LIGA DE ENSINO DO RIO GRANDE DO NORTE
CENTRO UNIVERSITÁRIO DO RIO GRANDE DO NORTE
CURSO DE PÓS GRADUAÇÃO ESPECIALIZAÇÃO EM GESTÃO FISCAL E TRIBUTÁRIA

DESENVOLVIMENTO DO SOFTWARE GERADOR DO ARQUIVO SPED: ESTUDO DE CASO DE UMA EMPRESA DESENVOLVEDORA DE SOFTWARES

Clean Cordeiro de Lima[1]
Joana D'arc Medeiros Martins[2]

RESUMO

Na atualidade, existem cada vez mais softwares controlando processos, seja em órgãos públicos ou empresas privadas. Vendo essa cena é necessário que os desenvolvedores de softwares tracem e definam a melhor metodologia para o desenvolvimento de seus projetos. Onde é imprescindível a engenharia de software que é a parte da informática que se tem o conhecimento e a especificação do desenvolvimento e manutenção do software. Onde se deve estudar e visualizar qual a melhor metodologia de desenvolvimento para determinado projeto, podendo até usar várias metodologias em um único projeto, procurando sempre a qualidade no processo de desenvolvimento, o compromisso das pessoas envolvidas no processo e a qualidade final do produto. Este trabalho mostrará um estudo de caso de todo o processo de desenvolvimento do software gerador do arquivo SPED (Sistema público de escrituração digital) em uma empresa, mostrará as metodologias usadas, o levantamento de requisitos contido no guia prático da escrituração fiscal digital - EFD (conforme Ato COTEPE/ICMS no 09, de 18 de 2008 e alterações), as pessoas envolvidas e o produto final.

Palavras-chaves: SPED, metodologias ágeis, engenharia de software.

[1] Discente do Curso de Pós Graduação Especialização em Gestão Fiscal e Tributária do Centro Universitário do Rio Grande do Norte.
[2] Docente e Orientadora do Curso de Pós Graduação Especialização em Gestão Fiscal e Tributária do Centro Universitário do Rio Grande do Norte.

ABSTRACT

Currently, there are increasingly controlling software processes, whether in public or private companies. Seeing this scene is necessary for software developers to devise and define the best methodology for the development of their projects. Where is the indispensable software engineering that is part of the computer that has the knowledge and specification development and maintenance of software. Where should study and see what the best development methodology for a particular project and may even use different methodologies in a single project, always looking for quality in the development process, the commitment of the people involved in the process and final product quality. This work shows a case study of the whole process of software development file generator SPED (Public system of bookkeeping digital) in a company, show the methodologies used, the survey requirements contained in the Practical Guide to digital book keeping - EFD (Act as COTEPE / GST at 09, 18, 2008 and amendments), the people involved and the final product.

Keywords: SPED, agile methodologies, software engineering.

1 INTRODUÇÃO

Com o grande avanço de infraestrutura da tecnologia da informação (TI) no Brasil, as empresas pautam suas estratégias de negócios tendo como base o uso da TI, seja para melhorar ou automatizar seus processos internos e externos.

Vendo esse cenário, o Governo Federal Brasileiro criou o SPED (Sistema Público de Escrituração Digital) que faz parte do PAC (Programa de Aceleração do Crescimento) onde se constitui mais um avanço na informatização da relação entre o fisco3 e os contribuintes. Onde seu maior objetivo é promover a integração dos fiscos, mediante a uma padronização e a um compartilhamento das informações. Foi a partir dessa nova obrigação, que uma empresa teve o dever de desenvolver um software gerador do arquivo SPED.

O desenvolvimento de software é caracterizado pelo conjunto de componentes encapsulados como: procedimentos, funções, módulos, objetos, os quais compõem a arquitetura do software. A definição de metodologias no processo de desenvolvimento oferece uma forma mais fácil no acompanhamento do progresso do projeto.

Neste trabalho será apresentado um estudo de caso sobre o desenvolvimento do software gerador do arquivo SPED em uma empresa desenvolvedora de softwares, onde abordará as principais metodologias, ferramentas, dificuldades e obrigações para o seu desenvolvimento.

2 REFERENCIAL TEÓRICO

2.1 ENGENHARIA DE SOFTWARE

Segundo Baues apud Araújo (2008) a engenharia de software “é a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe eficientemente em máquinas reais”. Segundo Martin e McClure apud Rezende (2002, p. 2), “engenharia de software é o estudo dos princípios e sua aplicação no desenvolvimento e manutenção de sistemas de software”. Basicamente, o que a engenharia de software tenta buscar em seus principais objetivos é alcançar a melhor qualidade do produto do software e o aumento da produtividade em seu desenvolvimento.

2.1.1 Processos de Software

Segundo Fuggetta apud Ribeiro (2009, p. 40), define processo de software “como um conjunto coerente de atividades, políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos necessários para conceber, desenvolver, dispor e manter um produto de software”. Os paradigmas diminuem os problemas encontrados durante o processo de desenvolvimento do software. Ainda Carvalho (2001, p. 29) “O paradigma é escolhido de acordo com a natureza do projeto e do produto a ser desenvolvido, dos métodos e ferramentas a ser utilizados e dos controles e produtos intermediários desejados”. Devido a sua grande importância já foram proposto vários paradigmas, a seguir serão abordados alguns desses paradigmas.

2.1.1.1 O Ciclo de Vida Clássico

O paradigma do ciclo de vida clássico tem um lugar definido e importante no trabalho da engenharia de software. “Ele produz um padrão no qual os métodos para análise, projeto, codificação, teste e manutenção podem ser colocados” (PRESSMAN, 1995, p. 35). É o paradigma clássico da engenharia de software, que consiste num desenvolvimento sequencial do software, onde se inicia pela engenharia, passa pela análise, projeto, codificação teste e manutenção.

O ciclo da engenharia convencional que abrange as seguinte atividades: *Análise da engenharia:* Como um software sempre faz parte de um sistema mais amplo,

primeiramente deve-se estabelecer os requisitos para todos os elementos do sistema, tais como hardware, pessoas e bancos de dados. Essa análise sempre envolve os requisitos em nível de sistema; *Análise de requisitos*: É processo de levantamento dos dados, para que seja entendido o porquê e o pra que do software, bem como desempenho e interface. Onde, se é documentado e revistos com o cliente; *Projeto*: É um processo onde se existe vários passos distintos, que são basicamente: estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização de interface; *Codificação*: É o processo de escrita do software; *Teste:* Depois do processo de codificação, dar-se inicio ao processo de teste onde primeiro testa os aspectos lógicos internos do software, garantindo que toda as instruções escritas estejam sem erros, e depois é testado os aspectos funcionais externos, ou seja, garantindo que uma entrada definida retorne resultados reais que esteja coerente com os resultados exigidos; *Manutenção*: Sempre o software receberá mudanças após a entrega para o cliente, pois erros foram encontrados devido a mudanças no ambiente externo (como por exemplo, o cliente exigiu acréscimos funcionais devido uma mudança na lei de ICMS4), sendo assim, se reaplica cada uma das etapas do ciclo de vida a um programa já existente, e não a um novo.

2.1.1.2 Técnicas da Quarta Geração (4GT)

O paradigma técnicas de quarta geração (4GT) utiliza um amplo conjunto de ferramentas, ferramentas essa que tem características em comum, que possibilitam que o desenvolvedor informe características do software, onde a ferramenta irá gerar automaticamente o código-fonte, tendo como base as informações do desenvolvedor.

O ambiente de desenvolvimento que sustenta o 4GT inclui as seguintes ferramentas: linguagens não procedimentais para consulta de banco de dados, geração de relatórios, manipulação de dados, interação e definição de telas, geração de códigos, capacidade gráfica de alto nível e capacidade de planilhas eletrônicas.

Os primeiros dados coletados em relação ao 4GT informam que o tempo, a quantidade de planejamento e a análise exigida para se produzir um software é reduzido para sistemas pequenos e intermediários. Já em relação a grandes sistemas, se exige tanto ou mais análise, planejamento e teste. O 4GT já se tornou importante no desenvolvimento de software.

2.1.1.3 Combinando Paradigmas

Em todos os paradigmas vemos que o primeiro passo é a obtenção dos requisitos. A partir desse passo qualquer caminho pode ser tomado, onde vemos a utilização do ciclo de vida clássico, do 4GT, da prototipação e do modelo espiral sendo utilizados praticamente no mesmo caminho. “A natureza da aplicação deve ditar a abordagem a ser tomada. Ao combinarmos abordagens, o todo pode ser maior que a soma das partes” (PRESSMAN, 1995, p. 45). Independente do paradigma de engenharia de software escolhido, o processo de desenvolvimento contém três fases genéricas, que são: definição, desenvolvimento e manutenção que são encontradas em todo e qualquer desenvolvimento de software, independentemente da área de aplicação, tamanho do projeto ou complexidade.

2.1.2 Metodologia de Desenvolvimento de Software

Maddison apud Roque (1998) define metodologia “como sendo um conjunto recomendado de filosofias, fases, procedimentos, técnicas, regras, ferramentas, documentação, gerenciamento e treinamento para o desenvolvimento de um sistema de informação”. As metodologias podem ser adaptadas para cada tipo de projeto, de acordo com seu ambiente, complexidade, prazo e nível econômico. As vantagens obtidas com o uso das metodologias são: ganho de produtividade, documentação, padronização e organização.

2.1.2.1 Metodologias Ágeis

O surgimento da definição de metodologias ágeis de desenvolvimento de software foi iniciado durante os anos 90, que foi uma reação contra os desenvolvimentos definidos como “pesados”, pois eram vistos como burocráticos lentos e colocava em contradição à ideia que o trabalho do engenheiro de software é eficaz.

Para Sato (2008, p. 23) “Os métodos ágeis apresentam uma abordagem bastante pragmática para o desenvolvimento de software”. Metodologias ágeis geralmente promovem um processo de gerenciamento de projeto que realiza revisões frequentes e adaptação, define uma filosofia de liderança que encoraja trabalho de equipe, auto-organização e responsabilização, utiliza um conjunto de boa práticas de engenharia que permitem entregas rápidas e alta qualidade de software e uma estratégia de negócios que alinha desenvolvimento com as necessidades do cliente e objetivos da empresa.

2.1.2.2 Scrum

Scrum, que é fundamentado na teoria de controle de processos empíricos, emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e controlar os riscos. O scrum permite manter o foco na entrega do maior valor de negócio no menor tempo possível, permitindo à rápida e contínua inspeção do software em produção, onde a necessidade do negócio determinam a prioridade do desenvolvimento, as equipes se auto organizam para definir a melhor maneira de entregar as funcionalidades de maior prioridade (SCHWABER, 2009). No scrum existe o processo prescritivo que funciona em ambientes controlados e o processo empírico que funciona para processos complexos e imprevisíveis (SATO, 2008).

2.1.2.2.1 Artefatos

Backlog do produto: Para Spenassato (2005) essas funcionalidades são priorizadas de acordo com os usuários e interessados, e estimadas em termos de tempo e de custo pela equipe de desenvolvimento. Essa fase é uma das mais importantes do scrum, é a fase que são coordenadas as reuniões com todos os envolvidos no projeto, no intuito de definir os requisitos funcionais e técnicos. O backlog do produto é uma lista das prioridades de tudo que é necessário no produto.

Sprint: Segundo Ferreira (2005, p. 3) o sprint tem um “Período não superior a 30 dias, onde o projeto (ou apenas algumas funcionalidades) é desenvolvido”. Para Zanatta (2004) e Schwaber (2009) O sprint é a principal etapa do scrum. É nessa que fase que se executa todas as tarefas definidas no backlog do produto. Para Spenassato (2005) o objetivo de um sprint é produzir um incremento para o produto, o qual possa ser entregue ao cliente e lhe tenha alguma significância.

Backlog do sprint: Para Ferreira (2005), consiste no trabalho a ser desenvolvido em um sprint de modo a criar um produto a apresentar ao cliente. Se existir um backlog anterior o próximo deve ser desenvolvido de forma incremental. Spenassato (2005, p. 87) afirma que “uma vez o sprint iniciado, somente a equipe de desenvolvimento pode alterar esta lista, adicionando ou removendo atividades”. O backlog do sprint é uma backlog do produto que através do sprint é transformado em um incremento do produto final.

Daily scrum: Para Schwaber (2009) daily scrum é uma reunião diária que dura de 15 há no máximo 30 minutos, onde é avaliado o que foi realizado desde a última reunião, o

que será feito antes da próxima reunião e quais as dificuldades encontradas. Além, de discutir a priorizar a lista de backlog do produto. Rising (2002) afirma que as reuniões não ajudam na conclusão de nenhuma tarefa, porém, satisfazem vários objetivos, onde o mais importante deles é a concentração de esforços no backlog. Spenassato (2005) Ressalta que o scrum master, conceituado a seguir e o líder de projeto são responsáveis por conduzirem as daily scrum e por garantir que a reunião seja curta e cumpra com seus objetivos.

2.1.2.2.2 Papéis

Segundo Schwaber (2009) O scrum é composto por um time, onde existe um scrum master, um product owner e os membros. Os membros do time scrum são chamados de “porcos” e qualquer outra pessoa é chamada de “galinha”. “Galinhas” não podem dizer aos “porcos” como eles devem fazer seu trabalho.

Scrum master (SM): Para Schwaber (2004) as principais responsabilidades do SM estão relacionadas com a gestão do processo scrum. O SM Trabalha com os clientes e a gerência para identificar e designar um product owner, ajuda o time e a organização a adotarem o scrum, é responsável por garantir que o time esteja aderindo aos valores do scrum, as práticas e as regras, educa e protege o time contra o excesso de confiança, treinado-o e levando-o a ser mais produtivo, a desenvolver produtos de maior qualidade, a entender e usar o autogerenciamento e interdisciplinaridade. No entanto, o scrum master não gerencia o time, o time é auto-organizável. Zanata (2004) aponta o SM como o responsável pela realização das tarefas e pelo sucesso direto do projeto.

Product owner (PO): O PO é o responsável por definir quais são as funcionalidades que compõem um software. É uma pessoal e não um grupo podem existir grupos que influenciem o PO, mas quem quiser mudar a prioridade de algum item, tem que convencer o PO a mudar essa prioridade. Ele é o único responsável pelo gerenciamento da lista de tarefas priorizadas (backlog) do produto, garante o valor do trabalho do time, é o representante dos clientes, entende do negócio do cliente e define o objetivo do sprint. Ele mantém o backlog do produto visível, assim todos saberão qual o item que tem a maior prioridade, de forma que todos sabem em que se irá trabalhar (FABRI, 2012).

Time: Times de desenvolvedores transformam o backlog do produto em incrementos- de funcionalidades potencialmente entregáveis em cada sprint. Os membros do time possuem conhecimentos especializados, como programação, controle de qualidade, análise de negócios, arquitetura, projeto de interface de usuário ou projeto de banco de dados,

devem compartilhar o conhecimento de pegar um requisito e transformá-lo em um produto utilizável. Todos contribuem, mesmo que isso exija aprender novas habilidades ou lembrar-se das antigas (SCHWABER, 2009).

2.1.2.3 XP (Extreme Proramming)

O processo de metodologia ágil XP resultou da experiência no projeto C3 Payroll na empresa Chrysler, onde foi desenvolvido um sistema de folha de pagamento. Devido ao sucesso desse projeto o XP começou a crescer no meio acadêmico e empresarial, se tornando alvo de várias pesquisas e discussões, os métodos referentes ao XP foram idealizados na década de 80. Jeffries em 2001 que fala sobre os detalhes técnicos do XP, Beck e Fowler que no mesmo ano escreveram sobre o seu planejamento. O XP é uma metodologia ágil voltada para equipes de pequenas e médias empresas.

2.1.2.3.1 Valores

Feedback: Segundo Brooks apud Teles (2005, p. 57) “nenhuma parte do trabalho conceitual é tão difícil quanto estabelecer detalhadamente os requisitos técnicos, incluindo todas as interfaces (…) Nenhuma outra parte é mais difícil de corrigir”. A necessidade de compreender o usuário é uma das atividades mais difíceis e importantes a serem realizadas pelos desenvolvedores de software, pois ela direciona os demais esforços. Compreender os requisitos é difícil, como também é complexo para os próprios usuários transmiti-los corretamente.

Comunicação: Para Teles (2005) Projetos de software normalmente envolvem a presença de pelo menos duas pessoas, um usuário e um desenvolvedor, o que causa a necessidade de comunicação entre elas. Segundo Teles (2004, p. 48) “A forma de se transmitir uma idéia exerce uma grande influência na compreensão correta da mesma”. Em muitos projetos não a o envolvimento de apenas duas pessoas, mas sim o envolvimento de grupos maiores compostos por diversos usuários e desenvolvedores, onde freqüentemente existem os equívocos no processo da comunicação, onde causam desentendimentos de algum aspecto do projeto. Para Cockburn apud Teles (2005, p. 62) “A proximidade dos participantes auxilia os processos de comunicação”. Um fator que influencia a qualidade da comunicação é a quantidade de pessoas envolvidas, por esta razão, projetos XP procuram contar com um número reduzido de participantes.

Simplicidade: Beck (2000, p. 31) afirma que “Quanto mais simples é o seu sistema, menos você precisa comunicar sobre ele, o que leva à comunicação mais completa, especialmente se você for capaz de simplificar o sistema suficientemente a ponto de necessitar de menos programadores”. Existe uma relação de ajuda mútua entre a simplicidade e a comunicação, pois quanto mais se comunica mais claramente se é capaz de saber o que realmente precisa ser feito e se tem mais confiança sobre o que não precisa.

Coragem: Alguns temores assombram os participantes de um projeto de software. Clientes temem: não obter o que pediram ou pedir a coisa errada, pagar demais por muito pouco, jamais ver um plano relevante, não saber o que está acontecendo e fixarem-se em suas primeiras decisões e não serem capazes de reagir a mudanças nos negócios. Desenvolvedores temem: ser solicitados a fazer mais do que sabem fazer, ser ordenados a fazer coisas que não façam sentido, ficar defasados tecnicamente, não receber declarações claras sobre o que precisa ser feito, sacrificar a qualidade em função do prazo e não ter tempo suficiente para fazer um bom trabalho.

2.1.2.3.2 Práticas

Cliente presente: Para Teles (2005, p. 70) o cliente mais participativo “será capaz de obter o software desejado se a equipe de desenvolvimento não programar corretamente o que é pedido e a melhor equipe não será capaz de produzir o software certo se o cliente não for capaz de especificá-lo adequadamente”. O XP se preocupa com essa questão e busca solucionar trazendo o cliente para participar da equipe de desenvolvimento.

Jogo do planejamento: Segundo Yourdon (2004) as funcionalidades de um sistema podem ser categorizadas em “tem que ser feita”, “deveria ser feita” e “poderia ser feita”. Segundo Teles (2005, p. 72) “decidir o que implementar é uma das atividades mais importantes a serem conduzidas durante o desenvolvimento de um sistema”. O planejamento do XP assegura que a equipe sempre esteja trabalhando na parte mais importante do projeto, é considerado uma atividade contínua que deve ser desempenhada em todo o projeto. Existe uma monitoração diária em cima do progresso da iteração utilizando o quadro de acompanhamento diário, que visa determinar a velocidade da equipe em um determinada iteração e acompanhar às horas trabalhadas em cada tarefa.

Stand up meeting: Para que ocorra tudo bem no funcionamento do projeto, precisa do bom funcionamento de cada um de seus componentes, bem como a iteração entre os mesmos. Para assegurar que isso ocorra, o XP utiliza breve reuniões diárias chamada de

stand up meeting. “Um dia de trabalho de uma equipe XP sempre começa com um stand up meeting. (…) Primeiramente, ele serve para que todos os membros da equipe comentem rapidamente o trabalho que executaram no dia anterior” (TELES, 2004, p. 87). Essa prática gera uma visibilidade de tudo que está acontecendo na equipe para cada um dos membros, o que permite encontrar rapidamente os problemas e soluções encontradas no dia anterior. Basicamente, o stand up meeting é uma reunião que força a aproximação da equipe de forma rápida, diária e contínua.

Programação em par: Williams e Kessler apud Teles (2005, p. 79) afirma que programação em par “É um estilo de programação no qual dois programadores trabalham lado a lado em um computador, continuamente colaborando no mesmo design, algoritmo, código e teste”. Um aspecto importante sobre a programação em par é o fato de ter duas pessoas pensando sobre a solução do mesmo problema, fazendo com que exista mais cenários simples e eficaz para a sua solução. Escolhendo frequentemente soluções mais simples, ajuda na redução do tempo de desenvolvimento das funcionalidades e acelera o progresso da equipe.

Código coletivo: É a complementação da programação em par, as equipes XP também praticam o conceito de código coletivo. Para Jeffries e Anderson apud Teles (2005, p. 88) “todas as classes e métodos pertencem à equipe e qualquer membro da equipe pode melhorar o que for necessário”. O desenvolvedor tem acesso a todas as partes do código-fonte, inclusive as parte que ele não programou. “ninguém precisa esperar até que outra pessoa apareça para consertar alguma coisa” (JEFFRIES e ANDERSON apud TELES, 2005, p. 89).

Código padronizado: Segundo Jeffries e Anderson apud Teles (2005, p. 90) “Não importa muito o formato. O que realmente importa é que todos os membros da equipe adotem o padrão e o utilizem sempre. (…) não são as especificidades (…) que contam; é a familiaridade com elas”. A definição de um padrão ajuda a simplificar a comunicação, a programar em par e a tornar o código coletivo.

Design simples: Para Teles (2005, p. 92) “Existem pelo menos quatro coisas que os usuários de um software esperam dele: que faça a coisa certa, que funcione, que seja fácil de utilizar e que possa evoluir com o tempo”. As práticas adotadas pelo XP buscam atingir esses objetivos, sempre buscando assegurar que o sistema esteja funcionando e fazendo as coisas certas. “Um programa limpo e elegante tem que apresentar a cada um de seus usuários um modelo mental coerente (…). A integridade conceitual, como percebida pelo usuário, é o fator mais importante na facilidade de uso” (BROOKS apud TELES, 2005, p. 93). Equipes que utilizam XP sempre procuram assegurar que o design seja simples.

Desenvolvimento orientado a testes: Segundo Teles (2005, p. 99) “Sistemas computacionais e projetos de software costumam vivenciar diversas dificuldades ao longo do tempo. Um dos problemas mais caros e recorrentes é a incidência de defeitos”. Os testes em XP são feitos antes da programação, o XP trabalha com dois tipos de teste: teste funcional e teste de unidade. O teste de unidade verifica tudo que pode dá errado, já o teste funcional são usados para verificar junto ao cliente o sistema como um todo. “é impossível testar absolutamente tudo, sem que os testes se tornem tão complicados e propensos a erros quando o código de produção. É suicídio não testar nada. (…) Você deve testar coisas que possam vir a quebrar” (BECK, 2000, p. 116-117).

Refatoração: Para Poppendieck & Poppendieck apud Teles (2005, p. 108) sempre existirá refatoração, pois “sem melhoria contínua, qualquer sistema de software irá sofrer. As estruturas internas irão se calcificar e se tornar frágeis”. Os projetos XP sempre buscam o aprimoramento do design e a identificação dos pontos da arquitetura que estejam degradados, no momento que esses pontos são encontrados, são corrigidos através da refatoração que “é um processo de fazer mudanças em um código existente e funcional sem alterar seu comportamento externo. Em outras palavras, alterar como ele faz, mas não o que ele faz. O objetivo é aprimorar a estrutura interna” (ASTELS apud TELES, 2005, p. 110). Esse processo sempre é feito por todos os desenvolvedores, mantendo o código minimizado e diminuindo as chances de introduzir bugs. As equipes que utilizam XP, executam a refatoração sempre que são produzidas novas funcionalidades e quando se existe a necessidade de melhorar o código, como por exemplo: duplicação de código e código sem intenção clara.

Integração contínua: Segundo Poppendieck & Poppendieck apud Teles (2005, p. 114) “sempre que diversos indivíduos estão trabalhando na mesma coisa, ocorre uma necessidade de sincronização”. No XP esse problema é resolvido pela integração contínua, os pares trabalham separados e depois que toda produção é testada fazem a integração no servidor com a versão mais recente do código. Onde somente um par pode integrar o seu código por vez. Isto é, os pares se sincronizam com freqüência à medida que terminam pequenas atividades de codificação (BECK, 2000).

Releases curtos: Segundo Teles (2005, p. 118) “O XP procura maximizar o retorno dos projetos assegurando que o maior valor de negócio possível seja entregue ao final de cada release e que cada release tenha uma duração curta”. Isso é definido através da priorização que sempre seleciona as histórias de maior importância para serem implementadas primeiro. Releases curtos significa colocar o sistema em produção mais vezes, em menores prazos, que giram normalmente em torno de dois a três meses. “clientes gostam de entregas

rápidas. (…) entrega rápida normalmente se traduz em aumento de flexibilidade no negócio" (POPPENDIECK e POPPENDIECK apud TELES, 2005, p. 116).

Metáfora: Para Teles (2005, p. 122) a "utilização de metáforas é um mecanismo poderoso para manter a integridade conceitual de um sistema e, portanto, torná-lo mais fácil de ser utilizado". A intenção da metáfora é mostrar uma visão geral do sistema, em uma forma mais simples, que possa ser entendida por programadores e clientes. A idéia principal da metáfora é que seja feita uma analogia entre o sistema que está sendo desenvolvido e um sistema que todos entendam, com o objetivo de manter um vocabulário em comum para a criação dos nomes das classes, métodos, subsistemas e etc.

Ritmo sustentável: Na prática XP sempre se busca evitar fazer horas-extra, ele recomenda que os membros da equipe trabalhem apenas o tempo de oito horas por dia. Todo planejamento e prazo é definido (através de priorização) em cima dessa carga horária de trabalho. Para o XP o aumento da produtividade está ligado diretamente com uma melhor qualidade de vida dentro e fora da empresa.

Segundo Teles (2004, p. 21) o Extreme Programming, ou XP, é um processo de desenvolvimento de software voltado para:

- Projetos cujos requisitos são vagos e mudam com freqüência;
- Desenvolvimento de sistemas orientados a objeto;
- Equipes pequenas, preferencialmente de até 12 desenvolvedores;
- Desenvolvimento incremental (ou iterativo), onde o sistema começa a ser implementado logo no inicio do projeto e vai ganhando novas funcionalidades ao longo do tempo.

2.2 FERRAMENTAS

Delphi é um compilador e também uma IDE para o desenvolvimento de softwares. Ele é produzido e comercializado pela Borland Software Corporation. A linguagem utilizada pelo Delphi é a Object Pascal, que a partir da versão 7 passou a se chamar Delphi Language. O Delphi originalmente é direcionado para a plataforma Microsoft Windows (CANTU, 1998). O Delphi não pode ser usado para desenvolvimento de software de base ou aplicativos de sistema. Entre os engenheiros de software o Delphi é muitas vezes caracterizado como um "aplicativo programável". O Delphi é largamente utilizado no desenvolvimento de aplicações desktop e aplicações multicamadas (cliente/servidor) (CANTU, 1998).

O Firebird é um sistema gerenciador de banco de dados que surgiu a partir do Interbase 6. Ele tem o código aberto e não possui licença dupla, a tecnologia usada no

Firebird tem mais de 20 anos, fazendo com que ele seja um produto muito maduro e estável (BORRIE 2006).

A linguagem PSQL (Procedural SQL) é uma linguagem nativa do Firebird para stored procedures e trigger, procedimentos de grande importância em uma sistema gerenciador de banco de dados. Temos como algumas de suas características: Os blocos terminam com um END seguidos por um terminador, este que é definido com o comando SET TERM; Possibilidade de gerar exceções dinâmicas podendo criar e exibir uma mensagem no momento em que a exceção é gerada; Possibilidade de chamar stored procedures em triggers internamente através do comando EXECUTE PROCEDURE.

2.3 O SPED (SISTEMA PÚBLICA DE ESCRITURAÇÃO FISCAL)

Segundo a Receita Federal do Brasil (www1.receita.fazenda.gov.br/Sped), O SPED consiste na modernização da sistemática atual do cumprimento das obrigações acessórias, transmitidas pelos contribuintes às administrações tributárias e aos órgãos fiscalizadores, utilizando-se da certificação digital para fins de assinatura dos documentos eletrônicos, garantindo assim a validade jurídica dos mesmos apenas na sua forma digital.

Podemos definir o SPED como um arquivo TXT com todas as informações de cadastros e movimentações fiscais do contribuinte. Informações essas que são os próprios dados do contribuinte, dados do contabilista, clientes, fornecedores, produtos, unidades de medida, notas fiscais de saída e entrada, cupons fiscais, notas fiscais de serviço que contenham ICMS, ativo imobilizado do contribuinte, estoque produto a produto e apuração de impostos. Onde é apresentado na figura 7.

Principais objetivos do SPED: Promover à integração dos fiscos, mediante a padronização e compartilhamento das informações contábeis e fiscais, respeitadas as restrições legais; Racionalizar e uniformizar as obrigações acessórias para os contribuintes, com o estabelecimento de transmissão única de distintas obrigações acessórias e diferentes órgãos fiscalizadores; Tornar mais célere a identificação de ilícitos tributários, com a melhoria do controle dos processos, a rapidez no acesso às informações e a fiscalização mais efetiva das operações com o cruzamento de dados e auditoria eletrônica.

3 ANÁLISE DOS RESULTADOS

Será apresentado o estudo feito sobre o desenvolvimento do software gerador do arquivo SPED em uma empresa. Onde, foi analisado todo seu processo de desenvolvimento, desde a análise dos requisitos contidos no guia prático da escrituração fiscal digital - EFD (conforme Ato COTEPE[5]/ICMS no 09, de 18 de 2008 e alterações), como também, todo seu desenvolvimento, passando pelos processos e metodologias utilizadas.

O intuito principal do software é satisfazer as regras estabelecidas pela RFB[6], seguindo rigorosamente todos os passos contidos no guia prático da escrituração fiscal digital – EFD que contém aproximadamente 350 páginas, que vem descrito toda estrutura de layout e os campos que o arquivo SPED deve seguir e conter. O layout e a disposição dos dados dentro do arquivo deve seguir uma sequência exata, seguindo os seguintes blocos: bloco 0 abertura, identificação e referência, bloco C documentos fiscais I (ICMS/IPI[7]), bloco D documentos fiscais II (ICMS), bloco E apuração do ICMS e do IPI, bloco G controle de crédito de ICMS do ativo permanente – CIAP, bloco H inventário físico, bloco 1 outras informações e bloco 9 controle e encerramento do arquivo digital.

Após a leitura dos requisitos do software, foi dado inicio ao seu desenvolvimento utilizando apenas o Delphi 7 e o banco de dados Firebird, sem aplicação de qualquer metodologia ágil. Desenvolvimento esse que contou com apenas um desenvolvedor, devido ao pequeno quadro de funcionários da empresa, o que dificulta ainda mais a aplicação de qualquer metodologia ágil de desenvolvimento.

Das metodologias ágeis apresentadas no referencial teórico, escolheu-se o XP para fazer a comparação com o processo utilizado na empresa. O XP foi escolhido por poder ser aplicado a equipes de desenvolvimento menores e por ser aplicado a pequenas e médias empresas, assim não poderíamos utilizar o Scrum, que é uma metodologia utilizada para escopos que não estão bem definidos, onde seus requisitos mudam constantemente e é necessário no mínimo uma equipe de cindo desenvolvedores.

Compararam-se as fases do XP com as fases aplicadas pela empresa. **Quanto aos valores**: Feedback: Não foi utilizado quanto as requisitos funcionais do sistema, pois todo os requisitos já foram definidos no guia prático da escrituração fiscal digital – EFD. Já em relação à utilização do software existiu o feedback com os clientes da empresa, procurando

sempre com que o software ficasse de forma simples e que seu manuseio fosse fácil. Comunicação: A comunicação no processo da empresa existiu e foi mais forte em relação aos seus clientes que iriam utilizar o software. Simplicidade: Foi um dos valores que mais a empresa buscou no seu software, apesar de ter todos os requisitos já definidos a empresa fez com que o seu software fosse de simples manuseio para seu cliente. Coragem: Foi um valor que não ficou aparente no processo já que o cliente e o desenvolvedor não tinham o que temer, pois todo o software foi definido e desenvolvido pelas regras da RFB**.**

Quanto às práticas: Cliente presente: Essa prática não foi tão utilizada em relação aos requisitos funcionais do software, o cliente foi mais presente na parte de sua utilização e interface. Jogo do planejamento: Não aconteceu planejamento no processo do desenvolvimento, tinha que seguir apenas o passo a passo do guia prático. Stand up meeting: Prática não utilizada e inviável pela empresa. Programação em par: Prática não utilizada pela empresa no desenvolvimento do SPED, projeto desenvolvido por um programador apenas. Código coletivo: Não utilizado. Código padronizado: Umas das práticas também não utilizadas pela empresa. Apesar de apenas um desenvolvedor ter participado do projeto, futuramente o desenvolvedor pode não fazer mais parte da empresa e isso dificultará futuras possíveis manutenções no software por parte de outro desenvolvedor. Design simples: Prática bem utilizada pela empresa, buscou sempre manter o design e interface do software com uma boa aparência e de fácil manuseio, tendo sempre o feedback dos usuários. Desenvolvimento orientado a testes: Prática não utilizada, o teste só foi feito após o software ter condições de gerar todos os blocos descritos no guia prático. Refatoração: Prática não utilizada durante o projeto. Existiu e existem algumas mudanças no software devido às atualizações do guia prático que trazem novos requisitos a serem implementados. Integração continua: Prática não utilizada. Releases curtos: Prática não utilizada, o projeto só foi entregue aos clientes após seu término. Metáfora: Os termos técnicos utilizados no guia prático foram “traduzidos” para um vocabulário mais fácil que o desenvolvedor e o cliente pudessem entender. Ritmo sustentável: Prática não utilizada pela empresa, o desenvolvedor trabalhou só onde em alguns momentos teve a necessidade de acelerar seu ritmo de desenvolvimento para que o projeto fosse entregue no prazo determinado pela RFB.

Ao fim de 60 dias o projeto teve sua primeira fase de desenvolvimento concluída, fazendo assim com que o software gere por completo todos os blocos solicitados pelo guia prático. Logo após, foi dado inicio a fase de testes que eram feitas através do PVA[8], que só

importava o arquivo se o mesmo tivesse todos os blocos em sua estrutura. O mesmo verifica a estrutura do layout e os campos que foram informados pelo software ao gerar o arquivo. O software ainda passou por alguns ajustes até sua versão final, que foi liberada em mais 1 mês de desenvolvimento e testes, levando um prazo total de 90 dias para todo o seu desenvolvimento.

4 CONSIDERAÇÕES FINAIS

Viu-se o quanto é importante avaliar a situação atual de um processo de desenvolvimento de software antes de iniciar seu desenvolvimento ou sua melhoria, pois no processo pressupõe mudança de uma situação inicial ou atual para uma situação desejada. Apesar de cada projeto ter suas características, é possível estabelecer elementos que devem estar presentes em toda e qualquer desenvolvimento.

Os resultados da análise evidenciaram que o processo de desenvolvimento do software na empresa possui alguns pontos a serem melhorados. Se a empresa tivesse uma equipe maior poderia com toda certeza aplicar mais algumas práticas do XP, e melhorar assim seu processo de desenvolvimento, onde ganharia inúmeros benefícios para empresa e para equipe, benefícios que aumentaria consideravelmente a produtividade, qualidade, padronização do código fonte, diminuiria os bugs e os custos no desenvolvimento do sistema.

Apesar de não seguir duramente qualquer tipo de metodologia de desenvolvimento, a empresa conseguiu desenvolver um software estável, com as funcionalidades requisitadas e conseguiu cumprir o prazo exigido pela RFB.

REFERÊNCIAS

BECK, Kent; **Extreme Programming explained**: embrance change. 1. ed. Reading, MA: Addison-Wesley, 2000;

BECK, Kent; FOWLER, Martin; **Planning Extreme Programming**. 1. ed. Boston: Addison-Wesley, 2001;

BORRIE, Helen; **Dominando Firebird**: Uma Referência Para Desenvolvedores de Bancos de Dados. Rio de Janeiro: Editora Ciência Moderna Ltda., 2006;

CANTÚ, Marco; **Dominando o Delphi** 4 – A Bíblia. São Paulo: MAKRON Books, 1998;

CARVALHO, Ariadne Maria Brito Rizzoni; **Introdução à engenharia de software** / Ariadne Maria Brito Rizzoni Carvalho, Thelma Cecília dos Santos Chiossi, Campinas, SP: Editora da Unicamp, 2001;

FABRI, Jose Augusto; **O product owner pode utilizar um mapa mental para especificar uma funcionalidade dentro do Scrum?**. Disponível em: <engenhariasoftware.wordpress.com/2012/07/20/o-product-owner-pode-utilizar-um-mapa-mental-para-especificar-uma-funcionalidade-dentro-do-scrum/> Acessado em: 19 de julho de 2015;

FERREIRA , Décio et al; **Um modelo ágil para gestão de projectos de software**. Artigo Cientifico. Universidade de Portugal, 2005;

JEFFRIES, Ronald E.; **What is extreme programming**. Disponível em: < http://xprogramming.com/book/whatisxp/> Acessado em: 20 de julho de 2015;
PRESSMAN, Roger S.; Engenharia de software. São Paulo: Pearson Makron Books, 1995;

RECEITA FEDERAL DO BRASIL; **Sistema Público de Escrituração Digital**. Disponível em: <www1.receita.fazenda.gov.br/Sped>. Acesso em: 20 de julho de 2015;

REZENDE, D. A.; **Engenharia de software e sistemas de informação**, 2 ed. Rio de Janeiro: Brasport, 2002;

RIBEIRO, Igor César de Souza; **Qualidade de software**. 2009, 49f. Trabalho de conclusão de curso (Especialização em engenharia de software e banco de dados) – Universidade Estadual de Londrina, Londrina;

ROQUE, Ruth Ferreira; **Estudo comparativo de metodologias de desenvolvimento de sistemas de informação utilizando a técnica Delphi**. Disponível em: <http://www.eps.ufsc.br/disserta98/ruth/> Acessado em: 27 de Novembro de 2012;

SATO, Danilo Toshiaki; **Uso eficaz de métricas em métodos ágeis de desenvolvimento de software**. 2007, 139f. Trabalho de conclusão de curso (Mestre em Ciências) – Universidade de São Paulo, São Paulo;

SATO, Danilo Toshiaki**; Gerenciamento de Equipes com Scrum – Curso de verão**, São Paulo: AgilCoop, 2008. Disponível em: <cssl.ime.usp.br/agilcoop/files/AgilCoop-Verao08-Scrum.ppt> Acessado em: 10 de julho de 2015;

SCHWABER, Ken; **Agile Project Management with Scrum**. Microsoft Press, 2004;

SCHWABER, Ken; **Guia do Scrum**: Scrum Aliance – Transforming The World Of Work, 2009;

SPENASSATO, Jan; **Uma análise do método ágil Scrum sob perspectiva do modelo CMMI nas áreas de processo de categoria Engenharia**. 2005, 100f. Trabalho de conclusão de curso (Graduação em computação) - Universidade de Passo Fundo;

TELES, Vinicius Manhães; **Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade**. São Paulo: Novatec, 2004;

TELES, Vinicius Manhães; **Um estudo de caso da adoção das práticas e valores do Extreme Programming**. 2005, 179f. Trabalho de conclusão de curso (Mestre em informática) – Universidade Federal do Rio de Janeiro, Rio de Janeiro;

YOURDON, Edward; **Declínio e queda dos analistas e dos programadores**. São Paulo: Makron Books, 1995;

ZANATTA, Alexandre Lazaretti; **xScrum**: uma proposta de extensão de um método ágil para gerência e desenvolvimento de requisitos visando adequação ao CMMI. 2004, 180f. Trabalho de conclusão de curso (Mestrado em ciências da computação) – Universidade Federal de Santa Catarina.