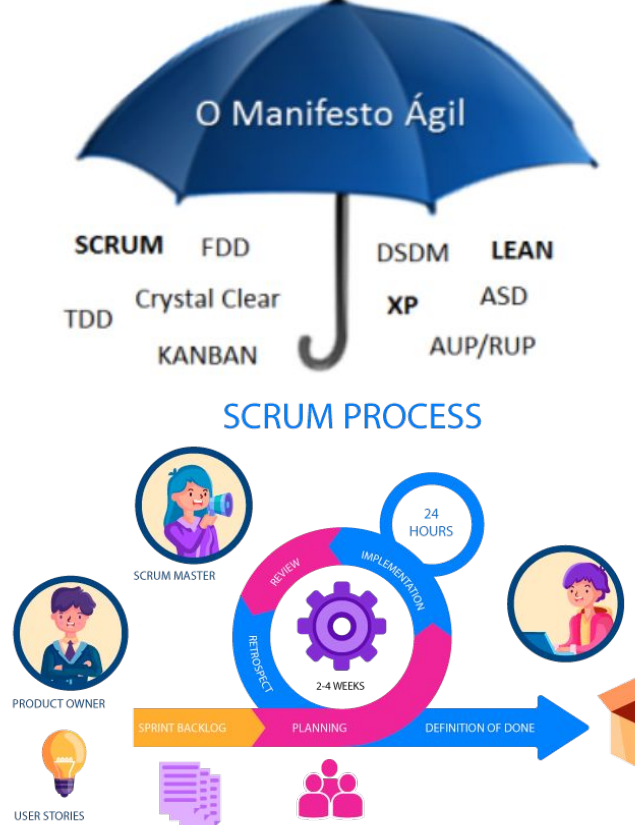


# Modelos de SDLC - Metodologias Ágeis vs. Tradicionais

- A principal diferença entre os métodos ágeis e os métodos tradicionais está no **modo como os projetos são planejados**.
  - Na gestão tradicional existe um período **antes da execução do projeto** destinado unicamente ao planejamento onde todas as especificações, prazos e responsabilidades são definidos;
  - Na gestão ágil **apenas o básico é decidido no começo, e o projeto se define com o passar do tempo, de forma iterativa e simultânea**. Portanto, as características do produto final podem ser alteradas, se necessário, em qualquer ponto de sua execução a critério do cliente.

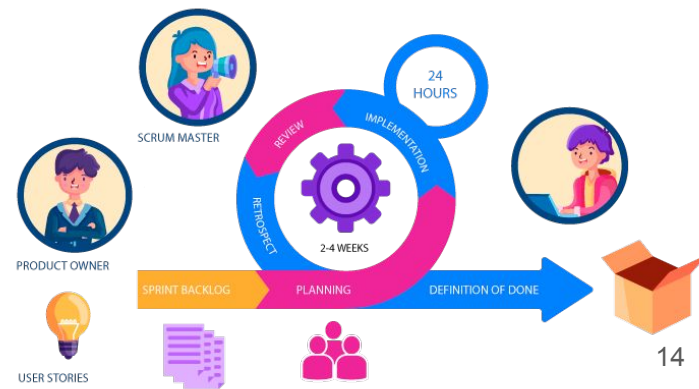
Metodologia Tradicional	Metodologia Ágil
Uma única entrega	Entrega em partes (iterativa)
Foco em uma tarefa por vez	Foco em várias tarefas simultaneamente
Planejamento completo antes da execução	Planejamento iterativo e incremental
Poucas alterações	Alterações sempre que necessário
Pouca interação com o cliente	Interações frequentes com o cliente

# Modelos de SDLC - Metodologias Ágeis - *Frameworks*

- “Agile is an umbrella term for all the methods and methodologies that follow the values and principles described in the Agile Manifesto” (Autor desconhecido);
  - Dentre os principais *frameworks* de metodologias ágeis mais adotadas atualmente, estão:
1. **Scrum:** desenvolvido como uma abordagem para gerenciar e controlar projetos complexos, especialmente no desenvolvimento de *software*.
    - 1.1. Os projetos são divididos em **sprints de 2 a 4 semanas**, cada um contendo um conjunto de tarefas específicas oriundas do **backlog**. Após a conclusão de uma *sprint*, a próxima começa imediatamente até que o projeto seja finalizado;
    - 1.2. A equipe Scrum é formada pelo (i) *Product Owner*: define as características do produto e os requisitos; (ii) *Time Scrum*: equipe responsável pela execução das tarefas de cada *sprint*; (iii) *Scrum Master*: facilita o trabalho da equipe e garante a aderência às práticas Scrum.
    - 1.3. **Daily Scrum:** Reuniões diárias para discutir o progresso, tarefas do dia e ajustar prioridades conforme necessário.  
**Sprint Meeting:** reunião de retrospectiva da *sprint*.
- 



## SCRUM PROCESS



# Modelos de SDLC - Metodologias Ágeis - *Frameworks*

- 2. **XP (*eXtreme Programming*)**: proposto por Kent Beck. Enfatiza práticas técnicas como o *pair programming*, integração contínua, refatoração e testes automatizados.
  - 2.1. O método XP conta com times pequenos e colaborativos trabalhando em pares. O cliente é parte ativa do time. Já o Scrum possui papéis definidos como *Product Owner*, *Scrum Master* e Time Scrum. O *Product Owner* representa os interesses do cliente;
  - 2.2. O XP conta com iterações ligeiramente mais curtas que o Scrum, geralmente de uma a duas semanas. A entrega é frequente e incremental;
  - 2.3. O XP dá mais importância a práticas técnicas, enquanto o Scrum fornece uma estrutura para a gestão do projeto e facilita a comunicação e organização da equipe.
  - 2.4. **Virtudes do XP:**
    - 2.4.1. **Comunicação**: visa evitar lacunas em processos e problemas entre clientes e equipes;
    - 2.4.2. **Simplicidade**: aplicada durante todo o processo, desde os requisitos até a entrega da solução;
    - 2.4.3. **Feedback**: consiste em obter e analisar informações entre os membros da equipe e clientes;
    - 2.4.4. **Coragem**: basicamente refere-se à coragem de dizer não quando necessário, focando no essencial.

# Modelos de SDLC - Metodologias Ágeis - *Frameworks*

3. **Kanban:** proposto por Taiichi Ohno, engenheiro da Toyota em 1940. A palavra *Kanban* significa **cartão visual** em japonês. O método utiliza cartões para visualizar o trabalho em andamento e otimizar o fluxo de produção.
- Em Engenharia de *Software*, o *Kanban* é uma **metodologia visual e simples baseada em quadros de trabalho e cartões**. Uma equipe pode trabalhar com vários quadros, dependendo dos tipos de funções que desempenham na empresa;
  - Os quadros são separados por colunas, como “pendentes”, “em andamento” e “concluídas”, e são customizáveis de acordo com o modo de trabalho da equipe. Cada cartão é uma tarefa a ser executada, e podem incluir a descrição, prazo de entrega e membros responsáveis por sua execução, entre outras informações.





# Software Quality Assurance (SQA)

- SQA (*Software Quality Assurance*) é um conjunto de atividades e práticas destinadas a garantir que o *software* atenda aos padrões de qualidade e requisitos especificados;
- Envolve processos de monitoramento, revisão, testagem e melhoria contínua durante o ciclo de desenvolvimento do *software*. São etapas do SQA;
  - **Planejamento de Qualidade:** define de padrões de qualidade, métricas de desempenho e planos de teste;
  - **Revisões e Auditorias:** Durante o ciclo de desenvolvimento, são realizadas revisões e auditorias para garantir os padrões estabelecidos e identificar problemas;
  - **Testes de Software:** visa identificar e corrigir *bugs* no *software* antes que ele seja lançado. Isso inclui (i) testes de unidade (adoção do TDD), (ii) testes de integração, (iii) testes de sistema e (iv) testes de aceitação do usuário;
  - **Gestão de Configuração:** envolve o controle das e garantia mudanças no *software* e na documentação associada forma eficaz;
  - **Garantia de Processo:** visa garantir que os processos utilizados no desenvolvimento de software sejam eficazes e eficientes, a partir de definição de processos e medições de desempenho;
  - **Relatórios e Melhoria Contínua:** envolve a geração de relatórios sobre o desempenho do processo de desenvolvimento e a identificação de áreas para melhoria contínua. Isso pode incluir análise de métricas de qualidade, *feedback* dos clientes e lições aprendidas em projetos anteriores.

# *Laboratório de Fundamentos em TIC*

---

Princípios de Engenharia de Software

Prof. Gabriel Resende Machado



[gabrielmachado@unifeso.edu.br](mailto:gabrielmachado@unifeso.edu.br)



<https://www.linkedin.com/in/machadogabriel>



<https://github.com/UNIFESO-Gabriel/fundamentos-em-tic>