**Titulo**

**Introdução**

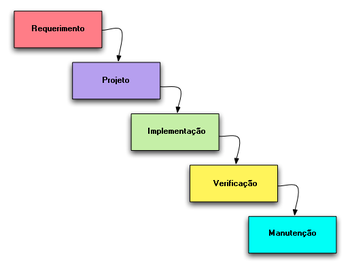
**Objetivos**

**Motivação.**

**Referencial Teórico.**

Modelos Tradicionais

As modelagens tradicionais se caracterizam por serem divididas em fases ou etapas. A maioria das modelagens tradicionais são feitas em cascata, um modelo clássico desenvolvido por Winston W. Royce em 1970 (Royce, 1970; Gustafson, 2002), que apesar de não fornecer feedback entre as fases, não suportar modificações nos requisitos e apresentar o software para o cliente apenas no final do desenvolvimento, é muito útil para a criação de softwares simples e que possuem requisitos conhecidos. De acordo com Ian Sommerville (Engenharia de Software, 2011), as fases do modelo em cascata são: Definição de Requisitos, Projeto de Software e Sistema, Codificação e Testes Unitários, Integração e Testes de Sistema, e Operação e Manutenção.



Fases do modelo em cascata. Fonte: Bruno Marota. Blog Engenharia de Software. Disponível em <http://brunomarota.blogspot.com.br/2012/04/engenharia-de-software-para-concursos\_26.html> Acesso em 14 de nov. 2013.

Processo Unificado

O Processo Unificado é uma tentativa de se aproveitar os melhores recursos dos modelos tradicionais de processos de software, mas adicionando-se alguns princípios do desenvolvimento ágil de software, como por exemplo a importância da comunicação com o cliente e a utilização de casos de uso para melhor compreensão do cliente. Ele também sugere um fluxo de projeto iterativo e incremental, desenvolvendo softwares de grande porte feitos de pequenos sistemas, melhorando a qualidade do projeto.

Fases do Processo Unificado

O Processo Unificado (PU) possui cinco fases metodológicas durante a vida de um sistema de software. Estas fases não ocorrem de forma sequencial, pois enquanto as fases de construção, transição e produção são conduzidas, é provável que já se tenha iniciado o incremento seguinte. Ocorrem de forma concorrente e escalonada.

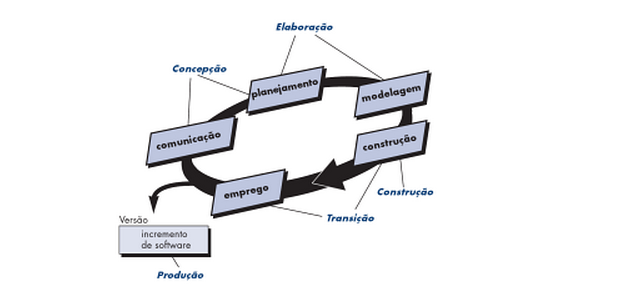
A concepção tem como principal objetivo estabelecer a viabilidade do sistema. Envolve a comunicação com o cliente, juntamente com o planejamento do software. É feito uma arquitetura candidata com os principais métodos, funções e requisitos do projeto. Posteriormente, a arquitetura é expandida para um conjunto de modelos diferentes. O principal marco associado à esta fase são os Objetivos do Ciclo de Vida. No seu planejamento, são levantados quesitos sobre o sistema, como riscos, recursos, custo, cronograma de atividades e uma base para as outras fases na medida que o projeto seja incrementado.

A elaboração envolve a comunicação e modelagem do modelo de processo genérico. Ela refina e expande os casos prático preliminares, desenvolvidos durante a concepção e amplia a arquitetura, adicionando cinco visões diferentes do sistema: modelo de caso prático, modelo de requisitos, modelo de projeto, modelo de implementação e modelo de emprego. Nesta fase, o plano é revisado afim de asseguram que os quesitos continuem razoáveis e algumas mudanças do planejo sejam feitas.

A construção, tendo como entrada o modelo da arquitetura candidata, desenvolve os componentes de software, responsáveis por fazer com que os casos de uso se tornem operacionais para os usuários finais. Para que isso ocorra, os modelos de requisitos e de projeto, que foram iniciados na fase de elaboração, sejam completados para servir de incremento final. Então, são implementados no código-fonte os métodos e funções exigidos para o incremento. Os testes são feitos em cada componente à medida que sejam implementados.

A transição envolve as últimas etapas da fase de construção e a entrega e feedback. É feita a entrega aos usuários finais e o feedback dos usuários em relação a ocorrência de defeitos e mudanças necessárias. Nesta fase, a equipe elabora materiais de informações necessárias para o lançamento da versão. Quando esta fase é concluída, o incremento torna-se uma versão do software utilizável.

Na produção, há o monitoramento do uso contínuo do software, disponibilidade de suporte para o ambiente operacional, a realização e avaliação dos relatórios de defeitos e mudanças.



Esquema do Processo Unificado. Fonte: PRESSMAN, 2011, p. 73.

Workflows do Processo Unificado

Cada workflow é uma sequência de atividades executadas por várias pessoas, seguindo padrões definidos para um entendimento mútuo entres os colaboradores.

O workflow do Processo Unificado está dividido em algumas subseções:

**\*\*Workflow de requisitos\*\* - p. 29 O processo unificado explicado, Kendall Scott.**

As principais atividades de workflow de análise visam construir modelo de análise, ajudando os programadores a melhorar os requisitos funcionais descritos no modelo de casos de uso. Este modelo de análise contém realizações de caso de uso mais apropriadas aos workflows de projeto e implementação do que o modelo de casos de uso.

O workflow de projeto tem como principais atividades a construção de um modelo de projeto, onde é descrito as realizações físicas do caso de uso a partir deste modelo e do modelo de análise. O modelo de projeto serve de base para o modelo incremental, além de definir a organização física dos métodos computacionais do sistema.

As atividades do workflow de implementação objetivam a produção do modelo de implementação, o qual serão descritos como os elementos do modelo de projeto, pedaços de código-fonte e componentes executáveis se tornarão componentes do software.

No workflow de testes é feito um modelo de teste. Neste modelo, são descritos como os testes de integração e de sistema se tornarão componentes executáveis a partir do modelo de implementação. Também é definido como esses testes serão feitos e o resultado de todos os níveis de teste.

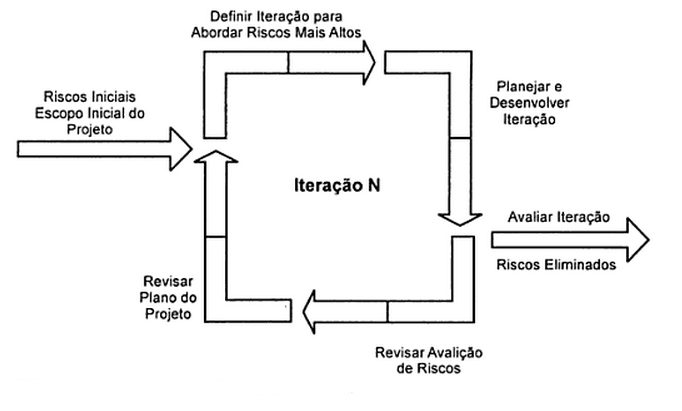
Iterações e incrementos

Iterações são miniprojetos que constituem cada fase do Processo Unificado. Uma iteração tradicional cruza os cinco workflows do PU. Por exemplo, uma interação de uma fase pode estar fortemente envolvida com uma ou várias atividades de workflows diferentes.

Um incremento resulta de cada iteração. Nele contém a versão da funcionalidade melhorada em relação à anterior.

Desenvolvimento iterativo e incremental

Segundo Kandall Scott (2003, p.31), o processo de desenvolvimento iterativo e incremental inicia-se “avaliando os riscos relevantes, inclusive os associados e requisitos, práticas, tecnologias e políticas, e assegurando que o escopo do projeto seja definido para a satisfação de todos os interessados.” A seguir, são realizados os seguintes passos: Prioridade à iteração considerada mais difícil; Planejar detalhadamente a iteração; Desempenhar as tarefas adequadas à iteração (neste caso, os workflows do Processo Unificado); Analisar, ao término da iteração, os resultados do incremento; Desconsiderar os riscos resolvidos pelo incremento e atualizar a lista de riscos de análises pendentes; Revisar o plano do projeto em resposta ao resultado da iteração; Avançar para a próxima iteração.



Desenvolvimento iterativo e incremental. Fonte: SCOTT, 2003, p.32.

Métodos ágeis

São um conjunto de práticas e metodologias que visam otimizar o projeto de software através de incrementos. Também melhoram o tempo de entrega ao cliente, pois este estará sempre recebendo os incrementos feitos pela equipe responsável. Além disso, esta metodologia se preocupa com a relação das pessoas, implicando em uma maior e melhor comunicação entre cliente e desenvolvedor.

De acordo com os desenvolvedores ágeis da organização, possuem os seguintes princípios:

"Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face. Software funcionando é a medida primária de progresso. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente. Contínua atenção à excelência técnica e bom design aumenta a agilidade. Simplicidade--a arte de maximizar a quantidade de trabalho não realizado--é essencial. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo."

Retirado de Princípios por trás do Manifesto Ágil. Disponível em <http://www.agilemanifesto.org/iso/ptbr/principles.html> Acesso em 9 nov. 2013.

Nesta metodologias ...

Antes de falar das metodologias, explicar os benefícios, citas as metodologias e depois explicar cada uma.

Scrum

A abordagem Scrum é um método ágil geral, mas por não fornecer práticas de programação, pode ser usado para fornecer um framework de gerenciamento do projeto(desenvolvido pela MPL, é baseado nas práticas descritas no PMI (Project Management Institute) e integra o gerenciamento de Recursos, Prazos, Custos, Escopo, Riscos, Qualidade, Entregas e Comunicação.). É composto por: o Time, formado pela equipe de desenvolvedores, o Scrum Master, que auxilia e orienta a equipe, e o Product Owner, que define os itens que compõem o Backlog do Produto e os prioriza.

Fases do Scrum

Esse framework possui três fases:

O planejamento, onde é definida a lista de Backlog do Produto, que são requisitos e necessidades apresentados pelo cliente acompanhada de uma revisão e ajustes dos itens do Backlog do Produto. Nesta fase também são feitos o mapeamento dos elementos do Backlog do produto para os itens do Backlog do Sprint, a seleção do Time e das ferramentas de desenvolvimento.

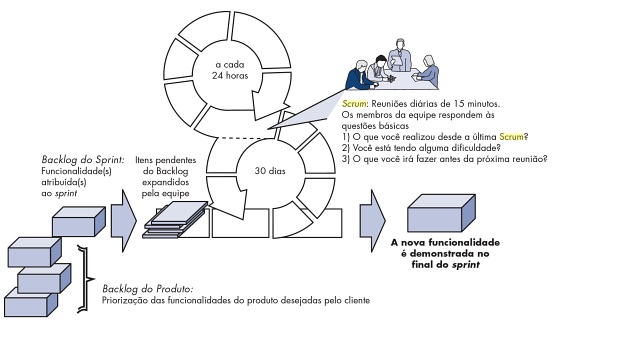
Os ciclos de Sprint, que duram de 15 a 30 dias e conta com uma série de reuniões: de planejamento, reuniões diárias, de revisão e retrospectiva. Na reunião do primeiro planejamento do Sprint é definida a meta da Sprint, as funcionalidades e prioridades do Backlog de Produto e o Planning Poker, que é a estimativa de custo feita pelo Time. Em seguida, é feita a reunião do segundo planejamento do Sprint, decidindo-se como será feito os incrementos do Backlog de Produto.

As reuniões diárias normalmente duram 15 minutos, onde cada membro do Time descreve o que foi feito desde a última reunião diária, dificuldades encontradas e o que planeja fazer antes da próxima reunião.

No final do Sprint, é feita a reunião de revisão do Sprint, onde o Time apresenta o resultado do Sprint ao Product Owner, que avaliará cada história feita, podendo ou não aprova-la e informa se a meta do Sprint foi atingida ou não.

A reunião de retrospectiva do Sprint é realizada para se discutir o que aconteceu durante o Sprint, levantando os pontos positivos e negativos, visando melhorar o desempenho para o próximo Sprint.

Em sua última fase, o Scrum é encerrado e seu projeto é concluído e entregue ao Product Owner.



Fases do Scrum. Fonte: PRESSMAN, 2011, p.96.

Extreme Programming

A XP é uma metodologia ágil usada principalmente por pequenas e médias empresas de software que desenvolvem projetos que estão em constante mudança.

Valores da XP

Este modelo ágil possui cinco valores básicos para o seu trabalho. São eles: comunicação, simplicidade, feedback, coragem e respeito.

A comunicação é essencial para o desenvolvimento de um projeto que atenda às especificações do cliente. Por isso, a XP enfatiza a colaboração entre desenvolvedores e clientes, apresentando conceitos importantes em linguagem de alto nível e realizando feedback constantemente.

Para atingir a simplicidade, a XP limita os programadores a lançar apenas as necessidades julgadas mais importante. O objetivo desta ação é produzir um projeto mais simples possível e de fácil implementação.

O feedback dá-se de três fontes: do software implementado, do cliente e da própria equipe. Através dos resultados obtidos nos testes, um feedback é enviado ao grupo. A forma que o software incrementa os requisitos do caso de uso é um tipo de feedback. E o feedback dado ao cliente sobre alterações do custo e cronograma em resposta às novas mudanças feitas pelo mesmo.

A equipe XP ágil deve ter coragem para pôr em prática ideias inovadoras, mesmo pondo em risco o futuro do incremento e, consequentemente, do software.

Além de coragem, o respeito é fundamental em todos os aspectos: entre seus membros e outros envolvidos (incluindo o próprio software).

Metodologia XP

A Programação Extrema tem como paradigma de desenvolvimento a abordagem orientada a objetos. Possui quatro atividades metodológicas: planejamento, projeto, codificação e testes.

A atividade de planejamento inicia com o ato de ouvir e entender o projeto solicitado pelo cliente. A seguir, é feita uma série de histórias que descreve as características, funções e resultados requisitados pelo cliente. Para cada história é atribuído pelo cliente um valor de prioridade e o custo de tempo para desenvolvimento, que é feito pelos desenvolvedores.

Clientes e desenvolvedores decidem juntos quais histórias serão inclusas, a ordenação das mesmas de acordo com sua importância, e a data prevista de entrega para a versão do software (também chamado de incremento de software). Quando o primeiro incremento de software é entregue, os desenvolvedores calculam a velocidade do projeto, e baseados no resultado, levantam a data de entrega final.

Ao longo do desenvolvimento do projeto, o cliente pode adicionar, modificar o valor ou eliminar as histórias quando desejar, e os desenvolvedores mudam seus planos de acordo.

O projeto segue o paradigma de preservar a simplicidade: quanto mais simples o projeto, melhor. O projeto apenas oferece um guia de implementação para cada história.

A XP aconselha o uso de cartões CRC (classe-responsabilidade-colaborador) servem para identificar e organizar as classes orientadas a objetos importantes para o incremento em desenvolvimento.

Se houverem problemas de projeto de difíceis soluções, a XP recomenda a criação imediata de um protótipo operacional da parte problemática do projeto. Denominada solução pontual, este protótipo é avaliado e implementado pelos desenvolvedores, diminuindo futuros problemas com o projeto original. Também sugere, em alguns casos, a refatoração do sistema, ou seja, a modificação do sistema internamente mas sem alterar sua essência.

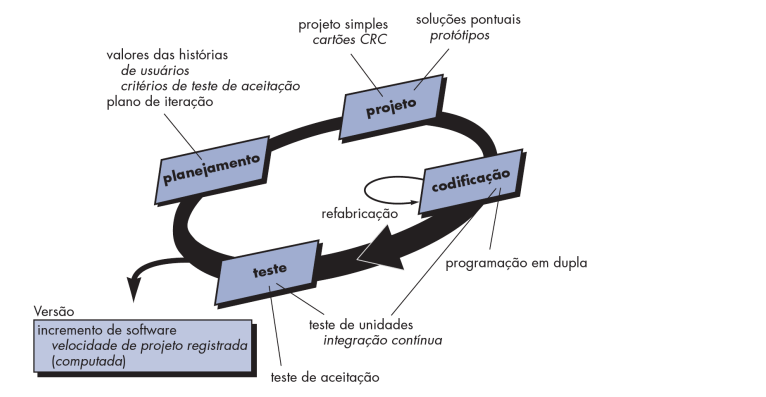
Antes da codificação, a equipe desenvolve um conjunto de teste de unidades para cada história que será inclusa no incremento, mantendo o foco no que deve ser implementado para a aprovação nos testes.

A codificação é feita por programação em duplas. Dois desenvolvedores ficam em uma mesma estação aprimorando a mesma história. Desta forma, o código é revisado e resolvido ao mesmo tempo em que é escrito. Quando a dupla conclui o seu trabalho, o código feito é integrado ao trabalho das outras duplas.

Assim que um código é feito, passa imediatamente pelos testes de unidade. Os testes de unidade devem ser automatizados através de metodologias, para ser usado de forma fácil e repetida.

Os testes de integração e validação do sistema são feitos diariamente, prevenindo graves problemas próximos à data de entrega.

Os testes de aceitação – ou testes de cliente, são feitos pelo cliente e focam nas funcionalidades e características do sistema que são apresentadas ao comprador através de histórias implementadas.



Esquema da metodologia da Programação Extrema. Fonte: PRESSMAN, 2011, p.88.

Lean para Desenvolvimento de Software (LSD)

O LSD é uma modelagem ágil que surgiu a partir do livro "The Machine That Changed the World: The Story of Lean Production”. Foi inspirado em alguns princípios do modelo de produção japonês Toyotismo (just-in-time, qualidade total, melhoria contínua e flexibilidade). Tem como principais objetivos evitar desperdícios no código-fonte do software em desenvolvimento, envolver as pessoas ao máximo durante o processo, valorizar a qualidade do produto e a necessidade do cliente.

Princípios do LSD

Os principais princípios seguidos no Lean aplicados à software são:

Eliminar Desperdícios: Tudo que não é valorizado pelo ciente é considerado desperdício. Pode incluir códigos e funcionalidades desnecessários, atraso no processo de software, exigências pouco claras, testes insuficientes, etc. Assim, a fim de evita-los, é preciso monitorar as atividades. A técnica de mapeamento de fluxo de valor serve para apontar e identificar o que for considerado desperdício. Depois disso, é revelada as fontes do desperdício para então eliminá-los.

Incluir Qualidade no Processo: Qualidade está sempre em primeiro lugar e é inegociável. A qualidade tem que estar presente no interior e no exterior do software, objetivando o reconhecimento de qualidade do projeto pelo cliente. Para garantir essa característica, são feitas algumas práticas, como: verificar a qualidade durante todo o processo, manter o foco na prevenção, corrigir problemas urgentemente, usar o TDD (Test Driven Development), refatorar o código sempre que possível, ter a integração, revisão, inspecionamento e testes automatizados do código continuamente.

Criar Conhecimento: A melhor abordagem para melhorar o desenvolvimento de software é ampliando o conhecimento de toda a equipe envolvida, de forma compartilhada entre seus membros. Práticas como ciclos de feedback, inspeções e adaptações, desenvolvimento interativo, equipe formadas por poucos membros, treinamentos, revisão de código e a utilização de Blog ou Wiki como meios de compartilhamento de informação.

Adiar Decisões e Compromissos: Como o objetivo de diminuição das incertezas, aconselha-se adiar as decisões até que possam ser feitas baseadas em fatos concretos, não em suposições. Para isso, usam-se práticas que permitam conter as decisões tardiamente, como interações, reuniões de planejamento, desenvolvimento orientado a características (BDD – Behaviour/Features Driven Development), entre outros.

Entregar Antecipadamente: Rápidas entregas garantem o colhimento do feedback do cliente do que foi entregue, correção de problemas sem aumentar o custo para o cliente, aprendizagem com as falhas encontradas e garantia da satisfação do cliente. Kanban, interações e simplicidade são ações sugeridas para a entrega ocorrer mais rapidamente.

Respeitar Pessoas e Decisões das Equipes: Uma empresa que se preocupa com o respeito, valorização e participação das pessoas é bem vista pelos seus funcionários e seu tratamento é retribuído pelos mesmos. Quanto melhor o tratamento que as pessoas de uma equipe têm pela organização, maior será seu desempenho, responsabilidade e envolvimento com o desenvolvimento de projetos. Auto-gestão e trabalho em equipe melhoram bastante a relação entre os envolvidos.

Otimiza o Todo: A otimização deve ser feita do início ao fim, utilizando métricas de desempenho e análises do desempenho da equipe para melhorar o processo, medindo o tempo das atividades um pouco acima do que se acha para não gerar atrasos, e valorizando principalmente a satisfação do cliente.

Kanban

De acordo com Flavio Mariotti, que teve seu artigo publicado na Revista Engenharia de Software nº 45, a forma mais comum de observar um fluxo de trabalho é usando o Kanban. Ele é considerado framework, usado para implementação de mudanças de forma incremental. É um tipo de quadro de atividades que pode físico, em que é colocado em uma das paredes local de trabalho, ou virtual, online.

No kanban para desenvolvimento de software, é feito inicialmente todos os workflows de trabalho, ou seja, as etapas para a produção do projeto. Em cada workflow, devem ser colocados cartões ou notas autoadesivas detalhando a atividade que está sendo executada, que a está executando e data de início.

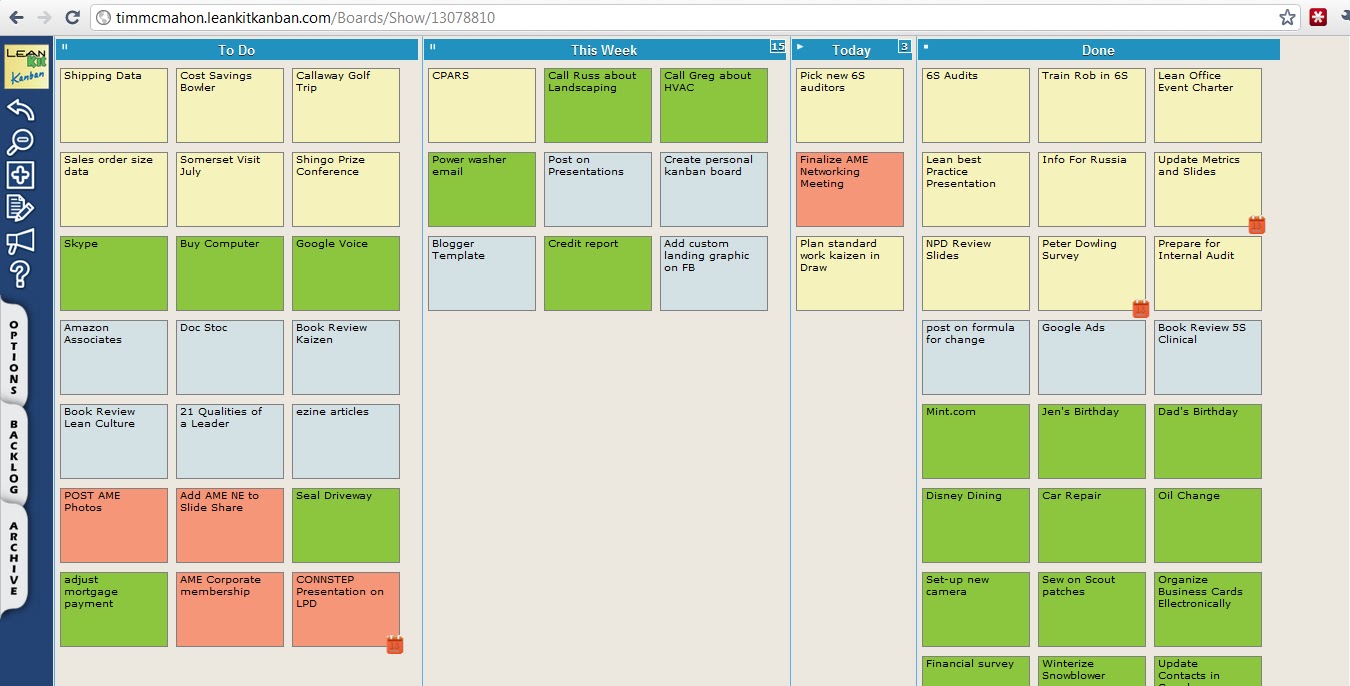
Para limitar a quantidade de trabalho em andamento, é necessário definir e monitorar um limite máximo de tarefas para cada etapa.

Estabelecido os limites, a equipe identifica os principais impedimentos do processo de produção e precisam terminar todo o trabalho inacabado para conseguir mais trabalho. Com isso, o trabalho torna-se um sistema puxado, onde é a equipe que dita a sua capacidade de produção, permitindo a implementação de mudanças no processo de modo incremental.

Outra vantagem do kanban é que uma pessoa execute apenas uma tarefa por projeto. Além desta, tem destaque o processo Just in time, responsável por evitar a acumulação de requisitos que não foram liberados para o cliente ao longo do desenvolvimento.



Sistema Kanban. Fonte: Alisson Vale - Blog Just in Time for Knowledge Work. Disponível em <http://alissonvale.com/englishblog/post/2008/08/26/A-Historia-de-um-Sistema-Kanban.aspx> Acesso em 09 nov. 2013.



Kanban online. Fonte: Tim McMahon - Blog A Lean Journey: New Sponsor. Disponível em <http://www.aleanjourney.com/2012/03/new-sponsor-leankit-kanban.html> Acesso em 09 nov. 2013.

Análise comparativa

Conclusão

REFERÊNCIAS BIBLIOGRÁFICAS:

* PRESSMAN, Roger S. **Engenharia de Software**: Uma abordagem profissional. 7ª Edição. São Paulo: McGraw Hill Brasil, 2011.681p.
* SCOTT, Kandall. **O Processo Unificado Explicado**. Porto Alegre: Bookman, 2003. 160 p.
* MARIOTTI, Flavio S. **Engenharia de Software Magazine- Kanban**: o ágil adaptativo
* https://www.ibm.com/developerworks/community/blogs/rationalbrasil/entry/lean\_para\_desenvolvimento\_de\_sw\_o\_que\_\_c3\_a9\_isso\_afinal12?lang=en
* http://igorrocha.com.br/lean-software-development/