UNIDADE 2 – MÉTODOS ÁGEIS E PROCESSO UNIFICADO

MÓDULO 1 – INTRODUÇÃO AOS MÉTODOS ÁGEIS DE DESENVOLVIMENTO DE *SOFTWARE*

01

1 – AGILIDADE

O que é ser ágil ou ter agilidade? Conforme Pressman, que cita uma discussão apresentada por Ivar Jacobson, atualmente, agilidade tornou-se a palavra da moda quando se descreve um moderno processo de *software*. E complementa:

"Uma equipe ágil é aquela rápida e capaz de responder apropriadamente a mudanças. Mudanças têm muito a ver com desenvolvimento de *software*. Mudanças no *software* que está sendo criado, mudanças nos membros da equipe, mudanças devido a novas tecnologias, mudanças de todos os tipos que poderão ter um impacto no produto que está em construção ou no projeto que cria o produto. Suporte para mudanças deve ser incorporado em tudo o que fazemos em *software*, algo que abraçamos porque é o coração e a alma do software. Uma equipe ágil reconhece que o *software* é desenvolvido por indivíduos trabalhando em equipes e que as habilidades dessas pessoas, suas capacidades em colaborar estão no cerne do sucesso do projeto.

Ainda segundo Jacobson, o **envolvimento com a mudança** é o ponto chave para a agilidade.

Conforme o Manifesto Ágil, agilidade é algo mais que uma atuação na mudança, pois existe uma filosofia proposta. Filosofia que incentiva a comunicação e envolvimento da equipe, prioriza a entrega em relação à documentação formal de um projeto comum. Outro fator é o envolvimento do cliente, que assume o papel fundamental no projeto. Importante frisar que nos projetos ágeis também existe documentação, mas não é fator determinante do projeto.

As metodologias ágeis nasceram como uma alternativa às metodologias até então existentes no mercado (tradicionais).

Os métodos ágeis visam à agilidade no desenvolvimento de *softwares*, onde é possível fazer, conforme necessidade, as modificações que venham a ocorrer no decorrer do projeto.

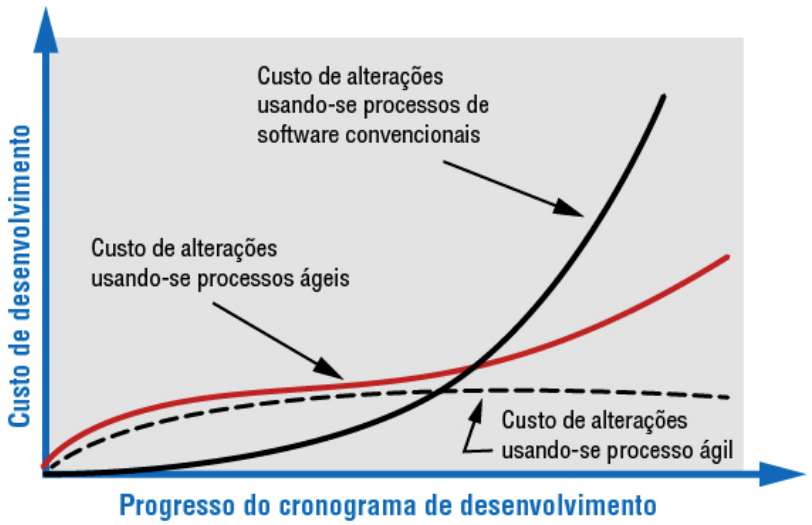
As metodologias ágeis mais usadas atualmente são a SCRUM e o XP (*Extreme Programming*).

02

Custo das mudanças em Métodos Ágeis

Pressman afirma que os custos de mudanças, em projetos que utilizam metodologias convencionais, aumentam de forma não linear conforme o projeto avança. Outro fator muito importante a destacar é que, quanto menos tempo de projetos melhor para ter algum tipo de mudança, os custos são mais baixos e os impactos também.

Na figura a seguir, os custos das mudanças em métodos ágeis é menor, devido às características do método. Mas o ideal é que tivéssemos o custo de alteração controlado, conforme a linha tracejada.



Custo de alterações com o tempo do projeto – Pressman

03

Imagine o seguinte exemplo. Você é engenheiro de uma construtora e o seu projeto está em fase de acabamento de uma casa térrea. A casa já tem as paredes, telhado, já foi até pintada, mas falta colocar os lustres e interruptores de luz. O seu cliente telefona e diz que, no futuro, ele quer construir mais um andar, portanto, a casa deve ter estrutura para isso. Para atender a essa nova demanda, provavelmente, você deverá quebrar o piso, que acabou de ser finalizado, e reforçar toda a base e a estrutura da casa, ou até mesmo deverá construir outra coluna (viga). Parece complicado? Sim, você vai desfazer várias coisas e terá de refazer tudo o que estragou e provavelmente gastará muito tempo e dinheiro para isso.

Agora imagine o mesmo exemplo, porém desta vez o projeto ainda está na fase de desenho da arquitetura. Provavelmente as mudanças não iam acarretar custos, talvez apenas o custo da adaptação do desenho, mas seria infinitamente menor que o caso anterior.

Enfim, os dois exemplos mostram os impactos da alteração do projeto em virtude do momento em que foi solicitado. Então concluímos que:

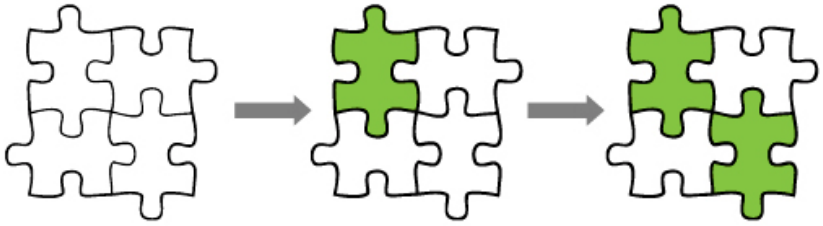
Quanto mais cedo forem feitas as alterações necessárias, menor o impacto nos custos, no cronograma e até mesmo no ânimo da equipe, que não precisará realizar o trabalho novamente.

Os métodos ágeis tendem a ter custos mais baixos nas mudanças, devido a sua particularidade de proximidade do cliente, que está o tempo todo testando as partes desenvolvidas. Mas mesmo com essa proximidade não deixa de ter seus problemas de mudanças de escopo por parte do cliente, mas o custo é bem menor que uma metodologia convencional.

04

2 - Princípios dos Métodos Ágeis

Os métodos ágeis utilizam o processo incremental, ou seja, vai fazendo aos poucos o *software*, pedaço a pedaço.



Método incremental

As metodologias ágeis utilizam os mesmos princípios, mas de forma diferente. Já abordamos o SCRUM e neste capítulo falaremos do **XP (Extreme Programming)**.

Na figura abaixo, Summerville apresenta alguns princípios básicos para os Métodos Ágeis.

Princípios	Descrição
Envolvimento do cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar suas iterações.
Entrega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos para serem incluídos em cada um.
Pessoas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos.
Aceitar as mudanças	Deve-se ter em mente que os requisitos do sistema vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças.
Manter a simplicidade	Focalize a simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para elimindar a complexidade do sistema.

O princípio dos Métodos Ágeis – Summerville, Engenharia de *Software* – 9ª Edição

Dos princípios citados, aprofundarem no item “Pessoas, não processos”, ou seja, os fatores humanos. Segundo Cockburn e Highsmith “O desenvolvimento ágil foca em talentos e habilidades de indivíduos, moldando o processo de acordo com as pessoas e as equipes específicas”. Segundo Pressman o ponto chave é que o processo se molda às necessidades das pessoas e da equipe e não o caminho inverso.

05

Fatores humanos são importantes para que o método ágil funcione. Pressman cita os seguintes fatores:

• Competência	No contexto do desenvolvimento ágil, a competência abrange talento inato, habilidades específicas relacionadas a software e conhecimento generalizado do processo que a equipe escolheu para aplicar. Habilidade e conhecimento de processo podem e devem ser ensinados para todas as pessoas que sejam membros de uma equipe ágil.
• Foco comum	Embora os membros de uma equipe ágil possam realizar diferentes tarefas e tragam diferentes habilidades para o projeto, todos devem estar focados em um único objetivo – entregar um incremento de software funcionando ao cliente, dentro do prazo prometido. Para alcançar essa meta, a equipe também irá focar em adaptações contínuas que farão com que o processo se ajuste às necessidades da equipe.
• Colaboração	Os membros da equipe devem colaborar entre si e com os demais envolvidos. Criar informações que ajudarão todos os envolvidos a compreender o trabalho da equipe e a construir informações para o cliente.
• Habilidade na tomada de decisão	Qualquer boa equipe de software deve ter liberdade para controlar seu próprio destino. Isso significa que seja dada autonomia à equipe – autoridade na tomada de decisão, tanto em assuntos técnicos como de projeto.
• Habilidade de solução de problemas confusos	Os gerentes de softwares devem reconhecer que a equipe ágil terá de lidar continuamente com a ambiguidade e que será continuamente atingida por mudanças. Lições aprendidas de qualquer atividade de solução de problemas podem ser futuramente benéficas para a equipe.
• Confiança mútua e respeito	A equipe ágil deve tornar-se uma equipe consistente, que demonstra confiança e o respeito necessários para torná-la fortemente unida.
• Auto-organização	No contexto do desenvolvimento ágil, a auto-organização implica três fatores: a equipe ágil se organiza para o trabalho a ser feito, a equipe organiza o processo para melhor se adequar ao seu ambiente local, a equipe organiza o cronograma de trabalho para melhor cumprir a entrega do incremento de software. A auto-organização serve para melhorar a colaboração e levantar o moral da equipe. Resumindo, a equipe faz seu próprio gerenciamento.

06

3 - XP - EXTREME PROGRAMMING

XP é um apelido carinhoso de uma nova metodologia de desenvolvimento designada **eXtreme Programming**.

Criada em 1997, o XP possui adeptos e outros que duvidam da sua real utilidade, muitos por falta de conhecimento ou entendimento, achando que no XP apenas o código é o que realmente interessa,

descartando o resto como planejamento, documentação etc. O XP é um método de desenvolvimento de *software*, leve, não é prescritivo, e procura fundamentar as suas práticas por um conjunto de valores. O XP teve sua primeira versão por Kent Beck, que definiu valores básicos para sua utilização, os quais serão vistos no próximo tópico.

XP é uma metodologia ágil para equipes pequenas e médias que desenvolvem *software* baseado em requisitos vagos e que se modificam rapidamente.

O primeiro projeto a usar XP foi o de folha de pagamento dos profissionais que trabalharam no carro C3, da Chrysler, após anos utilizando metodologias tradicionais, que fracassaram. Com o XP o projeto ficou pronto em pouco mais de um ano.

As principais **diferenças** da XP em relação às outras metodologias de mercado são:

- Feedback constante;
- Abordagem incremental;
- A comunicação entre as pessoas é encorajada.

XP foca no desenvolvimento rápidodo projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das datas planejadas.

Kent Beck
Kent Beck é americano, engenheiro de software criador do Extreme Programming (XP) e Test Driven Development (TDD). Beck foi um dos 17 signatários originais do Agile Manifesto em 2001. Manifesto que criou as Metodologias Ágeis.

07

Valores do XP

O XP possui um conjunto de valores, princípios e práticas fundamentadas, que orientam as atividades a serem realizadas durante o desenvolvimento do sistema. Os valores do XP são conceitos básicos que orientam o processo a ser seguido, define a filosofia de trabalho e como devem ser tomadas as decisões. São eles:

Princípios	Descrição
Comunicação	Segundo Pressman, para que o XP consiga a comunicação efetiva entre os envolvidos no projeto, deve valorizar a colaboração estreita, embora verbal, entre os clientes e os desenvolvedores. Assim o XP utiliza a comunicação para melhorar o relacionamento da equipe do projeto, preferindo conversas pessoais a outros meios de comunicação. Saiba+
Simplicidade	A metodologia foca nas necessidades imediatas, em vez das necessidades futuras. A simplicidade serve para criar códigos simples com requisitos atuais, deixando funcionalidades complexas mais para frente. Saiba+
Feedback	É gerado pelo próprio software quando implementado, pelos clientes e toda a equipe. O feedback também é realizado nos testes de cada funcionalidade implementada antes da entrega para o cliente. Saiba+
Coragem	XP exige coragem, ou melhor, disciplina. Pressman dá o exemplo: frequentemente há uma pressão significativa para a elaboração do projeto pensando em futuros requisitos. Grande parte das equipes de software fraqueja, argumentando que projetar o amanhã poupará tempo e esforço no longo do prazo. Equipe que utiliza o XP deve pensar no hoje, reconhecendo que as necessidades futuras podem mudar dramaticamente o projeto, gerando retrabalho em relação ao código implementado. É necessário ter coragem para implantar os três valores já descritos. Saiba+
Respeito	É importantíssimo o respeito entre os membros do projeto, outros envolvidos, ao processo do XP e indiretamente ao próprio software. Cada vez que a equipe realiza uma entrega por incrementos, ela está respeitando o processo XP.

Saiba + (Comunicação)
Outro fator importante é não gerar documentação volumosa como meio de comunicação e evitar a burocracia excessiva.

Saiba+ (Simplicidade)
A ideia é criar um projeto simples de fácil implementação com o que realmente será usado pelo cliente, até porque requisitos tendem a mudar sempre.

Saiba+ (Feedback)
Após cada necessidade que surge é realizado um novo Feedback, é levantado os impactos que o projeto pode sofrer no cronograma e nos custos e até mesmo na sua imagem. A prática do feedback constante significa que o programador terá informações constantes do código implementado e do cliente. O cliente com seu software com partes totalmente funcionando, retorna com novas ideias constantemente, sugerindo novas características e informações para os desenvolvedores. Importante destacar que eventuais erros e não conformidades são rapidamente identificados e corrigidos somente

nas próximas versões.

Saiba+ (Coragem)
Pois não são todas as pessoas que possuem facilidade de comunicação e têm capacidade de se relacionar em equipe. Cada pessoa tem uma maneira de ser e de pensar. Para ser simples tem que ter coragem também, o desenvolvimento de códigos simples para programadores avançados gera certo conflito. O Feedback é terrível para algumas pessoas, é necessário coragem para lidar com essa situação, pois a pessoa terá feedback da equipe e do cliente também.

08

4 - PRINCIPAIS DIFERENÇAS ENTRE METODOLOGIAS

As principais diferenças entre as metodologias, segundo Martin Fowler, são:

Metodologias ágeis	Metodologias Tradicionais
As metodologias ágeis são adaptativas, mais flexíveis, pois ajustam seus requisitos durante o desenvolvimento do *software*.	As metodologias tradicionais são previsíveis, resistentes às mudanças devido à excedente documentação feita com antecedência.
As metodologias ágeis são orientadas às pessoas, não aos processos, não possuem processos rígidos que impõem o que as equipes devem ou não fazer, as equipes são estimuladas a se envolver diretamente com o cliente e a tomar as decisões necessárias para melhor ajustar cada equipe.	As metodologias tradicionais são orientadas aos processos.

Além das diferenças listadas, é importante salientar que o XP **pode não servir para qualquer tipo de projeto**. Depende de como a empresa estrutura suas equipes e seus projetos, por exemplo, recomenda-se uma equipe de 2 a 10 integrantes, e que devem estar em todas as fases do projeto. Vários testes são realizados, o projeto pode ser alterado constantemente.

A **equipe deve ser muito interessada e pró-ativa**, para assegurar a alta produtividade, e o cliente deve estar totalmente disponível para tirar dúvidas e tomar decisões em relação ao projeto, caso contrário, pode atrasar o projeto e irá ferir um princípio do XP. A comunicação com o cliente é muito intensa, as dúvidas dos requisitos podem aparecer a qualquer momento.

Outro fator importante do XP é a **agilidade do planejamento**, pois não se define uma especificação completa e formal de requisitos. Os ciclos de desenvolvimento são curtos e frequentes. Veremos mais detalhes em “Boas práticas do XP”.

09

Boas práticas do XP

As boas práticas são atividades, procedimentos identificados como as melhores práticas para o sucesso na realização de alguma coisa ou para concluir um objetivo específico. No caso do XP as boas práticas são um conjunto de atividades que equipes ja utilizaram no XP e deram sucesso na produção dos *softwares*.

As boas práticas estão apoiadas nos princípios do XP e é importante respeitar esses valores para ter sucesso. Então listamos aqui resumidamente os pontos chaves para se ter sucesso em projeto utilizando XP.

O **cliente deve estar presente** e é essencial que ele participe ativamente do projeto. Cliente presente tira todas as dúvidas no momento que a equipe precisar, pois ele conhece o negócio.

Outro fator importante desse acompanhamento constante, é que o **cliente poderá fazer testes** à medida que as implementações forem realizadas e tomará as decisões necessárias sobre as prioridades do projeto. As implementações que o cliente irá testar são, na verdade, as*releases*, funcionalidades implantadas, as quais formam uma versão do sistema. Essa versão pode ser testada com as funcionalidades que foram implementadas. É importante no XP trabalhar com implantações de *releases* e fragmentar essas *releases* por iterações, que são subdivisões dentro de uma *release*.

Outra prática indicada é o que é chamado de **"Jogo de Planejamento"**, que nada mais é que um planejamento do que se tem de fazer no início de cada iteração. Cada iteração deve ser planejada com o cliente. O cliente deve escrever as *"user story"*, que são fichas com os requisitos do projeto, alguns dizem que é a menor quantidade de informação, na verdade são as funconalidades do sistema.

Iterações
Essas iterações podem conter atividades de documentação, análise, *feedback*, testes etc. Então temos um conjunto de iterações que formam uma *release*. O tempo de execução dessas etapas varia de projeto para projeto. Em algumas empresas o tempo de duração das iterações dura 2 semanas e a release não dura mais que 8 semanas. Essas *releases* são testadas pelo cliente para se verificar as necessidades de alteração e correção. A quantidade das *release* para concluir o projeto varia também, mas lembrem, XP é para ser rápido, ágil.

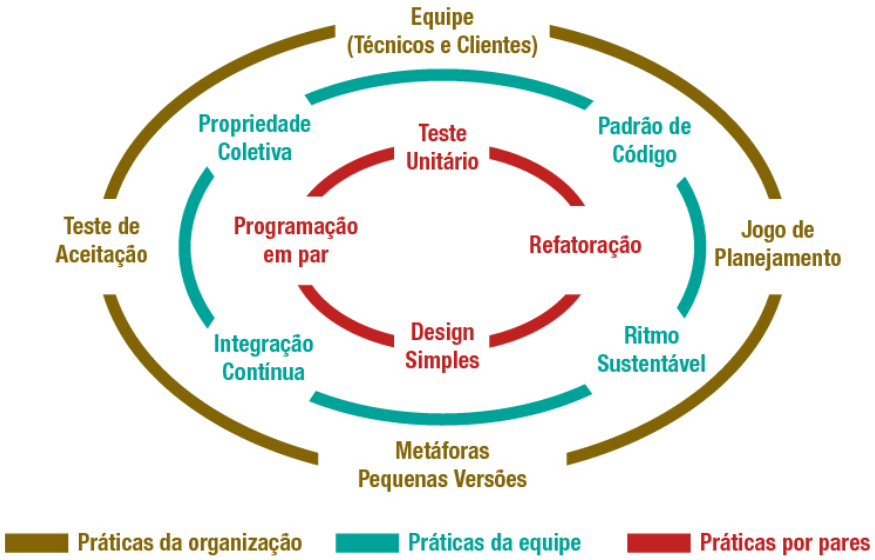
Fichas
Essas fichas possibilitam a equipe estimar quanto tempo o projeto vai levar e principalmente quanto tempo cada funcionalidade irá levar. Outra informação que podemos tirar dessas fichas é a complexidade e o custo de cada funcionalidade, assim o cliente poderá decidir o que implantar e o momento de construir as funcionalidades do sistema. Essa prática de planejamento no início das iterações é chamada de Jogo de Planejamento.

10

Outra prática do XP é a **metáfora**, segundo Teles, as metáforas têm o poder de transmitir ideias complexas de forma simples e clara. Por isso, o XP as utilizam para criar uma visão comum do projeto entre cliente e desenvolvedores. As metáforas devem estar de acordo com o vocabulário do cliente, assim a compreensão fica mais fácil e a comunicação é realizada com sucesso.

Programação por pares ou programação em par é outra prática, segundo Teles, os programadores sentam em um único computador e codificam as *user story*. O desenvolvedor de menor experiência é o responsável pela programação enquanto o desenvolvedor de maior experiência inspeciona o código a procura de defeitos, além de preservar a solução mais simples para o *software*. Uma das vantagens da programação em par é que o código está em constante revisão e o programador menos experiente ganha experiência com o desenvolvedor mais experiente nivelando o conhecimento técnico da equipe.

Mais uma boa prática do XP é a **refatoração**, que é a melhoria do código para melhorar a qualidade do *software*. Essa prática diminui o tempo da manutenção do programa e até mesmo deixa mais claro os códigos implementados. Segundo Teles, o risco para a refatoração é mexer em códigos que ja estão prontos, o que poderia causar uma parada na execução. Assim sendo, os testes devem ser realizados sempre após essa melhoria dos códigos. Para o XP, qualquer um da equipe pode realizar a refatoração.



11

Resumo

No momento mundial atual da TIC, empresas buscam a agilidade na construção de seus projetos e nas suas manutenções. Geralmente abocanha parte do mercado quem chega primeiro. As metodologias ágeis simplificaram o desenvolvimento do *software*, mas é necessário que se mantenha fiel aos seus valores e princípios. Os métodos SCRUM e XP tiveram suas origens devido a projetos fracassados, projetos que utilizavam outras metodologias de desenvolvimento. Suas origens estão baseadas em 5 princípios: Comunicação, Coragem, Simplicidade, Feedback e Respeito.

Algumas exigências fazem a diferença entre as metodologias tradicionais e as com métodos ágeis, a flexibilidade e o cliente mais participativo são pontos chaves e essenciais. O XP, apelido para eXtreme Programming tem como boas práticas a metáfora e refatoração. Melhoram a interpretação das ideias e a qualidade do código respectivamente.

Hoje em dia pode-se utilizar qualquer metodologia para desenvolver um *software*, o que define isso é se sua empresa está preparada para essa diversidade de metodologias. Lembrando que métodos ágeis devem estar baseados em cima de seus valores e princípios e de acordo com suas origens. Para qualquer metodologia é importante buscar as boas práticas de mercado e da própria empresa, o histórico diminui os caminhos do erro.

UNIDADE 2 – MÉTODOS ÁGEIS E PROCESSO UNIFICADO

MÓDULO 1 – APLICAÇÃO DE MÉTODOS ÁGEIS NO DESENVOLVIMENTO DE *SOFTWARE*

01

1 - APLICAÇÃO DO MÉTODO ÁGIL

Neste módulo nos aprofundaremos no estudo da metodologia SCRUM. Para alguns, SCRUM não é uma metodologia em si, mas um *framework*, com objetivo de ser eficaz, por isso pode empregar diversos processos e técnicas. A ideia de referir-se ao SCRUM como um framework é porque o SCRUM não vai indicar exatamente como agir, mas apresenta um conjunto de conceitos que podem ser usados para resolver o seu problema. Vale ressaltar que as melhores práticas de utilização ajudam no momento do desenvolvimento do projeto.

Os princípios do SCRUM também estão aderentes com o manifesto ágil e são usados para orientar o desenvolvimento do projeto, que possui algumas atividades importantes:

- requisitos,
- análise,
- projeto,
- evolução e
- entrega.

Pressman destaca que em cada atividade metodológica ocorrem tarefas que são realizadas dentro de um padrão de processo chamado de SPRINT. A quantidade necessária de sprints para cada atividade metodológica varia dependendo do tamanho e da complexidade do produto. O trabalho pode ser modificado ou adaptado dependendo do problema e até mesmo modificado em tempo real de execução. SCRUM utiliza boas práticas de desenvolvimento e padrões que provaram ser eficazes para aqueles projetos com prazos apertados, com requisitos muitas vezes alterados e para projetos críticos para o negócio.

Framework
Para efeitos deste curso, trataremos Framework como um conjunto de conceitos usado para resolver

um problema de um domínio específico. Sendo um modelo que pode ser utilizado, que contém atividades pertinentes a pessoas de várias funções para o atingimento de um resultado ou projeto específico.

02

O SCRUM abraça as mudanças, ao contrário de algumas metodologias tradicionais de desenvolvimento de *software*. Isso exige muita dedicação, disciplina e habilidade da equipe. No SCRUM as equipes têm seus papéis associados aos integrantes, bem como eventos com duração fixa e também seus artefatos.

Os times do SCRUM devem ter como objetivo a flexibilidade em prol da produtividade, sendo que para isso é necessário que sejam **auto-organizadas**, **com disciplina** e sempre **trabalhando com iterações**.

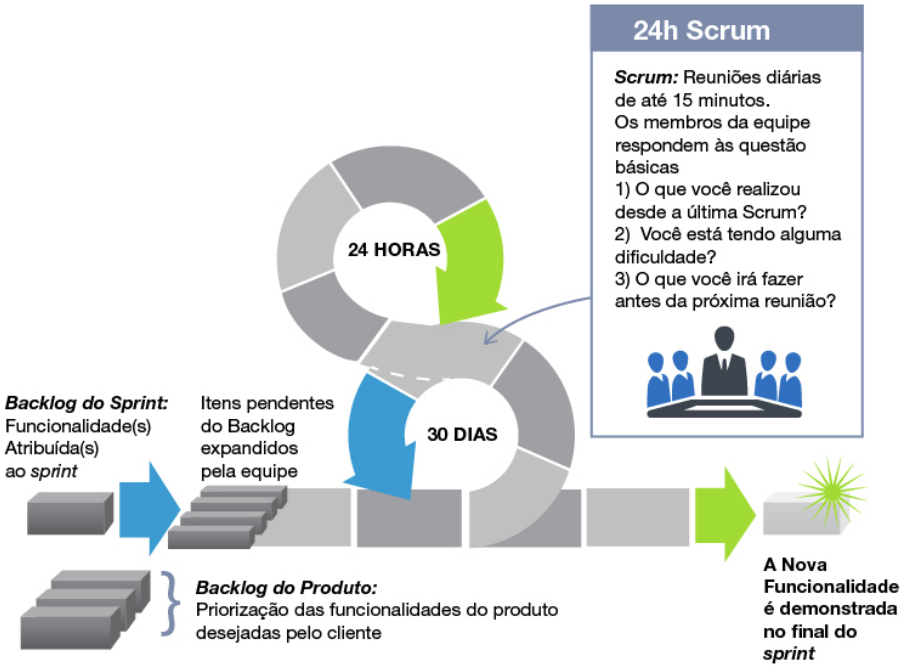
Como já visto em módulos passados, o SCRUM possui um time com pelo menos três papéis; vale aqui relembrar: ScrumMaster, Product Owner e o time ou equipe.

Product Owner	"Dono" do produto, ele tem a visão do negócio e o que de lucro ou benefício trará para a empresa dele ou para ele próprio, logo, sua missão é cuidar na lista de requisitos ou o Backlog do produto; outra atividade é repassar para a equipe uma visão clara e objetiva do produto. Quando o cliente não pode estar presente, é interessante colocar, desde o início do projeto, alguém com ele para ter todo o conhecimento do negócio. Essa pessoa fará as vezes do cliente quando este não estiver, pode ser um diferencial para o projeto.
ScrumMaster	Outro papel importantíssimo, pois é a pessoa que vai garantir que o processo está sendo seguido e vai tirar todos os impedimentos. Impedimentos são quaisquer coisas que possam atrapalhar o bom andamento do projeto. O ScrumMaster deve garantir que a equipe está seguindo os princípios do SCRUM.

Princípios
Muitas empresas dizem que utilizam SCRUM, mas não seguem os princípios básicos, então não é considerado SCRUM.

03

O gerenciamento do SCRUM é dividido entre os **membros da equipe**, isso deixa o projeto forte para buscar a agilidade. O **autogerenciamento** é um dos fatores de sucesso do SCRUM, ao contrário das metodologias tradicionais, que seguem um modelo definido para o desenvolvimento do projeto, onde contempla atividades, tarefas, marcos, documentação etc. Mas lembre-se de que autogerenciamento só serve para uma equipe madura.



04

2 - MELHOR FORMA DE UTILIZAÇÃO DO SCRUM

Conforme a figura “Fluxo de processo do SCRUM” temos um modelo padrão de utilização do SCRUM, mas que pode ser adaptado sem perder de vista os pilares do SCRUM. Na fase inicial o cliente escreve as User Story (história de usuário), que são descrições simples que representam uma funcionalidade. A User Story deve ser escrita do ponto de vista das necessidades dos clientes, isso é o recomendável. Os User Story são escritos em cartões, e essas histórias são organizadas por ordem de prioridade, juntamente com o cliente. Essa priorização seleciona as funcionalidades mais importantes chegando a descartar o excesso de funcionalidades inúteis.

Como as User Story serão escritas pelos clientes, provavelmente teremos problemas na interpretação daquilo que o cliente escreveu, o que é muito comum acontecer. Para evitar problemas dessa natureza é importante a proximidade com o cliente continuamente, ou seja, a escrita das User Story deve ser feita entre a equipe e o gestor. Quando falamos em gestor estamos falando do cliente, nesse caso são as mesmas pessoas. Segundo James Martin 82% do custo do projeto está envolvido em retrabalho, devido a má definição dos requisitos iniciais e 56% dos projetos entregues são considerados com alguma falha, e essas falhas acontecem nas fases iniciais do projeto.

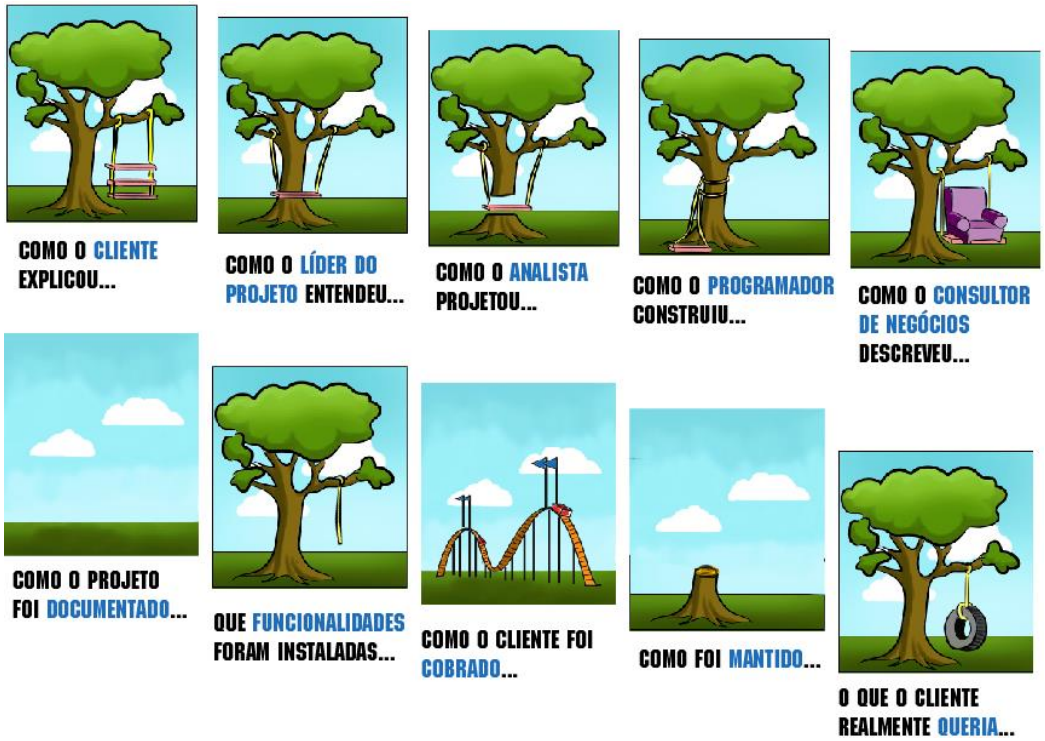
Surge então uma preocupação enorme com a fase inicial do projeto, mas necessariamente com a fase de levantamento de requisitos, quanto mais detalhado que for os requisitos, melhor será a compreensão de todos nas outras etapas do projeto. Deve usar práticas de levantamento de requisitos para manter as User Story bem simples e compreensíveis com facilidade de entendimento da real necessidade do cliente. No SCRUM qualquer um pode escrever a User Story, desde que tenha domínio do negócio, mas cabe ao Product Owner aceitar e bater o martelo das prioridades das User Story.

User Story são consideradas ferramentas de trabalho que definem todas as funcionalidades do sistema, consequentemente, têm todos os requisitos do *software*, onde devemos realizar o gerenciamento dessas User Story.

Funcionalidades inúteis
O termo "funcionalidades inúteis" pode parecer estranho, mas grosso modo é assim que funciona: o cliente quer o mundo, mas precisa apenas de um lote desse mundo. A priorização faz o cliente ter uma noção de tempo, que acaba planejando primeiro as Sprints com as histórias priorizadas.

05

Importante ressaltar que no SCRUM a linguagem comum de comunicação para construir um entendimento é a do cliente e não a da equipe de TI. É comum o uso de termos técnicos em projetos de TI, o gestor não é obrigado a saber desses termos, por isso a linguagem deve ser a do cliente, assim todos entendem. A User Story não é uma especificação de requisitos (veremos especificação de requisitos nos próximos módulos), mas sim as intenções de funcionalidades que o cliente necessita para o seu sistema. São curtas e de leitura fácil compreensíveis para todos os interessados. As avaliações são diárias do andamento do projeto, a cada dia de sprint a equipe faz uma breve reunião, chamada de Daily Scrum Meeting. O objetivo dessa reunião é disseminar conhecimento sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que começa. Resumindo, essas reuniões diárias servem para verificar se as pendências da reunião do dia anterior foram sanadas, as causas dos atrasos e o mais importante tomar algum tipo de decisão caso necessite.



06

Uma das melhores formas de utilizar o SCRUM é o planejamento dos sprints em versões ou releases. As entregas de versões existem um conjunto de sprints que permite o cliente testar o produto, ou seja, conjunto de funcionalidades que incrementam ao código que está sendo construído. Cada sprint é um marco de entrega e no final da release tem uma entrega de funcionalidades em produção.

Formas de utilização do SCRUM

Exemplo de User Story

Um exemplo de como **não** escrever uma User Story:

MySQL será usado para a persistência dos dados.

A forma correta para esse exemplo deveria ser:

Os dados serão persistidos em um banco de dados de qualidade e grátis.

Ainda que o MySQL seja da preferência do cliente, é importante evitar colocar nomes de ferramentas, pois elas mudam com o passar do tempo.

07

3 - Planejamento do Sprint

Depois da construção das User Story há uma reunião de planejamento de sprint. Nessa reunião faz o planejamento das releases, ou seja, quais sprints irão entrar em cada versão. Essa reunião tem a participação de todos da equipe. Todos devem concordar com as condições e responsabilidades que irão assumir, além de ser mais uma oportunidade de conhecer o produto que irá se construir. Os User Story são os itens de Backlog.

Depois de definido e realizado o planejamento, fica agora o momento de distribuir as tarefas pelos integrantes da equipe. As equipes que utilizam Método Ágil têm a prática de uso de uma ferramenta muito boa para controle: o quadro de tarefas. O **quadro de tarefas** é um quadro, ou uma lousa, onde o ScrumMaster coloca as tarefas que devem ser realizadas por cada integrante da equipe, são as tarefas do*backlog* (funcionalidades do produto a serem entregues). Essas tarefas são coladas no quadro por post its de forma organizada, conforme a situação que se encontra. O quadro de tarefas tem a finalidade

de identificar rapidamente o estágio que se encontra o projeto. O quadro normalmente é dividido com as seguintes colunas: estórias (itens do *backlog*), por fazer, fazendo, feito.

Ferramenta utilizada no SCRUM - Quadro de tarefas

08

O importante dessa fase é que as tarefas levantadas irão guiar todo o desenvolvimento da sprint. Todos enxergam tudo que está acontecendo. Acaba virando uma ótima ferramenta de comunicação e colaboração, pois todos acompanham o desenvolvimento de todos. Caso algum desenvolvedor tenha problema em sua tarefa, a mesma pode ser discutida ou redistribuída rapidamente, sem afetar os prazos do projeto.

As tarefas são User Story (histórias de usuários) quebradas em granularidade menor. São tarefas que possam ser feitas em um dia de trabalho, caso contrário deve aumentar a granularidade da tarefa.

Alguns autores oferecem dicas e técnicas para divisão de histórias, tais como:

- Em uma história que envolve múltiplos atores, dividi-la por ator;
- Dividir histórias de forma a maximizar o número de desenvolvedores que podem trabalhar em cada história;
- Dividir uma história de forma que as partes de alto risco fiquem separadas das partes de baixo risco;
- Dividir histórias de forma a facilitar os testes.

Independente do método utilizado para dividir as histórias, o fato é que esse trabalho de quebrar a User Story em tarefas é muito importante, por isso deve-se realizar um trabalho de análise detalhada da User Story a ser desenvolvida.

Essa granularidade facilita a vida de quem vai desenvolver, pois as tarefas são aparentemente como um guia e, ao finalizar todas as tarefas, significa que não tem mais nada para aquela sprint. Finalizando uma sprint, o desenvolvedor ataca a próxima.

09

4 - DOCUMENTAÇÃO – MÉTODO ÁGIL

Muitos comentam que Método Ágil peca pela falta de documentação, isso é devido à quantidade de documentos gerados nas metodologias tradicionais. A maioria das empresas do governo utiliza metodologias tradicionais. Suas áreas de auditoria cobram muito as documentações dos sistemas construídos. Isso gera um medo na utilização das metodologias ágeis e acaba inibindo o desenvolvimento desse tipo de processo.

Na verdade há um mito de que não se documentam em metodologias de desenvolvimento ágil.

A grande diferença das metodologias tradicionais e das ágeis é que, na tradicional às vezes o técnico documenta porque "tem de" documentar, sem saber ao certo o objetivo daquele documento, e na maioria das vezes nem utiliza o documento que está construindo, ou seja, é um documento que nasce morto, pois apenas cumpre uma regra metodológica. Já no método ágil não se deve perder tempo com esse tipo de documentação, que é aquela que não servirá para nada.

Um ponto importante é que **documentação não substitui a comunicação**. Muitas empresas e equipes se escondem atrás de documentos e algumas ferramentas para realizarem a comunicação. Isso é um erro gravíssimo. Fica aquela história: o analista construiu a especificação de requisitos e mandou por email para o programador, que senta exatamente ao seu lado. O programador não teve tempo de abrir sua caixa postal ainda, e isso já tem três dias, então não leu o email. O analista fica culpando o programador e começa a falar para o cliente que o código ainda não está pronto, porque o programador está ocupado com outras coisas. Bem, aí começam todos os problemas e o relacionamento da equipe piora, tudo por falta de uma conversa, uma comunicação oral.

A formalização pode ocorrer sem problemas, mas a conversa tem que existir. Muitos projetos têm seus problemas resolvidos no cafezinho, na copa e até mesmo no almoço, são nesses momentos que os técnicos conversam uns com os outros. Então se um dia você for um gestor ou dono de empresa não corte as conversas no cafezinho e nunca corte o café.

10

O Scrum prega que a documentação:

- tem que ser viva, não pode ser perecível;
- tem que ser leve e durável,
- deve ser atualizada de acordo com a evolução do software.

Há quem diga que é melhor não ter documentação do que ter uma documentação errada, desatualizada, que não serve para nada, do tipo que o sistema já alterou tanto que não bate o documento com o código construído. Documentação desatualizada é mesma coisa que um mapa antigo ou GPS desatualizado, pois pode guiar você para um caminho que provavelmente não existe mais, isso leva ao descarte da documentação, do mapa e até mesmo do GPS.

A documentação **não pode ter um nível de detalhe que a faça ser alterada todos os dias**, a não ser que realmente precise. Então quando você for definir quais documentos utilizará no seu projeto, pense se realmente você precisa dela e se ela será atualizada.

Procure a melhor maneira de deixar seu projeto SCRUM documentado. Uma dica é que **a documentação tem que ser rápida**, não pode dar trabalho demais. Quanto mais fácil documentar, as chances de ser atualizada são maiores. Outra dica é **não deixar a documentação para depois**. Aquela história de "depois que entregarmos o sistema, documentamos" é inaceitável, pois a documentação faz parte do projeto construído. O cliente pode cobrar por ela pronta e atualizada.

Vale destacar que temos dois tipos de documentação, a **do projeto** e a **do sistema**. A documentação do projeto tem tudo o que foi necessário para construir o sistema: cronogramas, planos, requisitos, testes, etc. Já a documentação dos sistemas prontos são os manuais de utilização, guias, garantias, etc.

Isso varia com o produto, por exemplo, um aparelho de DVD vem com seu manual de utilização, papel da garantia, locais de suporte. Essas documentações são construídas na fase de projeto. Elas fazem parte da entrega do produto. Entregas não são apenas os códigos com as funcionalidades, o exemplo demonstra isso, tudo depende das necessidades do produto e do cliente.

11

5 – COMUNICAÇÃO

No Método Ágil a comunicação é contínua, direta. No mesmo exemplo anterior, o analista poderia ter comunicado com o desenvolvedor da especificação pronta, assim, no boca a boca, sem excesso de burocracia formal de comunicação. Outro exemplo bom de comunicação direta é no Exército. O comandante dá os comandos à tropa em exercício no campo, sem a necessidade de uma ferramenta como o Outlook. Imagine você no campo de batalha esperando a ordem por e-mail.

Como funciona essa comunicação então? Por que os comandados obedecem às ordens sem uma comunicação escrita formal? Pois é, existe vários regimentos internos que pregam que a palavra vale como ordem. A mentira é uma infração gravíssima. O militar não pode falar e dizer que não falou, ou simplesmente dizer que "não está escrito não vale". Claro que o Exército tem sua comunicação formal e suas formalidades, mas no dia a dia de trabalho o que vale é o que se diz.

Muitas pessoas de empresas grandes se escondem atrás de e-mails e formulários, é o excesso de burocracia. Um dia desses mandaram uma fatura para validar e assinar, só que a fatura não tinha nada para pagar, pois estava com valor zerado. Era um serviço que tinha sido cancelado, mas mesmo assim pediram para validar, senão o pagamento não aconteceria. Como paga algo que não existe? Como avalia um serviço que não foi executado? A desculpa é que o sistema gerou, tem que assinar. Os técnicos não podem esconder atrás do Outlook e de rotinas erradas de sistemas, ligue e resolva os problemas do projeto. Outro ponto a observar é que e-mails são frios, a pessoa que recebe uma mensagem pode entender completamente diferente que você queria dizer.

12

RESUMO

Na economia atual, as condições do mercado mudam muito rapidamente, tanto o cliente como o desenvolvimento devem evoluir para os novos desafios da competitividade, que surgem sem avisar. A agilidade no desenvolvimento e entrega do sistema pode determinar muitos milhares de reais.

Os Métodos Ágeis podem ser utilizados para uma vasta quantidade de projeto, desde que sigam sua filosofia e seus princípios de desenvolvimento. O SCRUM teve sua criação em um projeto de um carro e serviu como uma luva para os projetos da TI. O importante no SCRUM são os incrementos no final de cada versão implantada, sendo que a documentação é construída também de acordo com a necessidade, sem excessos. A integração da equipe no projeto com o cliente é fundamental para o sucesso do projeto. A comunicação informal é valorizada e a burocracia deve ser evitada. O SCRUM tem seus princípios consistentes com o manifesto ágil e são usados para orientar as atividades de desenvolvimento. Os trabalhos são realizados dentro do número necessário para cada atividade metodológica e varia com o tamanho e complexidade do produto. A utilização de ferramentas pode ajudar os projetos que utilizam Métodos Ágeis a ganhar agilidade, organização e também a comunicação. Tudo isso só funcionará se seu time trabalhar com uma equipe.

UNIDADE 2 – MÉTODOS ÁGEIS E PROCESSO UNIFICADO

MÓDULO 3 – INTRODUÇÃO AO PROCESSO UNIFICADO PARA O DESENVOLVIMENTO DE SISTEMAS

01

1 - PROCESSO UNIFICADO

No módulo anterior aprendemos muito sobre o método ágil SCRUM. Falamos das diferenças entre os métodos ágeis e as metodologias tradicionais. Estudaremos nesse módulo uma metodologia considerada tradicional, o RUP.

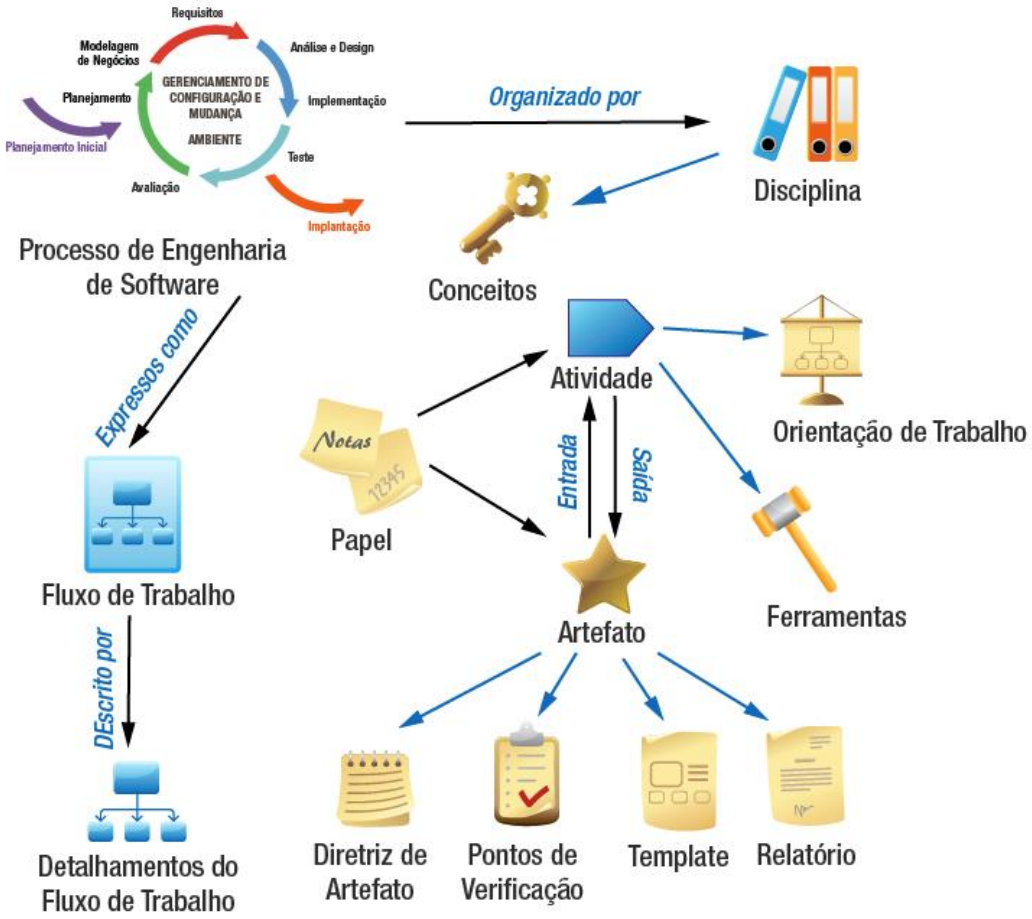
O **Processo Unificado Rational** (RUP) pode ser utilizado de forma adaptada. Já vimos também que o RUP pode ser utilizado para qualquer tamanho de projeto e a sua construção está baseada nos processos do CMMI.

A organização pode alterá-lo conforme necessidade e evolução dos seus processos de desenvolvimento. Relembrando de forma resumida os módulos anteriores, o RUP descreve:

- Papéis ou perfis de trabalho, onde se apresenta a responsabilidade de cada um;
- Artefatos, que representa o produto do processo, na verdade é o que vai ser gerado durante a execução das atividades;
- Fluxos de atividades orientam o caminho das atividades, define quando deve ocorrer a execução das atividades.

02

Os elementos descritos são organizados em um framework de processo de desenvolvimento de *software* e o RUP define um catálogo de elementos de processo que deve ser instanciado.



Elementos da estrutura do RUP

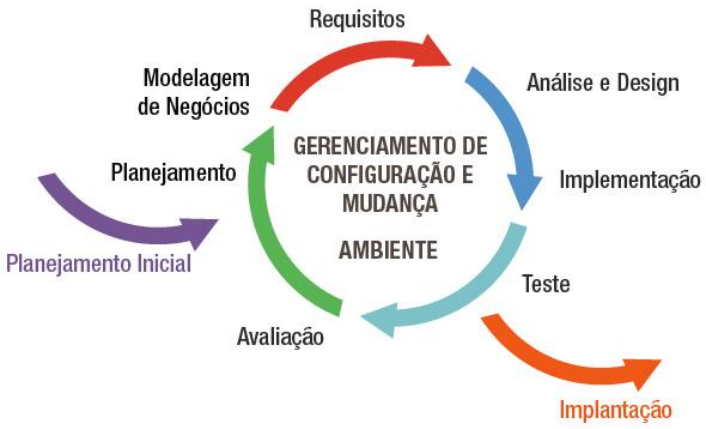
Isso significa que você terá que escolher os elementos que serão utilizados para compor o processo que você irá utilizar. Essa escolha depende de como sua organização trabalha. Algumas empresas personalizam os seus processos de desenvolvimento, outras simplesmente utilizam o RUP todo.

03

Como já vimos nos módulos anteriores, o RUP é baseado no **modelo espiral** e consiste em dividir a construção em iterações; essas iterações fazem parte de uma versão de software, um executável. Executável que contém as funcionalidades levantadas do cliente, ou seja, os requisitos do produto. A estruturação por iteração é importante para o cliente ter noção de como está seu projeto. Cada iteração tem suas atividades e riscos a serem vencidos.

Os *stakeholders* têm evidências do andamento do projeto e dos possíveis problemas, os quais já podem ser solucionados ao longo do processo.

Assim, o projeto tem um estado determinado em evidências do andamento, conforme os artefatos vão sendo entregues durante o projeto.



Ciclo de vida do RUP

04

2 - CAUSAS DE PROBLEMAS EM PROJETOS

Quando o RUP foi confeccionado ou criado, ocorreram vários estudos em “n” projetos da época. A ideia era levantar todos os problemas que ocorriam nos projetos para propor uma metodologia que resolvesse os problemas encontrados.

Mas quais eram os problemas daquela época que provocavam falha nos projetos?

Começamos elencando a **falta de entendimento das necessidades do cliente** ou o entendimento incompleto. Outro problema, a **falta de habilidade com as mudanças nos requisitos**, consequentemente problemas com integração de módulos de sistemas. A manutenção era difícil e as melhorias impensáveis, sendo melhor fazer outro projeto. As **falhas eram descobertas muito tarde**, o que causava um prejuízo enorme para o cronograma do projeto e os custos também. Como o processo era demorado, corria o risco de perder pessoas com conhecimento do negócio.

Na época não existiam ferramentas automatizadas de testes e de controles de versões, isso gerava um problema enorme com controles e se gastava muito tempo para isso.

Citamos alguns problemas da época, os quais ainda continuam existindo em muitas empresas. Por incrível que pareça, atualmente há uma grande incidência desses problemas que aconteciam com as metodologias estruturadas (cascata).

05

Em decorrência desses problemas, os autores do RUP criaram algumas práticas para tentar eliminar essas ocorrências que fracassavam os projetos, que são:

• **Desenvolver *software* iterativamente** – o RUP divide o projeto em versões, e essas versões possuem várias iterações, um conjunto de iterações que finalizadas, finalizam a versão. Saiba+

• **Modelar o *software* visualmente** – a criação de protótipos ajuda muito o entendimento das necessidades do cliente, além de corrigir as intenções de desenvolvimento equivocadas. Com o modelo o cliente pode visualizar e alterar os requisitos, além de possibilitar à equipe visualizar a arquitetura que será necessária para construção do *software*.

• **Gerenciar os requisitos** – Essa prática permite que o projeto ataque primeiro os requisitos de maior risco, ou seja, primeiro se trabalha com aqueles requisitos de maior criticidade e risco. O gerenciamento de risco aborda a elicitação de requisitos, a organização, documentação das funcionalidades, avalia as mudanças realizando a análise de impacto, realiza a priorização dos requisitos e uma das coisas mais importantes é que permite realizar a rastreabilidade dos requisitos.

• **Verificação da qualidade continuamente** – a verificação da qualidade do software acompanha o desenvolvimento iterativo e incremental. Saiba+

• **Arquiteturas baseadas em componentes** – de acordo com o RUP, a primeira coisa na fase de Elaboração que se deve fechar é a arquitetura do sistema. A arquitetura define a estrutura do sistema, tempos de respostas, comportamento da aplicação.

• **Gerenciar e controlar as mudanças** – todas as solicitações de mudanças devem estar devidamente levantadas e documentadas, onde se realiza uma validação e uma análise. Saiba+

Saiba+ (Desenvolver *software* iterativamente)
Versão é um código implantado em produção. Essas iterações são avaliadas e corrigidas a tempo, proporcionando o custo bem menor que anteriormente para as mudanças e correções. A evolução das iterações permite que o cliente fique sabendo em que momento ou fase se encontra o seu projeto e quais são as pendências e erros que estão acontecendo.

Rastreabilidade
O gerenciamento de requisitos é uma tarefa muito importante e que atualmente tem evoluído muito. Hoje no mercado há vagas específicas para analistas de requisitos, além de várias ferramentas que ajudam a gerenciar os requisitos do projeto e do sistema. A rastreabilidade possibilita que o analista de requisito saiba tudo que será afetado com a mudança solicitada. Isso garante que a alteração não trará danos e impactos para outras partes do sistema. Por exemplo, em uma reforma de uma casa o morador quer tirar uma parede da casa para ampliar a cozinha. Simples, mas olhando a planta da casa descobriu que na parede passa um cano de água além de uma fiação elétrica. Se essa análise não é realizada, provavelmente a casa ficaria algum tempo sem água e sem energia, sem contar que esses impactos não estavam previstos no orçamento inicial. Com a realização dessa análise preliminar, que chamaremos de análise de impacto, o morador pode prever antes se vale a pena tirar a parede do lugar, ou melhor, saberá exatamente o que vai ser alterado e seus riscos. Ele irá tomar uma decisão mais tranquila, pois já conhece o que vai ser afetado.

Saiba+ (Verificação da qualidade continuamente)
Cada iteração e cada versão são verificadas de acordo com os níveis exigidos e o momento que o *software* se encontra. São realizados testes nos incrementos entregues além de verificar se toda a documentação está de acordo com o que foi construído.

Arquitetura
A arquitetura trata também a capacidade do seu *software* de receber usuários. Já algumas vezes temos exemplos de sites que não estão preparados para receber grandes quantidades de usuários. Geralmente são os sites de promoção divulgados na TV. A arquitetura trata isso. Por outro lado a TI trabalha com códigos organizados, que podem ser usados por vários *softwares* na empresa, são os componentes. Por exemplo, você não vai desenvolver um código para tratar a validação do CPF, isso já existe, provavelmente é um simples componente, que todos os sistemas da empresa utiliza. A utilização de componentes ganha tempo na programação e possibilita o reuso.

Saiba+ (Gerenciar e controlar as mudanças)
Essa análise, que muitos chamam de análise de impacto, revela a rastreabilidade dos requisitos. A vantagem dessa rastreabilidade é que você saberá o que vai alterar com a mudança solicitada. Sabendo o que deverá mexer proporciona mais tranquilidade e menos riscos para o *software*, pois não se pode construir uma coisa e estragar outra. Depois que é realizada uma manutenção no aplicativo, o analista deve gerenciar as versões que estão entrando em produção, bem como as que estão em desenvolvimento, isso garante que a equipe estará implementando o código novo na versão de *software* correta.

06

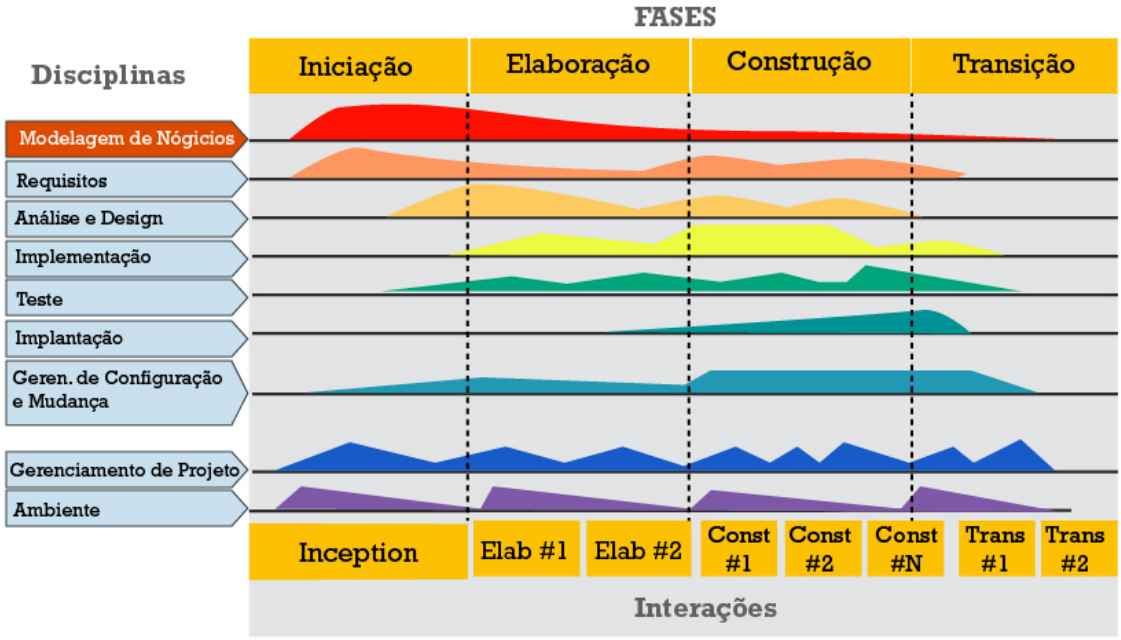
3 - Algumas Particularidades do RUP

Além das práticas o RUP possui três características nas quais se baseiam sua utilização:

• É uma metodologia orientada a caso de uso (requisito);
• O seu modelo (framework) de processo pode ser alterado de acordo com a necessidade da organização, geralmente a equipe da Qualidade gerencia todos os processos;
• Utiliza várias ferramentas automatizadas.

a) Estrutura do RUP

Como já vimos anteriormente,temos dois eixos na estrutura do RUP: horizontal, onde percorrem as disciplinas e o eixo vertical, com as fases e iterações.



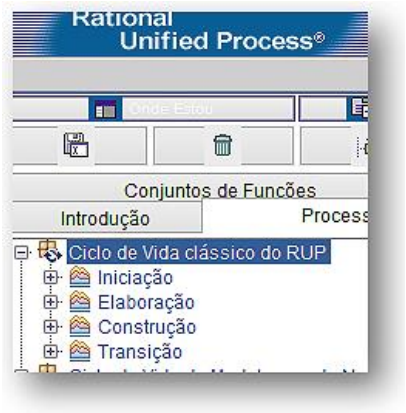
As disciplinas percorrem as fases ao longo do desenvolvimento do software. O gráfico no centro da imagem representa o gráfico das baleias, no qual conforme o tamanho da curva se descreve a intensidade de uso da disciplina na fase.

Essa estrutura é a padrão do RUP. Como o RUP pode ser personalizado então essa estrutura poderá ser alterada de acordo com as necessidades da empresa, mas sem perder as características essenciais: iterativo e incremental.

07

b) Fases do RUP

Apresentaremos abaixo, a descrição das fases retiradas do próprio RUP 7.0.



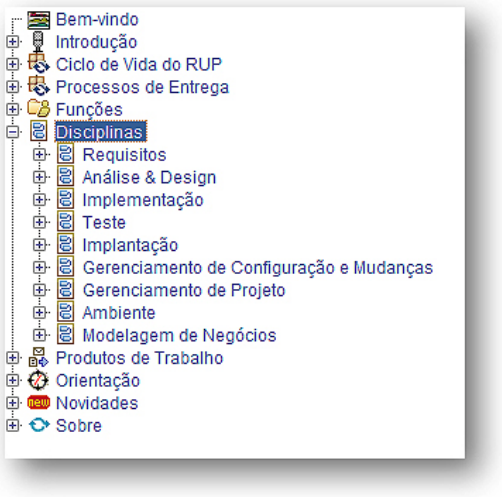
Fases do RUP

Iniciação	A meta da fase 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 devem ser tratados para que o projeto possa prosseguir. Para as manutenções que visam melhorias em um sistema existente, a fase de iniciação é mais rápida, mas ainda se concentra em assegurar que a manutenção seja compensatória e que seja possível fazê-la.
Elaboração	A meta da fase de elaboração é ter a arquitetura do sistema com 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.
Construção	A meta da fase de construção é esclarecer os requisitos restantes e concluir o desenvolvimento do sistema com base na arquitetura. 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.
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 isso 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 trabalhados muito antes no ciclo de vida do projeto.

08

c) Disciplinas do RUP

Apresentaremos abaixo, a descrição das disciplinas retiradas do próprio site do RUP 7.0.



RUP 7.0 –Disciplinas do RUP

• **Requisitos** - Esta disciplina explica como eliciar os requisitos dos clientes e transformá-los em um conjunto de requisitos detalhados sobre o que faz o sistema.Saiba qual é a finalidade dessa disciplina.

• **Análise e Design** - Esta disciplina explica como transformar os requisitos do produto em produtos de trabalho especificando o design do software que o projeto desenvolverá. Veja qual é a finalidade dessa disciplina.

Finalidade
A finalidade da disciplina Requisitos é:

- Estabelecer e manter concordância com os clientes sobre o que o sistema deve fazer;
- Oferecer aos desenvolvedores do sistema uma compreensão melhor dos requisitos do sistema;
- Definir os limites do sistema;
- Fornecer uma base para planejar o conteúdo técnico das iterações;
- Fornecer uma base para estimar o custo e o tempo de desenvolvimento do sistema;

Definir uma interface de usuário para o sistema, focando nas necessidades e metas dos usuários.

Finalidade
A finalidade de Análise e Design são:

- Transformar os requisitos em um design do sistema a ser criado;
- Desenvolver uma arquitetura sofisticada para o sistema;

Adaptar o design para que corresponda ao ambiente de implementação, projetando para um bom desempenho.

09

• **Implementação** - Esta disciplina explica como desenvolver, organizar, testar os códigos e integrar os componentes implementados de acordo com as especificações do design. Veja qual é a finalidade da implementação.

• **Teste** - Essa disciplina fornece orientação sobre como avaliar a qualidade do produto. A disciplina de Teste age como um fornecedor de serviços para as outras disciplinas de diversas maneiras. Os testes são direcionados principalmente na avaliação da Qualidade do Produto, que é realizada através de algumas práticas.

• **Implantação** - A disciplina de implantação descreve as atividades associadas para garantir que o produto de *software* esteja disponível aos seus usuários. O pico das atividades de implantação é na Fase de Transição, algumas das atividades ocorrem em fases anteriores como planejar e preparar a implantação.

• **Gerenciamento de Configuração e Mudança** - Esta disciplina explica como controlar e sincronizar a evolução do conjunto de produtos de trabalho que compõem o sistema de software. Importante ter uma ferramenta de gerenciamento de configuração e mudança para controlar os vários produtos de trabalho produzidos por muitas pessoas que trabalham em um projeto em comum. O controle ajuda a evitar confusões dispendiosas, e assegura que os produtos de trabalho resultantes não entrem em conflito devido a alguns problemas.

Finalidade da implementação

A finalidade da implementação é:

- definir a organização do código em termos de subsistemas de implementação organizados em camadas;
- implementar os elementos de design em termos de elementos de implementação (arquivos de origem, executáveis e outros);
- testar os componentes desenvolvidos como unidades;

integrar os resultados produzidos por implementadores individuais (ou equipes) ao sistema executável.

Práticas

A avaliação da Qualidade do Produto é realizada através das práticas a seguir:

- Localizar e documentar defeitos na qualidade do *software*;
- Sugestões sobre a qualidade do *software*;
- Validar e provar as suposições feitas nas especificações de projeto e requisitos através de demonstração concreta;
- Validar se o *software* funciona conforme o projeto;

Validar se os requisitos são implementados adequadamente.

Problemas
Além de evitar confusões dispendiosas, o controle assegura que os produtos de trabalho resultantes não entrem em conflito devido a alguns dos seguintes tipos de problemas:

- **Atualização simultânea** - Quando dois ou mais membros da equipe trabalham separadamente no mesmo produto de trabalho, o último membro a fazer mudanças desfaz o trabalho realizado pelo anterior. O problema básico é que, se um sistema não permite a atualização simultânea, isso leva a mudanças em série e diminui o ritmo do processo de desenvolvimento. Entretanto, com a atualização simultânea, o desafio é detectar se ocorreram atualizações simultaneamente e resolver quaisquer problemas de integração quando essas mudanças forem incorporadas.

- **Notificação Limitada** - Quando um problema é corrigido nos produtos de trabalho compartilhados por vários desenvolvedores, e alguns deles não são notificados da alteração.

- **Múltiplas versões**- A maioria dos programas de grande porte é desenvolvida em liberações evolutivas. Uma liberação pode estar sendo utilizada pelo cliente, enquanto outra está em teste e uma terceira ainda está em desenvolvimento. Se forem encontrados problemas em qualquer uma das versões, as correções deverão ser propagadas entre elas. Isso pode levar a confusões, que acarretam correções dispendiosas e retrabalho, a menos que as mudanças sejam cuidadosamente controladas e monitoradas.

10

• **Gerenciamento de Projeto** - Esta disciplina foca no planejamento do projeto, gerenciamento de riscos, monitoramento do progresso e métricas. O objetivo é facilitar a tarefa fornecendo algum contexto para o Gerenciamento do Projeto. Não se trata de uma receita de sucesso, mas apresenta uma abordagem para gerenciar o projeto que poderá ajudar a entregar o *software*. A disciplina não abrange todos os aspectos do gerenciamento do projeto. Saiba+

• **Ambiente** - A disciplina de ambiente fornece o ambiente para o desenvolvimento de *software*, que a equipe de desenvolvimento irá utilizar, incluindo os processos e ferramentas. A finalidade da disciplina de Ambiente é fornecer suporte à equipe de desenvolvimento.

• **Modelagem de Negócios** - Essa disciplina fornece orientação sobre diferentes técnicas de modelagem que podem ser utilizadas durante o levantamento das informações do negócio. As finalidades da modelagem de negócio são:

• Entender os problemas atuais na organização de destino e identificar as possíveis melhorias;
• Avaliar o impacto da alteração organizacional;
• Assegurar que os clientes, usuários, desenvolvedores e outros parceiros tenham uma compreensão comum da organização;
• Derivar os requisitos do sistema de *software* necessários para suportar as necessidades da

empresa;

• Entender como um sistema de *software* a ser implementado se ajusta à organização.

Saiba+
Por exemplo, não abrange questões como:

- Gerenciamento de pessoas: contratar, treinar, instruir;
- Gerenciamento de orçamento: definir, alocar e assim por diante;
- Gerenciamento de contratos com fornecedores e clientes.

11

4 - OpenUp

Alguns colegas da IBM começaram a pensar sobre como seria possível criar uma versão ágil do RUP. Ao mesmo tempo em que esse novo processo deveria ser ágil, também teria que refletir as boas práticas já contidas no RUP e consolidadas no mercado de *software*. Surgiu, então, o OpenUP.

O OpenUP é um processo para desenvolvimento de *software*, que está em conformidade com os princípios do Manifesto do Desenvolvimento de *Software* Ágil. Além de possuir uma abordagem iterativa e incremental, é um processo de baixa cerimônia, que não está associado a nenhuma ferramenta específica.

Trata-se de um processo considerado **mínimo**, **completo** e **extensível**, valorizando a colaboração entre a equipe e os benefícios aos interessados ao invés da formalidade e artefatos desnecessários.

O OpenUP é baseado em quatro **princípios** básicos, que são:

12

O processo OpenUp pode ser facilmente entendido através das três camadas descritas a seguir.

1. Ciclo de Vida do Projeto – **Fases** com foco nas necessidades dos *stakeholders*

O OpenUP apresenta a mesma distribuição de fases já conhecidas no RUP, onde o critério de saída de cada fase é no mínimo responder as seguintes perguntas:

- Iniciação: Todos os stakeholders concordam com o escopo e objetivos do projeto?
- Elaboração: Todos concordam com a arquitetura proposta e o valor entregue ao cliente, considerando os riscos levantados?
- Construção: Existe uma aplicação que está quase pronta rodando bem próxima a ser finalizada?
- Transição: A aplicação está finalizada e o cliente satisfeito?

2. Ciclo de Vida da **Iteração** com foco no time

Além da divisão por fases já conhecida, o OpenUP divide o projeto em iterações (também conhecidas como *sprints* segundo a metodologia SCRUM) planejadas que podem variar de alguns dias a algumas semanas. Ao final de cada iteração deve ser gerado um incremento ao produto e também é realizada uma retrospectiva e avaliação, onde são discutidas as lições aprendidas e a saúde do projeto.

3. **Microincrementos** com foco individual

Um microincremento é a execução de um pequeno passo que deve ser mensurável para alcançar os objetivos de uma iteração. Este pode representar o resultado de alguns dias ou horas de trabalho de uma pessoa ou um grupo determinado.

Retrospectiva
Importante salientar que o principal objetivo da retrospectiva e da avaliação é aprender com erros e acertos e não apontar culpados.

13

5 - Comparando o RUP ao OpenUP

Como citado, o OpenUP teve sua origem a partir do próprio RUP, portanto podemos encontrar uma série de similaridades considerando que o OpenUP apresenta uma quantidade bem menor de produtos de trabalho, papéis e tarefas.

Então temos algumas diferenças que são importantes destacar. Além da menor formalidade e quantidade de produtos de trabalho a primeira diferença é a introdução do conceito de micro incrementos. Como já descrito acima, o microincremento representa a execução de uma pequena unidade do trabalho e deve ser bem definido para que a equipe possa controlar e monitorar o progresso diário. Cada microincremento é especificado e controlado através de itens de trabalho onde o monitoramento é diário.

Para entendermos melhor esse conceito podemos considerar o exemplo abaixo:

Definir a visão do produto é uma tarefa que muitas vezes pode levar semanas, então para assegurar o controle diário é necessário dividir a tarefa em partes menores como, por exemplo:

• Identificar os *stakeholders* do projeto.	ou	• Identificar as principais necessidades.

Outro exemplo seria a tarefa de **desenvolver a solução**. O RUP recomenda que o caso de uso ou cenário seja a unidade de implementação, planejamento e controle de progresso do projeto, mas mesmo que consideremos um cenário de um caso de uso, poderíamos levar dias ou semanas para fazer a especificação, design, implementação e teste. Podemos então dividir essa tarefa em unidades menores para o controle diário como especificar, implementar ou testar um passo de um cenário de um caso de uso.

14

Outra diferença que merece uma atenção especial é a disciplina de **gerência de projetos**. O principal objetivo de um projeto ágil é reduzir o espaço de tempo entre os chamados, ou seja, garantir o monitoramento diário do gerente do projeto para assegurar o bom progresso e permitir que a equipe reaja o mais cedo possível a qualquer situação de mudança ou risco.

Para acompanhar o status do projeto, o OpenUP recomenda os “Scrum Daily meetings”, onde através de reuniões rápidas diárias, a equipe inteira do projeto possa compreender o que os outros membros já realizaram, remover possíveis barreiras e planejar o que será feito até a próxima reunião.

Outra prática não capturada pelo RUP atualmente é o que chamamos de **auto-organização do time de projeto**. Se você deseja utilizar o RUP de uma maneira ágil, é necessário obrigatoriamente adotar essa prática. A auto-organização impacta varias áreas incluindo como o time irá se planejar e comprometer-se com o que deve ser feito (pelo time e não por um indivíduo), como o trabalho deverá ser atribuído (membros do time se “auto atribuem” ao trabalho ao invés de serem alocados por algum gerente ou líder). E, por último, auto-organização tem a ver com a forma pela qual os membros do time interagem dentro do projeto, lembrando que o papel do individuo dentro do projeto é muito mais importante que seu cargo funcional.

Apesar das diferenças citadas acima, ambos os processos são bem semelhantes, pois são fundamentados basicamente nas mesmas práticas já conhecidas no Rational Unified Process, focando principalmente na redução de riscos e aumento de valor aos *stakeholders* do projeto.

15

RESUMO

O Processo Unificado é utilizado por uma boa parte do mercado de desenvolvimento de *software*. O RUP foi criado devido a problemas existentes com as metodologias estruturadas, e veio para suprir as necessidades do desenvolvimento do *software*. A sua estrutura de fases e disciplinas permite que as equipes adaptem de acordo com os processos de cada organização. Seus papéis, fluxos de atividades e artefatos são bem definidos e orientam do início ao fim do projeto a equipe de desenvolvimento. Sua

essência é iterativa e incremental, em que versões de*softwares* são incrementadas ao *software* que está sendo gerado. Essas iterações permitem as avaliações e testes durante todo o andamento do projeto. A sua estrutura conta também com fases (iniciação, elaboração, construção e transição) e disciplinas (requisitos, análise & design, implementação, teste, implantação, gerenciamento de configuração e mudanças, gerenciamento de projeto, ambiente e modelagem de negócios), tendo cada uma suas particularidades. O gráfico das baleias demonstra a intensidade de cada disciplina em todas as fases do projeto. O OpenUP também é um processo para desenvolvimento de *software*, que é baseado no RUP e está em conformidade com os princípios do Manifesto do Desenvolvimento de *Software* Ágil. Além de possuir uma abordagem iterativa e incremental, é um processo de baixa cerimônia, que não está associado a nenhuma ferramenta específica.

UNIDADE 2 – MÉTODOS ÁGEIS E PROCESSO UNIFICADO

MÓDULO 4 – APLICAÇÃO DO PROCESSO UNIFICADO NO DESENVOLVIMENTO DE SISTEMAS

01

1 - ATIVIDADES INICIAIS DE UTILIZAÇÃO DO PROCESSO UNIFICADO

Existem várias formas de se fazer alguma coisa. Com o *software* também não é diferente, podemos utilizar qualquer metodologia ou nem usar, como podemos utilizar uma linguagem ou outra para desenvolver os códigos para o sistema. Tudo depende das características organizacionais da empresa. Essas características englobam os conhecimentos de seus técnicos em relação às metodologias e linguagens de programação.

A estrutura RUP constitui uma orientação sobre um rico conjunto de princípios de engenharia de *software*. É aplicável a projetos de diferentes tamanhos e complexidades, bem como para diferentes ambientes e domínios de desenvolvimento. Isso significa que nenhum projeto ou organização se beneficiará do uso de tudo do RUP, devido à vasta biblioteca. Aplicando tudo do RUP provavelmente resultará em um ambiente de projeto ineficiente, em que as equipes se esforçarão muito para manter o foco nas tarefas importantes e para encontrar o conjunto certo de informações. Portanto, recomendamos que o RUP seja direcionado a fornecer orientação apropriada e personalizada sobre como desenvolver o*software* conforme características de cada empresa.

Essa orientação oferece um resumo de alto nível do conceito de adaptação do RUP. Equipes da Qualidade devem trabalhar na melhor forma de utilização do RUP pelas equipes de projeto. A adaptação do processo pode ocorrer em dois níveis:

- No nível organizacional;
- No nível do projeto.

Não importa para qual nível da organização o processo está sendo adaptado, a abordagem geral do Processo Unificado é a mesma, embora as preocupações sejam diferentes.

No nível organizacional
No nível organizacional os engenheiros de processo (Qualidade) modificam, aprimoram ou configuram

um processo comum para ser utilizado no nível da organização. A adaptação no nível de organização leva em consideração questões como, por exemplo, domínio de linguagens, arquitetura, práticas de reutilização e tecnologias controladas pela empresa. Uma organização pode ter mais de um processo para o desenvolvimento do projeto, cada um deles adaptado para um tipo diferente de desenvolvimento. Em alguns casos, a configuração clássica do RUP predefinida serve como o processo no nível da organização.

No nível do projeto
No nível do projeto os engenheiros de processo ou a própria equipe de projetos modificam, melhoraram ou configuram um processo comum a ser utilizado em um projeto específico. A adaptação do RUP leva em consideração vários aspectos: o tamanho do projeto, tecnologia, a reutilização dos ativos da empresa, o tipo de ciclo de vida do desenvolvimento (projeto ou manutenção) e assim por diante.

02

Para o bom uso do RUP é importante que a equipe conheça a sua estrutura, sendo que é muito aconselhável que tenha uma equipe da Qualidade na empresa, que acompanhe a evolução da metodologia e forneça treinamentos para seus gerentes e técnicos.

No decorrer do nosso estudo você conhecerá o **analista de negócio**. É um papel executado por um técnico, que tem habilidades para interpretar a linguagem do cliente e consegue relacionar-se facilmente com as partes interessadas pelo negócio. O analista de negócio desempenha um papel importantíssimo no processo de levantamento das necessidades do gestor.

Algumas empresas não investem nesse tipo de perfil ou até mesmo colocam pessoas não preparadas para executar essas atividades. Isso na maioria das vezes gera um atraso no cronograma, pois o entendimento necessário somente acontece tardiamente e com o projeto já em andamento, o que gera prejuízo em termos de custos, prazos, na imagem da equipe e consequentemente da empresa também.

Para iniciar a utilização do RUP é importante primeiramente desenvolver um **plano do desenvolvimento do projeto**. Esse plano define os requisitos de alto nível e descreve como a equipe irá desenvolver e testar a versão de *software*. O gerente de projetos é o responsável por essa atividade. Veremos a seguir uma sugestão de roteiro.

03

Para planejar o lançamento da versão do *software*, a equipe poderá executar o seguinte roteiro:

Conforme item 1 o analista de negócio trabalha para desenvolver uma visão clara das necessidades reais do produto do cliente. No RUP existe um artefato chamado de Visão. O documento de **Visão** captura requisitos de gestão, além das restrições do projeto e fornecerá à equipe um entendimento geral do sistema a ser desenvolvido. Ele fornece informações sobre o andamento do projeto e também de como serão realizadas as aprovações e o planejamento preliminar das entregas. Este documento está diretamente relacionado ao Caso de Negócio.

O documento de Visão informa as principais características do projeto e necessidades, apresenta como será realizada a comunicação e principalmente quem tomará as decisões para alguns conjuntos de tarefas.

Caso de Negócio
Caso de Negócio é um documento que informa as características do negócio como: justificativa de negócio para o projeto, questões financeiras do projeto, solução recomendada, como o projeto poderá ajudar a resolver o problema, satisfação do cliente, benefícios esperados e quais serão as consequências se o projeto não for realizado. Esse documento deve "contar a história" de *porque*, *o que*, *como*, *quando* e *aonde*.

04

O conteúdo do **documento de Visão**, juntamente com quaisquer outros artefatos de requisitos relacionados no RUP, deve responder às perguntas a seguir, que podem ser divididas em artefatos separados e mais detalhados, conforme necessário:

- Quais são os termos-chave? (Glossário)
- Que problemas estão tentando resolver? (Declaração do Problema)
- Quem são os investidores? Quem são os usuários? Quais são as suas necessidades?
- Quais são as características do produto?
- Quais são os requisitos funcionais? (Casos de Uso)
- Quais são os requisitos não funcionais?
- Quais são as restrições de design?

Uma declaração clara do problema, solução proposta, e as características de alto nível do produto ajudam a estabelecer as expectativas e reduzir os riscos.

O desenvolvimento de uma visão de um conjunto compreensível de requisitos é a essência da disciplina de requisitos e o princípio das prioridades de cada Caso de Uso. Isso envolve analisar o problema, entender as necessidades dos clientes, definir o sistema e gerenciar os requisitos à medida que são evoluídos e alterados.

Antes de desenvolver a visão do projeto, recomenda-se criar uma infraestrutura para o seu projeto. Instalar e configurar ferramentas, adicionar membros da equipe, atribuir funções e permitir o acesso a áreas do projeto. O proprietário do produto ou analista de negócios deve identificar as partes interessadas e estudar o domínio do problema.

05

Quando falamos em **ferramentas** estamos nos referindo a quê? Que ferramentas seriam essas? Ferramentas são *softwares* também, só que ajudam a construir e controlar os *softwares* de negócios ou para os clientes. Atualmente muitas equipes utilizam ferramentas que apoiam o desenvolvimento do *software* ao longo da fase do projeto. Hoje, em sua maioria, as grandes empresas possuem esse conjunto de ferramentas, que gerenciam e apoiam o ciclo de vida do *software* por inteiro. Essas ferramentas apoiam todas as disciplinas e permitem o projeto ter mais agilidade em seus controles.

Após a construção do documento de visão deveremos revisá-lo e aprová-lo com as partes interessadas do projeto, tais como os usuários e o próprio cliente. Depois que a revisão estiver concluída, você terá um documento com os objetivos de negócio e as necessidades das partes interessadas, uma declaração do problema, uma declaração de solução e requisitos de alto nível. Você também saberá os recursos financeiros necessários para as partes interessadas.

Outro fator importante é que a equipe que participou das revisões e aprovações do documento de visão terá conhecimento das características do negócio, o que facilitará na construção dos outros documentos necessários para o projeto. Quanto mais pessoas da equipe participando dessas atividades, melhor para a equipe e para o projeto, pois não perderão tempo para conhecer o projeto posteriormente.

06

Após essas reuniões e fechamento das datas, o projeto deve ser gerenciado. O **plano de projetos** reúne as informações necessárias para o seu gerenciamento. Ele é utilizado para fazer o planejamento do projeto, prever as necessidades de recursos e para acompanhar o progresso desse planejamento. Ele envolve itens do tipo: organização do projeto, planejamento, orçamento e recursos. Também pode incluir planos para gerenciamento de requisitos, gerenciamento de configuração, resolução de problemas, garantia de qualidade, avaliação e teste e aceitação do produto.

Em um projeto simples, muitos desses tópicos podem ser abrangidos por uma ou duas sentenças cada um. Por exemplo, o planejamento de gerenciamento de configuração pode simplesmente declarar: "No final de cada dia, o conteúdo da estrutura de diretórios do projeto será compactado, copiado em um disco de zip etiquetado e datado, marcado com um número de versão e colocado no arquivamento central.”

O formato dos artefatos não é tão importante quanto as atividades de planejamento e as ideias contidas nele. Não importa a aparência dos planos ou as ferramentas utilizadas para construí-los.

Muitas equipes de projeto e qualidade vetam documentos por não estarem em um padrão determinado, ou por estar com erros de escrita, ou seja, erros que não afetam o produto em si. Isso é um erro, o código implantado tem preferência e é o principal em um projeto. Esses tipos de erros podem ser ajustados posteriormente, não podem ser impeditivos que causem atraso no projeto todo.

Qualquer metodologia prega um gerenciamento dos riscos. No Processo Unificado não é diferente. É essencial identificar e combater os riscos mais altos no início do projeto e acompanhá-los juntamente com outros problemas relacionados. A lista de riscos foi projetada para capturar os riscos percebidos

para o sucesso do projeto. Ela identifica, em ordem decrescente de prioridade, os eventos que poderiam levar a um resultado negativo significativo. Juntamente com cada risco, deve haver um plano para diminuí-lo. Isso serve como um ponto focal para planejar atividades do projeto e é a base para a organização das iterações. Saiba+

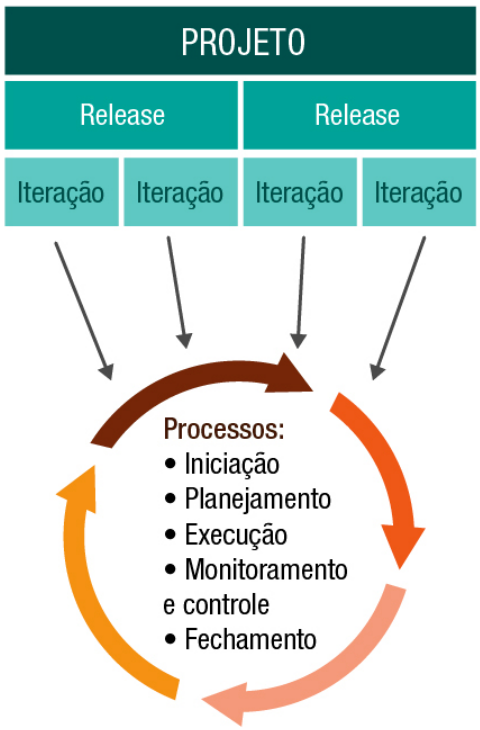
Saiba+
O Processo Unificado prega que a arquitetura deve ser a primeira coisa a ser tratada ou implantada. A arquitetura de um sistema de *software* é a estrutura para que o produto do cliente funcione adequadamente. São componentes significativos do sistema que mantêm a aplicação funcionando.
O RUP é uma abordagem iterativa de criação, de teste e de avaliação de versões executáveis do produto, a fim de afastar os problemas e resolver os riscos e as questões o mais cedo possível. Construir e testar incrementalmente os componentes do sistema são a "essência" das disciplinas de **Implementação** e de **Teste**.

07

2 - Pontos chave do Processo Unificado

A comunicação aberta contínua, com dados objetivos originados diretamente de atividades em andamento e as configurações do produto em desenvolvimento são importantes em qualquer projeto.

Avaliações regulares de *status* fornecem um mecanismo para endereçar, comunicar e resolver problemas de gerenciamento, problemas técnicos e riscos do projeto. Além de identificar os problemas, é necessário designar a cada um deles uma data de expiração e uma pessoa responsável pela resolução. Isso deve ser acompanhado e atualizado regularmente.



O acompanhamento do projeto fornece informações ao gerente de projeto sobre o andamento de todas as atividades. O histórico de projetos da empresa ajuda o gerente a tomar decisões para remover eventuais obstáculos que impeçam o andamento. Assim, a avaliação é essencial na abordagem iterativa, pois captura os resultados de cada iteração evidenciando o grau em que os critérios de avaliação foram atendidos, as lições aprendidas e as alterações a serem implementadas no processo, por isso, é importante sempre avaliar as iterações finalizadas para gerar um histórico, que será base de conhecimento. Dependendo do escopo e risco do projeto e da natureza da iteração, ela pode variar de um simples registro de demonstração e de resultados a um registro de revisão de teste formal e completo. O importante aqui é o foco nos problemas do processo, bem como nos problemas do produto.

08

Assim que o primeiro protótipo for colocado diante dos usuários, as alterações serão solicitadas. Para controlar essas mudanças e gerenciar eficazmente o escopo do projeto e as expectativas dos clientes, é importante que todas as mudanças em quaisquer artefatos de desenvolvimento sejam manutenidas por meio de **controles de mudanças** e gerenciadas com um processo consistente.

Os controles de mudança são usados para documentar e controlar defeitos, solicitações de melhorias e qualquer outro tipo de solicitação de mudança no produto. A vantagem dos controles de mudança é que eles fornecem um registro de decisões e devido a seu processo de avaliação, garantem que os impactos da possível mudança sejam compreendidos por todos os membros da equipe do projeto.

Os controles de mudanças são essenciais para o gerenciamento do escopo do projeto, bem como para a avaliação do impacto das mudanças propostas. Gerenciar e controlar o escopo do projeto à medida que as mudanças ocorrem em todo o ciclo de vida do projeto, enquanto mantém a meta de considerar todas as necessidades dos clientes e atendê-las ao máximo possível. Essa é a essência da disciplina de Gerenciamento de Mudanças e de Configuração.

09

3 - Finalidade do Processo

A finalidade de um processo é produzir um produto utilizável. Todos os aspectos do processo devem ser adaptados considerando essa meta. O produto é normalmente mais do que apenas o *software*. No mínimo, deve haver um **Guia do Usuário**, talvez implementado por meio da ajuda*on-line*. Também é possível incluir um **Guia de Instalação** e as **Notas** sobre o release. Dependendo da complexidade do produto, os materiais de treinamento também podem ser necessários, bem como uma lista de materiais juntamente com qualquer pacote do produto. As atividades associadas formam a disciplina de **Implantação**.

É essencial que seja escolhido um processo que se ajuste ao tipo de produto que está sendo desenvolvido. Mesmo depois que um processo é escolhido, ele não deve ser seguido às escuras. O bom senso e a experiência devem ser aplicados para configurar o processo e as ferramentas para atender às necessidades da organização e do projeto. A adaptação de um processo para um projeto é uma parte importante da disciplina Ambiente.

Os textos descritos acima percorreram as melhores práticas das disciplinas e são elementos essenciais, que fornecem um meio de avaliar rapidamente um processo e identificar áreas onde o aprimoramento é

mais vantajoso. É importante explorar o que ocorrerá se um desses elementos essenciais for ignorado. Por exemplo:

- Sem visão?
- Sem processo?
- Sem plano?
- Sem lista de riscos?
- Sem caso de negócio?
- Sem arquitetura?
- Sem protótipo?
- Sem avaliação?
- Não há solicitações de mudança?
- Sem suporte ao usuário?

Sem visão?
Sem visão você pode perder o rumo e ser facilmente confundido em desvios.

Sem processo?
Sem um processo comum, a equipe pode ter comunicações e compreensões errôneas sobre quem vai fazer o quê e quando.

Sem plano?
Sem plano você não conseguirá acompanhar o andamento.

Sem lista de riscos?
Sem lista de riscos você pode estar se concentrando nas questões erradas atualmente e pode explodir em uma mina inesperada daqui a cinco meses.

Sem caso de negócio?
Sem caso de negócio você se arrisca a perder tempo e dinheiro no projeto. Ele pode ser cancelado ou ir por água a baixo.

Sem arquitetura?
Sem arquitetura você pode não conseguir lidar com questões de comunicação, de sincronização e de acesso a dados, conforme elas surgem. Pode haver problemas com a capacidade de ajuste e o desempenho.

Sem protótipo?
Sem protótipo? Assim que possível, coloque um produto na frente do cliente. O simples acúmulo de papéis não garante a você ou ao cliente que o produto será bem-sucedido e aumenta o risco de estouro de orçamento e planejamento e/ou de defeito.

Sem avaliação?
Não finja que não é com você. É importante encarar a verdade. Quão próximo você realmente está do prazo? Das metas em qualidade ou do orçamento? Todas as questões estão sendo adequadamente acompanhadas?

Não há solicitações de mudança?
Se não há solicitações de mudança, como você acompanha as solicitações dos clientes? Como você as prioriza? E impede que as de prioridade mais baixa passem despercebidas?

Sem suporte ao usuário?
Se você não oferece suporte, o que acontece quando um usuário tem uma pergunta e não consegue entender como utilizar o produto? Com que facilidade se obtém ajuda?

10

4 - Estratégias de Planejamento

O RUP define em seu *framework* algumas estratégias de planejamento. Vejamos cada uma delas.

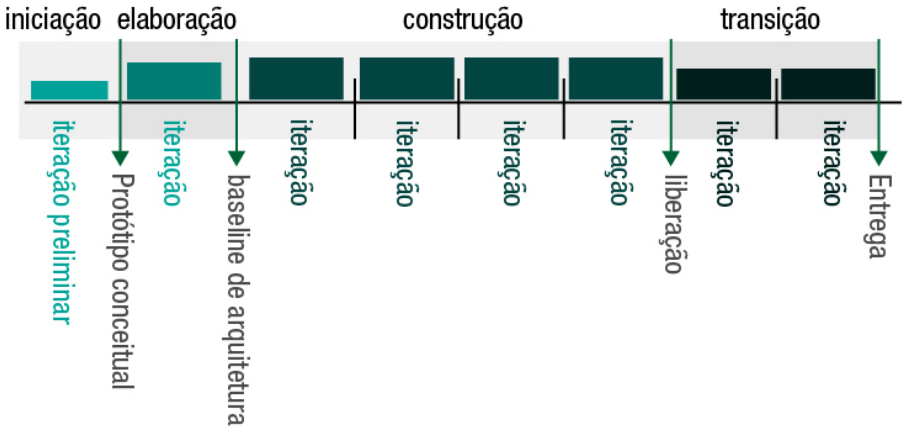
a) Estratégia incremental

A estratégia incremental é aquela que determina as necessidades do usuário e define os requisitos do sistema e, em seguida, desempenha o restante do desenvolvimento em uma sequência de construções.

A primeira construção incorpora partes dos recursos planejados, a próxima construção inclui mais recursos e assim por diante até o sistema estar completo. **Iterações que caracterizam essa estratégia**:

- Uma iteração curta de Iniciação para estabelecer o escopo e a visão, e para definir o caso de negócio;
- Uma única iteração de Elaboração, durante a qual os requisitos são definidos e a arquitetura estabelecida;
- Várias iterações de Construção durante as quais os casos de uso são realizados e a arquitetura

é aprimorada;
- Várias iterações de Transição para migrar o produto para a comunidade de usuários.



Essa estratégia é apropriada quando:

- o domínio do problema é familiar;
- os riscos são bem entendidos;
- a equipe do projeto é experiente.

Familiar
O domínio do problema é considerado *familiar* quando:

- A arquitetura e os requisitos podem ser estabilizados antecipadamente no ciclo de desenvolvimento;
- Há um baixo grau de novidade no problema.

11

b) Estratégia evolucionária

A estratégia evolucionária difere da incremental ao reconhecer que as necessidades do usuário não são completamente entendidas e todos os requisitos não podem ser definidos imediatamente, eles são apurados em cada construção sucessiva.

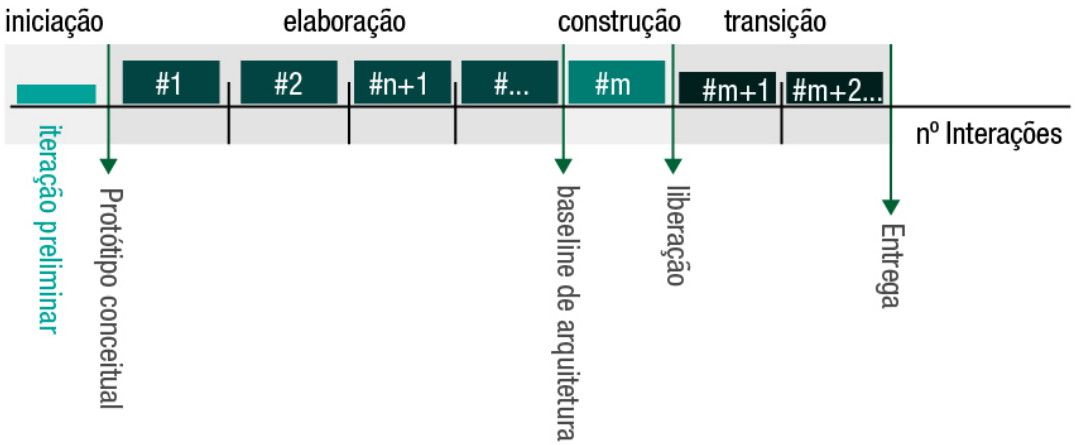
Iterações que caracterizam essa estratégia:

- Uma iteração curta de Iniciação para estabelecer o escopo e a visão, e para definir o caso de negócio;
- Várias iterações de Elaboração, durante as quais os requisitos são refinados em cada iteração;
- Uma única iteração de Construção, durante a qual os casos de uso são realizados e a arquitetura é expandida;

- Várias iterações de Transição para migrar o produto para a comunidade de usuários.

Essa estratégia é apropriada quando:

- o domínio do problema é novo ou não é familiar;
- a equipe é inexperiente.



12

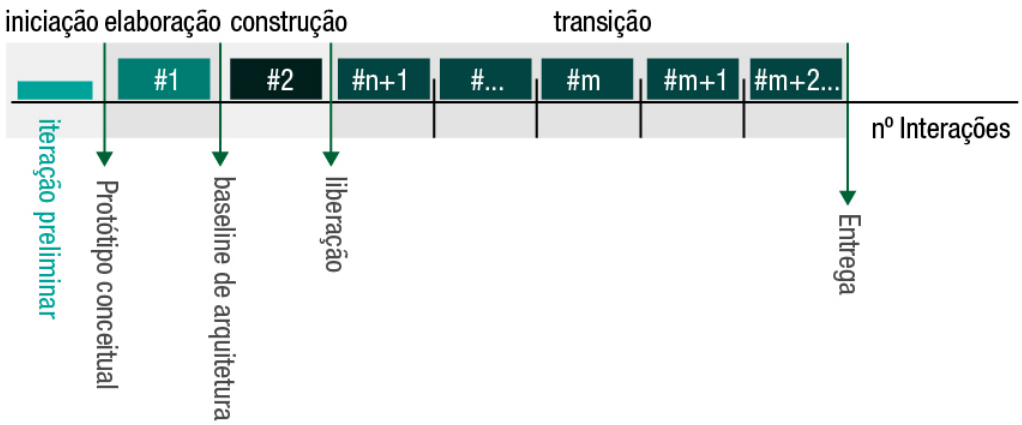
c) Estratégia de entrega incremental

A entrega incremental é quando a entrega da funcionalidade incremental é realizada em fases para o cliente.

Alguns autores consideram que a entrega incremental pode ser necessária quando há pressões de mercado sobre o tempo limitado, onde a liberação antecipada de certos recursos importantes pode render benefícios de negócio significativos.

A fase de transição começa cedo e tem a maioria das iterações. Essa estratégia requer uma arquitetura bastante estável, que é difícil de conseguir em um ciclo de desenvolvimento inicial para um sistema "sem precedentes". **Iterações que caracterizam essa estratégia**:

- Uma iteração curta de Iniciação para estabelecer o escopo e a visão, e para definir o caso de negócio;
- Uma única iteração de Elaboração, durante a qual é criada uma baseline de arquitetura estável;
- Uma única iteração de Construção, durante a qual os casos de uso são realizados e a arquitetura é aprimorada;
- Várias iterações de Transição para migrar o produto para a comunidade de usuários.



Essa estratégia é apropriada quando:

- o domínio do problema é familiar;
- a equipe é experiente;
- releases incrementais de funcionalidade têm alto valor para o cliente.

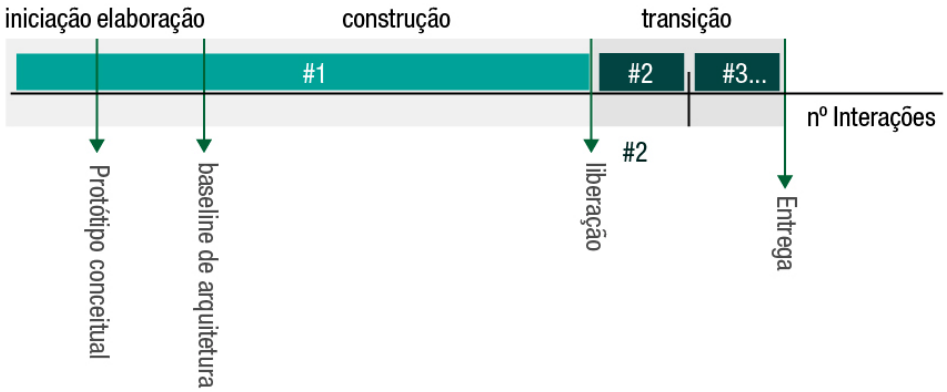
13

d) Cascata tradicional

A abordagem em **cascata tradicional** pode ser vista como um caso adulterado em que há apenas uma iteração na fase de construção. Na prática, é difícil evitar iterações adicionais na fase de transição.

As seguintes iterações são características:

- Uma iteração curta de Iniciação para estabelecer o escopo e a visão, e para definir o caso de negócio;
- Uma única iteração muito longa de Construção, durante a qual os casos de uso são realizados e a arquitetura é aprimorada;
- Várias iterações de Transição para migrar o produto para a comunidade de usuários.



Essa estratégia é apropriada quando:

- um pequeno incremento de funcionalidade bem definida está sendo adicionado a um produto muito estável;
- a nova funcionalidade é bem definida e bem entendida;
- a equipe é experiente, tanto no domínio do problema quando no produto existente.

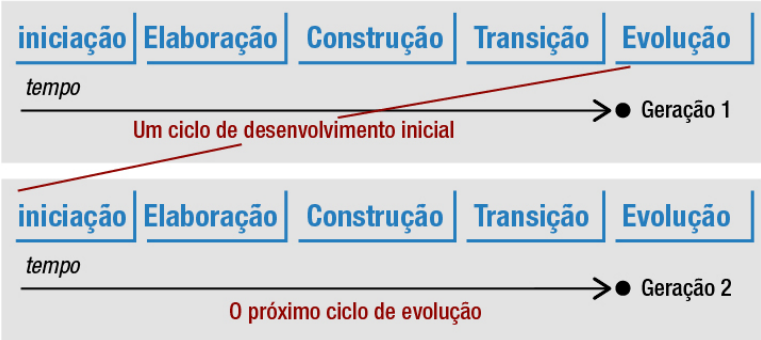
É importante destacar que, na prática, poucos projetos seguem exclusivamente uma única estratégia. É frequente acontecer uma evolução**híbrida**, no início, uma construção incremental e várias entregas.

14

5 - Releases ou versões

Reforçando os conceitos de Release e Versão, a passagem pelas quatro fases constitui um **ciclo de desenvolvimento** e a consequente**geração** do *software*. A menos que o produto "desapareça", ele irá desenvolver-se na próxima geração, repetindo a mesma sequência de fases de iniciação, elaboração, construção e transição, mas agora com ênfase diferente nas diversas fases. Esses ciclos subsequentes são chamados**ciclos de evolução**. À medida que o produto atravessa vários ciclos, são produzidas novas gerações.

A estratégia de planejamento definida para o projeto percorre as fases às quais pertencem as versões e releases entregues.



Os ciclos de evolução podem ser disparados por diversas razões:

- melhorias sugeridas pelos usuários,
- mudanças no contexto do usuário,
- mudanças na tecnologia subjacente,
- reação à concorrência e assim por diante.

Normalmente, os ciclos de evolução têm fases de Iniciação e Elaboração bem menores, pois a definição e a arquitetura básicas do produto foram determinadas pelos ciclos de desenvolvimento anteriores. Tudo depende da estratégia escolhida para o seu projeto ou manutenção.

15

Resumo

A estrutura RUP (Processo Unificado Rational) constitui uma orientação sobre um rico conjunto de princípios de engenharia de software e pode ser aplicável a projetos de diferentes tamanhos e complexidades. O RUP permite uma adaptação de seu processo, mas é importante que isso ocorra em dois níveis: nível organizacional e no nível do projeto. O artefato de chamado de Visão é o documento que informa as principais características do projeto e suas necessidades, apresenta como será realizada a comunicação e principalmente quem tomará as decisões. O Visão define também um dos pontos chave do Processo Unificado, como será a comunicação. Importante compreender a finalidade de um Processo em si, que é produzir um produto utilizável.

Um fator importante a destacar é a Estratégias de Planejamento a ser adotada para utilização do RUP. Podemos utilizar uma estratégia do tipo incremental, evolucionária, de entrega incremental, Cascata tradicional. Todas essas estratégias estão diretamente ligadas às entregas que são realizadas, que é um ponto muito importante do processo unificado, são as versões. Versões são os pedaços de sistemas que estão sendo acoplados um ao outro, o cliente gosta de ver a progressão do seu sistema, o que pode originar também a outras solicitações.