Projeto T2Ti ERP 3.0

RUP – XP – SCRUM

A T2Ti

Apresentação

A T2Ti nasce do sonho de três colegas que trabalhavam no maior banco da américa latina.

Tudo começa em 2007 com o lançamento do curso Java Starter. Logo depois veio o Siscom Java Desktop seguido de outros treinamentos.

Desde então a Equipe T2Ti se esforça para produzir material de qualidade que possa formar profissionais para o mercado, ensinando como desenvolver sistemas de pequeno, médio e grande porte.

Um dos maiores sucessos da Equipe T2Ti foi o Projeto T2Ti ERP que reuniu milhares de profissionais num treinamento dinâmico onde o participante aprendia na prática como desenvolver um ERP desde o levantamento de requisitos. Foi através desse treinamento que centenas de desenvolvedores iniciaram seu negócio próprio e/ou entraram no mercado de trabalho.

Em 2010 a T2Ti lança sua primeira aplicação para produção, o Controle Financeiro Pessoal. O sucesso foi tanto que saiu até em matéria no site Exame, ficando entre os 10 aplicativos mais baixados da semana.

Começa então a era de desenvolvimento de sistemas para alguns clientes exclusivos, pois o foco ainda era em desenvolvimento de treinamentos. A T2Ti desenvolve sistemas para o mercado nacional e internacional.

Atualmente a T2Ti se concentra nas duas vertentes: desenvolver sistemas e produzir treinamentos.

_____

Sumário

RUP | XP | SCRUM

RUP

Introdução; Princípios; Arquitetura Global; Elementos Estruturais; Disciplinas; Fases; Online.

SCRUM

Introdução; História; Definição; Teoria; Equipe; Eventos; Artefatos.

XP

Introdução; Valores Fundamentais; Princípios Básicos; Prática; Ciclo de Vida.

SCRUM + XP

É possível?

RUP

Introdução

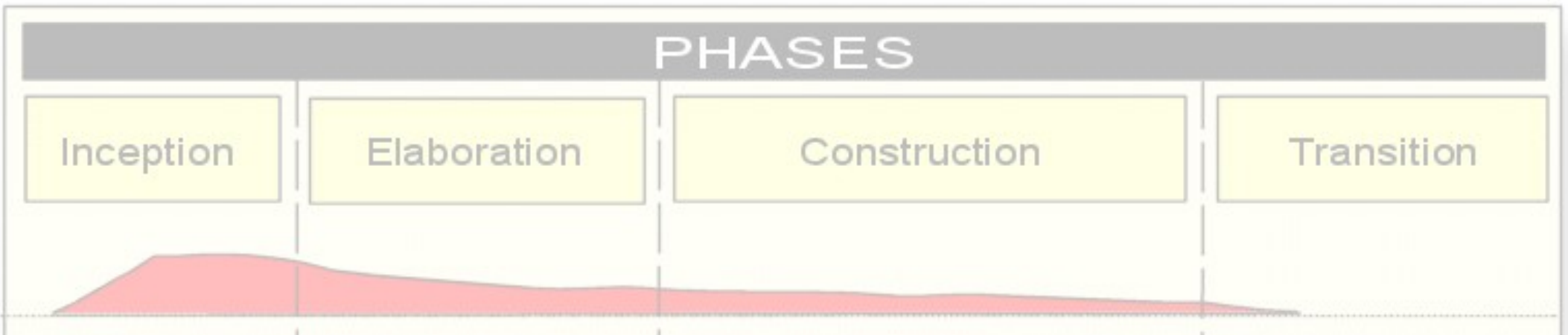
RUP é abreviação de Rational Unified Process. Tradução para o português: Processo Unificado da Rational. Trata-se de um processo proprietário de Engenharia de Software criado pela Rational Software Corporation (criadora também da UML).

A Rational Software Corporation foi adquirida posteriormente pela IBM. O RUP passou a ser chamado de IRUP que agora é uma abreviação de IBM Rational Unified Process.

O RUP fornece técnicas a serem seguidas pelos membros da equipe de desenvolvimento de software com o objetivo de aumentar a sua produtividade durante o desenvolvimento.

O RUP usa a abordagem da orientação a objetos em sua concepção e é projetado e documentado utilizando a notação UML (Unified Modeling Language) para especificar, modelar e documentar artefatos. Utiliza técnicas e práticas aprovadas comercialmente.

É um processo considerado pesado e preferencialmente aplicável a grandes equipes de desenvolvimento e a grandes projetos, porém o fato de ser amplamente customizável torna possível que seja adaptado para projetos de qualquer escala. Para os gerentes, o RUP fornece uma solução disciplinada para assinalar tarefas e responsabilidades dentro da organização de desenvolvimento de software.



RUP

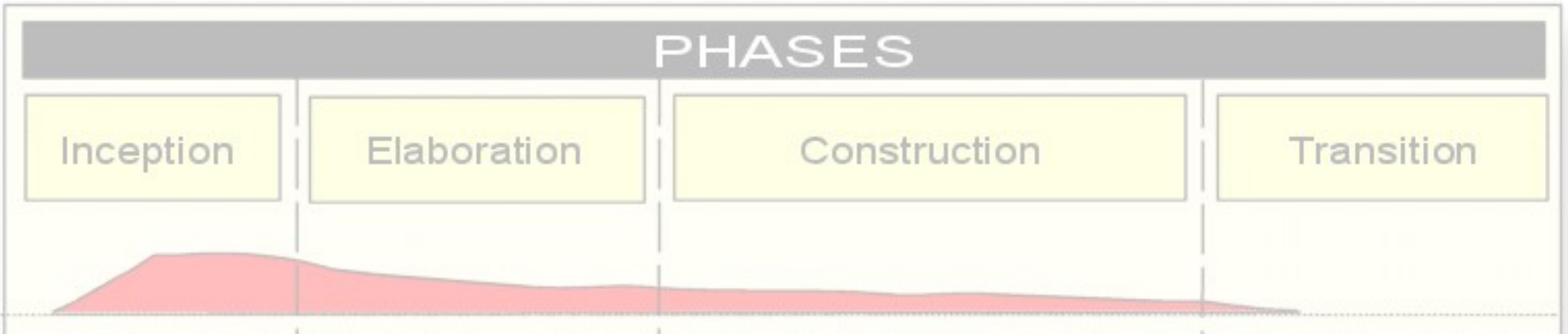
Introdução

O principal objetivo do RUP é atender as necessidades dos usuários garantindo uma produção de software de alta qualidade que cumpra um cronograma e um orçamento previsíveis. Assim, o RUP mostra como o sistema será construído na fase de implementação, gerando o modelo do projeto e, opcionalmente, o modelo de análise que é utilizado para garantir a robustez.

O RUP define perfeitamente quem é responsável pelo que, como as coisas deverão ser feitas e quando devem ser realizadas, descrevendo todas as metas de desenvolvimento especificamente para que sejam alcançadas.

Kroll e Kruchten (2003) fornecem três definições para o RUP:

1) É uma maneira de desenvolvimento de software que é iterativa, centrada na arquitetura e guiada por casos de uso.
2) É um processo de engenharia de software bem definido e bem estruturado. Ele define claramente quem é responsável pelo que, como as coisas devem ser feitas e quando fazê-las. Provê ainda uma estrutura bem definida para o ciclo de vida de um projeto, articulando os marcos e pontos de decisão.
3) É um produto de processo que oferece uma estrutura de processo customizável para a engenharia de software.



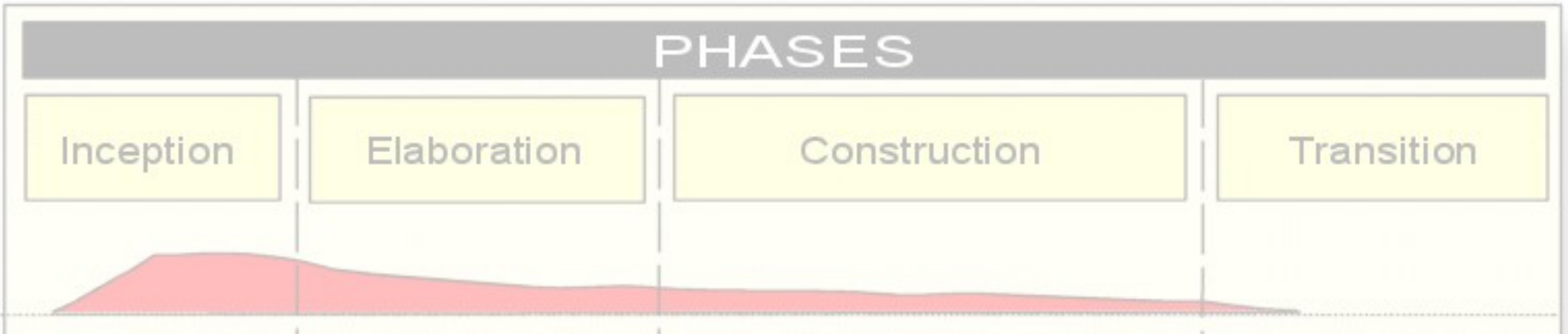
Princípios

Não existe uma fórmula para aplicação do RUP. Eele poder ser aplicado de várias formas diferentes para cada projeto e organização a ser apresentados. De acordo com Martins (2007), existem alguns princípios que podem caracterizar e diferenciar o RUP de outros métodos iterativos:

a) Atacar os riscos antecipadamente e continuamente.
b) Certificar-se de entregar algo de valor ao cliente.
c) Focar no software executável.
d) Acomodar mudanças antecipadas.
e) Liberar um executável da arquitetura antecipadamente.
f) Construir o sistema com componentes.
g) Trabalhar junto como uma equipe.
h) Fazer da qualidade um estilo de vida, não algo para depois.

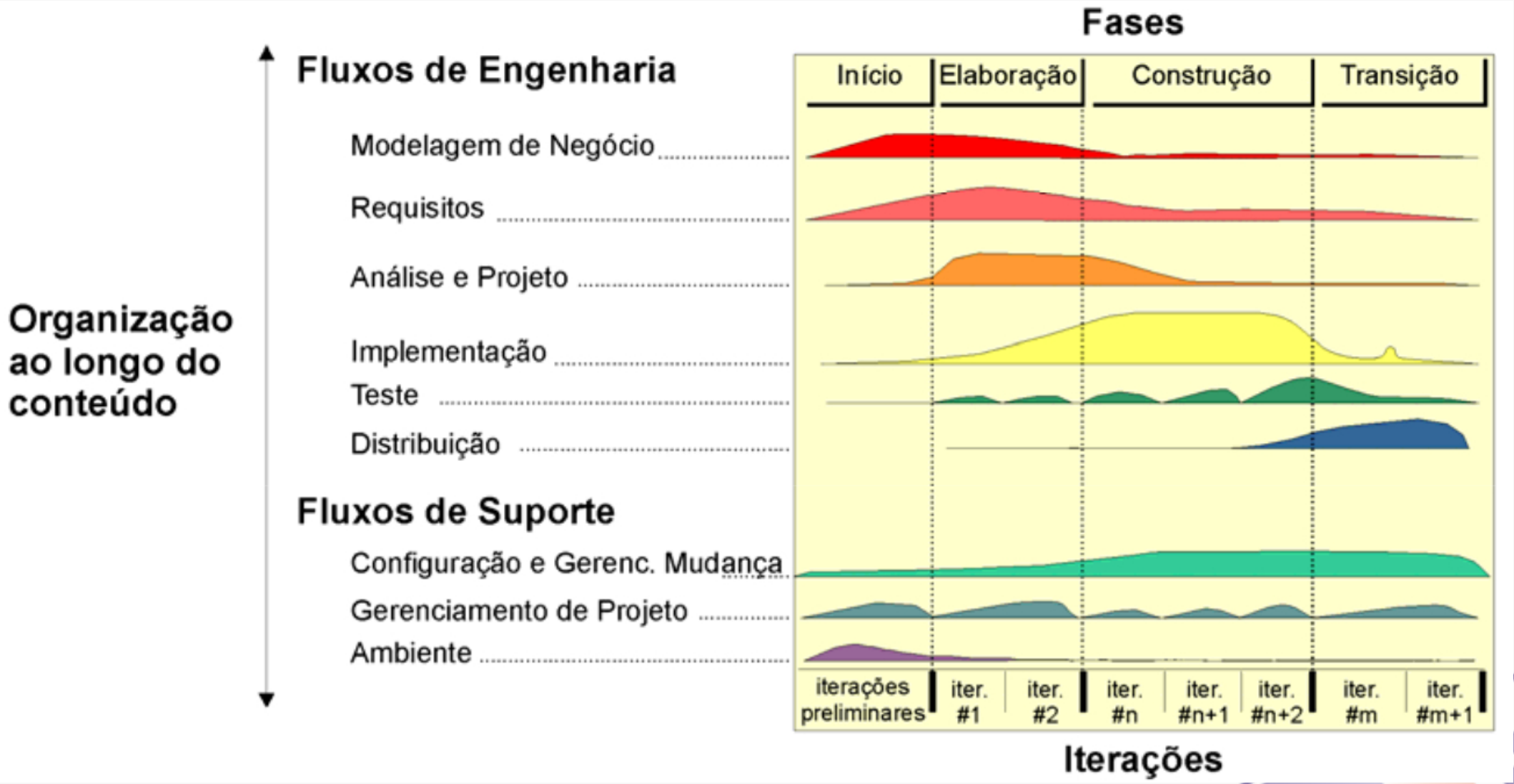
Os princípios do RUP acompanha as premissas referentes a garantia da qualidade do processo visando alcançar a garantia da qualidade do produto de software a ser desenvolvido.

A arquitetura global do RUP é organizada em duas dimensões. Observe na imagem da página seguinte.



RUP

Arquitetura Global



RUP

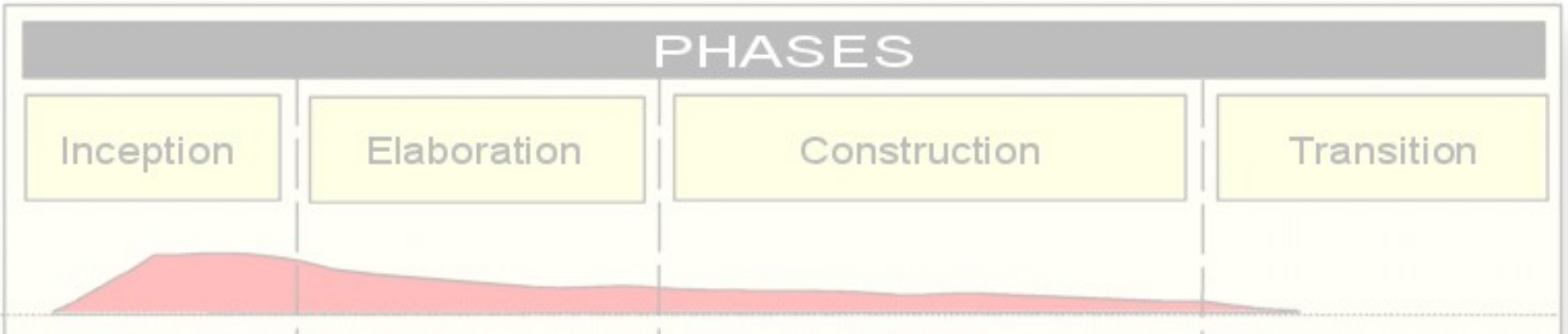
Arquitetura Global

O eixo horizontal evidencia o aspecto dinâmico do processo, descrevendo como ocorre o desenvolvimento ao longo do tempo em termos de fases, iterações e marcos. Também mostra como a ênfase varia ao longo do tempo. Por exemplo, nas iterações iniciais, gasta-se mais tempo com modelagem de negócio, requisitos, análise e projeto, enquanto nas iterações finais gasta-se mais tempo com implementação, teste e distribuição. Embora os nomes dos fluxos de engenharia possam evocar as fases sequenciais do modelo em cascata, estes fluxos são revisitados ao longo do ciclo de vida, variando de intensidade a cada iteração.

O eixo vertical representa o aspecto estático do processo, organizado em termos de disciplinas.

No RUP, processo é definido como sendo uma descrição de quem está fazendo o quê, como e quando – estes quatro elementos estruturais, correspondem a:

- Papel (quem).
- Atividade (como).
- Artefato (o quê).
- Fluxo (quando).

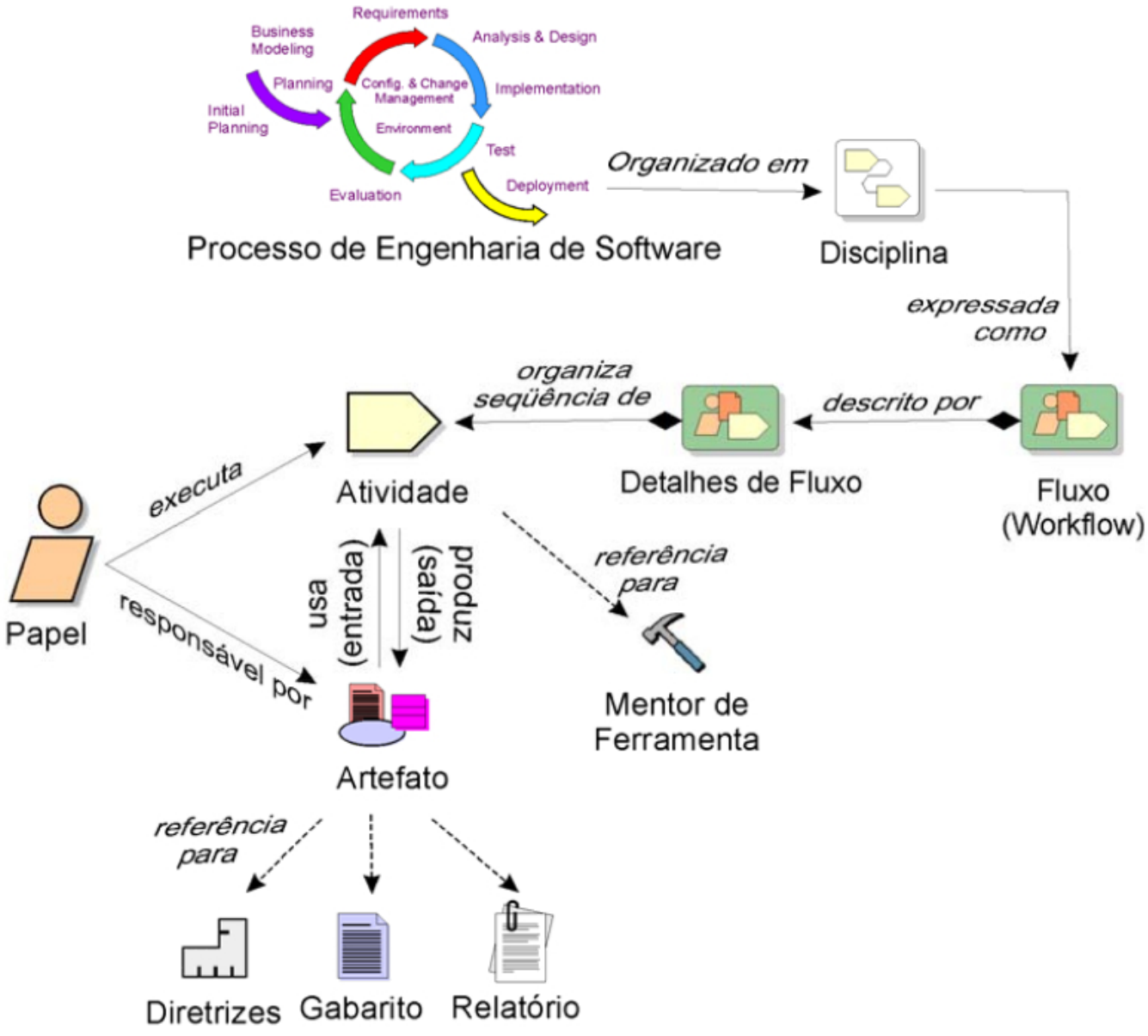


Elementos Estruturais

Na imagem ao lado podemos observar os conceitos-chave, os elementos estruturais estáticos que são definidos no RUP.

Fluxo

É a sequência de atividades que produz um resultado de valor observável. No RUP, o fluxo é expresso como um *diagrama de atividade* da UML. Há muitas maneiras de se organizar o conjunto de atividades em fluxos num processo de engenharia de software. No RUP, os fluxos são organizados em dois níveis: Fluxo central (Disciplina) e Detalhes de Fluxo.



Elementos Estruturais

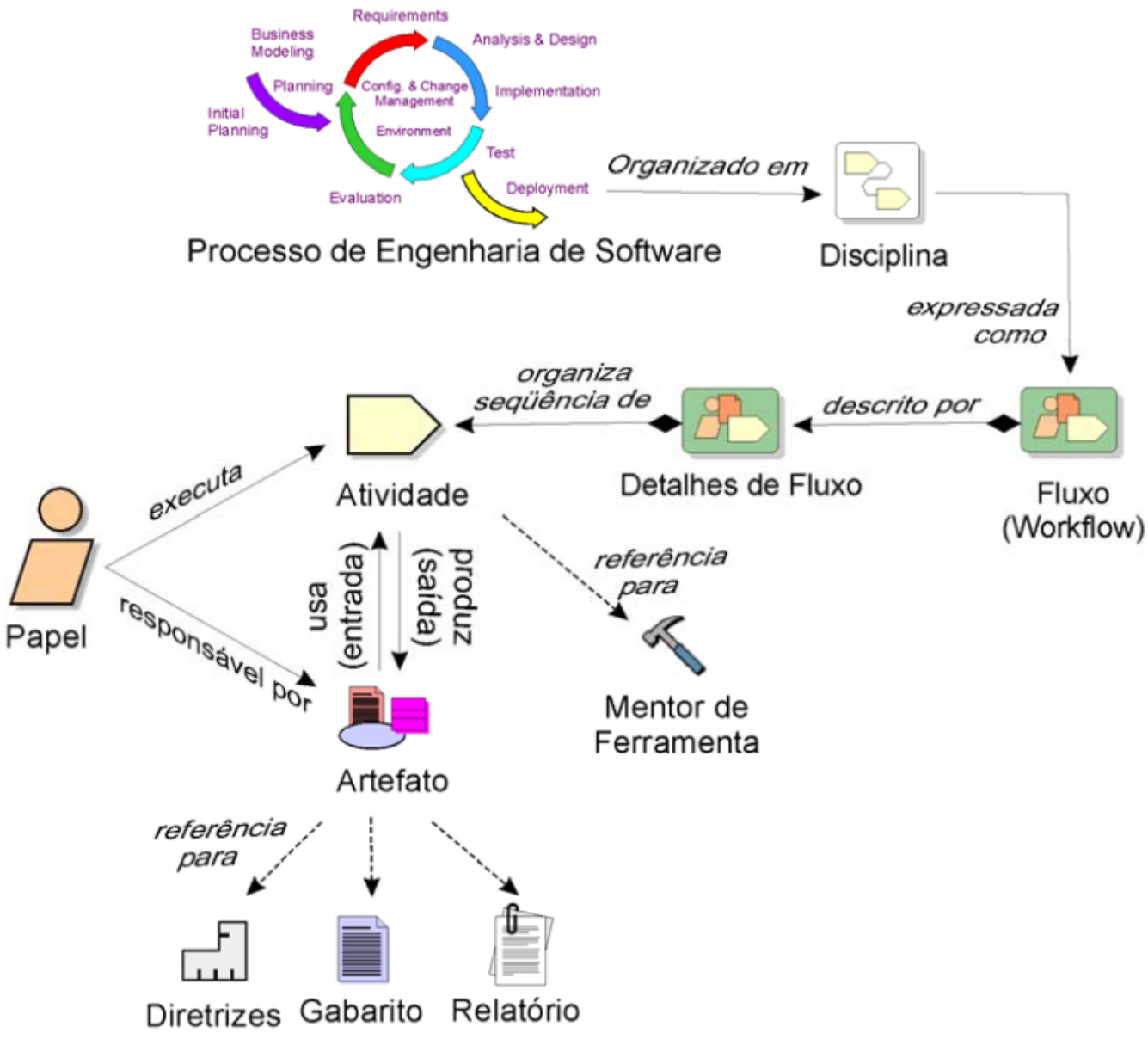
Atividade

É o trabalho executado para produzir um resultado significativo no contexto do projeto.

Consiste, geralmente, na criação ou atualização de artefatos.

Toda atividade é atribuída a um papel específico.

O *Mentor de Ferramenta* fornece diretrizes de como usar uma ferramenta de software específica na execução da atividade.



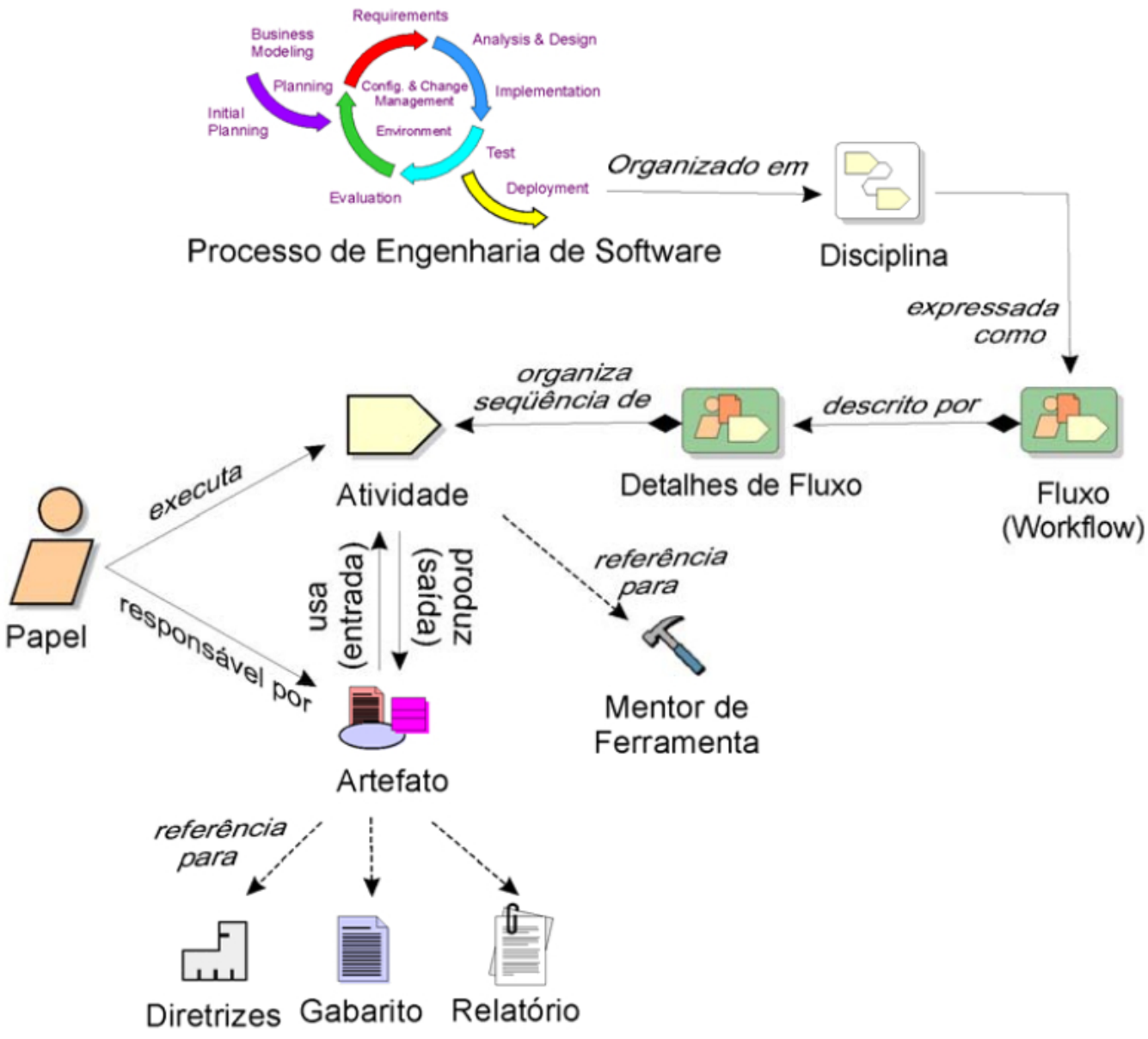
Elementos Estruturais

Papel

Define o comportamento e as responsabilidades de um indivíduo ou grupo de indivíduos trabalhando em equipe.

O comportamento é expresso em termos de atividades a serem executadas.

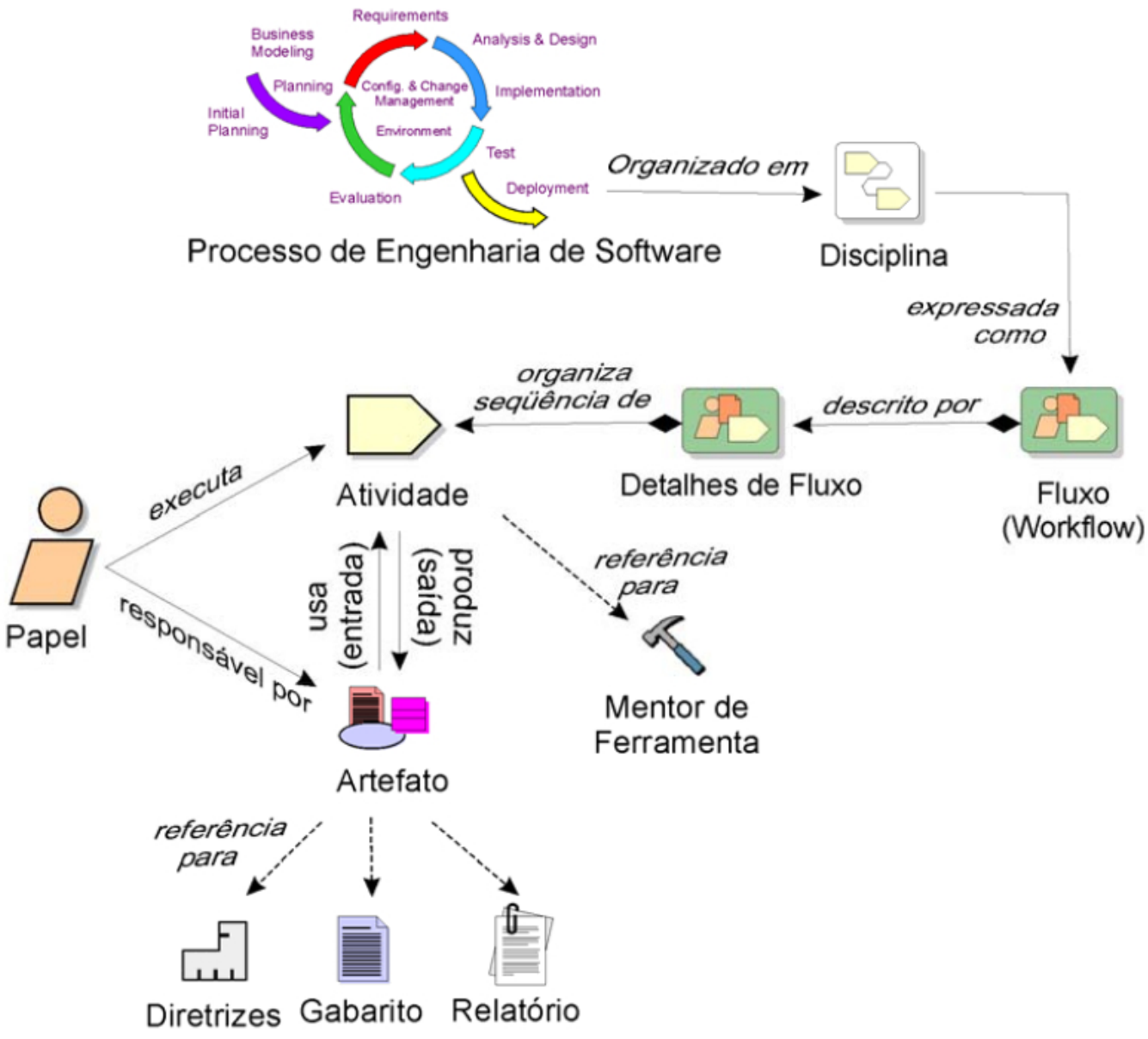
Responsabilidades são expressas em termos de artefatos que o papel cria, modifica ou controla.



Elementos Estruturais

Artefato

É um produto do projeto. Pode ser um documento, um modelo, um código-fonte, um executável etc. É de responsabilidade de um único papel, embora possa ser usado por vários papéis. São usados, produzidos ou modificados em atividades. Gabarito ou é um 'modelo' do artefato a ser usado em sua criação. Um Relatório consiste em informações que são extraídas de um ou vários artefatos. Diretrizes são informações sobre como desenvolver, avaliar e usar os artefatos. Uma atividade representa o trabalho a ser feito enquanto as diretrizes expressam como fazer o trabalho.



Disciplinas

Analisando novamente a figura, observe que temos o que foi chamado de Fluxos de Engenharia e Fluxos de Suporte.

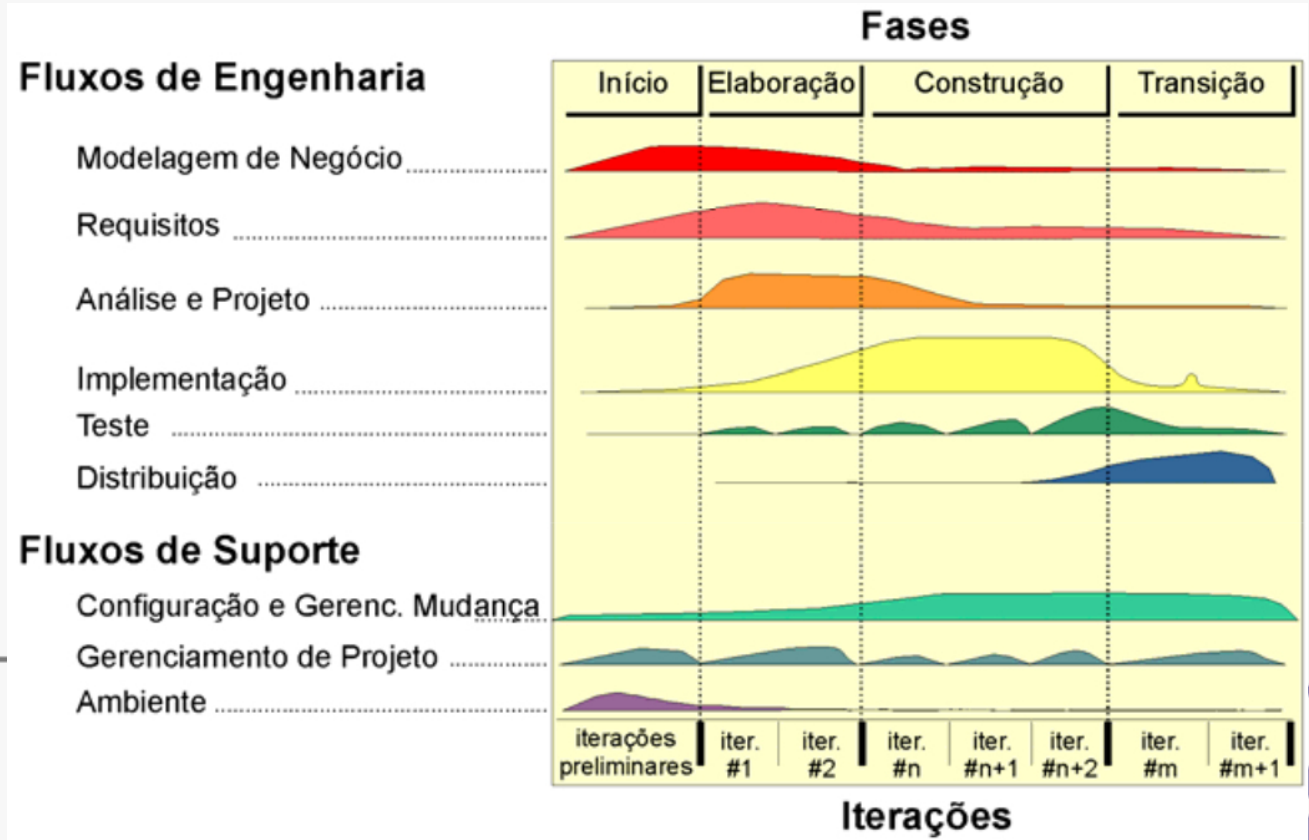
Dentro desses fluxos nós temos as chamadas disciplinas, que são nove no total. Seis delas pertencem ao fluxo de engenharia e três ao fluxo de suporte (também chamado de apoio e suporte).

Segundo Martins (2007) uma disciplina é uma coleção de atividades relacionadas que fazem parte de um contexto comum em um projeto.

As disciplinas proporcionam um melhor entendimento do projeto sob o ponto de vista tradicional de um processo cascata.

A separação das atividades em disciplinas torna a compreensão das atividades mais fácil, porém dificulta mais o planejamento das atividades.

Veremos agora uma descrição sobre cada uma das disciplinas.

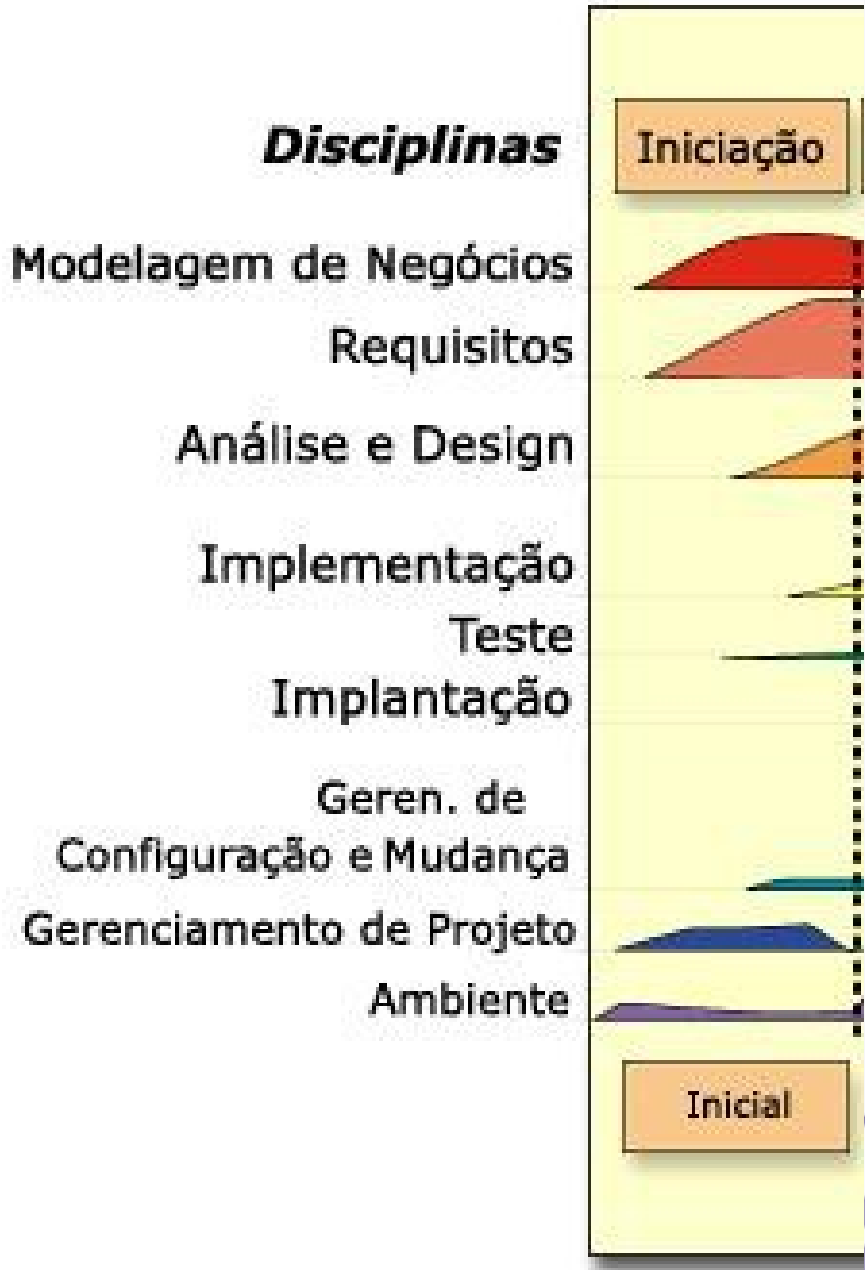


Disciplinas | Modelagem de Negócios

As organizações estão cada vez mais dependentes de sistemas de TI, tornando-se obrigatório que os engenheiros de sistemas de informação saibam como as aplicações em desenvolvimento se inserem na organização. As empresas investem em TI quando entendem a vantagem competitiva do valor acrescentado pela tecnologia.

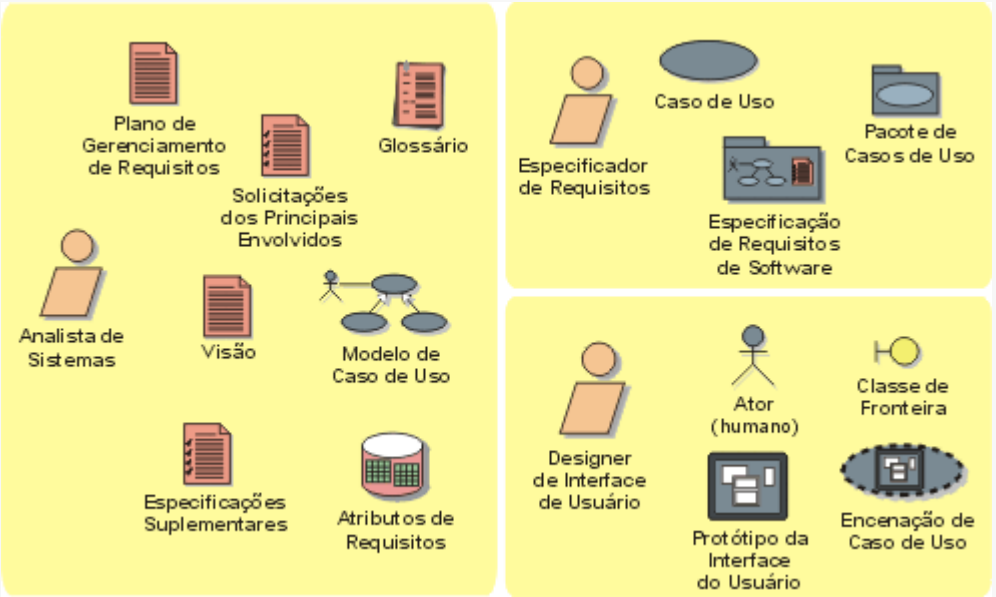
O objetivo da modelagem de negócios é estabelecer uma melhor compreensão e canal de comunicação entre a engenharia de negócios e a engenharia de software. Compreender o negócio significa que os engenheiros de software devem compreender a estrutura e a dinâmica da empresa alvo (o cliente), os atuais problemas da organização e as possíveis melhorias.

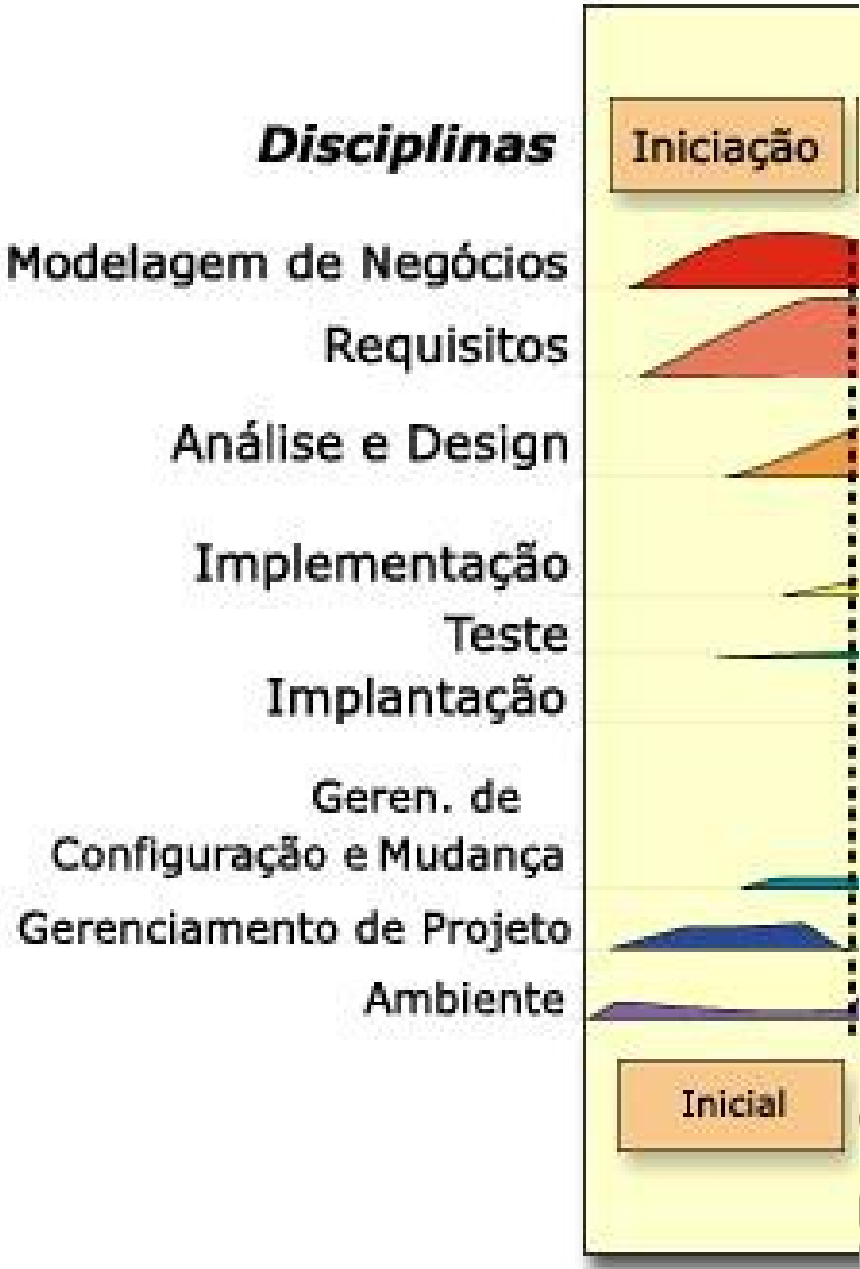
Eles devem ainda garantir um entendimento comum da organização-alvo entre os clientes, usuários finais e desenvolvedores. A modelagem de negócios explica como descrever uma visão da organização na qual o sistema será implantado e como usar esta visão como uma base para descrever o processo, os papéis e as responsabilidades.



Disciplinas | Requisitos

Esta disciplina explica como transformar as necessidades das partes interessadas em requisitos que serão usados para criar o sistema.



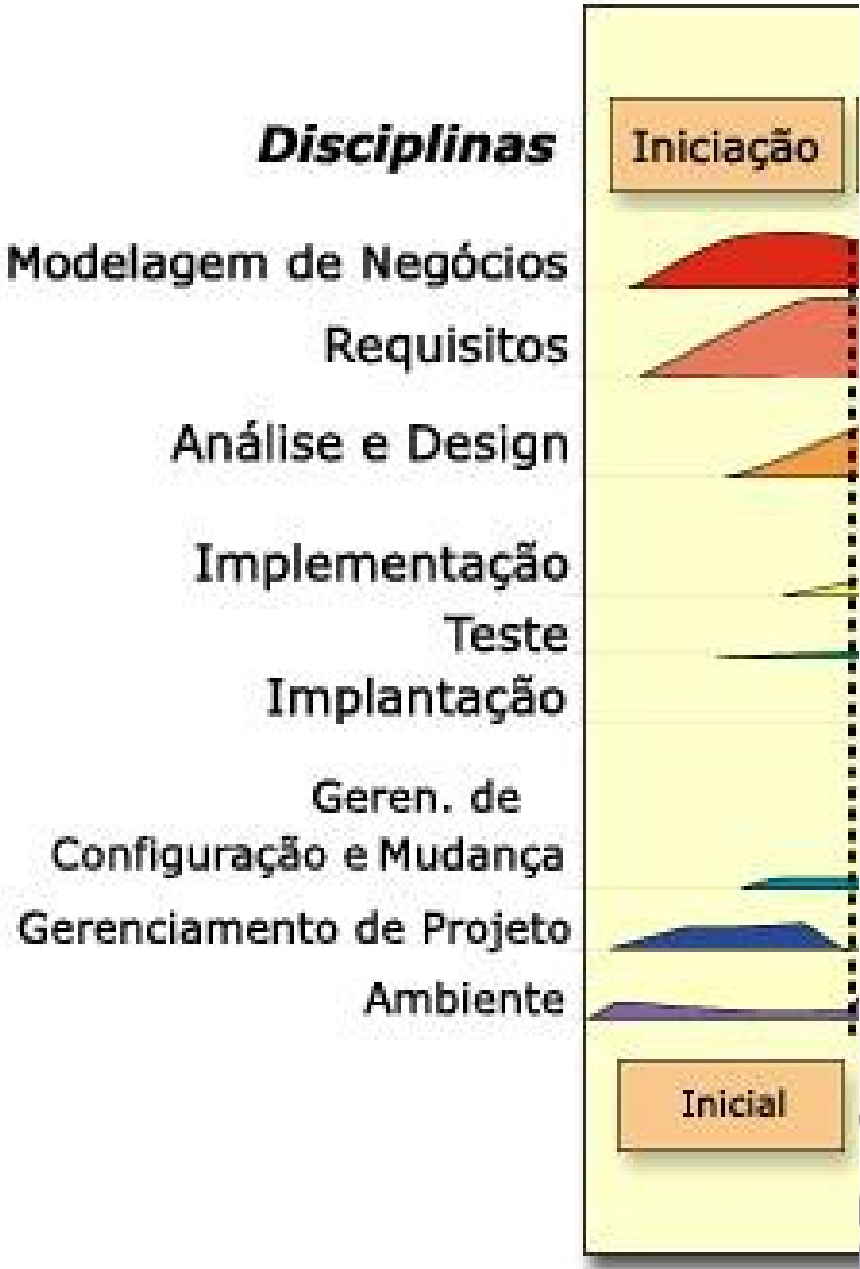


Disciplinas | Análise e Design

O Design é traduzido como Projeto. Ou seja, disciplina de Análise e Projeto. Mostrará como o sistema será realizado. O objetivo é construir um sistema que:

- Execute as tarefas e funções especificadas nas descrições de casos de uso (em um ambiente de execução específico).
- Cumpra todas as suas necessidades.
- Seja fácil de manter quando ocorrerem mudanças de requisitos funcionais.

Resultados de projeto em um modelo de Análise e Projeto tem, opcionalmente, um Modelo de Projeto. O Modelo de Projeto serve como uma abstração do código-fonte, ou seja, atua como um modelo de "gabarito" de como o código-fonte é estruturado e escrito. O Modelo de Projeto consiste em classes de projeto estruturadas em pacotes e subsistemas com interfaces bem definidas, representando o que irá se tornar componentes da aplicação. Ele também contém descrições de como os objetos dessas classes colaboram para desempenhar casos de uso do projeto.



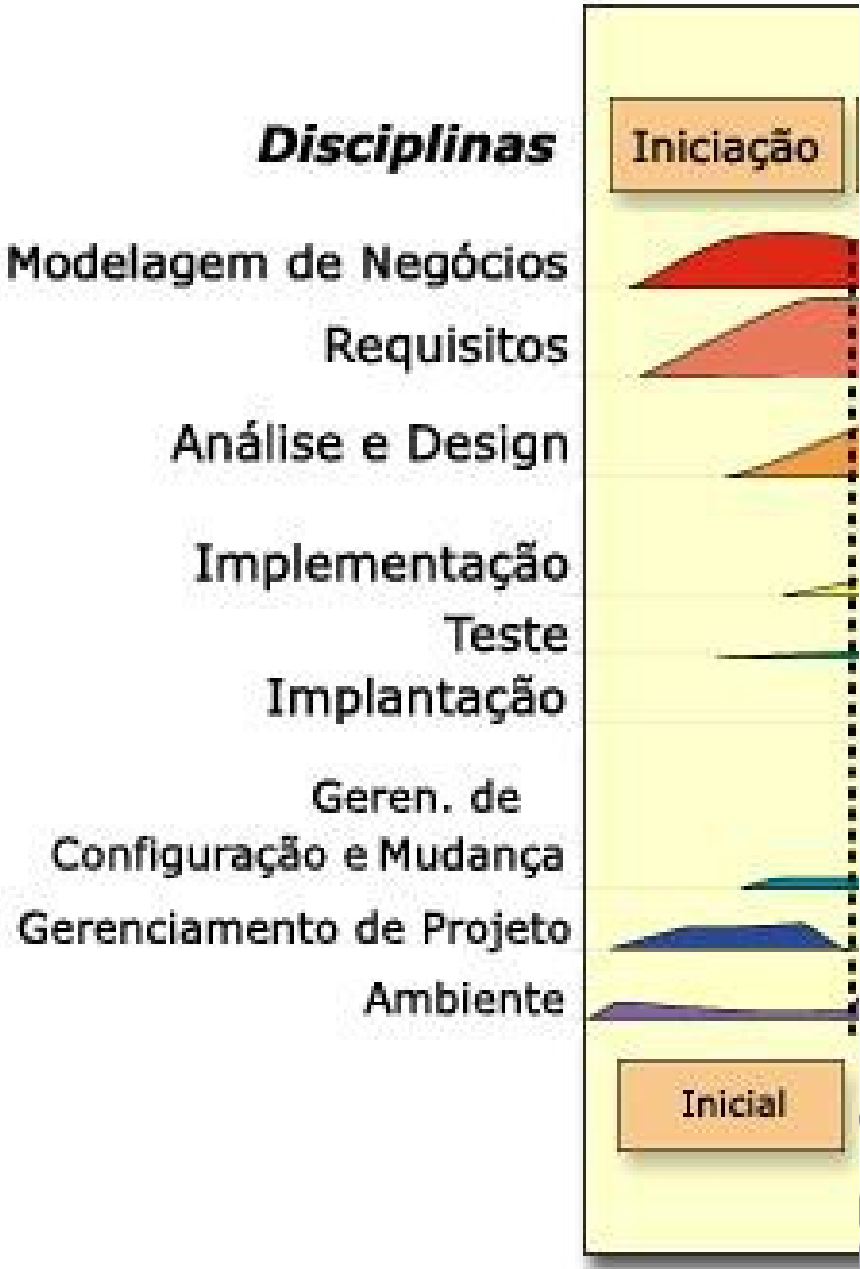
Disciplinas | Implementação

Sistemas são realizados através da aplicação de componentes.

O processo descreve como reutilizar componentes existentes ou implementar novos componentes com responsabilidades bem definidas, tornando o sistema mais fácil de manter e aumentar as possibilidades de reutilização.

Essa disciplina serve então para:

- Definir a organização do código em termos de subsistemas de implementação organizadas em camadas.
- Implementar classes e objetos em termos de componentes (arquivos-fonte, binários, executáveis e outros).
- Testar os componentes desenvolvidos como unidades.
- Integrar os resultados produzidos por implementadores individuais (ou equipes), em um sistema executável.

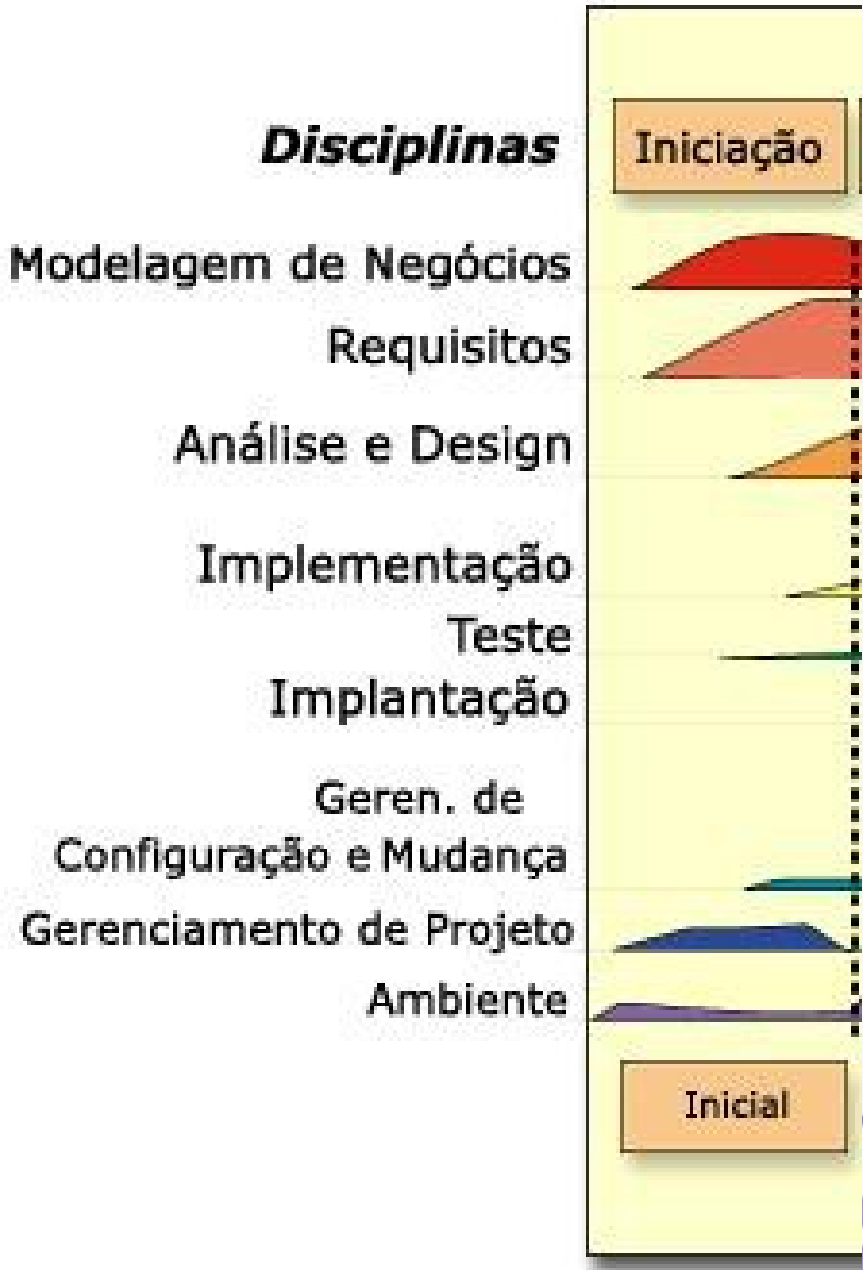


Disciplinas | Teste

O RUP propõe uma abordagem iterativa, o que significa que deve-se testar todo o projeto. Isto permite encontrar bugs tão cedo quanto possível, o que reduz radicalmente o custo de reparar o bug. Os testes são realizados ao longo de quatro dimensões da qualidade: confiabilidade, funcionalidade, desempenho da aplicação e desempenho do sistema. Para cada uma destas dimensões da qualidade, o processo descreve como passar pelo teste do ciclo de planejamento, projeto, implementação, execução e avaliação.

Finalidades desta disciplina:

- Verificar a interação entre objetos.
- Verificar a integração adequada de todos os componentes do software.
- Verificar se todos os requisitos foram corretamente implementados.
- Identificar e garantir que os bugs são abordados antes da implantação do software.
- Garantir que todos os bugs são corrigidos, reanalisados e fechados.



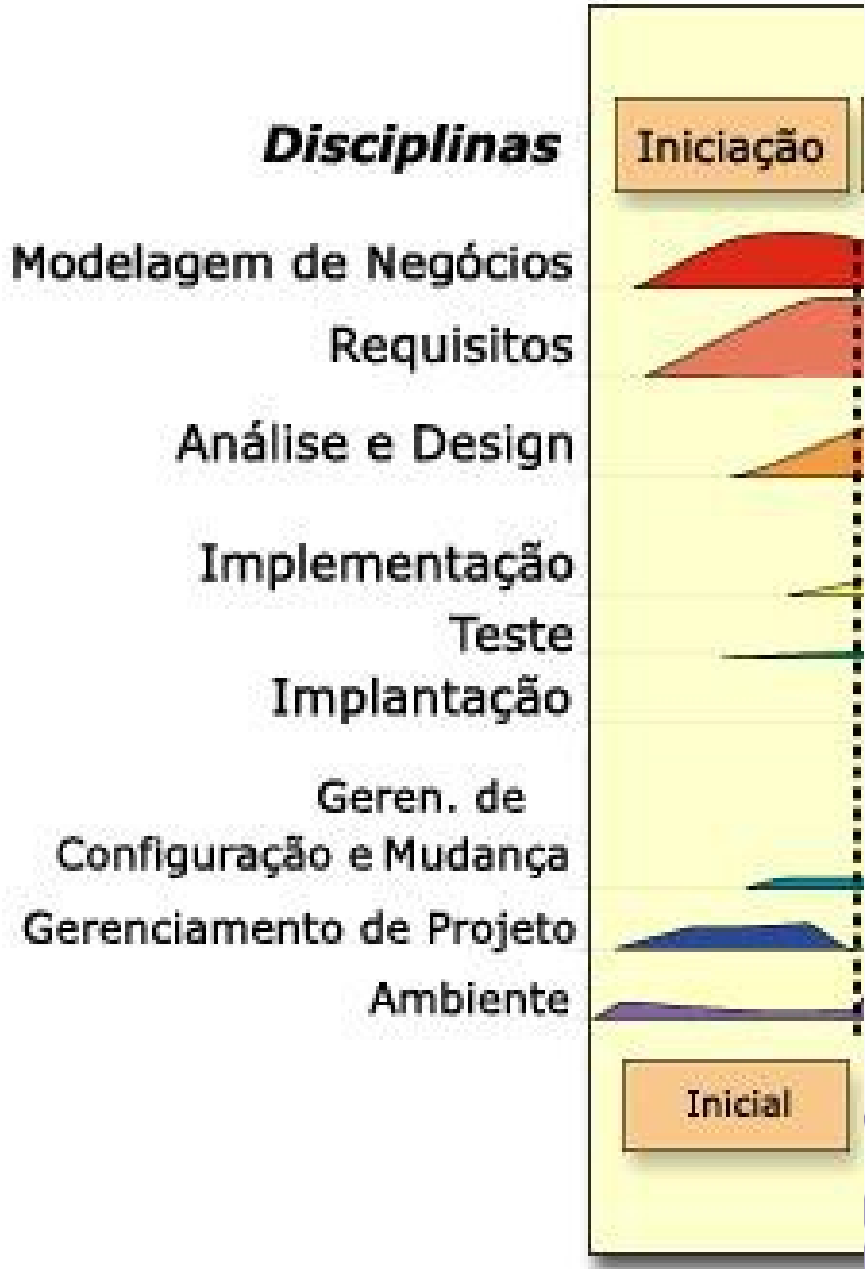
Disciplinas | Implantação

Também chamada de Distribuição, embora "distribuição" seja considerada uma das atividades da implantação. Seu objetivo é o de produzir com sucesso lançamentos de produtos e entregar o software para seus usuários finais.

Ela cobre uma vasta gama de atividades, incluindo a produção de *releases* externos do software, a embalagem do software e aplicativos de negócios, distribuição do software, instalação do software e prestação de ajuda e assistência aos usuários.

Embora as atividades de implantação estejam principalmente centradas em torno da fase de transição, muitas das atividades devem ser incluídas nas fases anteriores para se preparar para a implantação, no final da fase de construção.

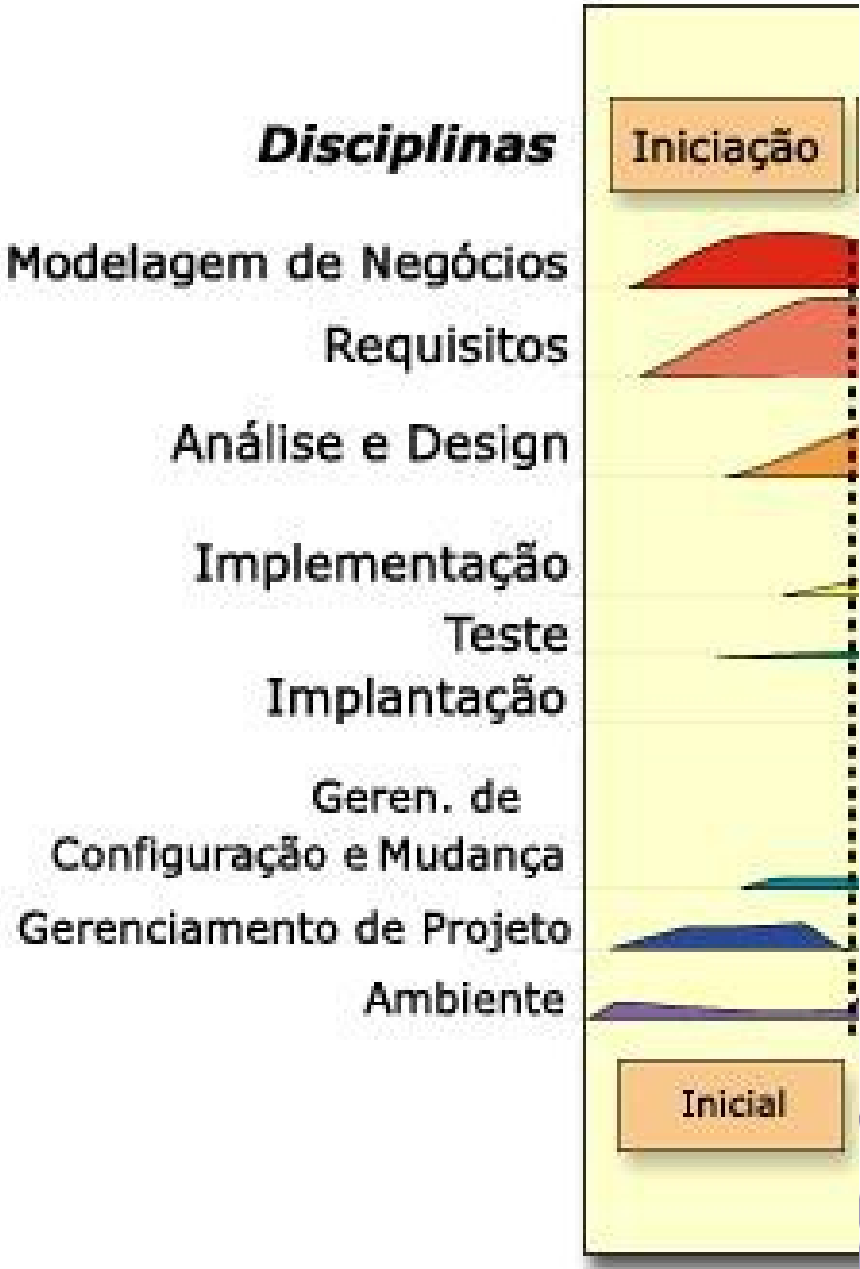
Os processos (workflows) de "Implantação e Ambiente" do RUP contêm menos detalhes do que outros workflows.



Disciplinas | Configuração e Mudança

A disciplina de Gestão de Mudança em negócios com RUP abrange três gerenciamentos específicos:

- Gerenciamento de Configuração: responsável pela estruturação sistemática dos produtos. Artefatos, como documentos e modelos, precisam estar sob controle de versão e essas alterações devem ser visíveis. Mantém, ainda, o controle de dependências entre artefatos para que todos os artigos relacionados sejam atualizados quando são feitas alterações
- Gerenciamento de Solicitações de Mudança: durante o processo de desenvolvimento de sistemas com muitos artefatos existem diversas versões. O controle das propostas de mudança é mantido aqui.
- Gerenciamento de Status e Medição: os pedidos de mudança têm os estados: novo, conectado, aprovado, cedido e completo. A solicitação de mudança também tem atributos como a causa raiz, ou a natureza, prioridade, etc. Esses estados e atributos são armazenados no banco de dados para produzir relatórios úteis sobre o andamento do projeto.



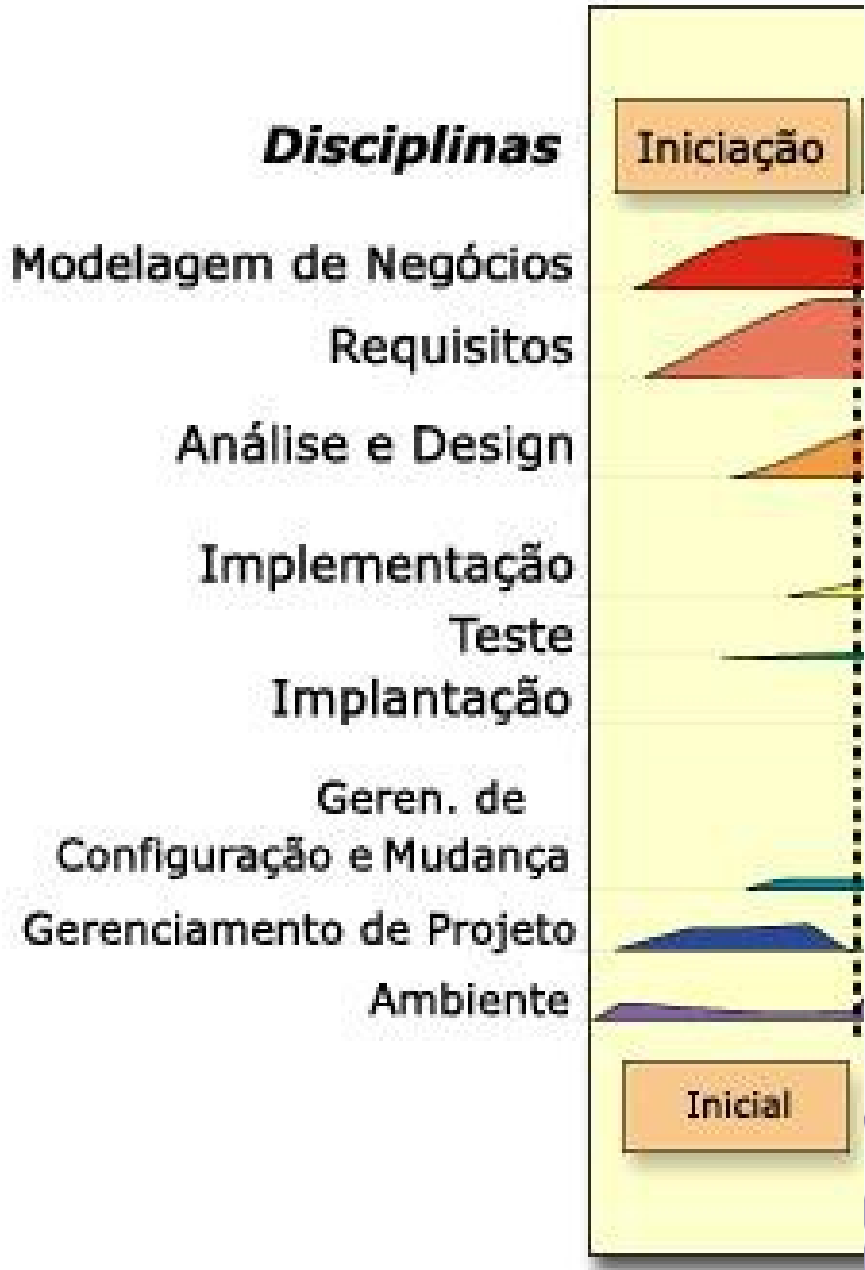
Disciplinas | Gerenciamento de Projeto

Ocorre em dois níveis. Há uma baixa granularidade ou planos de Fase que descreve todo o projeto, e uma série de alta granularidade ou planos de Iteração que descrevem os passos iterativos.

Esta disciplina concentra-se principalmente sobre os aspectos importantes de um processo de desenvolvimento iterativo:

- Gestão de riscos.
- Planejamento de um projeto iterativo através do ciclo de vida e para uma iteração particular.
- Processo de acompanhamento de um projeto iterativo.
- Métricas.

No entanto, esta disciplina do RUP não tenta cobrir todos os aspectos do gerenciamento de projetos. Coisas como gestão de pessoas, orçamento geral e gestão de contratos não são abrangidos aqui.



Disciplinas | Ambiente

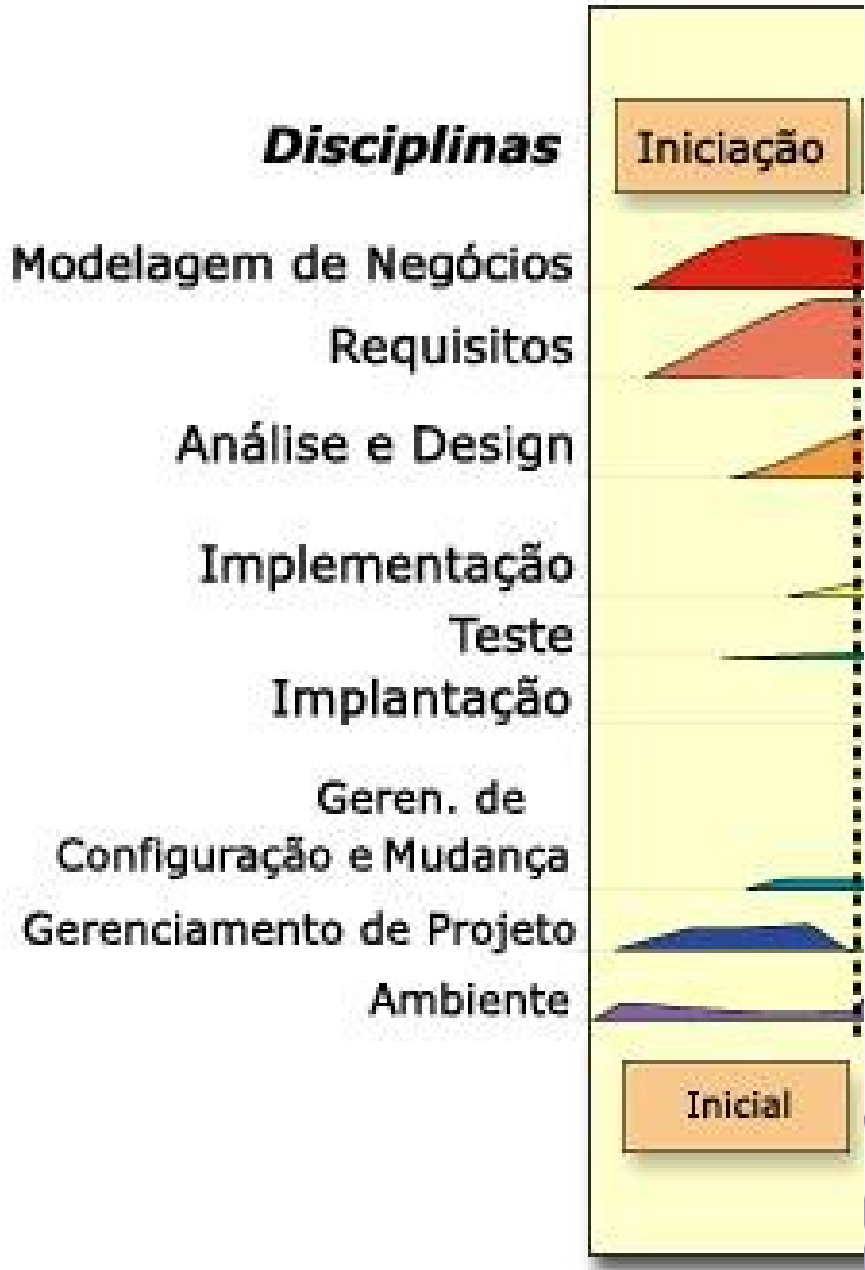
A disciplina de Ambiente oferece o ambiente de suporte para um projeto. Ao fazer isso, ela também serve de suporte a todas as outras disciplinas.

A disciplina de Ambiente concentra-se nas atividades necessárias à configuração do processo para um projeto.

Ela descreve as atividades para o desenvolvimento das diretrizes de suporte de um projeto.

A meta das atividades dessa disciplina é oferecer à organização o ambiente de desenvolvimento de software — processos e ferramentas — que dará suporte à equipe de desenvolvimento.

Já que "passeamos" pela vertical, vamos agora "viajar" pela horizontal e analisar as fases do RUP.



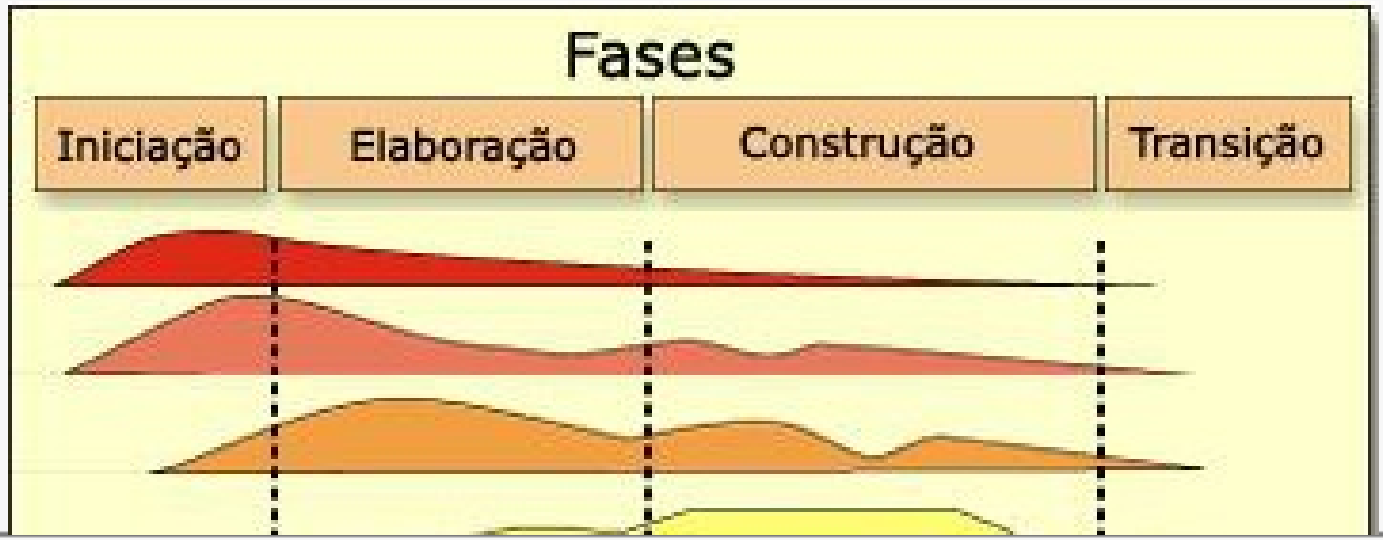
Fases

As fases indicam a ênfase que é dada no projeto em um dado instante. Para capturar a dimensão do tempo de um projeto, o RUP divide o projeto em quatro fases diferentes:

- Iniciação ou Concepção: ênfase no escopo do sistema.
- Elaboração: ênfase na arquitetura.
- Construção: ênfase no desenvolvimento.
- Transição: ênfase na implantação.

As fases são compostas de iterações. As iterações são janelas de tempo; as iterações possuem prazo definido enquanto as fases são objetivas.

Todas as fases geram artefatos. Estes serão utilizados nas próximas fases e documentam o projeto, além de permitir melhor acompanhamento.

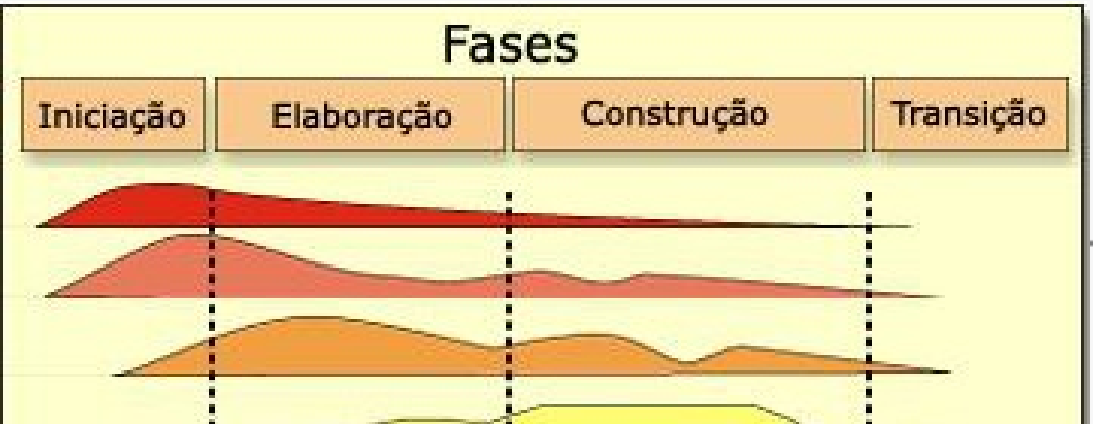


Fases | Iniciação ou Concepção

A meta dominante da fase de iniciação é atingir o consenso entre todos os envolvidos sobre os objetivos do ciclo de vida do projeto.

A fase de iniciação tem muita importância principalmente para os esforços dos desenvolvimentos novos, nos quais há muitos riscos de negócios e de requisitos que precisam ser tratados para que o projeto possa prosseguir.

Para projetos que visam melhorias em um sistema existente, a fase de iniciação é mais rápida, mas ainda se concentra em assegurar que o projeto seja compensatório e que seja possível fazê-lo.



Objetivos:

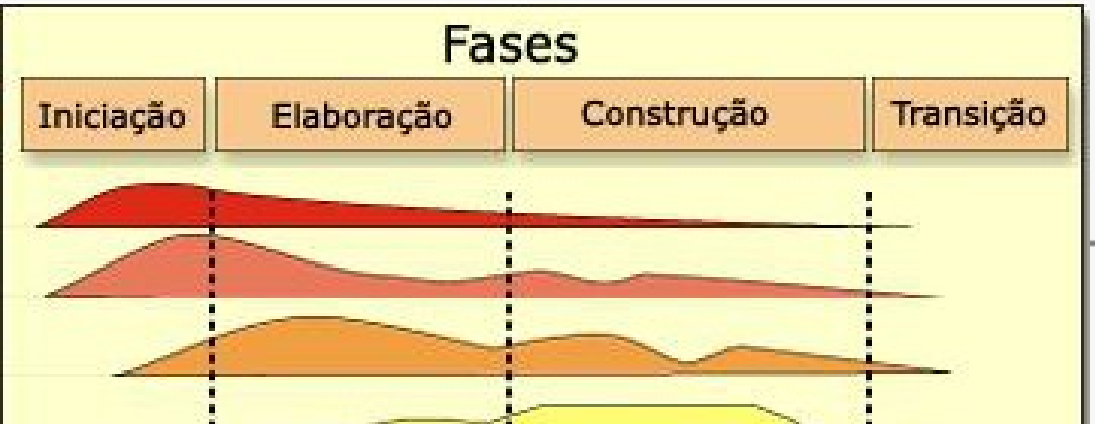
- Estabelecer o escopo do software do projeto e as condições limite, incluindo uma visão operacional, critérios de aceitação e o que deve ou não estar no produto.
- Discriminar os casos de uso críticos do sistema, os principais cenários de operação e o que direcionará as principais trocas de design.
- Exibir, e talvez demonstrar, pelo menos uma opção de arquitetura para alguns cenários básicos.
- Estimar o custo geral e a programação para o projeto inteiro (e estimativas detalhadas para a fase de elaboração imediatamente a seguir).
- Estimar riscos em potencial (as origens de imprevistos).
- Preparar o ambiente de suporte para o projeto.

Fases | Elaboração

A meta da fase de elaboração é criar a baseline (uma 'imagem' de uma versão de cada artefato no repositório do projeto) para a arquitetura do sistema a fim de fornecer uma base estável para o esforço da fase de construção.

A arquitetura se desenvolve a partir de um exame dos requisitos mais significativos (aqueles que têm grande impacto na arquitetura do sistema) e de uma avaliação de risco.

A estabilidade da arquitetura é avaliada através de um ou mais protótipos de arquitetura.



Objetivos:

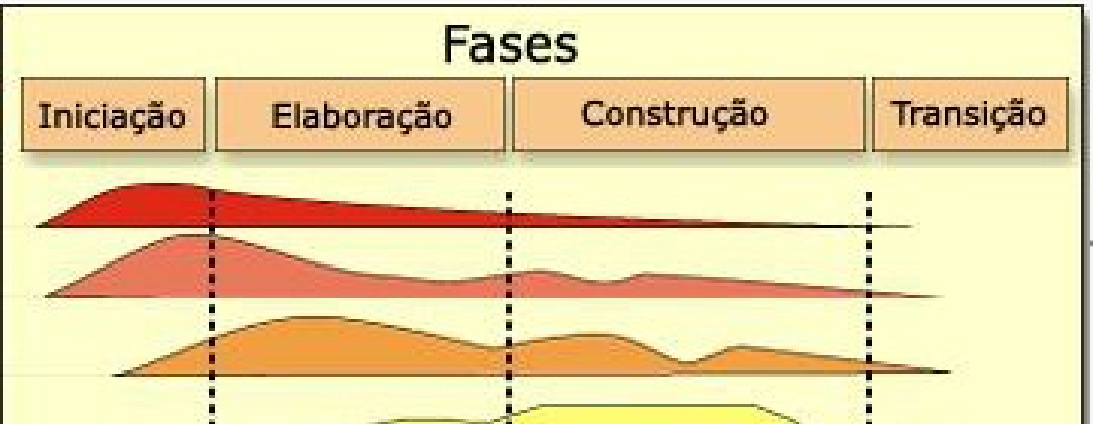
- Assegurar que a arquitetura, os requisitos e os planos sejam estáveis o suficiente e que os riscos sejam suficientemente diminuídos a fim de determinar com segurança o custo e a programação para a conclusão do desenvolvimento.
- Tratar todos os riscos significativos do ponto de vista da arquitetura do projeto.
- Estabelecer uma arquitetura da baseline derivada do tratamento dos cenários significativos do ponto de vista da arquitetura, que normalmente expõem os maiores riscos técnicos do projeto.
- Produzir um protótipo evolutivo dos componentes de qualidade de produção, assim como um ou mais protótipos descartados para diminuir riscos específicos.
- Demonstrar que a arquitetura de baseline suportará os requisitos do sistema a um custo justo e em tempo justo.
- Estabelecer um ambiente de suporte.

Fases | Construção

A meta da fase de construção é esclarecer os requisitos restantes e concluir o desenvolvimento do sistema com base na arquitetura da baseline.

A fase de construção é de certa forma um processo de manufatura, em que a ênfase está no gerenciamento de recursos e controle de operações para otimizar custos, programações e qualidade.

Nesse sentido, a mentalidade do gerenciamento passa por uma transição do desenvolvimento da propriedade intelectual durante a iniciação e elaboração, para o desenvolvimento dos produtos que podem ser implantados durante a construção e transição.



Objetivos:

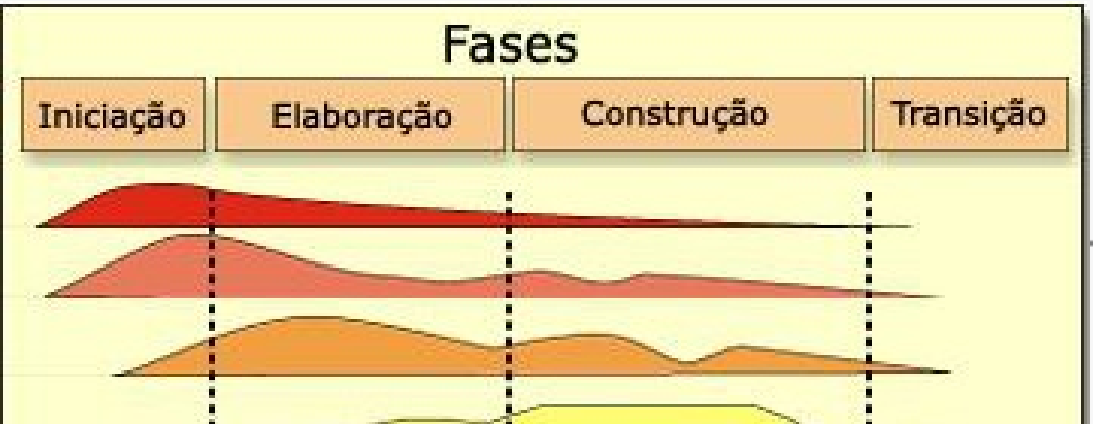
- Minimizar os custos de desenvolvimento, otimizando recursos e evitando retalhamento e retrabalho desnecessários.
- Atingir a qualidade adequada com rapidez e eficiência.
- Atingir as versões úteis (alfa, beta e outros releases de teste) com rapidez e eficiência.
- Concluir a análise, o design, o desenvolvimento e o teste de todas as funcionalidades necessárias.
- Desenvolver de modo iterativo e incremental um produto completo que esteja pronto para a transição para a sua comunidade de usuários.
- Decidir se o software, os locais e os usuários estão prontos para que o aplicativo seja implantado.
- Atingir um certo paralelismo entre o trabalho das equipes de desenvolvimento.

Fases | Transição

O foco da Fase de Transição é assegurar que o software esteja disponível para seus usuários finais.

A Fase de Transição pode atravessar várias iterações e inclui testar o produto em preparação para release e ajustes pequenos com base no feedback do usuário.

Nesse momento do ciclo de vida, o feedback do usuário deve priorizar o ajuste fino do produto, a configuração, a instalação e os problemas de usabilidade; todos os problemas estruturais mais graves devem ter sido trabalhado muito antes no ciclo de vida do projeto.



Objetivos:

- Teste beta para validar o novo sistema em confronto com as expectativas do usuário.
- Conversão de bancos de dados operacionais.
- Treinamento de usuários e equipe de manutenção.
- Introdução a marketing, distribuição e equipe de vendas
- Engenharia voltada para implantação, como preparação, empacotamento e produção comercial, introdução a vendas, treinamento de pessoal em campo
- Atividades de ajuste, como correção de erros, melhoria no desempenho e na usabilidade.
- Avaliação das baselines de implantação tendo como base a visão completa e os critérios de aceitação para o produto.
- Obtenção do consentimento dos envolvidos de que as baselines de implantação estão completas.

Fases | Iterações

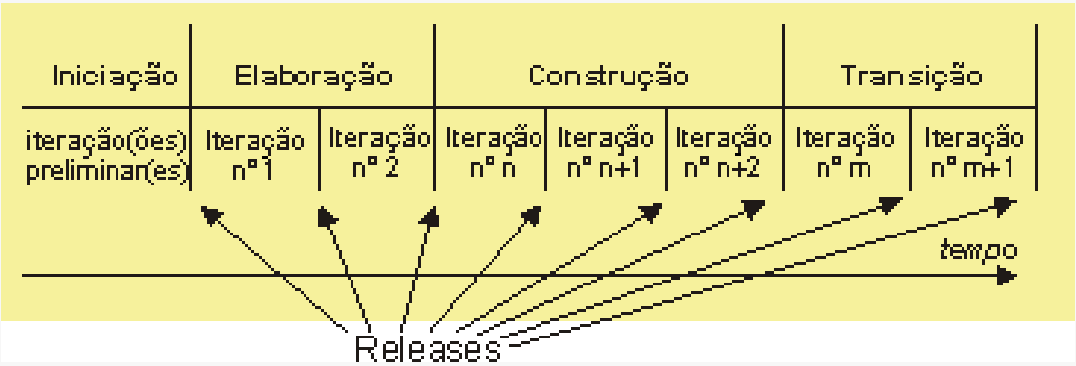
Cada fase do RUP é uma iteração. Do ponto de vista do desenvolvimento, o ciclo de vida do software é uma sucessão de iterações, por meio das quais o software se desenvolve de maneira incremental.

Cada iteração termina com a liberação de um produto executável. Esse produto pode ser um subconjunto da visão completa, mas mesmo assim ser útil do ponto de vista da engenharia ou do usuário. Cada release é acompanhado de artefatos de suporte: descrição do release, documentação do usuário, planos etc., bem como modelos atualizados do sistema.

Um release pode ser interno ou externo. Um release interno é usado apenas pela organização de desenvolvimento, como parte de um marco, ou para fazer uma demonstração para usuários ou clientes. Um release externo é liberado para os usuários finais.

Cada iteração é concluída por um marco menor, onde o resultado da iteração é avaliado em relação aos critérios de êxito do objetivo dessa iteração específica.

Cada fase no RUP pode ser ainda mais dividida em iterações. Uma iteração é um loop completo de desenvolvimento que resulta em um release (interno ou externo) de um produto executável, um subconjunto do produto final em desenvolvimento, que cresce por incrementos, a cada iteração, para se tornar o sistema final.

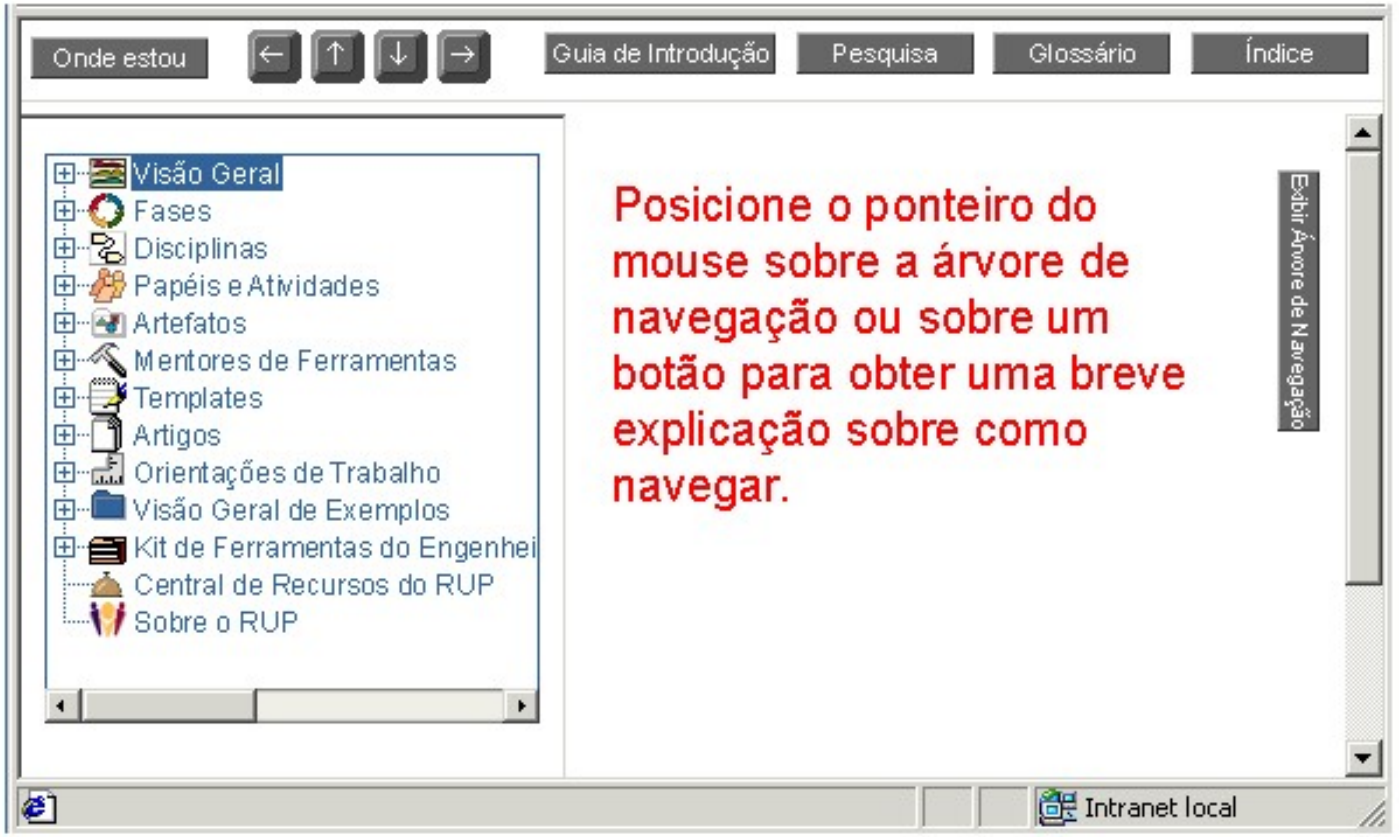


Online

Agora que você teve uma visão geral do Processo Unificado, chegou o momento de se apronfundar, caso seja de seu interesse.

Você deve clicar na imagem ao lado para acessar o RUP Online. Na página que abrir, selecione a opção "RUP 2003".

Navegue por todas as páginas do processo. Caso tenha dúvidas, clique na opção "Guia de Introdução". Leve o tempo que for necessário navegando nas páginas do processo.



Elementos do ambiente de navegação do Rational Unified Process (RUP)

SCRUM

Introdução

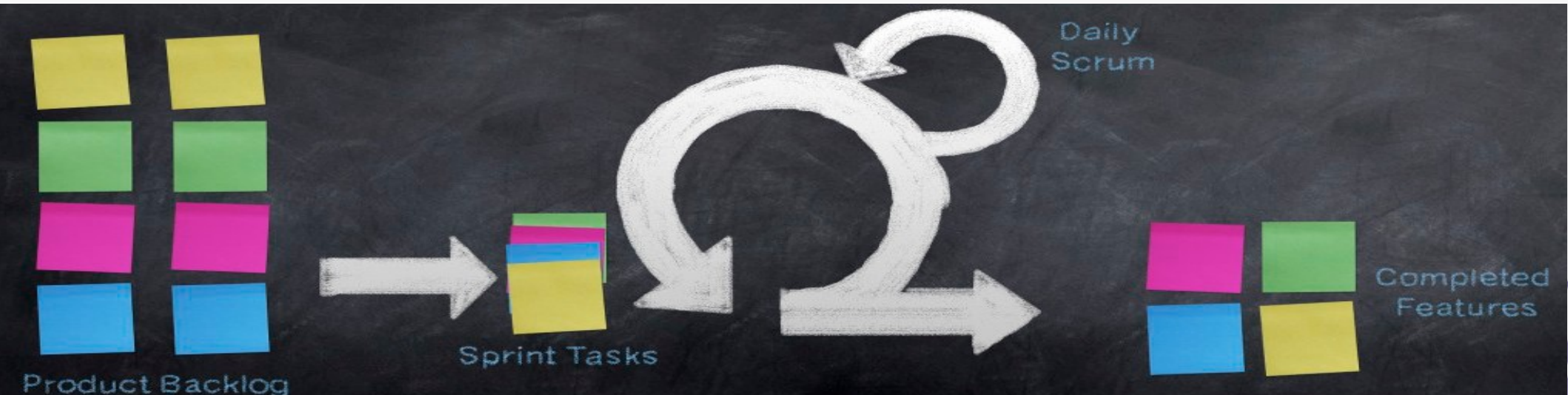
O Scrum é um framework de desenvolvimento iterativo e incremental utilizado no gerenciamento de projetos e desenvolvimento de software ágil.

Possui o foco no gerenciamento e projeto da organização onde é difícil planejar para o futuro.

Não é um processo preditivo, ou seja, ele não descreve o que fazer em cada situação. Ele é usado para trabalhos complexos nos quais é impossível predizer tudo o que irá ocorrer.

Além disso, o Scrum é um conjunto de valores, princípios e práticas que fornecem a base para que a sua organização adicione suas práticas particulares de engenharia e gestão e que sejam relevantes para a realidade da sua empresa.

Apesar de Scrum ter sido destinado para gerenciamento de projetos de software, ele pode ser utilizado em equipes de manutenção de software ou como uma abordagem geral de gerenciamento de projetos.



SCRUM

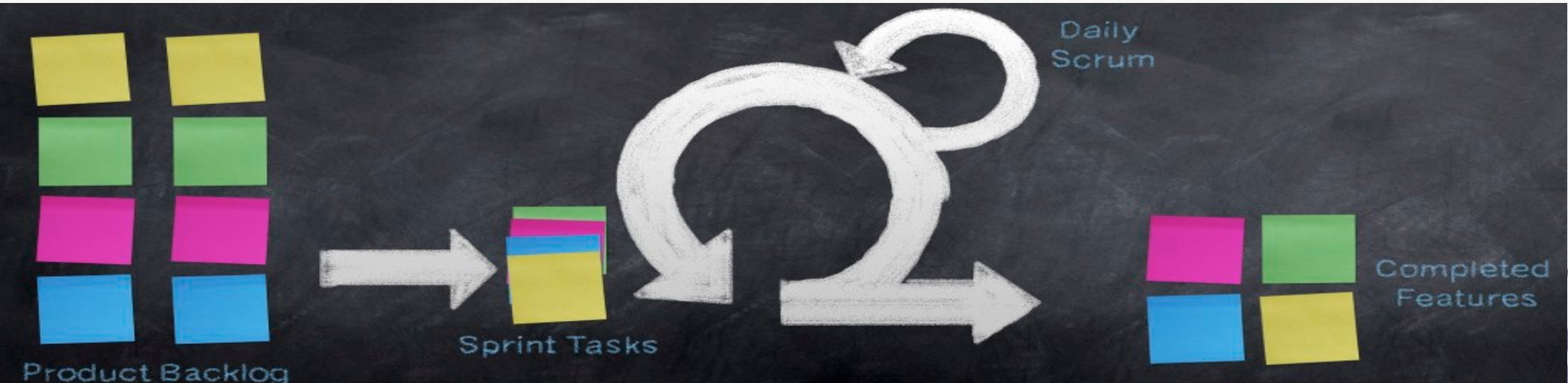
História

Inicialmente, o Scrum foi concebido como um estilo de gerenciamento de projetos em empresas de fabricação de automóveis e produtos de consumo, por Takeuchi e Nonaka no artigo "The New Product Development Game" (Harvard Business Review, Janeiro-Fevereiro 1986).

Eles notaram que projetos usando equipes pequenas e multidisciplinares (cross-functional) produziram os melhores resultados, e associaram estas equipes altamente eficazes à formação Scrum do Rugby (utilizada para reinício do jogo em certos casos).

Jeff Sutherland, John Scumniotales e Jeff McKenna conceberam, documentaram e implementaram o Scrum na empresa Easel Corporation em 1993, incorporando os estilos de gerenciamento observados por Takeuchi e Nonaka.

Em 1995, Ken Schwaber formalizou a definição de Scrum e ajudou a implantá-lo no desenvolvimento de softwares em todo o mundo.



SCRUM

Definição

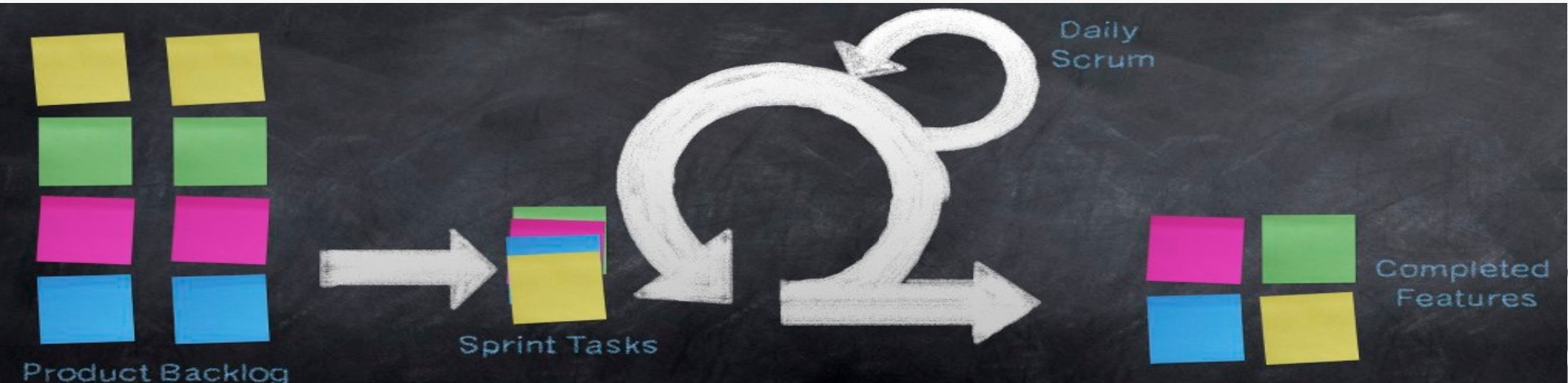
Um framework dentro do qual pessoas podem tratar e resolver problemas complexos e adaptativos, enquanto produtiva e criativamente entregam produtos com o mais alto valor possível.

Dentre suas características, podemos afirmar que ele é leve, simples de entender e muito difícil de dominar.

É um framework estrutural que está sendo usado para gerenciar o desenvolvimento de produtos complexos desde o início de 1990.

Não é um processo ou uma técnica para construir produtos. Em vez disso, é um framework dentro do qual você pode empregar vários processos ou técnicas.

O Scrum deixa claro a eficácia relativa das práticas de gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las.



SCRUM

Definição

O framework Scrum consiste nas equipes do Scrum associados a papéis, eventos, artefatos e regras.

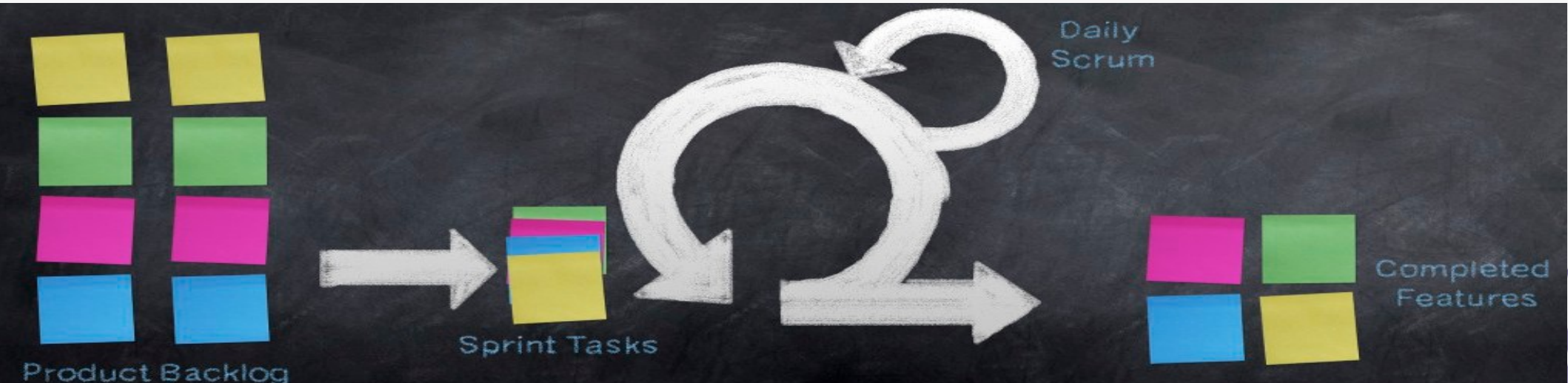
Cada componente dentro do framework serve a um propósito específico e é essencial para o uso e sucesso do Scrum.

As regras do Scrum integram os eventos, papéis e artefatos, administrando as relações e interações entre eles.

Estratégias específicas para o uso do framework Scrum variam e são descritas em vários documentos e livros.

A função primária do Scrum é ser utilizado para o gerenciamento de projetos de desenvolvimento de software. Ele tem sido usado com sucesso para isso, assim como Extreme Programming e outras metodologias de desenvolvimento (muitas vezes são usados em conjunto).

Porém, teoricamente pode ser aplicado em qualquer contexto no qual um grupo de pessoas necessitem trabalhar juntas para atingir um objetivo comum, como iniciar uma escola pequena, projetos de pesquisa científica, ou até mesmo o planejamento de um casamento.



SCRUM

Teoria

O Scrum é fundamentado nas teorias empíricas de controle de processo, ou empirismo.

O empirismo afirma que o conhecimento vem da experiência e de tomada de decisões baseadas no que é conhecido.

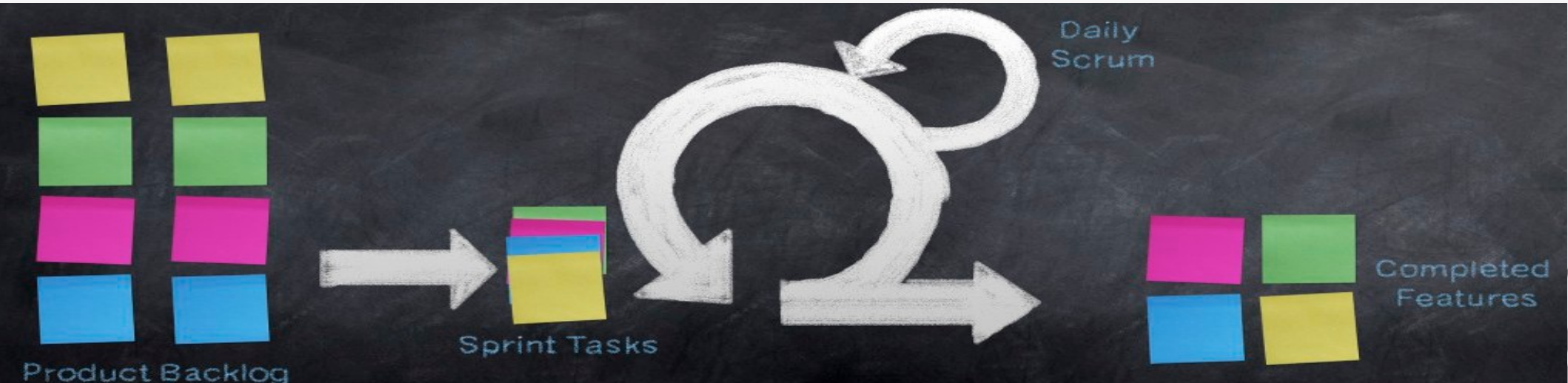
O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos.

Três pilares apoiam a implementação de controle de processo empírico: transparência, inspeção e adaptação.

Transparência

Aspectos significativos do processo devem estar visíveis aos responsáveis pelos resultados. Esta transparência requer aspectos definidos por um padrão comum para que os observadores compartilharem um mesmo entendimento do que está sendo visto.

Sendo assim, uma linguagem comum referindo-se ao processo deve ser compartilhada por todos os participantes.



SCRUM

Teoria

Inspeção

Os usuários Scrum devem, frequentemente, inspecionar os artefatos e o progresso em para detectar variações.

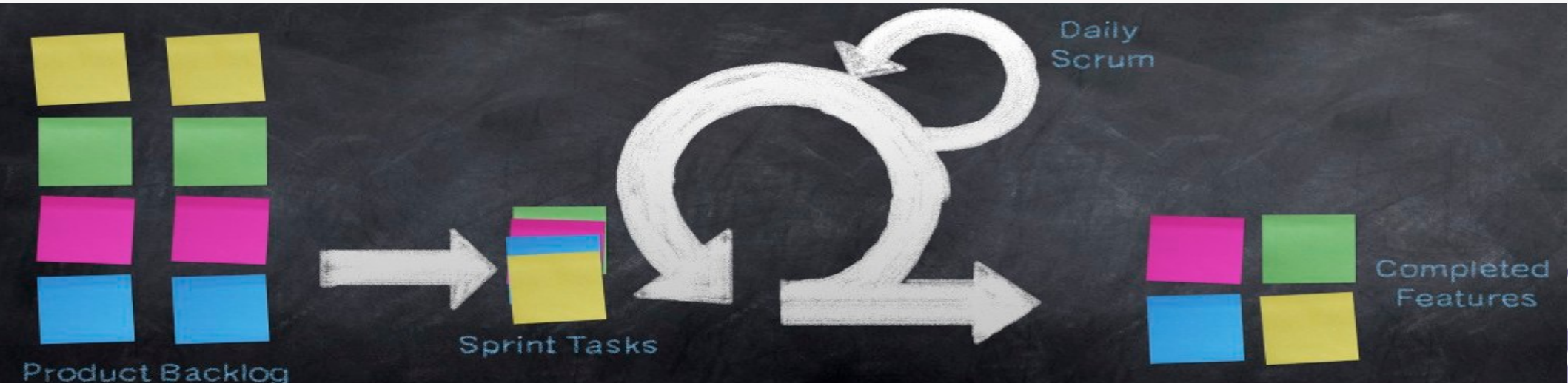
Esta inspeção não deve, no entanto, ser tão frequente que atrapalhe a própria execução das tarefas.

As inspeções são mais benéficas quando realizadas de forma diligente por inspetores especializados no trabalho a se verificar.

Adaptação

Se for detectado que houve um desvio significativo na condução do projeto, deve ser realizado um ajuste o mais breve possível para minimizar mais desvios. O Scrum prescreve quatro Eventos formais, contidos dentro dos limites do Sprint (unidade básica de desenvolvimento), para inspeção e adaptação:

- Reunião de planejamento do Sprint.
- Reunião diária.
- Reunião de revisão do Sprint.
- Retrospectiva do Sprint.



SCRUM

Equipe

A Equipe Scrum é composta pelo Product Owner, a Equipe de Desenvolvimento e o Scrum Master.

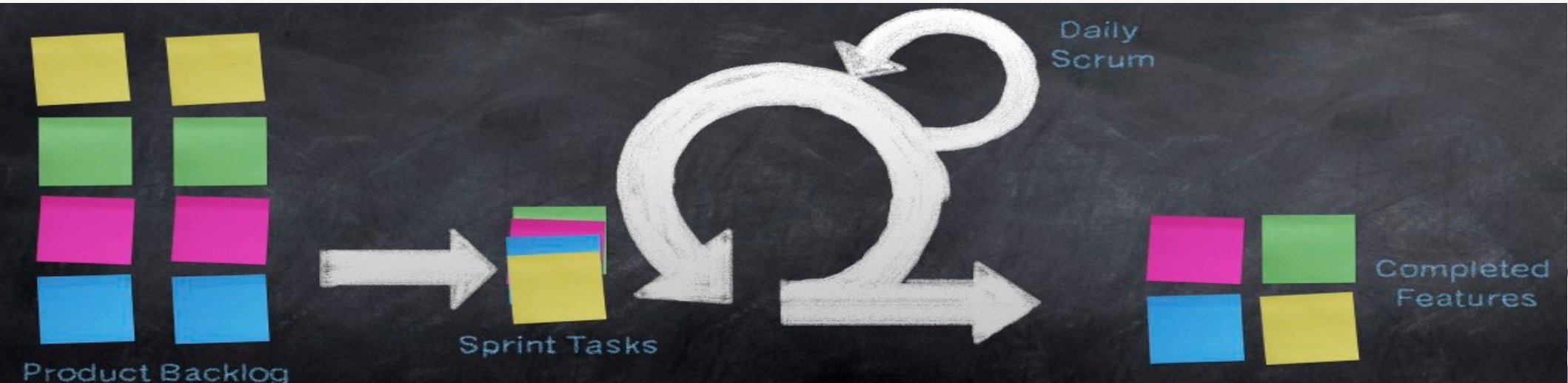
Equipes Scrum são auto-organizáveis, ou seja, escolhem qual a melhor forma para completarem seu trabalho, em vez de serem dirigidas por outros de fora da Equipe.

Além disso, são multifuncionais, ou seja, possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe.

O modelo de equipe no Scrum é projetada para aperfeiçoar a flexibilidade, criatividade e produtividade.

Equipes Scrum entregam produtos de forma iterativa e incremental, maximizando as oportunidades de realimentação.

Entregas incrementais de produto "Pronto" garantem que uma versão potencialmente funcional do produto do trabalho esteja sempre disponível.



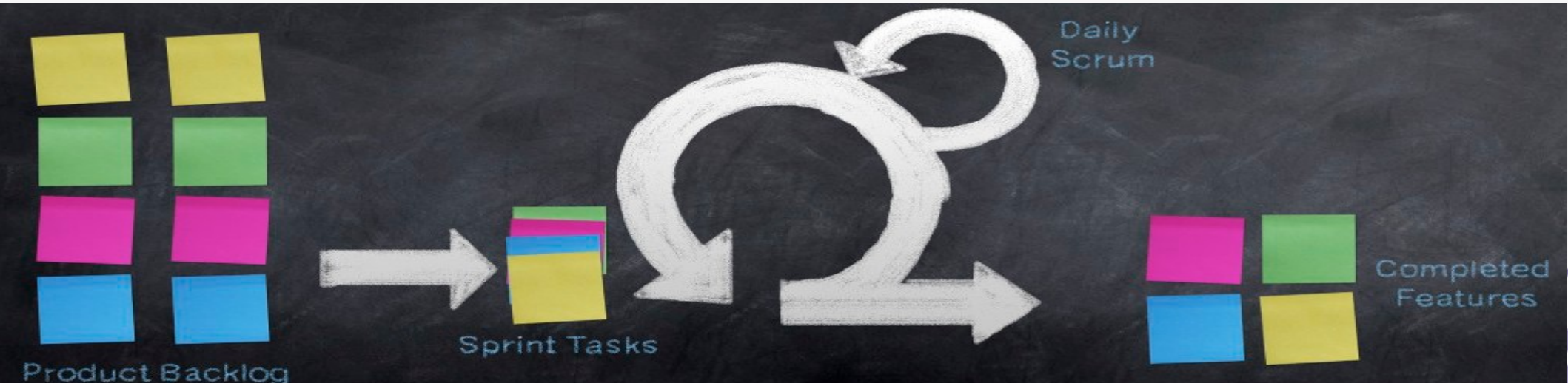
SCRUM

Equipe | Product Owner

É o dono do produto. O responsável por maximizar o valor do produto e do trabalho da Equipe de Desenvolvimento. Como isso é feito pode variar amplamente através das organizações, Equipes Scrum e indivíduos.

O Product Owner é a única pessoa responsável por gerenciar o Product Backlog. O gerenciamento do Product Backlog inclui:

- Expressar claramente os itens do Product Backlog.
- Ordenar os itens do Product Backlog para alcançar melhor as metas e missões.
- Garantir o valor do trabalho realizado pela Equipe de Desenvolvimento.
- Garantir que o Product Backlog seja visível, transparente, claro para todos, e mostrar o que a Equipe Scrum vai trabalhar a seguir.
- Garantir que a Equipe de Desenvolvimento entenda os itens do Product Backlog no nível necessário.



SCRUM

Equipe | Product Owner

O Product Owner pode fazer todo esse trabalho, ou delegar para a Equipe de Desenvolvimento fazê-lo. No entanto, o Product Owner continua sendo o *responsável* pelos trabalhos.

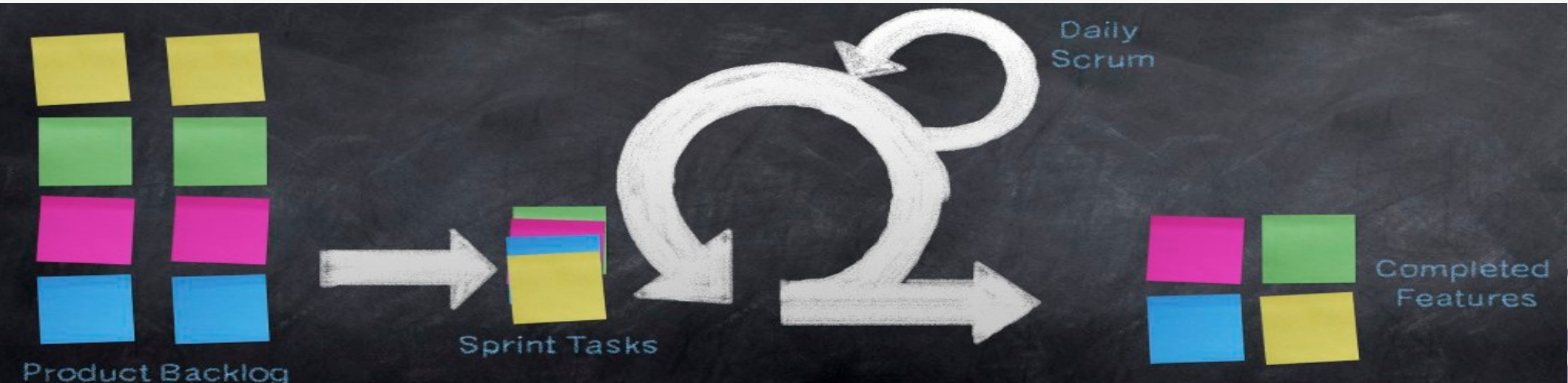
O Product Owner é uma pessoa e não um comitê.

O Product Owner pode representar o desejo de um comitê no Product Backlog, mas aqueles que quiserem uma alteração nas prioridades dos itens de Backlog devem convencer o Product Owner.

Para que o Product Owner tenha sucesso, toda a organização deve respeitar as suas decisões. As decisões do Product Owner são visíveis no conteúdo e na priorização do Product Backlog.

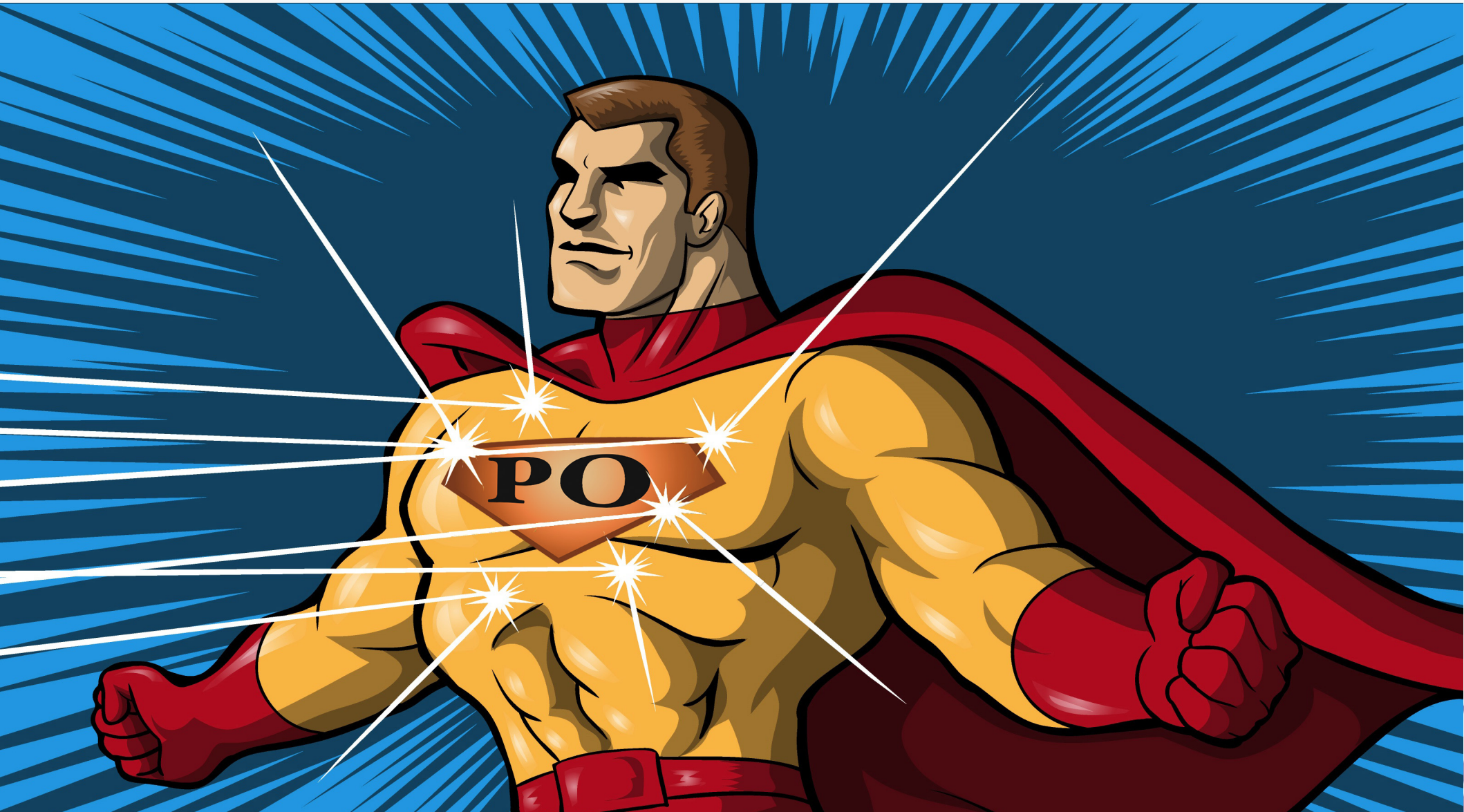
Ninguém tem permissão para falar com a Equipe de Desenvolvimento sobre diferentes configurações de prioridade, e a Equipe de Desenvolvimento não tem permissão para agir sobre o que outras pessoas disserem.

Esse cara deve ser escolhido com muito cuidado!



SCRUM

Equipe | Product Owner



SCRUM

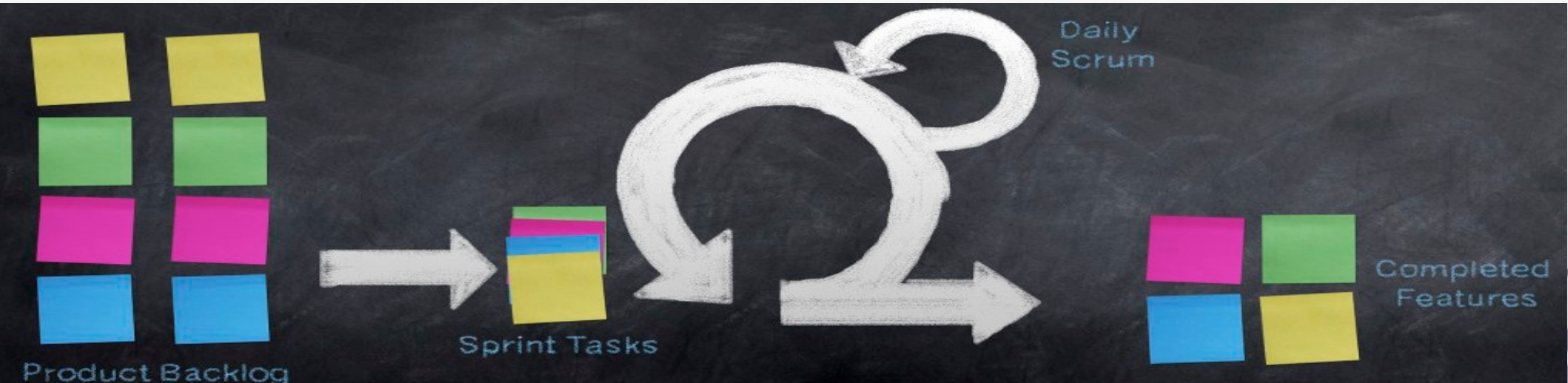
Equipe | Equipe de Desenvolvimento

É formada por profissionais que realizam o trabalho de entregar uma versão utilizável que potencialmente incrementa o produto "Pronto" ao final de cada Sprint. Somente integrantes da Equipe de Desenvolvimento criam incrementos.

São estruturados e autorizados pela organização para organizar e gerenciar seu próprio trabalho.

A Equipe de Desenvolvimento tem as seguintes características:

- Auto-organizadas. Ninguém (nem mesmo o Scrum Master) diz à Equipe de Desenvolvimento como transformar o Product Backlog em incrementos de funcionalidades potencialmente utilizáveis.
- Multifuncionais. Possuem todas as habilidades necessárias para criar o incremento do Produto.
- Todos os integrantes são conhecidos como "Desenvolvedor", independente do seu trabalho e sua especialização.
- Não contém sub equipes dedicadas a domínios específicos de conhecimento, tais como teste ou análise de negócios.



SCRUM

Equipe | Equipe de Desenvolvimento

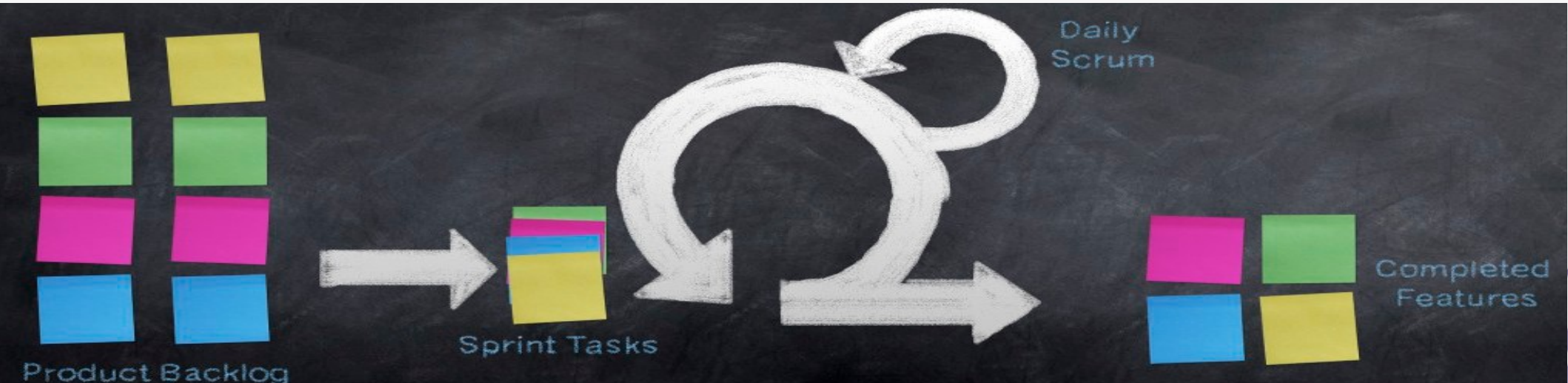
O tamanho ideal da Equipe de Desenvolvimento é pequeno o suficiente para se manter ágil e grande o suficiente para completar uma parcela significativa do trabalho dentro dos limites do Sprint.

Equipes de Desenvolvimento menores podem encontrar restrições de habilidades durante o Sprint, gerando uma Equipe de Desenvolvimento incapaz de entregar um incremento potencialmente utilizável.

Equipes de Desenvolvimento grandes geram muita complexidade para um processo empírico gerenciar.

Os papéis de Product Owner e de Scrum Master não são incluídos nesta contagem, a menos que eles também executem o trabalho do Sprint Backlog.

A quantidade ideal de pessoas na equipe é de três a nove. Menos que três diminui a interação e resulta em baixa produtividade, mais de nova exige muita coordenação.



SCRUM

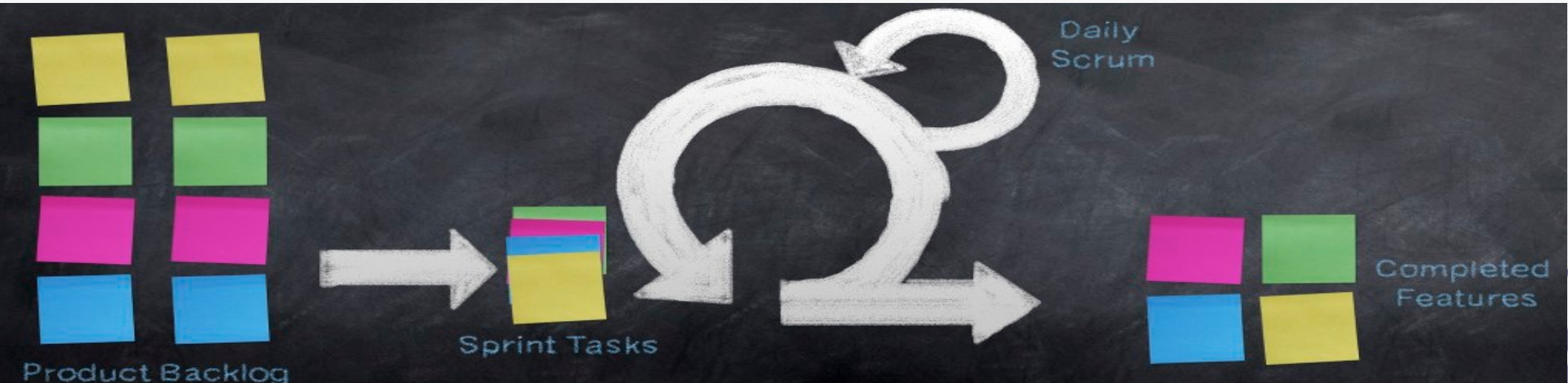
Equipe | Scrum Master

É responsável por garantir que o Scrum seja entendido e aplicado. O Scrum Master faz isso para garantir que a Equipe Scrum entenda e siga a teoria, práticas e regras do Scrum. O Scrum Master é um servo-líder para a Equipe Scrum.

O Scrum Master ajuda aqueles que estão fora da Equipe Scrum a entender quais as suas interações com a Equipe Scrum são úteis e quais não são. O Scrum Master ajuda todos a mudarem estas interações para maximizar o valor criado pela Equipe Scrum.

O Scrum Master serve o Product Owner de várias formas:

- Encontra técnicas para o gerenciamento efetivo do Product Backlog.
- Comunica a visão, objetivo e itens do Product Backlog para a Equipe de Desenvolvimento.
- Ensina a Equipe Scrum a criar itens do Product Backlog de forma clara e concisa.
- Compreende a longo prazo o planejamento do produto no ambiente empírico.
- Facilita os eventos Scrum conforme exigidos ou necessários.



SCRUM

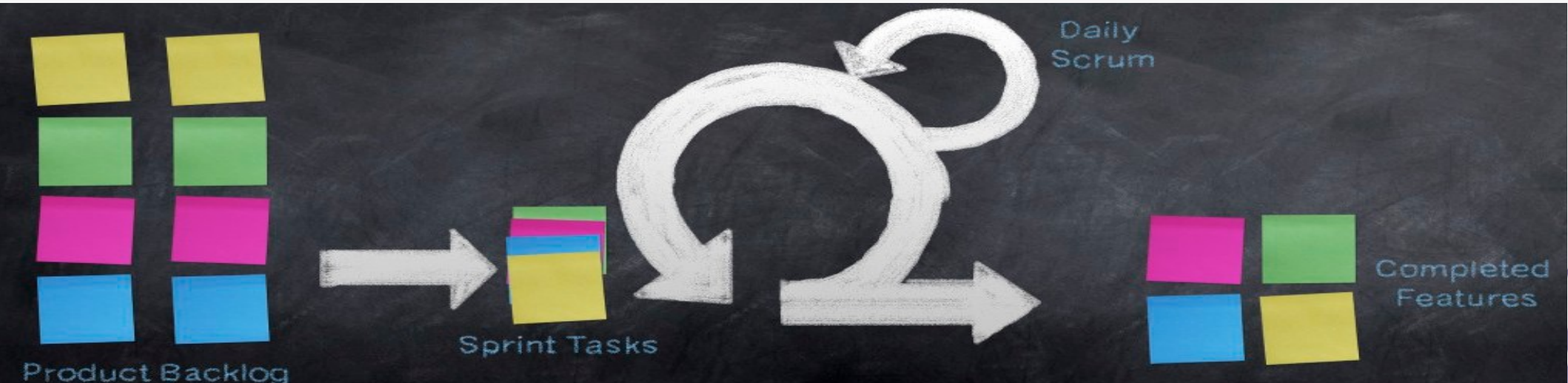
Equipe | Scrum Master

O Scrum Master serve a Equipe de Desenvolvimento de várias formas:

- Treina a Equipe de Desenvolvimento em auto gerenciamento e interdisciplinaridade.
- Ensina e lidera a Equipe na criação de produtos de alto valor.
- Remove impedimentos para o progresso da Equipe de Desenvolvimento.
- Facilita os eventos Scrum conforme exigidos ou necessários.
- Treina a Equipe de Desenvolvimento em ambientes organizacionais nos quais o Scrum não é totalmente adotado e compreendido.

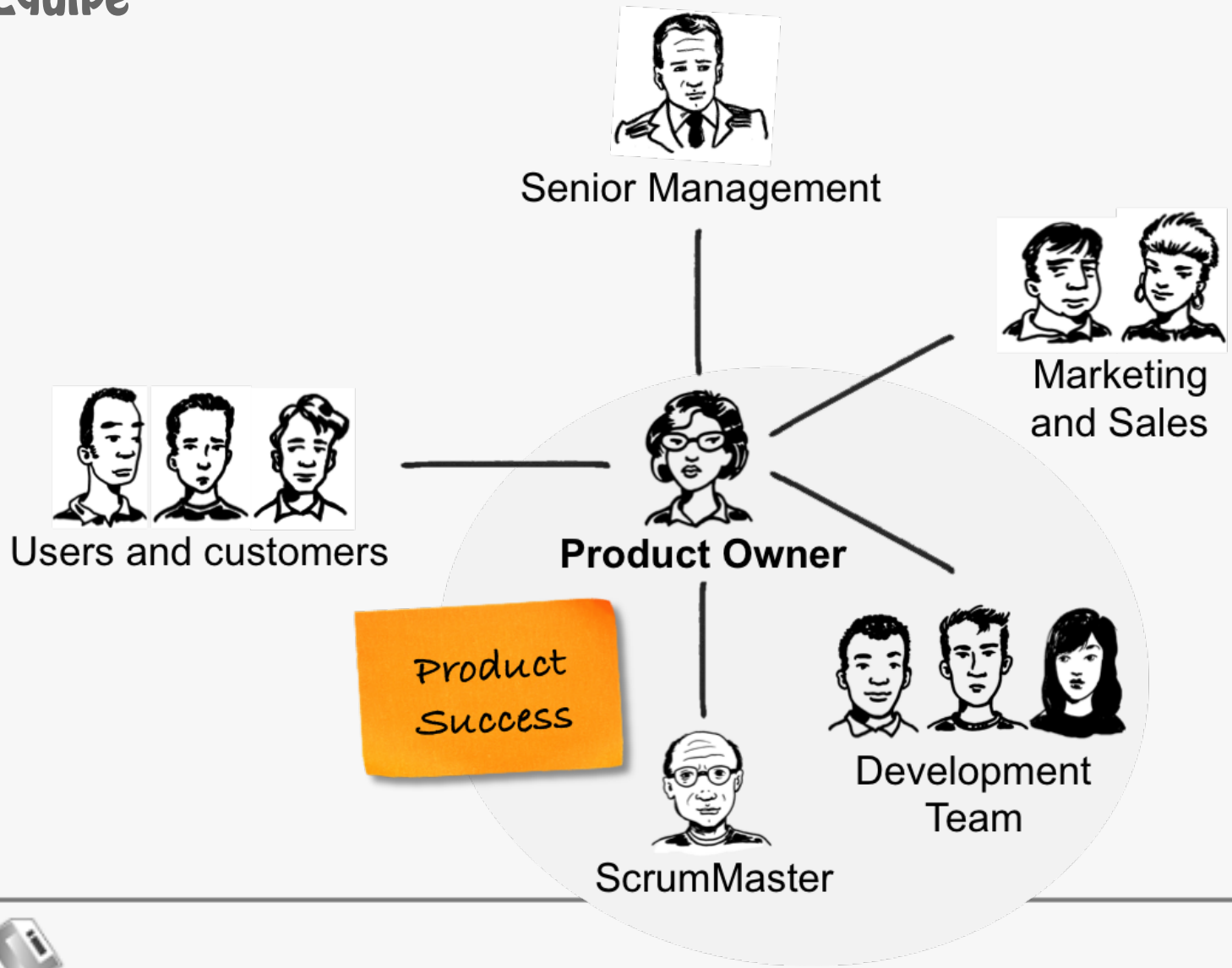
O Scrum Master serve a Organização de várias formas:

- Lidera e treina a organização na adoção do Scrum.
- Planeja implementações Scrum dentro da organização.
- Ajuda colaboradores e partes interessadas a compreender e tornar aplicável o Scrum e o desenvolvimento de produto empírico.
- Causa mudanças que aumentam a produtividade da Equipe Scrum.
- Trabalha com outros Scrum Masters para aumentar a eficácia da aplicação do Scrum.



SCRUM

Equipe



SCRUM

Eventos

Eventos prescritos são usados no Scrum para criar uma rotina e minimizar a necessidade de reuniões não definidas no Scrum.

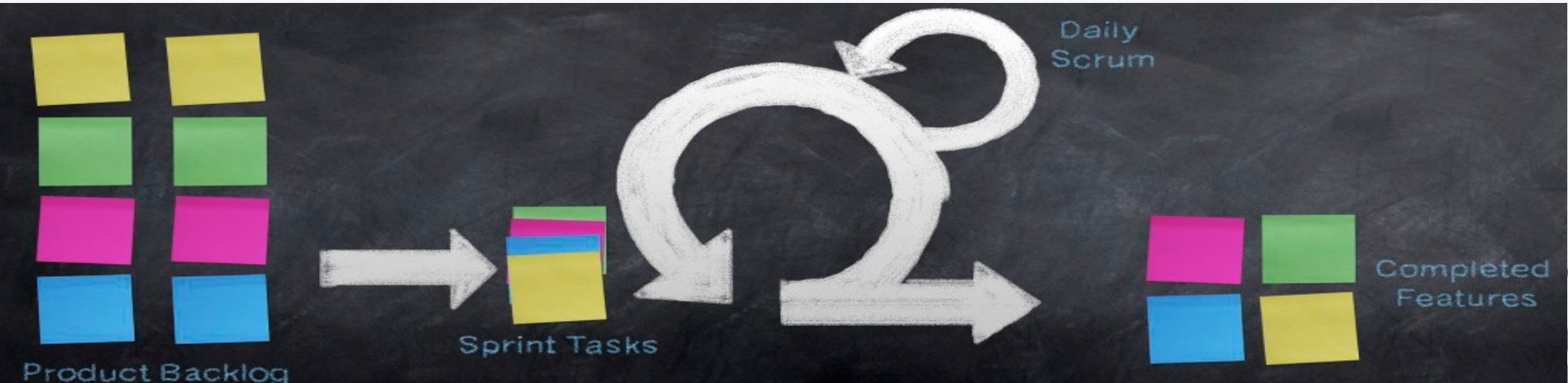
Todos os eventos são "time-boxed", ou seja, tem uma duração máxima definida. Uma vez que o Sprint começa, sua duração é fixada e não pode ser reduzida ou aumentada.

Os eventos restantes podem terminar sempre que o propósito do evento é alcançado, garantindo que uma quantidade adequada de tempo seja gasta sem permitir perdas no processo.

Além do Sprint, que é um container para outros eventos, cada evento no Scrum é uma oportunidade de inspecionar e adaptar alguma coisa.

Estes eventos são especificamente projetados para permitir uma transparência e inspeção criteriosa.

A não inclusão de qualquer um dos eventos resultará na redução da transparência e da perda de oportunidade para inspecionar e adaptar.



SCRUM

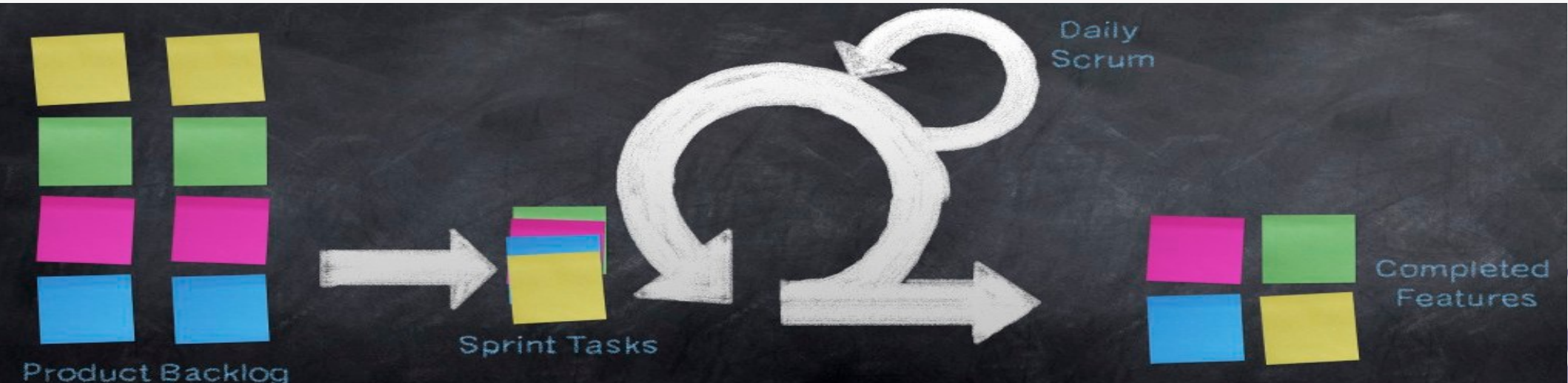
Eventos | Sprint

Um Sprint é a unidade básica de desenvolvimento em Scrum. Sprints tendem a durar entre uma semana e um mês, e são um esforço dentro de uma faixa de tempo (ou seja, restrito a uma duração específica) de comprimento constante.

Cada Sprint é precedido por uma reunião de planejamento (Sprint Planning), onde as tarefas para o Sprint são identificadas e um compromisso estimado para o objetivo do Sprint é definido e seguido por uma reunião de revisão ou de retrospectiva, onde o progresso é revisto e lições para os próximos Sprints são identificadas.

Cada Sprint pode ser considerado um projeto com horizonte não maior que um mês. Cada Sprint tem a definição do que é para ser construído, um plano projetado e flexível que irá guiar a construção, o trabalho e o resultado do produto.

Sprints são limitados a um mês corrido. Quando o horizonte do Sprint é muito longo, a definição do que será construído pode mudar, a complexidade pode aumentar e o risco pode crescer. Sprints permitem previsibilidade que garante a inspeção e adaptação do progresso em direção à meta.



SCRUM

Eventos | Sprint

Durante cada Sprint, a equipe cria um incremento de produto potencialmente entregável (por exemplo, software funcional e testado).

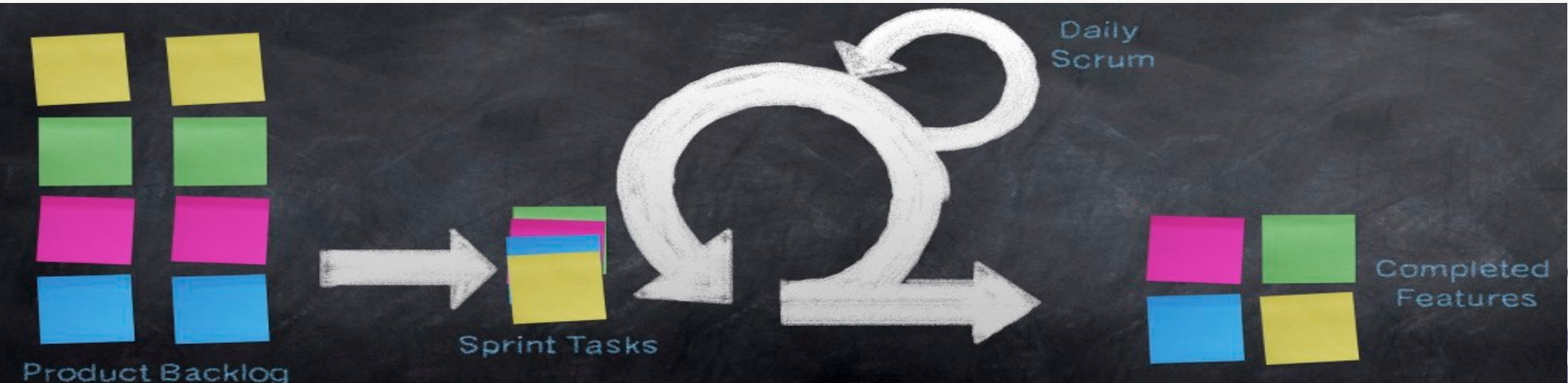
O conjunto de funcionalidades que entram em um Sprint vêm do Product Backlog, que é um conjunto de prioridades de requisitos de alto nível definidos pelo Product Owner.

Um Sprint pode ser cancelada antes do time-boxed do Sprint terminar. Somente o Product Owner tem a autoridade para isso.

O Sprint poderá ser cancelado se o objetivo dele se tornar obsoleto. Isto pode ocorrer se a organização mudar sua direção ou se as condições do mercado ou das tecnologias mudarem.

Geralmente o Sprint deve ser cancelado se ele não faz mais sentido às dadas circunstâncias. No entanto, devido a curta duração do Sprint, raramente cancelamentos fazem sentido.

Cancelamentos de Sprints são frequentemente traumáticos para a Equipe Scrum, e são muito incomuns.



SCRUM

Eventos | Reunião de Planejamento do Sprint

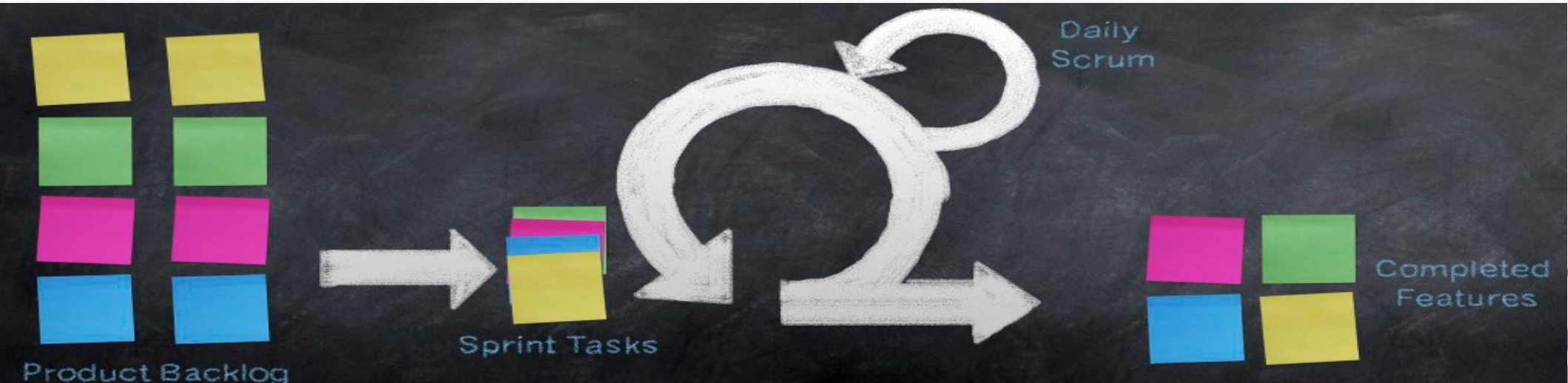
O trabalho a ser realizado durante o Sprint é planejado na Reunião de Planejamento do Sprint (Sprint Planning Meeting). Este plano é criado com o trabalho colaborativo de toda a Equipe Scrum.

Tal reunião possui um time-box com no máximo oito horas para um Sprint de um mês de duração. Para Sprints menores, este evento é usualmente menor. O Scrum Master garante que o evento ocorra e que os participantes entendam seu propósito. O Scrum Master ensina a Equipe Scrum a manter-se dentro dos limites do time-box.

A reunião será dividida em duas partes de quatro horas cada.

Na primeira parte a Equipe Scrum vai analisar o Product Backlog. Veremos posteriormente de que se trata esse artefato. Em suma, ele contém tudo o que o Product Owner deseja para o produto. A Equipe deve então selecionar quais desses itens podem ser feitos durante o período do Sprint.

Na segunda parte, a Equipe define como construirá as funcionalidades durante o Sprint, dando origem ao artefato "Sprint Backlog".



SCRUM

Eventos | Reunião Diária

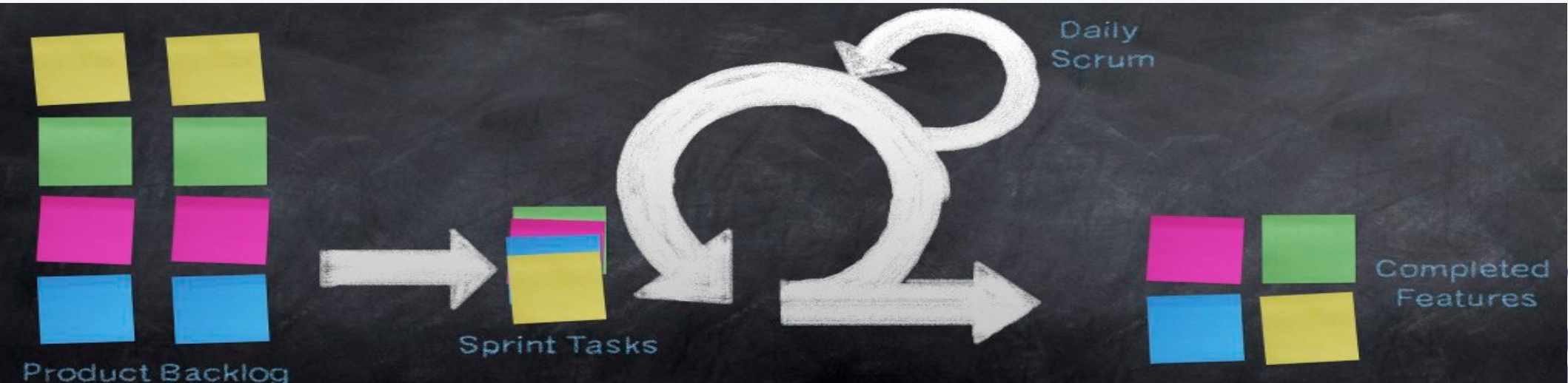
A Reunião Diária (Daily Scrum Meeting) é um evento time-boxed de 15 minutos, para que a Equipe de Desenvolvimento possa sincronizar as atividades e criar um plano para as próximas 24 horas.

Esta reunião é feita para inspecionar o trabalho desde a última Reunião Diária, e prever o trabalho que deverá ser feito antes da próxima Reunião Diária. Em suma, é a revisão do que foi feito no dia anterior e o planejamento para o dia em curso.

Durante a reunião os membros da Equipe de Desenvolvimento esclarecem:

- O que eu fiz ontem que ajudou a Equipe de Desenvolvimento a atender a meta do Sprint?
- O que eu farei hoje para ajudar a Equipe de Desenvolvimento atender a meta do Sprint?
- Eu vejo algum obstáculo que impeça a mim ou a Equipe de Desenvolvimento no atendimento da meta do Sprint?

O Scrum Master assegura que a Equipe de Desenvolvimento tenha a reunião, mas a Equipe de Desenvolvimento é responsável por conduzir a Reunião Diária. Somente os integrantes dessa equipe participam da reunião.



SCRUM

Eventos | Revisão do Sprint

A Revisão do Sprint (Sprint Review) é uma reunião executada no final do Sprint para inspecionar o incremento e adaptar o Product Backlog se necessário.

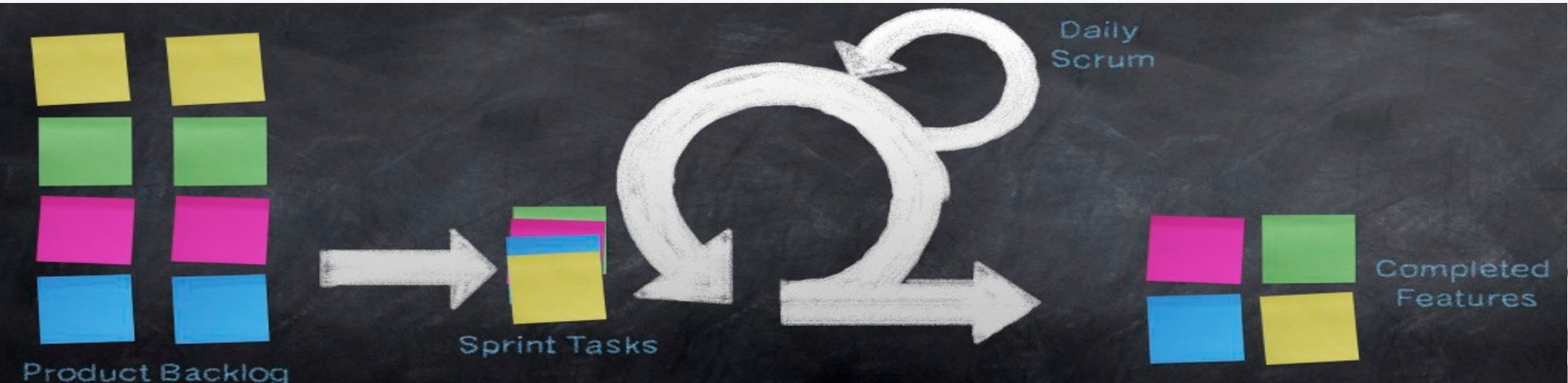
Esta é uma reunião informal, não uma reunião de status, e a apresentação do incremento (o produto que foi feito) destina-se a motivar e obter comentários e promover a colaboração.

É nessa reunião que a equipe vai apresentar o trabalho realizado para os stakeholders. Um trabalho incompleto não pode ser demonstrado.

Esta é uma reunião time-boxed de 4 horas de duração para um Sprint de um mês. Para Sprints menores, este evento é usualmente menor.

O Scrum Master garante que o evento ocorra e que os participantes entendam o seu objetivo. O Scrum Master ensina a todos a manter a reunião dentro dos limites do time-box.

O resultado da Reunião de Revisão do Sprint é um Product Backlog revisado e, em alguns casos, redefinido.



SCRUM

Eventos | Retrospectiva do Sprint

A Retrospectiva do Sprint (Sprint Retrospective) é uma oportunidade para a Equipe Scrum inspecionar a si própria e criar um plano para melhorias a serem aplicadas na próxima Sprint.

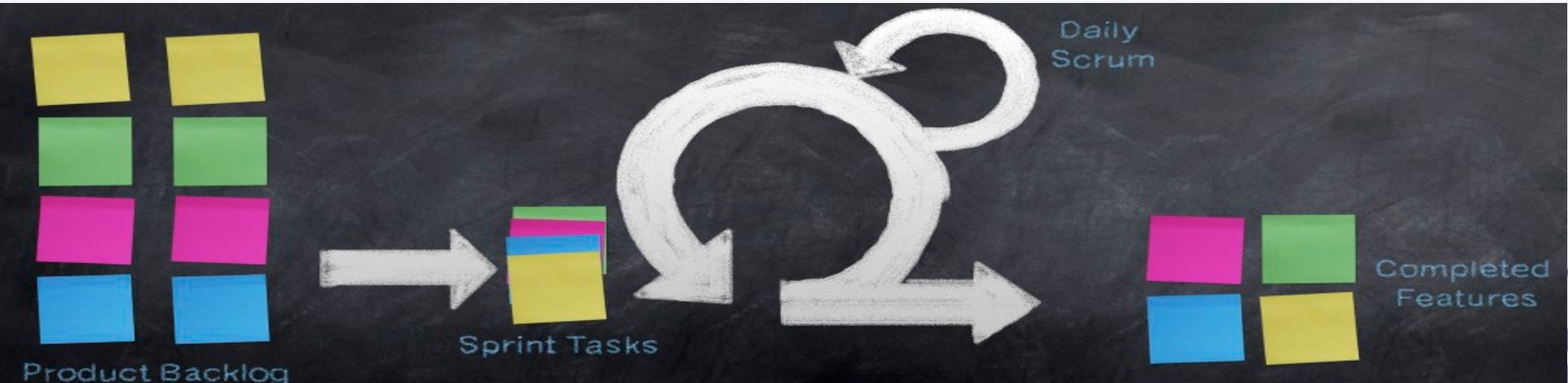
Observe que na reunião de revisão estavam os interessados no produto (stakeholders). Na reunião de retrospectiva participam apenas os membros da Equipe Scrum.

Essa reunião ocorre depois da reunião de revisão e antes da reunião de planejamento do próximo Sprint.

Esta é uma reunião time-boxed de três horas para um Sprint de um mês. Para Sprints menores, este evento é usualmente menor.

Propósitos:

- Inspecionar como o última Sprint foi em relação às pessoas, aos relacionamentos, aos processos e às ferramentas.
- Identificar e ordenar os principais itens que foram bem e as potenciais melhorias.
- Criar um plano para implementar melhorias na forma de trabalho da Equipe Scrum.



SCRUM

Artefatos

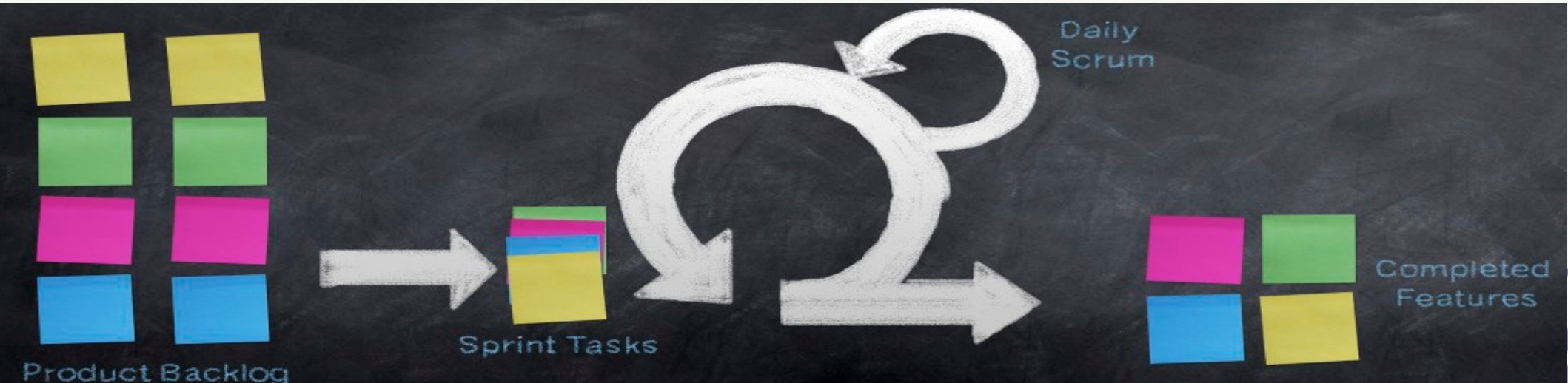
Os artefatos definidos para o Scrum são especificamente projetados para maximizar a transparência das informações chave de modo que todos tenham o mesmo entendimento dos artefatos.

Um backlog é uma lista de itens priorizados a serem desenvolvidos para um software. Este artefato é a principal fonte de informação para o planejamento do Sprint (Sprint Planning). O Product Owner mantém uma lista priorizada de itens de backlog, o Product Backlog, que pode ser repriorizado durante o planejamento do Sprint.

A Equipe seleciona itens do topo do Product Backlog. Eles selecionam somente o quanto de trabalho eles podem executar para terminar.

A Equipe então planeja a arquitetura e o design de como o Product Backlog pode ser implementado.

Os itens do Product Backlog são então destrinchados em tarefas que se tornam o Sprint Backlog.



SCRUM

Artefatos | Product Backlog

É uma lista ordenada de tudo que deve ser necessário no produto.

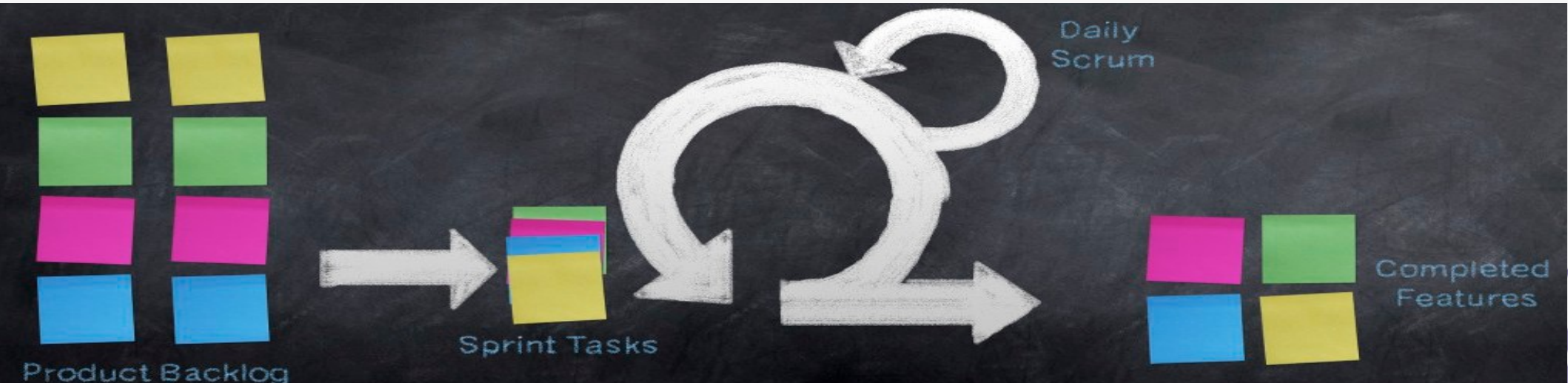
O Product Owner é responsável pelo Product Backlog, incluindo seu conteúdo, disponibilidade e ordenação. O Product Backlog evolui tanto quanto o produto e o ambiente no qual ele será utilizado evoluem.

O Product Backlog lista todas as características, funções, requisitos, melhorias e correções que formam as mudanças que devem ser feitas no produto nas futuras versões.

Os itens do Product Backlog possuem os atributos de descrição, ordem, estimativa e valor.

Os itens do Product Backlog de ordem mais alta (topo da lista) devem ser mais claros e mais detalhados que os itens de ordem mais baixa.

Os itens do Product Backlog que irão ocupar o desenvolvimento no próximo Sprint são mais refinados, de modo que todos os itens possam ser "Prontos" dentro do time-boxed do Sprint.



SCRUM

Artefatos | Sprint Backlog

É um conjunto de itens do Product Backlog selecionados para o Sprint, juntamente com o plano para entregar o incremento do produto e atingir o objetivo do Sprint.

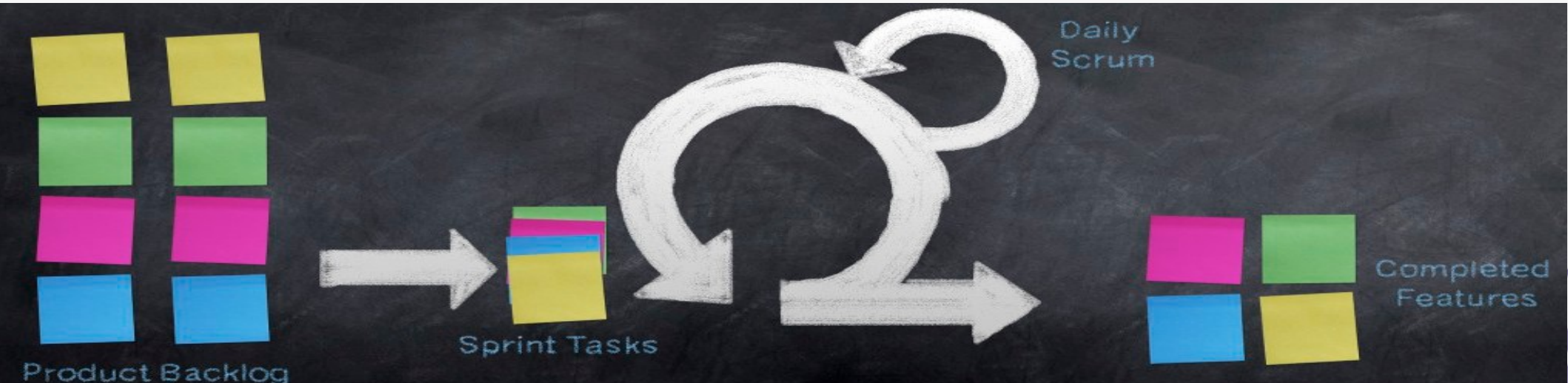
O Sprint Backlog é a previsão da Equipe de Desenvolvimento sobre qual funcionalidade estará no próximo incremento e sobre o trabalho necessário para entregar essa funcionalidade em um incremento "Pronto".

O Sprint Backlog torna visível todo o trabalho que a Equipe de Desenvolvimento identifica como necessário para atingir o objetivo do Sprint.

O Sprint Backlog é um plano com detalhes suficientes que as mudanças no progresso sejam entendidas durante a Reunião Diária.

A Equipe de Desenvolvimento modifica o Sprint Backlog ao longo de toda o Sprint, e o Sprint Backlog vai surgindo durante o Sprint.

Este surgimento ocorre quando a Equipe de Desenvolvimento trabalha segundo o plano e aprende mais sobre o trabalho necessário para alcançar o objetivo do Sprint.



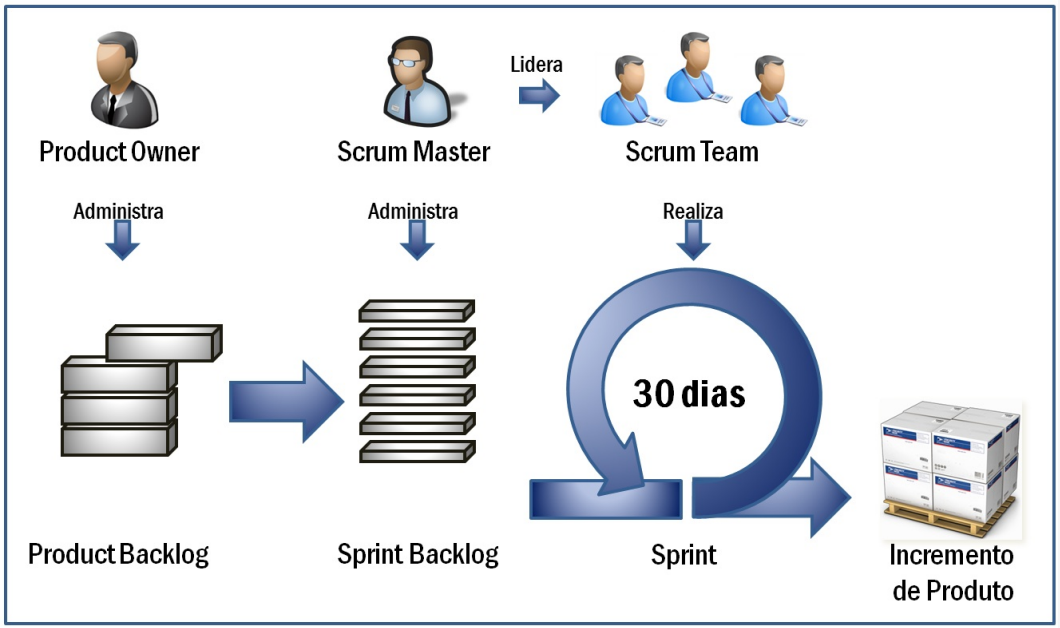
SCRUM

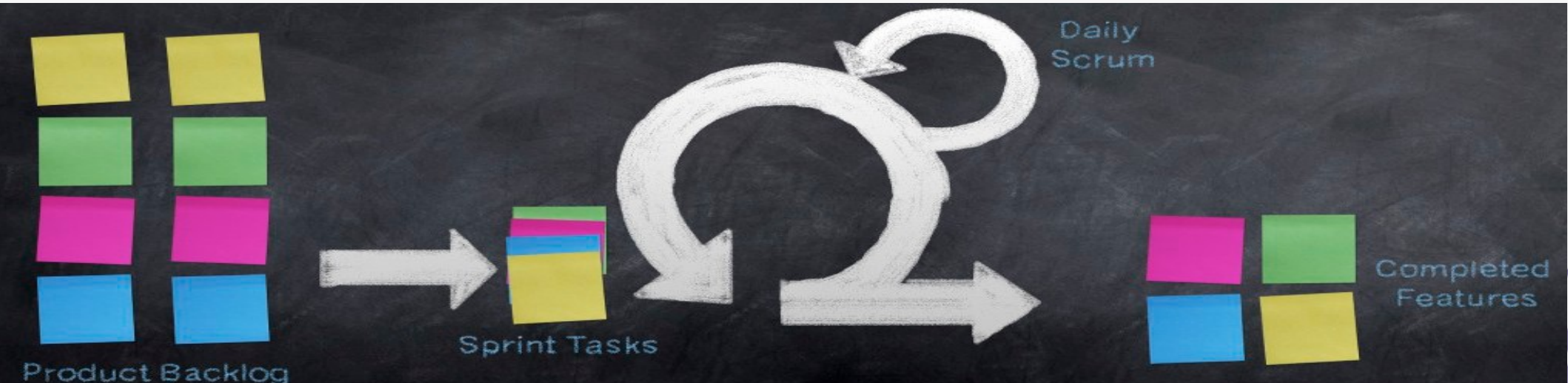
Artefatos | Incremento

O incremento é a soma de todos os itens do Product Backlog completados durante o Sprint e o valor dos incrementos de todas os Sprints anteriores.

Ao final do Sprint um novo incremento deve estar "Pronto", o que significa que deve estar na condição utilizável e atender a definição de "Pronto" da Equipe Scrum.

Este deve estar na condição utilizável independente do Product Owner decidir por liberá-lo realmente ou não.





Introdução

Programação extrema (do inglês eXtreme Programming), ou simplesmente XP, é uma metodologia ágil para equipes pequenas e médias e que irão desenvolver software com requisitos vagos e em constante mudança.

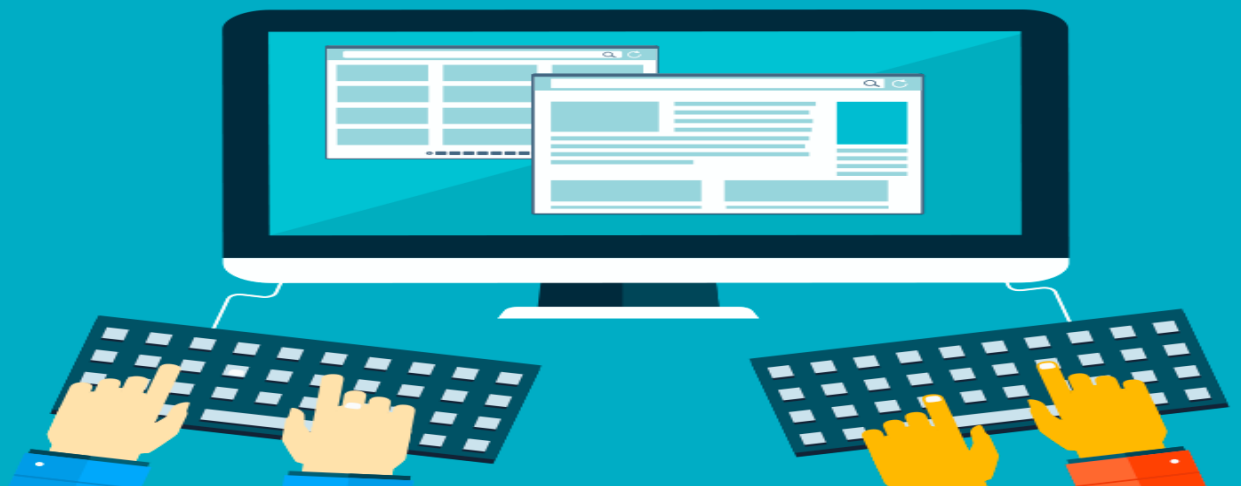
Com esse objetivo, adotam a estratégia de constante acompanhamento e realização de vários pequenos ajustes durante o desenvolvimento de software.

A XP possui alguns valores fundamentais e princípios básicos.

Os valores fundamentais são os seguintes: comunicação, simplicidade, feedback e coragem.

E os princípios básicos: feedback rápido, presumir simplicidade, mudanças incrementais, abraçar mudanças e trabalho de qualidade.

Mas como surgiu a eXtreme Program?



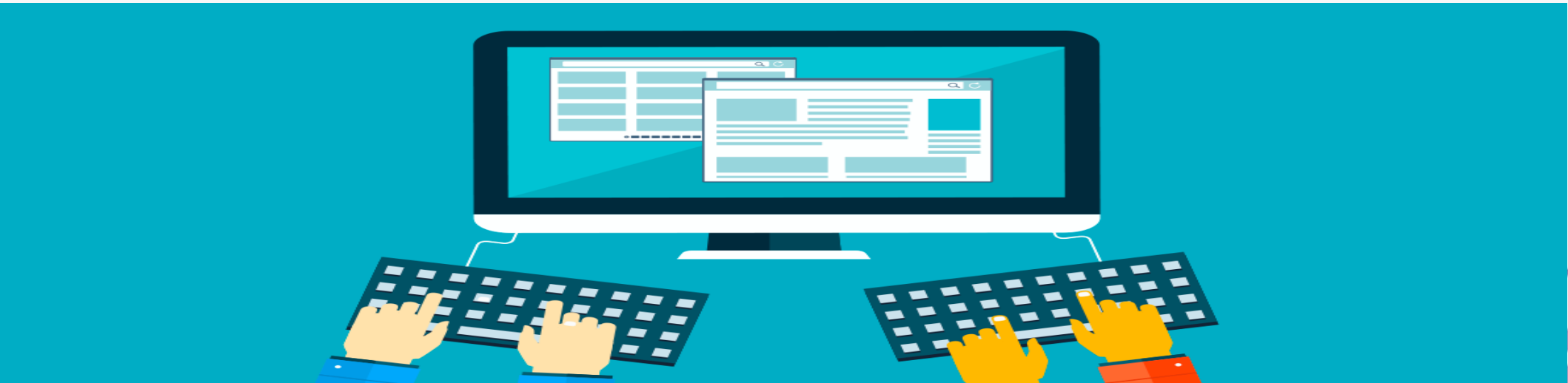
Introdução

A eXtreme Programming resultou da experiência no projeto C3 Payroll na empresa Chrysler. Este projeto consistia da implementação de um sistema de folha de pagamento que já havia fracassado anteriormente utilizando outras metodologias.

Após o sucesso nesse projeto, a XP começou a despontar no meio acadêmico e empresarial e se tornou alvo de inúmeras pesquisas e discussões, além de ser adotada em diversas empresas de software do mundo inteiro e apoiado por grandes "gurus" da orientação a objeto como Kent Beck, Ron Jeffr ies, Martin Fowler e Grady Booch.

O sucesso e a popularidade adquiridos pela XP se devem principalmente aos relatos de bons resultados obtidos em projetos, a motivação dos profissionais envolvidos com a XP e também devido a sua natureza simples e objetiva por se basear em práticas que já provaram sua eficiência no cenário do desenvolvimento de software.

Essas práticas têm como objetivo entregar funcionalidades de forma rápida e eficiente ao cliente. Além disso, a XP foi criada considerando que mudanças são inevitáveis e que devem ser incorporadas constantemente.



Valores Fundamentais

Comunicação

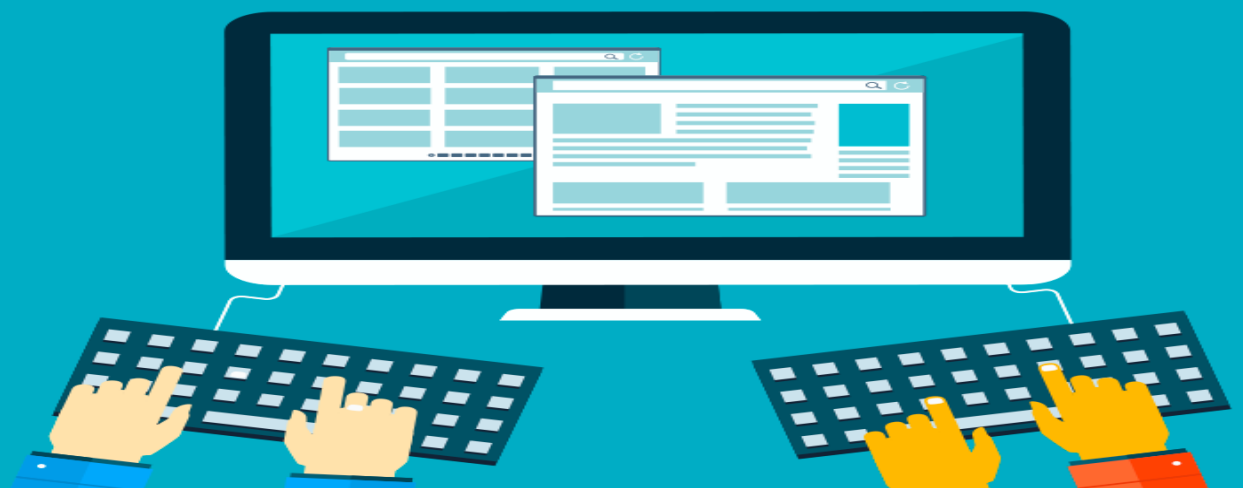
A XP foca em construir um entendimento pessoa a pessoa do problema, com o uso mínimo de documentação formal e com o uso máximo de interação cara a cara entre as pessoas envolvidas no projeto.

As práticas da XP como programação em pares, testes e comunicação com o cliente têm o objetivo de estimular a comunicação entre gerentes, programadores e clientes.

Simplicidade

A XP sugere que cada membro da equipe adote a solução mais fácil que possa funcionar. O objetivo é fazer aquilo que é mais simples hoje e criar um ambiente em que o custo de mudanças no futuro seja baixo.

O objetivo dessa abordagem adotada pela XP é evitar a construção antecipada de funcionalidades, como é feita em muitas metodologias tradicionais, que acabam muitas vezes nem sendo usadas.



Valores Fundamentais

Feedback

Os programadores obtêm feedbacks sobre a lógica dos programas escrevendo e executando casos de teste.

Os clientes obtêm feedbacks através dos testes funcionais criados para todas as estórias (casos de uso simplificados).

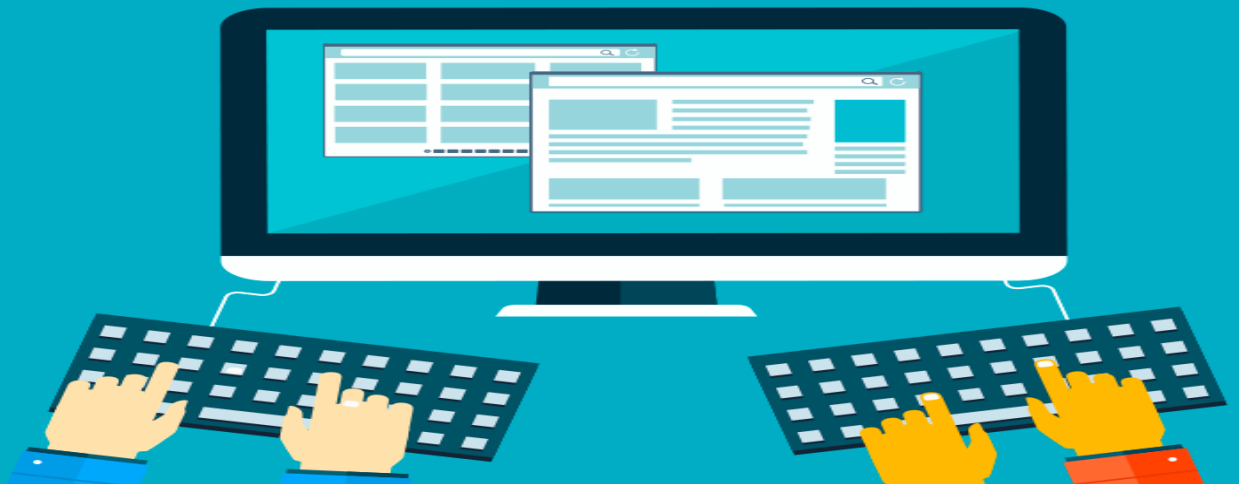
O feedback é importante, pois possibilita que as pessoas aprendam cada vez mais sobre o sistema e assim corrijam os erros e melhorem o sistema.

Coragem

Necessária para que realmente se aplique XP como deve ser aplicada.

Exemplos de atitude que exigem coragem são: alterar código já escrito e que está funcionando, jogar código fora e reescrever tudo de novo, permitir código compartilhado por todos, etc.

Estes exemplos de atitudes podem ser necessários para trazer melhorias ao projeto e não devem ser evitadas simplesmente devido ao medo de tentá-las



Princípios Básicos

Feedback Rápido

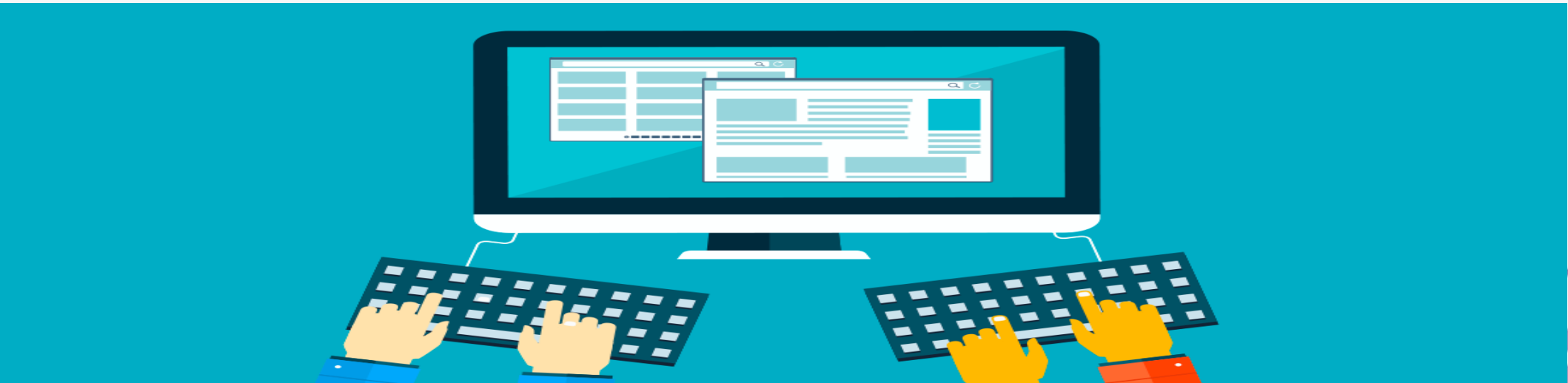
A ideia da XP é que os participantes de um projeto como clientes, programadores e gerentes devem estar sempre se comunicando para facilitar o aprendizado coletivo sobre o projeto que está sendo desenvolvido e de alertar rapidamente sobre dúvidas, riscos e problemas para facilitar eventuais ações de contingência.

Por isso o princípio básico do feedback rápido.

Presumir Simplicidade

Todo problema deve ser tratado para ser resolvido da forma mais simples possível.

A XP afirma que se deve fazer um bom trabalho (testes, refatoração, comunicação) para resolver hoje os problemas de hoje e confiar na sua habilidade de adicionar complexidade no futuro quando for necessário.



Princípios Básicos

Mudanças Incrementais

Quando muitas mudanças são realizadas todas de uma vez, não se obtém um bom resultado.

Em vez disso, esse princípio da XP diz que as mudanças devem ser incrementais e feitas aos poucos.

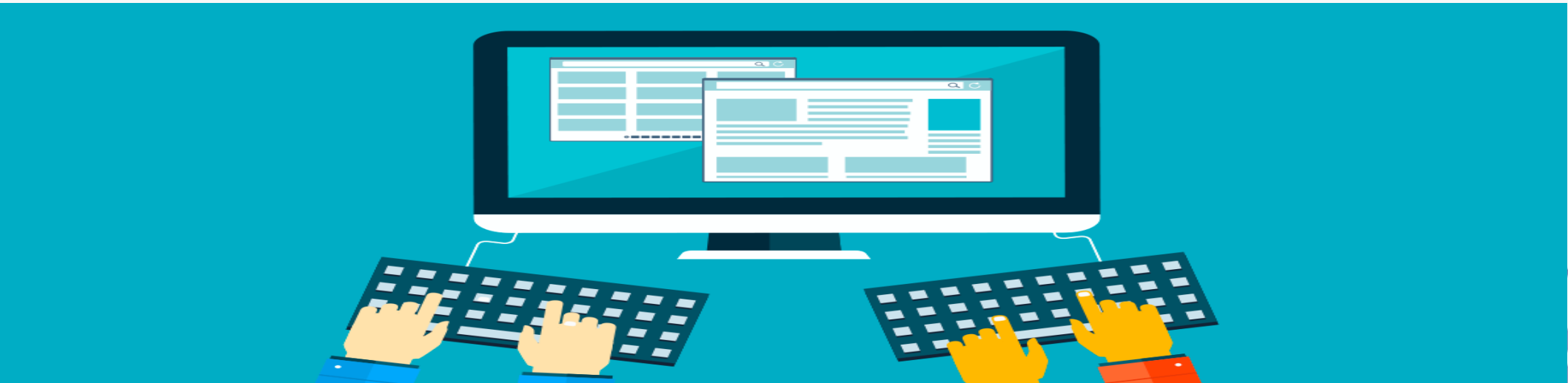
Os programadores têm a liberdade para estar sempre fazendo alterações de melhoria no código e as mudanças de requisitos são incorporadas de forma incremental.

Abraçar Mudanças

A XP procura facilitar o ato de incluir alterações através do uso de vários princípios e práticas.

A ideia da XP é a de que mudanças devem ser sempre bem vindas independentemente do estágio de evolução do projeto.

Isso é muito benéfico em projetos cujos requisitos são bastante voláteis devido aos clientes não saberem o que querem.

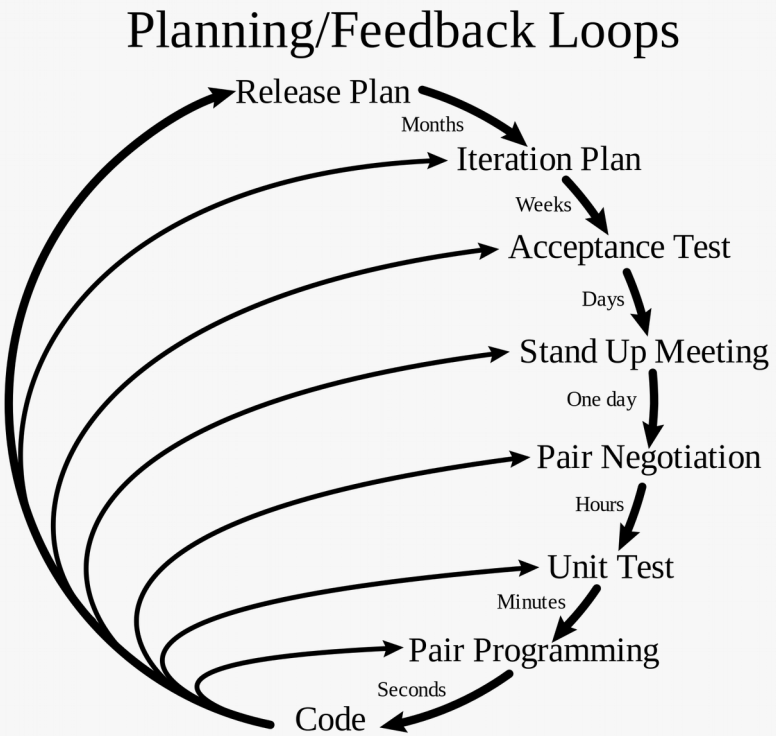


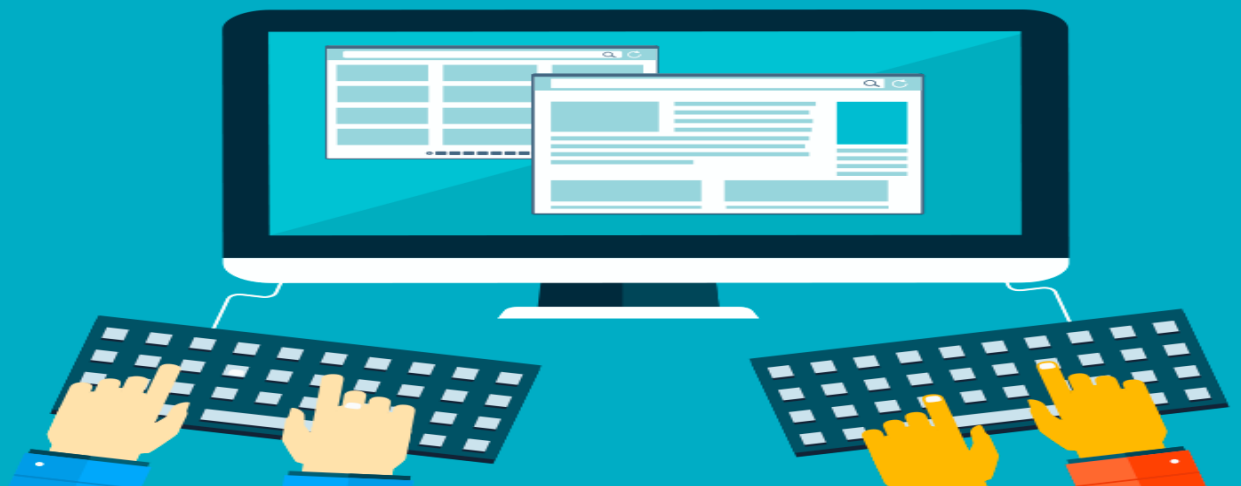
Princípios Básicos

Trabalho de Qualidade

Na XP, qualidade não é um critério opcional, mas sim obrigatório.

Embora a definição de qualidade possa variar de pessoa para pessoa, a XP trata qualidade no sentido de se ter um sistema que atenda aos requisitos do cliente, que rode 100% dos casos de teste e que agregue o maior valor possível para o negócio do cliente.





Práticas

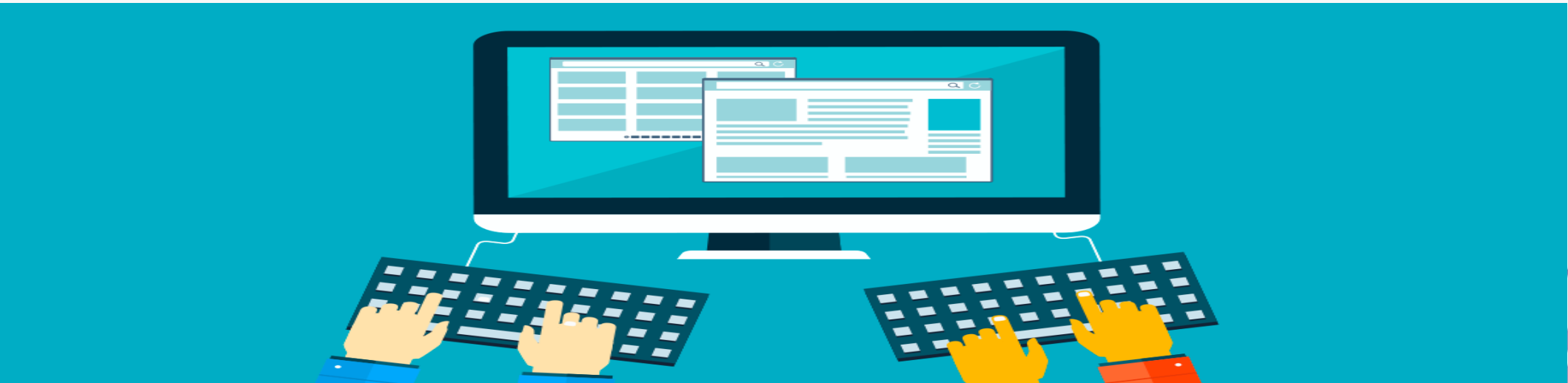
A XP possui doze práticas que consistem no núcleo principal do processo e que foram criadas com base nos ideais pregados pelos valores fundamentais e princípios básicos apresentados anteriormente.

Há uma confiança muito grande na sinergia entre elas, os pontos fracos de cada uma são superados pelos pontos fortes de outras.

Segundo um dos criadores da XP, Kent Beck, estas práticas não são novidades, mas sim práticas que já vêm sendo utilizadas há muitos anos, com eficiência, em projetos de software.

Muitas das práticas da XP não são unanimidades dentro da comunidade de desenvolvimento de software, como por exemplo, a programação em pares. Imagina você programando pela manhã com um cara fungando no teu cangote e se metendo no teu trabalho e a tarde você do lado dele observando ele programar!

No entanto, o valor e benefícios de tais práticas devem ser avaliados em conjunto, pois elas foram criadas para serem usadas coletivamente, de forma a reduzir as fraquezas umas das outras.



Práticas | Jogo do Planejamento

O desenvolvimento é feito em iterações semanais. No início da semana, desenvolvedores e cliente reúnem-se para priorizar as funcionalidades.

Essa reunião recebe o nome de Jogo do Planejamento (Planning Game).

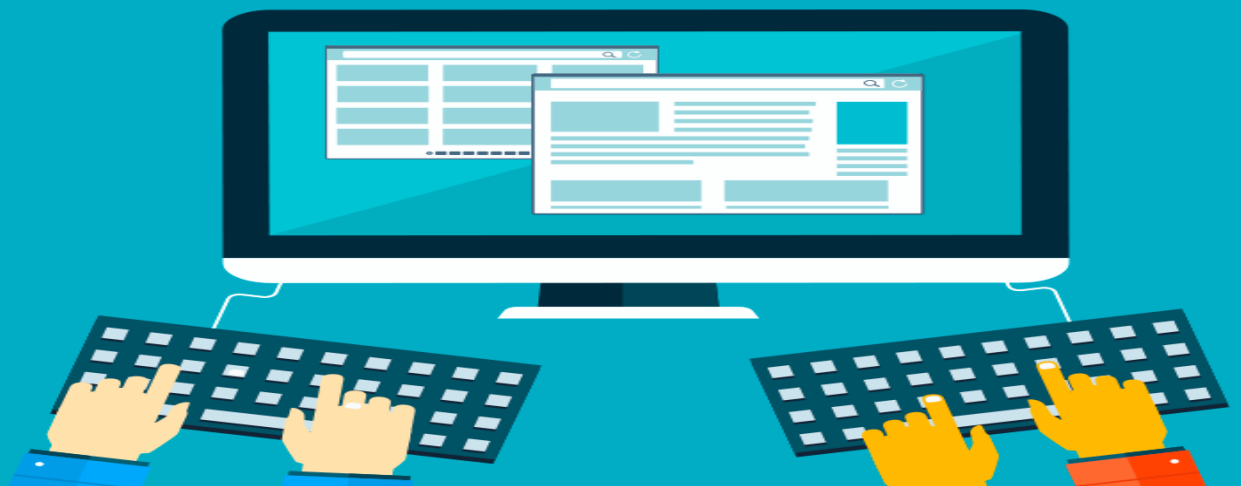
Nela, o cliente identifica prioridades e os desenvolvedores as estimam.

O cliente é essencial neste processo e assim ele fica sabendo o que está acontecendo e o que vai acontecer no projeto.

Como o escopo é reavaliado semanalmente, o projeto é regido por um contrato de escopo negociável, que difere significativamente das formas tradicionais de contratação de projetos de software.

Ao final de cada semana, o cliente recebe novas funcionalidades, completamente testadas e prontas para serem postas em produção.

O planejamento de um release e das iterações são feitos com base nas histórias (casos de uso simplificados).



Práticas | Releases Pequenos | Metáfora

É importante manter releases pequenos (Small Releases).

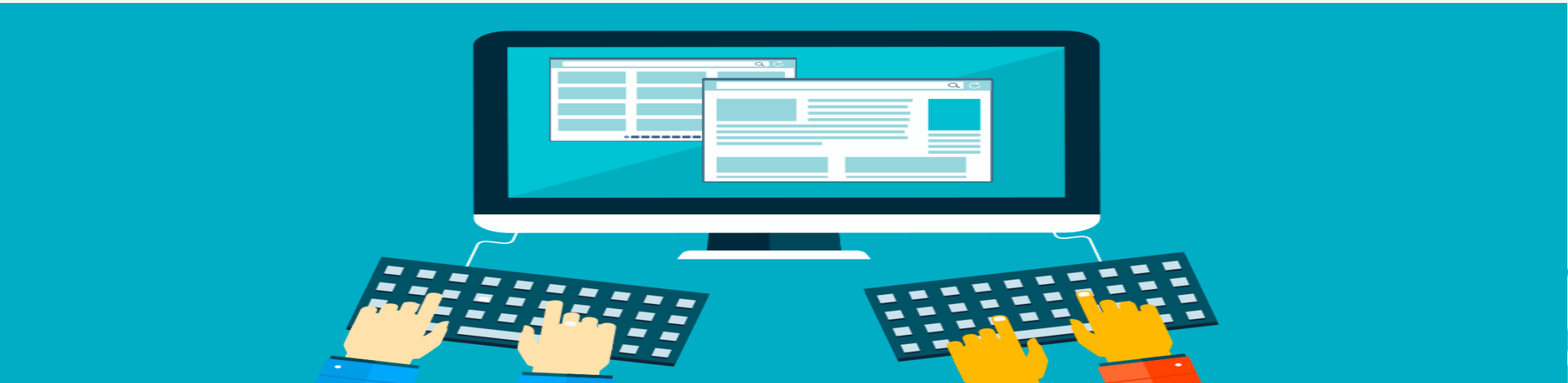
A liberação de pequenas versões funcionais do projeto auxilia muito no processo de aceitação por parte do cliente, que já pode testar uma parte do sistema que está comprando.

As versões chegam a ser ainda menores que as produzidas por outras metodologias incrementais, como o RUP.

A metáfora (Metaphor) procura facilitar a comunicação com o cliente, entendendo a realidade dele.

O conceito de rápido para um cliente de um sistema jurídico é diferente para um programador experiente em controlar comunicação em sistemas em tempo real, como controle de tráfego aéreo.

É preciso traduzir as palavras do cliente para o significado que ele espera dentro do projeto.



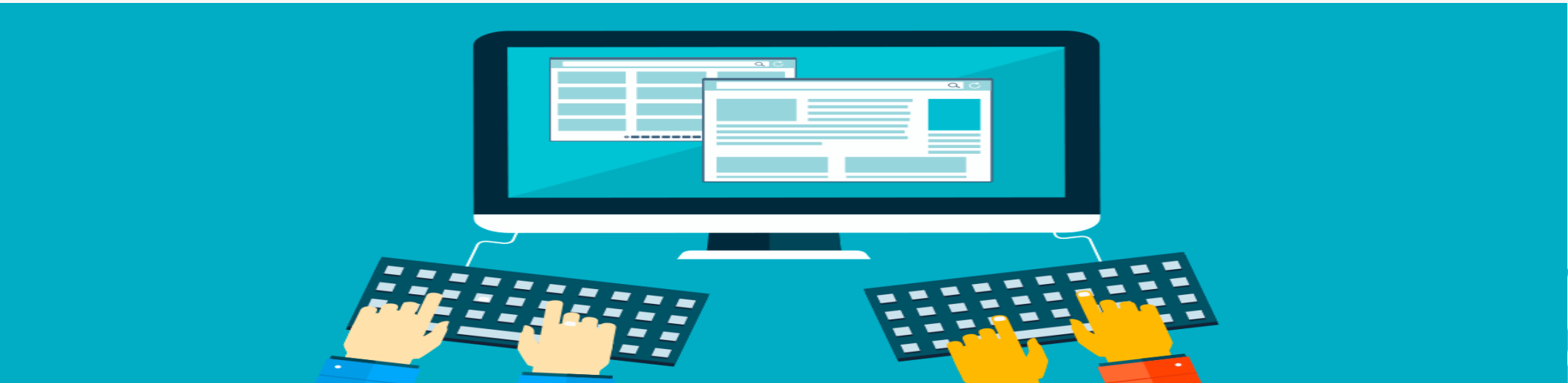
Práticas | Projeto Simples

Simplicidade é um dos valores fundamentais da XP.

Projeto simples significa dizer que caso o cliente tenha pedido que na primeira versão apenas o usuário "teste" possa entrar no sistema com a senha "123" e assim ter acesso a todo o sistema, você vai fazer o código exato para que esta funcionalidade seja implementada, sem se preocupar com sistemas de autenticação e restrições de acesso.

Pode-se explicar esta prática em duas partes: a primeira diz que devem ser projetadas as funcionalidades que já foram definidas e não as que poderão ser definidas futuramente. A segunda diz que deve ser feito o melhor projeto que possa entregar tais funcionalidades.

Esta prática tem o intuito de enfatizar que o projeto simples deve se concentrar em soluções simples e bem estruturadas para os problemas de hoje e que não se deve perder tempo investindo em soluções genéricas que procurem atender a funcionalidades futuras.



Práticas | Desenvolvimento Orientado a Testes

Conhecido como Test Driven Development.

Primeiro crie os testes unitários (unit tests) e depois crie o código para que os testes funcionem.

Esta abordagem é complexa no início, pois vai contra o processo de desenvolvimento de muitos anos.

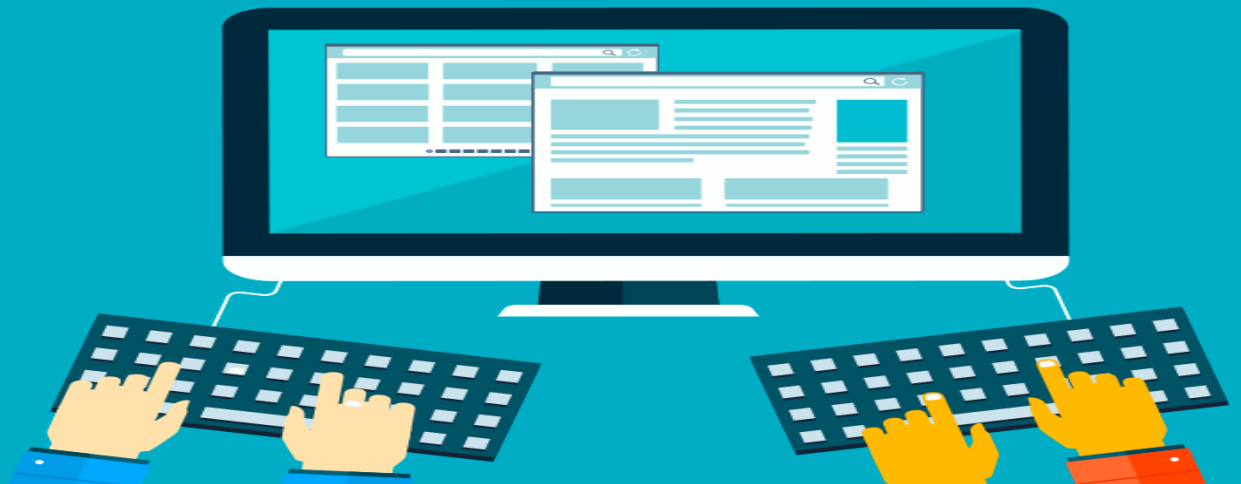
Só que os testes unitários são essenciais para que a qualidade do projeto seja mantida.

Os testes unitários são feitos para verificar tudo que possa dar errado.

Não é preciso escrever testes de unidade para todos os métodos. Os testes unitários são automatizados, e toda vez que o programador escrever código, ele irá verificar se o sistema passa em todos os testes.

Os testes funcionais são usados para verificação, junto ao cliente, do sistema como um todo.

Os testes servem como um mecanismo para assegurar que o sistema está sempre rodando livre de erros, e também servem para dar feedback aos programadores e clientes sobre as falhas encontradas.



Práticas | Refatoração | Semana de 40 Horas

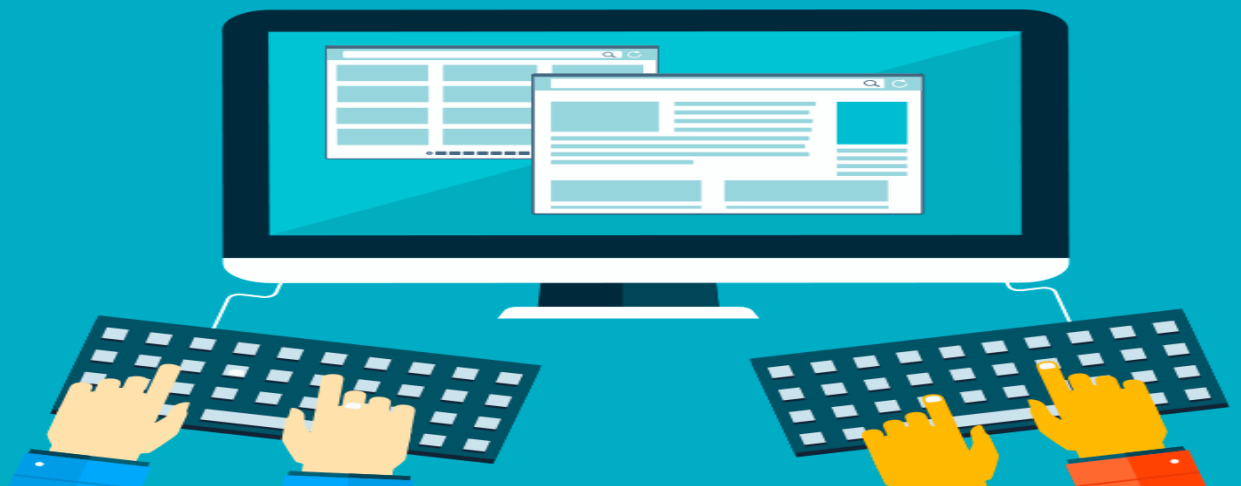
A Refatoração (Refactoring) é um processo que permite a melhoria continua da programação, com o mínimo de introdução de erros e mantendo a compatibilidade com o código já existente.

Refatorar melhora a clareza (leitura) do código, divide-o em módulos mais coesos e de maior reaproveitamento, evitando a duplicação de código-fonte.

O termo utilizado é Sustainable Pace (ritmo sustentável). Trabalhar com qualidade, buscando ter ritmo de trabalho saudável (40 horas por semana, 8 horas por dia), sem horas extras.

Horas extras são permitidas quando trouxerem produtividade para a execução do projeto.

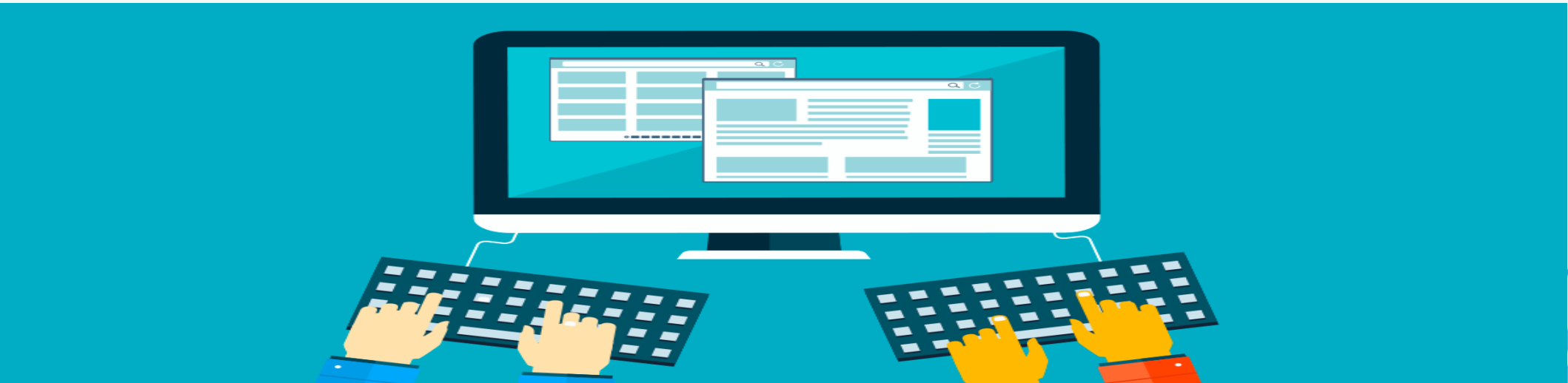
Outra prática que se verifica neste processo é a prática de trabalho energizado, onde se busca trabalho motivado sempre. Para isto o ambiente de trabalho e a motivação da equipe devem estar sempre em harmonia.



Práticas | Programação em Pares | Propriedade Coletiva

Todo o código produzido na XP é escrito por um par de programadores, que possuem papéis distintos, sentados lado a lado e olhando para o computador. Um parceiro será responsável pela codificação e pensará nos algoritmos e na lógica de programação. O outro parceiro observa o código produzido e tenta pensar mais estrategicamente em como melhorá-lo e torná-lo mais simples, além de apontar possíveis erros e pontos de falha. Além disso, as duplas são constantemente trocadas e os papéis também com o objetivo de que todos os membros da equipe possam ter conhecimento sobre todas as partes do sistema.

A programação em pares encoraja duas pessoas a trabalharem juntas procurando atingir o melhor resultado possível. A propriedade coletiva encoraja a equipe inteira a trabalhar mais unida em busca de qualidade no código fazendo melhorias e refatoramentos em qualquer parte do código a qualquer tempo. A princípio, pode-se pensar que esta prática vai gerar problemas, mas como todos devem respeitar um padrão de codificação e devem realizar todos os testes para verificação de erros esta atividade é feita de uma forma controlada e pode ser facilitada com o uso de uma ferramenta de controle de versão.



Práticas | Integração Contínua | Cliente no Local | Padrões

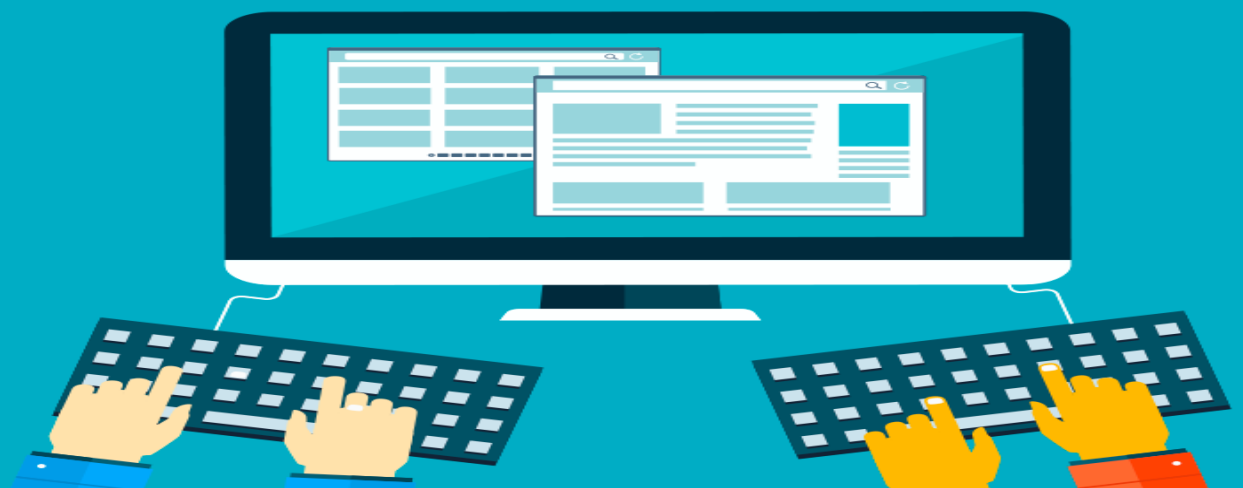
Integração Contínua (Continuous Integration): Sempre que produzir uma nova funcionalidade, nunca esperar uma semana para integrar à versão atual do sistema. Isto só aumenta a possibilidade de conflitos e a possibilidade de erros no código fonte. Integrar de forma contínua permite saber o status real da programação.

Cliente no local: Deve ser incluído na equipe uma pessoa da parte do cliente, que irá usar o sistema, para trabalhar junto com os outros e responder as perguntas e dúvidas.

Padronização do Código (Coding Standards): A equipe de desenvolvimento precisa estabelecer regras para programar e todos devem seguir estas regras.

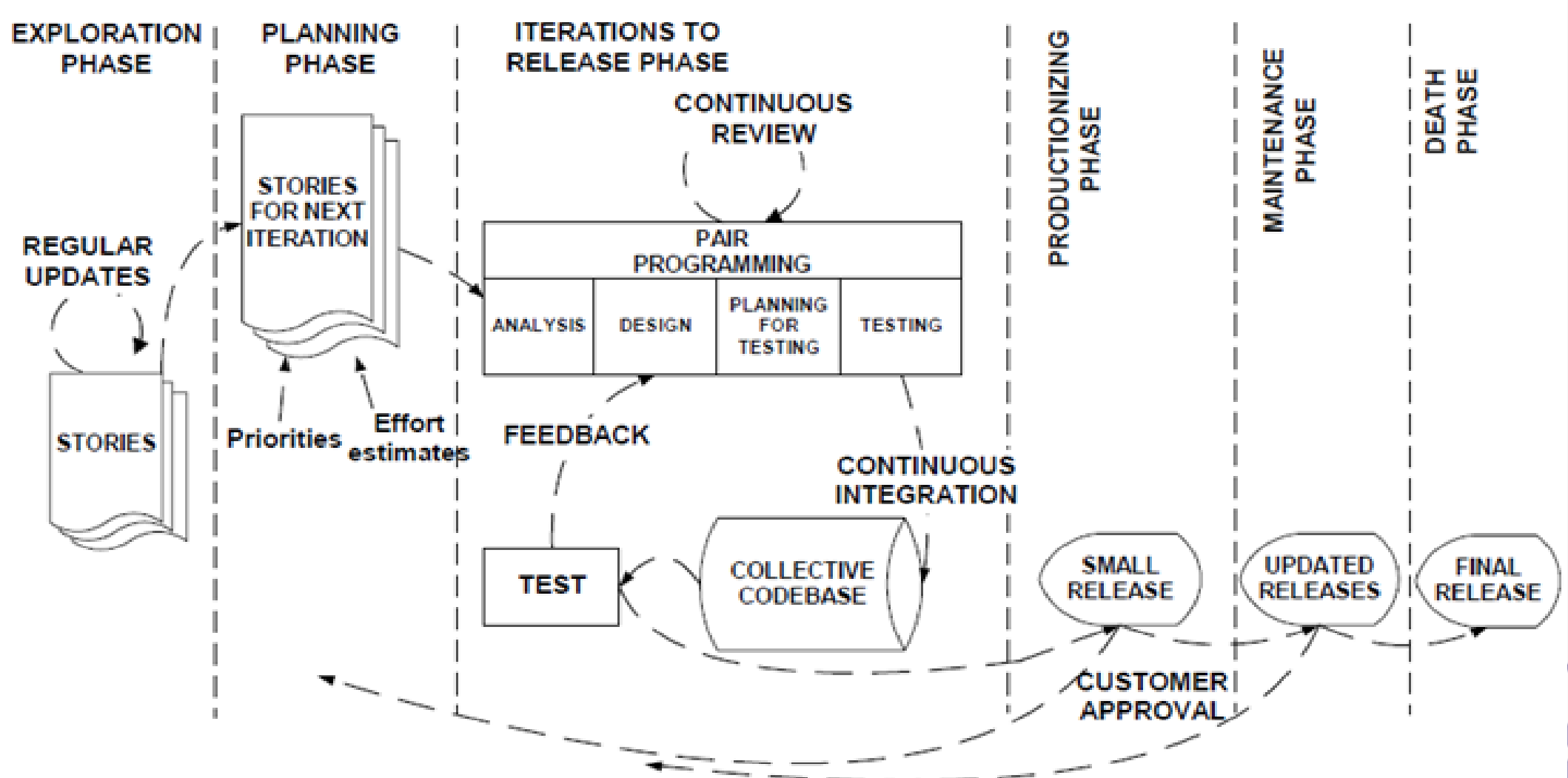
Desta forma parecerá que todo o código fonte foi editado pela mesma pessoa, mesmo quando a equipe possui 10 ou 100 membros.

O objetivo é que todos programem da mesma forma, facilitando o entendimento do código e as alterações.



Ciclo de Vida

Um projeto XP atravessa algumas fases durante o seu ciclo de vida. Essas fases são compostas de várias tarefas que são executadas.



Ciclo de Vida | Fase de Exploração

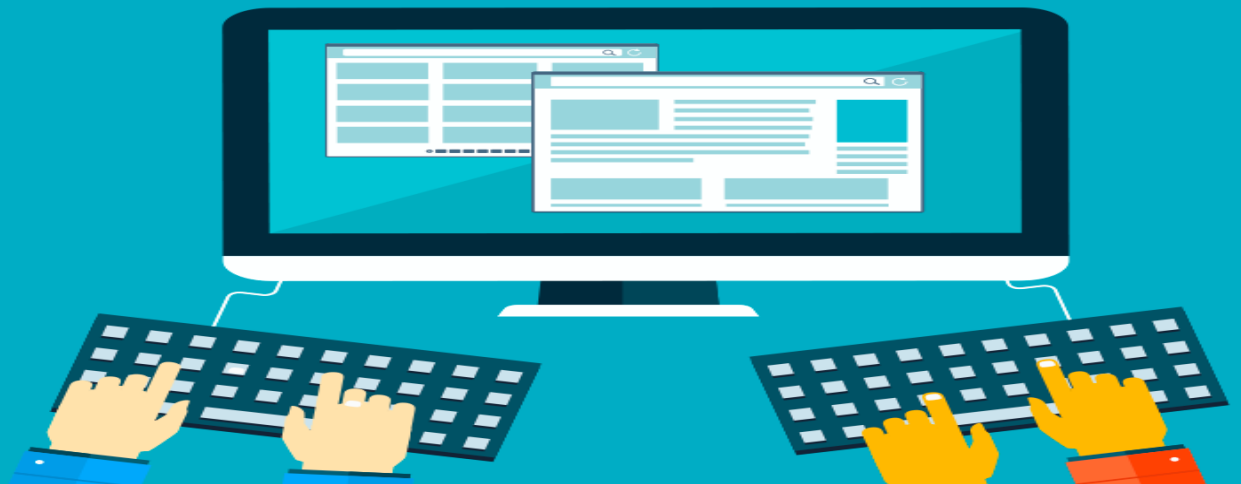
É anterior à construção do sistema.

Nela, investigações de possíveis soluções são feitas e verifica-se a viabilidade de tais soluções.

Os programadores elaboram possíveis arquiteturas e tentam visualizar como o sistema funcionará considerando o ambiente tecnológico (hardware, rede, software performance, tráfego) onde o sistema irá rodar.

Com isso, os programadores e os clientes vão ganhando confiança, e quando eles possuírem estórias suficientes, já poderão começar a construir o primeiro release do sistema.

Sugere-se que seja gasto no máximo duas semanas nessa fase.



Ciclo de Vida | Fase de Planejamento Inicial

Deve ser usada para que os clientes concordem em uma data para lançamento do primeiro release.

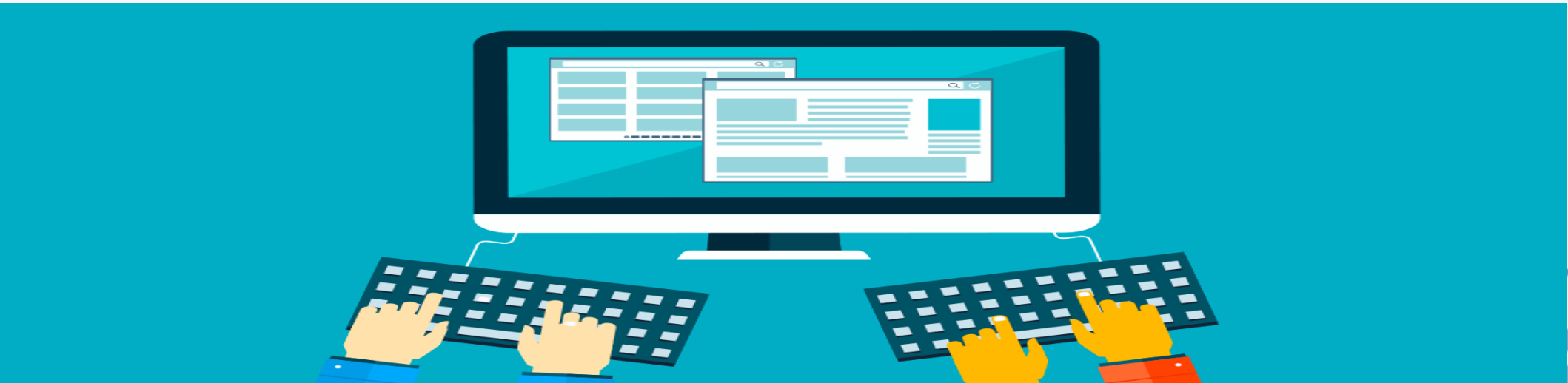
O planejamento funciona da seguinte forma: Os programadores, juntamente com o cliente, definem as estórias (casos de uso simplificados) a serem implementadas e as descrevem em cartões.

Os programadores assinalam uma certa dificuldade para cada estória e, baseados na sua velocidade de implementação, dizem quantas estórias podem implementar em uma iteração.

Depois, os clientes escolhem as estórias de maior valor para serem implementadas na iteração – isso é chamado planejamento de iteração.

O processo então se repete até terminar as iterações do release.

O tempo para cada iteração deve ser de uma a três semanas e para cada release de dois a quatro meses.



Ciclo de Vida | Fase das Iterações do Release

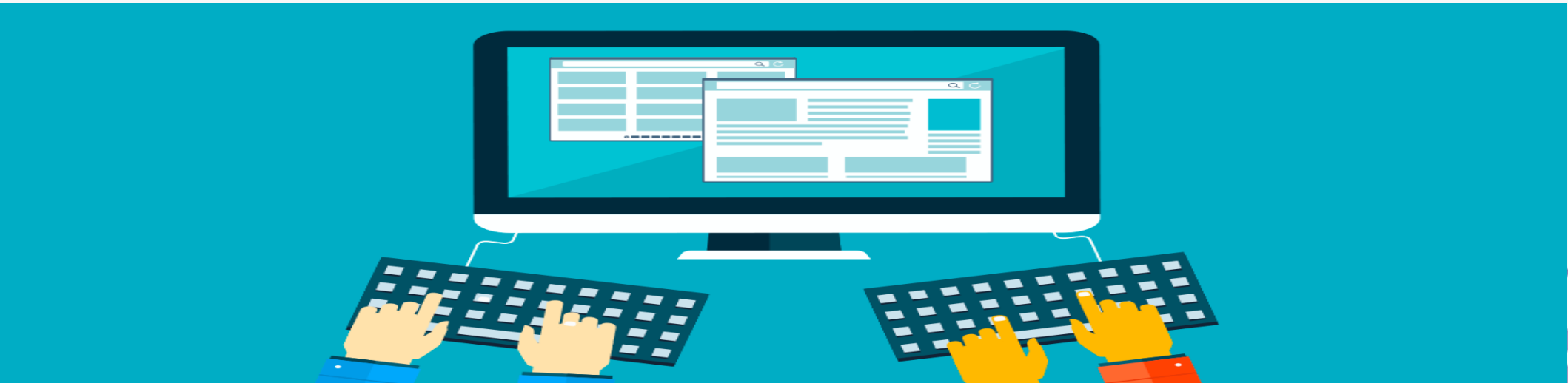
São escritos os casos de teste funcionais e de unidade.

Os programadores vão seguindo mais ou menos o seguinte fluxo de atividades na seguinte ordem (em cada iteração):

- Escrita dos casos de testes.
- Projeto e refatoração.
- Codificação.
- Realização dos testes.
- Integração.

À medida que esse fluxo vai sendo seguido, o sistema vai sendo construído segundo os princípios, valores e práticas apresentados anteriormente.

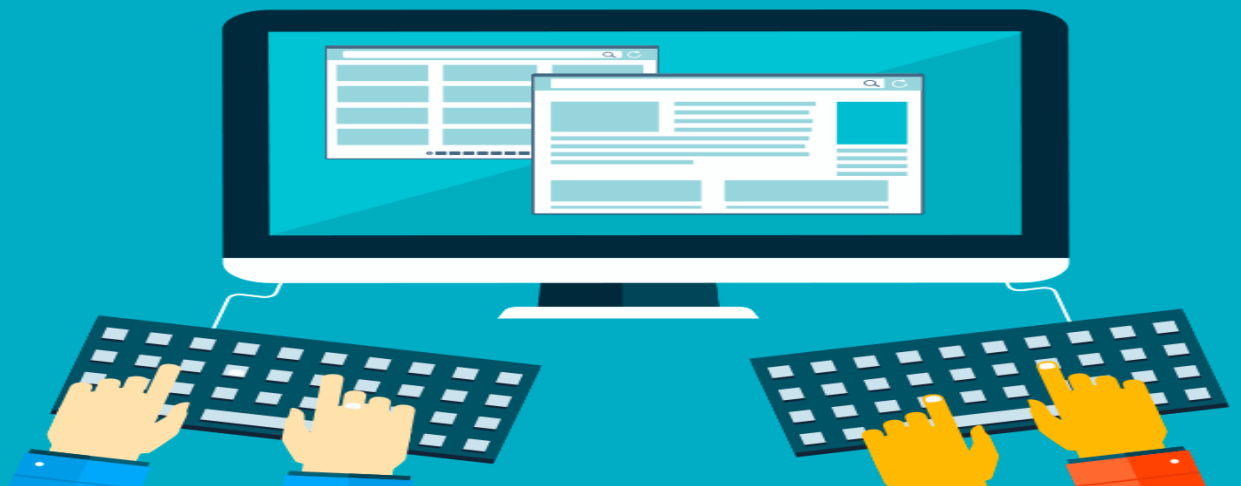
Depois de terminado o primeiro release, já se terá uma ideia melhor das tecnologias e do domínio do problema de modo que as iterações poderão ser mais curtas nos releases seguintes e já se podem fazer estimativas mais confiáveis com o que se aprendeu das iterações passadas.



Ciclo de Vida | Fase de Produção

Depois do final do primeiro release, considera-se o início da Fase de Produção onde cada release subsequente do sistema, depois de construído, é colocado para rodar em um ambiente que simula o ambiente de produção (ambiente de homologação) para ver seu comportamento em termos de performance.

Pode-se fazer testes de aceitação adicionais para simular o funcionamento real do sistema no ambiente alvo.



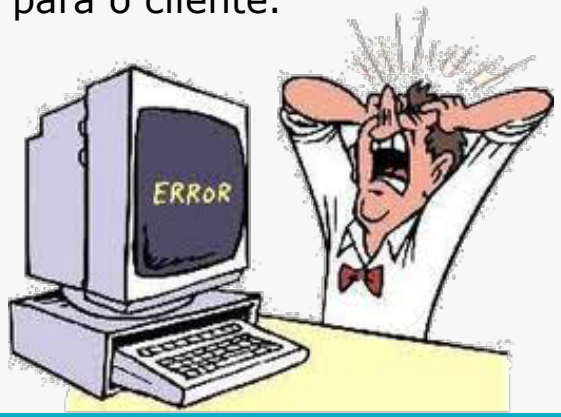
Ciclo de Vida | Fase de Manutenção

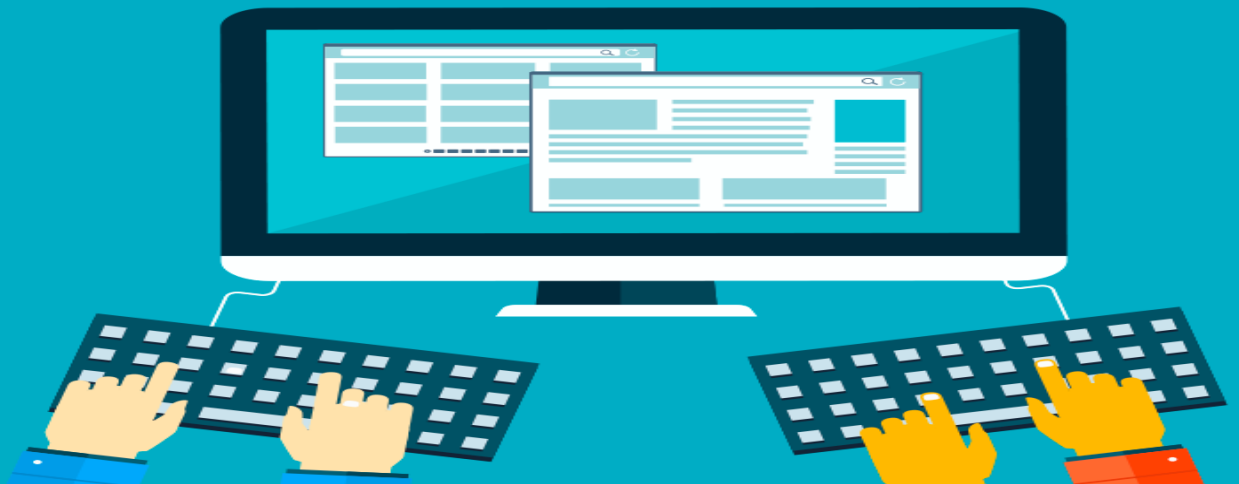
Pode ser considerada como uma característica inerente a um projeto da XP.

Na XP você está simultaneamente produzindo novas funcionalidades, mantendo o sistema existente rodando, incorporando novas pessoas na equipe e melhorando o código.

Mecanismos como: refatoração, introdução de novas tecnologias e introdução de novas ideias de arquitetura podem ser utilizados em um projeto XP.

É importante ressaltar que a manutenção dada em um sistema que já está em produção deve ser feita com muita cautela, pois uma alteração errada pode paralisar o funcionamento do sistema resultando em prejuízos para o cliente.





Ciclo de Vida | Fase da Morte

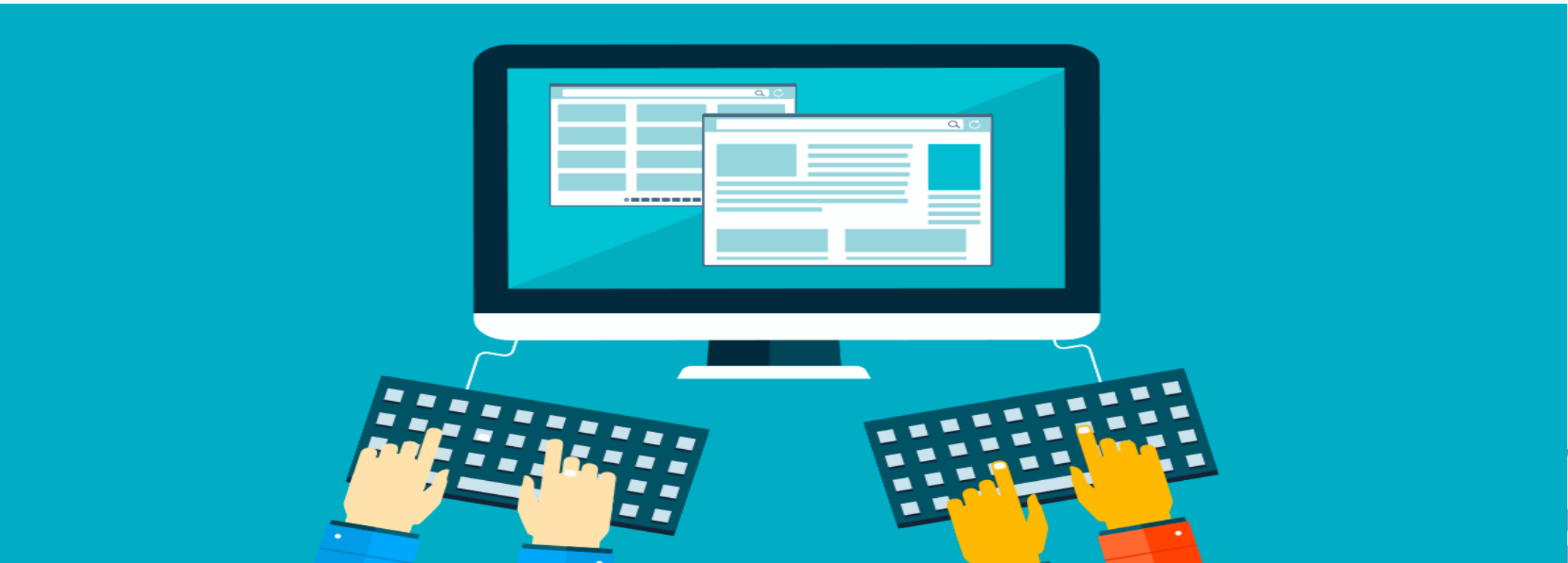
Corresponde ao término de um projeto XP.

Existem duas razões para se chegar ao final de um projeto, uma boa e a outra ruim.

A boa razão é quando o cliente já está satisfeito com o sistema existente e não enxerga nenhuma funcionalidade que possa vir a ser implementada no futuro.

A ruim seria o projeto ter se tornado economicamente inviável, devido a dificuldades de adicionar funcionalidades a um custo baixo e devido a uma alta taxa de erros.

De qualquer forma, é na Death Phase que tudo acaba. Às vezes essa fase não chega jamais. Existem pessoas que comparam um sistema a um filho que sempre estará em sua vida.



SCRUM + XP

É possível?

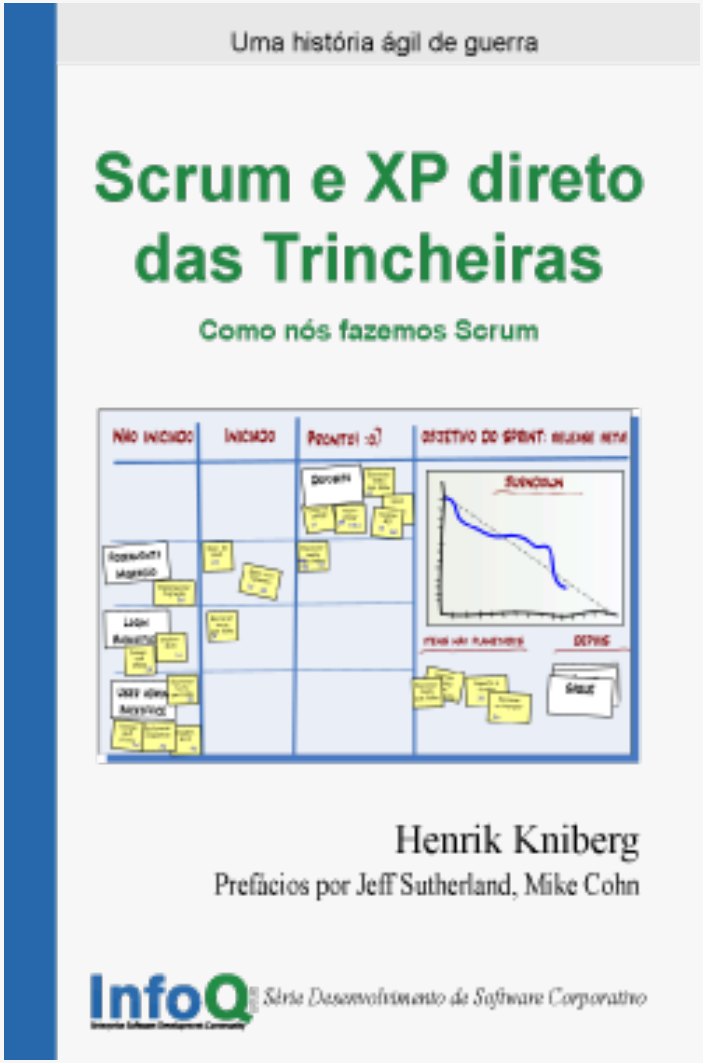
Sim. Totalmente.

E para aprender esse processo você vai acessar o livro online: Scrum e XP Direto das Trincheiras.

É fato que apenas conhecer os conceitos por trás do Scrum e XP e depois implementar no seu dia a dia pode se tornar uma tarefa difícil.

Este livro oferece um ponto de início, através de um conto detalhado sobre como uma empresa sueca implementou Scrum e XP com uma equipe de aproximadamente 40 pessoas e como eles continuamente melhoraram seu processo ao longo de 1 ano.

Clique na imagem ao lado e estude o livro com atenção.



Referências

PRESSMAN, Roger S. Engenharia de Software. São Paulo: Makron Books, 2011.
SOMMERVILLE, Ian. Engenharia de Software. 8. ed. São Paulo: Pearson Addison Wesley, 2007.
TONSIG, Sergio Luiz. Engenharia de Software: Análise e Projeto de Sistemas. 2. ed. Rio de Janeiro: Ciência Moderna, 2008.
Ken Schwaber e Jeff Sutherland. Scrum Guide. Disponível em http://www.scrum.org
BOEHM, B. Balancing Agility and Discipline: A Guide for the Perplexed. 2 ed. Boston,MA: Addison-Wesley, 2004
Manifesto for Agile Software Development. Disponível em http:www.agilemanifesto.org
MARTINS, J. C. C. (2007) Gerenciando Projetos de Desenvolvimento de Software com PMI, RUP e UML. 4 ed. Rio de Janeiro: Brasport.
KROLL, P. e Kruchten P. (2003) The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison Wesley.
BOENTE, A. N. P. (2003) Gerenciamento & Controle de Projetos. Rio de Janeiro: Axcel Books.
ROCHA, A. R. C.; Maldonado, J. C.; Weber, K. C. (2001) Qualidade de Software: Teoria e Prática. São Paulo: Prentice Hall.