Com certeza! Segue um resumo da Unidade 1, Seções 1 e 2, com base nas informações fornecidas:

Unidade 1: Introdução à Engenharia de Software e à Análise de Sistemas

Esta unidade aborda a evolução do desenvolvimento de software, a importância da engenharia de software, o desenvolvimento ágil, e o papel do analista de sistemas, além de processos e modelos de software. O objetivo é capacitar o leitor a reconhecer e identificar processos de software e suas metodologias.

Seção 1: Fundamentos da Engenharia de Software

Esta seção introduz os fundamentos da engenharia de software, incluindo a natureza e evolução do software, a prática da engenharia de software, os princípios da análise de sistemas e o papel do analista de sistemas.

- **Definição e Natureza do Software:**
    - **Software** é definido como programas de computador com documentação associada, ou como um produto desenvolvido e suportado por profissionais de TI, abrangendo instruções, estruturas de dados e documentação descritiva.
    - Um **sistema** é um conjunto de componentes inter-relacionados (software, hardware e recursos humanos) que funcionam de forma unificada para atingir um objetivo comum, podendo incluir vários programas e arquivos de configuração.
    - O software, diferentemente do hardware, não se desgasta fisicamente, mas pode deteriorar-se com as mudanças implementadas ao longo do tempo, o que pode aumentar a taxa de defeitos.
- **Evolução do Software:** A cronologia mostra um **processo evolutivo** significativo:
    - **Década de 1940:** Programas tinham controle total sobre o computador.
    - **Décadas de 1950 e 1960:** Surgimento de sistemas operacionais e linguagens de programação como COBOL, LISP, ALGOL, BASIC.
    - **Décadas de 1960 e 1970:** Período da "crise do software" devido ao crescimento informal e falta de planejamento, levando à necessidade de métodos de engenharia de software; surgem conceitos de orientação a objetos e sistemas operacionais como MS-DOS e Macintosh.
    - **Décadas de 1980 e 1990:** Ascensão dos computadores pessoais (desktop), evolução da internet, surgimento do Linux e da linguagem JAVA.
    - **Década de 2000:** Sistemas operacionais gráficos (Windows XP, Vista, 7), e softwares baseados na web.
    - **Décadas de 2010 e 2020:** Ampla utilização de computação em nuvem, aplicativos móveis e inteligência artificial (Deep Learning, Machine Learning). As necessidades dos clientes evoluem rapidamente com a tecnologia.
- **Leis de Lehman da Evolução do Software (década de 1980, ainda aplicáveis):**

    - **Mudança contínua:** O software deve ser continuamente adaptado para manter a eficácia.
    - **Aumento da complexidade:** Cada mudança tende a aumentar a complexidade do software.
    - **Evolução (autorregulação):** O processo de evolução é autoajustável.
    - **Estabilidade organizacional:** A taxa de desenvolvimento tende a ser constante.
    - **Conservação da familiaridade:** A taxa de mudanças é proporcional ao domínio da equipe.
    - **Crescimento contínuo:** As funcionalidades aumentam com novas versões.
    - **Declínio da qualidade:** A qualidade diminui se não houver um processo de evolução.
    - **Sistema de feedback:** Processos de manutenção incluem feedback para melhorias.
- **Prática da Engenharia de Software:** É necessária para produzir softwares **confiáveis e econômicos**, e para **reutilizar** partes do software em outros sistemas. É uma tecnologia de quatro camadas que visa disciplina, adaptabilidade e agilidade:
    - **Foco na qualidade:** Objetivo final, permeia todas as etapas do desenvolvimento.
    - **Processo:** Base da engenharia de software, liga as camadas, garante desenvolvimento racional e dentro do prazo, e permite controle e gerenciamento de projetos.
    - **Métodos:** Fornecem informações técnicas para o desenvolvimento, incluindo comunicação, análise de requisitos, modelagem, codificação, testes e manutenção.
    - **Ferramentas:** Fornecem suporte automatizado ou semi-automatizado para processos e métodos, facilitando o desenvolvimento.
- **Categorias de Software:**
    - Software de sistema (compiladores, drivers).
    - Software de aplicação (planilhas, editores de texto).
    - Software de engenharia/científico (meteorologia, genética).
    - Software embarcado (forno micro-ondas, módulo de carros).
    - Software para linha de produtos (contábil, RH).
    - Software de aplicações web/aplicativos móveis.
    - Software de Inteligência Artificial (robótica, reconhecimento de padrões).
- **Softwares Legados:** Softwares antigos ainda em uso, mas com baixa qualidade, lentidão e funcionalidades defasadas. Sua documentação pode ser deficiente ou inexistente, tornando a manutenção cara e demorada. A evolução geralmente envolve a criação de um novo software com tecnologias mais recentes.
- **Princípios da Análise de Sistemas:** Baseiam-se em métodos e técnicas de investigação e especificação para encontrar a melhor solução computacional para problemas de negócio. As fases incluem:
    - **Análise:** Estudo de viabilidade, definição de funcionalidades e escopo, alocação de recursos e orçamento.
    - **Projeto:** Definição lógica do software, layouts de telas, estrutura de banco de dados e diagramas gráficos.
    - **Implementação:** Codificação do software.
    - **Testes:** Procura de erros e verificação de funcionalidades.

    - **Documentação:** Registro de todos os processos e diagramas, serve como ferramenta de comunicação e parte de contrato.
    - **Manutenção:** Acompanhamento pós-implantação para correção de falhas, melhorias ou novas funcionalidades.
    - A não adoção de técnicas de engenharia de software pode levar a problemas como manutenção onerosa, falta de padronização na documentação e previsões imprecisas de prazos e custos.
- **O Papel do Analista de Sistemas:** Profissional fundamental na engenharia de software, responsável por pesquisas, planejamentos, coordenação de equipes e recomendação de soluções.
    - Suas tarefas incluem descobrir o que o sistema deve fazer, entender as necessidades dos usuários, e organizar/especificar/documentar os requisitos.
    - Atua como "ponte" entre programadores e usuários finais, interpretando as necessidades dos usuários e repassando-as tecnicamente aos programadores.
    - Possui atribuições como interagir com clientes, analisar custos, levantar informações e requisitos, criar modelagens, orientar programadores, acompanhar testes, gerenciar mudanças, garantir qualidade, monitorar, planejar treinamentos, implantar software e oferecer consultoria técnica.

Seção 2: O Processo de Software

Esta seção aborda as atividades do Processo de Software, que é uma série de atividades relacionadas que resultam na produção de um software. Inclui a introdução ao Processo de Software, sua estrutura genérica, tarefas e modelagem das atividades, e a integração e validação entre elas.

- **Definição do Processo de Software:**
    - Um Processo de Software é um **conjunto de atividades e resultados relacionados que levam à produção de um software**.
    - Não é uma determinação rígida, mas uma **abordagem adaptável** que permite à equipe escolher os processos que melhor se encaixam na filosofia da empresa, focando na **qualidade, prazo e redução de custos**.
    - É uma **abordagem sistemática** usada pela Engenharia de Software para a produção de software.
- **Importância e Características do Processo de Software:**
    - **Padronização** na geração de serviços e produtos.
    - Permite a **reutilização** de partes já produzidas e padronizadas.
    - **Retém o conhecimento** na empresa, garantindo continuidade.
    - Define e guia as atividades de um Projeto de Software.
    - Permite a especificação de todo o processo de desenvolvimento.
    - Determina as tarefas para a equipe e individualmente.
    - **Reduz riscos**, tornando os resultados mais previsíveis.
    - Proporciona **visões comuns** para a equipe, facilitando a comunicação.
    - Pode ser usado como **template** para novos projetos, conferindo agilidade.

    - Define parâmetros como evento de início, matriz de responsabilidades, atividades e sequências, entradas e saídas, regras, infraestrutura e resultados.
- **Estrutura de um Processo Genérico de Software:**
    - Todos os processos utilizam uma estrutura genérica com atividades pré-estabelecidas.
    - **Quatro atividades fundamentais** (Sommerville, 2011):
        - **Especificação de software:** Definição do que será desenvolvido, restrições e funcionalidades.
        - **Projeto e Implementação de software:** Projeto e programação do software.
        - **Validação de software:** Verificação do atendimento às solicitações do cliente.
        - **Evolução de software:** Acompanhamento das alterações solicitadas pelo cliente.
    - **Cinco atividades metodológicas** (Pressman, 2016):
        - **Comunicação:** Entendimento dos objetivos e requisitos com os envolvidos.
        - **Planejamento:** Criação de um "mapa" do projeto, incluindo tarefas, riscos, recursos, produtos e cronograma.
        - **Modelagem:** Criação de diagramas para melhor entendimento e validação do software.
        - **Construção:** Codificação do software e testes de código.
        - **Entrega:** Entrega parcial ou total do software para feedback do cliente, com adaptações e correções.
- **Fluxos de Processo:** Descrevem como as atividades metodológicas são organizadas:
    - **Linear:** Atividades sequenciais (Comunicação, Planejamento, Modelagem, Construção, Entrega).
    - **Interativo:** Repetição de uma ou mais atividades antes de avançar.
    - **Evolucionário:** Atividades executadas em ciclos, gerando versões mais completas (incremental).
    - **Paralelo:** Duas ou mais atividades executadas simultaneamente (e.g., Comunicação e Análise).
    - Projetos diferentes exigem conjuntos de tarefas e modelagem de atividades diferenciados.
- **Modelagem dos Processos de Software:**
    - O Modelo de Processo de Software é uma **descrição simplificada do processo** que especifica atividades, produtos, papéis e oferece um roteiro.
    - Tem como objetivo propiciar **estabilidade, controle e organização** das atividades.
    - Uma das vantagens da modelagem é a possibilidade de **examinar e aprimorar** os processos antes de finalizar o produto.
- **Integração e Validação entre as Atividades do Processo de Software:**
    - Para garantir a qualidade e o atendimento dos prazos e funcionalidades, é necessário **estabelecer procedimentos e padrões de qualidade** durante os processos.
    - Abordagens de avaliação e aperfeiçoamento incluem:

- **SCAMPI (Standard CMMI Appraisal Method for Process Improvement):** Modelo de avaliação do processo em cinco etapas para garantir objetividade na classificação das avaliações. O CMMI é um conjunto de práticas para aumentar a maturidade organizacional na área de TI.
- **CBA IPI (CMM Based Appraisal for Internal Process Improvement):** Técnica de diagnóstico para avaliar a maturidade de uma organização de software, identificando pontos fortes e fracos.
- **SPICE (ISO/IEC 15504):** Padrão que define requisitos para avaliação do Processo de Software, auxiliando na avaliação objetiva da eficácia.
- **ISO 9001:2000 para software:** Padrão genérico de qualidade aplicável a qualquer organização, com modelos de referência como ISO/IEC 9126 e ISO 9000-3 para software.

Esta é uma visão geral abrangente das Seções 1 e 2 da Unidade 1, com foco nos conceitos chave e sua relevância no desenvolvimento de software.

Oi! Que legal que você quer ver como esses conceitos de Engenharia de Software podem ser "rodados" em um ambiente como o Google Colab!

O resumo que fiz anteriormente aborda principalmente **conceitos teóricos e metodologias** de desenvolvimento de software, como a natureza e evolução do software, o papel do analista de sistemas e os modelos de processo (cascata, ágil, Scrum). Essas informações são sobre *como* o software é pensado, planejado e construído, e não contêm código-fonte direto que possa ser executado.

No entanto, posso te dar um **exemplo conceitual muito simplificado** de como você poderia usar o Python no Google Colab para *ilustrar* um dos modelos que estudamos: o **Scrum** e o seu **Quadro Kanban**. Isso vai te ajudar a visualizar de forma prática a gestão de tarefas e a evolução de um projeto, algo que o Analista de Sistemas faria no dia a dia.

Imagine que você está usando o Colab não para escrever o *código do software em si*, mas para **gerenciar as tarefas** de desenvolvimento desse software, como se fosse um quadro branco digital.

Exemplo Conceitual: Simulação de um Quadro Scrum (Kanban) no Google Colab

Este script em Python simula as categorias principais de um quadro Scrum ("A Fazer", "Em Andamento", "Concluído") e permite "mover" tarefas entre elas, demonstrando o fluxo de trabalho incremental e iterativo. Ele é uma representação **abstrata e didática**, não um sistema de gerenciamento de projetos completo.

```
# @title Simulação de Quadro Scrum Simplificado
# Este é um exemplo conceitual, não um software de gerenciamento de projetos completo.
# Ele ilustra como as tarefas podem ser organizadas e 'movidas' através das fases,
# inspirando-se nos princípios do Scrum e do Kanban, conforme abordado na Unidade 1.
```

```
print("--- Simulação de um Quadro Scrum (Kanban) no Google Colab ---")
print("O objetivo é visualizar a gestão das atividades de desenvolvimento de software.")
print("No contexto do Scrum, as tarefas seriam gerenciadas em 'Sprints' e as 'Histórias'
seriam os itens do 'Backlog'.")
print("-" * 70)

# 1. Definir o 'Product Backlog' e as categorias do Quadro Scrum
# Aqui, representamos as tarefas iniciais (Product Backlog) distribuídas em categorias.
# O 'Backlog' é uma lista com prioridades dos requisitos do projeto.
# As categorias do quadro (To Do, In Progress, Done) são mencionadas como 'Stories', 'To
Do', 'In Progress', 'Testing', 'Done' no Quadro Scrum.

quadro_scrum = {
    "A FAZER": [
        "Gerar Interfaces do Site (Wireframes)",  # Exemplo de tarefa do quadro
        "Definir Paleta de Cores do Site",        # Exemplo de tarefa do quadro
        "Definir Estrutura do Banco de Dados",    # Exemplo de tarefa do quadro
        "Desenvolver módulo de autenticação de usuários",
        "Implementar funcionalidade de aluguel de brinquedos (aplicativo)",
        "Configurar ambiente de nuvem para o banco de dados"
    ],
    "EM ANDAMENTO": [],
    "CONCLUÍDO": []
}

def exibir_quadro():
    """Função para exibir o estado atual do nosso quadro Scrum."""
    print("\n--- Estado Atual do Quadro Scrum ---")
    for categoria, tarefas in quadro_scrum.items():
        print(f"\n**{categoria}:**")
        if tarefas:
            for i, tarefa in enumerate(tarefas):
                print(f"  {i+1}. {tarefa}")
        else:
            print("  (Nenhuma tarefa nesta categoria no momento)")
    print("-" * 70)

def mover_tarefa(indice_tarefa, categoria_origem, categoria_destino):
    """Função para mover uma tarefa de uma categoria para outra, simulando o fluxo de
trabalho."""
    if categoria_origem not in quadro_scrum or categoria_destino not in quadro_scrum:
        print("Erro: Categoria de origem ou destino inválida.")
        return False
    if not (0 <= indice_tarefa < len(quadro_scrum[categoria_origem])):
        print("Erro: Índice de tarefa inválido na categoria de origem. Verifique o número da
tarefa.")
        return False
```

```
    tarefa_movida = quadro_scrum[categoria_origem].pop(indice_tarefa)
    quadro_scrum[categoria_destino].append(tarefa_movida)
    print(f"✅ Tarefa '{tarefa_movida}' movida de **'{categoria_origem}'** para
**'{categoria_destino}'**.")
    return True

# --- Demonstração do Fluxo de Trabalho ---

# 1. Exibir o quadro inicial (todas as tarefas em 'A FAZER')
exibir_quadro()

# 2. Simular o Analista de Sistemas ou a equipe iniciando o trabalho (movendo tarefas para
'EM ANDAMENTO')
print("\n--- Analista de Sistemas e equipe iniciando o trabalho ---")
# O Analista de Sistemas atua como 'ponte' entre programadores e usuários,
# organizando e especificando requisitos e gerenciando mudanças.
# Ele também estaria envolvido no 'Planejamento' do Scrum.

mover_tarefa(0, "A FAZER", "EM ANDAMENTO") # 'Gerar Interfaces do Site...'
mover_tarefa(0, "A FAZER", "EM ANDAMENTO") # 'Definir Paleta de Cores...'

exibir_quadro()

# 3. Simular uma tarefa sendo concluída (movendo para 'CONCLUÍDO')
print("\n--- Equipe concluindo uma tarefa (parte da 'Construção' e 'Entrega' do software) ---")
# A fase de 'Construção' envolve a codificação e testes.
# A 'Entrega' pode ser parcial para feedback do cliente.

mover_tarefa(0, "EM ANDAMENTO", "CONCLUÍDO") # 'Gerar Interfaces do Site...'

exibir_quadro()

# 4. Simular uma nova necessidade do cliente (evolução do software)
print("\n--- Nova funcionalidade solicitada pelo cliente (Evolução do Software) ---")
# O software deve ser continuamente adaptado para manter a eficácia.
# Os requisitos podem ser alterados, e o projeto deve ser flexível para aceitar as mudanças.

nova_tarefa = "Integrar sistema de pagamento online"
quadro_scrum["A FAZER"].append(nova_tarefa)
print(f"📝 Nova tarefa adicionada ao 'A FAZER': **'{nova_tarefa}'**.")

exibir_quadro()

# 5. Mais uma tarefa iniciada e concluída
print("\n--- Mais progressos no desenvolvimento ---")
mover_tarefa(0, "A FAZER", "EM ANDAMENTO") # 'Definir Estrutura do Banco de Dados'
mover_tarefa(0, "EM ANDAMENTO", "CONCLUÍDO") # 'Definir Paleta de Cores...'
```

```
exibir_quadro()

print("\n--- Fim da Simulação Conceitual ---")
print("Este exercício simples mostra como os princípios de organização de tarefas,")
print("fluxo de trabalho e adaptação a novas necessidades podem ser representados.")
print("O Analista de Sistemas estaria constantemente revisando o 'Backlog',")
print("participando das 'Reuniões Scrum' e garantindo a 'Evolução do software'.")
```

Como rodar este exemplo no Google Colab:

1. **Abra o Google Colab:** Vá para https://colab.research.google.com/ e clique em "New notebook" (Novo notebook).
2. **Cole o Código:** Copie o código Python acima e cole-o na primeira célula do seu notebook Colab.
3. **Execute a Célula:** Clique no ícone de "Play" (um triângulo) que aparece no canto superior esquerdo da célula, ou pressione `Shift + Enter`.

Você verá o texto de saída no console do Colab, simulando as etapas de um quadro Scrum. Este é um exemplo **puramente ilustrativo** dos conceitos, e não de um software funcional em si. A beleza da Engenharia de Software e da Análise de Sistemas reside justamente em definir e organizar esses processos para que o desenvolvimento real do código seja eficiente e atenda às necessidades do cliente.

Engenharia de Software e Análise de Sistemas: Fundamentos, Processos e Metodologias para o Desenvolvimento de Soluções Inovadoras

PALAVRAS-CHAVE: Engenharia de Software; Análise de Sistemas; Processos de Software; Modelos Prescritivos; Desenvolvimento Ágil; Scrum.

1. INTRODUÇÃO

A tecnologia tem se integrado cada vez mais ao estilo de vida moderno, com inovações como computação em nuvem, realidades virtual e aumentada, e reconhecimento facial. Nesse cenário, o **software** desponta como uma inovação tecnológica de extrema importância, proporcionando eficiência e agilidade quando combinado com o hardware. A crescente demanda por softwares eficientes e inovadores torna a mão de obra especializada uma necessidade premente no mercado. Diante disso, o conhecimento em **Análise e Modelagem de Sistemas** torna-se essencial para a criação eficaz de novos sistemas.

A Engenharia de Software, área que se preocupa com todos os aspectos da produção de software, torna-se necessária para produzir softwares mais confiáveis e econômicos. Este

artigo tem como **objetivo** apresentar uma visão abrangente sobre os fundamentos da Engenharia de Software e da Análise de Sistemas, explorando a natureza e evolução do software, o papel do analista de sistemas, os processos e modelos de software, e as metodologias ágeis, com foco no Scrum. Busca-se capacitar o leitor a reconhecer e identificar processos de software e suas metodologias, conforme o propósito da Unidade 1 da disciplina.

2. OBJETIVO

O objetivo deste artigo é **sintetizar e apresentar os conceitos fundamentais da Engenharia de Software e da Análise de Sistemas**, incluindo a definição, natureza e evolução do software, os princípios e a prática da engenharia de software, o papel do analista de sistemas, e a discussão dos processos e modelos de software, com ênfase nas metodologias prescritivas e ágeis.

3. MATERIAL E MÉTODOS

O presente trabalho constitui-se em uma **revisão bibliográfica** baseada nos excertos fornecidos de materiais didáticos da Unidade 1 – Seções 1, 2 e 3 de "Introdução à Engenharia de Software e à Análise de Sistemas", e de três exemplos de projetos e um regulamento de evento acadêmico. A metodologia empregada consistiu na **análise e síntese do conteúdo** presente nas fontes para construir uma narrativa coerente sobre os tópicos da Engenharia de Software e Análise de Sistemas, priorizando as definições, conceitos e classificações apresentadas pelos autores Sommerville (2011) e Pressman (2016). Os projetos e o regulamento do evento foram consultados para ilustrar a aplicação de princípios da análise e relevância do conhecimento na prática.

4. RESULTADOS E DISCUSSÃO

4.1. Fundamentos da Engenharia de Software

O **software** é definido como programas de computador com documentação associada, ou como um produto desenvolvido e suportado por profissionais de TI, abrangendo instruções, estruturas de dados e documentação descritiva. As **instruções** fornecem atributos e funções, as **estruturas de dados** manipulam informações, e a **documentação** descreve a operação do programa. Um **sistema**, por sua vez, é um conjunto de componentes inter-relacionados (software, hardware e recursos humanos) que atuam de forma unificada para um objetivo comum.

A **evolução do software** é um processo significativo. Na década de 1940, os programas controlavam totalmente o computador. As décadas de 1950 e 1960 viram o surgimento de sistemas operacionais e linguagens como COBOL e BASIC. As décadas de 1960 e 1970 foram marcadas pela "crise do software", que impulsionou a necessidade de métodos de engenharia de software devido ao desenvolvimento informal e problemas de planejamento. Nas décadas seguintes, houve a ascensão dos computadores pessoais, a evolução da internet, o surgimento do Linux e Java, e softwares baseados na web. A partir de 2010 e 2020, a computação em nuvem, aplicativos móveis e **Inteligência Artificial (IA)**, como

Deep Learning e Machine Learning, tornaram-se amplamente utilizados, e as necessidades dos clientes evoluem rapidamente.

Diferentemente do hardware, o software não se desgasta fisicamente, mas pode **deteriorar-se** com as mudanças implementadas ao longo do tempo, o que pode aumentar a taxa de defeitos. As **Leis de Lehman da Evolução do Software** (década de 1980, ainda aplicáveis) destacam que o software deve ser continuamente adaptado para manter a eficácia, a complexidade tende a aumentar a cada mudança, e a qualidade diminui se não houver um processo de evolução.

A **prática da Engenharia de Software** é essencial para produzir softwares confiáveis e econômicos, e para permitir a reutilização de partes do software. Ela é uma tecnologia de quatro camadas, focando na disciplina, adaptabilidade e agilidade:

- **Foco na Qualidade**: Objetivo final, permeando todas as etapas do desenvolvimento.
- **Processo**: Base da engenharia de software, ligando as camadas e garantindo o desenvolvimento racional e dentro do prazo.
- **Métodos**: Fornecem informações técnicas para o desenvolvimento, incluindo comunicação, análise de requisitos, modelagem, codificação, testes e manutenção.
- **Ferramentas**: Oferecem suporte automatizado ou semi-automatizado para processos e métodos.

Categorias de software incluem software de sistema, aplicação, engenharia/científico, embarcado, para linha de produtos, aplicações web/móveis e Inteligência Artificial. Os **softwares legados** são sistemas antigos ainda em uso, mas com baixa qualidade, lentidão e funcionalidades defasadas, frequentemente com documentação deficiente, tornando a manutenção cara e demorada. A evolução desses softwares geralmente implica na criação de um novo sistema com tecnologias mais recentes.

Os **princípios da Análise de Sistemas** baseiam-se em métodos e técnicas de investigação e especificação para encontrar a melhor solução computacional para problemas de negócio. As fases de um processo de análise de sistema incluem:

- **Análise**: Estudo de viabilidade, definição de funcionalidades e escopo, alocação de recursos e orçamento.
- **Projeto**: Definição lógica do software, layouts de telas, estrutura de banco de dados e diagramas gráficos.
- **Implementação**: Codificação do software.
- **Testes**: Procura de erros e verificação de funcionalidades.
- **Documentação**: Registro de todos os processos e diagramas, servindo como ferramenta de comunicação e parte do contrato.
- **Manutenção**: Acompanhamento pós-implantação para correção de falhas, melhorias ou novas funcionalidades.

A não adoção de técnicas de engenharia de software pode levar a problemas como manutenção onerosa, falta de padronização na documentação e previsões imprecisas de prazos e custos.

4.2. O Papel do Analista de Sistemas

O **Analista de Sistemas** é um profissional fundamental na engenharia de software. Ele atua como a "**ponte**" entre os programadores e os usuários finais, interpretando as necessidades dos usuários e as repassando tecnicamente aos programadores. Suas tarefas incluem descobrir o que o sistema deve fazer, entender as necessidades dos usuários, e organizar/especificar/documentar os requisitos.

As atribuições do analista de sistemas são amplas e englobam:

- Interagir com clientes e usuários finais.
- Analisar custos e verificar a viabilidade do projeto.
- Levantar informações e requisitos.
- Criar a modelagem do software.
- Orientar programadores e acompanhar o desenvolvimento.
- Acompanhar e executar testes.
- Preparar toda a documentação do software.
- Gerenciar eventuais mudanças no projeto.
- Determinar padrões de desenvolvimento.
- Garantir a qualidade final do software.
- Realizar o monitoramento e auditorias.
- Planejar e aplicar treinamentos.
- Implantar o software desenvolvido.
- Proporcionar consultoria técnica.
- Pesquisar novas tecnologias e fornecedores.

Este profissional precisa ter uma boa visão empresarial e conhecimento tecnológico atualizado.

4.3. O Processo de Software

Um **Processo de Software** é um conjunto de atividades e resultados relacionados que levam à produção de um software. Não é uma determinação rígida, mas uma abordagem adaptável que permite à equipe escolher os processos que melhor se encaixam na filosofia da empresa, focando na qualidade, prazo e redução de custos.

A importância e as características do Processo de Software incluem:

- **Padronização** na geração de serviços e produtos.
- Permite a **reutilização** de partes já produzidas e padronizadas.
- **Retém o conhecimento** na empresa, garantindo continuidade.
- Define e guia as atividades de um Projeto de Software.
- Permite a especificação de todo o processo de desenvolvimento.
- Determina as tarefas para a equipe e individualmente.
- **Reduz riscos**, tornando os resultados mais previsíveis.
- Proporciona **visões comuns** para a equipe, facilitando a comunicação.
- Pode ser usado como **template** para novos projetos, conferindo agilidade.

Todos os processos utilizam uma **estrutura genérica** com atividades pré-estabelecidas. Sommerville (2011) cita quatro atividades fundamentais:

- **Especificação de software**: Definição do que será desenvolvido, restrições e funcionalidades.
- **Projeto e Implementação de software**: Projeto e programação do software.
- **Validação de software**: Verificação do atendimento às solicitações do cliente.
- **Evolução de software**: Acompanhamento das alterações solicitadas pelo cliente.

Pressman (2016) apresenta cinco atividades metodológicas:

- **Comunicação**: Entendimento dos objetivos e requisitos com os envolvidos.
- **Planejamento**: Criação de um "mapa" do projeto (tarefas, riscos, recursos, cronograma).
- **Modelagem**: Criação de diagramas para melhor entendimento e validação do software.
- **Construção**: Codificação do software e testes de código.
- **Entrega**: Entrega parcial ou total do software para feedback do cliente.

Os **Fluxos de Processo** descrevem como as atividades metodológicas são organizadas, podendo ser lineares, interativos, evolucionários ou paralelos. A **modelagem dos processos de software** é crucial para propiciar estabilidade, controle e organização das atividades, permitindo examinar e aprimorar os processos antes de finalizar o produto. Para garantir a qualidade e o cumprimento de prazos, são necessárias avaliações e aperfeiçoamentos, como SCAMPI, CBA IPI, SPICE e ISO 9001:2000.

4.4. Modelos de Processos de Software

Os **Modelos de Processos de Software** são guias que definem o fluxo de atividades, a interação entre elas e os artefatos produzidos. Existem diferentes tipos:

4.4.1. Modelos de Processos Prescritivos (Tradicionais)

Caracterizam-se por tarefas sequenciais com diretrizes bem definidas. Incluem:

- **Modelo Cascata**: Abordagem sistemática e sequencial, onde cada fase (Análise e Especificação, Projeto, Implementação e Teste de Unidade, Integração e Verificação, Operação e Manutenção) inicia somente após a conclusão da anterior. É simples de gerenciar, mas pode ser demorado, e o cliente só vê o produto final no final, o que pode gerar insatisfação.
- **Modelo Incremental**: Cria pequenas versões (incrementos) do software, que são entregues ao cliente e expandidas em novas versões. Cada incremento incorpora parte da funcionalidade e passa por um ciclo completo de desenvolvimento.
- **Modelos Evolucionários**: Produzem versões cada vez mais completas do software, adaptando-se a requisitos de negócio que mudam. Exemplos são:
    - **Prototipação**: Começa com a comunicação de objetivos e funcionalidades, um protótipo é construído rapidamente e o cliente oferece feedback para aprimoramento.
    - **Espiral**: Iterativo como a prototipação, mas incorpora aspectos sistemáticos do Modelo Cascata, visando rápido desenvolvimento de versões mais completas a cada ciclo.

- **Modelos Concorrentes**: Representados por séries de tarefas e estados, usados em projetos com diferentes equipes. Não seguem uma sequência rígida, mas estabelecem uma rede de atividades que se movimentam.

4.4.2. Modelos de Processos Especializados

Utilizam características de modelos prescritivos para abordagens mais especializadas. Exemplos são:

- **Modelo Baseado em Componentes**: Reutiliza componentes de software previamente empacotados.
- **Modelo de Métodos Formais**: Envolve especificação matemática formal do software para descobrir e eliminar problemas como ambiguidade.
- **Desenvolvimento de Software Orientado a Aspecto**: Separa o código do software por importância e modela requisitos transcendendo funcionalidades.
- **Modelo de Processo Unificado (RUP)**: Combina características dos modelos tradicionais com princípios ágeis, sendo iterativo e incremental.
- **Modelos de Processos Pessoal e de Equipe**: Focam na medição pessoal da produção e na criação de equipes autodirigidas e de alta qualidade.

4.4.3. Modelo de Desenvolvimento Ágil

Surgiu como resposta à necessidade de maior velocidade no desenvolvimento de software e aplicativos. O desenvolvimento ágil busca resolver problemas da engenharia de software tradicional, oferecendo flexibilidade e dinamismo. Seus princípios incluem:

- **Envolvimento do Cliente**: Forte participação do cliente no processo.
- **Entrega Incremental**: Software desenvolvido em incrementos, com cliente indicando novos requisitos.
- **Pessoas e Não Processos**: Liberdade para a equipe explorar ao máximo sua capacidade, sem prisões a processos prescritivos.
- **Aceitar as Mudanças**: O projeto deve ser flexível para aceitar alterações nos requisitos.
- **Manter a Simplicidade**: Banir a complexidade, trabalhando de forma simples e ativa.
- Reduzir drasticamente a documentação e a burocracia.

Dois métodos ágeis se destacam:

- **XP (Extreme Programming)**: Feedback constante, abordagem incremental, comunicação primordial. Possui atividades de Planejamento, Projeto, Codificação e Testes. O cliente participa ativamente da equipe.
- **Scrum**: Processo de desenvolvimento iterativo e incremental, utilizado em processos gerenciais. Define um conjunto de regras e práticas de gestão, focando em trabalho em equipe e comunicação melhorada. Suas atividades incluem:
    - **Backlog**: Lista priorizada de requisitos (funcionalidades) do projeto, que pode ser alterada a qualquer momento.
    - **Sprints**: Unidades de trabalho para atingir um requisito do Backlog, com prazos definidos (Time Box).

- **Reuniões Scrum (Daily Meeting)**: Reuniões diárias de 15 minutos para acompanhar o progresso e identificar obstáculos.
- **Histórias**: Ideias e funcionalidades iniciais do produto, que formam o Product Backlog.
- **Quadro Scrum (Kanban)**: Ferramenta visual para acompanhar as tarefas, com categorias como "Stories", "A Fazer" (To Do), "Em Andamento" (In Progress), "Testando" (Testing) e "Concluído" (Done).

A metodologia Scrum seria uma abordagem possível para projetos complexos, como um site e aplicativo para uma loja de brinquedos online, permitindo entregas incrementais a cada Sprint e participação ativa do cliente.

5. CONCLUSÃO

O campo da Engenharia de Software e Análise de Sistemas é dinâmico e essencial para o desenvolvimento de soluções tecnológicas que atendam às crescentes demandas da sociedade. A compreensão da natureza evolutiva do software, das leis que governam sua mudança e das práticas da engenharia é fundamental para criar produtos confiáveis e eficientes. O Analista de Sistemas emerge como um profissional-chave, atuando como elo entre usuários e desenvolvedores, e garantindo que os requisitos sejam compreendidos e transformados em funcionalidades.

A adoção de processos de software bem definidos, sejam eles prescritivos ou ágeis, é crucial para a padronização, redução de riscos, retenção de conhecimento e agilidade no desenvolvimento. Modelos como o Cascata oferecem clareza e controle, enquanto metodologias ágeis como o Scrum proporcionam flexibilidade, entregas incrementais e maior satisfação do cliente, características cada vez mais valorizadas em um mercado de tecnologia em constante mutação. A capacidade de escolher e adaptar o modelo de processo mais adequado ao projeto e à equipe é um diferencial para os profissionais da área.

6. REFERÊNCIAS

- MATURANA, C. S. et al. A Importância da Ludoterapia na Conduta Terapêutica: Revisão Bibliográfica. In: **4º Encontro de Atividades Científicas – Unopar**, 20XX.
- [Documento sem título (1).pdf]
- WERLICH, C. **Introdução à Engenharia de Software e à Análise de Sistemas**. Londrina: Editora e Distribuidora Educacional S.A., 2020.
- PINTO, M. L. et al. Coração Feliz Fazendo Arte. In: **4º Encontro de Atividades Científicas – Unopar**, 20XX.
- CERNEV, F. K.; STEFANO, S. R. “Sonata Eventos”: Um Processo de Criação de Empresa. In: **4º Encontro de Atividades Científicas – Unopar**, 20XX.
- [bef85589-06ef-4edd-bb2d-978672abe60c.pdf]


```
**A FAZER:**
  1. Definir Estrutura do Banco de Dados
  2. Desenvolver módulo de autenticação de usuários
  3. Implementar funcionalidade de aluguel de brinquedos (aplicativo)
```

```
  4. Configurar ambiente de nuvem para o banco de dados

**EM ANDAMENTO:**
  1. Definir Paleta de Cores do Site

**CONCLUÍDO:**
  1. Gerar Interfaces do Site (Wireframes)
--------------------------------------------------------------------------

--- Nova funcionalidade solicitada pelo cliente (Evolução do Software)
---
📝 Nova tarefa adicionada ao 'A FAZER': **'Integrar sistema de
pagamento online'**.

--- Estado Atual do Quadro Scrum ---

**A FAZER:**
  1. Definir Estrutura do Banco de Dados
  2. Desenvolver módulo de autenticação de usuários
  3. Implementar funcionalidade de aluguel de brinquedos (aplicativo)
  4. Configurar ambiente de nuvem para o banco de dados
  5. Integrar sistema de pagamento online

**EM ANDAMENTO:**
  1. Definir Paleta de Cores do Site

**CONCLUÍDO:**
  1. Gerar Interfaces do Site (Wireframes)
--------------------------------------------------------------------------

--- Mais progressos no desenvolvimento ---
✅ Tarefa 'Definir Estrutura do Banco de Dados' movida de **'A FAZER'**
para **'EM ANDAMENTO'**.
✅ Tarefa 'Definir Paleta de Cores do Site' movida de **'EM
ANDAMENTO'** para **'CONCLUÍDO'**.

--- Estado Atual do Quadro Scrum ---

**A FAZER:**
  1. Desenvolver módulo de autenticação de usuários
  2. Implementar funcionalidade de aluguel de brinquedos (aplicativo)
  3. Configurar ambiente de nuvem para o banco de dados
  4. Integrar sistema de pagamento online

**EM ANDAMENTO:**
  1. Definir Estrutura do Banco de Dados

**CONCLUÍDO:**
  1. Gerar Interfaces do Site (Wireframes)
  2. Definir Paleta de Cores do Site
--------------------------------------------------------------------------
```

```
--- Fim da Simulação Conceitual ---
Este exercício simples mostra como os princípios de organização de tarefas,
fluxo de trabalho e adaptação a novas necessidades podem ser representados.
O Analista de Sistemas estaria constantemente revisando o 'Backlog',
participando das 'Reuniões Scrum' e garantindo a 'Evolução do software'.
```