REVISTA COLETA
Científica

Ano 1, Vol. I, n.1, jan.-jul., 2017

(CC BY 4.0) | ISSN 2763-6496

DOI: https://doi.org/10.5281/zenodo.7416079

ARK: https://n2t.net/ark:/24285/RCC.v1i1.4

SGHC Honorários Contábeis

SGHC Accounting Fees

Recebido: 16/02/2017 | Aceito: 11/06/2017 | Publicado: 20/06/2017

Maydson Filipe Ramos
https://orcid.org/0000-0001-7798-2952
http://lattes.cnpq.br/83932303824345346
Faculdade CNEC, Unaí, MG, Brasil
E-mail: maydsonunai@hotmail.com

Resumo
Este artigo se propõe a tornar o processo de confecção dos honorários contábeis mais ágil e prático através de meio eletrônico, desde o lançamento até o fechamento. Com base nisso, tivemos como objetivo a elaboração um software que possibilite ao usuário gerar o honorário mensal de cada cliente com todos os serviços realizados para estes, bem como gerar relatórios periódicos com a média dos honorários contábeis pagos. Dentre as principais características do software desenvolvido, abordado neste artigo, pode-se citar a opção que é disponibilizada ao usuário de gerar relatórios gráficos com a média de honorários contábeis mensais que determinado cliente pagou em certo período de tempo. Por fim, conclui-se que o objetivo proposto nessa pesquisa foi alcançado, pois o software pode facilitar o trabalho do profissional da área contábil, uma vez que, deixará de somar os honorários manualmente, economizando tempo útil.

Palavras-chave: Honorários Contábeis. Software. Gráficos. SGHC

Abstract
This article aims to make the process of making accounting fees more agile and practical through electronic means, from launch to closing. Based on this, we aimed to create software that allows the user to generate the monthly fee for each client with all services performed for them, as well as generate periodic reports with the average accounting fees paid. Among the main characteristics of the developed software, covered in this article, it is possible to mention the option that is available to the user to generate graphical reports with the average monthly accounting fees that a client paid in a certain period of time. Finally, it is concluded that the objective proposed in this research was achieved, since the software can facilitate the work of the professional in the accounting area, since it will no longer add the fees manually, saving useful time.

Keywords: *Accounting fees. Software. Graphics. SGHC*

Introdução

Este artigo foi elaborado através de uma pesquisa de campo em uma empresa que presta serviços de contabilidade, cujo o principal objetivo era tornar o processo de confecção dos honorários contábeis mais ágil e prático através de meio eletrônico, desde o lançamento até o fechamento, bem como agilizar o processo de confecção do honorário; oferecer transparência ao cliente; gerar relatórios periódicos aos administradores; armazenar todos os honorários dos clientes.

Vislumbrando a necessidade de modernização e celeridade para o desenvolvimento da empresa e satisfação do cliente no que se refere ao controle dos honorários contábeis foi pensado em um sistema que pudesse preencher tais necessidades e, ao mesmo tempo, contribuísse com a tecnologia que a modernidade exige em qualquer ambiente de trabalho para melhor realização das propostas da empresa.

A empresa para qual o software será desenvolvido não conta, atualmente, com sistema para gerência de seus honorários e serviços por ela prestados. Tendo em vista que sua clientela cresce em grandes proporções, o controle destes honorários torna-se cada vez mais difícil, assim, o administrador não consegue mensurar a média de gastos de cada cliente e nem seu faturamento mensal ou anual. O objetivo do projeto é elaborar um software que possibilite ao usuário gerar o honorário mensal de cada cliente com todos os serviços realizados para estes, bem como gerar relatórios periódicos com a média dos honorários contábeis pagos.

O Presente referencial tem o objetivo de demonstrar as principais atividades, comuns aos processos de desenvolvimento de software, utilizados em organizações que buscam um padrão de qualidade no desenvolvimento de suas aplicações levando em conta a opinião e obra de alguns profissionais da área. Um processo de desenvolvimento de software é um conjunto de atividades e resultados associados que produz um produto de software (Somerville, 2007)

Formulação do problema

Dentre os problemas enfrentados na confecção dos honorários podemos citar:

- Não há detalhamento dos serviços prestados ao cliente durante o mês;
- Os honorários mensais gerados não são armazenados para consulta que, caso necessite ser feita, deverá ser manualmente;
- Perca de tempo útil na soma dos serviços prestados que ainda é feita manualmente;
- Necessidade de integração da empresa com novas tecnologias que podem fortalecer a qualidade dos serviços prestados, bem como sua credibilidade.

Objetivo geral

Tornar o processo de confecção dos honorários contábeis mais ágil e prático através de meio eletrônico, desde o lançamento até o fechamento.

Objetivos específicos

Os objetivos específicos do sistema são:

- Agilizar o processo de confecção do honorário.
- Oferecer transparência ao cliente.
- Gerar relatórios periódicos aos administradores.
- Armazenar todos os honorários dos clientes.

Conceitos de processo desenvolvimento de software

Atividades comuns de processos de desenvolvimento de software

Definição dos Requisitos/implementação

É a parte inicial do desenvolvimento onde se encontram as tarefas de investigação, é nesta fase de elaboração dos sistemas que tem se as primeiras reuniões com os clientes definindo suas necessidades para levantamento dos requisitos.

> As definições de requisitos de sistema especificam o que o sistema deve fazer (suas funções) e suas propriedades essenciais e desejáveis. Como na análise de requisitos de software, a criação das definições dos requisitos de sistema envolve consultas aos clientes e usuários finais do sistema. Uma importante parte da fase de definição de requisitos é estabelecer um conjunto de objetivos que o sistema deve atender. Esse conjunto não deve necessariamente ser expresso em termos de funcionalidade de sistema, mas deve definir por que o sistema está sendo adquirido para um determinado ambiente.O estágio de implementação de software é o processo de conversão de uma especificação de sistema em um sistema executável. Ele sempre envolve os processos de projeto e de programação de software, mas se uma abordagem evolucionária for usada, pode também envolver o refinamento da especificação do software (Somerville, 2007).

Teste de Software

Segundo Sommerville (2009, p. 7), o software deve ser validado para garantir que ele faça o que o cliente deseja.

Como a maioria das atividades de engenharia, a construção de software depende principalmente da habilidade, da interpretação e da execução, das pessoas que o constroem; por isso, erros acabam surgindo, mesmo com a utilização de métodos e ferramentas de engenharia de software.

> Para que tais erros não perdurem, existe uma série de atividades, coletivamente chamadas de “validação verificação e teste”, com a finalidade de garantir que tanto o modo pelo qual o software esta sendo construído quanto o produto em si estejam em conformidade com o especificado. (Delamaro, Maldonado & Jino, 2009).

Metodologias de desenvolvimento de software

Segundo Sommerville (2007, p. 43) Os processos de softwares são uma representação abstrata, de um processo de software. Cada modelo de processo representa um processo sob determinada perspectiva e, dessa forma fornece somente informações parciais sobre esse processo. Esses modelos gernéricos não são definições definitivas de processo de software. Ao contrário, são abstrações do processo que podem ser usadas para explicar diferentes abordagens para desenvolvimento de software.

Os modelos de processo são:

- *Modelo em cascata*: Considera as atividades fundamentais do processo compreendendo especificação, desenvolvimento, validação e evolução, que as representa como fases de processos separadas, tais como especificação de requisitos, processo de software, implementação e teste.
- *Desenvolvimento evolucionário*: intercala as atividades de especificação, desenvolvimento e validação. No sistema inicial é desenvolvido rapidamente baseado em especificações abstratas. Este sistema é então refinado com as

entradas do cliente para produzir um sistema que satisfaça as necessidades do cliente.

- *Engenharia de software baseada em componentes*: Esta abordagem se baseia na existência de um numero significativo de componentes reusáveis. O processo de desenvolvimento do sistema enfoca a integração destes componentes ao invés de desenvolve-los a partir do zero.

Modelo Iterativo e Incremental

Desenvolvimento Incremental é uma estratégia de planejamento estagiado em que várias partes do sistema são desenvolvidas em paralelo, e integradas quando completas. Não implica, requer ou pressupõe desenvolvimento iterativo ou em cascata – ambos são estrategias de retrabalho. A alternativa ao desenvolvimento incremental é desenvolver todo o sistema com uma integração única.

Desenvolvimento Iterativo é uma estratégia de planejamento de retrabalho em que o tempo de revisão e melhorias de partes do sistema é pré-definido. Isto não pressupõe desenvolvimento incremental, mas funciona muito bem com ele. Uma diferença típica é que a saída de um incremento não é necessariamente assunto de um refinamento futuro, e seu teste ou retorno do usuário não é utilizado como entrada para planos de revisão ou especificações para incrementos sucessivos. Ao contrario, a saída de uma iteração é examinada para modificação, e especialmente para revisão dos objetivos das iterações sucessivas.

A idéia básica por trás da abordagem iterativa é desenvolver um sistema de software incremental, permitindo ao desenvolvedor tirar vantagem daquilo que foi aprendido durante a fase inicial de desenvolvimento de uma versão do sistema. O aprendizado ocorre simultaneamente tanto para o desenvolvedor, quanto para o usuário do sistema.

[FONTE: http://protocoloti.blogspot.com.br/2012/03/os-modelos-de-desenvolvimento-de.html]

Engenharia de Requisitos

Classificação dos requisitos

O conjunto de requisitos é o que determina o escopo do projeto, é tudo que o software deverá implementar e será a referência para toda a equipe de desenvolvimento. Segundo Sommerville (2007 p. 83, 84, 85) os requisitos classificam-se em:

Requisitos Funcionais: São declarações de funções que o sistema deve fornecer, como o sistema deve reagir ä entradas específicas e como deve se comportar em determinadas situações, podem também declarar explicitamente o que o sistema não deve fazer.

Requisitos Não Funcionais: São restrições sobre os serviços ou funções oferecidas pelo sistema. Entre eles destacam-se restrições de tempo, sobre o processo de desenvolvimento ou padrões.

Requisitos de Domínio: São requisitos que se originam do domínio de aplicacao do sistema e refletem características desse domínio. Podem ser requisitos funcionais ou não funcionais

Técnicas de Elicitação

Técnicas de Levantamento de Requisitos

As técnicas de levantamento de requisitos têm por objetivo superar as dificuldades relativas a esta fase. Todas as técnicas possuem um conceito próprio e suas

respectivas vantagens e desvantagens, que podem ser utilizadas em conjunto pelo analista. Serão apresentadas de forma resumida nesse artigo algumas técnicas de levantamento de requisitos.

Levantamento orientado a pontos de vista

As abordagens orientadas a ponto de vista, na engenharia de requisitos, reconhecem diferentes pontos de vista e os utilizam para estruturar e organizar o processo de levantamento e os próprios requisitos. Uma importante capacidade da análise orientada a pontos de vista é que ela reconhece a existência de várias perspectivas e oferece um *framework* para descobrir conflitos nos requisitos propostos por diferentes *stakeholders.*

O método VORD (*viewpoint-oriented requirements definition* – definição de requisitos orientada a ponto de vista) foi projetado como um *framework* orientado a serviço para o levantamento e análise de requisitos.

Etnografia

A etnografia é uma técnica de observação que pode ser utilizada para compreender os requisitos sociais e organizacionais, ou seja, entender a política organizacional bem como a cultura de trabalho com objetivo de familiarizar-se com o sistema e sua história. Os cientistas sociais e antropólogos usam técnicas de observação para desenvolver um entendimento completo e detalhado de culturas particulares.

Workshops

Trata-se de uma técnica de elicitação em grupo usada em uma reunião estruturada. Devem fazer parte do grupo uma equipe de analistas e uma seleção dos *stakeholders* que melhor representam a organização e o contexto em que o sistema será usado, obtendo assim um conjunto de requisitos bem definidos.

Prototipagem

Protótipo tem por objetivo explorar aspectos críticos dos requisitos de um produto, implementando de forma rápida um pequeno subconjunto de funcionalidades deste produto. O protótipo é indicado para estudar as alternativas de interface do usuário; problemas de comunicação com outros produtos; e a viabilidade de atendimento dos requisitos de desempenho. As técnicas utilizadas na elaboração do protótipo são várias: interface de usuário, relatórios textuais, relatórios gráficos, entre outras.

Entrevistas

A entrevista é uma das técnicas tradicionais mais simples de utilizar e que produz bons resultados na fase inicial de obtenção de dados. Convém que o entrevistador dê margem ao entrevistado para expor as suas idéias. É necessário ter um plano de entrevista para que não haja dispersão do assunto principal e a entrevista fique longa, deixando o entrevistado cansado e não produzindo bons resultados.

Questionários

O uso de questionário é indicado, por exemplo, quando há diversos grupos de usuários que podem estar em diversos locais diferentes do país. Neste caso, elaboram-se pesquisas específicas de acompanhamento com usuários selecionados, que a contribuição em potencial pareça mais importante, pois não seria prático entrevistar todas as pessoas em todos os locais.

Brainstorming
Brainstorming é uma técnica para geração de idéias. Ela consiste em uma ou várias reuniões que permitem que as pessoas sugiram e explorem idéias.

JAD
JAD (*Joint Application Design*) é uma técnica para promover cooperação, entendimento e trabalho em grupo entre os usuários desenvolvedores. O JAD facilita a criação de uma visão compartilhada do que o produto de software deve ser. Através da sua utilização os desenvolvedores ajudam os usuários a formular problemas e explorar soluções. Dessa forma, os usuários ganham um sentimento de envolvimento, posse e responsabilidade com o sucesso do produto.

Gerenciamento de Requisitos

Segundo Filho (2003, p.5) um problema comum no desenvolvimento de software é a **instabilidade dos requisitos,** que ocorre quando clientes e usuários trazem novos requisitos, ou alterações de requisitos, quando, o desenvolvimento já está em fase adiantada. A instabilidade costuma ter alto custo que significa perder trabalho já feito e remendar outras. A boa engenharia de requisitos reduz a instabilidade destes ajudando ä obter os requisitos corretos em um estágio anterior ao desenvolvimento. Entretanto alterações dos requisitos são as vezes inevitáveis. A engenharia de requisitos é sujeita à limitações humanas. A gestão dos requisitos é a disciplina de engenharia de software que procura manter sob controle o conjunto de requisitos de um produto mesmo diante de alterações inevitáveis.

Análise e Projeto de Software

Análise e Projeto Estruturado

Segundo Jones (1980, p. 3) Projeto estruturado é uma abordagem disciplinada de projeto de sistema computadorizados, uma atividade que no passado foi notoriamente acidentada e cheia de problemas. Tem como características, sistemas fáceis de entender, confiáveis, passíveis de “longa vida” , facilmente desenvolvidos, eficientes e que FUNCIONAM.

Análise e Projeto Orientados a Objetos

*Yourdon, Argila (1999, p. 7) definiram que u*m modelo de AOO retrata objetos que representam um domínio de aplicação específico, juntamente com diversos relacionamentos estruturais e de comunicação. O modelo de AOO serve para dois propósitos. Primeiro, para formalizar a “visão” do mundo real dentro do qual o sistema de software será construído. Ele estabelece os objetos que servirão como principais estruturas organizadas do sistema de software, bem como as principais estruturas organizacionais que o mundo real impõe em qualquer sistema de software construído naquele domínio de aplicação.

Projeto de Software Orientado a aspecto

A programação orientada a aspectos propõe utilizar aspectos [Kiczales ET AL 1997] para a separação entre o código que implementam os propósitos não funcionais de uma aplicação, chamados de preocupações ortogonais ou cross-sutting, do código que implementa a funcionalidade da aplicação. Aspectos são módulos que implementam código referente a essas preocupações.

Mecanismos de Armazenamento de Dados

Banco de Dados Relacional

Segundo Nassur, Setzer (1999 p. 5 e 35) no modelo relacional, os dados são armazenados em telas bidimensionais, denominadas relações, cada linha da tabela representa um elemento do conjunto de dados e cada coluna da tabela contem valore s de um conjunto definido, denominado domínio. Uma linha na relação é chamada tupla.

Banco de Dados Orientado a Objetos

Um banco de dados orientado a objeto é basicamente um sistema em que a unidade de armazenamento é o objeto, com o mesmo conceito das linguagens de programação orientadas a objetos. A diferença fundamental é a persistência dos objetos, ou seja, os objetos continuam a existir mesmo após o encerramento do programa. Podemos classificar sistema que oferecem persistências a objetos em quatro categorias de acordo com sua arquitetura que são eles, Sistema de banco de dados estendidos, Linguagem de Programação de banco de dados, gerenciadores de objetos, e Geradores de sistemas de banco de dados.

Teste Software

Atividades do Processo Teste

De acordo com Rios, Moreira (2006, p. 37) as atividades de teste são as seguintes:

- ✓ Elaborar a análise de riscos do projeto de teste.
- ✓ Analisar as possibilidades de minimizar os riscos do negócio através dos testes.
- ✓ Avaliar requisitos do sistema e o planejamento do sistema.
- ✓ Avaliar o modelo de dados preliminares e outros modelos e diagramas do sistema.
- ✓ Identificar indicadores de qualidades e desempenho
- ✓ Estabelecer que produtos serão verificados.
- ✓ Elaborar o documento Estratégia de Testes do Projeto.
- ✓ Estabelecer os tipos de testes a serem realizados no desenvolvimento e implantação.
- ✓ Definir as prioridades de testes dos móduloos ou subsistemas, visando os teste de integração.
- ✓ Identificar as necessidades de testes alfa e beta.
- ✓ Definir prioridades para os testes
- ✓ Definir Infra-estrutura (hardware/software/ferramentas/pessoal) a ser utilizada.
- ✓ Definir os ambientes operacionais para a realização dos testes.
- ✓ Identificar a necessidade de testes estáticos
- ✓ Identificar subdivisões (subsistemas, módulos) requeridas para o sistema, visando a melhor gestão do processo de testes através de sua modularização.
- ✓ Validar com áreas desenvolvidas as estratégias de teste.
- ✓ Fazer a revisão técnica dos produtos desta etapa.
- ✓ Divulgar a estratégia de testes para a equipe de testes.
- ✓ Identificar as técnicas a serem adotadas para os testes.

Segundo Fernando Rodrigues os testes devem ser realizados em vários níveis:

- ✓ **Teste de unidade ou testes unitários.**

- ✓ Esse nível de teste tem por objetivo testar a menor unidade do software, tentando provocar falhas de regra de negocio. Esse teste é feito por pequenos trechos de código isoladamente.

- ✓ **Teste de integração**
- ✓ Verifica se há falhas na entre os módulos ou interfaces, quando esses são integrados ao tentar fazer o funcionamento de um todo.

- ✓ **Teste de sistema**

Avalia o sistema como um todo, como se fosse um usuário final, inserindo dados reais e analisando se suas respostas atendem aos requisitos.

- ✓ **Teste de aceitação**

- ✓ No inicio do desenvolvimento são declarados quais serão as funcionalidades que serão testadas para que o software seja declarado como entregue, esse nível de testes é efetuado geralmente por um grupo de usuários que verificam algumas funcionalidades para que o software seja considerado como aceito

Teste caixa branca

O teste de caixa branca consiste em testar linha a linha de código, os fluxos, as condições, os loops, pra quem está familiarizado com o Visual Studio é o Teste unitário com a opção Code Coverage habilitada.
Este tipo de teste tem o objetivo alcançar o Maximo de cobertura possível

Teste caixa preta

O teste de caixa preta consiste em testar a funcionalidade como um todo, não importando o seu fluxo e sim o resultado esperado.

Este teste analisa o tratamento de erros (exceções) as validações, funções, para quem está familiarizado com Visual Studio, os testes unitários sem Code Coverage são considerados como teste de caixa preta.

Paradigmas de Desenvolvimento

Programação Estruturada x Programação OO

Flávia Neves explica da seguinte forma:
um sistema orientado a objetos é dividido em componentes e não mais em processos. Imagine um sistema financeiro, onde faríamos toda a administração de uma empresa. Teríamos as seguintes diferenças:

- **POO**: Teríamos um objeto de fornecedores, por exemplo, onde todas as funções estariam agrupadas no objeto e em nenhum outro lugar.

- **Estruturada**: As rotinas e funções de fornecedores estariam espalhadas em todo o sistema, como em contas a pagar, contas a receber, cadastro, etc.

Imagine agora o cadastro de fornecedores, com todas as suas rotinas e funções:

- **Estruturada**: Se você futuramente precisar alterar algum dado, função ou propriedade, o que em seu programa será afetado? O que terá que ser reestruturado? Imagine a você voltando a fase de testes e analisando todo o seu sistema até ter certeza que a alteração que você fez não desencadeou um finita listas de alterações que você terá que fazer em todo o sistema.

- **POO**: as propriedades, funções e rotinas do objeto fornecedores estão todas em um único objeto, encapsulados, facilitando essa necessidade futura de alterações e atualizações.

Programação Orientada a aspectos

O propósito da Programação Orientada a Aspectos (POA), que surgiu como um complemento para a Programação Orientada a Objetos (POO), é tornar mais fácil a modularização dos *crosscuttingconcerns* (características transversais), os quais, em um programa de computador, são aspectos que afetam outros. Muitas vezes, esses aspectos não podem ser claramente separados do resto do sistema, tanto no design quanto na implementação, o que resulta em dispersão ou embaraçamento do programa.

A tecnologia de orientação a aspectos tem muitos benefícios potenciais. Ela fornece um método de especificar e encapsular características transversais num dado sistema. Este poder, por sua vez, nos permite realizar um melhor trabalho de manutenção de sistemas ao passo que eles evoluem (e a tendência é que eles evoluam). A POA, nos permitiria adicionar de maneira organizada novas funcionalidades, na forma de características, a um sistema. Assim, as melhorias na estrutura, nos permitiriam manter sistemas funcionando por muito mais tempo e incrementá-los sem ter que realizar uma re-escrita completa do código.

Frameworks e Arquiteturas para Desenvolvimento WEB

Model-view-controller (MVC)

O Engenheiro Civil Christopher Alexander criou o que se considera o primeiro padrão de projeto em meados da década de 70. É considerado um padrão de projeto uma solução já testada e documentada que possa resolver um problema específico em projetos distintos. Através do trabalho de Alexander, profissionais da área de desenvolvimento de software utilizaram tais conceitos para iniciar as primeiras documentações de padrões de projetos, tornando-as acessíveis para toda a área de desenvolvimento.

O então funcionário da corporação Xerox PARC, Trygve Reenskaug, iniciou em 1979 o que viria a ser o nascimento do padrão de projeto MVC. A implementação original foi descrita no artigo “Applications Programming in Smalltalk-80: How to use Model-View-Controller”. A ideia de Reenskaug gerou um padrão de arquitetura de aplicação cujo objetivo é separar o projeto em três camadas independentes, que são o modelo, a visão e o controlador. Essa separação de camadas ajuda na redução de acoplamento e promove o aumento de coesão nas classes do projeto. Assim, quando o modelo MVC é utilizado, pode facilitar em muito a manutenção do código e sua reutilização em outros projetos.

Cliente/Servidor

Para Sommerville, (2007, p.166) o modelo de arquitetura cliente-servidor é um modelo em que o sistema é organizado com um conjunto de serviços e servidores e clientes associados que acessam e usam os serviços. Os principais componentes desse modelo são:

> 1. Um conjunto de servidores que oferecem serviços para outros subsistemas. Exemplos disso são servidores de impressoras que oferecem serviços de impressão, servidores de arquivos que oferecem serviços de gerenciamento de arquivos e um servidor de compiladores que oferece serviços de compilação de linguagens de programação.
> 2. Um conjunto de clientes que solicita os serviços oferecidos pelos servidores. Esses normalmente são subsistemas independentes. Pode haver várias instâncias de um programa cliente sendo executadas simultaneamente.
> 3. Uma rede que permite aos clientes acessarem esses serviços. Isso não é estritamente necessário quando ambos, clientes e servidores, podem ser executados em uma única máquina. Na prática, contudo, a maioria dos sistemas clientes-servidor é implementada como sistemas distribuídos.

Segundo Sommerville (2007 p. 16) a vantagem mais importante de um modelo cliente-servidor é que ele é uma arquitetura distribuída. O uso efetivo de sistemas em rede pode ser feito com muitos processadores distribuídos. É fácil adicionar um novo servidor e integrá-lo ao restante do sistema ou atualizar servidores de maneira transparente sem afetar outras partes do sistema.

Ciclo de Vida do Modelo de Processo de Software Utilizado

Ciclo de Vida Scrum

A idéia principal da SCRUM é que o desenvolvimento de softwares envolve muitas variáveis técnicas e de ambiente, como requisitos, recursos e tecnologia, que podem mudar durante o processo. Isto torna o processo de desenvolvimento imprevisível e complexo, requerendo flexibilidade para acompanhar as mudanças. O resultado do processo deve ser um software que é realmente útil para o cliente.

O ciclo de vida da SCRUM é baseado em três fases principais, divididas em sub-fases:

1) Pré-planejamento (Pre-game phase): os requisitos são descritos em um documento chamado backlog. Posteriormente eles são priorizados e são feitas estimativas de esforço para o desenvolvimento de cada requisito. O planejamento inclui também, entre outras atividades, a definição da equipe de desenvolvimento, as ferramentas a serem usadas, os possíveis riscos do projeto e as necessidades de treinamento. Finalmente é proposta uma arquitetura de desenvolvimento. Eventuais alterações nos requisitos descritos no backlog são identificadas, assim como seus possíveis riscos.

2)Desenvolvimento (game phase): as muitas variáveis técnicas e do ambiente identificadas previamente são observadas e controladas durante o desenvolvimento. Ao invés de considerar essas variáveis apenas no início do projeto, como no caso das metodologias tradicionais, na SCRUM o controle é feito continuamente, o que aumenta a flexibilidade para acompanhar as mudanças. Nesta fase o software é

desenvolvido em ciclos (sprints) em que novas funcionalidades são adicionadas. Cada um desses ciclos é desenvolvido de forma tradicional, ou seja, primeiramente faz-se a análise, em seguida o projeto, implementação e testes. Cada um desses ciclos é planejado para durar de uma semana a um mês.

3) Pós-planejamento (post-game phase): após a fase de desenvolvimento são feitas reuniões para analisar o progresso do projeto e demonstrar o software atual para os clientes. Nesta fase são feitas as etapas de integração, testes finais e documentação.

Métodos e Técnicas de Levantamento de Requisitos Utilizados

Etnografia

A etnografia é uma técnica de observação que pode ser utilizada para compreender os requisitos sociais e organizacionais, ou seja, entender a política organizacional bem como a cultura de trabalho com objetivo de familiarizar-se com o sistema e sua história. Os cientistas sociais e antropólogos usam técnicas de observação para desenvolver um entendimento completo e detalhado de culturas particulares.

Notações Utilizadas para Análise e Projeto de Software

Diagramas de Caso de Uso

Para Leandro Ribeiro, esse diagrama documenta *o que o sistema faz do ponto de vista do usuário*. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. Nesse diagrama não nos aprofundamos em detalhes técnicos que dizem *como o sistema faz*.

Este artefato é comumente derivado da especificação de requisitos, que por sua vez não faz parte da UML. Pode ser utilizado também para criar o documento de requisitos.

Diagramas de Seqüência

De acordo com Geraldo Magela, diagramas de Sequência são muito úteis em fornecer suporte real à implementação e em constituir rica documentação de alto nível. Eles representam os objetos participantes de uma colaboração enquanto emitem e recebem mensagens no intuito de realizar um caso de uso. As mensagens são apresentadas em sua ordem temporal, o que facilita a compreensão do fluxo de controle do caso de uso. A maior dificuldade associada à sua criação parece estar relacionada ao grau de detalhamento a ser aplicado a esses diagramas. Mas pode-se adotar uma perspectiva prática para criação de uma documentação realmente útil, e que não se torne uma tarefa ainda mais árdua do que a própria codificação do sistema.

Diagramas de Classe

Diagrama de classes a representação da estrutura e relações das classes que servem de modelo para objetos.

uma modelagem muito útil para o desenvolvimento de sistemas, pois define todas as classes que o sistema necessita possuir e é a base para a construção dos diagramas de comunicação, seqüência e estados.

Mecanismos Para Armazenamento de Dados Utilizados, Técnicas e Níveis de Testes Aplicados

Segundo Sommerville (2009, p. 7), o software deve ser validado para garantir que ele faça o que o cliente deseja.

Como a maioria das atividades de engenharia, a construção de software depende principalmente da habilidade, da interpretação e da execução, das pessoas que o constroem; por isso, erros acabam surgindo, mesmo com a utilização de métodos e ferramentas de engenharia de software.

> Para que tais erros não perdurem, existe uma série de atividades, coletivamente chamadas de "validação verificação e teste", com a finalidade de garantir que tanto o modo pelo qual o software esta sendo construído quanto o produto em si estejam em conformidade com o especificado. (Delamaro, Maldonado & Jino, 2009).

Teste de Sistema: avalia o software em busca de falhas por meio da utilização do mesmo, como se fosse um usuário final. Dessa maneira, os testes são executados nos mesmos ambientes, com as mesmas condições e com os mesmos dados de entrada que um usuário utilizaria no seu dia-a-dia de manipulação do software. Verifica se o produto satisfaz seus requisitos.

Teste de Unidade: também conhecido como testes unitários. Tem por objetivo explorar a menor unidade do projeto, procurando provocar falhas ocasionadas por defeitos de lógica e de implementação em cada módulo, separadamente. O universo alvo desse tipo de teste são os métodos dos objetos ou mesmo pequenos trechos de código.

Paradigmas de programação Utilizados

Programação Orientada a Objetos

A POO foi criada para tentar aproximar o mundo real do mundo virtual: a idéia fundamental é tentar simular o mundo real dentro do computador. Para isso, nada mais natural do que utilizar Objetos.

Na POO o programador é responsável por moldar o mundo dos objetos, e explicar para estes objetos como eles devem interagir entre si. Os objetos "conversam" uns com os outros através do envio de mensagens, e o papel principal do programador é especificar quais serão as mensagens que cada objeto pode receber, e também qual a ação que aquele objeto deve realizar ao receber aquela mensagem em específico.

Frameworks e Arquiteturas Para Desenvolvimento WEB Utilizados

Model-view-controller (MVC)

O Engenheiro Civil Christopher Alexander criou o que se considera o primeiro padrão de projeto em meados da década de 70. É considerado um padrão de projeto uma solução já testada e documentada que possa resolver um problema específico em projetos distintos. Através do trabalho de Alexander, profissionais da área de desenvolvimento de software utilizaram tais conceitos para iniciar as primeiras documentações de padrões de projetos, tornando-as acessíveis para toda a área de desenvolvimento.

O então funcionário da corporação Xerox PARC, Trygve Reenskaug, iniciou em 1979 o que viria a ser o nascimento do padrão de projeto MVC. A implementação original foi descrita no artigo “Applications Programming in Smalltalk-80: How to use Model-View-Controller”. A idéia de Reenskaug gerou um padrão de arquitetura de aplicação cujo objetivo é separar o projeto em três camadas independentes, que são o modelo, a visão e o controlador. Essa separação de camadas ajuda na redução de acoplamento e promove o aumento de coesão nas classes do projeto. Assim, quando o modelo MVC é utilizado, pode facilitar em muito a manutenção do código e sua reutilização em outros projetos (Cui, Na & Zhang, 2021).

Struts

Uma maneira de separar questões em uma aplicação de software é usar um Model-View-Controller arquitetura (MVC). O *modelo* representa o código de empresa ou banco de dados, a *Visão* representa o código de design da página, e o *controlador* representa o código de navegação. O framework Struts é projetado para ajudar os desenvolvedores a criar aplicações web que utilizam a arquitetura MVC.

Hibernate

O Hibernate é um framework ORM - ObjectRelationalMapping. É uma ferramenta que nos ajuda a persistir objetos Java em um banco de dados relacional. O trabalho do desenvolvedor é definir como os objetos são mapeados nas tabelas do banco e o Hibernate faz todo o acesso ao banco, gerando inclusive os comandos SQL necessário.

Factory

é um padrão de projeto que visa encapsular a criação de um objeto em um método. Factory é provavelmente um dos padrões mais utilizados porque ele é muito natural. Ele é usado, muitas vezes, sem consciência de que está sendo usado um padrão.

A implementação utilizando esse padrão de projeto afeta diretamente na qualidade do projeto facilitando alguma alteração futura no sistema. A empresa responsável pela área de negócio é a VLM Contabilidade, onde um funcionário será responsável pelas informações especificas relacionadas às áreas em que o sistema vai atuar.

O propósito deste documento é coletar, analisar e definir as necessidades e funcionalidades do projeto de software que será desenvolvido para a VLM Contabilidade, focando nas necessidades dos envolvidos que serão beneficiados pelo software.

O objetivo deste documento é mostrar de maneira o que será desenvolvido e quem são os envolvidos no desenvolvimento.

Seu escopo engloba a manipulação dos dados fornecidos pela empresa acompanhada pelo responsável do departamento financeiro da mesma.

Cenário

Atualmente a Empresa não utiliza nenhum sistema específico para realizar o detalhamento dos honorários prestados, porém utiliza-se um sistema on-line para gerar o valor total do serviço.

No mercado há sistemas similares que atendem a escritórios de contabilidade e contadores autônomos no geral, como o descrito abaixo:

Fortes Honorários Contábeis

O Fortes Honorários contábeis que foi desenvolvido pelo Grupo Fortes Informática Ltda. **(**http://www.honorarioscontabeis.com.br/index.php?/conteudo/id/2) oferece ao usuário serviços de cálculo e controle de custos dos serviços prestados por empresas de contabilidade, assessoria, auditoria, perícia e similares.

Oportunidade de Negócio

O software possibilitará a empresa melhor controle das informações de usuários, clientes e especificação de serviços honorários, possibilitando a extração de informações importantes por meio de relatórios, além de agregar controle adequado ao acesso as informações.

Descrição do Projeto

O escopo do projeto é o controle por meio digital de serviços, honorários contábeis prestados aos clientes, sistema de gerenciamento de usuários, emissão de relatórios detalhados dos serviços prestados e/ou acesso ao sistema

Conclusão

A partir da realização deste artigo, pode-se concluir que o projeto de software desenvolvido, apesar das dificuldades encontradas ao longo do seu desenvolvimento, atendeu ao seu propósito inicial, que é o gerenciamento de honorários contábeis.

O software se mostrou bastante eficiente no que se refere ao controle de honorário contábeis. O usuário tem uma interface de fácil compreensão onde pode cadastrar os honorários, efetuar os lançamentos ao longo do mês e gerar o honorário mensalmente, de forma sistema, por si só, busca e calcula todos os lançamentos de serviços automaticamente, simplificando o processo de cálculo de honorários contábeis que era feito manualmente.

Dentre as principais características do software desenvolvido, pode-se citar a opção que é disponibilizada ao usuário de gerar relatórios gráficos com a média de honorários contábeis mensais que determinado cliente pagou em certo período de tempo.

Por fim, conclui-se que o objetivo proposto foi alcançado, pois o software pode facilitar o trabalho do profissional da área contábil, uma vez que, deixará de somar os honorários manualmente, economizando tempo útil.

Referências

Delamaro, M. E., Maldonado, J. C., & Jino, M. (Orgs.). (2007). *Introdução ao teste de software*. Elsevier.

Dias, B. F., Barboza, L. G. S., Bertolini, G. R. F., & Vesco, D. G. D. (2016). Intervenção Aplicada a um Salão de Beleza para Adequação dos Aspectos

Financeiros. *International Journal of Professional Business Review*, *1*(2), 76–90. https://doi.org/10.26668/businessreview/2016.v1i2.21

Higor, M. (2013). *Padrão de Projeto Factory Method em Java*, disponível em <http://www.devmedia.com.br/padrao-de-projeto-factory-method-em-java/26348> Acessado em 23/04/2013.

Moraes, R. (2013). Artigo Engenharia de Software 2 - *Técnicas para levantamento de Requisitos*. Disponível em <http://www.devmedia.com.br/artigo-engenharia-de-software-2-tecnicas-para-levantamento-de-requisitos/9151> Acessado em 20/04/2013.

Neto, A. C. D. (2013). Artigo Engenharia de Software – *Introdução a Teste de Software* – disponível em <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035> Acessado em 02/05/2013.

Portela, Cristiano R.R. (2013). *Análise de requisitos Conceitos* – disponível em < http://www.paiossin.com/wordpress/wp-content/uploads/2011/11/Anlise-de-Requisitos-Conceitos.pdf> Acessado em 28/04/2013.

Prado, Alexandre A. (2013). *Fundamentos do Java Struts*, disponível em < http://www.devmedia.com.br/fundamentos-do-java-struts/7238> Acessado em 24/04/2013.

Protocoloti, (2013). *Os Modelos de Desenvolvimento de Software*. Disponível em <http://protocoloti.blogspot.com.br/2012/03/os-modelos-de-desenvolvimento-de.html> Acessado em 21/04/2013.

Sommerville, I. (2007). *Ian Engenharia de Software trad*. Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edílson de Andrade Barbosa -8. Ed. - São Paulo: Pearson Addison Wesley, 2007.

Sommerville, I. (2003). *Engenharia de software/ Ian Sommerville*; tradução André Mauricio de Andrade Ribeiro; revisão técnica Kechi Hirama.São Paulo: Addison Wesley, 2003.