Engenharia Clássica X Desenvolvimento Ágil

Este artigo aborda as dificuldades enfrentadas no processo de desenvolvimento de software, desde os seus primórdios aos dias atuais, observando o surgimento de um novo modelo de desenvolvimento, o desenvolvimento ágil, que promete diminuir as lacunas encontradas na engenharia de software clássica, mas que ainda encontra as mesmas dificuldades deste último, a comunicação das partes.

De que se trata o artigo

Este artigo aborda as dificuldades enfrentadas no processo de desenvolvimento de software, desde os seus primórdios aos dias atuais, observando o surgimento de um novo modelo de desenvolvimento, o desenvolvimento ágil, que promete diminuir as lacunas encontradas na engenharia de software clássica, mas que ainda encontra as mesmas dificuldades deste último, a comunicação das partes.

Em que situação o tema é útil

Este artigo é útil para todas as partes envolvidas no desenvolvimento de software. Não só a parte da equipe de T.I., mas clientes que desejam adquirir uma solução informatizada para seu negócio, pois um dos pontos que mais contribui para o insucesso de projetos de software é a falha de comunicação entre as partes.

Engenharia Clássica X Desenvolvimento Ágil

O processo de desenvolvimento de software é muitas vezes considerado uma arte. Nos primórdios, era a arte de conseguir produzir um software de conteúdo relevante em um hardware de capacidade limitada. Atualmente é a arte de conseguir fazer com o que o software cumpra com o que foi prometido ao cliente. A engenharia de software clássica, em tese, é um modelo de desenvolvimento a ser aplicado para garantir qualidade ao software, mas problemas constantes como prazos, falha de requisitos e confiança do cliente, fez com que o manifesto ágil fosse elaborado; um método de alta interatividade com o cliente, mas que ainda assim não resolve um problema em comum: a maturidade tecnológica.

Autores: Bruno Manguinho, Francisco Matos Pereira e Rodrigo Oliveira Spínola

Software: um problema no horizonte

Começaremos o artigo utilizando uma frase de Roger S. Pressman publicada em seu livro Engenharia de Software no ano de 1992: “Percebe-se que o processo de desenvolvimento de software naquela época já encontrava dificuldades na entrega da qualidade do produto e que este seria um problema que ainda assombraria por muitos anos os desenvolvedores de software, e por que não, os clientes que pagam por ele”.

O software é um produto feito para ser executados em máquinas tecnológicas e que, seguindo um determinado conjunto de ações, atende as necessidades do usuário final. Com base nesse conceito, é fácil imaginar que uma equipe de T.I. composta por profissionais competentes, operando computadores com ferramentas de desenvolvimento, tendo em mãos documentos com as especificidades que o software deve seguir, o produto final será entregue seguindo suas conformidades e o cliente sairá satisfeito.

Entretanto, trazer isso para a realidade não é fácil como parece. A engenharia de software é uma área do conhecimento que vem, a cada dia, aprimorando as técnicas de desenvolvimento de software. O desenvolvimento ágil faz parte da engenharia de software, priorizando algumas métricas e minimizando (quase extinguindo) outras. Neste contexto, este artigo apresentará uma breve introdução a essas duas áreas (engenharia clássica e ágil), primeiro explanando um pouco sobre a engenharia clássica, abordando de forma sumária alguns ciclos de vida e o processo de desenvolvimento. Depois discutiremos o desenvolvimento ágil, explicando como surgiu o seu manifesto, as propostas de melhoria em relação ao modelo clássico e algumas práticas aplicadas do modelo. Por fim, iremos comparar os dois modelos de desenvolvimento, apontando suas vantagens, desvantagens e seus pontos em comum, além de uma ponderação final a respeito desta comparação.

Engenharia de Software

A engenharia em si é um antigo campo de estudo que abrange diversos ramos, tais como, engenharia civil, elétrica, mecânica, robótica, química, etc., que “num sentido amplo é a aplicação da ciência de maneira econômica para as necessidades humanas”. A engenharia de software não é diferente de nenhuma das engenharias citadas anteriormente, ela utiliza desses princípios econômicos para a construção de um produto que funcione em máquinas reais, que seja confiável e agrade às necessidades do cliente.

A preocupação com o estudo do processo de desenvolvimento de softwares iniciou-se com o avanço exponencial da capacidade do hardware, o qual levou empresas de todos os portes a adquirirem máquinas a um baixo custo para automatizar seu desenvolvimento. Como consequência, surgiu a necessidade de aquisição de softwares customizáveis que auxiliassem na tomada de decisões da organização.

Por que Engenharia?

Criar um software que atenda completamente a todas as necessidades do cliente não é uma tarefa simples. Os requisitos funcionais devem ser apresentados pelo cliente, a

avaliação destes é realizada para transformá-los em linguagem de máquina, passando então pela análise seguida da escrita do código, o período de testes, aceitação do cliente e, por fim, a implantação. Este é um cenário romântico de desenvolvimento de software, pois, se em pelo menos uma das etapas descritas houver uma falha, toda a estrutura do software será abalada e o produto final terá pouca ou nenhuma serventia.

Observa-se então a necessidade de se seguir um padrão de desenvolvimento utilizando os princípios de engenharia para construir um software de qualidade. Vários modelos de ciclo de vida de desenvolvimento foram estudados e aplicados. Com o passar do tempo, esses modelos precisaram ser aperfeiçoados ou ficaram obsoletos por não conseguir atingir um nível de qualidade desejado. A seguir, uma breve explanação dos principais ciclos de vida.

Modelo de ciclo de vida em cascata

O modelo cascata foi o primeiro modelo idealizado e surgiu na década de 70, onde toda a programação era feita de forma estrutural (a orientação a objetos só veio surgir na década de 90). O modelo de desenvolvimento era fundamentado em seis fases:

• Levantamento de Requisitos: Nesta fase todos os envolvidos no projeto devem ter a compreensão do problema a ser resolvido;

• Análise de Requisitos: Com base nos requisitos levantados, é feita a análise de todo o projeto antes que se escreva qualquer tipo de código. Costuma ser a fase que consome mais tempo do projeto;

• Projeto: é a fase de escolha da representação do software antes que a codificação se inicie (arquitetura do software, estrutura de dados, interfaces);

• Implementação: É a escrita do código em linguagem de máquina com base na análise realizada previamente;

• Testes: é a fase na qual o software passará por avaliações para medir sua qualidade;

• Implantação/Manutenção: fase de implantar o software para uso do usuário final.

O ciclo de vida cascata foi o mais utilizado durante muito tempo. Porém, com o passar do tempo, foi apresentando algumas dificuldades. Uma de suas premissas era de que apenas uma fase do modelo pode estar em execução em um dado momento e o ciclo deve seguir na ordem previamente definida pelo modelo. Ocorre que no levantamento de requisitos nem sempre (quase nunca) o cliente consegue especificar tudo o que deseja, necessitando de uma interatividade entre as fases. Além disso, o modelo exige paciência do cliente, pois provavelmente este só verá o sistema funcionando na fase de testes.

Observem que a última fase não é apenas a implantação, é adicionada nesta fase a manutenção do software, visto que holisticamente, o software apresentará erros (de código ou de regra de negócio mal definida) ou o cliente solicitará melhorias, ou algum requisito não foi passado na fase de especificação. Enfim, o modelo exige esta fase, na

qual voltará a todo o início do ciclo, consumindo mais tempo e mais paciência do cliente.

Fica claro que com o tempo o modelo foi deixando de ser utilizado. Apesar disso, ele pode ser considerar um modelo padrão para todos os modelos que se seguiram.

Prototipação

No modelo em cascata o cliente não tinha nenhuma visão do software funcionando até que se iniciasse a fase de testes ou na implantação. Até então ele só tinha fornecido os requisitos necessários do software em algum modelo de documento. Acontece que só então, nestas últimas fases, é que ele perceberia que algo está errado ou não foi especificado e todo o ciclo era recomeçado com os novos requisitos.

Foi proposto então o modelo de prototipação, onde em um determinado período de tempo o cliente tem uma parte do modelo do software em execução. Desta forma, fica mais fácil a identificação de problemas ou de novos requisitos e o software, à medida que evolui, ganha mais a “paciência” do cliente.

Este modelo, assim como o cascata, segue um ciclo em uma quantidade de tempo menor. O levantamento e análise dos requisitos são feitas de forma bem rápida, quase não há uma separação entre estas fases. O mesmo acontece com as fases de projeto e implementação, que são montados rapidamente para que a construção do protótipo seja apresentado ao cliente e que este faça as suas críticas. Neste momento, a fase de testes é realizada no ato e depois com a manutenção o ciclo volta ao início.

A prototipação ganha mais interatividade entre os ciclos de vida e com o cliente, e é um modelo largamente utilizado atualmente. Porém, pouco tempo é dado à fase de análise. A fase de testes também não é efetuada pragmaticamente. A depender da quantidade de ciclos que o modelo necessitar, seu prazo será encurtado e os testes de integração poderão não ser aplicados, diminuindo a confiabilidade e aceitação do software.

Desenvolvimento Ágil

Como visto, a metodologia de desenvolvimento na engenharia clássica causava altos índices de desconfiança e insatisfação dos clientes devido à demora de resposta que este obtia na solicitação de seu produto.

Idealizado em 2001 por Kent Beck e outros 16 desenvolvedores, o desenvolvimento ágil não faz um planejamento em longo prazo, tudo é subdividido em pequenos períodos (15 a 30 dias) para determinar o que deve ser feito, como deve ser feito, qual a prioridade, o prazo e a construção de um protótipo para apresentação. Este ciclo é chamado de sprint. Cada sprint é uma pequena parte do sistema que deve ser feita para concluir determinada operação. Se o resultado final não for satisfatório, a sprint é recomeçada, avaliando o que deu errado, o que foi sugerido e o que deve ser feito. É a resposta a mudanças do desenvolvimento ágil, que deve ser feita de forma mais pragmática, ao invés de refazer todo um replanejamento, com documentações longas e clientes insatisfeitos.

Sua principal premissa é: o software tem que estar funcionando. Apesar de isso parecer meio óbvio, na engenharia clássica isso parecia não ser considerado. A sprint é o principal meio para conseguir isso, pois ela consegue identificar o que é mais importante para o cliente, aumentando a comunicação com este. Um elemento importante do ágil é o Product Owner (PO), que é a pessoa que conhece o sistema e toda a sua regra de negócio. Ele é o elo entre o cliente e os desenvolvedores. Quando o cliente necessita passar especificações, mudanças e discutir as regras do sistema, o PO é a pessoa com quem ele interage. A partir desta interação, o PO traduzirá aos desenvolvedores exatamente como o sistema deve funcionar. Pode-se dizer que é uma espécie ágil de levantamento de requisitos. Qualquer dúvida que o desenvolvedor venha a ter sobre o negócio do produto, é com o PO que ele deve conversar.

O desenvolvimento ágil, ao contrário do que possa parecer, utiliza técnicas da engenharia clássica. Todas as fases do ciclo de vida clássico são usadas, mas de uma forma mais sucinta. A fase de planejamento e análise acontece todos os dias, na citada reunião diária de curta duração. Não há a modelagem completa do sistema, mas há a visão de uma parte do sistema que deve ser entregue na sprint.

A seguir vamos fazer um comparativo entre esses dois paradigmas de desenvolvimento para ficar mais clara a diferença mais latente entre eles.

Engenharia Clássica X Desenvolvimento Ágil

Quando Kent Beck e seus comparsas idealizaram o desenvolvimento ágil, ressaltaram as seguintes premissas:

• *Indivíduos e operações* são mais importantes que *processos e ferramentas;*

• *Softwares funcionando* são mais importantes que *documentação abrangente;*

• *Colaboração do cliente* é mais importante que *negociação de contratos*;

• *Respostas a modificações* é mais importante que *seguir um planejamento.*

Os itens menos importantes são as principais premissas da engenharia clássica e, ainda que classificados dessa maneira, não significaria a sua nulidade no ciclo de desenvolvimento. Entretanto, os itens citados como mais importantes significaria em um maior retorno quanto à qualidade final do software. Vamos avaliar cada um desses itens descritos como os mais importantes.

O termo indivíduos está relacionado à equipe de tecnologia responsável pelo desenvolvimento de software. Como em qualquer outra área, deve haver profissionais competentes de desenvolvimento, um gestor de negócios e o PO que está sempre interagindo entre o cliente e a equipe de desenvolvimento.

Assim como na engenharia clássica, o software tem de estar funcionando, só que no modelo clássico muitas etapas eram feitas antes, consumindo muito do tempo total do projeto, e quando o software era entregue, algumas vezes não satisfazia o usuário final.

O desenvolvimento ágil considera o desenvolvimento das funcionalidades do software que mais agregam valor para cliente.

Para ilustrar esta metodologia vamos pensar em um workflow de um sistema básico de compras de materiais. O cliente tem a necessidade de obter um ou mais insumos de material em uma quantidade desejada. A Solicitação é cadastrada no sistema, o responsável avalia a solicitação, verifica qual a quantidade de cada insumo em estoque e retorna para o sistema qual a quantidade de cada item que deve ser comprada e aprova a solicitação. Uma vez aprovada a solicitação, uma Coleta é gerada com os insumos e as quantidades a serem compradas de cada um. Então será feita a cotação dos fornecedores, informando os valores relativos a cada insumo, condição de pagamento, prazo de entrega, etc. Com base nessas informações, um mapa de coleta é gerado informando ao usuário qual fornecedor é mais vantajoso adquirir cada um desses insumos. O usuário assim escolhe os fornecedores, o responsável pela coleta aprova a coleta e, por fim, é gerado um Pedido de Material. No recebimento deste pedido, a nota fiscal vai ser informada no sistema para que haja então uma integração com o módulo financeiro, gerando assim um contas a pagar.

Todo este workflow seria realizado passo a passo pela engenharia clássica, partindo da solicitação até a integração com o módulo financeiro. Na prototipação era apresentado ao cliente cada passo e, mesmo no caso mais otimista imaginável de não haver nenhum erro, o cliente era obrigado a esperar a integração com o financeiro para avaliar o software. Acontece que em um sistema de compras, muitas vezes não há tempo para fazer uma solicitação, quem dirá uma coleta. O cliente já tem exatamente a quantidade, os fornecedores e os valores de cada item. O desenvolvimento ágil, neste exemplo, começa a desenvolver o módulo de compras diretamente do cadastro de pedidos e recebimento destes (existem casos de já receber o pedido no momento de sua inclusão).

Este modelo de desenvolvimento parte do ponto que mais agrega valor ao cliente: a integração do módulo de compras com o financeiro. Desta forma, fica mais fácil para o cliente avaliar o software e dar mais confiança, aumentando assim a colaboração do cliente com o desenvolvimento do software. Em alguns casos, o cliente pode pedir para que a solicitação e a coleta sejam suprimidas do contrato.

Essa situação reflete, de certa forma e considerando uma visão generalista, a diferença básica entre o desenvolvimento ágil e a engenharia clássica. Mas será que o desenvolvimento ágil é a resposta para tudo? Vamos fazer algumas considerações. A principal característica da engenharia ágil é o acompanhamento frequente do cliente, tornando-o uma peça fundamental no desenvolvimento de software. Outro fator do modelo ágil é que uma Sprint só é iniciada quando a Sprint anterior estiver completa e com a aprovação do cliente.

Estas duas características apresentam uma visão muito “romântica” no desenvolvimento de software. O cliente não tem disponibilidade de estar sempre acompanhando cada Sprint do software. O modelo também apregoa que todos os requisitos do software vão conseguir ser passados e captados pelo PO. É muito comum, no nosso exemplo, que após finalizado o pedido e o recebimento do mesmo e depois quando a equipe começar a desenvolver a solicitação, percebam que faltam requisitos que não foram especificados no pedido e que é necessário para a conclusão do módulo. Então o cadastro do pedido

sofrerá manutenções do mesmo jeito que a engenharia clássica, e sem o acompanhamento do cliente, a aflição crônica da manutenção de software permeará.

É claro que o modelo ágil apresenta uma evolução, mas ainda há muito o que ser estudado e otimizado.

O que falta à Engenharia de Software?

Uma comparação muito comum que se pode fazer é entre a engenharia de software e a engenharia civil. Antes de iniciar a fase de construção de um empreendimento, é feita uma análise do solo e a devida preparação do mesmo. Depois é iniciada a parte de fundação do empreendimento, onde todo um planejamento é feito, calculando o peso de cada pavimento que a estrutura deve suportar, para só assim iniciar a fase de projeto. Este modelo é muito similar ao modelo cascata, onde cada fase é executada em separado até a entrega do produto final. Não é possível assim, planejar a fundação e iniciar a fase de projetos ao mesmo tempo.

Este modelo funciona para a engenharia civil, afinal, é uma ciência muito antiga e que muito já se foi estudado. Funciona tão bem que o cliente compra o empreendimento antes mesmo de qualquer uma dessas fases de construção, pois ele sabe que o produto final será entregue. Na engenharia civil, não há espaço para o cliente pedir a entrega do empreendimento com um prazo menor do estipulado. Simplesmente não há. O cliente tem esta maturidade de quanto tempo leva pra ser feito, de quanto custa e de que o produto final será de qualidade.

O que falta então a Engenharia de Software? Estamos estudando e nos aperfeiçoando cada vez mais para construir softwares de qualidade, mas a aflição crônica ainda não está nem perto de acabar. É claro que é uma ciência nova e que há um longo caminho pela frente. O desenvolvimento ágil já mostrou que é de fundamental importância a participação do cliente na construção de software. Talvez, se houvesse nos clientes a mesma maturidade que ele tem para a engenharia civil, ou seja, uma maturidade tecnológica, nossos problemas começassem a diminuir.