O Desenvolvedor Ágil

A arte de fazer uma solução crescer
naturalmente através de mãos distintas.

Sidney Lima Filho

Indice

Introdução a Agilidade

Você na história

Se você está lendo este livro, então provavelmente você deve ter entre 20 e 35 anos e isso significa que você nasceu em um mundo já em transformação, a internet fez parte da sua juventude, assim como o mobile está fazendo parte da juventude atual, você aprendeu a viver num mundo onde praticamente qualquer informação está no Google, um mundo interconectado.

Para você, globalização não é um bicho de 7 cabeças, afinal você cresceu num mundo já globalizado. Viajar para o exterior, comprar coisas da China, conversar com americanos pela Internet, trabalhar de casa, tudo isso é algo natural para você (ou pelo menos já viu ou ouviu pessoas falando sobre isso).

O desenvolvimento de software, que começou com cartão perfurado (*imagina uma época em que se um cartão/código havia um erro, então deveria jogar o cartão/código todo fora e escrever tudo do zero*) também evoluiu e novas linguagens surgiram, propondo novas maneiras de resolver os mesmos problemas, conceitos e técnicas apareceram e desapareceram na mesma velocidade.

O Java, por exemplo, surgiu com uma proposta inovadora de modularização, popularizou o conceito de maquina virtual (mesmo sendo um conceito bem antigo), se espalhou pelo mundo, e quer você use ou não, temos que reconhecer o grande avanço que ela trouxe.

Nos anos 2000, enquanto você pegava no compasso da loira e da morena escutando É o Tchan (as vezes até mesmo quando você não queria), um grupo de caras fodas se reuniram num evento para discutir porque o desenvolvimento de software era do jeito que era, sendo que alguns deles tinham um jeito bem diferente de fazer software.

Por fim eles concluíram que esse jeito era mais ágil do que a forma padrão da época.

A agilidade surgiu reconhecendo a importância dos valores atuais, porém apresentando outros valores que no dia-a-dia são tão importantes ou mais, que os atuais. Isso traz luz a algumas práticas, eleva de patamar a qualidade do software, introduz novas necessidades, condizentes com um mundo dinâmico e fluido como o atual.

Se tudo mudou, o mundo agora é outro, não faz sentido continuar trabalhando do mesmo jeito que sempre foi não há mais espaço para amadorismo, ninguém mais quer chamar o sobrinho para fazer aquele site bacana, por um preço de matinê de domingo (nossa como estou velho).

Cá entre nós, eu sei que você adora ficar navegando horas a fio, aprendendo os mais recentes frameworks, vive querendo saber quais bibliotecas Javascript são as mais badaladas do momento, quer conhecer um monte de plugins novos, tudo que tiver sigla você quer saber,

ORM, XML, DDD, TDD, BDD, IoC, DCI, SPA, C#, SOA, Linq, RoR, PHP, Design Pattern (até porque DP pega mal), .NET e etc, só para colocar mais siglas no currículo. Quanto mais melhor, as vezes você nem sabe para que serve, nem quando usar, mas é importante conhecer de tudo certo? Errado.

Apenas por precaução, etc significa “et cetera”, um texto em latim com significado de ”entre outros”, então não é uma sigla e nem ponha isso no currículo, ok?

Um desenvolvedor moderno não busca conhecimento em que ele não vai empregar, muitas pessoas podem até discordar neste ponto, outras vão apenas ignorar, mas preste bastante atenção, um desenvolvedor excelente é aquele que consegue resultados excelentes. Saber de tudo um pouco te tornará exatamente isso, aquele que para cada cenário **só sabe um pouco**.

Não seja chato

Não seja o cara chato que só pensa em usar a ultima ferramenta da moda, porque dizem ser mais rápida. Descubra primeiro o quanto mais rápida é, faça um experimento. Não fique querendo forçar a barra, para a equipe adotar uma ferramenta que você não domina, só com intuito de ganhar 2% de velocidade no contexto global. Primeiro estude, aprenda, pratique, domine e por ultimo, use nos projetos que você entregará a seus clientes. Seja responsável.

Você comeria num restaurante onde cada dia os cozinheiros usam um ingrediente diferente só para eles aprenderem? Você confiaria sua causa na justiça a um advogado que para o mesmo tipo de ação, usa diferentes embasamentos jurídicos, só para aprender mais no julgamento? Você aceitaria se tratar com um médico que te receita medicamentos apenas pelo interesse de indústrias farmacêuticas? Não né, você não é cobaia, então faça com seu cliente o que você não gostaria que fizessem com você.

Você é um privilegiado.

Neste exato momento a profissão de desenvolvedor de software está em plena efervescência, com oportunidades em diversos ramos, desde a nossa “velha” Web, até TV e agora óculos também, o que mais vem por aí?

A nossa área amadureceu e nos proporciona diversas possibilidades que diferencia profissionais de amadores, dinâmicos de conservadores, agora você pode construir uma nova carreira na sua vida. Você tem a sua disposição conhecimento, técnicas, metodologias e práticas que juntas são 10x mais eficientes que quaisquer ferramentas, processos e burocracias.

Se você se incomodou com tudo que escrevi acima, então atingi seu coração, agora você só precisa saber o caminho para se transformar de cavaleiro solitário, cowboy coder, para um verdadeiro desenvolvedor ágil e vou te mostrar passo a passo.

Só depende de você.

Agora você pode se tornar um desenvolvedor excelente, eficiente, mais profissional, confiável, e finalmente, mais ágil.

Um pouco de contexto

Os desenvolvedores de software tem se interessado bastante pela agilidade e o ponto de partida começa na leitura obrigatória do manifesto ágil, porém falta explicar para os desenvolvedores como colocar os valores e princípios em prática, principalmente na parte técnica.

Se você acha agilidade interessante, quer colocar em prática, mas não faz idéia de onde começar, ou que isso é coisa de gerente, então esse texto é para você.

O manifesto ágil surgiu em meados do ano 2001, onde profissionais, reconhecidos pela sua excelência técnica, declararam praticar um conjunto de valores e princípios, identificados como responsáveis por tornar o processo de desenvolvimento de software mais eficaz e eficiente.

Ele, por si só, cumpre bem seu papel de explicar o porquê é tão mais eficiente trabalhar desse jeito, porém carece de mais informações que permita, nós desenvolvedores, colocarmos em prática.

Nós precisamos formar uma nova geração de desenvolvedores que já tenham o método ágil de desenvolvimento como principal conhecimento e justamente com você para iniciarmos nessa jornada.

Esse pensamento não é apenas meu, então você não precisa acreditar em mim (e também não estou tão louco assim), sobre a parte prática da agilidade, os autores do manifesto aos poucos estão se manifestando, pontuando essa preocupação.

A maior preocupação atual é que apesar dos valores do manifesto terem se espalhado e mudado a forma de vermos o desenvolvimento, a base fundamental da agilidade, a excelência técnica, não tem evoluído junto com os valores.

As pessoas tentam “instalar” a agilidade nas empresas, mas para entregar funcionalidades relativamente simples continuam levando 1 mês, os patrocinadores da transição começam a questionar a efetividade. O outro extremo do desenvolvimento são as demandas originadas dos defeitos, quando a qualidade não faz parte do processo desde o começo ao fim, então a primeira decisão gerencial é criar uma etapa de qualidade a cada entrega, isso gera um silo e demanda uma documentação para cada entrega, com finalidade de elaboração de testes. Pronto, aos poucos a agilidade de contexto e entra aos poucos o modelo cascata, sem ninguém perceber.

“When agile equals nothing more than sprints,
standups & storypoints, poor software quality is guaranteed.”
Joshua Kerievsky, author of Refactoring to Patterns.

O cenário acima é o comportamento que mais ouço em todas equipes, onde muitos culpam a metodologia, outros culpam as pessoas, mas na verdade a origem do problema é um só. Curioso? Continue lendo que você vai entender.

Manifesto Ágil para desenvolvedores

Se você está aqui lendo isso, então provavelmente você é um desenvolvedor de software e vou presumir que já leu o manifesto ágil certo? Eu coloquei o link para o manifesto 3 vezes e você ainda não leu? Então para tudo e vai lá rapidinho, eu espero.

Agora vamos analisar o manifesto juntos. Lembre-se o manifesto não foi escrito por estagiários, ele foi escrito por pessoas que sabem desenvolver qualquer tipo de software, desde Web CRUD’s até sistemas de missão critica, controladores de mísseis e algoritmos de varredura de ressonância magnética.

Os valores do manifesto é a base que sustentam os princípios, eles são como pilares. Imagine o manifesto como uma casa, onde os valores são 4 pilares que sustentam o telhado de 12 princípios. Os pilares são:

- **Pessoas** Indivíduos e Interações mais que Ferramentas e Processos
- **Confiança** Colaboração com o Cliente mais que Negociação de Contratos
- **Valor** Software funcionando mais que Documentação Abrangente
- **Complexidade** Responder a Mudança mais que Seguir um Plano

Repare que os 4 pilares tratam de aspectos do processo de desenvolvimento de software e a excelência técnica é uma habilidade que sustenta aos 4 pilares e que garante uma entrega continua de software, é o alicerce da agilidade, então sem ela até existe agilidade mas é frágil e perecível, qualquer incerteza, limitação e status-quo podem derrubar.

Vamos começar pelo primeiro pilar, Pessoas.

Pessoas

Você chega de manhã, pega um café, coloca o fone no ouvido e começa a programar, então chega um momento que você acredita que terminou a funcionalidade, então você elabora um belo texto explicando o que fez e libera para o usuário. Pronto agora uma navegada na internet para relaxar a mente e pegar o próximo post-it.

A rotina acima parece um dia normal de uma equipe ágil, porém esconde várias incertezas e pode não ter nada de ágil. Quem vai testar? e quando? Como a equipe de testes sabe que tem código para testar? e como eles saberão o que testar? O código terminado vai funcionar no ambiente de produção? O código possui bugs que inviabilizam o uso? Qual a definição de pronto para produção? Quem pode afirmar que este código está pronto?

Código parado no versionador não é software funcionando.

Se não houver técnicas que garantam que o código está pronto para produção, então o caminho natural é ter cada vez mais ferramentas e processos, ao invés das pessoas interagirem e trabalharem em conjunto.

Um desenvolvedor ágil interage com todos os envolvidos durante sua codificação, emprega técnicas que viabilizem um feedback rápido do código fonte, onde expõe cenários ocultos e elimina todas essas incertezas através de técnicas de testes, automação de processos, descobre mais informações e compartilha com o testador em paralelo, além de inúmeras outras formas.

O mundo ideal seria que o cliente pudesse escrever um requisito, que automaticamente se transforma em código, dispara alguma rotina de checagem de sintaxe e testes de todos os cenários catalogados pelo cliente, então se tudo estiver OK, envia o código direto para produção. Pronto, software funcionando e valor agregado para o cliente imediato. Mas sabemos que isso ainda não é possível.

Quanto mais tempo seu código demora de ir para produção, mais sua equipe irá precisar de ferramentas e processos para controlar o desenvolvimento.

Confiança

A aparência é um dos atributos mais valorizados hoje em dia, todo mundo quer estar bonito, ser bem visto e elogiado, mas por mais que este atributo seja bastante valorizado, quando se vai ao barbeiro ou salão de beleza, ninguém costuma assinar contratos, negociar valores, muito menos pressionar por velocidade. Por quê?

Nesses lugares quem manda é o cliente, do começo ao fim, o cliente diz exatamente como quer sua aparência, como o profissional deve fazer, o resultado final esperado é transparente desde o começo. Fotos, vídeos, imagens de referencia, são usados o tempo todo.

Se o cliente quer mais rápido, ele aceita um trabalho sem acabamento. Se o cliente quer mais barato, ele corta o escopo do trabalho. Se o cliente quer mais qualidade, ele dá mais tempo.

Em desenvolvimento de software, principalmente quando se trabalha em consultorias, o cliente costuma determinar contratos leoninos, tenta baixar ao máximo os valores do projeto e trata como commoditie, aquilo que não é. Por quê?

Os softwares foram construídos durante décadas por empresas que buscavam determinar o que o cliente deveria receber, o cliente, por sua vez, sempre usou o contrato como uma forma de forçar o fornecedor a entregar o que ele quer. O resultado? A confiança e transparência foram completamente expurgadas do processo.

Quando você está desenvolvendo uma tela costuma chamar o cliente para checar se ele gosta do que vê? Como você garante que os textos das telas estão claras para o usuário final? Manda para o time de UX? A tela está fazendo exatamente o que o cliente espera? Como você pode ter certeza disso?

Em média 25% do tempo usado para produzir de software é gasto no refinamento ao final do processo, porque o cliente não foi consultado a cada etapa.

O software é um artigo subjetivo e existe infinitas formas de chegar ao mesmo resultado final, portanto um desenvolvedor ágil precisa criar mecanismos que permitam maior transparência de informações e reestabeleça a confiança do cliente de que receberá o que tem em mente.

Valor

O que importa para o usuário final é o valor agregado que o software irá proporcionar quando está pronto, então na agilidade a maior prioridade é satisfazer o cliente o quanto antes através de entrega continua de software com valor agregado.

Our highest priority is to satisfy the customer
through early and continuous delivery
of valuable software.

Um desenvolvedor ágil checa constantemente se ele investirá seu tempo em produzir algo que o cliente realmente quer, e se possível garante isso antes de fazer, de forma que futuras funcionalidades não estraguem este valor gerado.

Se você não se preocupa em garantir isso, então posso afirmar que além de você não ser ágil, você está trabalhando de maneira ineficiente, levando muito mais tempo para entregar o que o cliente quer.

Complexidade

Este pilar eu poderia chamar apenas de adaptabilidade que já seria um bom nome, porém complexidade vai além e engloba também o conceito de aquilo que só pode ser analisado após o fato já consumado.

A Teoria dos Sistemas Complexos declara que complexo é tudo aquilo que depende do estado anterior para determinar o estado atual, então com base nessa premissa, o processo de

desenvolvimento de software é um sistema complexo, a próxima funcionalidade a ser desenvolvida sempre dependerá de como as funcionalidades já existentes resolvem o problema atual.

Quando falamos que responder a mudança é mais importante que seguir um plano, primeiro temos que ter certeza que isso é possível, pois se mudar for caro, ou lento, ou arriscado, então seguir um plano passa ter um melhor custo beneficio.

Um desenvolvedor ágil é aquele que busca a todo momento criar mecanismos para que mudar seja rápido, barato e seguro, assim permite o cliente aprender com as funcionalidades existentes e identificar com maior precisão o que agrega valor para ele naquele momento.

Veja só como agilidade tem tido a ver com programação:

- Um código com componentes acoplados deixa o processo de responder mudanças mais lento e custoso.
- Um código sem testes automatizados não é capaz de informar imediatamente se uma mudança irá afetar outra parte do projeto.
- Um código de difícil leitura transmite insegurança para quem realizará a mudança e esta pessoa por precaução irá inflar prazos e estimativas.
- Um código sem arquitetura definida e de conhecimento da equipe não permite que um desenvolvedor novo possa realizar mudanças no código.

A lista com a relação entre o código e agilidade é imensa, mas acho que já deu para passar a idéia dos inúmeros impactos que um código pode causar na agilidade da equipe. O mais importante neste momento é que você tenha entendido que a sua forma de programar impacta e muito na agilidade da sua equipe.

Cada pilar do manifesto apresenta ao desenvolvedor que ao alavancar sua excelência técnica, ele contribui diretamente para aumentar a agilidade da equipe.

A agilidade não é só uma coisa de gerente, mas também uma forma mais inteligente de desenvolver software. Sem desperdício. Sem rodeios desnecessários. Direto ao que importa para o cliente.

Cada ação que o desenvolvedor realiza, desde o IF até ao Commit, impactará na agilidade do desenvolvedor e da equipe.

"Programmers spend the first 5 years of their
career mastering complexity, and the rest of
their lives learning simplicity." - Buzz Andersen @buzz

Capitulo 2

A teoria explica, a prática ensina.

O perfil de um desenvolvedor ágil

O tempo está cinza e você está ligeiramente com frio, seu chefe chega bem cedo e feliz, cumprimenta a todos e quando chega a sua vez, com um sorriso matinal de quem transou na noite anterior, te pergunta:

- Bom diaaaa, como tá o desenvolvimento, tudo certo?

Você sem pestanejar (e também para não estragar o raro momento de felicidade de quem não tem vida) diz que está indo bem, porém não faz a menor idéia de como testar o que você está fazendo, muito menos se o código vai realmente atender ao cliente.

O desenvolvedor ágil é um profissional diferente, ele sabe o que está fazendo, sabe quando o que está fazendo está pronto e sabe se adaptar rapidamente para entregar o máximo de valor possível.

Vamos ver algumas características essenciais de um desenvolvedor ágil.

Só desenvolve, quando sabe qual é o resultado esperado.

Já vi muitos desenvolvedores, que ao serem perguntados sobre o motivo de desenvolverem uma determinada funcionalidade, respondem: “Sei lá, é o que está na minha fila”.

Você não sabe o motivo de estar fazendo algo, não sabe quem pediu, porque pediu, não faz a menor idéia se está resolvendo o problema do usuário. Só sabe que no post-it está escrito “Eu como Usuário não quero preencher o formulário inteiro, portanto os campos X, Y e Z devem conter com um valor padrão.”, como é uma tarefa simples, nem se preocupa em saber mais detalhes.

Se você não sabe o motivo de fazer, você não sabe o impacto que aquilo terá no dia a dia do cliente, e nem se isso impactará os outros usuários. Se o campo X iniciar preenchido, irá tendenciar o resultado no relatório por X? O campo Z é realmente necessário para todos? Se ao invés de apresentar já preenchido, ele fosse oculto de acordo com o tipo de usuário?

Quando se trabalha sem se interessar pelo real problema, o que ocorre é que a feature é entregue e quando está em produção, o usuário repara imediatamente o impacto e então solicita mais alterações. A partir deste ponto, não se sabe mais se as alterações são bugs, inconformidades ou novas features disfarçadas de bugs.

O desenvolvedor ágil busca incessantemente saber qual é a real definição de pronto, quais são os impactos possíveis, transforma essa definição de pronto em alguma forma de validação, escreve o código estritamente necessário para que passe pela validação e que não tenha impacto no que já estava pronto, tudo isso no menor tempo possível.

Preocupação com o todo

Por falar em impacto, um desenvolvedor ágil precisa se preocupar com o todo, e quando digo todo, me refiro a tudo aquilo que está fora das functions. Classes, Interfaces, Programa, Escopo, Requisitos, Projeto, Datas, e principalmente PESSOAS.

A sua maior fonte de insights para a melhoria continua são as pessoas. Elas fornecerão informações valiosas que podem fazer você patinar por horas para resolver um problema, ou pode simplesmente indicar um atalho.

Quando um desenvolvedor inicia numa nova empresa é sempre aquela alegria, alta energia, participativo, tem idéia para tudo, é falante e expansivo. Após três meses corrigindo bugs no sistema legado de 200 mil linhas, onde dessas tem um super IF de 20 mil linhas com 10 níveis de profundidade, já está reclamando que o café tem sabor ruim, a porta do banheiro não fecha direito e a empresa não valoriza as pessoas.

O maior problema os sistemas legados é a arquitetura caótica que impede implementar testes unitários com facilidade, que atrapalha ao estender o sistema, e por isso a preocupação com o todo é fundamental.

Desenvolver software é a capacidade de usar uma máquina que processam dados, para transformar idéias em sistemas de informação.

Softwares não são soluções para problemas. Eles são automações de padrões reconhecidos, logo a capacidade do desenvolvedor de reconhecer padrões determinará a capacidade dele gerar um software, que pode ser a solução para um ou mais problemas. Não existe problema 2.0, imbróglio alpha, muito menos situação beta, por isso não se cria solução.

A solução somos nós e o software é a materialização da nossa qualidade.

Parece meio clichê, mas realmente quanto mais nos preocuparmos com o todo, melhor será nossa capacidade de criar soluções.

Auto-gerenciado

Você consegue se gerenciar? Essa é uma daquelas perguntas que as pessoas não faz idéia do que seja e imediatamente responde sim.

Eu não consigo me gerenciar sozinho, mas eu consigo ser auto-gerenciado.

Se gerenciar significa ter compreensão de suas forças e fraquezas, usar as forças nas coisas necessárias do dia-a-dia e blindar suas fraquezas delegando para quem as tem como forças.

Para ser auto-gerenciado, você precisa garantir que não é necessário ninguém para te gerenciar. Isso é um desafio, até porque muitas pessoas não têm isso na sua cultura familiar, elas crescem sem muitas responsabilidades, sempre com alguém supervisionando ou alternando o castigo e recompensa para obter o comportamento esperado.

A auto-gerência é essencial para um desenvolvedor ágil, pois quando o desenvolvedor não apresenta esta característica, ativa o que há de pior nos chefes e gerentes, a capacidade de ser um capitão do mato e nos tratar como escravos, que trabalha por comida.

A revolucão da informação que está acontecendo agora no século 21 e está permitindo aos profissionais de desenvolvimento de software possuir o emprego que quiserem, então se você atualmente não está no emprego que quer para você, só tem duas alternativas: Ou você já é um profissional sério, mas está em um lugar medíocre; Ou seu chefe te trata como mão de obra barata porque você é medíocre e ele sabe que se não te der chicotada, você abre o navegador e fica na internet.

Este livro não é auto-ajuda para desenvolvedores de software, então como você está lendo este livro, eu acredito que você não seja medíocre, apenas busca orientação para alavancar sua carreira, então busque ser mais auto-gerenciado e tire a poeira do currículo.

Agora se você acha tudo muito difícil, chato, filosofia de bar, por favor, não continue lendo este livro, faça algo melhor da sua vida, clique aqui e divirta-se.

Tem coragem para mudar o status quo

A primeira coisa que um desenvolvedor ágil faz ao entrar num novo ambiente é observar e catalogar, todos os cenários de status quo.

A partir de hoje você não é apenas um desenvolvedor, mas é também um agente de mudanças, como agente de mudanças sua prioridade numero um é não mudar. Como assim? Eu vou explicar.

Quem muda sai de um lugar e vai para outro lugar, porém nem sempre para melhor, apenas troca de situação, sai de um cenário A para um cenário B. Um agente de mudanças não faz isso, ele é a pessoa responsável por facilitar a evolução da equipe, do sistema, do desenvolvimento de software em si e apenas promove a mudança quanto há estudos e indícios comprovados de melhoria.

> - Poxa chefe você é o PO, tem que explicar melhor essa issue para colocarmos na Sprint.
> - Eu sou o QUE?

Um desenvolvedor ágil não usa jargão hype para se auto-proclamar ágil, não confunde as pessoas, nem usa a agilidade como trampolim salarial. Dinheiro é bom, você gosta e eu também gosto, mas confie em mim, ser capaz de garantir resultados comprovados é a forma mais orgânica de alavancar seu crescimento profissional.

Estabelece nova fundação para sistemas legados

Se a primeira coisa que um agente de mudanças faz é observar e catalogar o ambiente, com o código é a mesma coisa.

Comece catalogando como o sistema se organiza quais as possibilidades de melhoria (mas não mexa em nada ainda), pergunte aos mais antigos da equipe se houveram tentativas de refatoração fracassadas. Pergunte qual era a finalidade da refatoração e também o porquê deu errado.

Uma vez essas informações levantadas, como um agente de mudanças você sabe que precisa tracar um plano para mudar o status quo e como um desenvolvedor ágil também precisa arregaçar as mangas.

"As pessoas não resistem as mudanças,
elas resistem serem mudadas" - Bruce Lee

Neste primeiro momento não mude o status quo, use os processos atuais, a burocracia atual para conseguir uma primeira vitória de preferência bem pequena.

Lembra que o desenvolvedor ágil não faz nada sem garantir que vai dar certo? ;)

Pegue uma pequena feature, que precisa de refatoração, e convença a necessidade de entrar no planejamento/backlog/listinha/excel do chefe.

Neste momento a vitória é mais importante para dar moral a equipe, valorizar o grupo, e principalmente você mostrar que sim é possível um futuro melhor para todos.

Todas as minhas tentativas de mudar o status-quo, só funcionaram quando precediam de resultados consistentes e garantidos, suaves e mudanças de processos, ao invés de mudar pessoas.

Este livro é um exemplo.

Eu não busco mudar quem você é, nem suas convicções, mas com certeza busco te apresentar uma nova postura profissional,

Não há agilidade sem excelência técnica

"Software funcionando é a medida primária de progresso,
então não enche o saco se você não entende o que escrevi"
- Rick Marlboro, cowboy coder que faleceu
por uma picada de bug desconhecido

O mundo está em constante evolução, o cenário dos usuários mudam o tempo todo, novas informações surgem, alterando completamente a tomada de decisões, portanto, precisamos adaptar nossos sistemas constantemente e rápido, pois se o cliente demorar um mês para ver o sistema funcionando, mesmo que seja exatamente o que ele pediu, pode não servir mais para o propósito dele, com isso nosso o maior inimigo não é o waterfall (que em alguns cenários tem muitas vantagens por sinal), mas sim o cowboy coding.

O cowboy coding é geralmente a primeira forma como aprendemos a desenvolver software, onde fazemos do nosso jeito, sem regras, padrões, a qualidade do produto final será aquilo que julgarmos melhor, afinal estamos aprendendo e o foco está no aprendizado, mas ser um profissional é o extremo oposto, consiste em ter capacidade de produzir resultados confiáveis, com qualidade, sempre.

"Quando eu era menino, falava como menino,
pensava como menino e raciocinava como menino.
Quando me tornei homem,
deixei para trás as coisas de menino."
- Paulo de Tarso em 1Cor 13:11

A falta de disciplina é o principal motivo alegado para a erosão da agilidade nas equipes, o que não causa surpresa, pois como exigir disciplina de uma comunidade jovem, vibrante, criativa, puro trabalhadores do conhecimento como os desenvolvedores de software? Ops pera ae.

Disciplina é diferente de rotina.

Ok, não gostamos de rotina, porém disciplina é mais do que isso, é a capacidade de práticar exaustivamente, até adquirir a habilidade de repetir um resultado com o mesmo nível de performance e isso não tem a ver com rotina.

Quando somos profissionais disciplinados, coisas maravilhosas acontecem, conseguimos facilmente enxergar soluções onde antes não existiam, só porque já passamos pelo mesmo problema diversas vezes, outros problemas aparecem disfarçados e por pura abstração, nosso cérebro reconhece o padrão e transcodifica a solução.

A disciplina é a característica fundamental que fará um desenvolvedor alcançar a excelência técnica.

Outro ponto importante é que se a disciplina for aplicada somente aos processos ágeis, e negligenciar a excelência técnica, torna a agilidade frágil e perecível, pois todas as pessoas

envolvidas começarão aos poucos forçar os limites e a equipe começará a ficar sem argumentos para as pressões.

Correções eternas de bugs highlander, bug bumerangue, que é corrigido e volta a aparecer meses depois, prazos gigantes para funcionalidades simples, falta de tempo para testes, sistema frágil que apresenta erro em qualquer cenário incomum, lentidão de homolgação pela fase de testes, retrabalho de equipes inteiras com entregas sem valor para o cliente, sem contar com o grande e temido Refucktoring, que todo desenvolvedor tenta empurrar a necessidade de refazer alguma parte do código do zero.

O desenvolvedor é a peça mais importante do processo de produção de software e a engenharia ágil que os desenvolvedores forem capaz de implantar, determinará se a agilidade da equipe será duradoura ou perecível.

A excelência técnica é uma premissa básica onde todos os autores do manifesto se basearam para estabelecer os valores e princípios, num raciocínio: “já que somos excelentes técnicamente, então é viável trabalhar de forma que X é mais que Y”

O Scrum, por exemplo, sem excelência técnica não é agilidade, é um waterfall fatiado em sprints.

O waterfall é como um trem com 20 vagões, onde tem força e uma razão clara da sua necessidade, mas é grande, pesado e lento. O scrum é como um trem com apenas um vagão, que pára nas estações de 15 em 15 dias, menos pesado, mais rápido, porém ainda em cima dos trilhos, o papel do scrum master, por exemplo, é manter o trem nos trilhos.

Se a metodologia que você usa (não importa o quão "agil" ela diz ser) impede adaptação imediata, determina uma data para as coisas acontecerem, possui cerimônias mesmo que desnecessárias, então lamento dizer, não é ágil.

A agilidade é como uma moto, onde a capacidade de se adaptar é mais importante que a velocidade, envolve mais riscos, porém é recompensado com a possibilidade de usufruir de imediato.

Continuous attention to
technical excellence *and good design*
enhances agility.

As técnicas de desenvolvimento de software têm um potencial mais transformador que as tecnologias em si, pois elas justificam as mudanças de valores dos tomadores de decisão.

Esqueça as metodologias ágeis

O processo de desenvolvimento ágil em essência não precisa de metodologias ágeis, pois a agilidade é um manifesto. É como se os médicos fizessem um manifesto por uma higiene melhor nos hospitais, as novas metodologias de como lavar as mãos são legais, porém todo

mundo sabe que o problema da higiene não são os médicos e sim a baixa prioridade de como o hospital trata da limpeza.

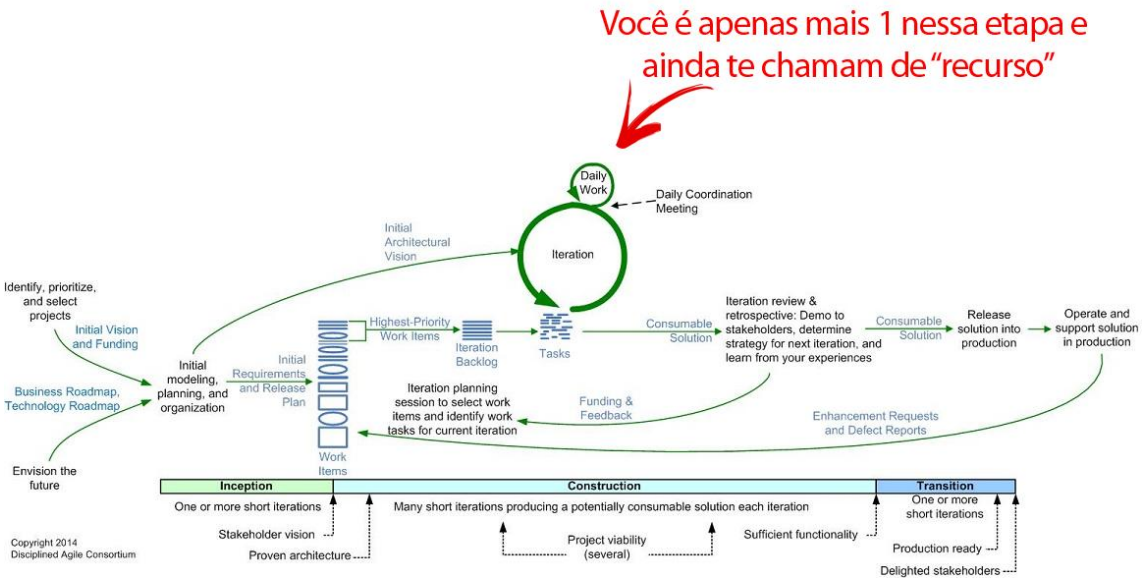
No desenvolvimento de software acontece a mesma coisa, o manifesto está lá explicando tudo, porém as pessoas precisam do “como” e as metodologias vem para cumprir este papel.

No meu caso resolvi escrever este livro, pois estou preocupado com os desenvolvedores, afinal são os desenvolvedores os mais afetados em todo este processo.

Ninguém explica para o desenvolvedor como praticar os valores do manifesto ao escrever código.

As metodologias ágeis são ótimas, cumprem um papel importante na disseminação dos valores e na adpatação do mundo antigo ao mundo novo, porém implantá-las demanda tempo e não garante a agilidade.

Este livro é sobre o que você pode começar a fazer amanhã de manhã para promover a agilidade na sua equipe.



A maioria dos comportamentos atuais deriva de uma engenharia obsoleta, então se não mudarmos a forma de codificar, dificilmente mudará a forma de entregar.

Eu vejo muitos fracassos em transições ágeis, principalmente devido ao fator que a engenharia é obsoleta, quando digo obsoleta não me refiro as tecnologias, mas sim a forma arcaica de tentativa e erro. Escrever. Testar. Debugar. Escrever mais um pouco. Debugar mais. Entregar no último dia do prazo com bugs.

Metodologias ágeis não entregam software, quem entregam são os desenvolvedores, portanto são eles que precisam ser ágeis e não o processo.

O Waterfall é a maneira mais simples, mais básica, menos arriscada e também mais lenta de fazer software.

Quando você tenta explicar para uma criança que 7 gatinhos possui 8 frutas cada um, você precisa fazer um processo passo-a-passo, cascata de soma, então começa explicando que 8 + 8 + 8 + 8 + 8 + 8 + 8 + 8 = 56. Você não explica a operação de multiplicação, 7 x 8 = 56, pois isso requer abstração e uma criança ainda não possui essa habilidade (tem muitos adultos que até hoje não possui essa habilidade e nem sabe de cabeça quanto é 7 x 8).

Nas equipes é a mesma coisa, o Waterfall busca mitigar qualquer risco durante o processo, abrindo mão da velocidade. A agilidade é a capacidade de mudar de direção a qualquer momento, mantendo a velocidade.

Então você deve estar se perguntando. E o que eu como desenvolvedor tenho a ver com isso?

A sua tarefa agora é ser um desenvolvedor que promove a agilidade, para isso explicarei um único algoritmo infalível, que serve para tudo e que você não pode esquecer.

Quando o desenvolvedor não consegue manter um alto nível de agilidade, através da excelência técnica, aos poucos os princípios mitigadores de risco do Waterfall começam a voltar para o contexto.

Se a equipe não consegue manter um nível alto de qualidade dos requisitos, então o gestor achará o Scrum sensacional, pois tem um papel chamado PO que tem a obrigação de fazer isso e vai criar outro gargalo na etapa inicial do projeto para elucidar todos os requisitos antes de escrever código.

Se a equipe não consegue manter um nível alto de qualidade do código, então o gestor achará que é o momento de ter uma equipe de arquitetura que validará todo o código escrito antes de ir para a etapa de qualidade.

Se a equipe não consegue manter um nível alto de qualidade na entrega, então o gestor buscará colocar a equipe de qualidade numa etapa ao final do processo que acumulará os itens e fará os testes de regressão manualmente.

Pronto, uma engenharia obsoleta obriga a gestão desenhar processos obsoletos.

Aos poucos estamos reconstruindo um novo Waterfall, não deixe isso acontecer.

O maior inimigo da agilidade é o Cowboy Coding

Se a todo o momento estamos falando de agilidade, então qual seria o contrário de agilidade? O contrário é algo grande, pesado e lento, porém como já falei também isso as vezes é necessário, não tem como um navio cargueiro de 200 toneladas ser ágil, no nosso caso você acha realmente que um software controlador de ressonância magnética deveria ser feito em sprints?

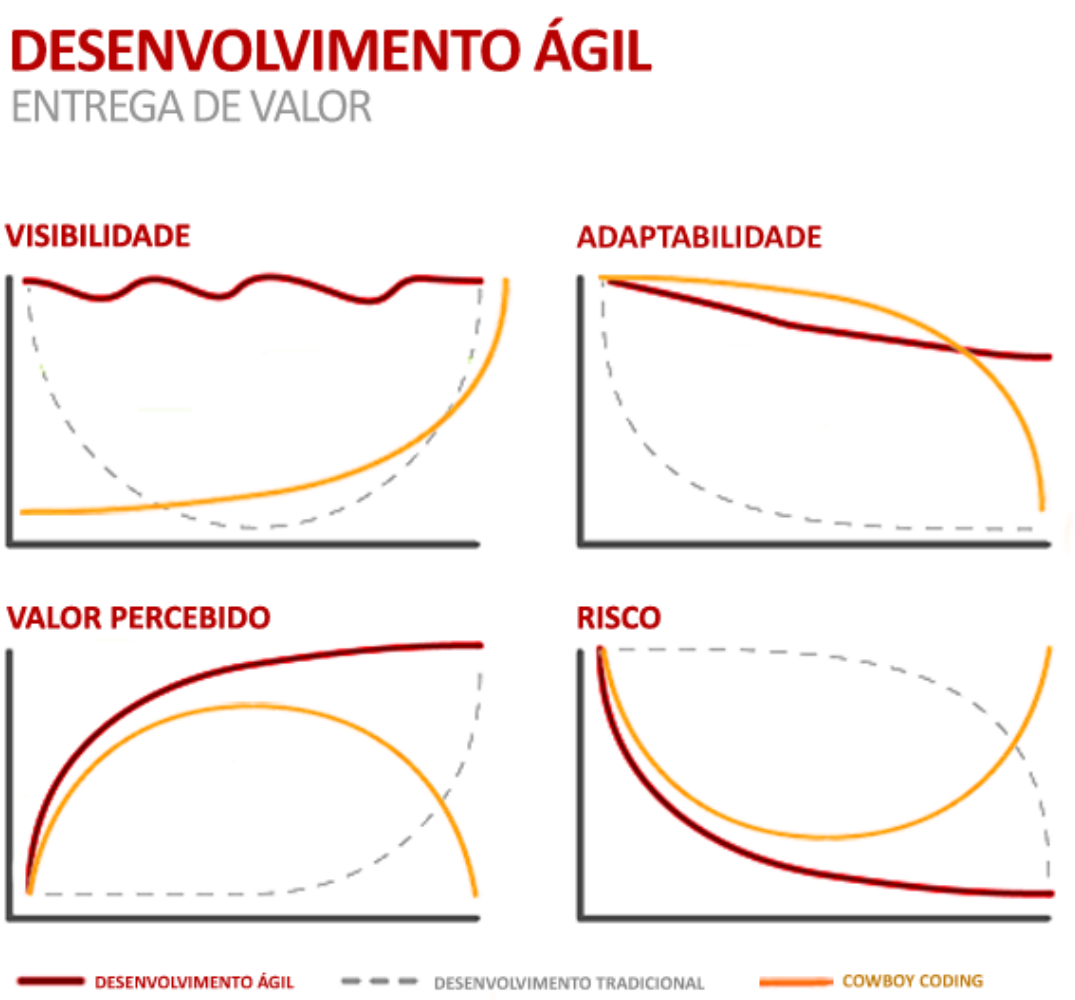
O modelo cascata possui um grande benefício, uma etapa grande de análise prévia permite uma maior qualidade na arquitetura do software, em contra partida aumenta o tempo necessário para entregar valor. Outro ponto importante é que somente agrega valor em ambientes estáticos.

Se você atua em cenários estáticos, então a agilidade não seja de grande valor para você, agora se você for como eu e atua num ambiente de grande dinamismo, com alta dependência de comunicação com clientes (sendo algum deles com transtorno bipolar) para determinar requisitos e prazo de término indeterminado então o seu maior problema não é o modelo cascata, mas o cowboy coding.

Cowboy coding é um modelo anárquico, onde cada um faz do jeito que quiser, se for para agradar o cliente.

A entrega de valor do cowboy coding (CC) é a mais rápida que existe afinal tudo será feito para que o cliente fique satisfeito com o trabalho do cowboy, porém nada integra com nada, o código é de difícil manutenção, despreocupação total com a evolução do sistema, além de ser altamente perecível.

O gráfico abaixo ilustra bem as diferenças entre os paradigmas de desenvolvimento de software.



Visibilidade

A visibilidade de um projeto é a capacidade do usuário final tem de visualizar o andamento em comparação com suas expectativas.

Repare no primeiro gráfico que o modelo tradicional apresenta boa visibilidade no começo do projeto, aos poucos essa visibilidade se perde pois as complexidades e imprevistos que surgem no decorrer do projeto fazem os prazos das etapas intermediárias serem estourados.

No meio do projeto as coisas começam a ser entregues (para a área de qualidade, não para o cliente) e para compensar todo o atraso dos prazos intermediários perdidos, começam as jornadas extras, pressão, apresentação para o cliente de partes inacabadas e então finalmente a visibilidade retorna a ser mais nítida.

No cowboy coding as coisas são bem diferentes, o cliente fala o que ele quer, o malandrão diz que entendeu perfeitamente e começa a trabalhar sem participação do cliente. A síndrome do estudante e a procrastinação imperam na cabeça do desenvolvedor, até quando começam a surgir os primeiros questionamentos do cliente, e então, começa o trabalho "go horse", sem

nenhuma técnica, sem nenhum padrão, testando o cenário mínimo para funcionar. Em comparação com o modelo tradicional até começa apresentar maior visibilidade antes, porém como tudo é deixado a manutenção futura a visibilidade total só depois de pronto.

Agora analisando o modelo ágil, as ondulações que aparecem no gráfico são referentes aos timeboxes praticados, por exemplo, um time scrum que pratica uma sprint de um mês, terá incremento da visibilidade nas entregas de mês em mês.

No modelo ágil, o ideal é entregar software funcionando o mais rápido possível, buscando manter a visibilidade do andamento do projeto sempre o mais transparente possível, logo cada funcionalidade ao ficar pronta deveria ir para produção imediatamente, fazendo a linha do gráfico ficar sempre reta no topo.

Alcançar esse modelo ideal é possível e mais adiante vamos entrar em cada etapa do desenvolvimento, implementando técnicas que permitirão você colocar software em produção o mais rápido possível, desde a primeira funcionalidade.

Adaptabilidade

A adaptabilidade é a capacidade de um projeto se adaptar a alterações de escopo, de equipe, de prazo e de recursos importantes para sua conclusão.

Se adaptar a mudanças não é uma tarefa fácil, requer muita disciplina para não atrapalhar o processo original, requer muito foco para não fazer código desnecessário.

No gráfico percebemos que o modelo tradicional, como era de se esperar, que a adaptabilidade cai rapidamente, pois só o fato de documentar todo o projeto no início, impede de se adaptar rápido afinal neste modelo quando uma mudança é realizada todo o projeto precisa ser reanalisado.

Agora vem a verdadeira tentação, o cowboy coding durante o projeto possui uma alta capacidade adaptabilidade, que motiva muitas pessoas enxergarem valor nessa iniciativa, o que ninguém percebe (e ao começar novos projetos esquecem disso) é que essa "qualidade" diminui drasticamente ao se aproximar do fim do projeto devido as gambiarras colocadas no início do projeto, falta de versionador, código duplicado e etc

O desenvolvedor ágil mantém sempre o código limpo, claro, simples e enxuto, isso facilita a entrada de novas demandas e mudanças de requisitos, mas ainda não é o suficiente para manter a adaptabilidade em 100%, devido a falta de excelência técnica que é a principal carência nossa times ágeis ultimamente.

Excelência técnica não é a mesma coisa que senioridade. É totalmente possível ter uma equipe de plenos e iniciantes que praticam excelência técnica.

Valor Percebido

O valor percebido pelo cliente é o principal fator de decisão no mundo atual, pois ninguém mais agüenta esperar 3 meses para ver seu sistema pronto e funcionando, então se não houvesse o movimento da agilidade em desenvolvimento de software, nasceria de qualquer jeito.

Desenvolver software é fácil, basta escrever qualquer código. Nas faculdades de TI o foco da grade curricular é apresentar o máximo de conhecimento sobre programação sem apresentar como "plugar" o desenvolvimento de software ao mundo real.

Na minha carreira em desenvolvimento eu descobri, até um pouco tarde, que a "magia" ocorre quando você desenha a solução, ainda na fase de requisitos, o resto é escrever código.

Antigamente desenvolver software era bastante trabalhoso, mas simples, apenas uma linguagem, todos os requisitos eram desenvolvidos e validados na mesma plataforma. Só que escrever código hoje em dia não é mais assim, para web por exemplo, você precisa conhecer no mínimo 5 linguagens diferentes (HTML, CSS, Javascript, uma linguagem Server-side e SQL), então para o usuário perceber valor é necessário mudamos a forma como desenvolvemos. Vamos dar uma olhada no gráfico.

O gráfico de Valor Percebido em tese deveria ser quase igual ao gráfico de Visibilidade, afinal se tenho visibilidade do andamento então posso perceber o valor correto? Errado, nesse momento entra um fator chamado Teoria da Complexidade, e por definição, desenvolver software é uma atividade complexa.

Complexo é tudo aquilo que só é possível determinar a relação de causa e efeito, após acontecer.

No gráfico vemos isso claramente no modelo tradicional, pois para descobrir todos os cenários possíveis (relação causa-efeito), 80% do processo é gasto no mapeamento de uso, logo o valor do começa a ser percebido pelo usuário na reta final do projeto. Isso dificulta por exemplo o usuário experimentar a funcionalidade depois de pronta, e validar com o mundo real, que em muitos casos o usuário só, consegue certificar se uma funcionalidade realmente atende, usando.

O CC (só pelo nome não é coisa boa) por sua vez nos prepara uma nova armadilha, começa apresentando valor para o cliente muito rápido, mas novamente o valor começa a perder velocidade, com as refatorações constantes, grandes esforços de manutenção e etc.

O modelo ágil faz com que o valor seja entregue rapidamente em produção, nas mãos do cliente desde a primeira funcionalidade, assim permite o usuário priorizar melhor sobre o que é mais importante para ele. Talvez estender um relatório estatístico pode ter mais valor do que um simples cadastro de cidades, e como profissionais de desenvolvimento é nossa responsabilidade guiarmos o usuário neste processo de descoberta.

Risco

O risco de um projeto é inversamente proporcional a velocidade de entrega de valor, quanto mais rápido validar se a ferramenta software soluciona um problema real, menor é o risco do projeto.

Os gráficos de Gantt que os gerentes de projeto adoram usar para medir andamento de projeto não resistem a uma prova de realidade. Na primeira entrega de software para o cliente, ele normalmente fará diversas observações sobre o que foi desenvolvido e que por sua vez, se tornarão diversas demandas de manutenções, que obrigará refazer todo o gráfico. Pelo menos até a próxima entrega.

Acho que você já entendeu a pegada e pode até analisar sozinho o gráfico de Risco. Vou deixar você fazer este exercício.

O método tradicional a um bom tempo já não se mostrava muito eficiente, por isso muitas pessoas já estavam abandonando de vez o uso, entretanto começou uma adoção do cowboy coding, como modelo extremo, sem regras, sem padrões, sem direção, apenas código, código, código. Uma falsa noção de velocidade que termina em desperdício, atrasos, pressão, viradas de noite, horas extras e muito stress.

As páginas a seguir são rascunhos da Parte 2 e estarão disponíveis caso haja pelo menos 100 interessados.

O único algoritmo que todo desenvolvedor precisa saber

Em toda história do desenvolvimento de software sempre houve um algoritmo que gera todos os outros, porém essa magia negra ninguém explica direito (porque todo mundo do meio acadêmico tem mania de ensinar as coisas separadas e sem contexto), então vou TENTAR simplificar o máximo possível ok?

desenvolver software é um processo de descoberta.

Todos algoritmos inventados são frutos de incertezas que precisaram ser validadas.

As funcionalidades que criamos todos os dias, até mesmo os cruds, são possibilidades de solução para um problema que o usuário tem, ou seja, são incertezas.

Será que vai resolver realmente o problema? Totalmente ou só parcialmente? Existe alguma particularidade no cenário do usuário que não foi pensada? Se algumas dessas perguntas tiver uma resposta inesperada, como ter certeza que não haverá retrabalho ou desperdício construindo uma funcionalidade que não será usada?

80% das funcionalidades nos sistemas atuais, são usados apenas por 20% dos usuários.

O desenvolvedor ágil não escreve código, sem saber quando estará realmente pronto, sem saber exatamente o que o código deve fazer.

O desenvolvedor ágil garante através de técnicas que o código gerado pela equipe estará em produção no menor intervalo possível.

o desenvolvedor agil garante que qualquer código escrito por qualquer membro da equipe, sempre permitirá que manutenções futuras sejam curtas, rápidas e mantenha o comportamento anterior do sistema.

O desenvolvedor ágil só escreve a primeira linha de código, quando sabe o que o cliente realmente quer.

O processo: Escreve um pequeno pedaço de código; Roda; Testa Manualmente; Não Funciona; Coloca breakpoint; Entende o bug; Para o sistema; Esboça uma solução; Volta ao passo 1

Como você pode escrever um código de primeira com certeza absoluta que vai funcionar? Porque escrevemos código que não fazemos a minima ideia se vai funcionar?

Você operaria seu corpo com um médico que precisa ir no Google pesquisar qual ferramenta usar para fazer os procedimentos? Você comeria num restaurante onde os pratos são preparados com nenhuma higiene? Para se defender em um processo na justiça, você contrataria um advogado que nunca pegou casos semelhantes, precisa buscar petições similares no Google para copiar e colar?

Ciclo de Avaliação de Incertezas

Intelisense

Analise sintática em realtime

? - min(t) - !

Cliente, qual é o seu problema?

Se você não sabe a definição de pronto, quando vai terminar?

O levantamento de requisitos nunca termina

Eu nem escrevi a primeira linha de código e já tem que refatorar?

A refatoração existe antes de escrever a primeira linha de código

prototyping,

user stories, mockups, provas de conceito,

refactoring,

TDD,

BDD,

integração Contínua,

Lei Demeter

Command Query Separation

REST

Design Patterns

DDD

TDD

Refactoring

pair programmming

Code Review

Débito Técnico

Arquitetura Emergente

Melhorando o sistema

Quais incertezas estão impedindo de entregar software?

Como lidar com o código legado?

Apendice A: Um pouco de teoria.

Qual é o melhor processo de desenvolvimento de software?

Como desenvolver software de forma correta? Porque desenvolver software é tão trabalhoso? Como desenvolver software sem bugs? Afinal qual é o jeito certo de desenvolver software?

Essas perguntas são muito difíceis de encontrar uma resposta definitiva e geralmente a resposta ou é incompleta ou é imprecisa, porém como um líder de uma equipe com 30 profissionais que desenvolvem software todos os dias, responder esta pergunta é imperativo para produzir os melhores resultados. (Além de ter certeza que não estou louco fazendo um trabalho sem sentido)

A minha humilde tentativa de colocar minha visão por escrito, também é uma forma de compartilhar e raciocinar ao mesmo tempo sobre uma coisa que faço a mais de 15 anos e nunca me convenci da minha própria definição sobre os pilares da profissão que exerço.

Se você é como eu e geralmente desenvolve software, onde você precisa ser quase um jornalista para levantar com clareza os problemas do usuário, ou pessoas usam o

software que você faz como solução das suas atividades diárias, ou sua empresa o vende como produto, então esse artigo é para você.

Qual a base do meu raciocínio?

Antes de começar a explicar a minha visão, quero deixar claro que as perguntas acima expõem um erro conceitual, onde se busca fazer do software uma ciência exata, e não é.

Ciências exatas buscam estabelecer uma relação causa-efeito onde um conjunto de informações de entrada sempre dará o mesmo resultado na saída.

Uma ciência humana é qualquer campo da ciência onde os conhecimentos são criteriosamente organizados para tratar dos aspectos do ser humano como indivíduo e como ser social

(http://pt.wikipedia.org/wiki/Ci%C3%AAncias_humanas)[]

Análise a frase a seguir e veja se define nossa profissão.

Desenvolver software é o ato de escrever um texto que uma máquina é capaz de transforma-lo em utilidade para seu usuário.

o que acha? fez sentido? Repare que essa mesma frase também define os compositores, os roteiristas, os advogados (em alguns casos), os tradutores, os redatores, etc. Todas essas profissões são de humanas, o desenvolvimento de software hoje em dia se distanciou da engenharia igual a arquitetura um dia se distanciou, por mais que a engenharia seja essencial (e sempre será) para determinar a melhor maneira de executar um projeto, são as características arquitetônicas que o usuário usa.

O software é muitas vezes escrito de acordo com a personalidade do autor, seu entendimentoda necessidade, sua capacidade lógica no momento, sua vontade de aplicar as últimas técnicas (nem sempre necessárias), entupir de comentários TODO, declarar as variáveis apenas com siglas, porém a essência geralmente é negligenciada e se esquece que o código precisa resolver um problema, materializando um fluxo de informações e será constantemente revisado e evoluído por outras pessoas.

A relação entre como os seres humanos entendem o problema e o desenho da solução vai determinar o software final, muito mais do que predições precisas ou métodos

rigorosos de desenvolvimento. Eu fiz questão de no começo em falar no uso de software como produto, pois além do software fazer o necessário para informar o resultado, existe o desafio de apresentar este resultado com o menor atrito possível na interação entre o humano e a máquina.

Os cenários que descrevi acima além de já ser complicado por si só, ainda coloca uma porção de incertezas que as pessoas subestimam e faz o complicado virar complexo, ou seja, você não conseguirá pensar tudo antes de começar pois as incertezas são infinitas, então precisamos mudar a abordagem para planejar o mínimo viável, fazer/experimentar, analisar e corrigir. (muito similar ao que a ciência usa para provar uma teoria que ela formulou, mas não sabe o que pode acontecer)

A minha metodologia pessoal busca a todo momento eliminar as incertezas do sistema (entenda sistema como toda relação de pessoas que são fonte de informações necessárias a desenvolvimento de software), a forma de medir as incertezas se chama entropia.

Essas incertezas podem surgir de várias maneiras, por exemplo:

Qual é o requisito mínimo para o software funcionar?
Qual o resultado esperado se entrar com essas informações?
A sintaxe deste código está correta?
Mesmo se o código funcionar é o que o usuário espera?
Essa linha de código que escrevi funciona?
Está funcionando na minha máquina, será que vai funcionar em produção?

Não se iludam, essas perguntas são infinitas e tentar catalogar para criar um processo de desenvolvimento de software é novamente cair na falácia de ciência exata, então eu busco criar um ambiente onde essas perguntas sejam respondidas o mais rápido possível.

Quem já não desenvolveu baseado num documento, que o próprio cliente deu seu aceite e depois o próprio cliente disse que não funciona para seu cenário, pois o cenário mudou?

E quando para entregar um software no prazo, coloca alguns débitos técnicos e depois nunca mais volta para acertar as contas? Tá funcionando? Não mexe.

Desenvolver software é como criar uma máquina que realiza um trabalho específico, para uma necessidade específica, se fizer errado, mal feito, com desleixo ou negligente, o preço a pagar vem rápido. Por exemplo:

O software funciona mas consome muito o hardware.
Software funciona mas a manutenção é cara, pois conserta uma coisa estraga outra.
Os usuarios reclamam de lentidão na aplicação.
Adicionar novas funcionalidades leva mais tempo que o usual.

(http://pt.m.wikipedia.org/wiki/Entropia)[]