



Métodos Ágeis em TI

UNIDADE 05

Lean

Nesta Unidade, estudaremos a linha de pensamento lean. Ela influencia toda a linha de pensamento ágil e, de forma mais forte, três métodos ágeis importantes.

Lean

A linha de pensamento *lean* tem origem no sistema de produção da Toyota, descrito por Taiichi Ohno, em 1978. Esse sistema tinha como base certas características: eliminar desperdícios, produção sem estoque, exposição rápida de problemas e qualidade. Tais

características fazem com que esse sistema seja puxado, tornando o fluxo das atividades contínuo e maximizando o valor agregado para o cliente. O sistema evolui por meio de constante inspeção e ações de melhoria contínua.

Os autores precursores em adaptar a linha de pensamento lean para o desenvolvimento de software são Mary e Tom Poppendieck. A Figura 1, apresentada a seguir, mostra a trilogia de livros deles sobre o assunto.

Figura 1: *Lean software development*



Trilogia de livros sobre o desenvolvimento de software *lean*. Fonte: Poppendieck (2015).

Serão descritos, na sequência, os princípios *lean* aplicados ao desenvolvimento de software.

Princípio 1 – elimine desperdícios

“

Desperdício é tudo aquilo que não agrupa valor ao cliente. (Ohno, 1997)

Este é o principal princípio *lean*. Somente software funcionando é o que vai trazer valor ao cliente.

Serão descritos a seguir os sete desperdícios do desenvolvimento de software.

Trabalho incompleto (em progresso)



Artefatos inacabados consomem recursos sem trazer retorno. Alguns exemplos de trabalho incompleto são:

- Requisitos especificados muito cedo.
- Documentação não codificada.
- Código não sincronizado com a ferramenta de controle de versão.
- Código não testado, tanto de forma manual quanto de forma automatizada.
- Código não implantado em produção.

Processos a mais



Para identificar processos excedentes no seu ambiente é bom sempre se perguntar: Existe algo ou alguém esperando pelo que está sendo produzido? Exemplos de processos a mais são:

- Burocracia desnecessária.
- Documentação desnecessária.
- Atividades de gerenciamento.

Funcionalidades a mais



Pesquisas indicam que uma boa parte das funcionalidades de software desenvolvidas nunca são ou são raramente utilizadas pelos clientes. Porém, todo o código dessas funcionalidades torna o produto mais complexo, dificultando a manutenção desse software.

Troca de tarefas



Isso já é senso comum. Ter muitas atividades ao mesmo tempo é prejudicial para o nosso desempenho. Porque a cada troca de contexto, entre uma atividade e outra, perdemos tempo e energia para retomar a atividade de onde paramos. O ideal é termos o menor número de atividades simultâneas, tanto individualmente quanto no time.

Repasses (handoffs)



Conhecimento é difícil de transmitir. Portanto, quanto mais repasses, maior é a perda de conhecimento. Dicas para melhorar o desempenho do time: reduza o número de repasses de atividades ao mínimo possível, use meios de comunicação eficazes, preferencialmente presencialmente e entregue partes do trabalho para revisão o mais cedo possível.

Atrasos



Desenvolvedores de software precisam tomar decisões frequentemente. É impossível assumir que toda informação necessária estará disponível no momento da decisão. Portanto, a cada decisão, deve-se rapidamente percorrer esse ciclo: tentar descobrir as informações necessárias para a decisão, avaliar a possibilidade de alternar para outra tarefa que já possua todas as informações necessárias ou usar seu melhor julgamento e estimar as informações faltantes e prosseguir.

Exemplos comuns de atrasos são causados por esperas de:

- Entendimento completo dos requisitos.
- Aprovação do projeto.
- Alocação das pessoas.
- Disponibilidade das pessoas alocadas.
- Processo de controle de alterações.
- Sistema inteiro ficar completo para ter as funcionalidades-chave.

- Execução de testes.
- Comunicação de defeitos (testes somente no final do projeto).

Defeitos



O custo da resolução dos defeitos aumenta com o tempo. Portanto, equipes ágeis se esforçam ao máximo para encontrá-los o mais rápido possível. O uso da prática desenvolvimento orientado a testes do XP é um exemplo disso. Testes automatizados são um investimento bastante rentável no decorrer do projeto. E, em caso de defeito, é importante fazer o máximo para resolver a causa raiz do problema.

Princípio 2 – inclua a qualidade no processo

“

Inspecionar para prevenir defeitos é bom; inspecionar para encontrar defeitos é desperdício. (Shingo, 1996)

Um processo de qualidade inclui qualidade sempre, em vez de deixar para depois o teste da qualidade do produto. Se você encontra defeitos somente no fim do seu processo, ele é ineficiente. Não deixe os testes para o final, pois ciclos de teste muito longos geralmente gastam mais tempo corrigindo defeitos. Em vez de se esforçar para gerenciar defeitos, evite-os.

Boas práticas de qualidade incluem:

- Criar código que revele a intenção.
- Revisar seu *design* e código.
- Criar testes automatizados.
- Parar o desenvolvimento se seus testes não passarem.
- Usar integração contínua.
- Analisar por que defeitos ocorreram durante o processo.

Princípio 3 – crie conhecimento

Incentive o compartilhamento de conhecimento. O processo deve ser continuamente melhorado. Promova melhoria contínua do seu processo usando ciclos como:

- Enquadrar o problema.
- Procurar pela raiz do problema.
- Propor uma solução.
- Implementar a solução.
- Verificar os resultados
- Analisar e adaptar seus padrões.

Princípio 4 – adie comprometimentos

Decisões irreversíveis devem ser tomadas o mais tarde possível (*last responsible moment*). Para isso, invista em buscar opções: avaliem tecnologias, construam protótipos e promovam discussões. Tudo isso trará mais informações para quando as decisões tiverem de ser tomadas.

Princípio 5 – entregue rápido

“

A moral da história é que devemos encontrar uma maneira de entregar software tão rápido, que nossos clientes não tenham tempo de mudar de ideia. (Poppendieck, 2003)

Competir com base na velocidade traz grande vantagem competitiva. Boas práticas para entregar rápido incluem:

- Normalizar a entrada de trabalho.
- Minimizar o número de atividades em andamento simultaneamente.
- Minimizar o tamanho das atividades.
- Estabelecer uma cadência regular.
- Limitar o trabalho à capacidade do time.

Princípio 6 – respeite as pessoas

Isso também é senso comum: as pessoas são o ativo mais importante das empresas. Times *lean* possuem líderes que entendem que as pessoas possuem conhecimento para executar suas atividades, portanto, planejam e controlam as atividades considerando que as pessoas tenham responsabilidade. As pessoas são motivadas em times que cuidam de seus propósitos, participação, segurança, competência e progresso.

Princípio 7 – otimize o todo

É preciso olhar para o processo todo. Não adianta resolver os sintomas, é preciso resolver as causas. E a cada melhoria proposta ao processo, é necessário voltar a olhar para o processo todo. Isso garante que uma melhoria em uma etapa do processo não piora o processo.

Agora, você saberá mais da história dessa linha de pensamento e analisará em detalhes os princípios mais importantes. Em seguida, apresento a você um bate-papo com um profissional da área sobre a linha de pensamento *lean*.

Você conhece o Lean?



| Você conhece o Lean?

Neste vídeo, está apresentada um pouco da história da linha de pensamento lean e seus principais princípios.

Entrevista sobre Lean



Entrevista sobre Lean

Neste vídeo, será entrevistado um profissional de métodos ágeis. Discutiremos as principais características do lean, com uma visão atual e prática de mercado.

Conclusão

O sucesso da indústria de manufatura de veículos japonesa é a comprovação da efetividade dos princípios da linha de pensamento *lean*. Não é difícil entender a importância dos sete princípios que estudamos nesta Unidade. Como vimos nos vídeos, conceitos importantes, como sistemas puxados e *just in time*, são princípios essenciais da linha de pensamento *lean*. Para práticas mais concretas, vamos estudar na próxima Unidade o método *kanban*, influenciado por essa linha de pensamento.

Referências

POPPENDIECK, M. et al. **Lean software development: an agile toolkit**. Alemanha: Addison-Wesley, 2003.

POPPENDIECK, T. The lean mindset. **Poppendieck.llc**, 2015. Disponível em: <http://www.poppendieck.com/>. Acesso em 24 de maio de 2021.

OHNO, T. O Sistema Toyota de Produção – Além da produção em larga escala. 1. ed. Porto Alegre: Bookman, 1997.

SHINGO, SHINGEO. O Sistema Toyota de Produção. 2. ed. Porto Alegre: Artes Médicas, 1996.



© PUCPR - Todos os direitos reservados.