**Centro Universitário 7 de Setembro**

**Análise e Projeto de Sistemas 1**

**Projeto SCRUM**

Professor: Franciso Araújo

Equipe:

Lázaro Sobreira Rodrigues

Matheus Martins Lira

André Minerva

**INTRODUÇÃO**

O desenvolvimento de um Sistema de Gerenciamento de Projetos Ágeis (SGPA), utilizando a metodologia Scrum e técnicas de modelagem com UML (Unified Modeling Language), proporcionou uma visão prática de como essas ferramentas podem ser integradas em um ambiente de desenvolvimento de software. Este documento discute como a UML auxiliou no design do sistema, os desafios enfrentados na implementação do Scrum e reflete sobre possíveis expansões e melhorias no projeto.

**DIAGRAMAS UML**

1. **Diagrama de Casos de Uso**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

1. **Diagrama Classes**

Diagrama

Descrição gerada automaticamente

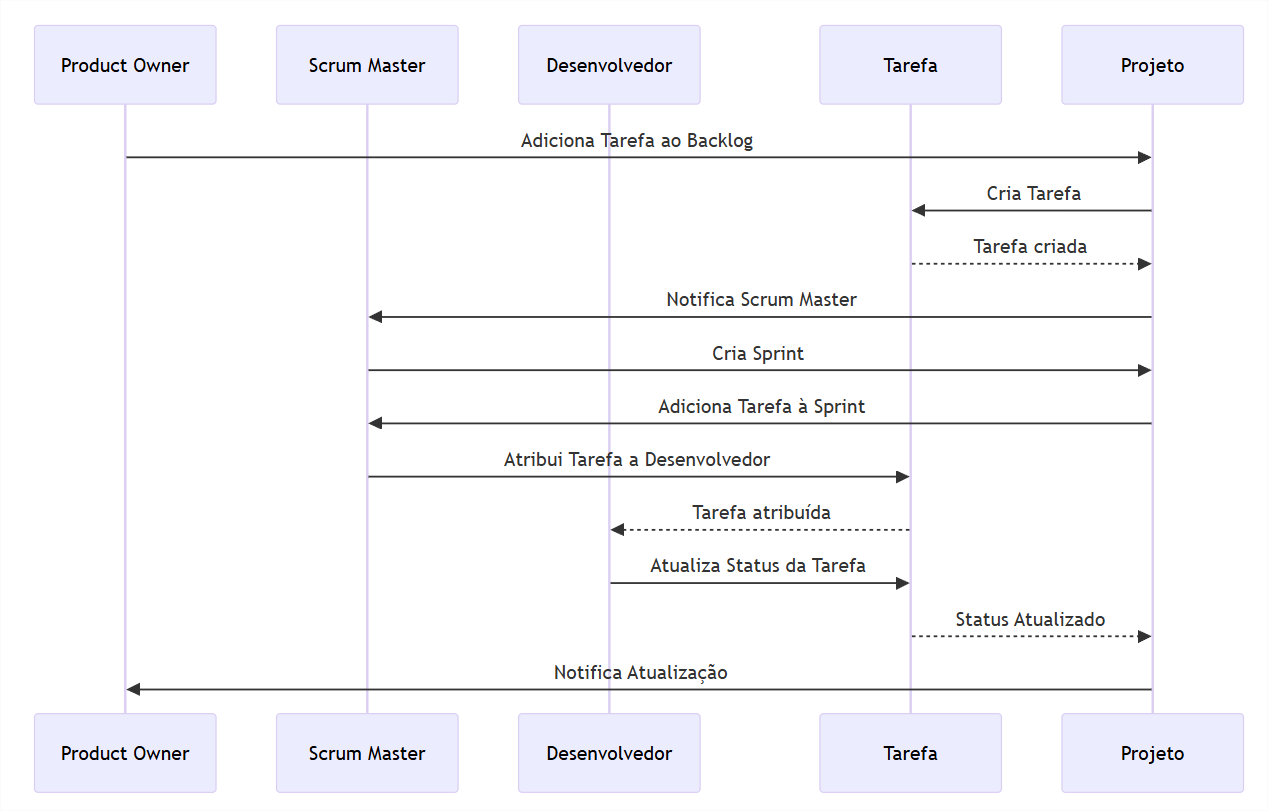
1. Uma imagem contendo Diagrama

   Descrição gerada automaticamente**Diagrama de Atividades**

Diagrama

Descrição gerada automaticamente

1. **Diagrama de Sequência**



**IMPLEMENTAÇÃO EM PYTHON**

import random  
from datetime import datetime, timedelta  
  
  
# Classes de Usuário e Subclasses  
class Usuario:  
 def \_\_init\_\_(self, nome, email, papel):  
 self.nome = nome  
 self.email = email  
 self.papel = papel  
  
  
class Desenvolvedor(Usuario):  
 def \_\_init\_\_(self, nome, email):  
 super().\_\_init\_\_(nome, email, "Desenvolvedor")  
  
  
class ScrumMaster(Usuario):  
 def \_\_init\_\_(self, nome, email):  
 super().\_\_init\_\_(nome, email, "Scrum Master")  
  
  
class ProductOwner(Usuario):  
 def \_\_init\_\_(self, nome, email):  
 super().\_\_init\_\_(nome, email, "Product Owner")  
  
  
# Classe Tarefa  
class Tarefa:  
 def \_\_init\_\_(self, descricao, estimativa\_horas):  
 self.descricao = descricao  
 self.status = "Pendente"  
 self.responsavel = None  
 self.estimativa\_horas = estimativa\_horas # Estimativa de horas para completar a tarefa  
 self.horas\_trabalhadas = 0 # Horas trabalhadas na tarefa  
  
 def atribuir\_responsavel(self, usuario):  
 self.responsavel = usuario  
  
 def atualizar\_status(self, status):  
 self.status = status  
  
 def registrar\_trabalho(self, horas):  
 self.horas\_trabalhadas += horas  
 if self.horas\_trabalhadas >= self.estimativa\_horas:  
 self.atualizar\_status("Concluído")  
  
  
# Classe Backlog  
class Backlog:  
 def \_\_init\_\_(self):  
 self.tarefas = []  
  
 def adicionar\_tarefa(self, tarefa):  
 self.tarefas.append(tarefa)  
  
 def remover\_tarefa(self, tarefa):  
 self.tarefas.remove(tarefa)  
  
 def exibir\_tarefas\_pendentes(self):  
 print("\nBacklog de Tarefas Pendente:")  
 for tarefa in self.tarefas:  
 if tarefa.status == "Pendente":  
 print(f"Tarefa: {tarefa.descricao}, Estimativa: {tarefa.estimativa\_horas}h")  
  
  
# Classe Sprint  
class Sprint:  
 def \_\_init\_\_(self, numero, duracao\_dias, scrum\_master, product\_owner):  
 self.numero = numero  
 self.tarefas = []  
 self.scrum\_master = scrum\_master  
 self.product\_owner = product\_owner  
 self.data\_inicio = datetime.now()  
 self.data\_fim = self.data\_inicio + timedelta(days=duracao\_dias)  
 self.duracao\_dias = duracao\_dias  
  
 def adicionar\_tarefa(self, tarefa):  
 self.tarefas.append(tarefa)  
  
 def tempo\_restante(self):  
 dias\_restantes = (self.data\_fim - datetime.now()).days  
 return dias\_restantes if dias\_restantes > 0 else 0  
  
  
# Classe Projeto  
class Projeto:  
 def \_\_init\_\_(self, nome):  
 self.nome = nome  
 self.sprints = []  
 self.membros = []  
 self.backlog = Backlog()  
  
 def criar\_sprint(self, numero, duracao\_dias, scrum\_master, product\_owner):  
 sprint = Sprint(numero, duracao\_dias, scrum\_master, product\_owner)  
 self.sprints.append(sprint)  
 return sprint  
  
 def adicionar\_membro(self, usuario):  
 self.membros.append(usuario)  
  
 def gerar\_relatorio\_progresso(self):  
 print(f"\nRelatório de Progresso do Projeto: {self.nome}")  
 for sprint in self.sprints:  
 print(f"\nSprint {sprint.numero} (Fim em {sprint.data\_fim.strftime('%d/%m/%Y')}):")  
 print(f"Scrum Master: {sprint.scrum\_master.nome} | Product Owner: {sprint.product\_owner.nome}")  
 for tarefa in sprint.tarefas:  
 status = f"{tarefa.status} (Responsável: {tarefa.responsavel.nome})" if tarefa.responsavel else "Não atribuído"  
 print(  
 f"Tarefa: {tarefa.descricao} - Status: {status} - Horas trabalhadas: {tarefa.horas\_trabalhadas}/{tarefa.estimativa\_horas}")  
  
 self.backlog.exibir\_tarefas\_pendentes()  
  
  
# Função para distribuir tarefas e atualizar status  
def distribuir\_tarefas\_e\_atualizar\_status(projeto, tarefas\_por\_sprint):  
 membros\_desenvolvedores = [m for m in projeto.membros if isinstance(m, Desenvolvedor)]  
  
 for sprint in projeto.sprints:  
 tarefas = tarefas\_por\_sprint[sprint.numero - 1]  
 for tarefa in tarefas:  
 # Atribuir uma tarefa a um desenvolvedor aleatório  
 responsavel = random.choice(membros\_desenvolvedores)  
 tarefa.atribuir\_responsavel(responsavel)  
  
 # Distribuir status variado para as tarefas  
 status\_opcoes = ["Pendente", "Em progresso", "Concluído"]  
 status\_atualizado = random.choices(status\_opcoes, weights=[2, 3, 1], k=1)[  
 0] # Mais chance de "Em progresso"  
 tarefa.atualizar\_status(status\_atualizado)  
  
 # Registrar horas aleatórias de trabalho, mas só para as tarefas em progresso ou concluídas  
 if status\_atualizado != "Pendente":  
 horas\_trabalhadas = random.randint(0, tarefa.estimativa\_horas)  
 tarefa.registrar\_trabalho(horas\_trabalhadas)  
  
  
# Simulação do projeto  
def simular\_projeto\_scrum():  
 # Criar o projeto  
 projeto = Projeto("Sistema de Gerenciamento de Projetos Ágeis")  
  
 # Criar membros da equipe  
 desenvolvedor1 = Desenvolvedor("Alice", "alice@empresa.com")  
 desenvolvedor2 = Desenvolvedor("Bob", "bob@empresa.com")  
 desenvolvedor3 = Desenvolvedor("Charlie", "charlie@empresa.com")  
 scrum\_master = ScrumMaster("Carlos", "carlos@empresa.com")  
 product\_owner = ProductOwner("Lucas", "lucas@empresa.com")  
  
 # Adicionar membros ao projeto  
 projeto.adicionar\_membro(desenvolvedor1)  
 projeto.adicionar\_membro(desenvolvedor2)  
 projeto.adicionar\_membro(desenvolvedor3)  
 projeto.adicionar\_membro(scrum\_master)  
 projeto.adicionar\_membro(product\_owner)  
  
 # Adicionar tarefas ao backlog  
 todas\_tarefas = [  
 Tarefa(f"Desenvolver funcionalidade {i}", random.randint(5, 20)) for i in range(1, 21)  
 ]  
 for tarefa in todas\_tarefas:  
 projeto.backlog.adicionar\_tarefa(tarefa)  
  
 # Criar sprints e distribuir tarefas  
 sprint1 = projeto.criar\_sprint(1, 14, scrum\_master, product\_owner) # Sprint de 14 dias  
 sprint2 = projeto.criar\_sprint(2, 14, scrum\_master, product\_owner) # Sprint de 14 dias  
 sprint3 = projeto.criar\_sprint(3, 14, scrum\_master, product\_owner) # Sprint de 14 dias  
  
 # Distribuir tarefas entre os sprints  
 tarefas\_por\_sprint = [  
 todas\_tarefas[0:7], # Sprint 1: 7 tarefas  
 todas\_tarefas[7:14], # Sprint 2: 7 tarefas  
 todas\_tarefas[14:20] # Sprint 3: 6 tarefas  
 ]  
  
 # Remover tarefas do backlog ao serem adicionadas às sprints  
 for sprint, tarefas in zip(projeto.sprints, tarefas\_por\_sprint):  
 for tarefa in tarefas:  
 sprint.adicionar\_tarefa(tarefa)  
 projeto.backlog.remover\_tarefa(tarefa)  
  
 # Distribuir tarefas e atualizar status com maior variação  
 distribuir\_tarefas\_e\_atualizar\_status(projeto, tarefas\_por\_sprint)  
  
 # Gerar relatório de progresso  
 projeto.gerar\_relatorio\_progresso()  
  
  
# Executar a simulação do projeto Scrum  
simular\_projeto\_scrum()

**VANTAGENS DA UML NO DESENVOLVIMENTO**

A principal vantagem da UML foi a sua capacidade de criar uma documentação visual que facilitou a comunicação entre os membros da equipe. Mesmo desenvolvedores com diferentes níveis de experiência puderam compreender a arquitetura do sistema e contribuir de maneira mais eficaz. Além disso, o uso da UML garantiu que o sistema fosse desenvolvido com uma visão coerente desde o início, evitando que decisões críticas fossem adiadas para fases mais avançadas do projeto, o que poderia gerar retrabalho e inconsistências.

A UML desempenhou um papel essencial na definição da arquitetura do sistema. Através dos diagramas de classes, foi possível organizar a estrutura dos objetos envolvidos no SGPA, como Usuários, Projetos, Tarefas e Sprints. Esses diagramas ajudaram a visualizar as interações entre os diferentes componentes e a garantir que as relações e dependências fossem claramente entendidas antes da implementação.

Além disso, os diagramas de casos de uso foram cruciais para mapear as funcionalidades principais do sistema, como a criação de projetos, gerenciamento de tarefas e acompanhamento de sprints. Cada ator e sua interação com o sistema foram definidos de forma clara, permitindo que a equipe de desenvolvimento identificasse as necessidades de cada usuário desde o início.

Os diagramas de sequência complementaram esse processo ao descrever o fluxo de ações no sistema, ilustrando como os componentes interagem para realizar tarefas específicas, como a criação de uma tarefa ou o fechamento de uma sprint.

**DESAFIOS NA IMPLEMENTAÇÃO DO SCRUM**

Implementar o Scrum no desenvolvimento do SGPA trouxe vários desafios, principalmente relacionados à adaptação da equipe e ao planejamento das sprints. O Scrum é uma metodologia ágil que requer uma estrutura bem definida para o planejamento e acompanhamento do progresso, e a transição para esse modelo de trabalho nem sempre foi fluida.

No código, as tarefas são distribuídas aleatoriamente entre os desenvolvedores, sem uma clara priorização baseada em critérios como dificuldade, importância ou dependências. No Scrum, a priorização de tarefas é crucial para garantir que o time trabalhe nas funcionalidades mais importantes primeiro, conforme definido pelo Product Owner.

O código permite que horas de trabalho sejam registradas para cada tarefa, mas isso é feito de forma aleatória e não considera possíveis revisões de estimativas. No Scrum, a estimativa inicial de horas é uma parte vital do planejamento, e deve ser revisada ao longo do sprint conforme novas informações surgem.

O código gera relatórios de progresso que fornecem uma visão geral do andamento das tarefas e sprints. No entanto, esses relatórios são baseados em dados atuais e não preveem possíveis atrasos ou problemas futuros. No Scrum, os relatórios de progresso devem ser usados de forma preditiva, ajudando o Product Owner e o Scrum Master a tomar decisões informadas sobre como ajustar o backlog ou replanejar o trabalho para evitar atrasos.

**EXPANSÃO E MELHORIA**

Implementar um sistema que leve em consideração a experiência e a capacidade dos desenvolvedores ao atribuir tarefas. O Scrum Master poderia usar uma abordagem mais informada para distribuir o trabalho, garantindo que as tarefas sejam atribuídas às pessoas mais qualificadas para realizá-las.

Criar mecanismos para revisar as estimativas com base no progresso real e ajustar os objetivos de cada sprint. Isso pode envolver técnicas de **revisão de estimativas** e mecanismos de feedback para que os desenvolvedores e o Scrum Master possam recalcular o tempo necessário para completar as tarefas.

Monitorar o progresso de forma mais detalhada, integrando ferramentas de análise de desempenho durante o sprint. O Scrum Master teria dificuldade em monitorar o real progresso com base apenas em atribuições aleatórias e sem o uso de métricas que identifiquem se a equipe está atrasada ou adiantada.

Incorporar ferramentas de análise preditiva que utilizem os dados de progresso para identificar potenciais riscos. Por exemplo, se uma tarefa está atrasada, o sistema deve ser capaz de alertar a equipe para que medidas corretivas possam ser tomadas antes que o sprint falhe em atingir suas metas.

**CONCLUSÃO**

O código apresenta uma simulação básica de um projeto Scrum com sprints, backlog e atribuição de tarefas, mas esconde muitos dos desafios complexos enfrentados ao implementar o Scrum em um ambiente real de desenvolvimento. Para resolver esses desafios, seria necessário introduzir mecanismos mais sofisticados de acompanhamento de progresso, controle de tarefas e distribuição de trabalho, além de melhorar a visibilidade e análise de dados para apoiar a tomada de decisões dentro do Scrum.

Além disso, todo o trabalho está disponível para visualização através do GitHub, por meio do link: **COLOQUE O LINK AQUI**