PUC Minas Virtual
educação sem distância

Desenvolvimento de Aplicações Web

Gerenciamento de Projetos de Aplicações Web

SÍNTESE DA DISCIPLINA

Isabella de Araujo Fonseca Campos

Sumário

1. Unidade I - Introdução ao Gerenciamento de Projetos Ágeis

Fundamentos dos Métodos Ágeis

A história da indústria de software teve sua origem por volta de 1960. Antes dessa época, o software era considerado parte integrante do hardware e não havia nenhuma comercialização desse produto como componente distinto das máquinas. Portanto, o hardware era comercializado juntamente com o software e o preço era estabelecido sobre o conjunto da solução. Entretanto, em um certo momento, ficou evidente que o desenvolvimento de software envolvia custos, os quais haviam-se tornado tão dispendiosos quanto o investimento total no hardware, e em alguns casos chegavam a superar o valor. Esse aspecto foi decisivo para que se iniciasse um movimento para a apropriação de um valor comercial ao software, desmembrando-o do valor inerente ao hardware.

Diante deste cenário, a indústria de software se viu diante de buscar incessantemente melhorias e maior número de acertos na construção de seus produtos. Já na década de 70, ocorreu a primeira grande mudança neste sentido – a Análise Estruturada – análise a ser desenvolvida com foco em suas funções: decomposição dos processos da organização em estudo, para daí serem derivadas as estruturas de dados necessárias para o estabelecimento do sincronismo entre essas funções. Na década de 80, foi o momento de foco nos dados, conhecida como Engenharia da Informação e a Análise Essencial. Se iniciou o processo de análise do software identificando-se o modelo conceitual de dados - modelo que representa a visão dos usuários sobre as informações necessárias, que é decomposto nos conjuntos de dados do software (utilizando-se uma técnica conhecida como "normalização") e, a partir desses conjuntos de dados, são implementadas as operações que aplicam-se sobre eles. Na década de 90, a Análise Orientada a Objetos prevaleceu.

A evolução do conjunto de todas estas técnicas permitiu uma melhoria na qualidade intrínseca dos produtos de software. Além das técnicas de construção citadas acima, ferramentas de projeto e programação, metodologias, processos, dentre outros, contribuíram para um maior amadurecimento e crescimento da indústria de software. Entretanto, problemas relacionados ao controle sobre os projetos de software permaneceu. Desvios no cronograma, custos excedentes, insatisfação quanto aos requisitos levantados e produto entregue ainda eram (e são!) algumas das dificuldades apresentadas. Aliado a estes problemas, os softwares necessários e construídos pelas organizações deixaram de ser departamentais. Para suprir as demandas do mercado, as empresas passaram a ter que desenvolver softwares capazes de comunicar com outras fontes de informação – diferentes das fontes internas conhecidas. Ou seja, as organizações passaram a ter necessidade de interagir através dos sistemas com fornecedores, clientes, parceiros, etc. Dessa forma, a comunicação e entendimento do que realmente é preciso fazer e o que é realmente executado se tornou um grande “gap“ que ainda deveria ser preenchido. Com a Internet, as coisas ficaram ainda mais complexas em função do aumento do número de variáveis, o aumento do número de usuários acessando (e o desconhecimento destes: quem são, como estão acessando, de linha discada ou não, qual a configuração da máquina, versão de software, etc).

Os métodos ágeis foram se fortalecendo dirigindo para este sentido de entendimento de uma nova forma de se construir software e se tornou uma vertente que está complementando e expandindo o conhecimento do que é software.

Em 2001, um grupo interessado em métodos iterativos e ágeis - estes termos estavam, na verdade, sendo ainda cunhados no contexto do desenvolvimento de software - se reuniram para encontrar um denominador comum. Este movimento gerou a "Agile Alliance" (www.agilealliance.com), com um manifesto e a declaração de alguns princípios. Muitas das práticas concretas existentes atualmente no universo ágil derivaram destes princípios.

Manifesto Ágil

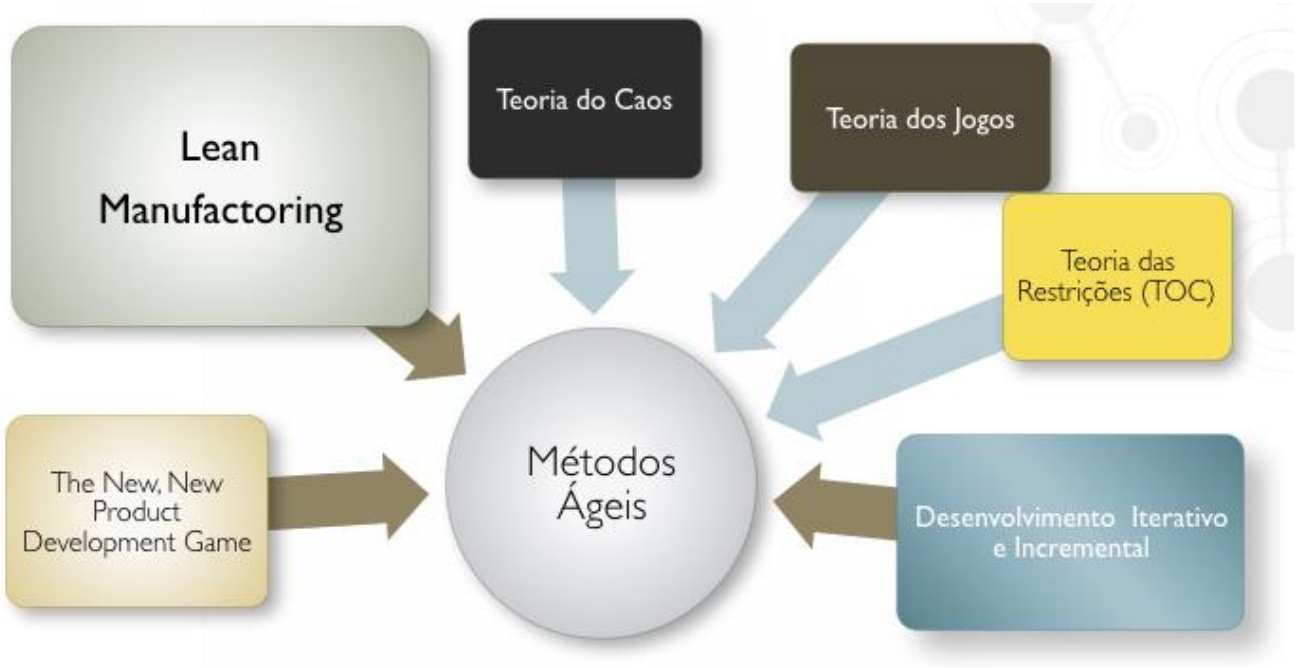
Apesar de reconhecer que há valor nos itens da direita, valorizamos mais os itens da esquerda:

- Indivíduos e interação entre eles mais que processos e ferramentas
- Software em funcionamento mais que documentação abrangente
- Colaboração com o cliente mais que negociação de contratos
- Responder a mudanças mais que seguir um plano

Os 12 princípios da Agilidade

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
4. Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
6. O método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
7. Software funcionando é a principal medida de progresso.
8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter conversação pacífica continuamente.
9. A contínua atenção à excelência técnica e bom design, aumenta a agilidade.
10. Simplicidade: a arte de maximizar a quantidade de trabalho não feito, é essencial.
11. As melhores arquiteturas, requisitos e projetos emergem de times auto-organizáveis.
12. Em intervalos regulares, o time deve refletir em como se tornar mais efetivo, então, se ajustar e otimizar seu comportamento de acordo.

Para que possamos entender os conceitos que embasam os métodos ágeis, abaixo seus principais fundamentos serão detalhados:



Desenvolvimento Iterativo e Incremental

Os termos "iterativo e incremental", cunhados no contexto das abordagens ágeis para desenvolvimento de software, possuem atualmente uma ampla utilização, porém nem sempre carregando o significado que realmente merecem.

Quando dizemos uma abordagem "iterativa", queremos dizer que o **produto** construído mediante esta abordagem será feito de forma gradual, iniciando-se a partir de uma ideia inicial, não acabada, recebendo novas avaliações e aprimoramentos até que esteja em condições aceitáveis em termos de valor para o cliente.

Quanto ao termo "incremental", significa a produção do produto final por meio de incrementos, criando oportunidades de análise e adaptação do **processo** de produção do produto a cada novo incremento.

Iterativo e incremental referem-se, portanto a uma abordagem em que o produto é construído gradualmente, em ciclos que buscam o alinhamento de prioridades e necessidades conforme a evolução do produto alcançada até o momento, permitindo, ao mesmo, tempo uma revisão constante do processo mediante qual está sendo construído este produto.

Lean Manufactoring

O lean é uma abordagem derivada do Sistema Toyota de Produção e tem como principais regras:

- Adicionar somente valor
    1. **Eliminar o desperdício** - Eliminar do processo tudo que não traga valor ao produto final.

2. **Minimizar inventário (estoques)** - O inventário no desenvolvimento de software corresponde à documentação que não é parte do produto final. Documentações de requisitos e design devem ser mantidos em níveis mínimos para maximizar o fluxo de desenvolvimento.
3. **Fazer corretamente na primeira vez (incorporar o feedback)** - Assegurar que cada componente possui qualidade a cada entrega. Parar a produção até que um produto de qualidade seja entregue (evitar criar frentes com foco em retrabalho).

- Centralizar naqueles que adicionam valor
  4. **Ampliar o poder daqueles que adicionam valor (decidir no nível mais baixo possível)** - oferecer a autoridade e as ferramentas necessárias para as pessoas no "chão de fábrica" tomarem as decisões necessárias
  5. **Criar uma cultura de melhoria contínua** - Adotar abordagens que favoreçam a execução do ciclo PDCA (*Plan, Do, Check, Action*) no ambiente do projeto de software e entre projetos.
- Obter valor dos clientes
  6. **Ir em direção aos requisitos do cliente (agora e no futuro)** - A abordagem de levantar, reunir e demandar do cliente uma assinatura sobre seus requisitos é considerada atualmente falha, porque, ao invés de fomentar a colaboração, tende a criar um ambiente de conflito entre clientes e desenvolvedores. A forma mais eficiente de ir em direção aos requisitos do cliente é a adoção de uma abordagem iterativa e incremental.
  7. **Puxar da demanda (decidir o mais tardiamente possível)** - Buscar a previsão perfeita pode não ser o caminho correto. O foco, portanto, deve ser em responder à mudança, e não em predizer todas as necessidades do produto.
  8. **Maximizar o fluxo** - Identificar e reduzir as frentes de WIP (*Work In Progress* - Trabalho em Andamento). Se o WIP for reduzido, então o tempo necessário para cada ciclo será também reduzido.

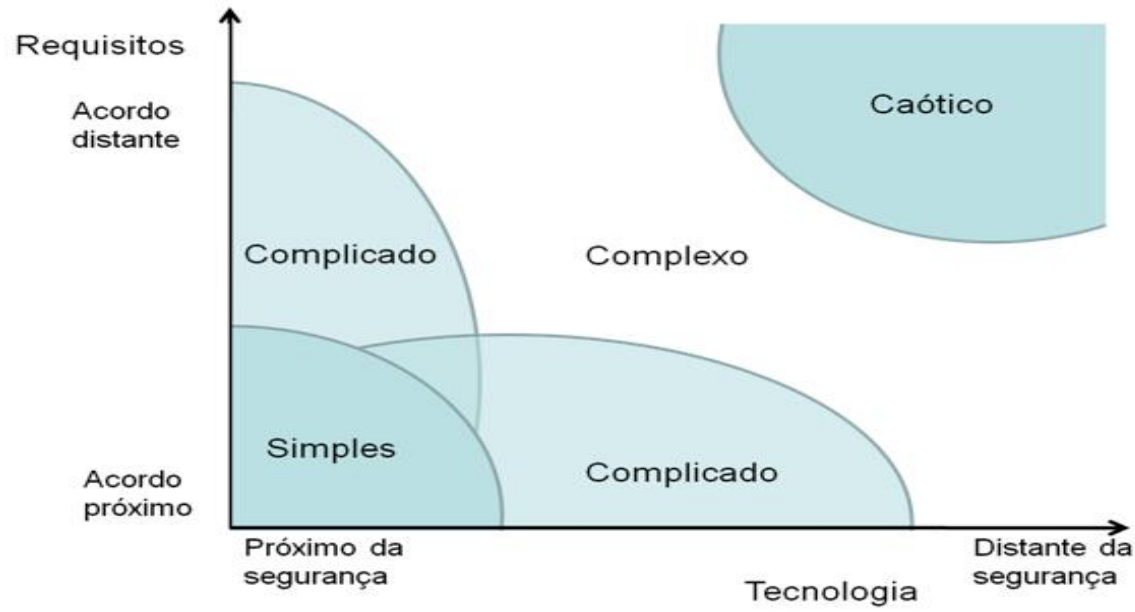
9. Otimizar o fluxo de valor
   10. **Abolir as otimizações locais** - Otimizações locais podem criar sub-otimizações no processo total e devem ser evitadas.
   11. **Forme parcerias com os fornecedores (use aquisições evolutivas)** - relações de confiança com um número limitado de fornecedores podem promover grandes vantagens. Com a diminuição do conflito entre fornecedor de software e cliente para controle do escopo, os esforços podem ser canalizados para a produção do melhor software possível para o cliente.

The New New Product Development Game

O terceiro fundamento que deve ser apresentado é a grande contribuição do artigo de Nonaka e Takeuchi - “The New New Product Development Game”. Foi publicado em 1986 na Harvard Business Review. Este artigo estuda as práticas de gestão que diferenciavam as empresas Fuji Film, Toyota, 3M e Xerox, Epson, Brother, NEC e Honda e identificam diferenças chave entre empresas japonesas e americanas inovadoras que se apresentavam tão a frente das demais.

Teoria do Caos

O quarto fundamento a ser estudado é a Teoria do Caos, que avalia o funcionamento de sistemas complexos e dinâmicos.



Esta figura explicita que processos preditivos não são adequados para o desenvolvimento de softwares complexos. Se já se conhece integralmente o escopo, as tecnologias e tudo mais que está envolvido no projeto de desenvolvimento de software, então provavelmente o Scrum não é necessário. No gráfico acima, a interseção dos tipos de complexidade ligados aos requisitos e à tecnologia definem a complexidade total do projeto. Adicionando-se a esta análise a dimensão de pessoas, a complexidade dos projetos de software amplia-se ainda mais, reforçando a visão de que, atualmente, a quase totalidade dos projetos de desenvolvimento de software constitui-se em processos complexos.

Teoria dos Jogos

O quinto fundamento é a Teoria dos Jogos, que estuda e analisa o CAS (Complex Adaptive System).

“CAS é um sistema formado por uma rede dinâmica de agentes (os quais podem ser células, espécies, indivíduos, firmas, nações, etc.) atuando em paralelo, agindo e reagindo constantemente ao que os outros agentes estão fazendo. O controle de um CAS tende a ser altamente disperso e descentralizado. Se há alguma coerência no comportamento do sistema, ele emerge da competição e cooperação dos agentes. O comportamento geral do sistema é resultado de um grande número de decisões tomadas a cada momento por cada indivíduo”

Fonte: Wikipédia

Aplicações da Teoria dos Jogos tentam encontrar equilíbrio nestes jogos, como o “equilíbrio de Nash” em que nenhum jogador pode melhorar a sua situação dada a estratégia seguida pelo jogador adversário. Agilistas consideram projetos de software como um CAS!

Teoria das Restrições

O sexto fundamento a ser estudado é a Teoria das Restrições (TOC). TOC é uma filosofia global de gerenciamento empresarial para contínua otimização do desempenho de uma organização através de ações nos elementos que a restringem.

A analogia é comparar a empresa com uma **corrente**. Se tracionarmos uma corrente, onde ela quebrará? No seu elo mais fraco, sua restrição. A restrição é que define o ganho máximo. Se quisermos aumentar o desempenho do sistema, precisamos explorá-la. Se aumentarmos a resistência de qualquer outro elo que não o mais fraco, não estaremos melhorando o desempenho da corrente como um todo.

“Usando esse processo podemos enfocar nossos esforços nos poucos pontos de um sistema que determinam seu desempenho (nas suas restrições), e assim podemos melhorar significativamente no curto prazo “

Eliyahu Goldratt – A Meta

Unidade II - Métodos e Frameworks Ágeis

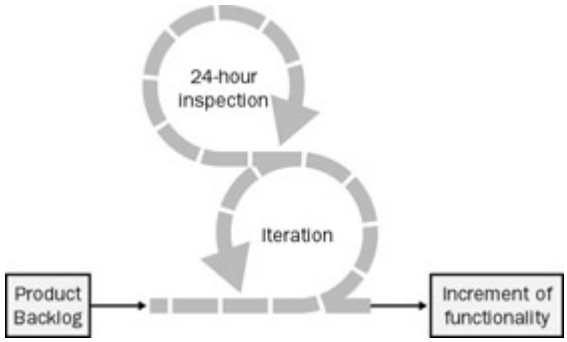
Scrum

O Scrum auxilia a controlar projetos de desenvolvimento de software, porém isto não significa que o projeto irá exatamente como esperado, produzindo resultados idênticos àqueles esperados. Ele controla o processo de desenvolvimento de software para guiar o trabalho em direção ao resultado de maior valor possível.

Seguem abaixo alguns tópicos para nos auxiliar a entender o que o Scrum é:

- Abordagem iterativa e incremental para o gerenciamento de projetos de software.
- Processo empírico para gerenciamento de projetos complexos nos quais é impossível predizer tudo que irá ocorrer.
- Processo criado para ampliar a probabilidade de desenvolver software com sucesso.
- Processo de gerenciamento de projetos de software que permite antecipar e avaliar com maior precisão o retorno do investimento.
- Oferece um arcabouço (framework) e um conjunto de práticas que mantêm tudo visível.
- Permite aos seus praticantes saber exatamente o que está acontecendo e realizar ajustes locais para manter o projeto movendo na direção de seus objetivos.
- Permite exercitar constantemente o bom-senso.

O Scrum sustenta todas as suas práticas em um esqueleto de processo iterativo e incremental.



Fonte: [1]

O ciclo inferior representa uma iteração de atividades de desenvolvimento, que ocorrem em sequência, tendo como saída um incremento de produto. O ciclo superior representa a inspeção diária que ocorre durante a iteração, na qual os membros da equipe se reúnem para inspecionar as atividades e encaminhar as adaptações necessárias. O que direciona a iteração é uma lista de requisitos. O ciclo é repetido até que o projeto não tenha mais fundos.

O processo se desenvolve da seguinte forma: no início da iteração, a equipe revisa o que precisa ser feito. Em seguida, seleciona aquilo que considera um incremento entregável de produto para constituir-se no objetivo da iteração. A equipe então foca no trabalho para produção do objetivo acordado; durante a iteração não há alteração de prazo ou de objetivo. Ao final da iteração, a equipe apresenta o incremento de software

produzido, para que possa ser avaliado pelas partes interessadas do projeto e, se necessário, adaptações necessárias no projeto possam ser realizadas.

A produtividade do Scrum reside no processo criativo que se realiza dentro de cada iteração. A equipe é responsável por organizar-se em torno do objetivo da iteração, analisando questões tecnológicas, arquiteturais e de requisitos, conforme as competências e habilidades de seus membros.

Extreme Programming – XP

Extreme Programming (XP) é um método ágil de desenvolvimento de software bastante conhecido. Lançado em 1999 por Kent Beck, ele enfatiza colaboração, criação rápida e antecipada de software e novas práticas de desenvolvimento. Está fundamentada em quatro valores: **comunicação, simplicidade, feedback e coragem**. Na segunda edição do seu livro-base *Extreme Programming Explained: Embrace Change*, de 2005 [10], Kent Beck adicionou um novo valor, que sustenta todos os demais: o **respeito.**

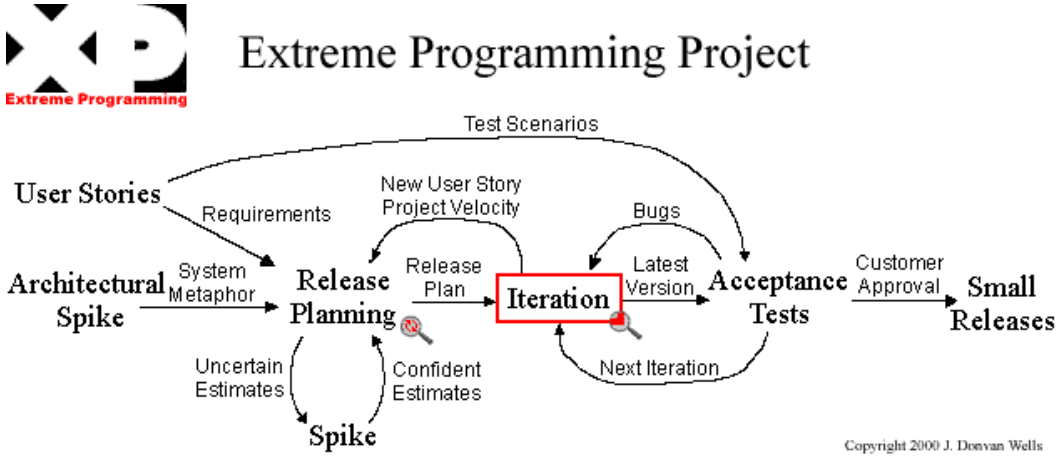
O XP encara o desenvolvimento de software em uma ótima pragmática e, ao mesmo tempo, voltada para o negócio. Isto se materializa na forte busca da satisfação do cliente, seja antecipando entregas de software operacional e com qualidade, seja respondendo de forma adaptativa a requisitos em mudança, mesmo em fases avançadas do ciclo de vida do projeto.

A proposta do XP para melhoria de projetos de software parte dos valores já mencionados:

- ✓ Comunicação - programadores comunicam sempre com seus clientes e colegas de projeto.
- ✓ Simplicidade - o design do software é mantido simples e limpo.
- ✓ Feedback - o feedback é obtido a partir dos testes que são realizados já no primeiro dia do desenvolvimento e das entregas feitas aos clientes o mais rapidamente possível.
- ✓ Respeito - cada contribuição individual da equipe é valorizada e respeitada como parte importante do sucesso da equipe.
- ✓ Coragem - programadores podem responder com coragem frente a requisitos em mudança e aos desafios das tecnologias.

Devido à sua forte aderência à simplicidade e ao foco em satisfazer as necessidades reais do cliente, XP também é conhecido como a "arte de maximizar a quantidade de software que não precisará ser feito". De fato, entender a real necessidade do cliente e focar no desenvolvimento que atenderá aos seus objetivos permite que as energias sejam melhor direcionadas e sejam evitadas várias fontes de desperdício.

O diagrama abaixo apresenta as regras do XP funcionando conjuntamente:



Fonte: <http://www.extremeprogramming.org/>

Principais práticas do XP:

- ✓ Releases pequenas e frequentes: O desenvolvimento é feito em iterações semanais. No início da semana, desenvolvedores e cliente reúnem-se para priorizar as funcionalidades. O cliente identifica prioridades e os desenvolvedores as estimam. Como o escopo é reavaliado semanalmente, o projeto é regido por um contrato de escopo negociável. Ao final de cada semana, o cliente recebe novas funcionalidades, completamente testadas e prontas para serem postas em produção. Além disso, ocorre a liberação de pequenas versões funcionais do projeto auxilia muito no processo de aceitação por parte do cliente, que já pode testar uma parte do sistema que está comprando.

- ✓ Utilização de metáforas para o sistema: Procura facilitar a comunicação com o cliente se adaptando a realidade dele. Conceitos são alinhados através desta técnica. É preciso traduzir as palavras do cliente para o significado que ele espera dentro do projeto.

- ✓ Design simples: Tem como foco construir códigos fáceis de se entender e de serem alterados. Significa manter o design simples desde o início e melhorá-lo continuamente, ao invés de tentar deixá-lo perfeito desde o início e então congelá-lo.

- ✓ Testes: Composto de testes construídos pelo cliente em conjunto de analistas e testadores, para aceitar um determinado requisito do sistema. E TDD: Primeiro crie os testes automatizados e depois crie o código para que os testes funcionem. Esta abordagem é complexa no início, pois vai contra o processo de desenvolvimento de muitos anos. Testes de regressão, por exemplo, são essenciais para que a qualidade do projeto seja mantida -> Maior segurança para se efetuar refatorações.

- ✓ Refactoring frequente: É um processo que permite a melhoria continua da programação, com o mínimo de introdução de erros e mantendo a compatibilidade com o código já existente. Refatorar melhora a clareza do código, divide-o em módulos mais coesos e de maior reaproveitamento, evitando a duplicação de código-fonte -> maximização do reuso

- ✓ Propriedade do código compartilhada: O código fonte não tem dono e ninguém precisa solicitar permissão para poder modificar o mesmo. O objetivo com isto é fazer a equipe conhecer todas as partes do sistema.

- ✓ Programação em pares: A dupla é formada por um iniciante na linguagem/negócio e outra pessoa funcionando como um instrutor. Como é apenas um computador, o novato é que fica à frente fazendo a codificação, e o instrutor acompanha ajudando a desenvolver suas habilidades (o contrário também é interessante). Desta forma o programa sempre é revisto por duas pessoas, evitando e diminuindo assim a possibilidade de erros. Com isto busca-se sempre a evolução da equipe, melhorando a qualidade do código fonte gerado.

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

- ✓ Ritmo sustentável: Trabalhar com qualidade, buscando ter ritmo de trabalho saudável, sem horas extras. Há uma busca constante de trabalho motivado. Para isto o ambiente de trabalho e a motivação da equipe devem estar sempre em harmonia.

- ✓ Time trabalhando junto: A equipe de desenvolvimento é formada pelo cliente e pela equipe de desenvolvimento. Comunicação!

- ✓ Padrões de codificação: A equipe de desenvolvimento precisa estabelecer regras para programar e todos devem seguir estas regras.

Kanban

É um processo de gerenciamento de projetos ou desenvolvimento de software. Introduz mudanças no ciclo ao mapear fluxo de valor e utilizar WIP (Work in Progress) - atividades em andamento -> algo novo só deve ser iniciado quando uma peca é liberada, por exemplo. Esta prática traz previsibilidade de tempo em ciclos. Abaixo, um exemplo de quadro de acompanhamento de atividades Kanban com os valores WIP indicados em cada fase do desenvolvimento de software.

Encorajamento de melhoria contínua:

Diminuição de inventário em função do WIP e de parar para concentrar toda a equipe na solução do problema e desbloquear o item restaurando o fluxo. Com isso, consegue atingir níveis altos de qualidade e queda de retrabalho.

Por ter também características de reuniões curtas, pode tomar decisões mais tardiamente sobre a próxima atividade a ser trabalhada utilizando o conceito de sistemas pull (puxam o sistema a medida da capacidade

da equipe e não é empurrado como em um cronograma comum), priorizando trabalho novo na entrega de trabalho existente. Apos cada iteração e demonstração de software entregável, pode ocorrer atualização de prioridade de execução. No Kanban, iterações de duração fixa não são prescritas. Pode-se escolher quando planejar, melhorar processo e entregar. As atividades podem ter numa periodicidade regular ou por demanda.

Transparência:

Através do quadro, expõe métricas de velocidade, gargalos, filas, variabilidade e desperdício. Com isso, incentiva discussões sobre melhorias e equipes as promovem rapidamente nos processos. Modifica o comportamento e incentiva maior colaboração no trabalho, promovendo consenso entre trabalhadores e colaboradores.

Kanban + Scrum

São ambos altamente adaptativos, sendo o Scrum mais prescritivo que Kanban. O Scrum traz mais restrições. Por exemplo:

- ✓ O Scrum prescreve o uso de iterações de duração fixa, enquanto o Kanban não.
- ✓ O Scrum Prevê equipes multifuncionais, não obrigatórias no Kanban.
- ✓ O Scrum prescreve a necessidade de 3 papéis – PO, Scrum Master e Scrum Team, o Kanban não determina seu time.

O Kanban tem como únicas restrições: Visualize Seu Fluxo de Trabalho e Limite Suas Atividades em Andamento.

O raciocínio comum tanto em Scrum quanto em Kanban é: menos é mais! Então, na dúvida, comece com menos!

Unidade III - O Framework Scrum

Uma Sprint é uma janela de tempo definida, na qual o Time cumprirá um objetivo determinado ao seu início, entregando um incremento de produto que possa ser apresentado e avaliado ao final deste período.

Considerando que as organizações e as pessoas possuem recursos limitados, é sempre importante identificar as prioridades de forma a canalizar para elas o foco de atenção dos recursos disponíveis. A Sprint, como janela de tempo relativamente curta e com objetivos determinados, possui a virtude de criar um contexto de canalização de energias para os itens selecionados como prioritários a partir do Backlog do Produto.

As primeiras Sprints geralmente concentram um volume maior de atividades de arquitetura e infraestrutura do sistema.

Papéis do Scrum

Há somente três papéis no Scrum: o Product Owner, a Equipe (ou Development Team) e o Scrum Master. Todas as responsabilidades de gerenciamento estão divididas entre esses três papéis.

O Scrum faz uma clara distinção entre os que estão efetivamente comprometidos com o projeto - os que desempenham os três papéis descritos - e os que estão interessados ou possuem algum envolvimento. Os que são responsáveis devem possuir a autoridade para realizar o que for necessário para o sucesso do projeto, enquanto os que não são responsáveis não podem interferir desnecessariamente.

É importante notar que os papéis do Scrum são responsabilidades no processo de desenvolvimento de software, e não posições na organização.

O Scrum é estruturado para regularmente tornar o estado do projeto visível para os três gerentes: o Product Owner, o Scrum Master e o Time, de forma que estes possam rapidamente ajustar o projeto para que atinja da melhor forma possível os objetivos.

O quadro abaixo auxilia no entendimento de como o gerenciamento está dividido entre cada papel no Scrum:

Product Owner	Scrum Master	Time (Development Team)
Gerenciamento Macro (negócio)	Facilitação	Gerenciamento Micro (execução)

O Scrum Master

O Scrum Master é o "gerente de projeto" que lidera o projeto Scrum. Provê liderança, instrução e orientação. É o responsável por ensinar os demais como utilizar o processo Scrum para lidar com as dificuldades encontradas durante o projeto. Em Scrum, o termo "gerente de projeto" não é utilizado.

O Scrum Master é o responsável por assegurar que todos os envolvidos no projeto sigam as regras do Scrum. Se as regras não são seguidas, as pessoas perdem tempo pensando o que precisam fazer. Se as regras são debatidas a todo momento, o tempo é perdido enquanto todos aguardam por uma resolução.

Se alguém quiser alterar as regras de trabalho da Equipe, deve utilizar a reunião de análise retrospectiva da Sprint, momento específico para este tipo de discussão. Mudanças nas regras devem surgir da equipe, não do gerenciamento.

Nenhuma regra deve ser modificada enquanto o Scrum Master não entender que a equipe possui o entendimento, experiência e profundidade necessários para realizar alterações de valor nas regras.

As responsabilidades do Scrum Master podem ser sumarizadas nos seguintes itens:

- Remover as barreiras entre o desenvolvimento e o Product Owner, de forma que este possa direcionar diretamente o desenvolvimento.
- Ensinar o Product Owner como maximizar o ROI e atingir seus objetivos através do Scrum.
- Melhorar a vida do time de desenvolvimento, facilitando a criatividade e a delegação de poder.
- Ampliar a produtividade do time de desenvolvimento de todas as formas possíveis.

- Aprimorar as práticas de engenharia e ferramentas de forma que cada incremento de funcionalidade seja potencialmente entregável.
- Manter informação sobre o progresso do time visível e atualizada para todas as partes.

Não é papel do Scrum Master gerenciar o Time; ele tem que se auto-gerenciar, para poder constantemente ajustar seus métodos e maximizar as chances de sucesso.

O Time Scrum (Development Team)

De forma reversa ao observado nas práticas gerenciais tradicionais, o Scrum faz com que o próprio Time seja o responsável por gerenciar as atividades de desenvolvimento.

A Equipe, ou Time, é responsável por desenvolver as funcionalidades do sistema. No Scrum, as equipes são auto-gerenciáveis, auto-organizáveis e compostas de pessoas com diferentes níveis de habilidades e competências, trabalhando juntas em direção a uma meta comum. É responsabilidade da Equipe identificar como transformar o Product Backlog em incrementos de funcionalidade em uma iteração e gerenciar o próprio trabalho para alcançar este objetivo.

A Equipe é coletivamente responsável pelo sucesso de cada iteração e, consecutivamente, do projeto como um todo.

Um aspecto novo do Scrum e, a princípio, não muito intuitivo, é que a Equipe é responsável por realizar o próprio gerenciamento. Utilizando a analogia dos "porcos" (pessoas comprometidas) e das "galinhas" (pessoas envolvidas), todos os três papéis do Scrum relacionam-se nesta analogia aos porcos, porque estão integralmente comprometidos com o sucesso do projeto. Outras partes interessadas no projetos são galinhas, e não possuem autoridade direta sobre a execução do projeto.

O Product Owner

O Product Owner é o responsável por representar todos os interesses daqueles envolvidos com o projeto ou com o sistema que será produzido por meio deste. É responsabilidade do Product Owner obter a verba do projeto por meio da criação dos requisitos gerais do sistema, os objetivos de retorno do sistema (ROI) e os planos de entrega. O foco do Product Owner é o retorno do investimento (ROI).

A lista de requisitos é denominada Product Backlog. O Product Owner é responsável por utilizar o Product Backlog para assegurar que as funcionalidades de maior valor para o negócio são produzidas primeiro. Isso se realiza por meio do sequenciamento adequado do Product Backlog para a próxima iteração.

O Product Owner deve concentrar-se em priorizar aquelas funcionalidades que endereçarão os problemas de negócio mais críticos, bem como alavancar e viabilizar o desenvolvimento de releases de funcionalidades cujo benefício ultrapassa o custo do desenvolvimento. O papel do Product Owner é muito importante no fluxo do Scrum, porque sua atuação adequada e focada permite a construção de um elo forte com o negócio, na forma de resultados entregues rapidamente e comprovação antecipada do retorno do investimento.

Artefatos

Product Backlog

O Product Backlog é uma lista priorizada de requisitos funcionais e não-funcionais do projeto, com tempos estimados para transformá-los em funcionalidade do sistema. As estimativas são em dias e devem ser mais precisas para os itens mais altos em termos de prioridade. A lista evolui, modificando-se na medida em que as condições de negócio ou a tecnologia mudam.

"O Product Backlog é o coração do Scrum. É como tudo se inicia. O Product Backlog é basicamente uma lista priorizada de requisitos, ou histórias, ou funcionalidades ou qualquer outra coisa. Coisas que o cliente quer, descritas usando a terminologia do cliente." [4]

O Product Backlog pode ser composto de requisitos funcionais, requisitos não- funcionais e problemas, priorizados em ordem de importância para o negócio e dependências e então estimados. A precisão da estimativa depende da prioridade e granularidade do item, sendo que os itens de mais alta prioridade, que podem ser selecionados para a próxima Sprint, têm estimativa bem granular e precisa.

O Product Backlog deve conter, além dos requisitos do produto, também os requisitos do projeto, tais como a realização de auditorias da qualidade, preparação de ambiente de produção, etc. Segue abaixo um exemplo de backlog de produto.

Requisitos			**Dados**			
ID	**Área**	**Item**	**Impor-tância**	**Est. Inicial**	**Release**	**Sprint**
1	Interfaces	Prova de conceito do sistema de mensageria	10	2	1	1
2	Administração	Controle de usuários, permissões e auditoria	9	3	1	1
3	Produção	Visualizar programação da produção	9	5	1	1
4	Produção	Executar etapa de produção	8	8	1	2
5	Estoque	Alterar posição de material no estoque	7	5	1	2

Sprint Backlog

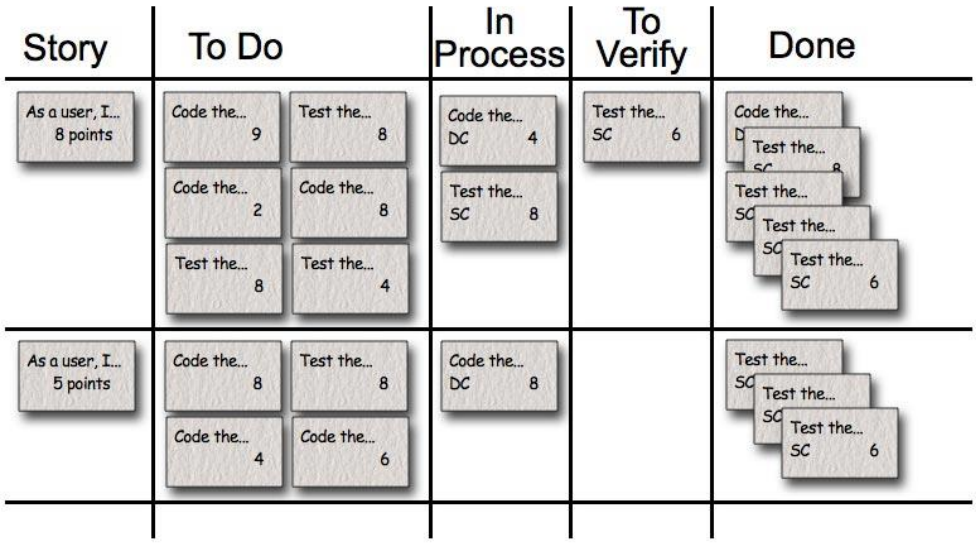
Pilha de tarefas (atividades), definida pelo time e para o time a partir dos itens de Selected Backlog, contendo detalhamento em 2-3 itens de Sprint Backlog, do que deve ser feito em um Sprint. Esta atividade é primariamente feita durante o Sprint Planning Meeting 2, sendo revisada diariamente durante o Daily Scrum.

- ✓ É utilizada para o Sprint Burndow Chart e não deve ser utilizada para gerenciamento externo (realizado por resultados).
- ✓ Unidade em HH Previsto (Tamanho x Velocidade Atual)

O Sprint Backlog é uma lista de tarefas que define o trabalho da Equipe para uma Sprint. A lista emerge durante o Sprint, sendo que uma primeira compilação é gerada na segunda parte da reunião de planejamento da Sprint. É responsabilidade da Equipe gerar e manter atualizado o Sprint Backlog. Cada tarefa identifica o responsável por executar o trabalho e o esforço remanescente para concluir a tarefa em um

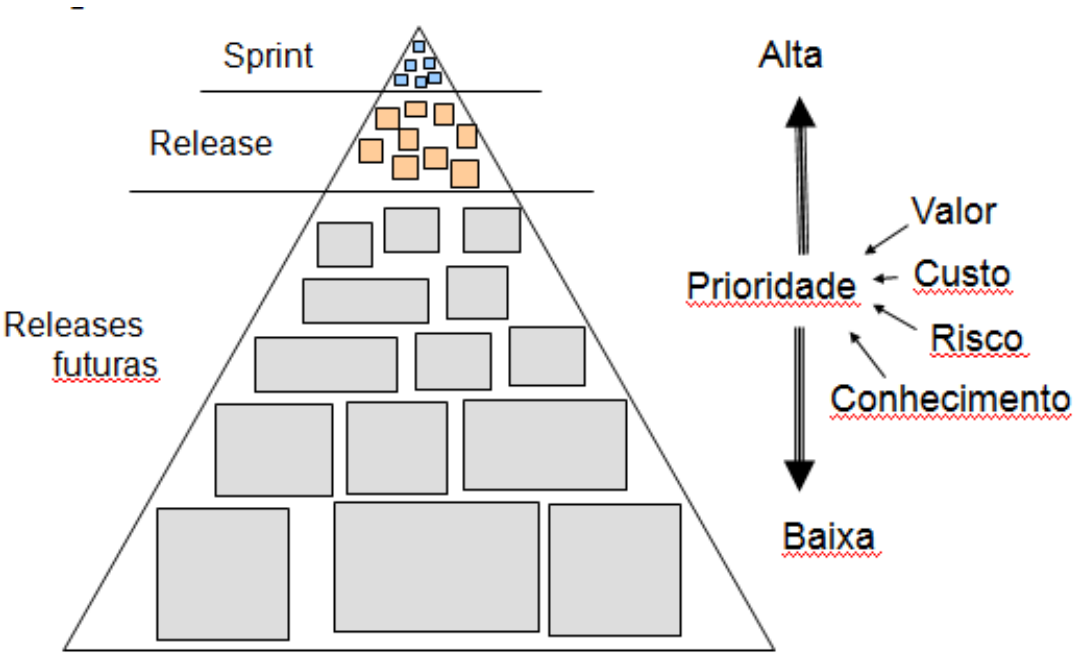
dia específico durante a Sprint. Tipicamente, uma tarefa possui estimativa entre 4 e 16 horas e tarefas com estimativa maior significam que ainda não foram suficientemente detalhadas.

Existem várias maneiras práticas de se trabalhar com um Sprint Backlog, desde planilhas Excel a sistemas de gestão baseados em Scrum. Um mecanismo bastante simples e difundido, no entanto, é a utilização de um quadro de tarefas, conhecido também como "Agile Radiator", conforme exemplo abaixo.



Durante o projeto, itens a serem trabalhados nos Sprints devem estar melhor definidos e possuem maior prioridade em relação aos demais. O desenho abaixo, confirma a granularidade maior que itens do Product Backlog devem ter quando estão planejados para serem executados durante o Sprint. Quando estão planejados em uma próxima release, podem ter uma granularidade menor e em releases futuras, menores ainda.

A priorização dos itens de um Product Backlog deve seguir premissas estratégicas definidas pela organização levando em consideração aspectos relativos a custo, dificuldade de implementação (conhecimento da tecnologia), risco, ROI (valor na entrega ao cliente), etc.



Exceto nos contextos em que um detalhamento arquitetônico e funcional mais profundo é necessário para se chegar em estimativas com grau de precisão maior - como é o caso, por exemplo, de propostas para RFPs de contratos de custo e prazo fixos - em Scrum dedica-se esforço moderado para as estimativas iniciais, dado que, devido à natureza complexa do projeto, existe grande chance de um esforço para detalhamento maior ser desperdiçado, pois mudanças sutis nas variáveis do problema podem promover grandes alterações na forma de endereçá-lo e, consequentemente, nas estimativas geradas inicialmente. Lembrar do conceito discutido acima – Cone da Incerteza.

No entanto, deve haver clareza e alinhamento entre o Product Owner, o Time e o Scrum Master acerca do que realmente está contemplado por esta estimativa inicial: todos os testes unitários, de integração, desempenho, etc., estão contemplados? Está previsto esforço para desenvolvimento de testes automatizados e *refactoring* do código? Este alinhamento é chamado **conceito de pronto**, e será muito importante ao longo de todo o ciclo do Scrum.

Eventualmente, pode ser necessário mais uma iteração para percorrer novamente os requisitos e refinar um pouco mais a estimativa, assegurando que a base de avaliação das mesmas está entendida e é compartilhada por todos.

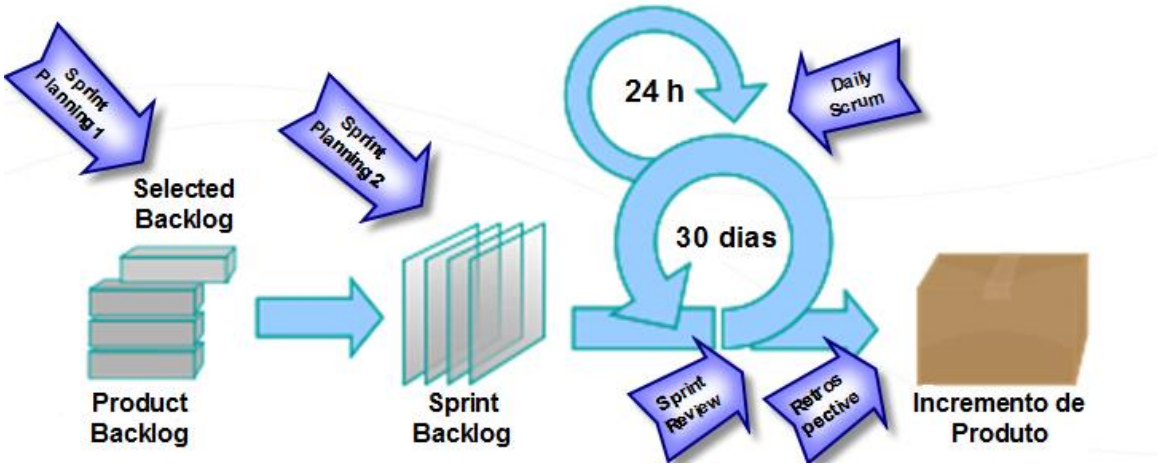
O planejamento do projeto deve evoluir conforme são vencidas as barreiras iniciais do projeto. Após a primeira Sprint, o planejamento pode ser ajustado para considerar com maior precisão a velocidade da equipe nas tecnologias envolvidas, questões ambientais da empresa, etc. Isso permitirá ajustes como a alteração da equipe, a revisão do escopo ou outras iniciativas que possam favorecer que o projeto atinja as expectativas das partes interessadas.

Incremento

O incremento é a soma de todos os itens do Backlog do Produto completados durante a sprint e o valor dos incrementos de todas as sprints anteriores. Ao final da sprint um novo incremento deve estar “Pronto”, o que significa que deve estar na condição utilizável e atender a definição de “Pronto” do time scrum. Este deve estar na condição utilizável independente do product owner decidir por liberá-lo realmente ou não.

Portanto, a cada Sprint, a equipe deve construir um incremento de produto de software que seja potencialmente entregável, ou seja, que possa a qualquer momento ser colocada em produção por determinação do Product Owner. Para isso, é necessário que o produto de cada Sprint seja um software executável, cujo código seja bem estruturado e bem escrito, tenha sido consistentemente testado e possua a documentação necessária.

Eventos



Sprint Planning Meeting 1

Reunião dedicada ao planejamento da Sprint, limitada a 8 horas de duração, sendo estas divididas em duas partes de até 4 horas cada: a primeira parte é dedicada à seleção dos itens do backlog do produto que o time irá se comprometer a entregar na Sprint, enquanto a segunda parte é dedicada à criação do Sprint Backlog. A primeira é conhecida como Sprint Planning 1 e a segunda como Sprint Planning 2.

Características principais:

- ✓ Realizada todo início de **Sprint**, com duração de 4 (quatro) horas
- ✓ Participação do Product Owner (PO), Scrum Master (SM) e Scrum Team. Alguns stakeholders podem ser convidados, mas devem atuar como "galinhas".
- ✓ Passagem de entendimento sobre cada item do **Selected Backlog** pré-selecionado do PO para o Scrum Team.
- ✓ O Scrum Team realiza a estimativa inicial de tamanho, para o cada item do Selected Backlog.
- ✓ O PO estabelece o **Sprint Goal.**

IMPORTANTE: O Sprint Goal é uma meta "visionária" do Sprint estabelecido pelo Product Owner ao final da reunião de Sprint Planning 1. É sempre verificado durante a reunião de Sprint Review, confrontado com a demonstração do Scrum Team e os itens de backlog (Selected Backlog) planejados e comprometidos.

A reunião de planejamento poderá ter uma duração menor, desde que isto seja acordado entre Product Owner, Scrum Team e Scrum Master, e se mantenha o limite acordado em todas as reuniões. A aplicação do conceito de etapas delimitadas por tempo (*"time-boxed"*) é muito importante no Scrum.

Nesta reunião, devem participar o Scrum Master, o Product Owner e o Time. Outras pessoas podem ser convidadas pontualmente, caso possam contribuir com algum detalhe tecnológico ou do domínio de aplicação correspondente.

O Product Backlog deve estar preparado antes da reunião. Esta preparação é muito importante para o correto desenvolvimento da reunião. Apesar do Scrum não determinar detalhadamente o que significa esta preparação (define apenas que deve ser uma lista de requisitos priorizada e estimada em alto nível), normalmente considera-se um backlog pronto quando:

- ✓ Os requisitos funcionais e não-funcionais foram organizados na forma de uma lista;
- ✓ A lista foi priorizada em virtude do retorno ao negócio (ROI);
- ✓ Os itens prioritários da lista (os candidatos a serem endereçados na próxima Sprint) foram descritos na forma de Estórias (*);
- ✓ As Estórias foram detalhadas até o nível de "teste de aceitação", ou seja, uma descrição um pouco mais detalhada do que se espera da funcionalidade, de forma que esta possa ser validada ao final.

(*) Uma Estória é uma descrição textual de um requisito, que deve responder às seguintes perguntas:

- ✓ Por quem - perfil de usuário de maior interesse na funcionalidade
- ✓ O que - descrição da funcionalidade esperada
- ✓ Por que - justificativa de negócio da funcionalidade

Na ausência do Product Owner, o Scrum Master deve se responsabilizar pelo preparo do backlog e pelo papel de Product Owner durante a reunião – prática não recomendada e não deve ocorrer de forma recorrente.

O Time pode fazer sugestões, mas a decisão final sobre os itens do Backlog a integrarem a Sprint é do Product Owner.

A reunião de planejamento da Sprint, na perspectiva do Time, é o momento onde começa o exercício de ser auto-gerenciável e auto-organizável, pois ninguém chegará com uma lista de tarefas a serem delegadas para a equipe. As estimativas iniciais dos requisitos no backlog do produto são revistas, ao serem abordados os detalhes de cada um com o Product Owner. A técnica de estimativa mais popular que pode ser utilizada durante esta reunião é conhecida como Planning Poker, explicada a seguir.

Planning Poker

O *"Planning Poker"* é uma técnica de estimativa baseada em consenso, criada a partir de um método denominado "Whideband Delphi", e que é utilizada para estimar o tamanho de requisitos em projetos de software. Este método vem sendo bastante utilizado, principalmente entre a comunidade ágil, como uma ferramenta importante para a produção de estimativas acordadas entre os membros da equipe, em um tempo significativamente reduzido, se comparado a discussões livres acerca do esforço de cada requisito.

O Planning Poker não é parte integrante do Scrum, porém frequentemente é utilizado durante as reuniões de planejamento da Sprint, para que a equipe elabore uma estimativa mais detalhada dos itens do backlog do produto, de forma a identificar quais requisitos será capaz de entregar durante a Sprint. O grande benefício do Planning Poker é que todos participantes têm que expressar simultaneamente sua estimativa de tamanho para determinado requisito, evitando que a opinião da maioria se ancore a partir da opinião dos primeiros a se manifestar.

Saídas da reunião de planejamento da Sprint:

- ✓ Uma meta para a Sprint – Sprint Goal
- ✓ Uma lista dos membros do Time comprometidos
- ✓ O Sprint Backlog
- ✓ Uma data definida para a demonstração da Sprint (Sprint Review Meeting)
- ✓ Um horário e local definido para a reunião diária (Daily Scrum)

Sprint Planning Meeting 2

É realizada após o Sprint Planning 1, tem duração de 4 (quatro) horas, sem a participação do Product Owner e stakeholders. Tem como características principais:

- ✓ Auto-organização: planejamento do Scrum Team para seu próprio trabalho: ele refina a análise de cada item do Selected Backlog, criando a lista de Sprint Backlog – tarefas que irão implementar cada item.
- ✓ O time finaliza a estimativa iniciada no Sprint Planning 1. A estimativa pode ser reavaliada durante a reunião (as discussões técnicas podem gerar alterações), reportando ao Product Owner qualquer ajuste necessário.
- ✓ O Agile Radiator deve então retratar o Sprint corrente.

Daily Scrum Meeting

A reunião diária do Scrum (Daily Scrum Meeting) é um rápido momento de reunião de todos os membros do Time, dedicado à resposta das três perguntas mencionadas anteriormente, visando exercitar os conceitos da inspeção e adaptação diariamente entre a equipe. Esta reunião tipicamente dura 15 minutos e acontece no

mesmo local e horário todos os dias. Realizá-la no primeiro momento do dia pode ser uma boa alternativa para convidar os membros do Time a refletir sobre o que realizou no dia anterior e o que fará no dia de hoje.

Uma das grandes virtudes da reunião diária (Daily Scrum) é a criação de uma plataforma de cooperação entre os membros do time em uma base diária.

Outro aspecto é a transparência que a Daily Scrum confere aos processos que estão atualmente em execução na organização, de forma que, caso haja algum processo organizacional ou outra interferência qualquer presente diariamente na reunião como um elemento impeditivo do avanço, então caberá uma ação para eliminar este impedimento. O Scrum Master terá, desta forma, os fundamentos necessários para acionar as engrenagens da empresa na busca do melhor contexto para a produtividade do Time.

Além disso, o Scrum Team deve promover a atualização do Agile Radiator com as informações levantadas durante esta reunião.

Sprint Review Meeting

- ✓ Realizada após cada Sprint, com duração de entre 2 a 4 horas e participação do Product Owner.
- ✓ O Scrum Team demonstra o trabalho realizado no Sprint e é avaliado pelo Product Owner.
- ✓ A demonstração deve ser do produto funcionando e comprovar que os itens de Selected Backlogs prioritários (e associado ao Sprint Goal) foram realizados com objetivo de maximizar o ROI.

O Time e o Product Owner devem estar em constante colaboração, definindo em conjunto como atingir o maior valor ao negócio a partir das tecnologias disponíveis. A reunião de apresentação da Sprint (Sprint Review Meeting) é um momento muito oportuno e conveniente para que haja forte interação face a face entre o Time e os stakeholders, em um espírito de colaboração que traz um valor muito maior do que qualquer relatório escrito, porque as ideias são trabalhadas em cima de algo real, que é o software produzido durante a Sprint.

Um aspecto importante da Sprint Review Meeting é que, ao criar uma oportunidade de colocar frente a frente, periodicamente, clientes e equipe de desenvolvimento, cria-se um ambiente de colaboração e uma maior percepção nos clientes de "resultados entregues" pela equipe de desenvolvimento. Em pouco tempo, o que foi priorizado na última reunião encontra-se entregue na forma de software real, e não de documentação intermediária que pouco ou nada interessa aos clientes. Este ambiente de colaboração, se bem conduzido pelo Product Owner, tende a formar um ciclo virtuoso entre equipe e clientes, ou ao menos diminuir os conflitos e reclamações de insatisfação por parte de clientes.

Apesar de haver, em casos especiais, Sprints que produzem artefatos intermediários como especificações ou ainda atividades de implantação, tipicamente a Sprint produz software. O objetivo da Sprint review é apresentar este software em funcionamento para todos os interessados que queiram e que seja conveniente que participem.

O Scrum Master tem um papel importante de assegurar a participação de todos os que devem participar.

Idealmente, o Product Owner já conhece o produto que está sendo desenvolvido e não se espera que haja surpresas na reunião de apresentação.

Sprint Retrospective Meeting

O objetivo da reunião retrospectiva da Sprint é promover reflexões e alavancar adaptações e melhorias na equipe nas práticas do Scrum. O funcionamento do processo na última Sprint é analisado e ajustado para funcionar melhor na próxima Sprint. Estas reuniões são limitadas ao tempo de duas horas cada.

- ✓ Realizada após o Sprint Review, com duração de 1-2 horas -> lições aprendidas.
- ✓ Timeline pode ser relembrado em grupo.
- ✓ 2 perguntas: "O que foi bem?" (WWW-What Went Well?) e "O que pode ser melhorado? (WCBI - What Can Be Improved?).
- ✓ Separação de responsabilidades pelo WCBI (organizacional e time) e atualização no Agile Radiator

Unidade IV - Gerenciamento Ágil com Scrum

Ser ágil diz respeito à habilidade em responder rápido às mudanças: de requisitos, prioridades, tecnologias e ferramentas, pessoas, complexidade de desenvolvimento de software, etc. Para isso, utiliza conceitos como iteratividade, técnicas incrementais, times multi-funcionais, auto-gerenciamento e auto-organização.

Características ágeis importantes e essenciais para o correto gerenciamento de projetos Scrum:

- ✓ Cliente sempre por perto e fazê-lo um participante ativo. É importante que ele entenda suas responsabilidades e sua grande parcela de contribuição para o sucesso do projeto.
- ✓ Iterações curtas levam ao *feedback* real e imediato dado pelo cliente, e como resultado, auxiliam nos possíveis ajustes que não acontecem mais tardiamente e garantem a entrega de software de valor.
- ✓ Atendimento ao goal x Controle da WBS
- ✓ A identificação e o gerenciamento dos riscos em metodologias ágeis é feita em taxas diárias (através de reuniões curtas e diárias) e durante toda a iteração
- ✓ O controle da qualidade dos trabalhos é avaliado continuamente através de, por exemplo, testes, revisão por pares, inspeção contínua e acompanhamento pelo cliente
- ✓ Impedimentos são levantados e estes devem ser resolvidos no prazo máximo de 24 horas para garantir que ações de resolução rápidas são executadas e para que todos conheçam os impedimentos que podem se tornar potenciais riscos para o projeto.
- ✓ Mudanças no projeto são bem aceitas, pois elas existem e irão acontecer. É por isso que o comprometimento de todos os envolvidos é tão importante em um projeto. Caso o cliente queira mudar algo que solicitou ou o mercado peça neste momento algo diferente, pode-se mudar o rumo rapidamente sem afetar todo o projeto, pois o planejamento é refeito a todo o momento.
- ✓ E se uma iteração não foi conforme o esperado pelo cliente, pode-se mudar a abordagem de levantamento de dados, os recursos envolvidos, a forma de troca de informações e corrigir o rumo ainda a tempo do final do projeto.

Estimativas Ágeis

Estimativa de Software é a disciplina da Engenharia de Software que trata da elaboração de estimativas de tamanho, esforço, prazo, custo e qualidade no desenvolvimento de software. Estes indicadores são utilizados

para correlatar contra os desempenhos observados no passado e então criar previsões de desempenho futuro.

Tamanho x Esforço:

O tamanho é “unitless” e é uma medida inicial que não leva em consideração o recurso que implementará a atividade, enquanto o esforço é dependente do(s) recurso(s) alocado(s) para desenvolvê-lo.

Imagine uma parede 1mx1m. Independente de quem a concebeu, ela continuará tendo o mesmo tamanho. Por outro lado, seu tempo de entrega (esforço), variará de acordo com o executor designado para tal. Esforço deriva de tamanho.

Exemplos de medidas de tamanho:

- Pontos de função
- Pontos de caso de uso
- Ideal Day
- Story Point, etc

Custo e prazo podem ser calculados a partir do esforço e qualidade de um conjunto maior além dos citados!!!

Técnicas de estimativas:

✓ Contar, Computar e Julgar: Consiste em contabilizar linhas de código, requisitos, casos de uso, paginas em um sistema web, funções, etc.

  – LOC, APF...

✓ Opinião do Especialista: Consiste em utilizar a opinião do especialista com base no seu conhecimento no domínio da questão. E podemos incrementar este método utilizando formulas que vão do mais realista ao otimistas e até levando em considerações margens de erro e outros métodos existentes como Cocomo.

✓ Decomposição e Recomposição: Dividir para conquistar. Um exemplo é a forma de construir a WBS. Consiste em quebrar o que deve ser entregue em pequenas unidades de trabalho.

✓ Estimativas por Analogia: Consiste em comparar comportamentos semelhantes. Quando você já fez um software antes e está fazendo algo similar com o anterior.

✓ Estimativas Baseadas em Proxy: É baseado em alguma escala e em um conjunto de informações. Como exemplos, Story Points, T-Shirt Sizing, entre outras.

✓ Julgamento de Especialistas em grupo: Consiste em realizar diversas estimativas através de um grupo de pessoas especialistas. Como exemplo, o Planning Poker do Scrum e o Wideband Delphi. Neste caso, utiliza-se ou as médias no caso do Wideband ou a votação no caso do Scrum.

Medidas de Tamanho Ágeis
Story Points

Baseia-se no tamanho da estória levando em consideração:

✓ Sua dificuldade e complexidade

- ✓ Conhecimento

São “unitless” e times diferentes podem ter story points diferentes para uma estória dado ponto de referência que escolherem.

Como principais técnicas para estimar:

- ✓ Opinião de especialista (muito comum em métodos ágeis)
- ✓ Analogia
- ✓ Quebra de estórias (menor granularidade)

É comum estimar com Story points utilizando a série de Fibonacci ou números múltiplos: 1 – 2 – 3 – 5 – 8 – 13 – 20 – 40 - 100. A sugestão a ser utilizada tem o propósito de ser não linear. Quanto menor o item, mais precisa a estimativa – por isso, utiliza-se quebra de estórias independente da medida ágil de tamanho utilizada. Os tamanhos 40 e 100 devem ser utilizados somente para itens de menor prioridade no Backlog, conhecidos como épicos.

Formas de iniciar a estimativa por Story Points:

- ✓ 1ª. – escolher uma estória que se espera ser a menor que será trabalhada e estimá-la em 1 SP.
- ✓ 2ª. – escolher uma estória considerada de tamanho médio e associar um valor de 5 SP (considerando range de 1 a 10 SP)
- ✓ Melhor maneira é tentar, experimentar!!

Ideal Day

Um Ideal Day corresponde à quantidade de trabalho que um profissional de nível sênior, com fluência nas tecnologias e ferramentas envolvidas (Ideal Developer) consegue realizar, em 08 (oito) horas de trabalho dedicadas (sem interrupções).

É importante que se compreenda que o "Dia Ideal", com 08 (oito) horas de trabalho sem interrupções, de um "desenvolvedor ideal", raramente ocorrerá na prática, e, portanto, deve ser utilizado unicamente como "moeda" estável para quantificação de tamanho de referência e balizador ideal de produtividade.

É uma estimativa empírica, executada por especialistas ("Expert Judment") para desenvolvimento com base em "exploração adaptativa". Segundo estudos mais recentes da escola ágil, a estimativa empírica é uma maneira sensata de se prever o tamanho de requisitos em uma dinâmica de "requisitos evolucionários", com práticas de "exploração e adaptação", especialmente se acompanhada por:

- ✓ Realimentação iterativa da "velocidade", a partir de dados históricos, preferencialmente, coletados durante o mesmo projeto, para a mesma equipe.
- ✓ Previsão sobre uma mesma "ordem de grandeza", neste caso que não ultrapasse o espaço de algumas horas para alguns poucos dias.

- ✓ Realização de consenso entre especialistas, com técnicas de comunicação e convergência como a do Planning Poker
- ✓ Utilização da técnica de PERT (Program Evaluation and Review Technique)
- ✓ Utilização de balanceamento como a técnica Cocomo (COnstructive COst Model)

Irão contribuir para que um "Ideal Day" não aconteça, na prática, em um dia típico:

- ✓ Natureza humana do desenvolvedor (comer, beber, alongar, socializar, sono, mal-estar eventual, etc.)
- ✓ Deficiências técnicas do desenvolvedor (desconhecimentos do assunto ou tecnologia específicos)
- ✓ Interrupções da empresa (reuniões administrativas, conversa com o 'chefe', ligações de clientes)
- ✓ Interrupções pessoais
- ✓ etc...

Dessa forma, a equipe deve ter sua velocidade medida pelo tempo gasto para se implementar um Ideal Day. Quanto menos tempo, maior a velocidade, e maior a produtividade da mesma. Na realidade, não é importante conhecer a velocidade individual e sim a média da equipe. Para se manter uma unidade, não é interessante expor se um integrante executa suas atividades em mais ou menos tempo. É uma dinâmica do grupo! Ele deve aprender como interagir melhor para a busca de entrega de maior retorno de valor para o cliente. Se for necessário utilizar de técnicas como pair-programming para agilizar o desenvolvimento e validação de um requisito, o time deve escolher este caminho. Também pode-se utilizar peer-review para verificações e validações, assumir outro papel (trocar de “chapéu” dentro da equipe), dentre outras práticas, para convergir para o objetivo definido

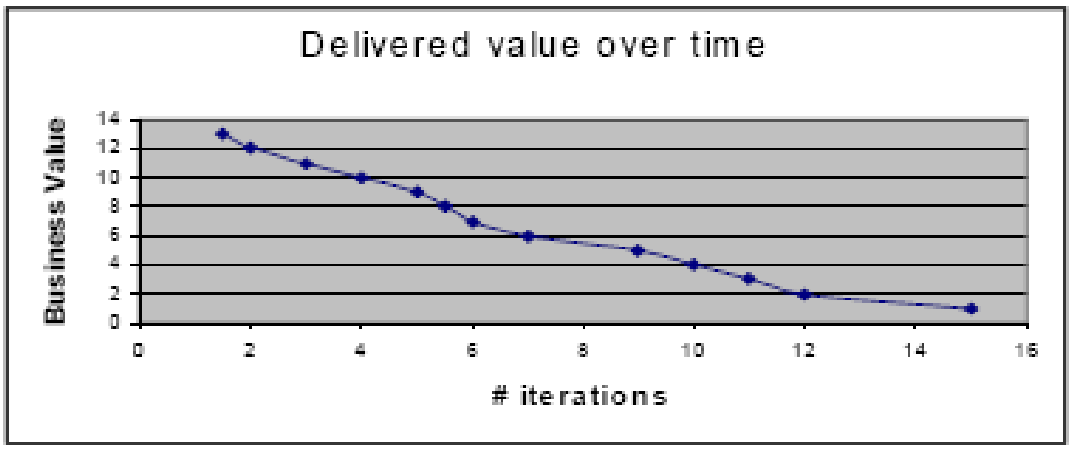
Priorização Ágil

Business Value (BV) ou Valor de Negócio reflete a importância estratégica de uma funcionalidade do produto para o mercado. O Product Owner deve manter a lista de Product Backlog constantemente atualizada e avaliada com seu business value atribuído, podendo ser revisado a qualquer momento.

1a. Etapa de Priorização
Esta prática faz parte da primeira etapa de priorização, que permite ao Product Owner classificar os itens de maior valor para o mercado e obtenha maior retorno para o negócio sobre o investimento. Ao iniciar uma Release, será possível identificar e selecionar os possíveis itens do Product Backlog que farão parte do escopo a ser desenvolvido. É interessante estipular um valor limite para a distribuição entre os itens da pilha. Cada funcionalidade deve ter seu valor compreendido na escala estabelecida pela organização, por exemplo, de 0 a 100.

O gráfico abaixo exibe a quantidade de BVs entregues ao longo dos Sprints de um Release. Este valor deve ser sempre maior nas primeiras iterações!

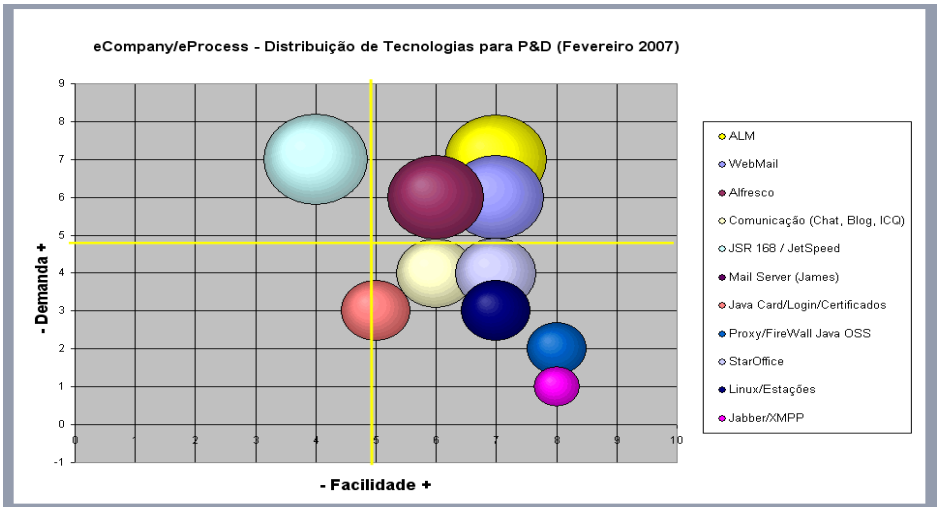
- 80% do valor de um software vem de 20% das funcionalidades - PARETO
- 60% das funcionalidades entregues em projetos de sucesso são raramente ou nunca utilizadas.



2a. Etapa de Priorização

Outra variável que deve ser inserida na fórmula de priorização dos itens que irão compor o escopo da Release (Release/Selected Backlog) é a facilidade de implementação do requisito, executando assim a segunda etapa de priorização. Quanto maior a facilidade, menor deve ser seu esforço de implementação.

Um gráfico de quatro quadrantes de itens do escopo que foram classificados em função seu BV e sua facilidade de implementação. Itens que se concentram no quadrante superior direito são os que devem ser priorizados e, seguramente, são os que mais representam a maximização do resultado.



Fonte: http://www.powerlogic.com.br

As coordenadas do gráfico seguem os valores de BV determinados pelo responsável do produto e a facilidade de implementação estimada em consenso pela equipe de desenvolvimento. O tamanho de cada bolha representa proporção de valor de negócio de cada item considerando a lista em que ele está inserido. Caso seja necessário, cabe a cada organização determinar um diferente “peso” para esta representação. Portanto, esta etapa organiza a ordem de execução dos requisitos da pilha selecionados e norteia a equipe de desenvolvimento.

Dado isso, a fórmula sugerida deve ser a seguinte:

- ✓ Priorização final da pilha = BV/ Tamanho do requisito

3a. Etapa de Priorização
Como critério de desempate, a criticidade pode ser considerada, uma vez que ela determina o momento atual e pode ser utilizada como "alteração manual" dentre a pilha de requisitos:

- 1 – Baixa
- 3 – Normal
- 5 – Alta
- 7 – Urgência
- 9 – Emergência

A fórmula final sugerida de priorização para desempate deve ser representada como:

- (BV/Tamanho do requisito) * criticidade.

Item de Product Backlog			
Tamanho (Em Ideal Day ou Story Point) Ex.: 20	Retorno para Negócio (Em BV) Ex.: 200	Prioridade Calculada por Fórmula BV / ID Ex.: 10	Ordem Ajustada diante da criticidade "Alta" Ex.: 10 * 5

Outro ponto a ser destacado e comum em projetos de manutenção de produtos, é a presença de erros que deverão entrar na pilha do Product Backlog. Pode-se dar um "peso" e modificar a fórmula acima para contemplar este cenário. Uma vez que BV diz respeito a valor de negócio que a inclusão da funcionalidade irá prover para o mercado e erro não acrescenta valor ao produto, sugerimos determinar BV negativo para este caso. Para que este requisito entre corretamente na priorização da pilha, deve-se aplicar seu valor absoluto.

A partir deste resultado, a equipe de desenvolvimento consegue determinar prazos para cada entrega que contempla o escopo acordado. O número de Sprints de implementação necessários é então comunicado ao responsável pelo produto. A partir da definição de sua restrição – escopo, tempo, custo ou qualidade - ele tomará uma decisão e definirá o objetivo maior da Release e do primeiro Sprint. O Scrum prega o planejamento contínuo, portanto, as demais iterações serão planejadas oportunamente.

Após esta definição, a equipe de desenvolvimento consegue implementar os requisitos seguindo, rigorosamente, do topo para baixo da pilha.

Em sistema enxutos (Lean) de "push", como o Scrum, resolve-se o problema de dependências de "caminho crítico" simplesmente ajustando-se a ordem de execução dos itens de Product Backlog: os que dependem são colocados abaixo de suas dependências.