SIN 5005 – TÓPICOS EM ENGENHARIA DE SOFTWARE

AULA 02 – MÉTODOS ÁGEIS

Prof. Marcelo Medeiros Eler
marceloeler@usp.br

Objetivos

- Apresentar os valores e os princípios difundidos pelo Manifesto Ágil e pela Aliança Ágil
- Apresentar o método de desenvolvimento XP

Introdução

- Características dos métodos tradicionais de desenvolvimento
  - Planejamento rigoroso
  - Extensa documentação
  - Etapas e papéis bem definidos
  - Pontos de controle rígidos
  - Foco na documentação e no processo
  - Antecipação de todos os detalhes de desenvolvimento

Introdução

- Críticas aos métodos tradicionais
  - Divisão distinta de fases no projeto gera inflexibilidade uma vez que raramente os projetos seguem um fluxo sequencial.
  - Requisitos totalmente especificados e “congelados” na primeira fase do projeto dificultam futuras mudanças.
  - Arquitetura especificada e “congelada” na segunda fase do projeto torna a arquitetura pouco confiável diante de possíveis mudanças de requisitos.
  - Grande dificuldade de alterações no projeto depois de decisões já tomadas.
  - A excessiva documentação torna o processo “pesado”
  - O excessivo planejamento “aprisiona” os envolvidos no desenvolvimento

Métodos ágeis

- Os modelos alternativos aos processos tradicionais de desenvolvimento são baseados em “abraçar” a mudança como um fato inevitável da vida: desenvolvedores deveriam continuamente refinar protótipos com o *feedback* dos clientes a cada iteração até que eles estejam satisfeitos com o resultado.
- O ciclo de vida ágil é tão rápido que novas versões do software são disponibilizadas a cada uma ou duas semanas, ou até mesmo todos os dias.

Métodos ágeis

- Métodos mais modernos de desenvolvimento foram criados já em meados de 1980 (Scrum - 1986, Crystal Clear – 1988, Adaptative Software Development - 1995, Feature Driven Development - 1995, Dynamic Systems Development Method – 1995, XP - 1996).
- Mas um dos pontos de maior ruptura com os métodos tradicionais foi o chamado “Manifesto Ágil”, uma reunião realizada entre 17 gurus da comunidade de desenvolvimento para buscar alternativas aos processos de desenvolvimento tradicionais

Manifesto Ágil

- Conteúdo do manifesto:
  - *Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:*
    - ***Indivíduos e interações*** *mais que processos e ferramentas*
    - ***Software em funcionamento*** *mais que documentação abrangente*
    - ***Colaboração com o (e do) cliente*** *mais que negociação de contratos*
    - ***Responder a mudanças*** *mais do que seguir um plano*
  - *Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.*

Manifesto Ágil

- Signatários do manifesto:

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Ron Jeffries
Jon KernBrian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas
Andrew Hunt
Scott Ambler

12 Princípios Ágeis

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.

12 Princípios Ágeis

5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.

12 Princípios Ágeis

9. Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10. Simplicidade, a arte de maximizar a quantidade de trabalho (desnecessário) não realizado, é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Métodos ágeis

- Duas abordagens de desenvolvimento serão enfatizados nesta apresentação:
  - eXtreme Programming (XP)
  - Scrum
  - Kanban

eXtreme Programming (XP)

- Surgiu nos EUA em 1996 e foi criado por Kent Back
- Kent Back é formado na Universidade de Oregon e atualmente trabalha no Facebook, Iterate AB e Lifeware
- “Uma metodologia leve para equipes pequenas e médias que desenvolvem software com requisitos vagos ou que mudam rapidamente” (Kent Beck)
- É um processo ágil:
  - Incremental: versões pequenas do software, com ciclos rápidos
  - Cooperativo: clientes e desenvolvedores trabalham constantemente juntos
  - Direto: o próprio método é fácil de aprender e modificar, bem documentado
  - Adaptativo: capaz de realizar mudanças a qualquer momento do desenvolvimento do software.

eXtreme Programming (XP)

- Filosofia: Levar todas as boas práticas ao Extremo
  - Se testar é bom, vamos testar toda hora!!
  - Se projetar é bom, vamos fazer disso parte do trabalho diário de cada pessoa!
  - Se integrar é bom, vamos integrar a maior quantidade de vezes possível!
  - Se iterações curtas é bom, vamos deixar as iterações realmente curtas!

eXtreme Programming (XP)

- Componentes:
  - Valores
  - Princípios
  - Práticas

eXtreme Programming (XP)

- Valores
  - Simplicidade
  - Respeito
  - Comunicação
  - Feedback
  - Coragem

eXtreme Programming (XP)

- Valores – Simplicidade
  - Segundo o Chaos Report, mais da metade das funcionalidades introduzidas em sistemas nunca é usada
  - A equipe deve concentrar-se nas funcionalidades efetivamente necessárias e não naquelas que poderiam ser necessárias (sem previsões), mas de cuja necessidade não se tem evidência
  - Pensar sempre : “O que pode ser feito que seja o mais simples que funciona?”
  - Comece por soluções simples e que funcionem.
  - Melhorias são adicionadas depois.
  - “O ótimo é inimigo do bom”.

eXtreme Programming (XP)

- Valores – Respeito
  - Respeito entre os membros da equipe, assim como entre a equipe e o cliente, é um valor que dá sustentação a todos os outros
  - Se não há respeito, não há comunicação.
  - Pessoas são valorizadas e ninguém é ignorado.

eXtreme Programming (XP)

- Valores – Comunicação
  - A comunicação é essencial para que o cliente consiga dizer aquilo de que realmente precisa
  - A preferência é para a encontros presenciais em vez de videoconferências, videoconferências em vez de telefonemas, telefonemas em vez de e-mails, e assim por diante.
  - Quanto mais pessoal e expressiva a forma de comunicação, melhor.
  - A comunicação diária e informal entre a equipe é essencial.

eXtreme Programing (XP)

- Valores – Feedback
  - Os desenvolvedores devem sempre buscar o feedback rápido para evitar eventuais falhas de comunicação com o cliente e com a equipe
  - Os feedbacks permitem maior agilidade:
    - Erros detectados e corrigidos imediatamente
    - Requisitos e prazos reavaliados mais cedo
    - Permite estimativas mais precisas

eXtreme Programing (XP)

- Valores: Coragem
  - Ter coragem de abraçar as inevitáveis modificações em vez de ignorá-las por estar fora do planejamento inicial ou por ser difícil de ser acomodada
  - Não deixe para amanha o que pode ser feito agora!
  - Não ficou bom? Refaça o quanto necessário!
  - Se não precisa, jogue fora!
  - Pedir ajuda aos que sabem mais.

eXtreme Programming (XP)

- Princípios:
  - Feedback rápido
  - Presumir simplicidade
  - Mudanças incrementais
  - Abraçar mudanças
  - Trabalho de alta qualidade

eXtreme Programming (XP)

- Princípios - Feedback rápido
  - O retorno entre os desenvolvedores é rápido
    - Cliente sabe se o produto que está sendo desenvolvido atende às suas necessidades
  - Modele um pouco, mostre ao cliente e então modele novamente.
    - Garante que o seu modelo será preciso enquanto seu conhecimento do projeto aumenta

eXtreme Programming (XP)

- Princípios - Presumir simplicidade
  - Deixe o seu modelo tão simples quanto possível e assuma que a solução mais simples é a melhor
  - O design do sistema deve ser feito para a iteração corrente. Não deve ser feito design sobre uma possível necessidade futura.

eXtreme Programming (XP)

- Princípios - Mudanças incrementais
  - O modelo não será perfeito na primeira tentativa, ele irá mudar de acordo com o desenvolvimento do projeto
  - Os problemas devem ser solucionados com um conjunto de pequenas modificações

eXtreme Programming (XP)

- Princípios - Abraçar mudanças
  - Mudanças ocorrerão no projeto de acordo com o crescimento do entendimento do mesmo
  - Aceite as mudanças e tenha coragem para reconstruir

eXtreme Programming (XP)

- Princípios - Trabalho de alta qualidade
  - A qualidade do trabalho nunca deve ser comprometida
  - XP eleva a importância da codificação e do teste antes da programação – test-first programming, test-driven development

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Jogo de planejamento
  - Semanalmente, a equipe deve se reunir com o cliente para priorizar as funcionalidades a serem desenvolvidas. Cabe ao cliente identificar as principais necessidades e à equipe de desenvolvimento estimar quais podem ser implementados por ciclo semanal que se inicia.
  - Ao final da semana, essas funcionalidades são entregues ao cliente. Este tipo de modelo de relacionamento com o cliente é adaptativo, em oposição aos contratos rígidos usualmente estabelecidos.

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas – Metáfora
  - É preciso conhecer a linguagem do cliente e seus significados.
  - A equipe deve aprender a se comunicar com o cliente na linguagem que ele compreende.
  - Usa-se analogias com algum outro sistema (computacional, natural, abstrato) que facilite a comunicação entre os membros da equipe e cliente
  - Facilita a escolha dos nomes de métodos, classes, campos de dados, etc.
  - Serve de base para estabelecimento de padrões de codificação

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Equipe coesa
  - A equipe deve ter integrantes para os diversos papéis:
    - programadores
    - testadores (que ajudam o cliente com testes de aceitação)
    - analistas (que ajudam o cliente a definir requerimentos)
    - gerente (garante os recursos necessários)
    - coach (orienta a equipe, controla a aplicação de XP)
    - tracker (coleta métricas)
  - O cliente também faz parte da equipe de desenvolvimento e a equipe deve ser estruturada de forma que eventuais barreiras de comunicação sejam quebradas

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Reuniões em pé
  - Como no Scrum, reuniões em pé tendem a ser mais objetivas e efetivas
  - As reuniões em pé tendem a durar menos tempo e a ser mais informais, facilitando a comunicação entre os membros da equipe

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Design simples
  - Implica atender a funcionalidades solicitadas pelo cliente sem sofisticar desnecessariamente.
  - Deve-se fazer aquilo de que o cliente precisa, não o que o desenvolvedor gostaria que ele precisasse.
  - Por vezes, design simples é confundido com design fácil. Nem sempre o design mais simples é o mais fácil de ser obtido e implementado. O design fácil pode não atender às necessidades ou então gerar problemas de arquitetura.
  - Não é permitido que se implemente nenhuma função adicional que não será usada na atual iteração
  - Implementação ideal é aquela que
    - Roda todos os testes
    - Expressa todas as ideias que você deseja expressar
    - Não contém código duplicado e tem o mínimo de classes e métodos

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Versões pequenas
  - A liberação de pequenas versões do sistema pode ajudar o cliente a testar as funcionalidades de forma contínua.
  - Esta prática é levada ao extremo quando versões do software podem ser liberadas todos os dias e às vezes mais do que uma vez por dia.
  - Disponibiliza, a cada iteração, software 100% funcional:
    - Benefícios do desenvolvimento disponíveis imediatamente
    - Menor risco (se o projeto não terminar, parte existe e funciona)
    - Cliente pode medir com precisão quanto já foi feito
    - Feedback do cliente permitirá que problemas sejam detectados cedo e facilita a comunicação entre o cliente e os desenvolvedores
  - As novas versões podem ser destinadas a
    - usuário-cliente (que pode testá-lo, avaliá-lo, oferecer feedback)
    - usuário-final (sempre que possível)

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável (ou saudável)
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Ritmo sustentável (ou ritmo saudável)
  - Trabalhar com qualidade um número razoável de horas por dia (não mais do que 8 horas).
  - Horas extras só são recomendadas quando efetivamente trouxerem aumento de produtividade mas não podem ser rotina.
  - Semanas de 40 horas de trabalho.
  - Projetos com cronogramas apertados que sugam todas as energias dos programadores não são projetos XP.
    - "Semanas de 80 horas" levam à baixa produtividade
    - Produtividade baixa leva a código ruim, relaxamento da disciplina (testes, refactoring, simplicidade), dificulta a comunicação, aumenta a irritação e o stress da equipe
    - O tempo "ganho" será perdido depois

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Posse coletiva
  - O código não tem dono e não é necessário pedir permissão para modifica-lo
  - Todo o código em XP pertence a um único dono: a equipe
  - Todo o código recebe a atenção de todos os participantes resultando em
    - maior comunicação
    - Maior qualidade (menos duplicação, maior coesão)
    - Menos riscos e menos dependência de indivíduos
  - Todos compartilham a responsabilidade pelas alterações

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Programação em pares
  - A programação é sempre feita por duas pessoas em cada computador. Em geral um programador mais experiente e um aprendiz.
  - O aprendiz deve usar a máquina, enquanto o mais experiente deve ajuda-lo a evoluir em suas capacidades.
  - Com isso, o código gerado terá sempre sido verificado por pelo menos duas pessoas, reduzindo drasticamente a possibilidade de erros.
  - Existem sugestões também de que a programação em pares seja feita por desenvolvedores de mesmo nível de conhecimento, os quais devem se alterar no uso do computador.
  - Benefícios:
    - melhor qualidade do design, código e testes
    - Revisão constante do código
    - Nivelamento da equipe
    - Maior comunicação

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Padrões de codificação
  - A equipe deve estabelecer e seguir padrões de codificação, de forma que o código pareça ter sido desenvolvido pela mesma pessoa, mesmo que tenha sido feito por dezenas delas.
  - Deve-se estabelecer:
    - Padrão para nomes de métodos, classes, variáveis
    - Organização do código (chaves, etc.)
    - Código com estrutura familiar facilita e estimula
  - Benefícios:
    - Posse coletiva
    - Comunicação mais eficiente
    - Simplicidade
    - Programação em pares

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Testes de aceitação
  - São testes planejados e conduzidos pela equipe em conjunto com o cliente para verificar se os requisitos foram atendidos

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas - Desenvolvimento dirigido por testes
  - Antes de programar uma unidade (método/classe/módulo), devem-se definir e implementar os testes pelos quais ela deverá passar.

eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

eXtreme Programming (XP)

- Práticas – Refatoração
  - Não se deve fugir da refatoração quando ela for necessária.
  - A refatoração permite manter a complexidade do código em um nível gerenciável, além de ser um investimento que traz benefícios em médio e longo prazo (evoluções e manutenção)
  - A refatoração deve ser realizada o tempo todo durante o projeto
  - Objetivos:
    - melhorar o design
    - simplificar o código
    - remover código duplicado
    - aumentar a coesão,
    - reduzir o acoplamento

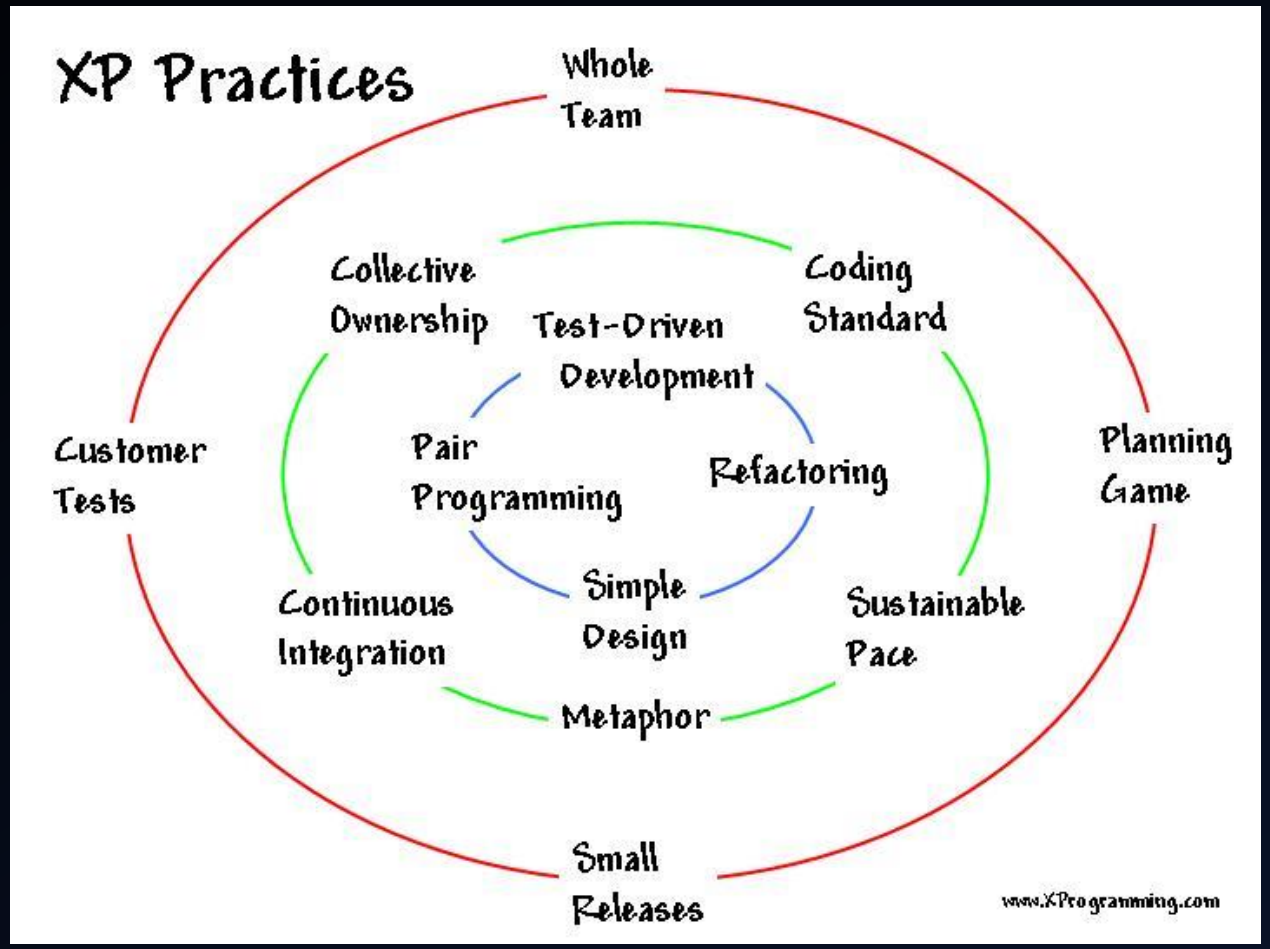
eXtreme Programming (XP)

- Práticas:
  - Jogo de planejamento
  - Metáfora
  - Equipe coesa
  - Reuniões em pé
  - Design simples
  - Versões pequenas
  - Ritmo sustentável
  - Posse coletiva
  - Programação em pares
  - Padrões de codificação
  - Testes de aceitação
  - Desenvolvimento dirigido por testes
  - Refatoração
  - Integração contínua

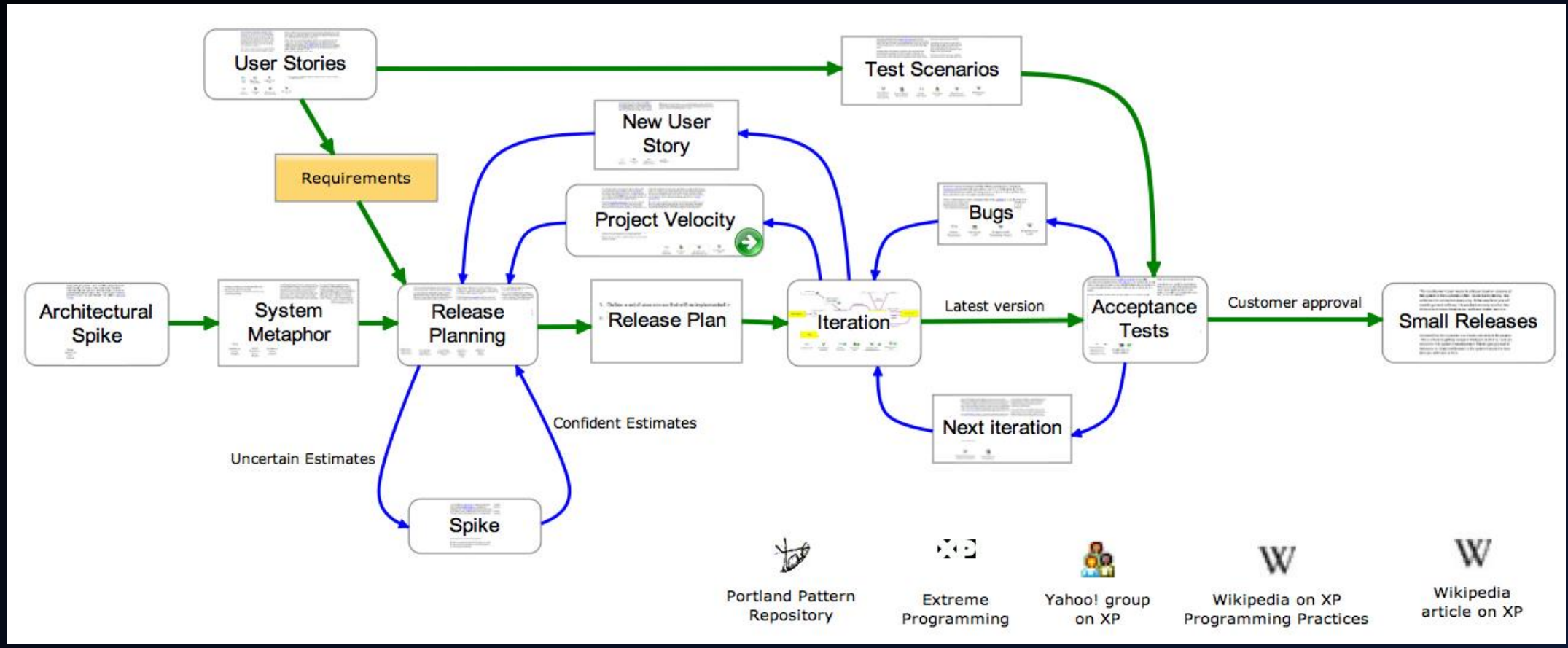
eXtreme Programming (XP)

- Práticas - Integração contínua
  - Nunca se deve esperar até o final do ciclo para integrar uma nova funcionalidade. Assim que estiver viável, ela deverá ser integrada para evitar surpresas.
  - Integração de todo o sistema pode ocorrer várias vezes ao dia (pelo menos uma vez ao dia)
  - Todos os testes (unidade e integração) devem ser executados
  - Benefícios
    - Expõe o estado atual do desenvolvimento (viabiliza lançamentos pequenos e frequentes)
    - Estimula design simples, tarefas curtas, agilidade
    - Oferece feedback sobre todo o sistema
    - Permite encontrar problemas de design rapidamente

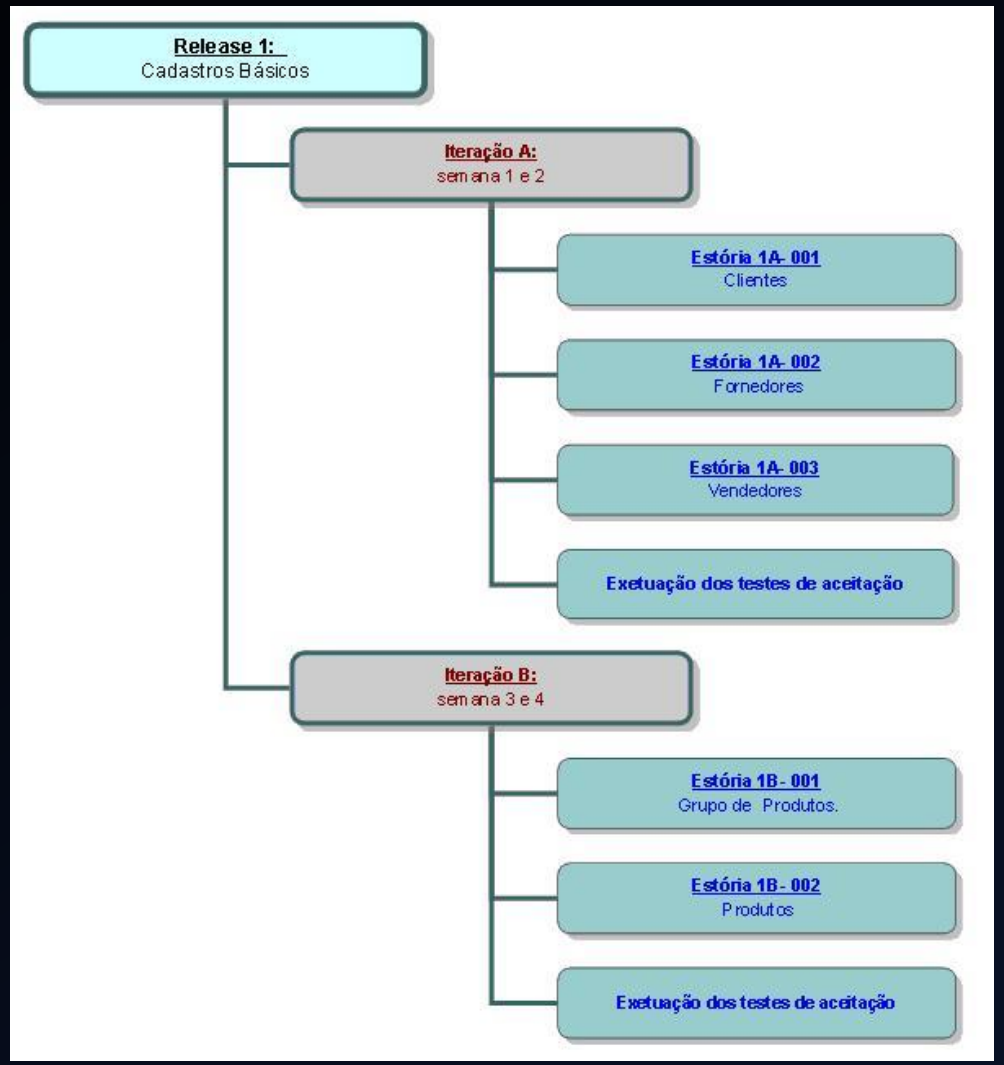
eXtreme Programming (XP)



eXtreme Programming (XP)

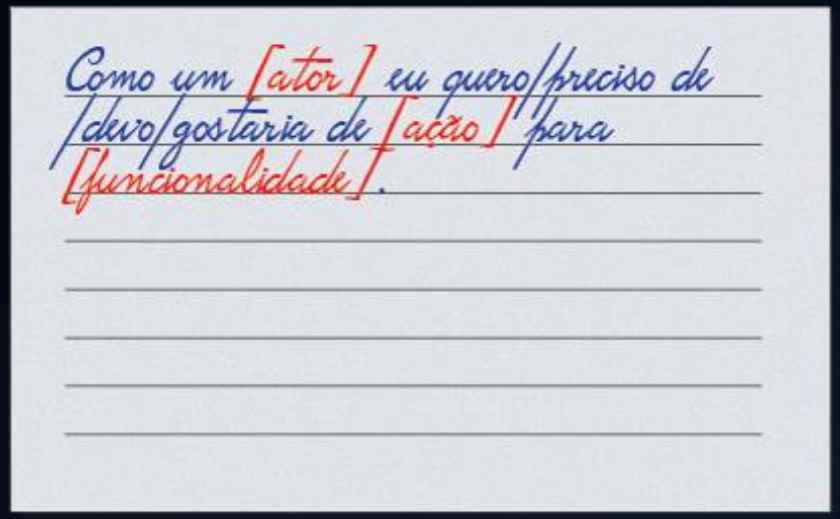


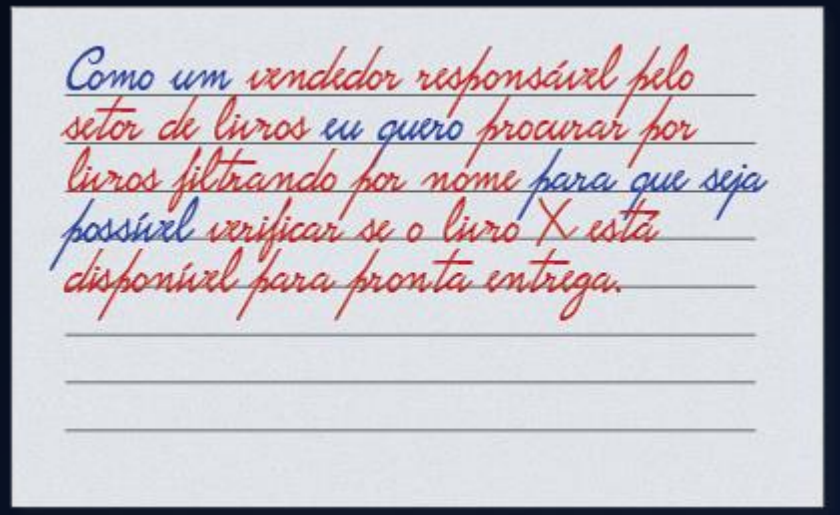
eXtreme Programming (XP)



eXtreme Programming (XP)

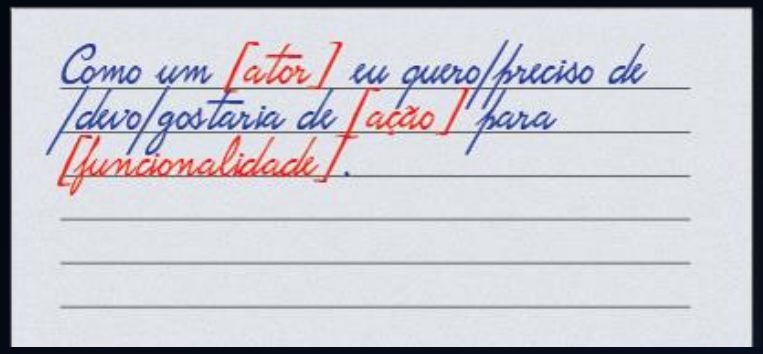
- Estórias: é uma forma de expressar os requisitos do software de maneira informal e do ponto de vista de situações específicas determinadas pelos usuários

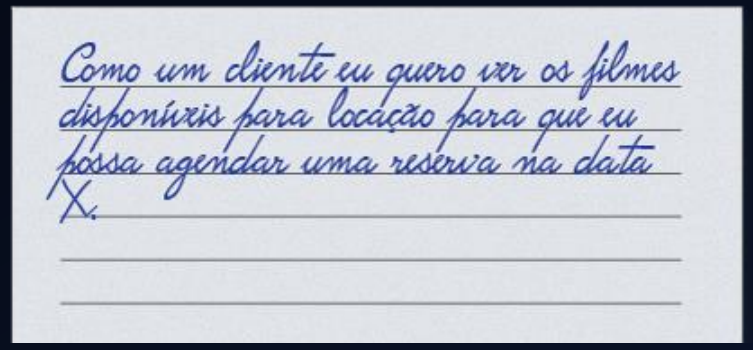


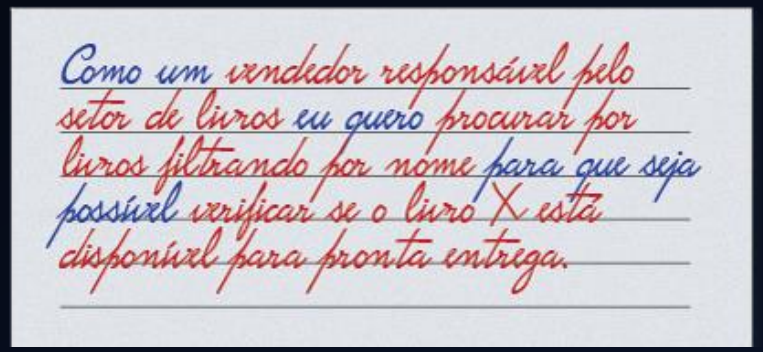


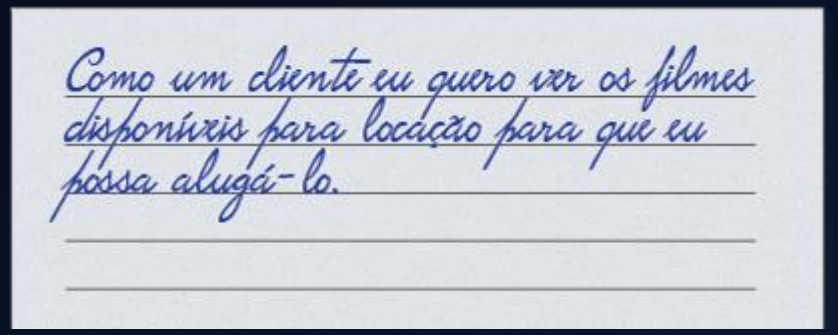
eXtreme Programming (XP)

- Estórias: é uma forma de expressar os requisitos do software de maneira informal e do ponto de vista de situações específicas determinadas pelos usuários









Projetos ágeis

- O ciclo de vida de um projeto ágil é tipicamente baseado em iterações que duram de uma a quatro semanas e terminam com um protótipo ou uma pequena versão executável do software.
- Cada iteração implementa algumas estórias de usuário (requisitos), que funcionam como testes de aceitação ou testes de integração do software desenvolvido
- Os interessados avaliam o produto após cada ciclo para ver se ele é aquilo que ele espera, e então sugere/prioriza/altera outras estórias.

Projetos ágeis

- Não é aceitável a ideia de que um desenvolvedor com incríveis habilidades individuais desenvolva um software sozinho
- Produtividade e qualidade hoje significam o desenvolvimento em equipes, e de preferência equipes pequenas.
- Jeff Bezos caracterizou as equipes de desenvolvimento como “two-pizza” teams, ou seja, duas pizzas devem ser o suficiente para alimentar uma equipe de desenvolvimento.
- Alguns sugerem que uma equipe tenha de 4 a 9 componentes, outros de 6 a 12, mas o fato é que as equipes não podem ser muito grandes para não prejudicar a comunicação

Scrum

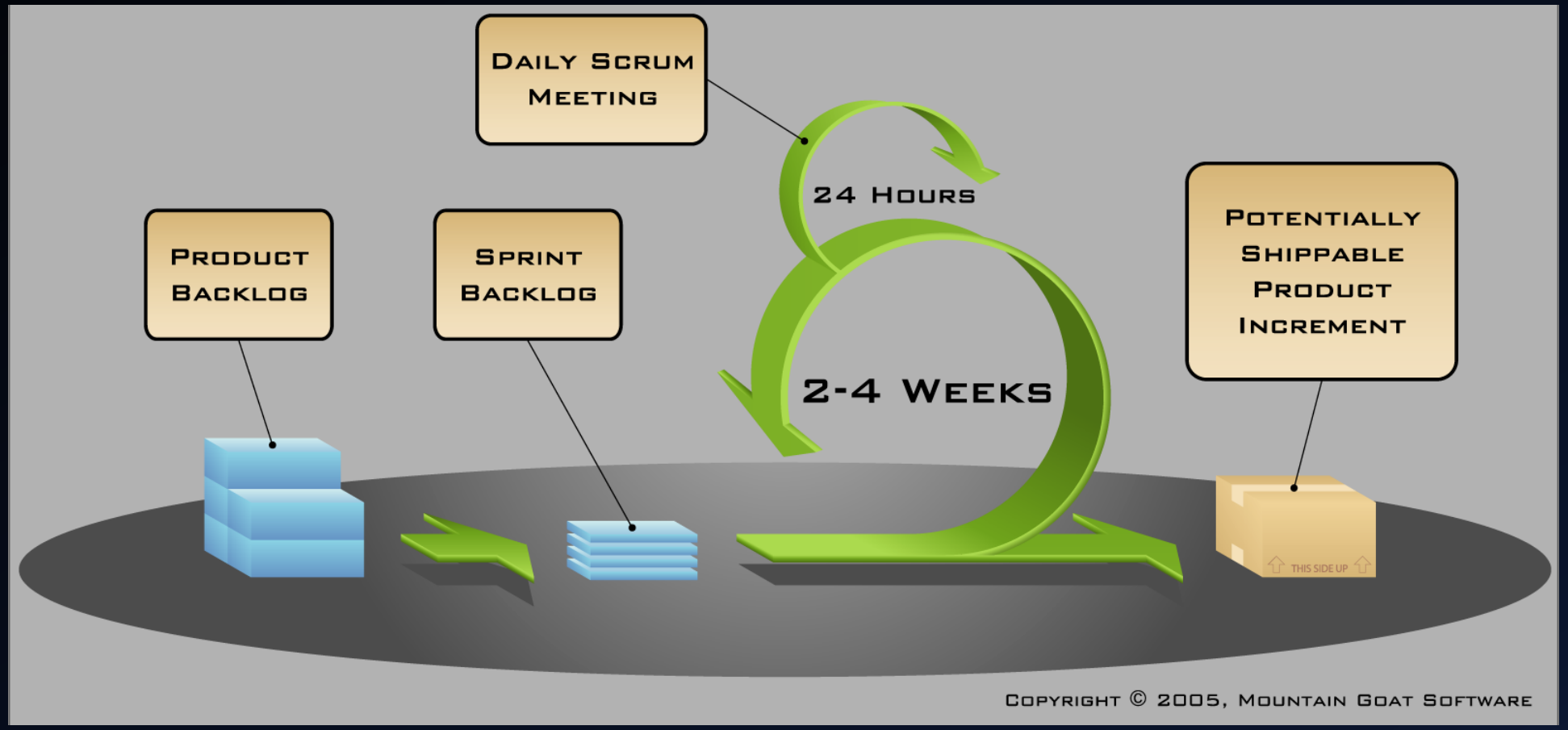
- O Scrum é uma alternativa de utilizar métodos ágeis na gerência de projetos
- Ele é simples
  - Processo, artefatos e regras são poucos e muito fáceis de entender
  - A simplicidade pode ser decepcionante aos acostumados com metodologias clássicas
  - Não é um método muito prescritivo, pois não define previamente tudo que deve ser feito em cada situação
- Aplica o senso comum
  - Combinação de experiência, treinamento, confiança e inteligência de toda a equipe
  - Senso comum em vez do senso de uma única pessoa é uma das razões do sucesso do Scrum

Scrum

- Ênfases
  - Comunicação
  - Trabalho em equipe
  - Flexibilidade
  - Fornecer software funcionando – incrementalmente

Scrum

- Visão geral



Scrum

- Principais elementos
  - Backlog:
    - Lista gerada incrementalmente de todas as funcionalidades desejadas
  - Equipes
    - Sem hierarquia, mas com distinção de papéis
  - Sprint
    - Unidade básica de tempo (no máximo 30 dias)
  - Encontros Scrum
    - Encontros diários da equipe para falar sobre o que foi feito no dia anterior e planejar as tarefas do dia
  - Revisões Scrum/Demos
    - No final de cada Sprint é feita uma reunião com todos os interessados para avaliar os resultados e o processo

Scrum

- Fases
  - Planejamento geral
  - Sprints
    - Planejamento do Sprint
    - Reuniões Diárias
    - Revisão
    - Retrospectivas
  - Encerramento

Scrum

- Papéis
  - Scrum master
  - Product owner
  - Scrum team

Scrum

- Papéis – Scrum Master
  - Responsável pelo sucesso do Scrum
  - Ensina o Scrum para os envolvidos com o projeto
  - Implementa o Scrum na empresa de forma adaptada a sua cultura, para continuamente gerar benefícios
  - Certifica se cada pessoa envolvida está seguindo seus papéis e as regras do Scrum
  - Certifica que pessoas não responsáveis não interfiram no processo

Scrum

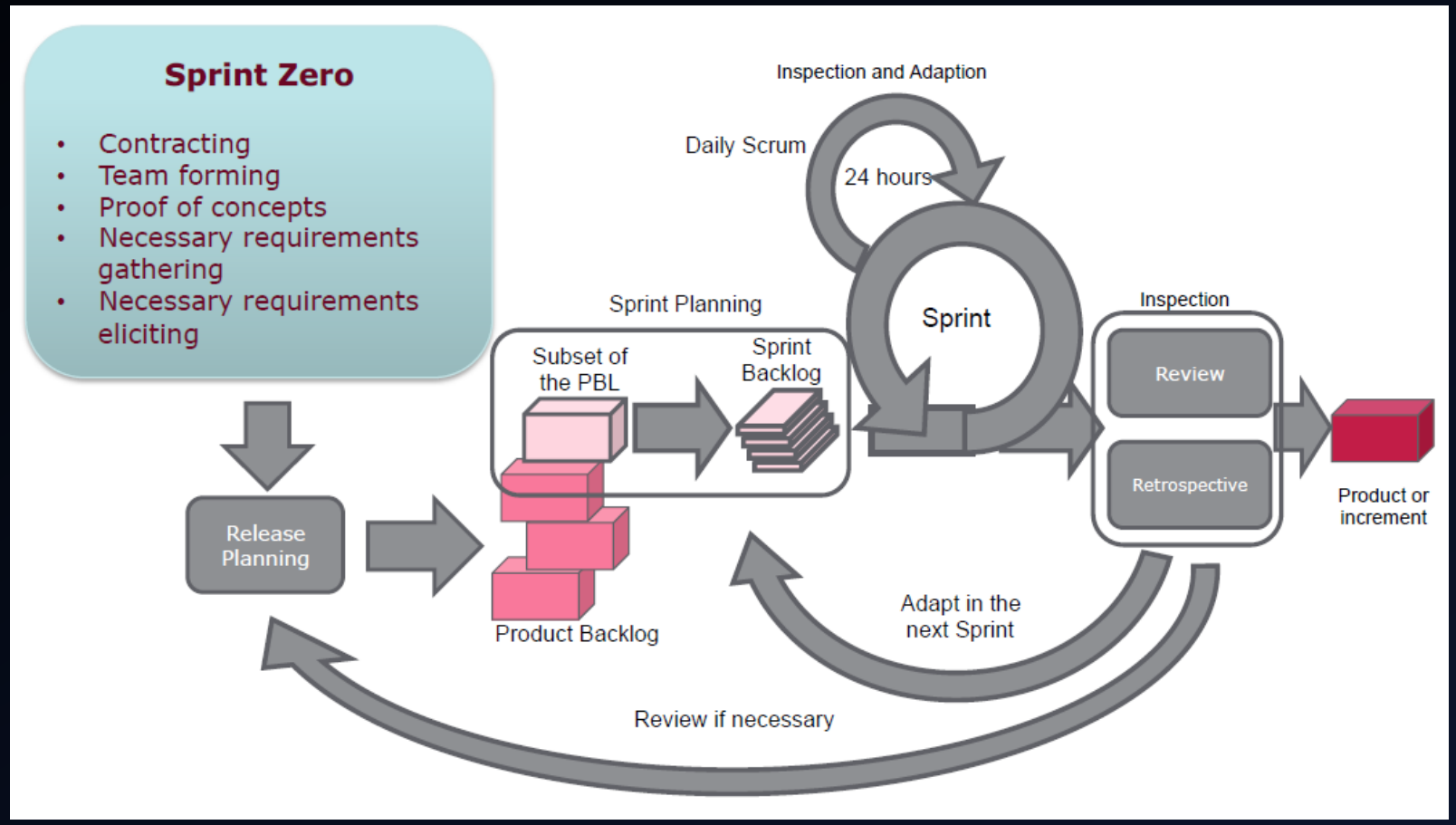
- Papéis - Product owner
  - Responsável por apresentar os interesses de todos os stakeholders
  - Define fundamentos iniciais do projeto, objetivos e planos de release
  - Responsável pela lista de requisitos (Product Backlog)
  - Certifica se as atividades com maior valor para o negócio são desenvolvidas primeiro
  - Priorização frequente das funcionalidades antes de cada iteração

Scrum

- Papéis - Scrum team
  - Responsável por escolher as funcionalidades a serem desenvolvidas em cada interação e desenvolvê-las
  - O time se auto gerencia, se auto organiza, portanto não há hierarquia
  - Todos os membros do time são coletivamente responsáveis pelo sucesso de cada iteração
  - Deve ter entre 6 e 10 pessoas
  - Membros do time devem assumir diferentes papéis

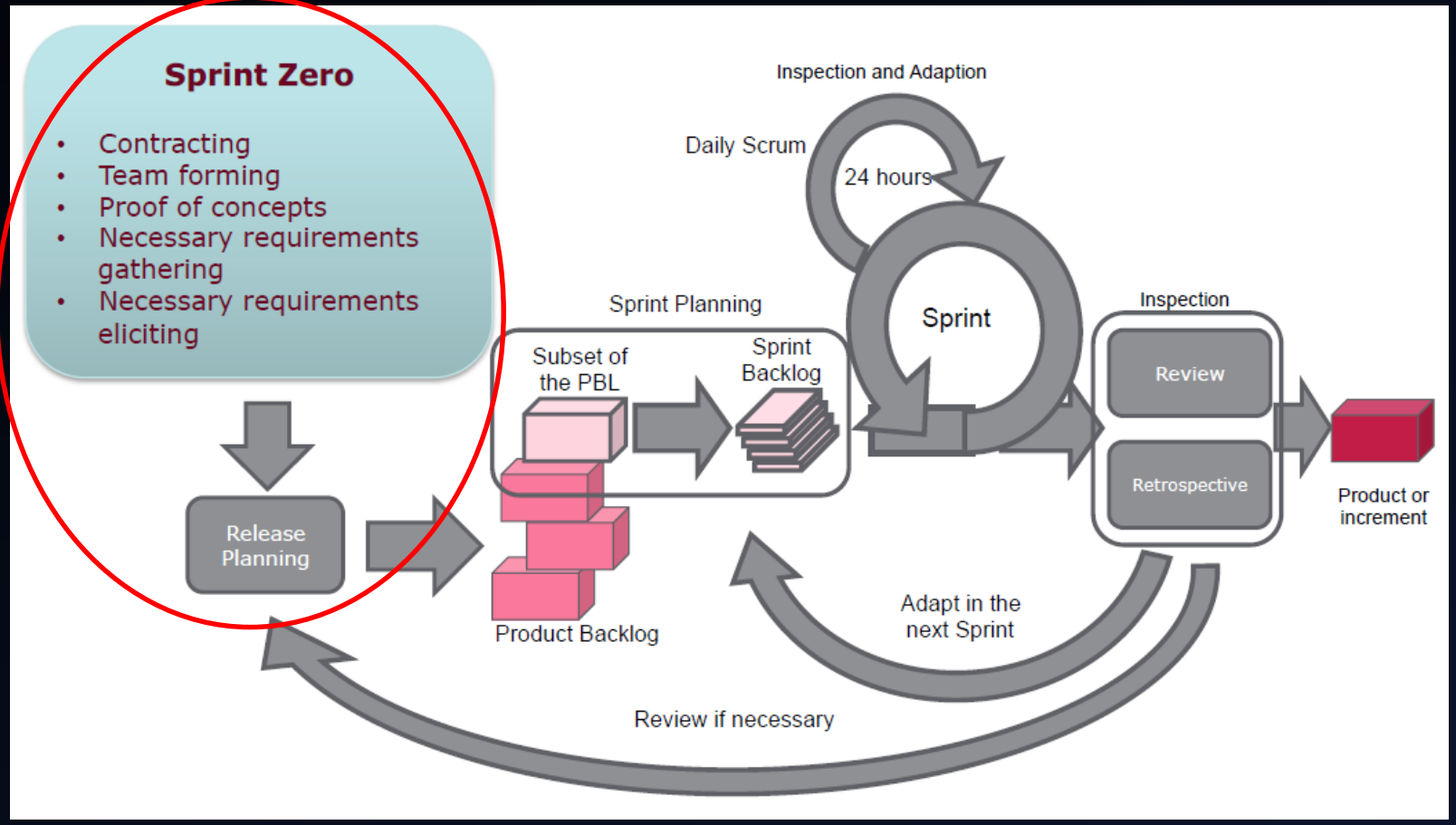
Scrum

- Com mais detalhes



Scrum

- Com mais detalhes

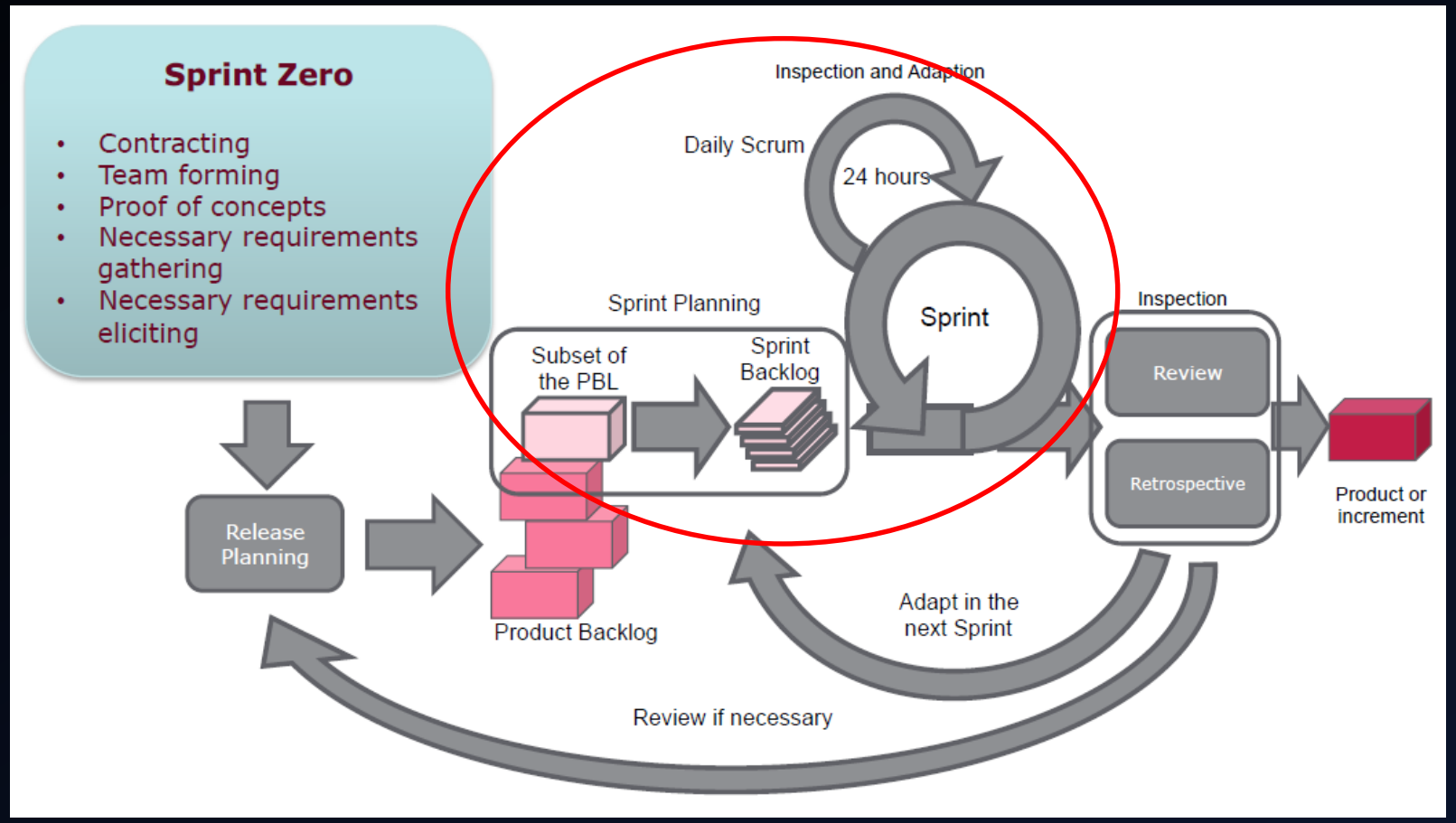


Scrum

- Planejamento geral
  - Relativamente curto
  - Projeto da arquitetura do sistema
  - Estimativas de datas e custos
  - Criação do backlog
    - Participação de clientes e outros departamentos
    - Levantamento dos requisitos e atribuição de prioridades
  - Definição de equipes e seus líderes
  - Definição de pacotes a serem desenvolvidos

Scrum

- Com mais detalhes



Scrum

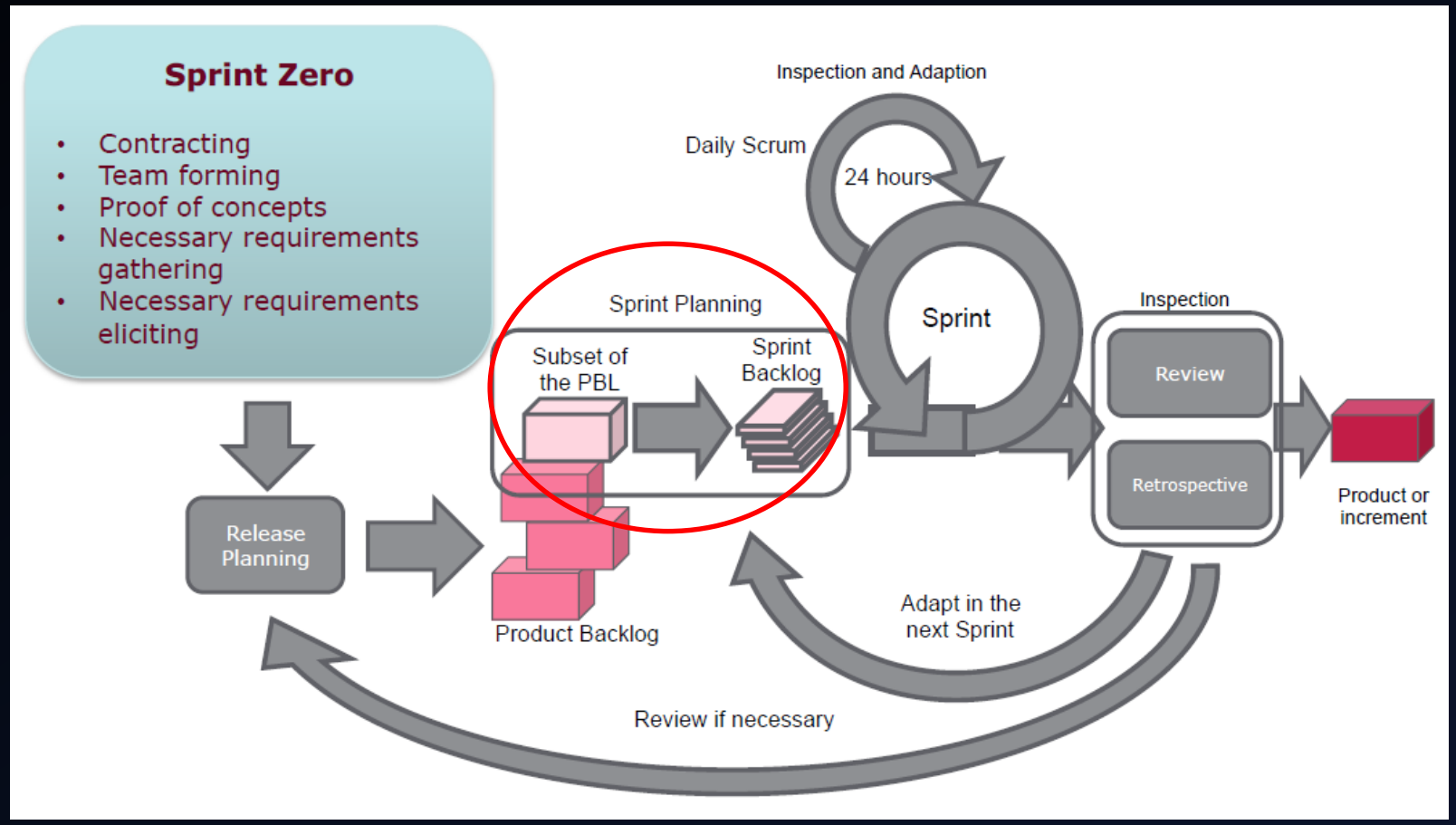
- Sprint
  - Não deve ser maior do que 30 dias consecutivos
    - Duração de 1 a 4 semanas
  - Sem considerar outros fatores, este é o tempo necessário para produzir algo de interesse para o Product Owner e os stakeholders
  - O time recebe uma parte do backlog para desenvolvimento
  - O time se compromete com o Product Backlog
  - O backlog não sofrerá modificações durante o Sprint
  - Sempre apresentam um executável ao final

Scrum

- Sprint
  - Responsabilidades do time durante o Sprint:
    - Participar das reuniões diárias do Scrum
    - Manter o Sprint Backlog atualizado
    - Disponibilizar o Sprint Backlog publicamente
    - O time tem o compromisso de implementar todos os itens selecionados

Scrum

- Com mais detalhes



Scrum

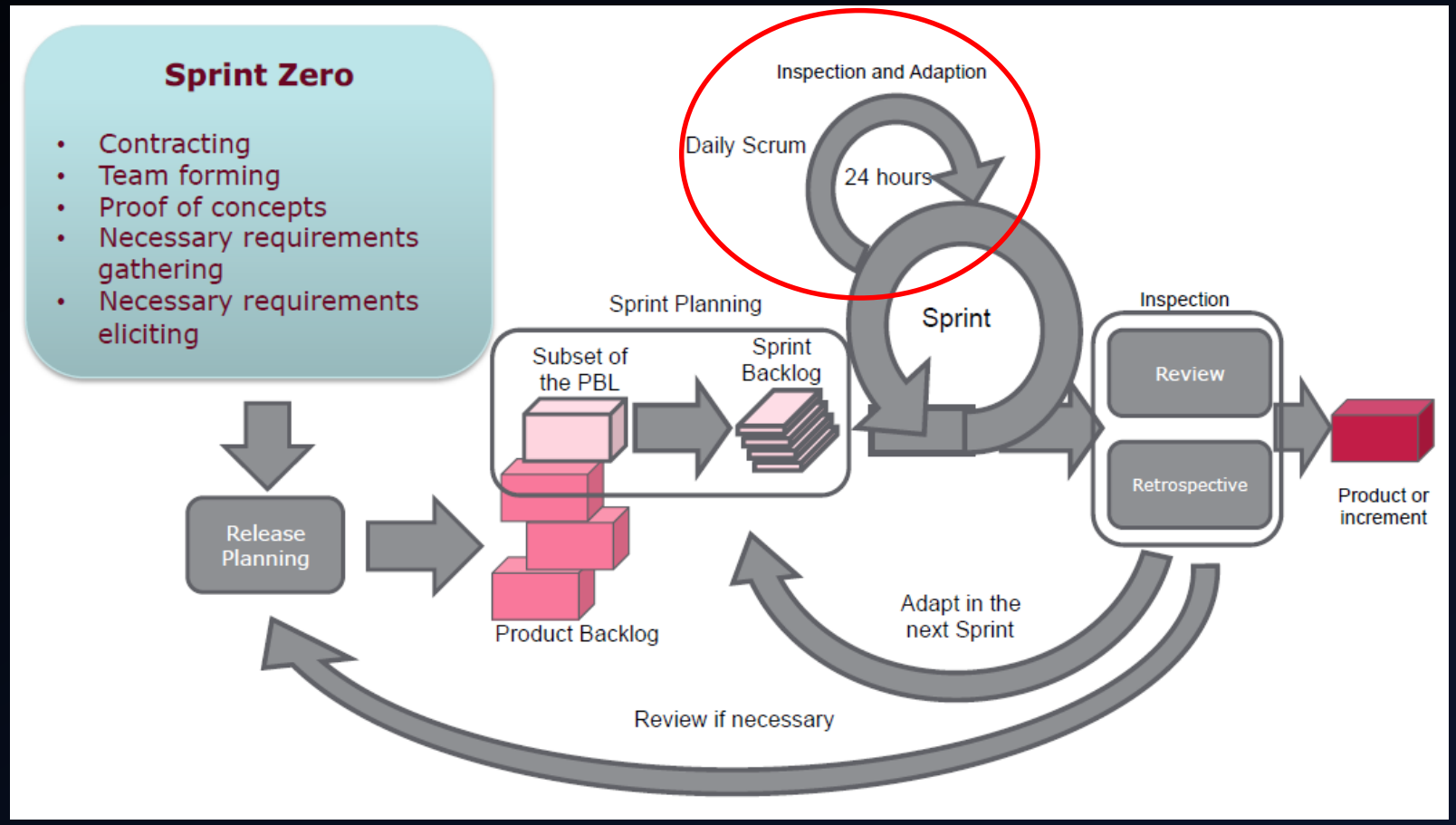
- Sprint – Planejamento
  - A reunião de planejamento do Sprint deve ocorrer dentro de 8 horas com duas partes de 4 horas
  - Primeiro seguimento:
    - Product Owner deve preparar o Product Backlog antes da reunião
    - Seleção dos itens do Product Backlog que o time se compromete em torná-los incrementos potencialmente implementáveis
    - Decisão final é do Product Owner
    - Stakeholders não devem participar

Scrum

- Sprint – Planejamento
  - Segundo seguimento:
    - Ocorre imediatamente após o primeiro
    - Product Owner deve estar disponível para o que o time faça perguntas sobre o Product Backlog
    - O time deve decidir sozinho como os itens selecionados serão implementados
    - Nenhum outro participante pode fazer perguntas ou observações nesta parte
    - Resultado deste seguimento é o Sprint Backlog

Scrum

- Com mais detalhes



Scrum

- Sprint: reuniões diárias em pé
  - Reunião de no máximo 15 minutos, a menos que o time seja grande o suficiente para precisar de mais tempo
  - Deve ser feita no mesmo lugar onde o time trabalha
  - Resulta em melhores resultados se realizada no inicio do dia de trabalho
  - Todos os membros do time devem participar desta reunião

Scrum

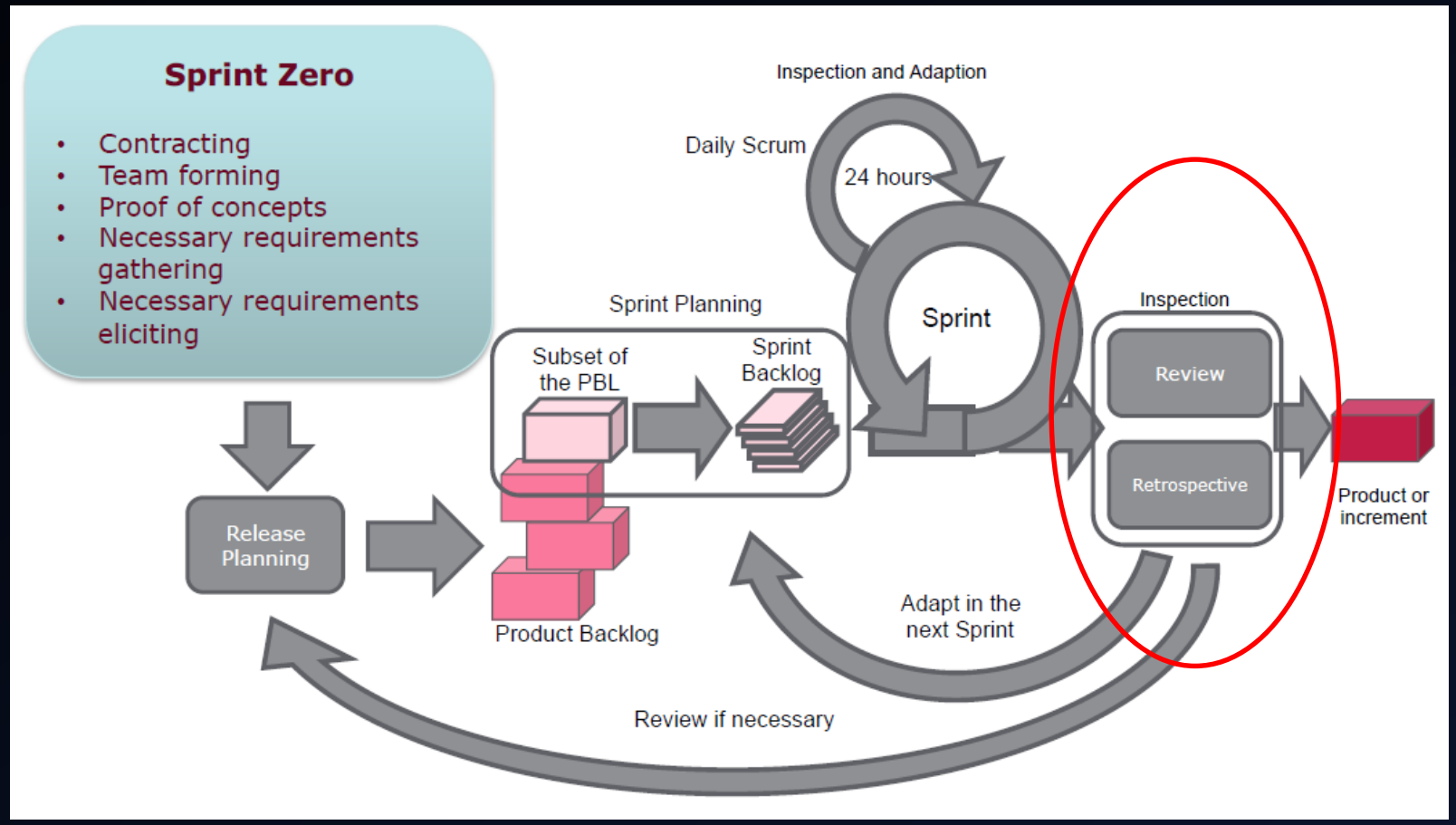
- Sprint: reuniões diárias em pé
  - ScrumMaster faz as seguintes perguntas para cada membro do time:
    - O que você realizou desde a última reunião?
    - Quais problemas você enfrentou?
    - Em que você trabalhará até a próxima reunião?
  - Os membros devem responder apenas a estas perguntas para não estender a reunião

Scrum

- Sprint: reuniões diárias em pé
  - Benefícios:
    - Maior integração entre os membros da equipe
    - Rápida solução de problemas
    - Promovem o compartilhamento de conhecimento
    - Progresso medido continuamente
    - Minimização de riscos

Scrum

- Com mais detalhes



Scrum

- Sprint: revisão
  - Reunião de no máximo 4 horas sob responsabilidade do ScrumMaster
  - O time não deve gastar mais de 1 hora na preparação desta reunião
  - Objetivo:
    - Mostrar ao Product Owner e stakeholders as funcionalidades que foram feitas
  - Artefatos não devem ser apresentados, pois não são funcionalidades
  - No final da reunião
    - Cada stakeholder fala suas impressões e sugere mudanças com suas respectivas prioridades
    - Possíveis modificações no Product Backlog são discutidas entre o Product Owner e o time
    - ScrumMaster anuncia a data e o local da próxima reunião de revisão do Sprint ao Product Owner e a todos stakeholders

Scrum

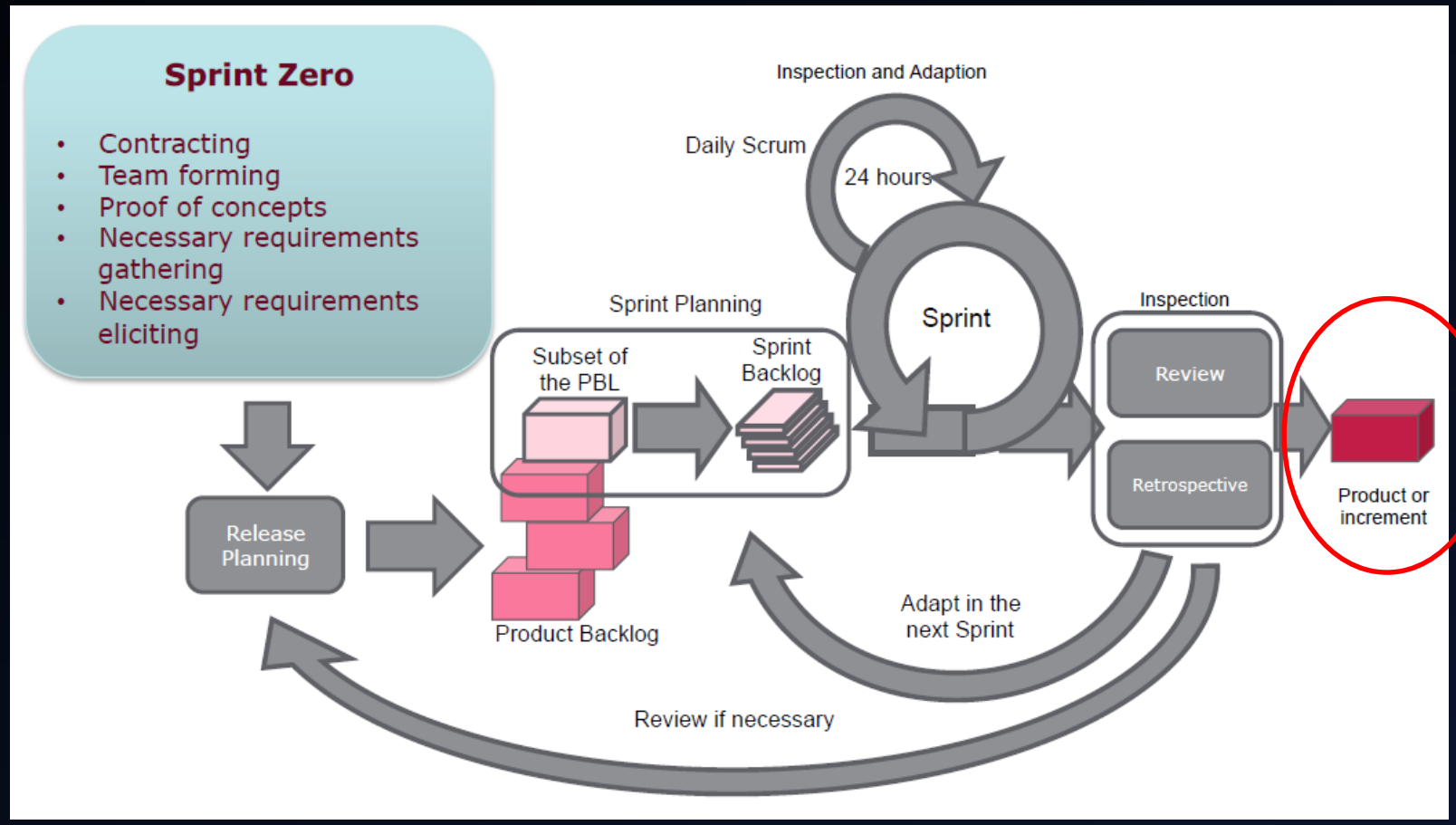
- Sprint: revisão
  - Deve obedecer à data de entrega
  - Permitida a diminuição de funcionalidades
  - Sugestões de mudanças são incorporadas ao backlog
  - Benefícios:
    - Apresentar resultados concretos ao cliente
    - Integrar e testar uma boa parte do software
    - Motivação da equipe

Scrum

- Sprint: retrospectiva
  - Avaliação do processo utilizado no Sprint
  - Não deve ser maior do que 3 horas
  - Participam desta reunião
    - Time, ScrumMaster e, opcionalmente, Product Owner
  - Os membros do time devem responder a duas questões:
    - O que aconteceu de bom durante o último Sprint?
    - O que pode ser melhorado para o próximo Sprint?
  - ScrumMaster escreve as respostas e prioriza na ordem que deseja discutir as potenciais melhorias
  - ScrumMaster nesta reunião tem o papel de fazer com que o time encontre melhores formas de aplicar o Scrum

Scrum

- Com mais detalhes

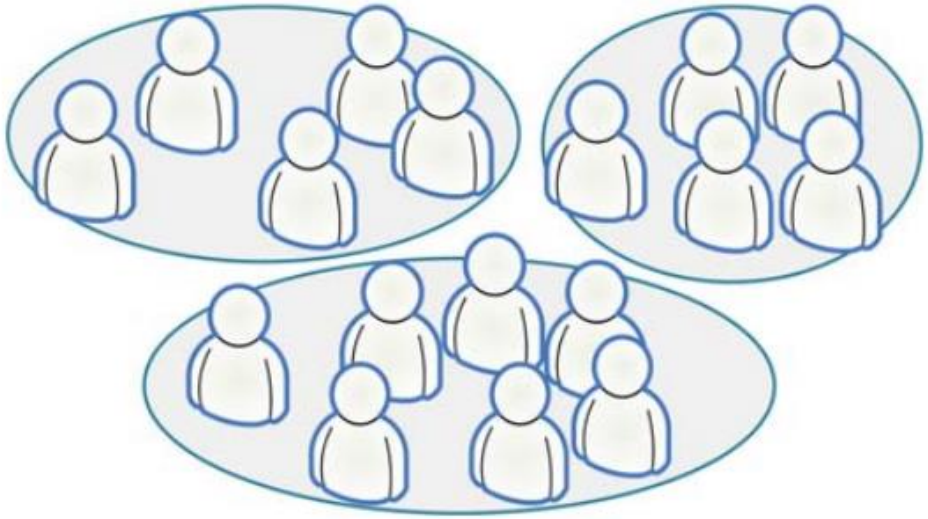


Scrum

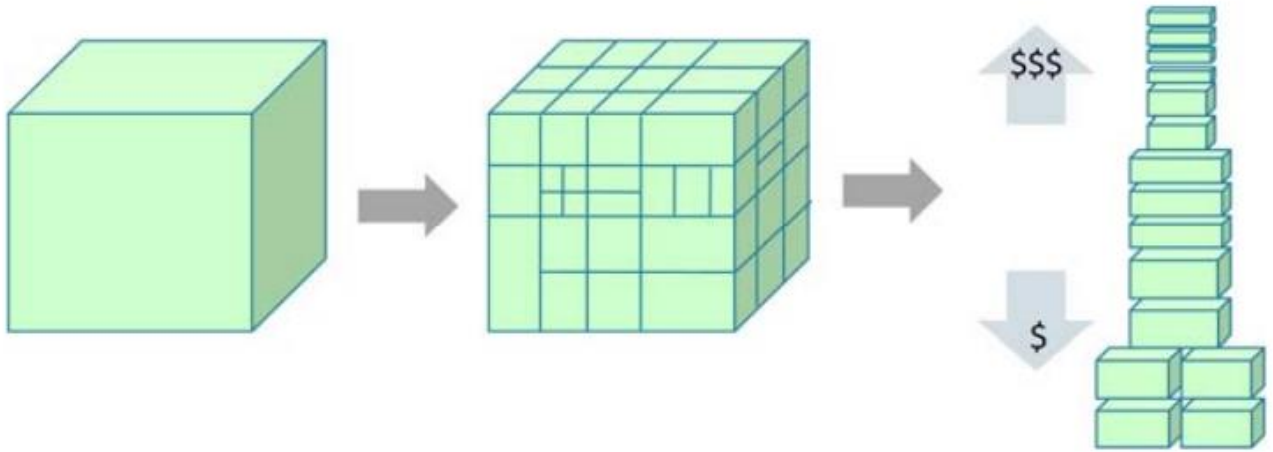
- Encerramento
  - Finalização do projeto
  - Atividades:
    - Testes de integração
    - Testes de sistema
    - Documentação do usuário
    - Preparação de material de treinamento
    - Preparação de material de marketing

Scrum in a nutshell

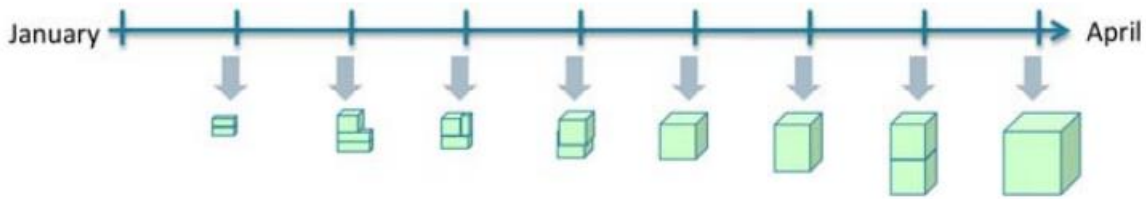
- **Split your organization** into small, cross-functional, self-organizing teams.



- **Split your work** into a list of small, concrete deliverables. Sort the list by priority and estimate the relative effort of each item.



- **Split time** into short fixed-length iterations (usually 1 – 4 weeks), with potentially shippable code demonstrated after each iteration.



- **Optimize the release plan** and update priorities in collaboration with the customer, based on insights gained by inspecting the release after each iteration.

- **Optimize the process** by having a retrospective after each iteration.

So instead of a **large group** spending **a long time** building a **big thing**, we have a **small team** spending a **short time** building a **small thing**. But **integrating regularly** to see the whole.

121 words... close enough.

http://www.infoq.com/resource/minibooks/kanban-scrum-minibook/en/pdf/KanbanAndScrumInfoQVersionFINAL.pdf

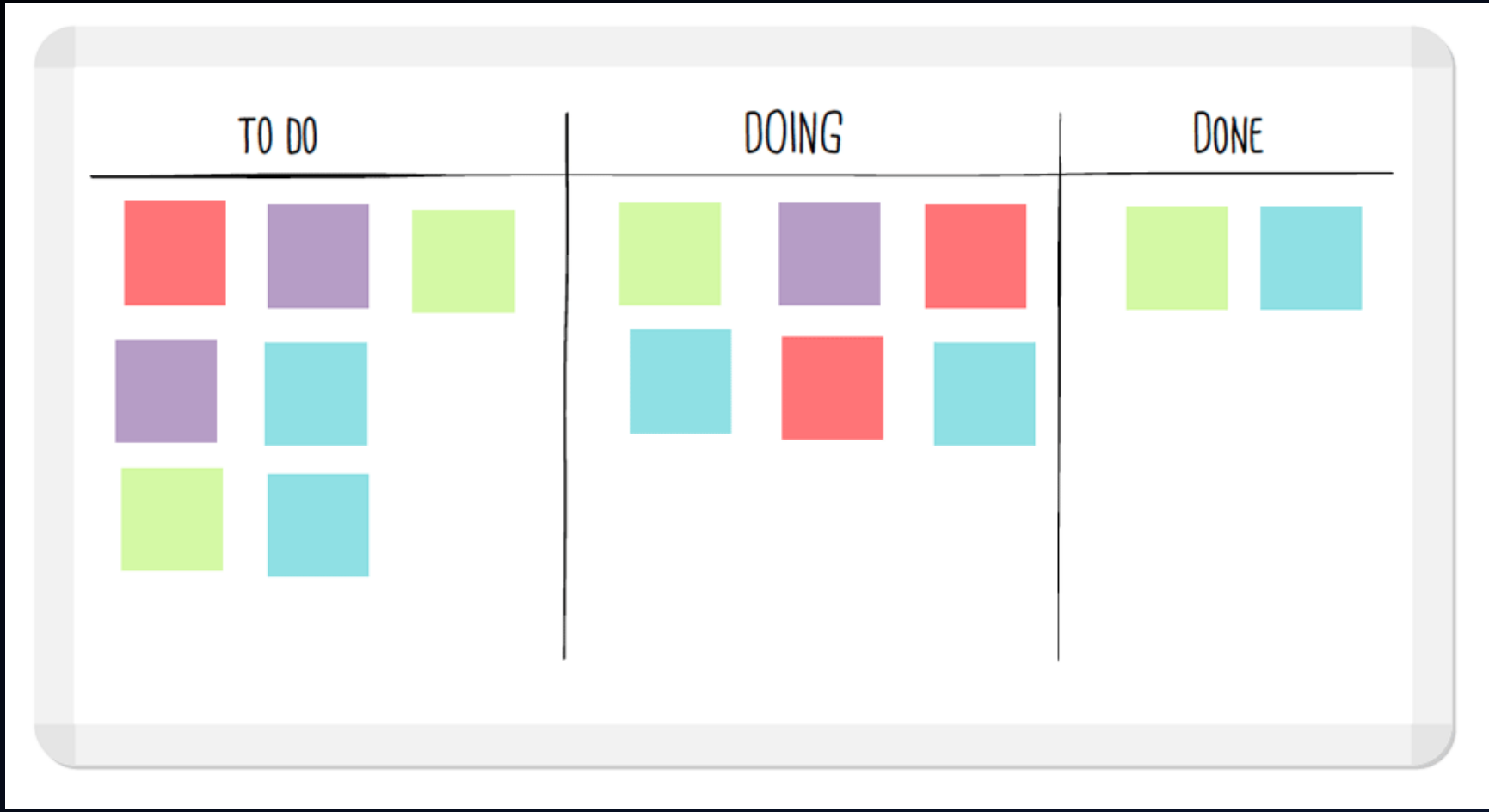
Sistema Kanban para Desenvolvimento de Software

- Kanban é um palavra em japonês que significa literalmente registro ou placa (cartão) visível.
- O sistema Kanban é uma das variantes mais conhecidas do Just in Time (JIT), metodologia desenvolvida e aperfeiçoada em 1940 por Taiichi Ohno e Sakichi Toyoda conhecida como Sistema Toyota de Produção.
- O Kanban para desenvolvimento de software é um modelo adaptado da indústria, mais especificamente da Toyota. David Anderson foi o grande responsável por essa adaptação.

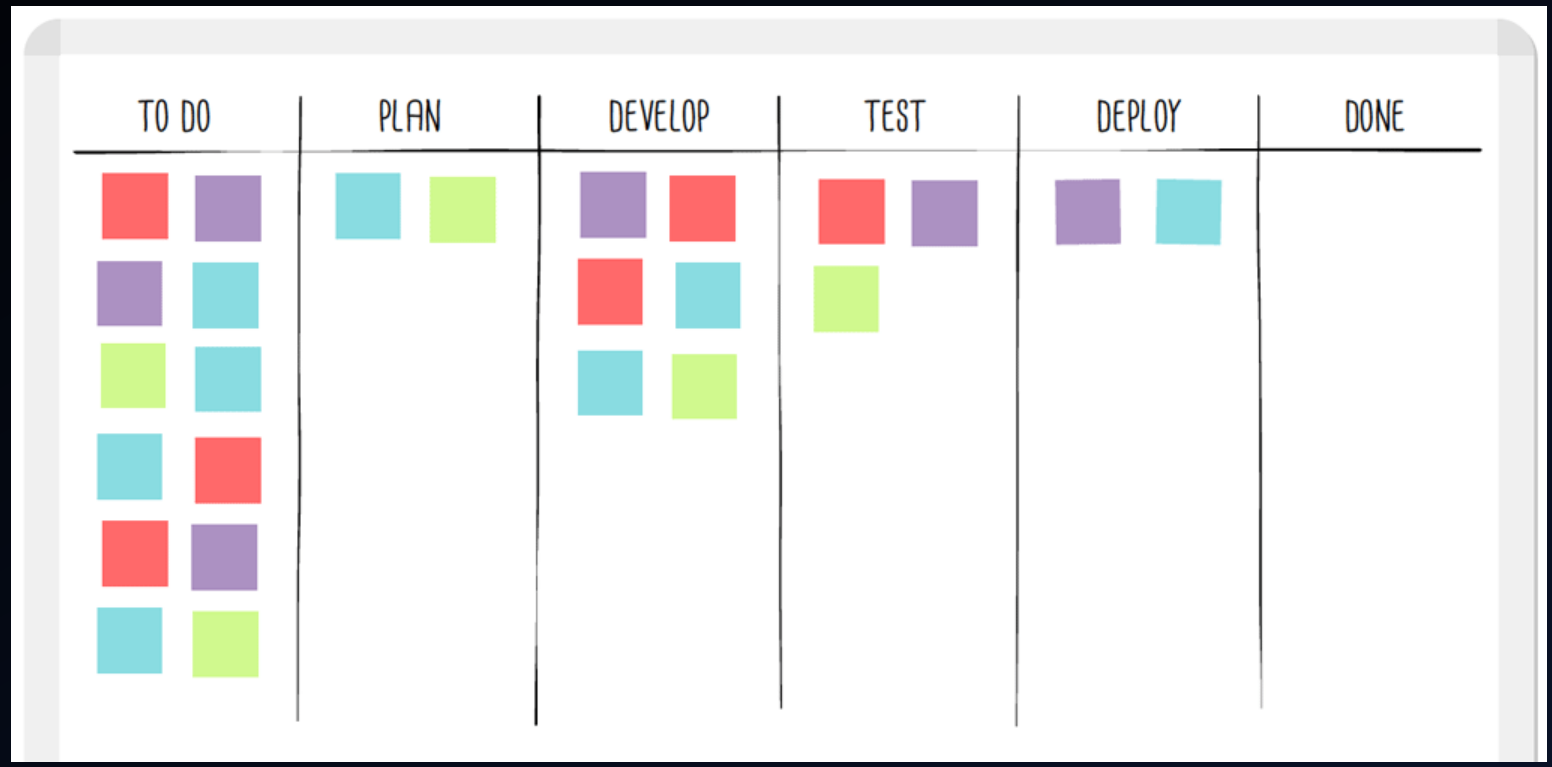
Sistema Kanban para Desenvolvimento de Software

- Recomendações:
  - Visualize o fluxo de trabalho
    - Divida o trabalho em partes, escreva cada item em um cartão e os coloque em uma parede
    - Nomeie colunas para ilustrar em que parte do processo cada item está
  - Limite o trabalho em andamento (Work In Progress - WIP)
    - Determine limites para a quantidade de itens que pode estar em cada coluna
  - Meça o tempo médio para a realização de cada item
    - Otimize o processo para fazer com que este tempo seja o menor e o mais previsível possível

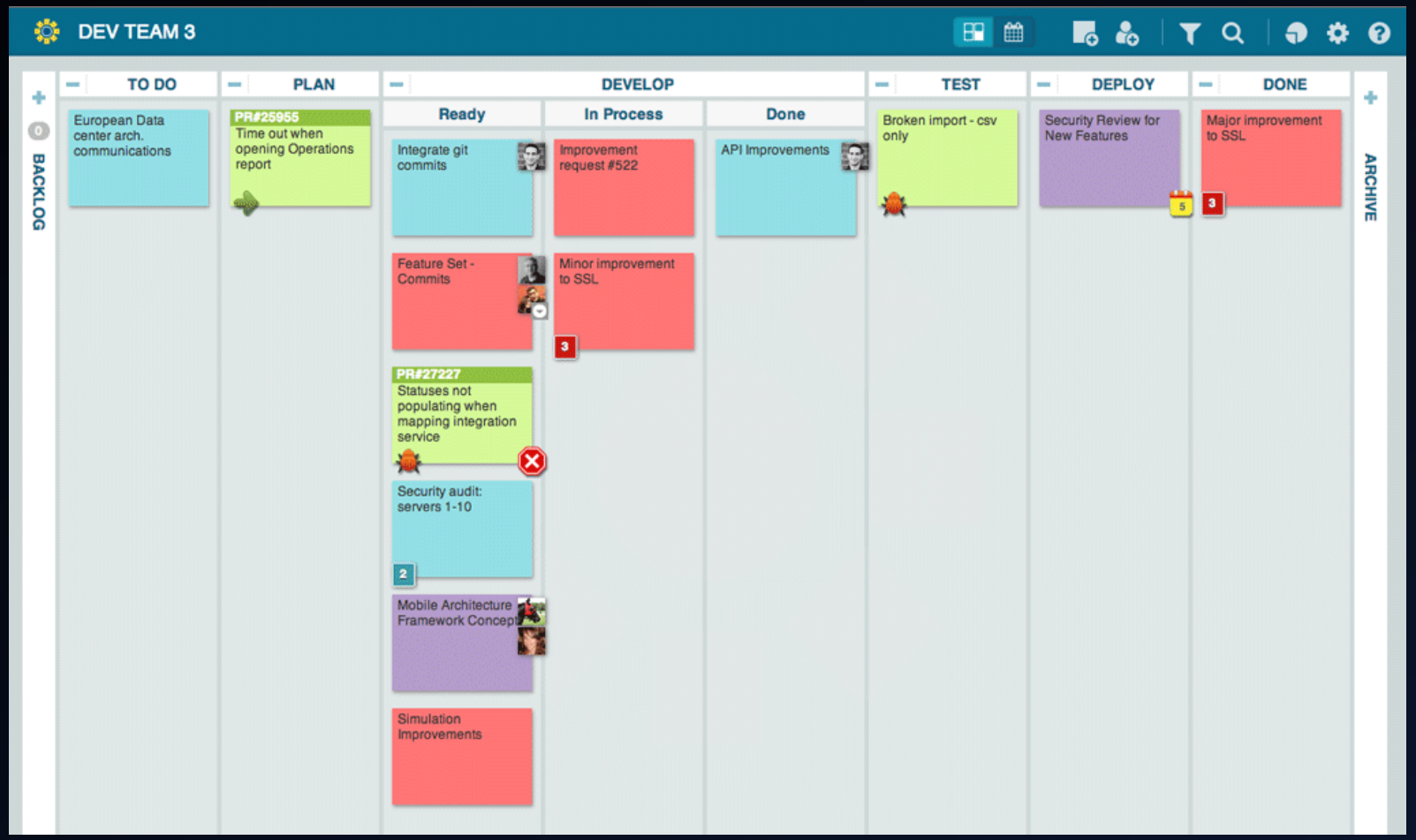
Sistema Kanban para Desenvolvimento de Software



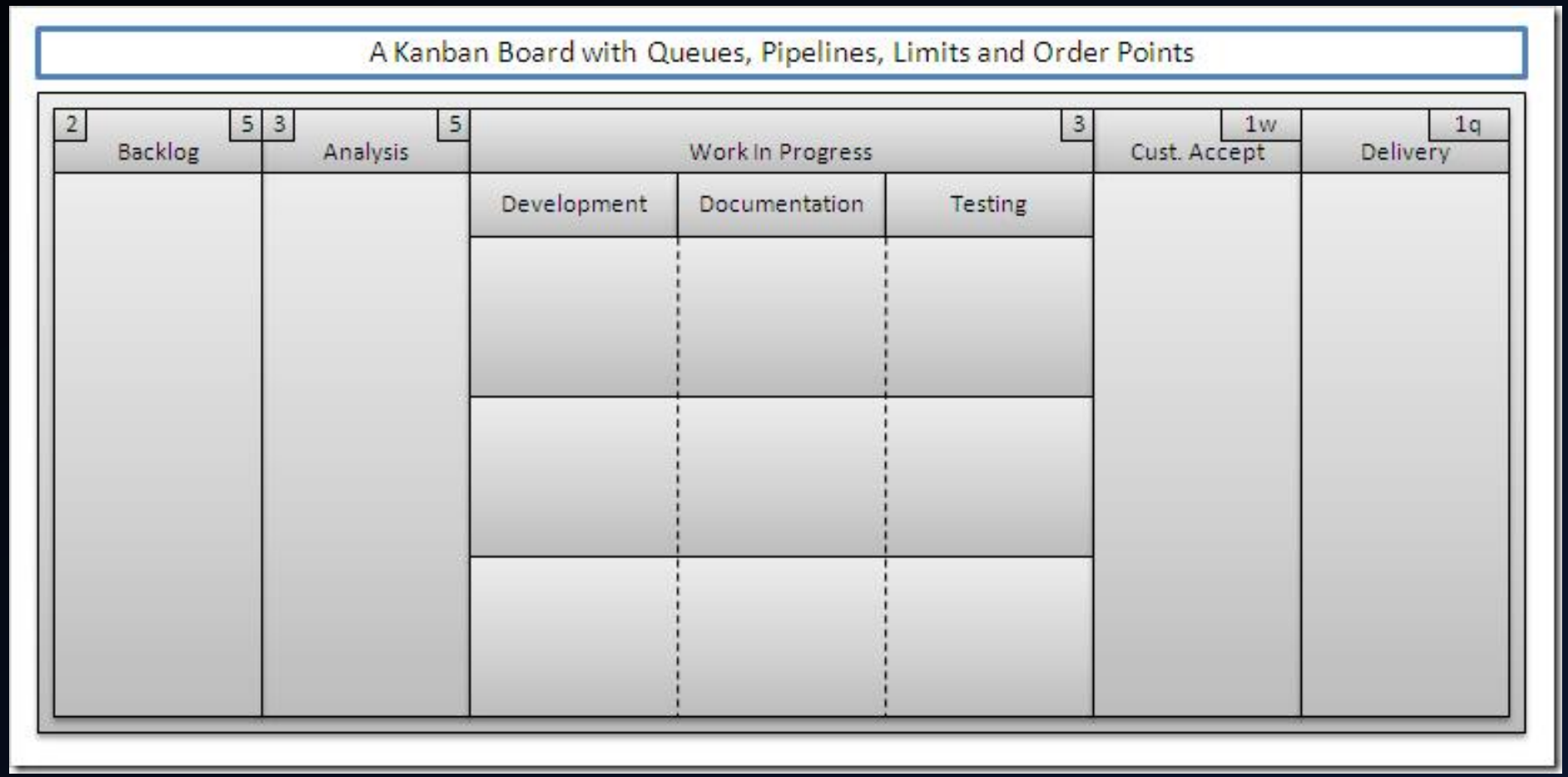
Sistema Kanban para Desenvolvimento de Software



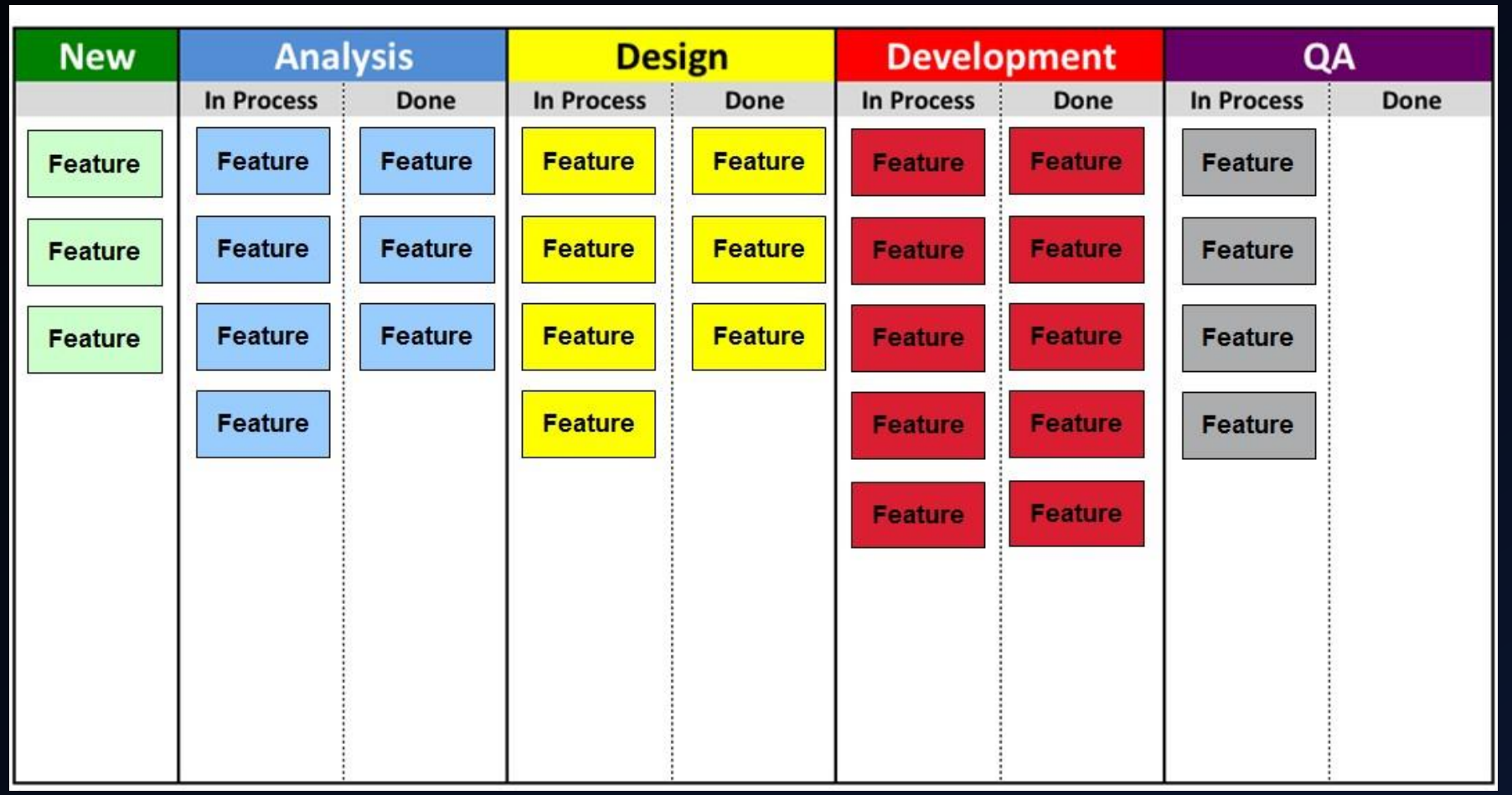
Sistema Kanban para Desenvolvimento de Software



Sistema Kanban para Desenvolvimento de Software



Sistema Kanban para Desenvolvimento de Software

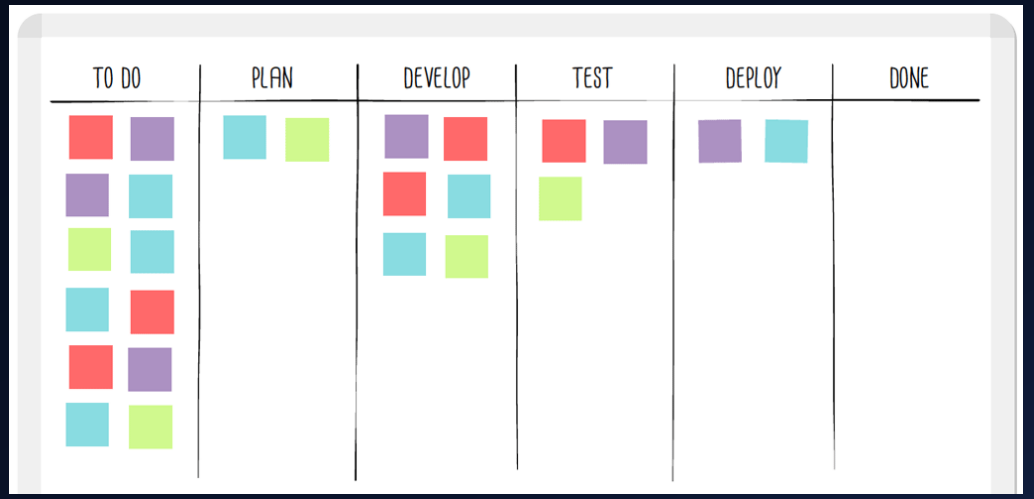


Sistema Kanban para Desenvolvimento de Software

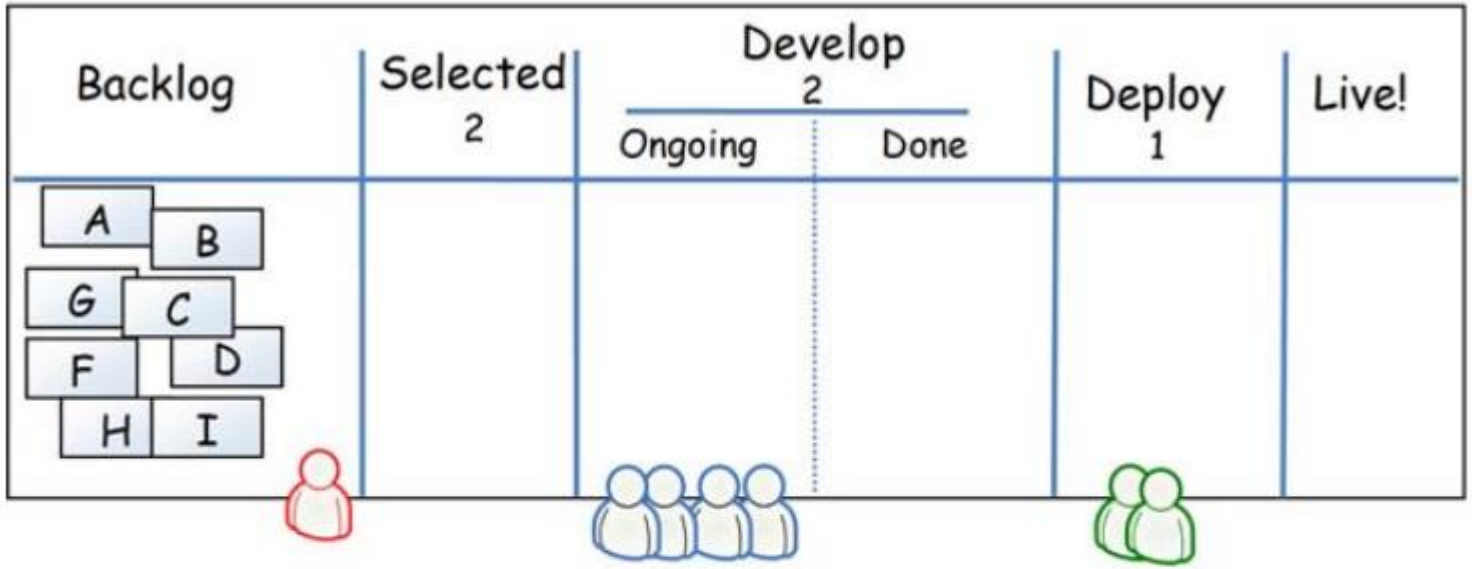
- O Kanban não é um Sistema muito prescritivo para gerenciamento de projetos de software, e por isso permite muitas adaptações
- Entretanto, suas pequenas regras devem ser seguidas como, por exemplo, a visualização do fluxo de trabalho em um quadro com cartões representando as tarefas e a limitação do número de cartões por coluna (etapa do fluxo de trabalho)

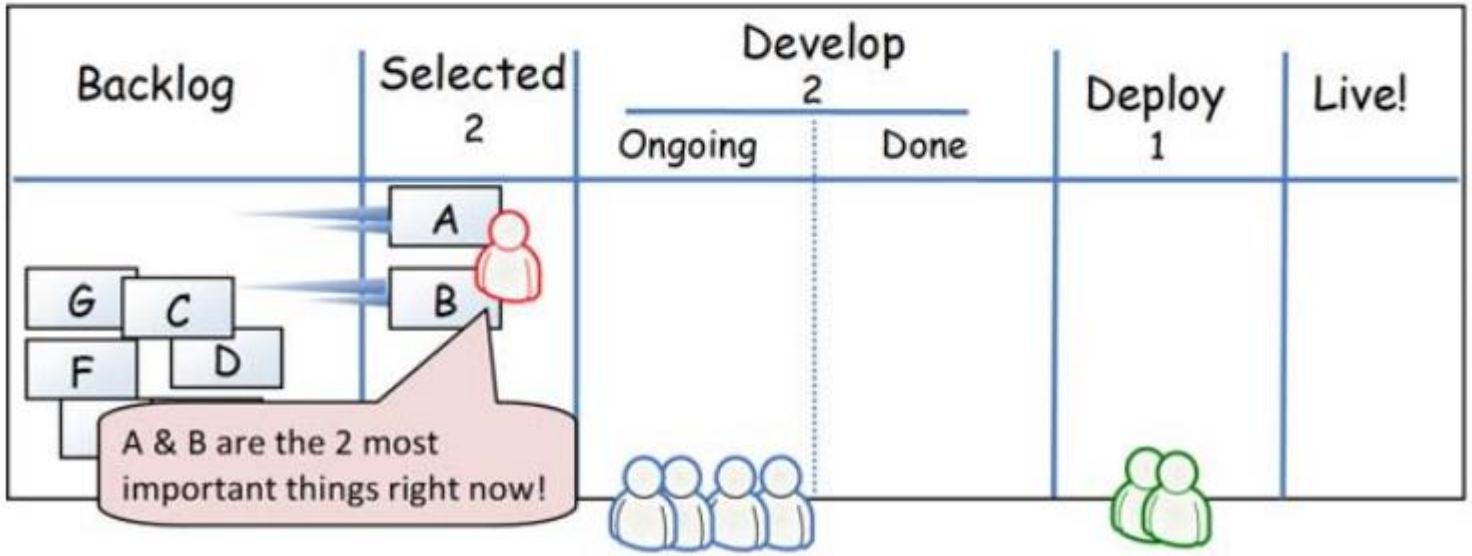
Sistema Kanban para Desenvolvimento de Software

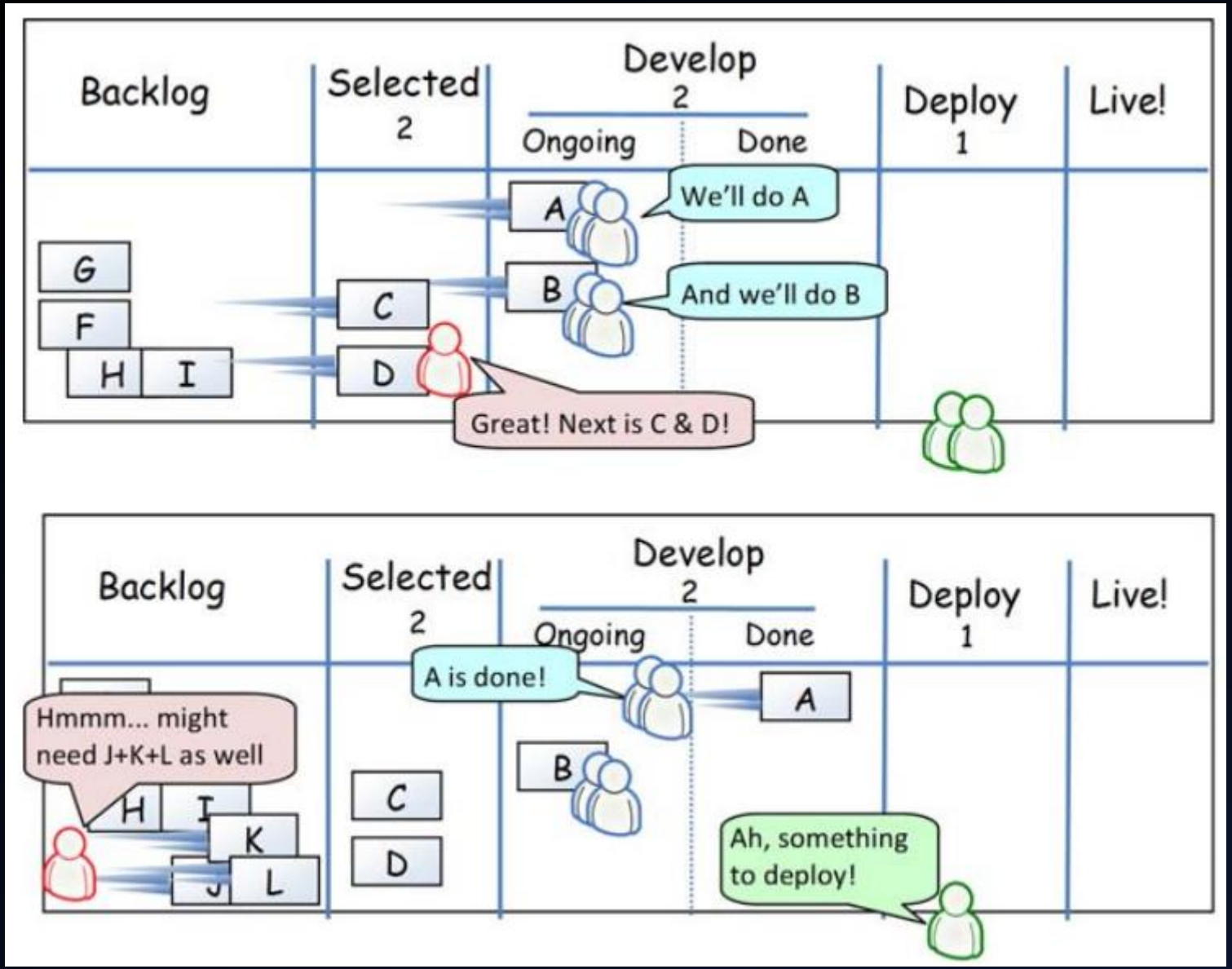
- A limitação do número de tarefas a ser desenvolvida em cada etapa é fundamental para aumentar o número de tarefas encerradas ao invés de iniciar novas tarefas antes de finalizar aquelas que já foram iniciadas
- Considere, por exemplo, o seguinte quadro com as seguintes limitações: deploy (2), test (3), develop (6)
- Enquanto os desenvolvedores protelarem a finalização do deploy, nenhuma outra tarefa poderá ser iniciada (develop), pois nenhuma tarefa em desenvolvimento poderá ir para a etapa de teste

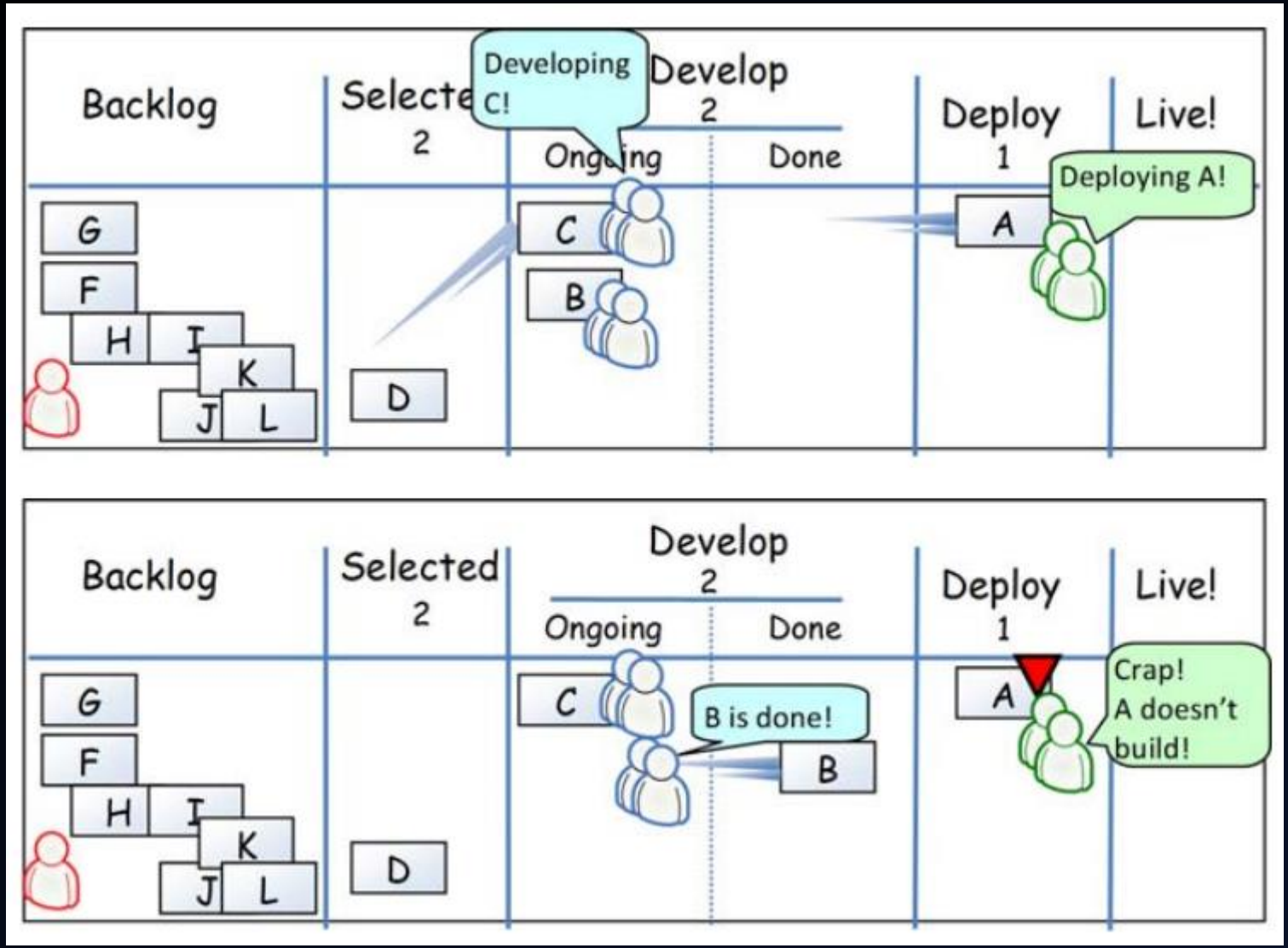


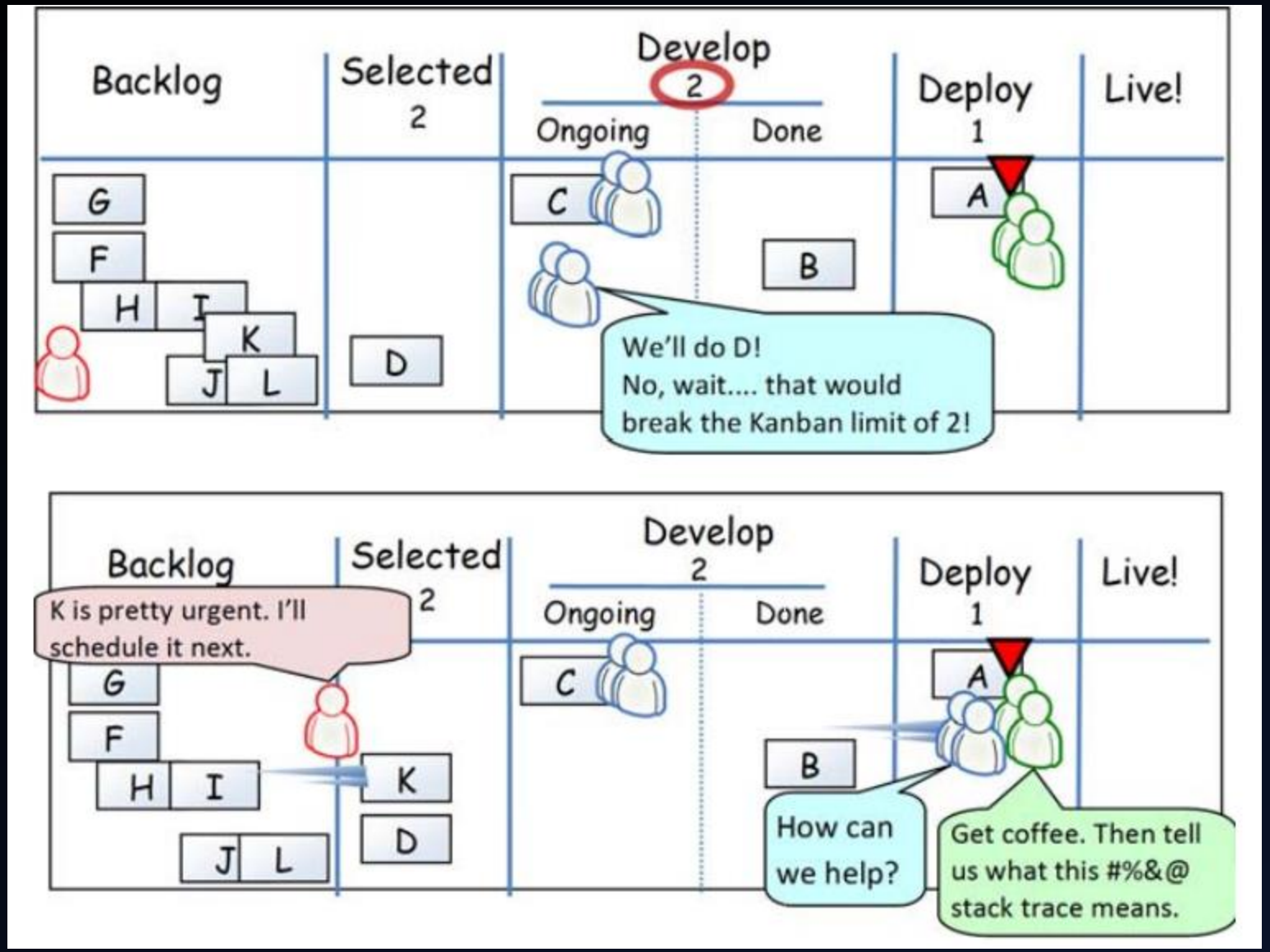
One day in Kanban-land

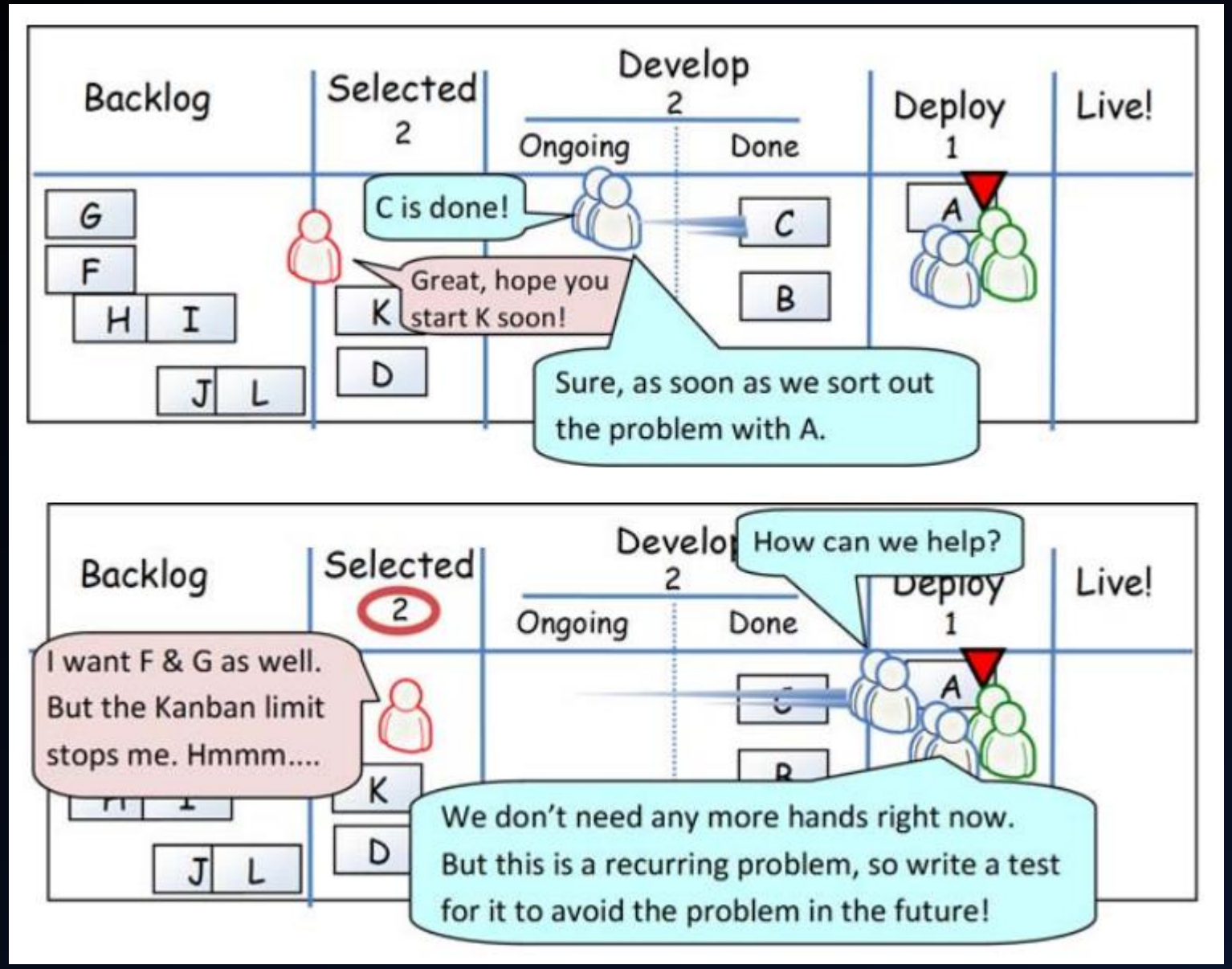


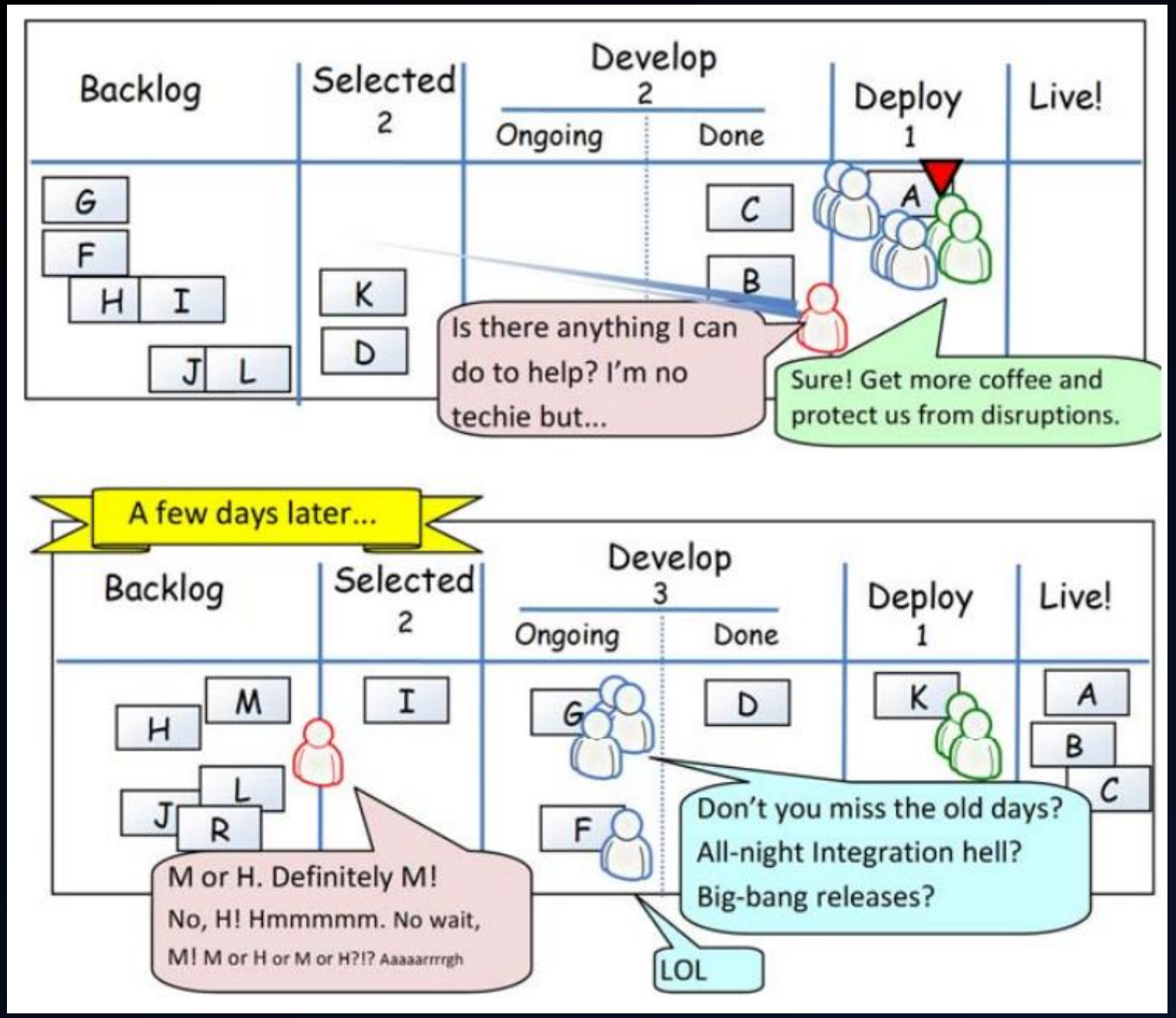



Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
We'll do A
And we'll do B
Great! Next is C & D!
G
F
H
I
C
D
A
B
Hmmm... might need J+K+L as well
A is done!
Ah, something to deploy!
J
K
L


Backlog
Selecte
2
Developing C!
Develop
2
Ongoing
Done
Deploy
1
Live!
Deploying A!
G
F
H
I
K
J
L
D
C
B
A
Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
B is done!
Crap!
A doesn't
build!


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
G
F
H
I
K
J
L
D
C
B
A
We'll do D!
No, wait.... that would
break the Kanban limit of 2!
Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
K is pretty urgent. I'll
schedule it next.
G
F
H
I
K
J
L
D
C
B
A
How can
we help?
Get coffee. Then tell
us what this #%&@
stack trace means.


Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
G
F
H
I
J
L
K
D
C
B
A
C is done!
Great, hope you start K soon!
Sure, as soon as we sort out the problem with A.
Backlog
Selected
2
Develop
2
Ongoing
Done
Deploy
1
Live!
How can we help?
I want F & G as well. But the Kanban limit stops me. Hmmm....
K
D
J
L
A
We don't need any more hands right now. But this is a recurring problem, so write a test for it to avoid the problem in the future!



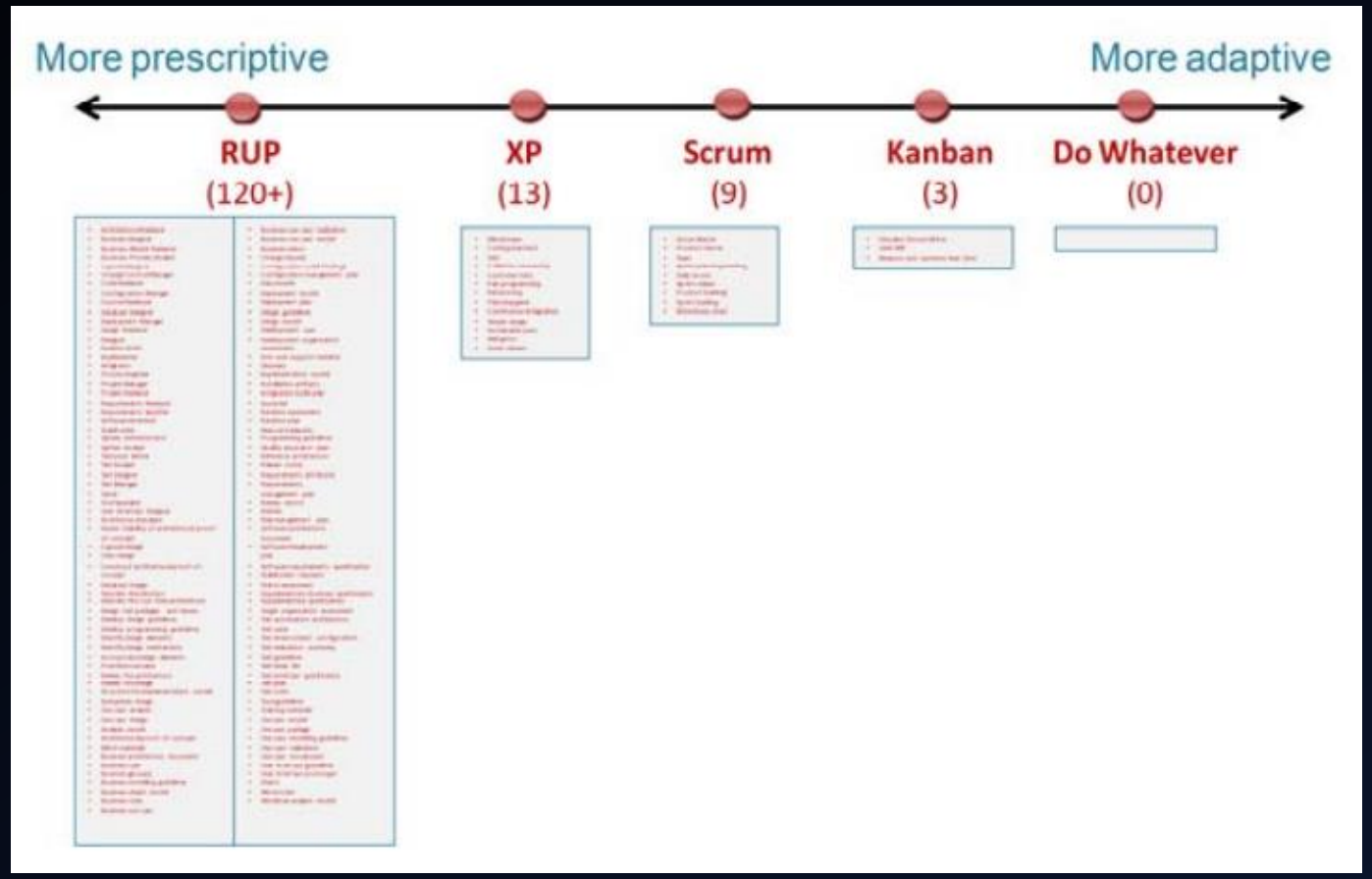
http://www.infoq.com/resource/minibooks/kanban-scrum-minibook/en/pdf/KanbanAndScrumInfoQVersionFINAL.pdf

Scrum e Kanban

- Scrum e Kanban são duas ferramentas para apoiar o processo de software
- A análise de ferramentas em Engenharia de Software deve ser feita para entender seu contexto de aplicação para então decidir se é adequada ou não para o contexto-alvo
- Da mesma forma, Scrum e Kanban são ferramentas distintas, mas que podem ser combinadas

Scrum e Kanban

- Scrum é mais prescritivo do que Kanban



http://www.infoq.com/resource/minibooks/kanban-scrum-minibook/en/pdf/KanbanAndScrumInfoQVersionFINAL.pdf

Scrum e Kanban

- Scrum prescreve papéis
- Kanban não prescreve nenhum papel, mas isso não significa que não é permitido definir papéis. É importante, por exemplo, definir um papel semelhante ao Product Owner do Scrum

Scrum e Kanban

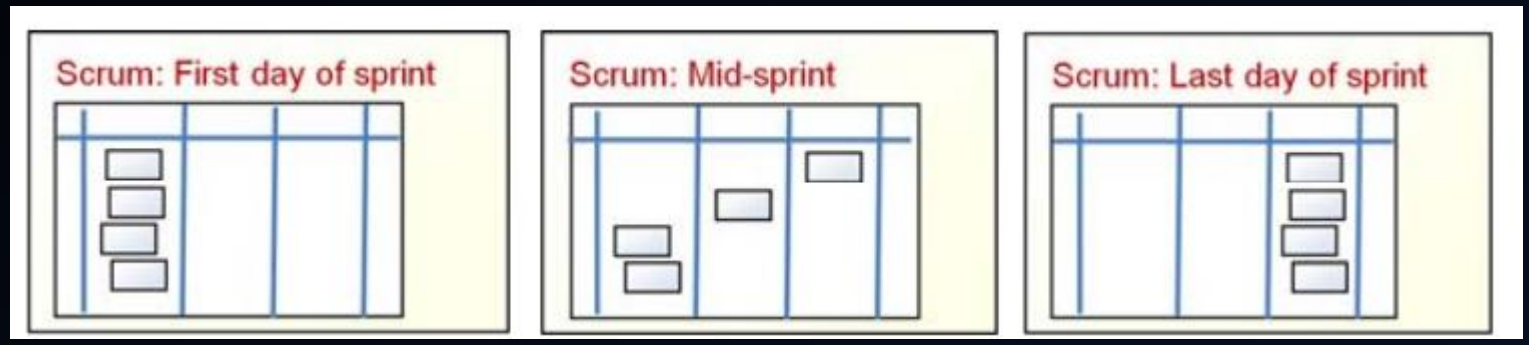
- Scrum prescreve iterações com tempo determinado (1 a 4 semanas), e, idealmente, o resultado de cada iteração deve ser novas funcionalidades prontas para entrar em operação
- Na iteração do Scrum, o conjunto de tarefas a ser realizado é congelado, não pode mudar.
- Kanban não prescreve iterações com tempo fixo. É possível escolher quando fazer planejamento, quando refletir sobre o processo, e quanto liberar uma nova versão do software. É possível escolher fazer essas tarefas regularmente (toda segunda, ou a cada 10 dias), ou sob demanda.

Scrum e Kanban

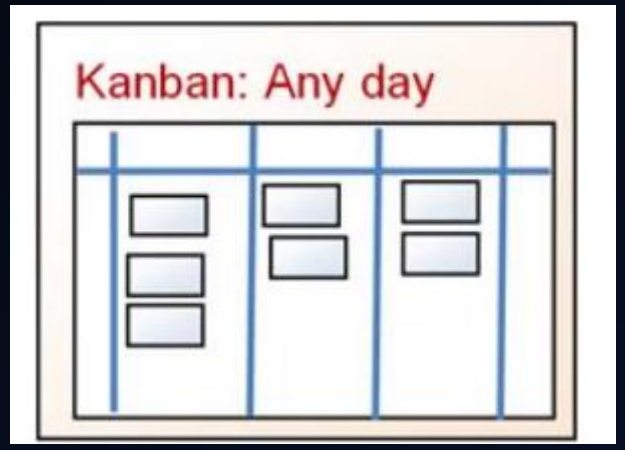
- Kanban limita a quantidade de tarefas possíveis em cada etapa do fluxo de trabalho
- Scrum limita a quantidade de tarefas possíveis em cada iteração (funcionalidades que podem ser desenvolvidas dentro do tempo de uma iteração)

Scrum e Kanban

- O quadro de tarefas do Scrum é “zerado” após cada Sprint

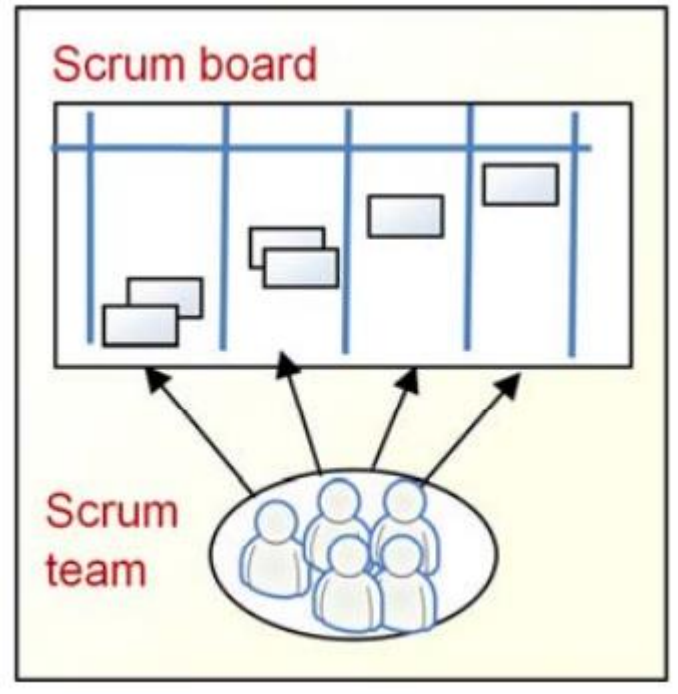


- No Kanban o quadro está continuamente cheio



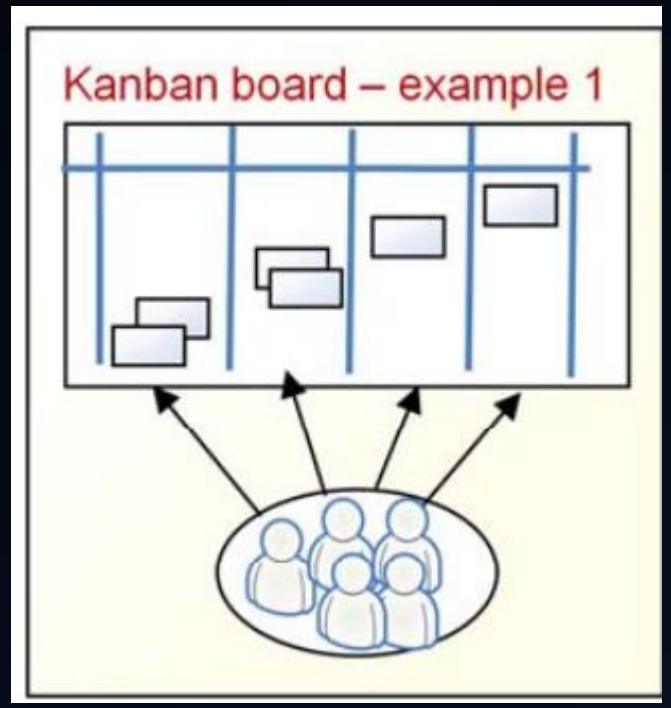
Scrum e Kanban

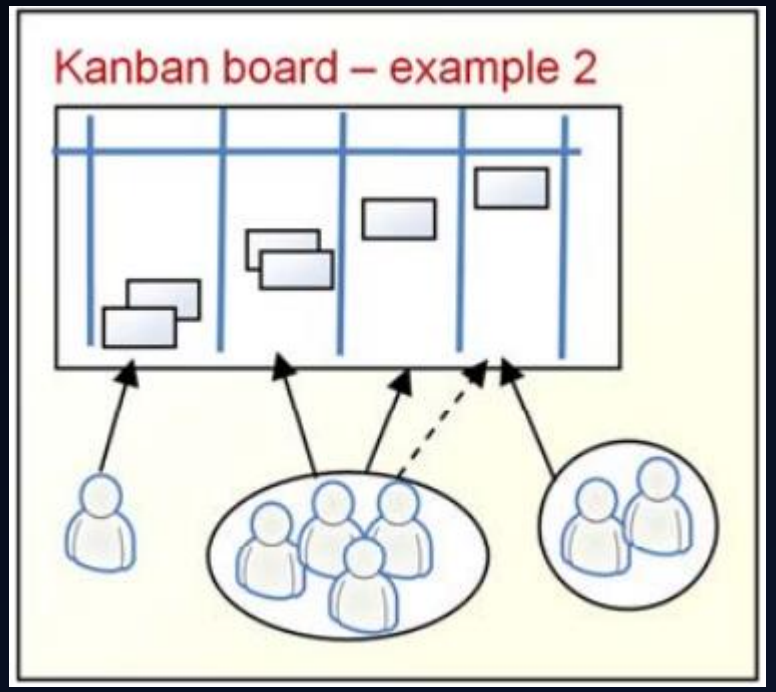
- Scrum prescreve times coesos, em que a equipe possui pessoas para todas as funções básicas (analista, desenvolvedor, tester, etc)



Scrum e Kanban

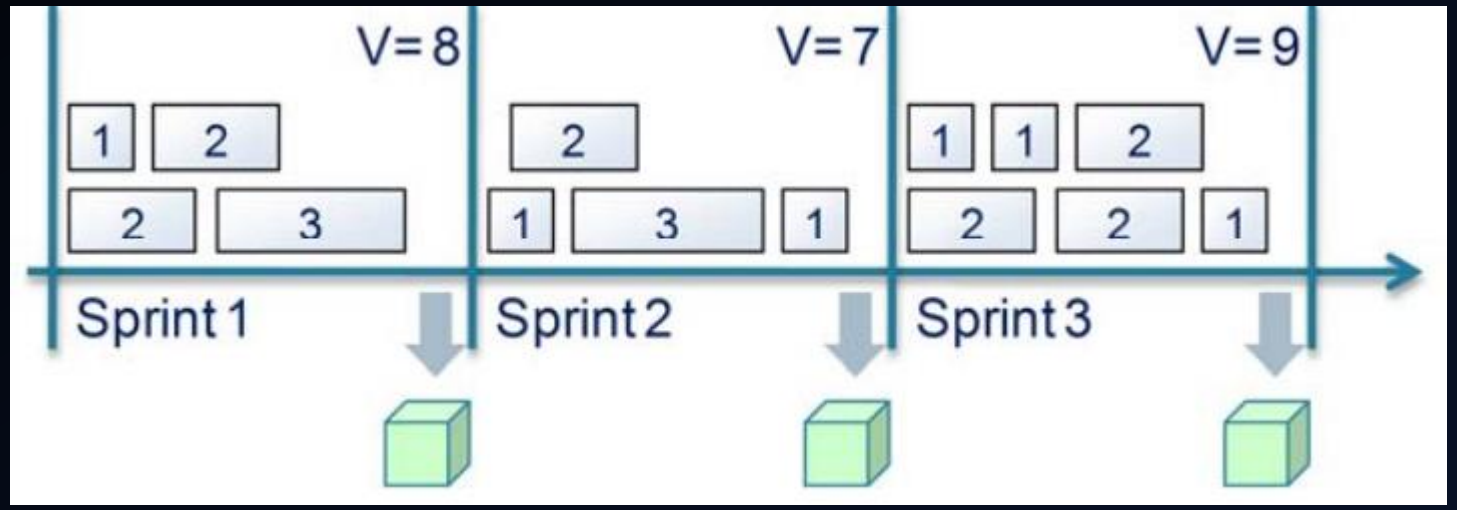
- No Kanban, é possível escolher times que possuem todas as funções necessárias para o desenvolvimento de software, ou times com funções específicas





Scrum e Kanban

- Scrum prescreve a realização de estimativas do “tamanho” de cada funcionalidade e o cálculo de velocidade de desenvolvimento



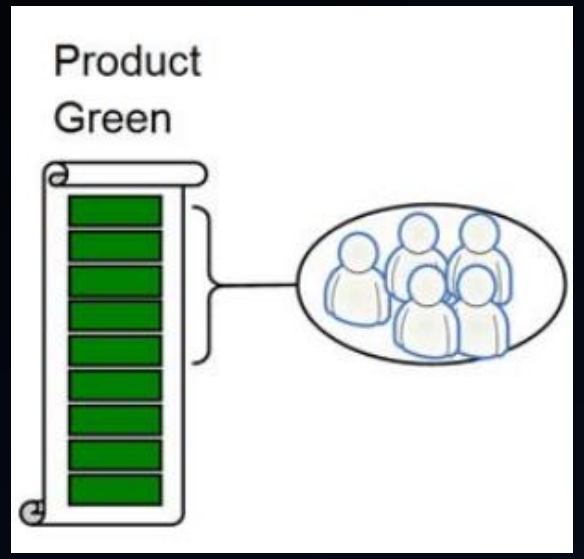
Scrum e Kanban

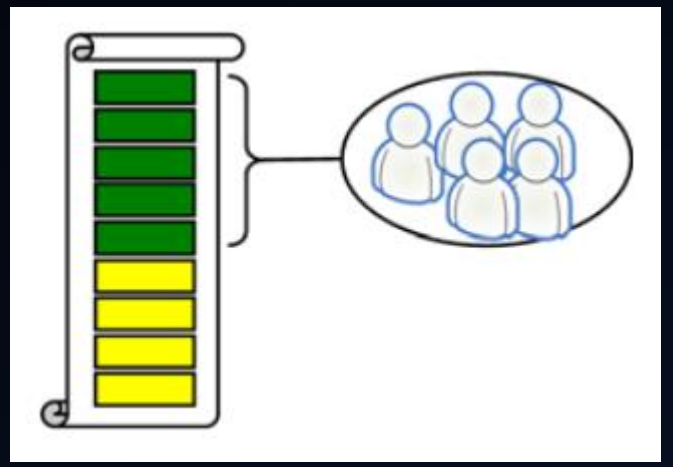
- O Kanban não prescreve estimativas e nem cálculo de velocidade.
- Alguns times kanban escolhem fazer estimativas e cálculo de velocidade assim como os times scrum.
- Outros times tentam não fazer estimativas. Para isso, tentam quebrar todas as tarefas em tamanhos semelhantes para calcular a velocidade apenas sabendo quantas tarefas são desenvolvidas por período de tempo (um dia, uma semana).

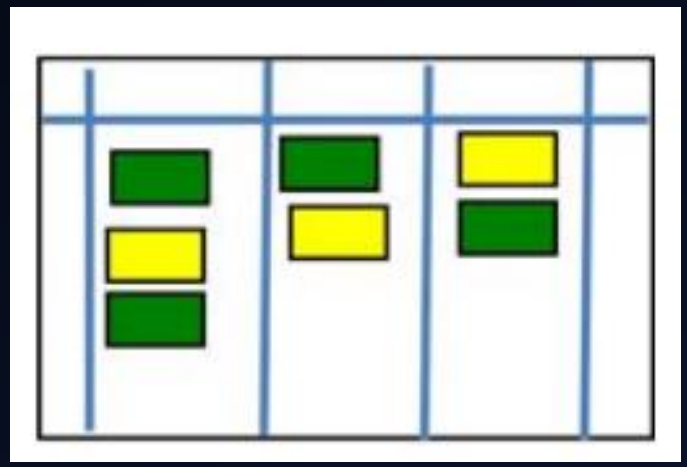
Scrum e Kanban

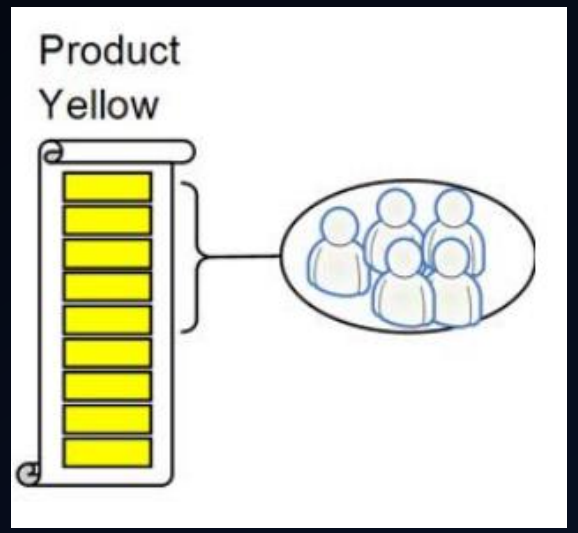
- Ambos permitem que se trabalhe em dois produtos ao mesmo tempo

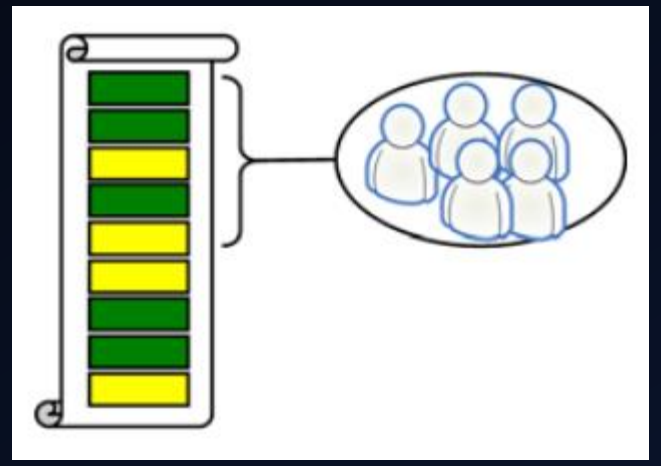
Scrum e Kanban

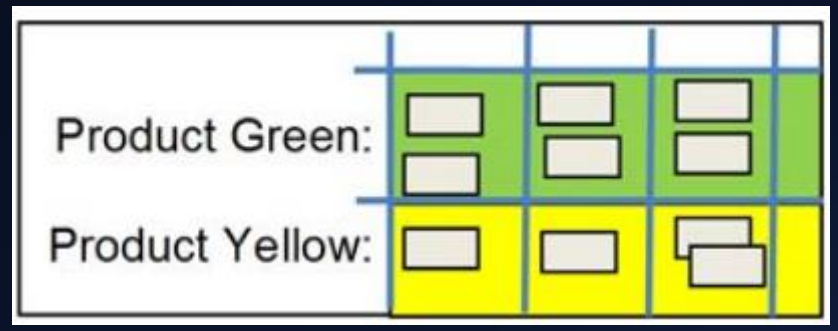












Scrum e Kanban

- Scrum prescreve um backlog priorizado, em que mudanças nas prioridades só tem efeito no próximo Sprint
- Em Kanban, o backlog não precisa ser priorizado. Uma coluna específica pode ser criada para definir quais tarefas foram selecionadas para serem as próximas.
- Assim, no kanban as mudanças nas prioridades tem efeito assim que uma nova tarefa em espera já pode ser colocada em desenvolvimento

Scrum e Kanban

- Scrum prescreve reuniões diárias
- Em Kanban não é prescrita essa atividade, mas muitos times adotam essa excelente prática

Scrum e Kanban

- Resumindo
  - Ambos são ágeis
  - Ambos limitam o número de tarefas em andamento
  - Ambos possuem espaço para melhoria no processo
  - Ambos tem o foco em entregar software pronto para uso o quanto antes e frequentemente
  - Ambos são baseados em equips auto-organizáveis
  - Ambos requerem quebrar o trabalho em pequenas partes
  - Ambos otimizam seu planejamento com base em dados empíritos (velocidade, ciclo de cada tarefa, etc)

Métodos ágeis

- Críticas:
  - A filosofia de desenvolvimento e as rupturas trazidas pelo manifesto ágil receberam muitas críticas quando o manifesto foi publicado e nos anos que seguiram.
  - Por desencorajar o planejamento, documentação, e especificações estabelecidas em contrato, o Manifesto Ágil vai contra a sabedoria convencional dos intelectuais da engenharia de software, então ele não foi universalmente aceito de braços abertos:
  - Dois críticos publicaram críticas contra a filosofia ágil em um livro de 432 páginas:
    - Stephens, Matt; Rosenberg, Doug. Extreme Programming Refactored: The Case Against XP.

Métodos ágeis

- Críticas:
  - *“O Manifesto ágil é uma outra tentativa de minar a disciplina de engenharia de software. Nesta profissão, há engenheiros e há hackers. Esta me parece uma tentativa de legitimar o comportamento de um hacker. A engenharia de software só vai melhorar quando os clientes se recusarem a pagar pelo software que não faz o que está no contrato. A engenharia de software será um disciplina respeitável somente quando a cultura que encoraja o comportamento hacker mudar uma cultura que promove as práticas previsíveis de engenharia de software. (Steven Ratkin, “Manifesto Elicits Cynicism”, IEEE Computer, 2001.)*

Métodos ágeis

- Críticas:
  - Embora a ideia de envolvimento do cliente no processo de desenvolvimento seja atraente, seu sucesso depende de um cliente disposto e capaz de passar o tempo com a equipe de desenvolvimento, e que possa representar todos os interessados (stakeholders) do sistema. Frequentemente, os representantes dos clientes estão sujeitos a diversas pressões e não podem participar plenamente do desenvolvimento de software.
  - Membros individuais da equipe podem não ter personalidade adequada para o intenso envolvimento que é típico dos métodos ágeis e, portanto, não interagem bem com outros membros da equipe.

Métodos ágeis

- Críticas:
  - Priorizar as mudanças pode ser extremamente difícil, especialmente em sistemas nos quais existem muitos interessados. Normalmente, cada interessado dá prioridade a diferentes tipos de mudanças.
  - Manter a simplicidade exige um trabalho extra. Sob a pressão de cronogramas de entrega, os membros da equipe odem não ter tempo para fazer as simplificações desejáveis.
  - Muitas organizações, principalmente as grandes empresas, passaram anos mudando sua cultura para que os processos fossem definidos e seguidos. É difícil para eles mudar para um modelo de trabalho em que os processos são mais informais e definidos pela equipe de desenvolvimento.

Métodos ágeis

- Críticas:
  - O documento de requisitos de um software é normalmente parte do contrato entre o cliente e o fornecedor. Com a especificação incremental é inerente aos métodos ágeis, escrever contratos para este tipo de desenvolvimento pode ser muito difícil. Consequentemente, os métodos ágeis tem que contar com contratos nos quais o cliente paga pelo tempo necessário para o desenvolvimento do sistema, e não pelo desenvolvimento de um conjunto de requisitos.
  - A manutenção de software desenvolvido com métodos ágeis pode ser mais difícil uma vez que não há documentação formal. O contraponto é que métodos ágeis pregam a estruturação simples de código e arquitetura, e seus desenvolvedores estão acostumados a alterar código, mesmo sem documentação extensiva.

Métodos ágeis

- Críticas:
  - Métodos ágeis dependem de os membros da equipe compreenderem os aspectos do sistema sem consultar a documentação. Se uma equipe é alterada, esse conhecimento implícito é perdido, e é difícil para os novos membros da equipe construir o mesmo entendimento do sistema e de seus componentes.

Métodos ágeis

- Durante muitos anos os fiéis defensores das duas filosofias (ágil x tradicional) se “agrediram” com duras críticas, e isso acontece até hoje.
- Mas é importante notar que de 2001 para cá a cultura está mudando e a filosofia ágil tem sido mais bem aceita na indústria de desenvolvimento de software.
- Mesmo tradicionais defensores dos processos baseados em documentação começaram a entender a diferença de natureza do software a mudança no cenário de negócios atual, que em geral requer uma abordagem de desenvolvimento diferente da tradicional.

Métodos ágeis

- Lembram-se do Healthcare.gov (ACA)? Antes do programa ter sido lançado, uma alteração na forma de contratação de software nos EUA havia sido proposta:
  - *"O DOD (Departamento de Defesa) é prejudicado por práticas culturais e de contratação que favorecem grandes programas, alto nível de controle, e uma abordagem deliberadamente serializada para desenvolvimento e teste (modelo cascata). Programas que devem ser completos, perfeitos e que demoram anos para serem concluídos é a norma no DOD. Essas abordagens vão contra as práticas de aquisição ágil em que i) o produto é o foco primário; ii) usuários finais estão engajados no processo; iii) a supervisão do desenvolvimento do produto é delegado ao nível prático; e iv) o time de desenvolvimento tem a flexibilidade de ajustar o conteúdo do incremento do software para atingir a meta em tempo. Abordagens ágeis tem permitido que seus desenvolvedores ultrapassem os gigantes industriais que são assolados por pesados processos e antigos métodos de gerenciamento. Abordagens ágeis conseguiram que seus desenvolvedores reconhecessem os problemas que realmente trazem riscos ao projeto e mudaram suas práticas de gerenciamento e desenvolvimento para eliminar esses riscos" (National Research Concil, 2010)*

Métodos ágeis

- Declaração do presidente Barack Obama após o desastre do lançamento do ACA website (Healthcare.gov):
  - *“Uma das coisas que eu reconheço é que a forma como compramos tecnologia no governo federal é pesada, complicada e atrasada. Isto é parte da razão do porquê, cronologicamente, os programas de TI da federação estão estourando o orçamento e o cronograma. Como agora eu sei que o governo federal não tem sido bom neste quesito, dois anos atrás, quando estávamos pensando nisso, nós deveríamos ter feito mais para garantir que nós estávamos de fato quebrando o modelo de como tradicionalmente fazemos essas coisas”. (presidente Barack Obama, 14 de Novembro de 2013)*

Métodos ágeis

- Ensinando Engenharia de Software com métodos ágeis:
  - Os métodos tradicionais de desenvolvimento podem ser entediantes para algumas pessoas, principalmente para estudantes de um curso de engenharia de software.
  - Por outro lado, os métodos ágeis trazem mais dinamicidade e diversão para o processo.

Métodos ágeis

- Ensinando Engenharia de Software com métodos ágeis:
  - *Você se lembra quando programar era divertido? Não foi por isso que você se interessou por computadores e mais tarde por uma profissão na área? Bem, pode haver muitos promissores e respeitáveis métodos de desenvolvimento adequado para este tipo de pessoa. O estilo ágil é divertido, porque além de não nos afundar em uma pilha de documentação, os desenvolvedores trabalham face a face com os clientes durante todo o processo de desenvolvimento e consegue ter versões funcionando do software mais cedo. (Renee McCauley, "Agile Development Methods Poised to Upset Status Quo", ACM Technical Symposium on Computer Science Education, 2001)*

Métodos ágeis

- Para obter informações mais detalhadas sobre os métodos ágeis, aqui estão os livros recomendados:
  - Kent, B.; Andres, C. Extreme Programming: Explained. 2nd Edition. Addison-Wesley, 2004.
  - Sommerville, Ian. Engenharia de Software. São Paulo Pearson Prentice Hall, 2011. Capítulo 3.
  - Wazlawick, Raul Sidnei. Engenharia de Software – Conceitos e Práticas. Capítulo 4.

Referências

- Wazlawick, Raul Sidnei. Engenhariade Software –Conceitose Práticas. EditoraCampus, 2013. 1a edição.
- Sommerville, Ian. Engenhariade Software. São Paulo Pearson Prentice Hall, 2011. 9a edição.
- Kent, B.; Andres, C. Extreme Programming: Explained. 2nd Edition. Addison-Wesley, 2004.
- Notas de aula do Prof. Dr. Sandro Bezerra, UFPA
- http://manifestoagil.com.br/

SIN 5005 – TÓPICOS EM ENGENHARIA DE SOFTWARE

AULA 02 – MÉTODOS ÁGEIS

Prof. Marcelo Medeiros Eler
marceloeler@usp.br