UNIVERSIDADE SAGRADO CORAÇÃO

PAULO ROBERTO JUSTULIN JUNIOR

SOFTWARE PARA ESTIMATIVAS DE PROJETOS BASEADO EM ANÁLISE DE PONTOS DE FUNÇÃO

BAURU
2013

PAULO ROBERTO JUSTULIN JUNIOR

SOFTWARE PARA ESTIMATIVAS DE PROJETOS BASEADO EM ANÁLISE DE PONTOS DE FUNÇÃO

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Sociais Aplicadas como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, sob a orientação do Prof. Dr. Elvio Gilberto da Silva.

BAURU
2013

Justulin Junior, Paulo Roberto

J968s

Software para estimativas de projetos baseado em análise de pontos de função / Paulo Roberto Justulin Junior -- 2013.

82f. : il.

Orientador: Prof. Dr. Elvio Gilberto da Silva.

Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade do Sagrado Coração – Bauru – SP.

1. Pontos de função. 2. Estimativas. 3. Métricas. 4. Java. I. Silva, Elvio Gilberto da. II. Título.

PAULO ROBERTO JUSTULIN JUNIOR

SOFTWARE PARA ESTIMATIVAS DE PROJETOS BASEADO EM ANÁLISE DE PONTOS DE FUNÇÃO

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Sociais Aplicadas como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, sob a orientação do Prof. Dr. Elvio Gilberto da Silva.

Banca examinadora:

_____

Prof. Dr. Elvio Gilberto da Silva
Universidade Sagrado Coração

_____

Prof. Ms. Patrick Pedreira da Silva
Universidade Sagrado Coração

_____

Prof. Esp. Henrique Pachioni Martins
Universidade Sagrado Coração

Bauru, 03 de dezembro de 2013.

Dedico este trabalho aos meus pais que,
para mim, são exemplos de vida e de caráter.

AGRADECIMENTOS

Sobretudo a Deus, por estar do meu lado sempre e principalmente nos momentos difíceis.

Agradeço à minha família por acreditarem em mim e por todo o apoio que me deram desde o início.

Agradeço imensamente ao meu orientador, Prof. Dr. Elvio Gilberto da Silva, pela paciência e motivação incondicional que me proporcionou para a realização deste trabalho.

E a todos os demais Mestres e Doutores que me ajudaram a absorver um pouco de seus ensinamentos.

“Você não consegue controlar aquilo que não consegue medir”.
(Tom de Marco)

RESUMO

O projeto consiste na criação de um protótipo de um *software* desenvolvido na linguagem JAVA, que calcule o tamanho de um projeto em Pontos de Função, para que deste resultado, apresente as estimativas necessárias que o engenheiro de *software* precisa: estimativa de tamanho, esforço, prazo e custo. O protótipo apresenta uma *interface* simples e clara, com linguagens naturais, para que qualquer tipo de usuário possa utilizar, poupando-se de projeções de cálculos manuais. Também armazena todo um histórico de estimativas de projetos passados, para que a partir destes, tenha-se um indicador para futuros *softwares*, tornando-se a estimativa cada vez mais exata. Este trabalho também apresenta a importância de estimar durante a fase de inicial do projeto, citando diversas métricas, com ênfase em Análise de Pontos de Função da IFPUG.

Palavras-chave: Pontos de Função. Estimativas. Métricas. JAVA.

ABSTRACT

The project consists in creating a prototype software developed in JAVA language, which calculates the size of a project in Function Points, so this result, submit the necessary estimates that the software engineer needs: estimating size, effort, time and cost. The prototype has a simple and clear interface with natural language, so that any user can use, saving projections of manual calculations. Also stores a historical estimates of past projects, so that from these, has become an indicator for future software becoming increasingly accurate estimate. This work also shows the importance of estimating during the initial phase of the project, citing various metrics, with emphasis on analysis of the IFPUG Function Points.

Keywords: Function Points. Estimates. Metrics. JAVA.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AIE:	Arquivo de Interface Externa.
ALI:	Arquivo Lógico Interno.
APF:	Análise de Pontos de Função.
AR:	Arquivo Referenciado.
BFPUG:	*Brazilian Function Point Users Group.*
CE:	Consulta Externa.
CPM:	*Counting Practices Manual.*
EE:	Entrada Externa.
EI:	*External Input.*
EIF:	*External Interface File.*
EO:	E*xternal Output.*
EQ:	*External Inquiry.*
FAV:	Fatores de Ajuste de Valor.
FP:	*Function Point.*
ID:	Item de Dado.
IFPUG:	*International Function Point Users Group.*
ILF:	*Internal Logical Files.*
KLOC:	*Kilo Lines of Code.*
LOC:	*Lines of Code.*
RL:	Registro Lógico.
SE:	Saída Externa.
SGBD:	Sistema de Gerenciamento de Banco de Dados.
SQL:	*Structured Query Language.*
TD:	Tipo de Dado.
TR:	Tipo de Registro.
UML:	*Unified Modeling Language.*
VAF:	*Value Adjustment Factors.*

SUMÁRIO

1 INTRODUÇÃO

Quando um *software* é bem sucedido, ou seja, atende todas as necessidades dos usuários funcionando perfeitamente durante um longo período, de fácil utilização e manutenção, ele é realmente capaz de mudar as coisas para melhor. Porém quando o *software* falha e os usuários estão insatisfeitos por ser difícil de utiliza-lo e mais ainda modifica-lo, problemas certamente irão acontecer. Todos querem construir *softwares* que facilitem o trabalho, evitando tais pontos negativos que sempre assombram os usuários. Para ter êxito em tal fato, precisamos de disciplina no projeto e na construção de um *software* (PRESSMAN, 2011).

A garantia da qualidade é umas das principais preocupações da indústria de desenvolvimento de *software*, pois a maioria das empresas hoje utiliza esse tipo de aplicação para gerir seus códigos, produtos e relacionamento com clientes. Há diversas medidas de garantia de qualidade para o sucesso de uma aplicação, dentre elas, umas das mais simples e menos custosa, é a medição de *software*. Através de dados quantitativos, ela auxilia a tomada de decisão e é capaz de informar que aspectos de produto atendem ao padrão de qualidade especificado, além de permitir o entendimento e aperfeiçoamento do processo de produção. Sem contar que irá tornar o gerenciamento de projetos baseados em fatos e não em “achismo” (ABREU, c2013).

Parte dos problemas de *software* decorre do mau desenvolvimento, onde realmente não se tem a métrica, o planejamento e as estimativas de prazos e custos. Com isso, acabará resultando num *software* de pouca qualidade, difícil manutenção e utilidade para o usuário. Sem contar que não terá o controle algum no andamento do *software* na fase de desenvolvimento.

Sendo assim, para se medir um *software* com qualidade, são utilizadas diversas métricas que são como tipos de medições aplicadas a um sistema, documentação ou processo relacionado. Através dessas métricas é possível determinar esforço, tempo, custo e tamanho do produto, por exemplo. Para obter resultados realmente significativos, as métricas devem ser aplicadas em um ciclo constante, envolvendo planejamento, medição, análise de resultados, tomada de decisão e implementação das

decisões. Devem, também, ser aplicadas durante as fases de desenvolvimento do *software*, garantindo maior impacto positivo no final.

Porém, muitas empresas, como pré-requisito, precisam determinar o tamanho do *software* antes do desenvolvimento. Mesmo tendo várias métricas orientadas ao tamanho, como descreve nos próximos capítulos, não são consideradas satisfatórias e não tão precisas. Segundo Jones (1986 citado por PRESSMAN, 1995), as métricas orientadas ao tamanho provocam controvérsias e não são universalmente aceitas como a melhor maneira de se medir o processo de desenvolvimento de *software*.

Métricas orientadas ao tamanho, conforme citado pelo autor, são métricas diretas, ou seja, com atributos observáveis.

Há ainda o conceito de medidas indiretas, que são obtidas através de outras métricas, tais como complexidade, confiabilidade e facilidade de manutenção. Estas podem ser aplicadas em produtos ou em processo.

Medidas orientadas à função faz parte das medidas indiretas, nas quais invés de contar as linhas de código concentram-se nas funcionalidades do *software*, ou na avaliação conhecida como “pontos por função”, provendo uma métrica de medição para apoiar a análise de produtividade e qualidade do *software* a fim de fazer as estimativas para o projeto (PRESSMAN, 2011).

Cinco características principais são utilizadas como parâmetros de medida em métricas orientadas à função: número de entradas externas, número de saídas externas, número de consultas externas, número de arquivos lógicos internos, números de arquivos de *interface* externos (PRESSMAN, 2011).

Sua identificação e classificação exige um especialista, que neste caso, o especialista será o protótipo desenvolvido neste trabalho, mostrando a importância que a métrica de *software* oferece no sentido de estimar durante a fase de projeto.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Desenvolver um protótipo de um *software* para métricas e estimativas em projeto de sistema, baseado em Análise de Pontos de Função, visando facilidade e confiabilidade para que o engenheiro de *software* possa apenas executá-lo e gerencia-lo, não necessitando de projeções manuais.

2.2 OBJETIVOS ESPECÍFICOS

- Desenvolver uma pesquisa bibliográfica sobre Métricas de *Software*;
- Aprofundar-se no conhecimento específico de Análise de Pontos de Função;
- Planejar o *software* e fazer a sua modelagem;
- Desenvolver o programa em uma *interface* clara e de fácil utilização utilizando a linguagem JAVA;
- Programar todas as variáveis necessárias que a análise de pontos de função utiliza, no sentido de deixar o cálculo e a estimativa com maior exatidão e confiabilidade.
- Armazenar no banco de dados, em MySQL, o histórico de estimativas de projetos *software* para utilizar-se com um indicador nos próximos trabalhos.
- Testar o *software* com base no desenvolvimento do próprio projeto.

3 JUSTIFICATIVA

Existem poucos *softwares* no mercado voltados para a Análise de Pontos de Função e, dos que existem, usam *interfaces* confusas e de difícil entendimento ao usuário.

Este contexto acabou motivando a escolha deste tema, pois existe uma necessidade de um programa de fácil utilização e de *interface* simples e clara, com linguagens naturais, para que qualquer tipo de usuário possa utilizar, poupando-se de projeções de cálculos manuais, atendendo tanto a profissionais, que terão mais tempo para a gestão do projeto, e focar em outros assuntos pertinentes, quanto a alunos, onde poderão ver na prática os resultados de uma métrica e onde isso poderá ajuda-los no decorrer do projeto.

4 *SOFTWARE*

O *software* de computador é um produto que os profissionais da área da computação desenvolvem e dão suporte a longo prazo. É mais elemento de sistema lógico do que físico, tornando-o diverso do que os seres humanos constroem. Um *software*, de modo geral, consiste em instruções que, quando executadas, satisfaçam o esperado, tendo as funções, características e desempenho desejado (PRESSMAN, 2011).

Pressman (1995) complementa a definição de sua obra mais atual citada acima com algumas características do *software*, tais como que os custos estão concentrados no trabalho de engenharia. Isso significa que o *software* não pode ser gerido como se fosse projetos de manufatura.

Outra característica que Pressman (1995) cita de extrema importância é que o *software* não se “desgasta”, diferentemente do *hardware*. Porém, o *software* se deteriora devido a mudanças durante sua vida. Quando ocorrem essas “mudanças”, ou manutenções, os defeitos antigos são corrigidos, mas é provável que novos defeitos sejam introduzidos, fazendo com que o índice de falhas apresente picos, como mostra a Figura 1.

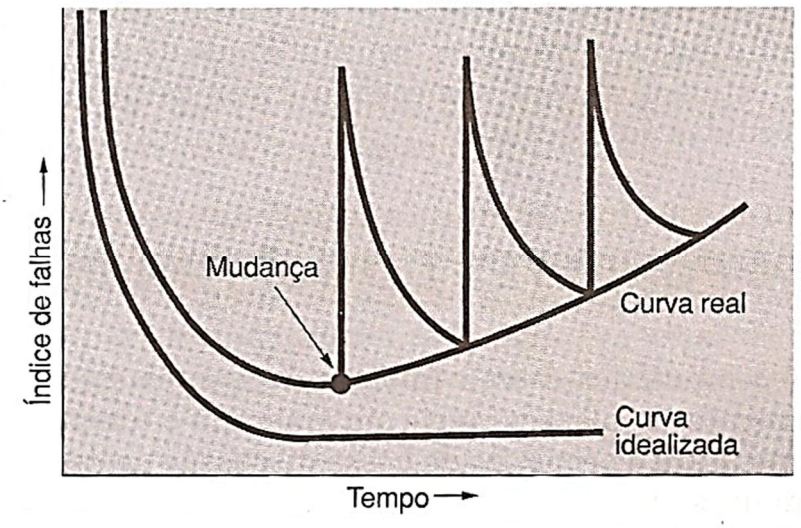


Figura 1 - Curva de falhas para o *software.*
Fonte: Pressman (1995).

Os defeitos não descobertos no início elevarão estes índices de falha logo no começo de vida de um programa.

Pensando em qualidade, Pfleeger (2004) inclui o engenheiro de *software* no chamado “bom *software*”, onde cita que da mesma maneira, que fabricantes buscam modos de assegurar a qualidade dos produtos que fabricam, os engenheiros de *software* também devem utilizar alguma estratégia para que seus produtos saiam com qualidade.

5 ENGENHARIA DE *SOFTWARE*

Segundo Sommerville (2011), a engenharia de *software* é uma disciplina de engenharia, cujo foco está na produção do *software*, desde a fase inicial da especificação do sistema até sua manutenção, quando o sistema já esta em uso.

Em geral, os engenheiros de *software* adotam uma sistemática de trabalho, onde costuma ser de maneira mais eficiente produzir um *software* de alta qualidade. No entanto, tem tudo a ver com selecionar o método mais adequado para um conjunto de circunstâncias.

O autor acima citado define a engenharia de *software* de forma direta e objetiva, explicando que cada projeto é diferente e deve ser tratado muitas vezes de maneira divergente um do outro, escolhendo o melhor método.

Pressman (1995) define a engenharia de *software* como um rebento da engenharia de sistemas e de *hardware* e também inclui a importância dos métodos, como também das ferramentas e procedimentos, afirmando que esses conjuntos formam a base para a construção de um *software* de alta qualidade produtivamente.

O autor ainda examina brevemente cada um desses elementos:

- **Métodos de engenharia de *software***: proporcionam os detalhes de "como fazer" para construir o *software*. O método envolve várias tarefas que incluem: planejamento e estimativa, análise de requisito de *software* e de sistemas, projeto da estrutura de dados, arquitetura de programa e algoritmo de processamento, codificação, teste e manutenção.
- **As ferramentas da engenharia de *software***: proporcionam apoio automatizado as tarefas citadas acima.
- **Os procedimentos da engenharia de *software***: constituem o elo de ligação que mantem os métodos e as ferramentas juntos. Os procedimentos definem a sequência em que os métodos serão aplicados, os produtos, os controles que ajudam a qualidade e a coordenação de mudanças, e o marcos de referência que auxiliam a avaliação do progresso.

6 PLANEJAMENTO E ESTIMATIVA DE *SOFTWARE*

Ao identificar a necessidade de desenvolvimento ou manutenção de um *software* para atender a novas demandas de uma organização, Vazquez (2012) diz que no mínimo dois questionamentos sempre estão presentes:

- Quanto tempo será necessário para concluir o projeto?
- Quanto o projeto vai custar para a organização?

Vazquez (2012) ainda relata que há dificuldades em encontrar respostas confiáveis a estas perguntas, uma vez que há muitas particularidades que podem atrapalhar o projeto, tais como:

- Os requisitos traduzem com fidelidade as necessidades do negócio dos usuários?
- A equipe de desenvolvimento possui conhecimento na área de negócio que será atendida pelo projeto de *software*?
- A equipe possui experiência na utilização das ferramentas necessárias a conclusão do projeto?
- Há conflitos internos na equipe?
- Pode ser facilmente absorvida caso entrar novos integrantes na equipe?
- Exige um pessoal com algum tipo de especialização?

Devido a estas e outras inerentes a cada projeto, determinar exatamente seu custo final, por exemplo, só seria possível no final do projeto. Antes disso, o que pode ser feito são as estimativas.

Presmman (1995) frisa que em muitos casos, estas estimativas são feitas de experiências passadas como um único guia. Se o projeto passado for semelhante, o custo, tempo estimativas de esforço considera-se iguais este trabalho antigo. Porém, se o projeto romper novos horizontes, somente experiência passada poderá não ser suficiente.

O autor informa as técnicas de estimativas utilizadas tem os seguintes pontos em comum:

- O escopo do projeto deve ser estabelecido antecipadamente;

- O projeto é dividido em pequenas partes que são estimadas individualmente;
- Devem-se utilizar métricas de *software*;
- Histórico de aferições passadas é usado como uma base.

Segundo Vazquez (2012), a partir da manutenção de uma base de dados históricos estimados e realizados dos projetos, a organização pode extrair indicadores de produtividade e qualidade cada vez mais próximos da realidade, portanto, mais confiáveis. Com esses indicadores, as estimativas dos futuros projetos podem ser realizadas com mais segurança, confiabilidade e mais cedo possível, ocasionando decisões mais rápidas e com menor custo para a organização.

Para ajudar a extrair tais indicadores e também a obter respostas aos questionamentos do início deste capítulo, em relação às estimativas, define-se uma forma de medir o *software*, ou seja, uma forma de definir um tamanho para o *software.*

A medição do *software* é chamada de métrica e é discutida no capítulo seguinte.

7 MÉTRICAS

7.1 MEDIDAS E MÉTRICAS DE *SOFTWARE*

Para Pressman (2011), a medição é um elemento chave de qualquer processo de engenharia. Através dela que os engenheiros de *software* visualizam o projeto e a construção do *software*, focalizando atributos específicos e mensuráveis dos artefatos da engenharia de *software*. Mas diferentemente de outras disciplinas, a engenharia de *software* não é fundamentada nas leis quantitativas da física. Por serem medidas indiretas, elas estão abertas ao debate, onde ainda alguns membros da comunidade de *software* continuam a argumentar que o *software* é “incomensurável” ou que tentativas de medições deverão ser adiadas até entendermos melhor o *software* e os atributos que deverão ser usados para descrevê-los. Isso é um erro.

Como relata o autor, ainda há muita desconfiança quanto a métricas de *software*, porém, ela é primordial em qualquer projeto. É claro que essas medidas não são tão precisas fisicamente e são bastante difíceis de quantificar, mas através delas são tomadas decisões importantes ao caminhar do projeto.

Pressman (1995) diz que essas medidas que indicam aspectos de funcionalidade, qualidade, eficiência, ou seja, as mais difíceis de quantificar são chamadas de medições indiretas. Há ainda as medidas diretas, que são aquelas que são mais fáceis de serem obtidas, por serem mais “físicas”. O autor nos fornece também alguns exemplos como: o número de linhas de código produzidas, o tamanho de memória ocupado, a velocidade da execução, o número de erros registrados num dado período de tempo, etc.

Sommerville (2011) destaca que métrica de *software* é uma característica do sistema, da documentação ou de um processo de desenvolvimento que pode ser objetivamente medido. O autor ainda afirma que métricas de *software* podem ter como objetivo fazer o controle, suportando os processos de gerenciamento, ou para fazer previsões, ou seja, prevendo características do *software*. Essas métricas podem influenciar a tomada de decisão de gerenciamento, conforme ilustrado na Figura 2.

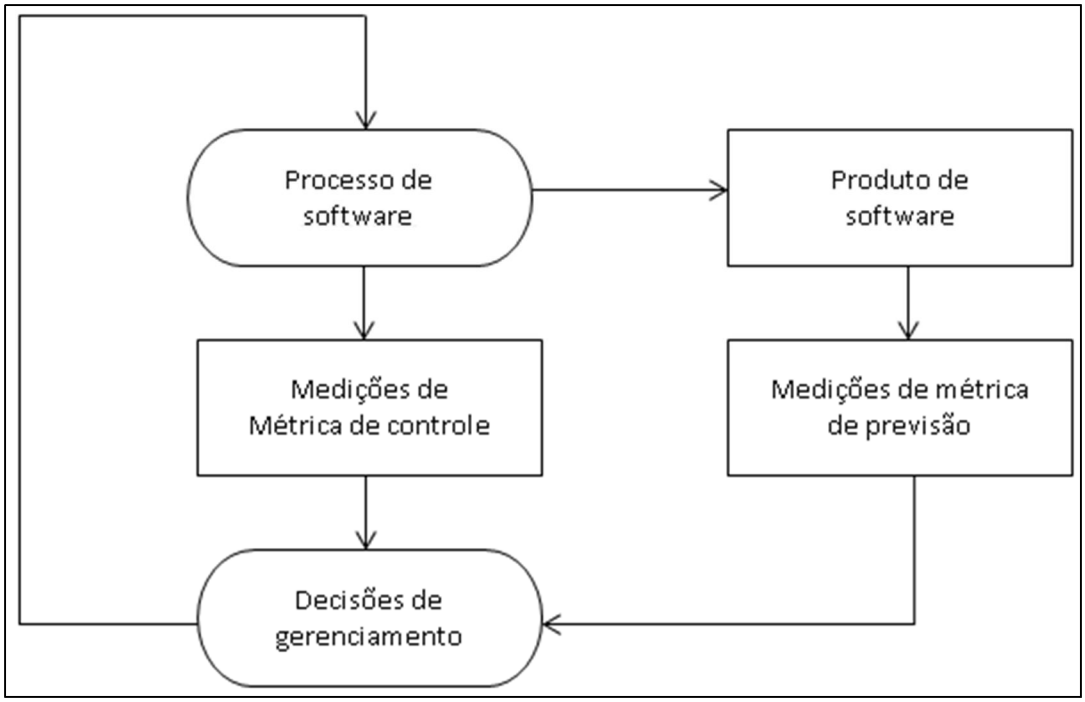


Figura 2 - Medições de previsões e de controle.
Fonte: Sommerville (2011).

Schach (2009) concorda em relação à ambos os autores anteriormente citados, e ainda cita exemplos de métricas que podem ser aplicadas ao longo de processo de *software*, como: para cada fluxo de trabalho pode-se medir o esforço em homens-mês; métrica de custo que é fundamental e que deve ser acompanhada durante todo o projeto; a rotatividade de mão de obra, uma vez que o índice de *turn-over* seja alto, prejudica a equipe, tendo perda de tempo com integração do novo colaborador.

Os autores supramencionados entendem que a métrica de *software* é essencial não só naquele momento, mas sim durante toda a “vida” do projeto.

E para que medir um *software*? Pressman (1995) cita muitas razões, tais como: indicar a qualidade do produto, avaliar a produtividade das pessoas, formar uma linha básica para estimativas, ajudar a justificar os pedidos de novas ferramentas ou treinamento adicional.

O autor cita a razão “avaliar a produtividade das pessoas”, que não deve ser entendido como uma disputa ou competição, ou talvez, ficar com comparações de colaboradores com outros colaboradores. A intenção é ter uma medida em que, de um

conjunto, o engenheiro de *software* desenvolve métricas para obter indicadores. O autor, em sua obra mais atual, exemplifica muito bem isso:

> [...] Quando é coletado um único ponto de dado (por exemplo, o número de erros descobertos em um componente de *software*), foi estabelecida uma medida. A medição ocorre como resultado da coleção de um ou mais pontos de dados (por exemplo, um conjunto de revisões de componente e testes de unidade são investigados para coletar medidas do número de erros para cada um). Uma métrica de *software* relaciona as medidas individuais de alguma maneira (por exemplo, o número médio de erros encontrados por revisão ou o número médio de erros encontrados por teste de unidade). Um engenheiro de *software* coleta medidas e desenvolve métricas para obter indicadores (PRESSMAN, 2011, p. 539).

7.2 PRINCIPIOS DE MEDIÇÃO

Antes de introduzir qualquer métrica de produto, Roche (1994 citado por PRESSMAN, 2011), sugere um processo para uma medição, que pode ser caracterizado por cinco atividades:

- **Formulação**: Criar medidas e métricas de *software* apropriadas para representar o *software* considerado.
- **Coleção**: O mecanismo para acumular dados necessários para criar as métricas formuladas.
- **Análise**: Computar as métricas e aplicar as ferramentas matemáticas;
- Interpretação: Avaliar as métricas que resultam em informações sobre a qualidade da representação.
- ***Feedback***: Transmitir as recomendações derivada da interpretação de métricas de produto para a equipe de trabalho.

O autor cita a equipe de trabalho no último item, recomendando deixa-la ciente das recomendações derivadas da interpretação das métricas. Esta atividade é importante e deve ser primordial tanto quanto as outras quatro. Talvez até, implementá-la, deixando a equipe a par de todas as informações do projeto todo. Isso ajuda muito tanto na motivação no trabalho quanto na delegação de tarefas e aprendizado.

7.3 DIVISÕES DE MÉTRICAS EM CATEGORIAS

Dentro do contexto de gerenciamento de projetos de *software*, há uma maior preocupação com métricas de produtividade e de qualidade, é o que afirma Pressman (1995), que são medidas do resultado do desenvolvimento de *software* como uma função de esforço aplicado e medidas da “adequação ao uso” dos resultados obtidos.

O autor explica que o domínio das métricas pode ser divido em mais categorias e, não somente, em medidas diretas e indiretas. Essas divisões podem ser feitas em métricas de produtividade, métricas de qualidade e métricas técnicas. E que outra divisão restringe-se em métricas orientadas ao tamanho, orientadas à função e orientadas às pessoas. Verifica-se melhor essa duas classificações segundo a Figura 3.

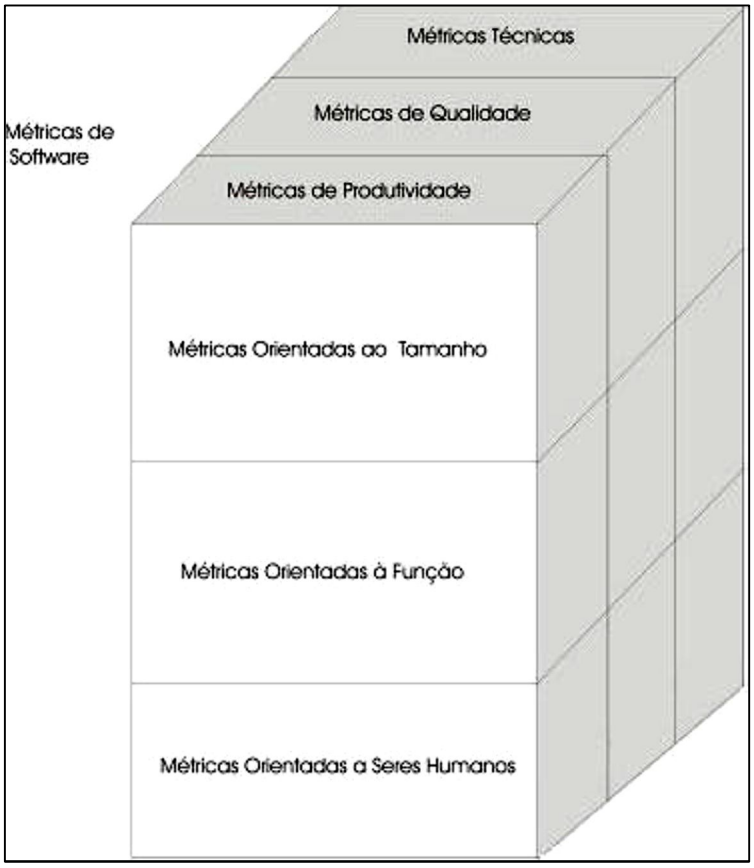


Figura 3 - Divisão das métricas em categorias.
Fonte: Pressman (1995).

Pressman (1995) faz a explicação da primeira divisão da Figura 3 dizendo que as métricas de produtividade concentram-se na saída do processo de engenharia de *software.* As métricas de qualidade oferecem uma indicação de quão estreitamente o sistema conforma-se as exigências implícitas e explicitas do cliente. Já as métricas técnicas concentram-se nas características do *software* e não no processo do qual o sistema foi desenvolvido.

Observando a segunda divisão da Figura 3, o autor conceitua de forma breve essas métricas, que nos próximos capítulos serão tratados mais especificadamente.

> [...] *Métricas orientadas ao tamanho* são usadas para compilar as medições diretas da saída e da qualidade da engenharia de *software*. *Métricas orientadas para a função* oferecem medidas indiretas, e *Métricas orientadas às pessoas* compilam informações sobre a maneira segundo a qual as pessoas desenvolvem *software* de computador e percepções humanas sobre a efetividade das ferramentas e métodos (PRESSMAN, 1995).

7.4 MÉTRICA DE PRODUTO

Schach (2009) vai direto ao ponto e define que métricas de produto indicam algum aspecto do próprio produto, por exemplo, seu tamanho.

Sommerville(2011) afirma que métricas de produto são feitas para previsão e usadas para medir atributos internos de um sistema. O tamanho do sistema em linhas de código e número de métodos associados a cada classe são exemplos de métricas de produto, tais que são divididas em duas classes:

- Métricas dinâmicas, que são coletadas por meio de medições efetuadas de um programa em execução. Podem ser coletados durante o teste ou após o sistema estar em uso. Pode ser o número de relatórios de *bugs* por exemplo. Estes tipos de métricas ajudam na avaliação de eficiência e a confiabilidade de um programa;
- Métricas estáticas, que são coletadas por meio de medições feitas de representações do sistema, como o projeto e a documentação. Pode ser o

> número de variáveis usadas por exemplo. Estes tipos de métricas ajudam na avaliação de complexidade e manutenibilidade do sistema;

Pressman (2011, p. 540), complementa que “essas métricas serão úteis só se forem efetivamente caracterizadas e validadas de forma que demonstrem valer a pena”. Lethbridge (2003 citado por PRESSMAN, 2011) cita alguns princípios que podem ser seguidos, como um exemplo abaixo:

> Uma métrica deve ter as propriedades matemáticas desejadas. O valor da métrica deverá estar em um intervalo significativo (por exemplo, 0 a 1, em que 0 significa ausência, 1 indica o valor máximo e o 0,5 representa “ponto médio”). Além disso, uma métrica que deve estar em uma escala racional não deve ser composta de componentes medidos apenas em escala ordinal (PRESSMAN, 2011).

7.5 MÉTRICAS ORIENTADAS AO TAMANHO

Segundo Pressman (1995), métricas de *software* orientadas ao tamanho são medidas diretas do *software* e do processo pelo qual ele é produzido. Se a empresa possuir registros simples, uma tabela como a da Figura 4, poderá ser criada.

projeto	esforço	$	KLOC	págs. docum.	erros	pessoas
aaa-01	24	168	12.1	365	29	3
ccc-04	62	440	27.2	1224	86	5
fff-03	43	314	20.02	1050	64	6
.	.	.	.	.		
.	.	.	.	.		
.	.	.	.	.		

Figura 4 - Métricas Orientadas ao Tamanho.
Fonte: (Pressman, 1995).

A Figura 4 relaciona cada projeto de desenvolvimento de *software* que foi concluído no decorrer dos últimos anos aos correspondentes dados orientados ao tamanho do projeto.

Verificando a primeira linha desta Figura 4, para o projeto aaa-01 foram desenvolvidas 12.1 KLOC (12.100 linhas de código), com 24 pessoas-mês esforço a um custo de 168 mil dólares. O esforço e o custo representam todas as atividades de engenharia de *software*, desde a análise de requisitos até os testes, e não apenas a codificação. Um total de 365 páginas de documentação foram escritas e 29 erros encontrados após o primeiro ano de uso do *software* pelo cliente. Três pessoas trabalharam neste projeto aaa-01.

A partir dos dados desta tabela, um conjunto de métricas de qualidade e produtividade orientadas ao tamanho pode ser desenvolvido para cada projeto, sendo poderiam ser computadas médias, como por exemplo: Qualidade = defeitos/KLOC.

Segundo Jones (1986 citado por PRESSMAN, 1995), as métricas orientadas ao tamanho provocam controvérsias e não são universalmente aceitas como a melhor maneira de se medir o processo de desenvolvimento de *software*. O motivo principal é o uso de LOC como medida, já que seu uso em estimativas é complexo, necessitando de um nível de detalhes muito grande e seu uso em estimativas requer um nível de detalhes que pode ser difícil de conseguir, isto é, o planejador deve estimar o número de linhas de código que serão desenvolvidas muito antes que a análise e o projeto tenham sido concluídos.

Schach (2009) tem um ponto de vista mais conservador. Ele explica que se deve ficar muito atento ao uso de métrica para o tamanho de um produto usando linhas de código, pois existem muitos problemas associados a essa métrica. A criação do código-fonte é apenas uma pequena parte do esforço total do desenvolvimento de *software*, uma vez que todo levantamento de necessidade, documentação, planejamento e testes não pode ser expresso única e exclusivamente em função de número de linhas.

Sem falar nos comentários, que se não contarem, o programador pode escrever páginas e páginas de código sem comentar e, se contarem, pode haver linhas e linhas de comentários improdutivos.

O autor conclui e alerta ainda que o número de linhas de código pode ser determinado quando o produto estiver completamente pronto. Não deve tomar como base para estimativas de custos, pois será extremamente perigoso. Para iniciar o

processo de estimativas, o número de linhas deve estar concluído, para ai sim, calcular o custo do produto.

7.6 MÉTRICAS DE QUALIDADE DO *SOFTWARE*

A qualidade pode ser medida ao longo do processo de engenharia de *software* e depois que o *software* foi entregue ao cliente. Pressman (1995) afirma que métricas derivadas antes que o *software* seja entregue oferecem uma base quantitativa para tomar decisões. Métricas de qualidade neste caso incluem a complexidade do programa, modularidade efetiva e, acima de tudo, tamanho do programa. Já as métricas usadas após a entrega concentram-se no número de defeitos descobertos.

7.6.1 Medida da Qualidade

Embora existam muitas medidas de qualidade de *software*, há quatro principais que fornecem indicadores úteis para a equipe de projeto. Gilb (1988 citado por PRESSMAN, 2011) sugere e define cada uma delas:

- **Correção**: Um programa deve operar corretamente. A correção é o grau com o qual o *software* executa sua função. A medida mais comum é o numero de defeitos por milhares de linhas de programação, em que um defeito é definido como uma ocorrência de falta de conformidade com os requisitos. Para fins de avaliação de qualidade, os defeitos são contados durante um período de tempo, em geral um ano;
- **Manutenibilidade**: é a facilidade com que um programa pode ser corrigido caso um problema for encontrado. Não existe nenhuma forma de se medir a manutenibilidade diretamente, ou seja, devemos usar medidas indiretas. Uma métrica bem simples orientada ao tempo é o tempo médio para mudança (MTTC). Um programa com boa manutenibilidade terá um MTTC menor do que *softwares* difíceis de serem mantidos;

- **Integridade**: Esse atributo mede a capacidade que um *software* tem de suportar ataque (acidentais ou intencionais) à sua integridade. Para se medir a integridade, dois atributos devem ser definidos: ameaças e segurança. Ameaça é a probabilidade de que um ataque de um tipo específico ocorre em um determinado tempo. Segurança é a probabilidade de que o ataque de um tipo específico será repelido. Ambos podem ser estimados a partir de evidencia empírica;
- **Usabilidade**: Se um programa for difícil de usar, seu destino pode ser um fracasso, mesmo ele tendo funções muito importantes. A usabilidade é uma tentativa de quantificar a facilidade do uso.

7.7 MÉTRICAS ORIENTADAS À FUNÇÃO

Pressman (1995) afirma que as métricas de *software* orientadas à função são medidas indiretas de *software* e do processo ao qual é desenvolvido. A métrica orientada à função não conta linhas de código, e sim a “funcionalidade” ou ”utilidade” do programa.

Métricas orientadas à função foram propostas pela primeira vez por Albrecht (1979), onde sugeriu uma abordagem à medição da produtividade chamada método do ponto-por-função (*function point).* Esses pontos-por-função, ou FPs, são baseados em medidas de informações e complexidade do *software*.

Já Dekkers (2003 citado por HAZAN, 2008) faz as mesmas afirmações acima de Pressman (1995) e completa que a métrica pontos de função é uma medida de tamanho funcional de projetos de *software*, considerando as funcionalidades implementadas, sob o ponto de vista do usuário. A contagem de pontos de função é independentemente da metodologia de desenvolvimento utilizados do andamento *software.* Portanto, a autora recomenda a utilização desta métrica nas estimativas de tamanho de projetos de *software*.

Métricas orientadas à função serão melhor discutidas no capítulo a seguir.

8 ANÁLISE DE PONTOS DE FUNÇÃO – MEDIÇÃO FUNCIONAL

A técnica de análise de pontos de função surgiu na IBM, inicio da década de 70, como alternativa às métricas baseadas em linhas de código. O encarregado de medir a produtividade de vários projetos de *software* desenvolvidos naquela época foi Allan Albrecht e o que o motivou a uma busca por uma medida que fosse independente da linguagem de programação utilizada, foi pelo fato de que aqueles projetos tinham sido desenvolvidos em linguagens de programação distintas, tornando-o inviável uma análise conjunta de produtividade utilizando métricas por linhas de código.

Após a apresentação da técnica a comunidade, final da década de 70, e de sucessivos trabalhos efetuados por Capers Jones, houve um alto crescimento em usuários utilizando a análise de pontos de função, culminando em 1986, com a fundação da *International Function Point Users Group* (IFPUG), onde dois anos depois foi publicado um Manual de Práticas de Contagem (CPM – *Counting Practices Manual)*, com o objetivo de padronização da técnica.

A versão mais recente é a CPM 4.3.1. Embora tenham surgido outras técnicas de medição funcional, nenhuma delas possui ainda a aceitação tão ampla pelo mercado quanto os pontos de função do IFPUG (VAZQUEZ, 2012).

Complementando o histórico, Engholm (2010) cita que, no Brasil existe, a BFPUG (*Brazilian Function Point Users Group),* a qual administra a certificação no país.

8.1 IFPUG

A International Function Point Users' Group (IFPUG) é uma organização sem fins lucrativos, membro da organização governada. A missão do IFPUG é ser reconhecida como líder na promoção e incentivo à gestão eficaz de desenvolvimento de software aplicativo e atividades de manutenção através do uso de Análise de Pontos de Função (APF) e outras técnicas de medição de software. IFPUG endossa a APF como metodologia padrão para software de dimensionamento. Em apoio a esta, IFPUG mantém o Manual de Práticas de Contagem. IFPUG também fornece um fórum para

relacionamento e troca de informações que promove e incentiva o uso de produtos de software e métricas de processo. Membros IFPUG beneficiam de uma variedade de serviços, tais como: conferência anual, seminários e oficinas educacionais, certificação profissional, comissões de trabalho e grupos de tarefas.

IFPUG produz e mantém um conjunto de publicações oportunas sobre as normas e diretrizes de medição de software.

VAZQUEZ (2012) afirma que o manual de práticas de contagem está disponível somente a membros do IFPUG. Para não membros, somente mediante a aquisição, dificultando um pouco a difusão da técnica.

8.2 REQUISITOS FUNCIONAIS

Requisitos funcionais são aqueles que capturam o que o *software* deve fazer em termos de tarefas. São requisitos relativos às práticas e procedimentos do negócio do usuário, particulares de determinas tarefas (VAZQUEZ, 2012).

8.3 DEFINIÇÃO E OBJETIVO DA ANÁLISE DE PONTOS DE FUNÇÃO

Vazquez (2012) define análise de pontos de função de forma clara e objetiva. Ele diz que é uma técnica de medição das funcionalidades fornecidas por um *software* do ponto de vista do usuário. A medição é independente da tecnologia utilizada para o desenvolvimento do sistema, ou seja, a APF busca medir o que o produto faz, e não como ele foi construído.

O processo de medição desta técnica baseia-se em uma avaliação padronizada dos requisitos lógicos do usuário.

Dekkers (1998) explica que pontos de função não medem a produtividade ou o esforço. Pontos de função medem o tamanho do que o *software* faz ao invés de como ele é desenvolvido e implementado. Isto significa que, dado um conjunto de requisitos

de usuário, o tamanho funcional do *software* será o mesmo, independente de sua linguagem.

A autora anteriormente citada complementa que saber o tamanho do *software* é um dos primeiros passos do processo de estimativa de esforço, prazo e custo, porém, ele destaca que pontos de função não medem diretamente essas estimativas, mas sim, mede exclusivamente o tamanho funcional do *software*. Este tamanho com outras variáveis é que pode ser usada para as estimativas citadas acima.

Engholm (2010) destaca que a APF deve ser simples o suficiente a fim de minimizar o esforço necessário para efetuar a medição, com isso, o método pode ser utilizado como uma medida consistente entre varias aplicações e organizações.

Quanto ao objetivo, o IFPUG (2010 citado VAZQUEZ, 2012) define como objetivos primários os seguintes:

- Medir a funcionalidade que o usuário solicita e recebe.
- Medir o desenvolvimento e manutenção do *software* de forma independente da tecnologia utilizada para sua implementação.

8.4 CONTAGEM DE PONTOS DE FUNÇÃO

Engholm (2010) resume brevemente a contagem de pontos de função, que consiste em contagens de função apenas do ponto de vista funcional, que estão divididas em transações (funcionalidades que processam as informações) e dados (depósito de dados).

Para calcular o tamanho de uma aplicação, deve-se identificar as funcionalidades conforme observadas pelo usuário. Por exemplo, existe a aplicação de “Cadastro de Pedidos” com as funcionalidades de “Incluir Cliente” e “Lista Cliente”. Ao aplicar as regras de contagem da APF, chega-se a um valor que determina o tamanho da funcionalidade, logo o tamanho da aplicação.

Vazquez (2012) ressalta que a APF mede especificadamente os requisitos funcionais do usuário, e essa dimensão é dada o nome de tamanho funcional. Ela pode ser descrita com base em quais tipos de elementos são contados e no processo pelo

qual esses elementos são identificados e a eles atribuídos um peso. O diagrama da Figura 5 apresenta as etapas deste processo, de acordo com a IFPUG, bem como as relações de interdependência entre seus passos.

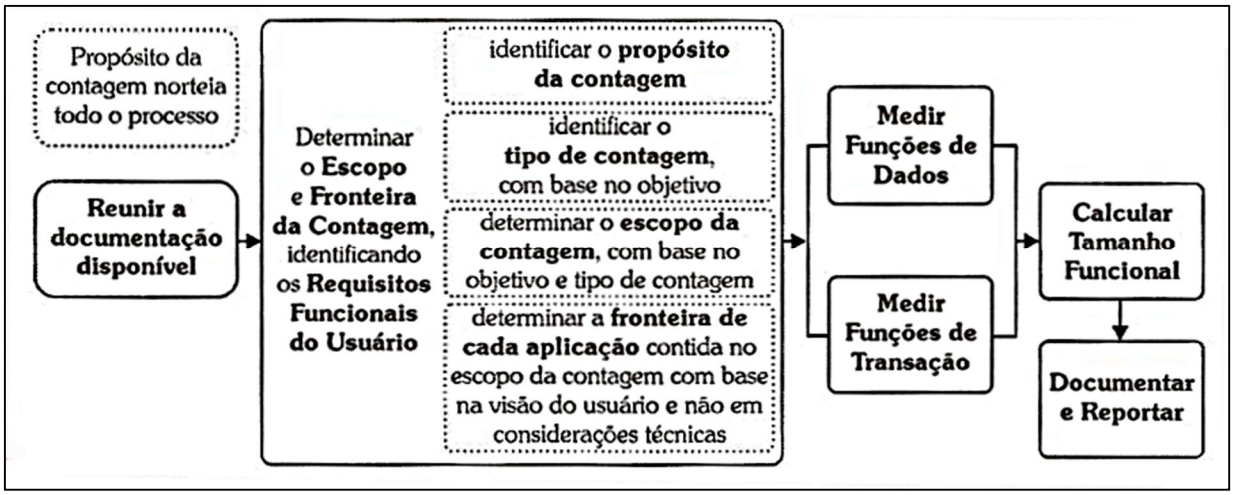


Figura 5 - Visão geral do processo de medição funcional do IFPUG.
Fonte: Vazquez (2012).

O autor explica que, como a medição da APF atua sobre os requisitos de software e dificilmente são documentados de maneira perfeita, surge a possibilidade de que haverá várias interpretações no mesmo requisito, consequentemente afetando o resultado da medição. Portanto, a qualidade das documentações e dos requisitos de software afeta diretamente a qualidade da medição.

Basicamente a medição consiste em decompor o projeto em elementos denominados componentes funcionais básicos (ou funções). Vazquez (2012) ainda afirma que o método de medição conceitua abstrações, os tipos de função, nos quais os componentes básicos são classificados. Essa classificação é feita conforme sua função de armazenamento ou transação, sendo necessária a solicitação do usuário, conforme ilustra a Figura 6:

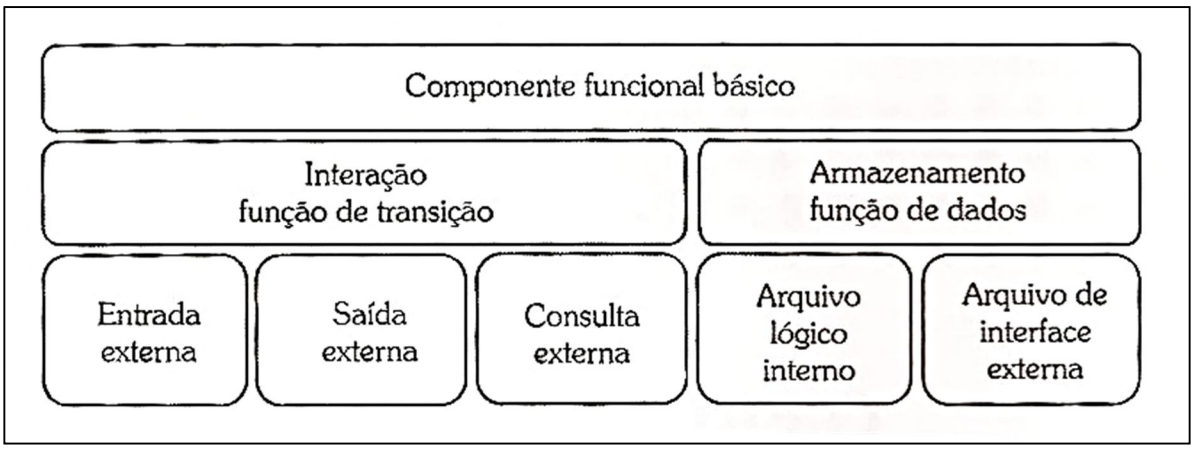


Figura 6- Classificação dos tipos de função.
Fonte: Vazquez (2012).

O método de contagem define referências para a identificação de cada componente funcional básico, classificação quanto ao tipo de função e à complexidade funcional, e a determinação de sua contribuição individual. Esses componentes funcionais básicos são denominados de função ou funcionalidades. O diagrama apresentado na Figura 7 define o que é externo e interno no sistema, bem como apresenta os tipos de função e o principal referencial na identificação das funções: a fronteira da aplicação.

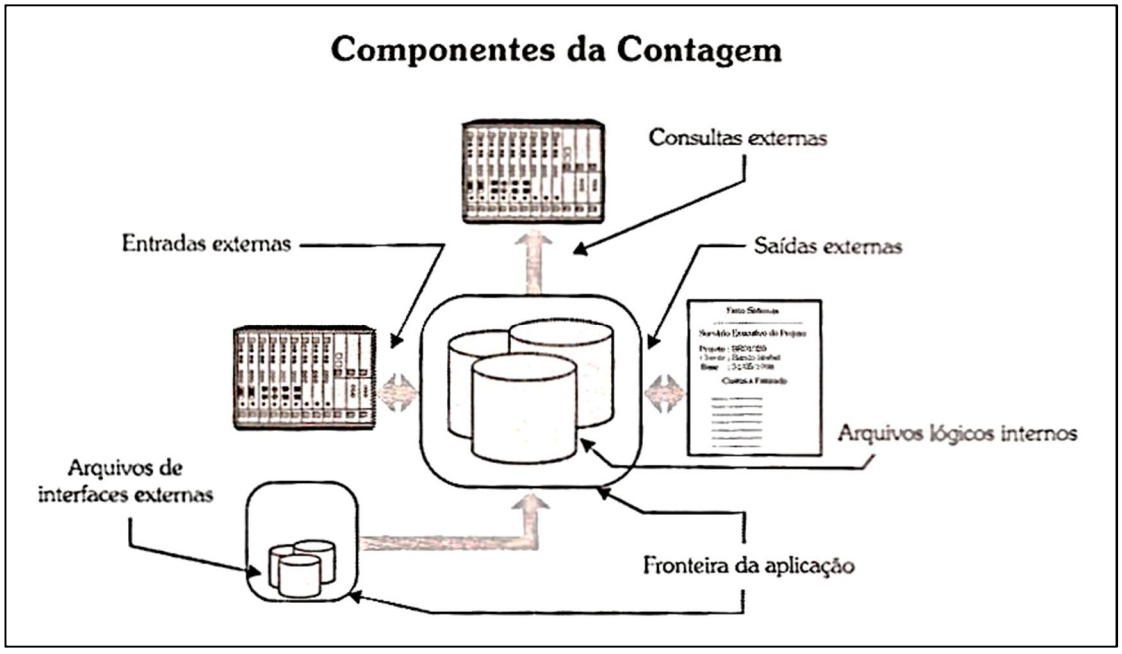


Figura 7 - Elementos de contagem de pontos de função.
Fonte: Vazquez (2012).

9 ANÁLISE DE PONTOS DE FUNÇÃO – VISÃO GERAL DO PROCESSO

Este capítulo apresenta a visão geral da medição de *software* em pontos de função de acordo com a IFPUG.

Todo o processo se baseia de acordo com o diagrama da Figura 8.

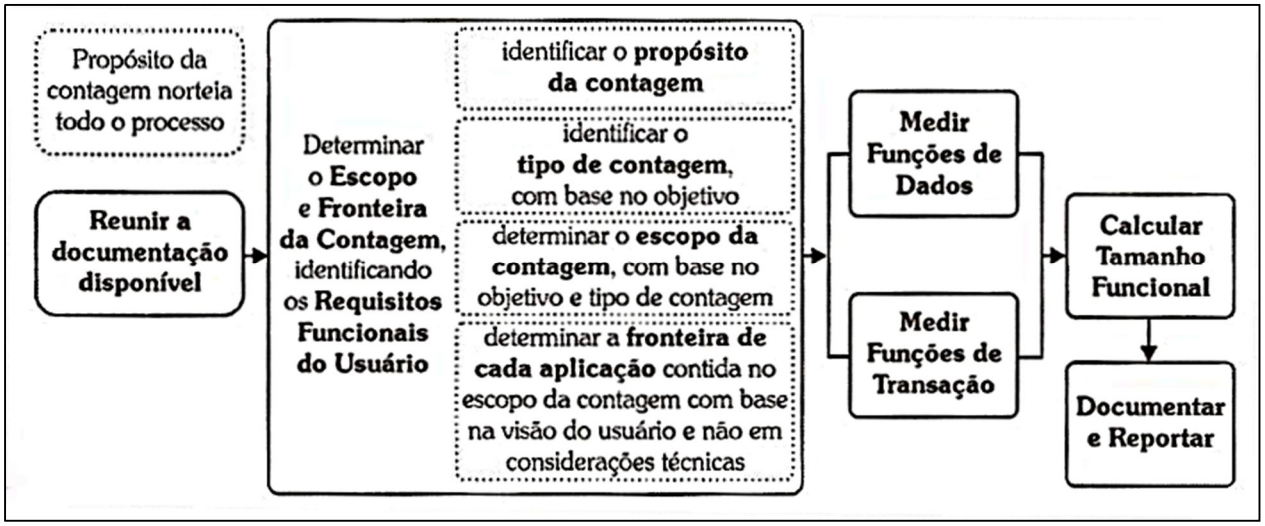


Figura 8 - Visão geral do processo de medição funcional do IFPUG.
Fonte: Vazquez (2012).

9.1 PROPÓSITO DA CONTAGEM

Para tudo sempre há uma motivação ou um propósito para fazer determinada tarefa e não é diferente com a APF. Vazquez (2012) fornece alguns exemplos de propósitos da contagem de pontos de função, como por exemplo, medir o serviço entregue para sua posterior remuneração. Ou ainda, contagem com o propósito de fornecer elementos para uma estimativa de custo de um projeto. De acordo com o propósito, é possível assumir alguns objetivos que podem agilizar o processo.

9.2 REUNIR A DOCUMENTAÇÃO DISPONÍVEL

O primeiro passo da medição consiste em buscar a documentação disponível sobre o projeto ou sistema que será medido.

Segundo Vazquez (2012), o propósito da contagem citado no tópico anterior deste trabalho ajuda a definir que documentos são mais interessantes para o trabalho de medição. De qualquer forma a medição ideal deve descrever a funcionalidade entregue pelo *software* ou descrever a funcionalidade que é impactada pelo projeto de *software* medido.

Os documentos que podem ser utilizados na medição são os que a engenharia de *software* conhece bem, tais como diagramas de classe, diagramas de fluxo de dados, casos de uso, relatórios.

O autor ainda explica que o se o analista de métricas tem à sua disposição um conjunto de documentos já devidamente organizados do projeto, essa etapa do processo acontece quase de forma instantânea. Por outro lado, se não há documentação suficientemente disponível, é preciso buscar o acesso a especialistas no negocio para complementar. Isso implica em agendamento, pesquisas e entrevistas, onde implica em esforço adicional na medição.

9.3 DETERMINAÇÃO DO TIPO DE CONTAGEM

Nessa fase, os responsáveis pela medição estabelecem o tipo de contagem que será utilizado para medir o *software* (VAZQUEZ, 2012).

Dekkers (1998), explica que existem três tipos de contagem:

1. Contagem de um projeto de novo desenvolvimento;
2. Contagem de um projeto de melhoria;
3. Contagem básica de aplicação.

Vazquez (2012) faz a explicação desses três tipos de contagem:

- **Projeto de novo desenvolvimento:** Mede a funcionalidade fornecida aos usuários finais do *software* quando da sua primeira instalação. Abrange também as eventuais funções de conversão de dados à implantação do sistema, como por exemplo, quando a organização importar informações seu sistema antigo para o nome.
- **Projeto de melhoria:** Mede as funções adicionadas, modificadas ou excluídas do sistema pelo projeto, e também em eventuais conversões de dados.
- **Aplicação:** Conhecida também como *baseline,* mede a funcionalidade fornecida aos usuários por uma aplicação já instalada. Ele é inicializado no final da contagem do número de pontos de função do projeto em desenvolvimento, sendo atualizado no término de todo projeto de melhoria que altera a funcionalidade da aplicação, como demonstra a Figura 9.

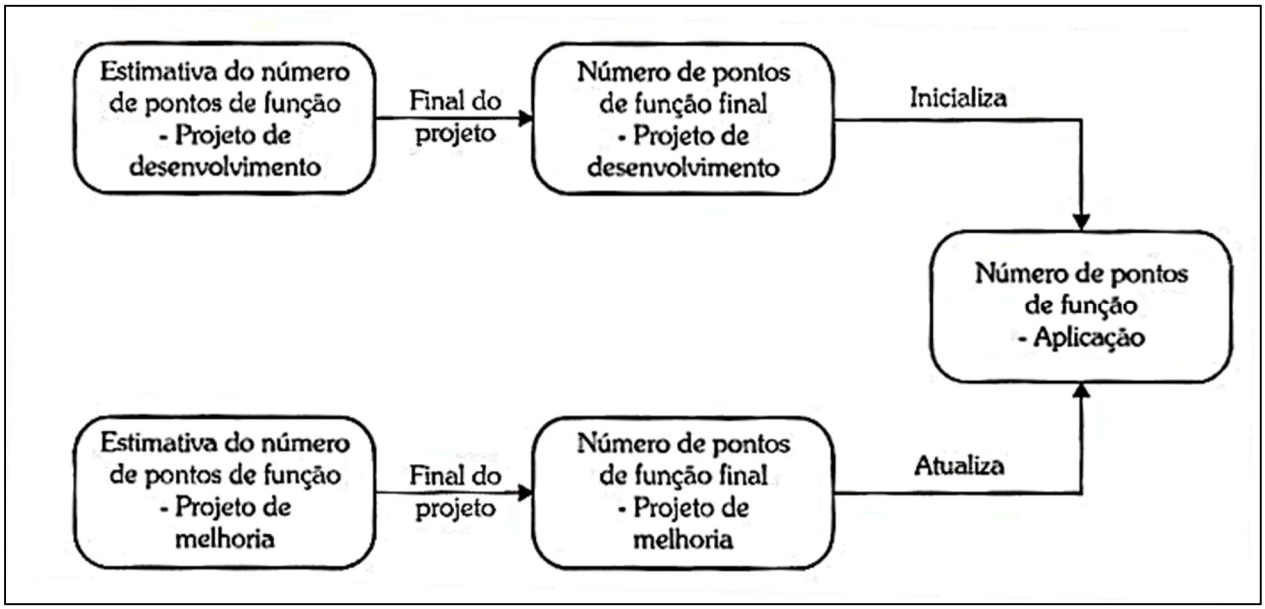


Figura 9 - Relacionamento entre os tipos de contagem.
Fonte: Vazquez (2012).

9.4 FRONTEIRA DA APLICAÇÃO

Vazquez (2012) define a fronteira da aplicação como a *interface* conceitual que delimita o *software* que será medido e o mundo exterior, no caso, os usuários.

Fazendo uma analogia com o mundo real para exemplificar, o autor diz que a fronteira da aplicação seria a cerca que delimita uma fazenda. Da mesma forma que não seria possível medir a área da fazenda se não houvesse a cerca estabelecendo seus limites, não é possível medir o tamanho funcional de um sistema sem que antes tenham estabelecido claramente a sua fronteira, principalmente se houver mais de uma aplicação incluída no escopo de uma contagem. Daí, várias fronteiras devem ser identificadas.

O IFPUG (2010 citado por VAZQUEZ, 2012) especifica as seguintes regras para determinação da fronteira da aplicação:

- Sua determinação deve ser feita com base no ponto de vista do usuário. O foco deve estar no que ele pode entender;
- A fronteira entre aplicações deve ser baseada na separação das funções conforme estabelecidos pelos processos do negócio;
- Em projetos de melhoria, a fronteira estabelecida no início do projeto deve estar de acordo com a fronteira já estabelecida para a aplicação sendo modificada.

9.5 ESCOPO DA CONTAGEM

O escopo da contagem define quais funções serão utilizadas na contagem, se abrange um ou mais sistemas ou parte de um sistema, onde ele pode abranger todas as funcionalidades ou apenas algumas especificas. Este conceito de escopo é mais visto durante a contagem de um projeto de melhorias (VAZQUEZ, 2012).

9.6 FUNÇÕES DE DADOS

Segundo Vazquez (2012), as funções do tipo dado representam as funcionalidades fornecidas pelo sistema ao usuário com o objetivo de atender as suas necessidades de armazenamento. São classificados como Arquivos Lógicos Internos

(ALI) e Arquivos de *Interface* Externa (AIE). Pressman (2011) exemplifica cada um deles:

- **Arquivo Lógico Interno (ALI):** Também conhecido como (*internal logical files* – ILFs), cada arquivo logico interno é um agrupamento lógico de dados que reside dentro das fronteiras do aplicativo e é mantido através de entradas externas.

IFPUG (1999 citado MACORATTI, c2010) fornece exemplos de ALIs:

- Dados da aplicação (arquivos mestres como cadastro de clientes ou funcionários).
- Arquivos de dados de segurança da aplicação.
- Arquivos de dados de auditoria.
- Arquivos de mensagem de auxílio.
- Arquivos de mensagens de erro.
- Arquivo de cópia de segurança. Considerado somente se for solicitado pelo usuário para atender requisitos da aplicação.
- Arquivo que sofra manutenção por mais de uma aplicação.

Ao identificar um grupo dados como um ALI, Vazquez (2012) explica que o foco deve estar em como o negócio manipula e armazena esse grupo de um plano conceitual e não na forma como a aplicação irá implementar. Um exemplo disso é se um usuário tem o requisito de armazenar compromissos financeiros e envolve também informações de parcelamento e rateio, ficará uma implementação de três tabelas. Sob a ótica do negócio, não é lógico que um compromisso esteja subdividido em vários diferentes arquivos. Logo, há um único ALI, conforme ilustra a Figura 10.

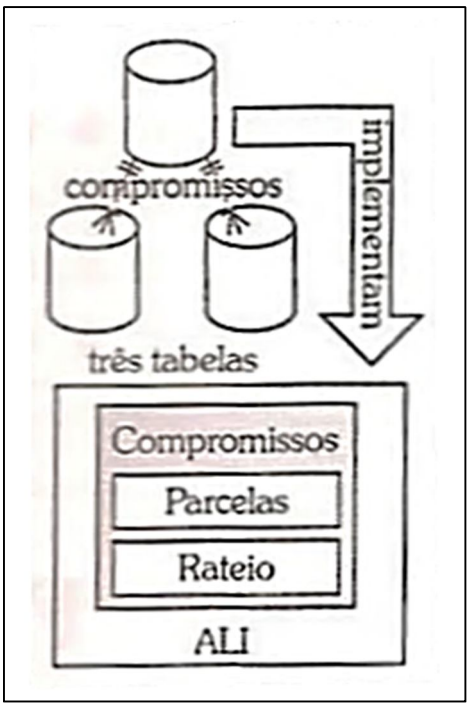


Figura 10 - ALI Compromisso implementado em três tabelas.
Fonte: Vazquez (2012).

O autor anteriormente citado destaca que para entender melhor a visão de negócio para a medição, deve-se imaginar a operação de negócio sem o uso de *softwares*, apenas com processos manuais e papel.

- **Arquivo de *Interface* Externa (AIE):** Também conhecido como (*external interface files* – EIFs), cada arquivo de *interface* externo é um agrupamento lógico de dados que reside fora da aplicação, mas fornece informações que podem ser usadas pela aplicação.

IFPUG (1999 citado por MACORATTI, c2010) fornece exemplos de AIEs:

- Arquivos de mensagens de auxílio.
- Arquivos de mensagens de erro.

Não são considerados AIEs:

- Dados recebidos de outra aplicação usados para adicionar, alterar ou remover dados em um ALI.

- Dados cuja manutenção é feita pela aplicação que esta sendo avaliada, mas que são acessados e utilizados por outra aplicação.
- Dados formatados e processados para uso por outra aplicação.

Pode-se notar que a principal diferença entre ALI e AIE é que o Arquivo de *Interface* Externa não é mantido pela aplicação. Ele esta fora da fronteira da aplicação, enquanto o Arquivo Lógico Interno está dentro. O AIE é um ALI de outra aplicação (VAZQUEZ, 2012).

9.6.1 Determinação da Complexidade

Segundo Macoratti (c2010), cada Arquivo de *Interface* Externa e Lógico Interno deve ser classificado de acordo com sua complexidade funcional, que é baseada no número de Registros Lógicos (RL) e no número de Itens de Dados (ID) do arquivo.

Vazquez (2012) trata a denominação Itens de Dados como Tipos de Dados (TD) e de Registros Lógicos como Tipos de Registros (TR) e faz uma classificação com relação à complexidade fornecida pela tabela ilustrada na Figura 11.

		Tipos de Dados		
		< 20	**20 - 50**	**> 50**
Tipos de Registros	**1**	Baixa	Baixa	Média
	2 - 5	Baixa	Média	Alta
	> 5	Média	Alta	Alta

Figura 11 - Complexidade funcional dos ALI e AIE.
Fonte: Vazquez (2012).

Como se pode perceber através desta figura, caso um ALI ou um AIE com 47 tipos de dados e 3 tipos de registros, sua complexidade será média.

9.6.2 Tipo de Dado (TD)

Vazquez (2012) explica que o Tipo de Dado (TD) é um campo único, reconhecido pelo usuário e não repetido. Em termos práticos, pode-se considerar um TD como um campo de arquivo, embora essa relação não seja perfeita, como mostra Figura 12.

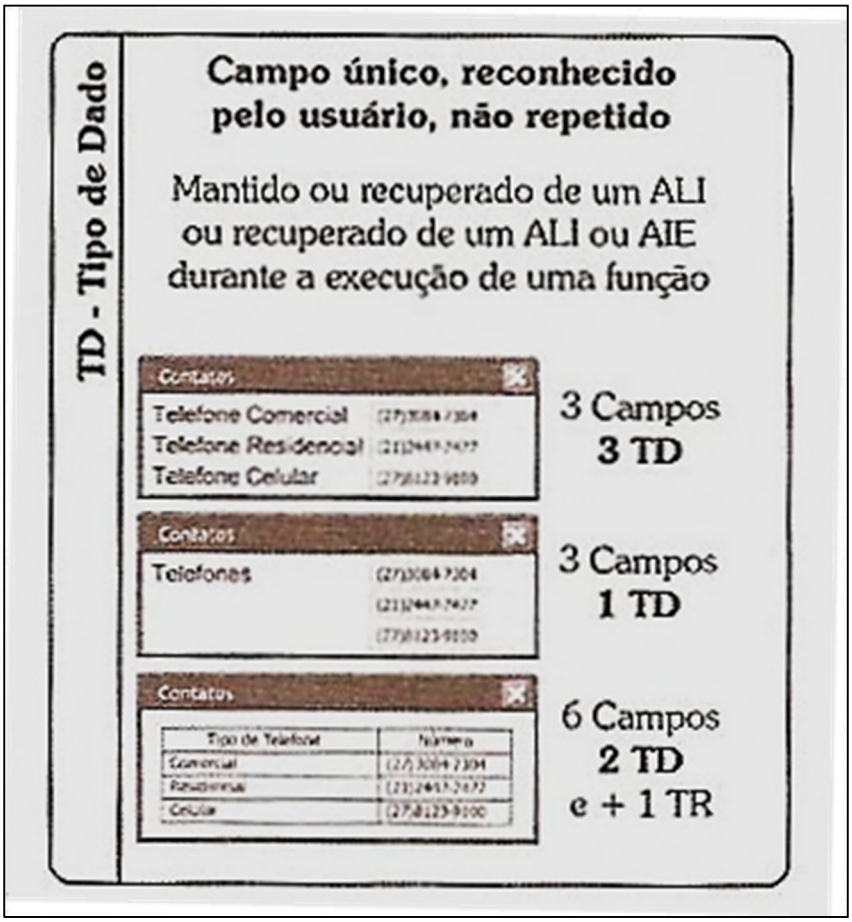


Figura 12 - Diferença entre tipo de dado e campo de um arquivo.
Fonte: Vazquez (2012).

Esse termo de campo não repetido é realmente levado ao “pé da letra”. O autor acima citado ainda explica que caso uma suposta aplicação mantém ou referencia os seguintes campos: NOME e CPF e outra aplicação mantém ou referencia os seguintes campos do mesmo arquivo: NOME, CEP, CIDADE, ESTADO, a primeira aplicação deverá ser contados dois tipos de dados referentes ao arquivo e a segunda aplicação

devem ser contados três tipos de dados para o mesmo arquivo, não considerando o campo NOME novamente.

Outro exemplo que Vazquez (2012) apresenta é em cima de gráficos. Um do tipo pizza conforme ilustra a Figura 13, que é contado três tipos de dados, sendo eles, um referente ao nome do produto, o outro ao valor faturado e o terceiro em “%” de participação do faturamento.

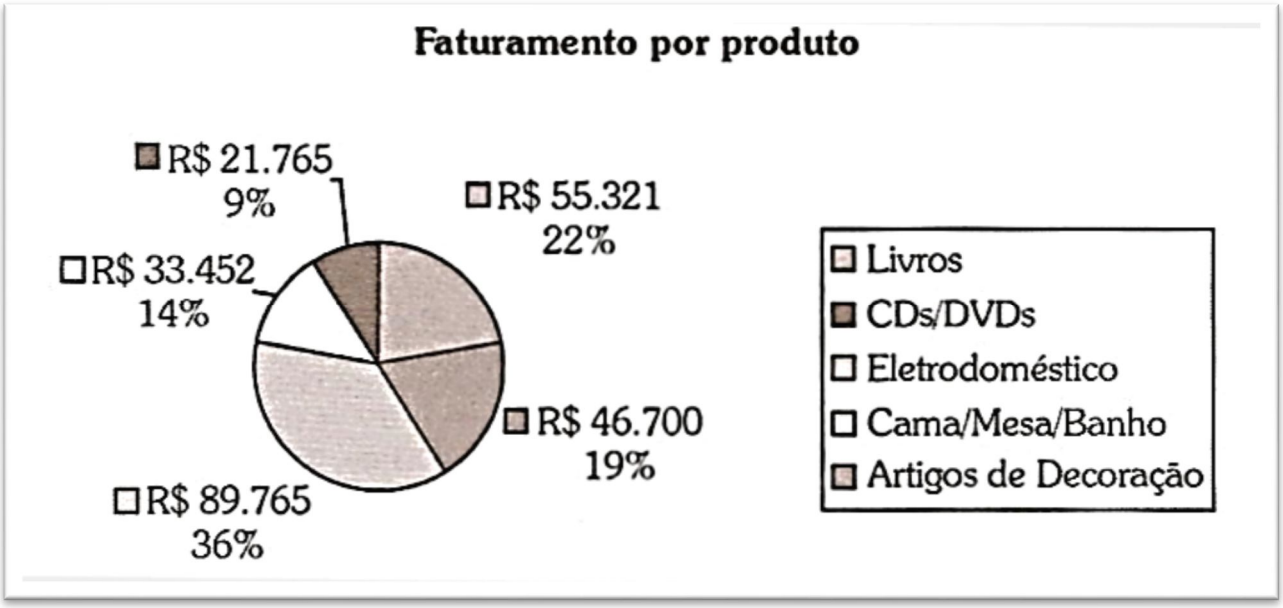


Figura 13 - Exemplo de gráfico com três tipos de dados.
Fonte: Vazquez (2012).

9.6.3 Tipo de Registro (TR)

Um Tipo de Registro (TR) é um subgrupo de dados, reconhecido pelo usuário e componente dos ALIs e dos AIEs (VAZQUEZ, 2012).

O autor acima descrito também explica que há dois tipos de subgrupo:

- Opcionais: são aqueles em que o usuário tem a opção de não informar no processo elementar que cria ou adiciona dados;
- Obrigatórios: são aqueles que o usuário requer que sejam sempre utilizados pelo processo elementar que cria ou adiciona dados.

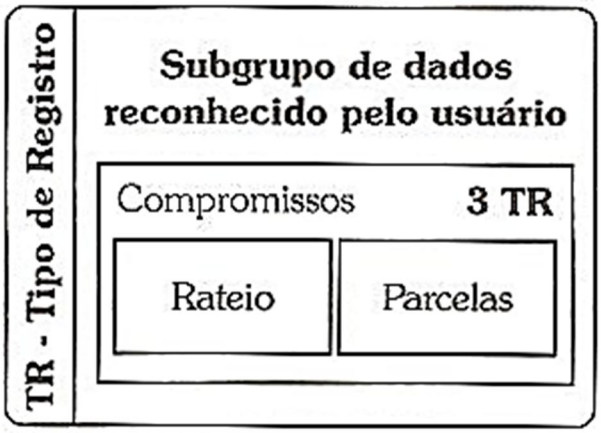


Figura 14 - Exemplo de tipos de registros.
Fonte: Vazquez (2012).

Macoratti (c2010) cita que há regras que devem ser aplicadas para contagem das TRs:

- Conte um registro lógico (ou Tipo de Registro) para cada subgrupo identificado, opcional ou obrigatório, conforme mostra a Figura 14.
- Considerar apenas um registro lógico, caso o ALI não possuir subgrupos.

9.6.4 Determinação da Contribuição

Após a determinação da complexidade dos arquivos mostrada na seção 9.3.1 deste trabalho, deve-se calcular sua contribuição, utilizando a tabela ilustrada na Figura 15.

Tipo de Função	Baixa	Média	Alta
Arquivo Lógico Interno	7 PF	10 PF	15 PF
Arquivo de *Interface* Externa	5 PF	7 PF	10 PF

Figura 15 - Contribuição dos pontos de função das funções do tipo dado.
Fonte: Vazquez (2012).

Como se pode perceber, caso um ALI tenha uma complexidade baixa, ela irá contribuir com 7 pontos de função. Já se um AIE tenha uma complexidade alta, irá contribuir com seus 10 pontos de função.

9.7 FUNÇÕES DE TRANSAÇÃO

Vazquez (2012) explica que as funções de transação representam a funcionalidade fornecida ao usuário para atender às suas necessidades de processamento de dados pela aplicação. São classificados como Entrada Externa, Saída Externa e Consulta Externa. Pressman (2011) apresenta as classificações e descrições das mesmas:

- **Entrada Externa (EE):** Também conhecido como (*external inputs* – EIs), cada entrada externa é originada de um usuário ou transmitida de outra aplicação e fornece dados distintos orientados a aplicação ou informações de controle. As entradas são muitas vezes usadas para atualizar arquivos lógicos internos (ALIs).

Vazquez (2012) apresenta alguns exemplos de EE: incluir cliente, alterar cliente e excluir cliente. Processamento em lotes de atualização de bases cadastrais a partir de arquivos de movimento.

As telas de *login*, cuja principal intenção é dizer se o usuário tem ou não acesso não é considerado Entrada Externa, assim como as telas de filtros de relatórios.

Macoratti (c2010) afirma que a lógica de processamento da EE é visto como um conjunto de críticas, cálculos, algoritmos e referências de ALI ou AIE, requeridos pelo usuário para realizar um processo elementar.

- **Saída Externa (SE):** Também conhecido como (*external outputs* – EOs), cada saída externa é formado por dados derivados da aplicação e fornece as informações para o usuário. Pode-se exemplificar que saídas externas se referem a relatórios, telas, mensagens de erros etc.

Vazquez (2012) destaca que consultas e relatórios sem nenhum totalizador, que não atualizam arquivos e que não tenham dados derivados ou que não modificam o comportamento do sistema não é considerado um exemplo de SE.

IFPUG (1999 citado por MACORATTI, c2010) afirma que a lógica de processamento é definida como o conjunto de críticas, cálculos, algoritmos e referência a arquivos requisitados pelo usuário que visa completar um processo elementar.

- **Consulta Externa (CE):** Também conhecido como (*External Inquiries* – EQs), é definida como uma entrada *online* que resulta na geração de alguma resposta imediata do *software* na forma de uma saída *online*, onde muitas vezes é obtida de um Arquivo Lógico Interno.

Vazquez (2012) cita alguns exemplos de Consultas Externas, tais como: informações em formato gráfico, consulta de cadastro de clientes, telas de *login* (sem criptografia).

Macoratti (c2010) afirma que a lógica de processamento da CE não deve conter fórmula matemática ou cálculo nem criar dados derivados ou atualizar nenhum ALI.

9.7.1 Processo Elementar

Vazquez (2012) conceitua que a EE, SE e CE tem como característica comum a de ser um processo elementar. O autor explica que um processo elementar é a menor unidade de atividade que satisfaz todas as seguintes regras:

- Tem significado para o usuário.
- Constitui uma transação completa.
- É autocontido.
- Deixa o negócio da aplicação sendo contada em um estado consistente.

O autor acima descrito explica que o conceito de processo elementar é o mais importante para a medição das funções de transação. Se por um lado ele evita que vários processos elementares sejam contados como um único, por outro lado impede que subprocessos componentes de processos elementares também sejam contados. Levando para a parte prática, a validação de CPF não pode ser considerada um processo elementar. De forma análoga, uma tela de cadastro que permite incluir, alterar e excluir deve ser contada como três processos elementares.

9.7.2 Lógica de Processamento

Pode-se perceber que em todas as funções de transações falou-se em lógica de processamento.

Vazquez (2012) explica que a lógica de processamento é visto como qualquer dos seguintes requisitos especificadamente solicitados pelo usuário para completar um processo elementar.

A Figura 16 exemplifica um resumo de lógicas de processamento que podem ser executadas por cada tipo de função de transação. Como pode ser observada, a diferença primordial entre elas está na sua principal intenção, onde é necessário que seja executada determina lógica de processamento para que sua principal intenção seja alcançada.

Tipo de Lógica de Processamento	EE	SE	CE
1. Realização de validações. Exemplo: ao adicionar um novo cliente, deve-se validar se as informações de CNPJ ou CPF estão corretas.	Pode	Pode	Pode
2. Realização de cálculos e fórmulas matemáticas. Exemplo: ao listar todos os clientes, calcula-se o número total de clientes pessoa física, pessoa jurídica e total geral.	Pode	Deve*	Não
3. Conversão de equivalência entre montantes. Exemplo: uma transação referencia taxas de conversão de reais em outras moedas. Isso é feito pela recuperação de valores de tabelas, de maneira que não há necessidade de cálculos.	Pode	Pode	Pode
4. Dados são filtrados e selecionados utilizando determinados critérios para comparar múltiplos conjuntos de dados. Exemplo: para listar clientes com parcelas que vencem no mês, a transação compara as datas de vencimento das parcelas com o final do mês para selecionar e listar as parcelas apropriadas.	Pode	Pode	Pode
5. Condições são analisadas para determinação de qual se aplica. Exemplo: para conceder um crédito, a transação analisa se o valor solicitado é inferior a 20% da renda declarada pelo cliente.	Pode	Pode	Pode
6. Um ou mais ALI são atualizados. Exemplo: ao adicionar um cliente, o processo elementar atualiza o ALI de clientes para manter os seus dados.	Deve*	Deve*	Não
7. Um ou mais ALI ou AIE são referenciados. Exemplo: ao incluir um cliente e informar seu CEP, o AIE de CEPs é referenciado para buscar o endereço completo.	Pode	Pode	Deve
8. Dados ou informações de controle são recuperados. Exemplo: para poder visualizar uma lista de clientes, as informações de clientes são recuperadas do sistema.	Pode	Pode	Deve
9. Dados derivados são criados pela transformação dos dados existentes em novos dados. Exemplo: após a inclusão de um usuário, a identificação de login é gerada automaticamente pelo sistema por meio da concatenação de algumas letras do seu nome e sobrenome.	Pode	Deve*	Não
10. O comportamento do sistema é alterado. Exemplo: ao mudar o parâmetro da quantidade de dias que se armazena a lista de sites visitados, o browser muda de comportamento.	Deve*	Deve*	Não
11. Preparar e apresentar informação para fora da fronteira da aplicação. Exemplo: uma lista de clientes é apresentada ao usuário.	Pode	Deve	Deve
12. Capacidade de aceitar dados ou informação de controle que entram na fronteira da aplicação. Exemplo: o usuário entra com várias informações para incluir um agendamento de recebimento.	Deve	Pode	Pode
13. Dados são ordenados ou organizados. Exemplo: o usuário solicita uma lista de clientes ordenados alfabeticamente pelo sobrenome.	Pode	Pode	Pode
* Ao menos uma das lógicas de processamento deve estar presente.			

Figura 16 - Resumo das lógicas de processamento.
Fonte: Vazquez (2012).

9.7.3 Determinar a Complexidade

Segundo Macoratti (c2010), a complexidade funcional da Entrada Externa, Saída Externa e Consulta Externa é baseada no número de Arquivos Referenciados (AR) e no número de Itens de Dados (ID).

Vazquez (2012) trata a denominação de número de Itens de Dados como apenas Tipos de Dados (TD) e apresenta as tabelas de complexidade, ilustrada pelas Figuras 17 e 18, após a determinação da quantidade de arquivos referenciados e de tipos de dados.

		Tipos de Dados (TDs)		
		< 5	**5 – 15**	**> 15**
Arquivos Referenciados (ARs)	**< 2**	Baixa	Baixa	Média
	2	Baixa	Média	Alta
	> 2	Média	Alta	Alta

Figura 17 - Complexidade para entradas externas (EEs).
Fonte: Vazquez (2012).

		Tipos de Dados (TDs)		
		< 6	**6 – 19**	**> 19**
Arquivos Referenciados (ARs)	**< 2**	Baixa	Baixa	Média
	2 – 3	Baixa	Média	Alta
	> 3	Média	Alta	Alta

Figura 18 - Complexidade para saídas externas (SEs) e consultas externas (CEs).
Fonte: Vazquez (2012).

Como se pode perceber, uma EE com 17 tipos de dados e 3 arquivos referenciados é de complexidade alta. Uma SE com 6 tipos de dados e 2 arquivos referenciados é de complexidade média. Por sua vez, uma CE com 2 tipos de dados e 3 arquivos referenciados é de complexidade baixa.

9.7.4 Arquivo Referenciado (AR)

IFPUG (1999 citado por MACORATTI, c2010) define que um arquivo referenciado é qualquer ALI que foi consultado ou atualizado pelo processo ou qualquer AIE que foi consultado.

Vazquez (2012) afirma que as seguintes regras são válidas para a contagem de um arquivo referenciado, sendo que as duas primeiras, tratam de atualização de arquivos, ou seja, não são aplicáveis para Consultas Externas.

- Conte um AR para cada ALI mantido.
- Conte apenas um AR para cada ALI que seja tanto mantido como lido (Ex: caso for um “Incluir Cliente” e precisa verificar se há alguma irregularidade. Logo, o ALI deve ser lido antes de ser atualizado, porém conta-se apenas uma vez o AR).
- Contar um arquivo referenciado para cada ALI ou AIE lido durante o processamento.

Tomando como este útimo exemplo: numa tela com o botão de incluir um cliente, um arquivo lógico é lido para verificar há alguma irregularidade com o mesmo. Ao selecionar, por exemplo, o funcionário que fez a venda para este cliente, o arquivo lógico é lido para verificar se o funcionário está ativo ou demitido (VAZQUEZ, 2012).

9.7.5 Determinação da Contribuição

Após a determinação da complexidade das funções do tipo transação mostrada na seção 9.7.4 deste trabalho, deve-se calcular sua contribuição, utilizando a tabela ilustrada na Figura 19.

Tipo de Função	Baixa	Média	Alta
Entrada Externa	3 PF	4 PF	6 PF
Saída Externa	4 PF	5 PF	7 PF
Consulta Externa	3 PF	4 PF	6 PF

Figura 19 - Contribuição dos pontos de função das funções do tipo transação.
Fonte: Vazquez (2012).

Como se pode perceber, caso um EE tenha uma complexidade baixa, ele irá contribuir com 3 pontos de função. Um SE de complexidade baixa contribui com 4 pontos de função. Já se um CE que tenha uma complexidade também baixa, irá contribuir com seus 3 pontos de função.

9.8 CÁLCULO DO TAMANHO FUNCIONAL

Cada função do tipo dado e transação possui um peso em PF, que é determinado pela complexidade da função, podendo ser baixa, média ou alta.

Conforme mencionado em capítulos anteriores, a complexidade das funções do tipo dado é determinada por dois parâmetros, sendo quantidade de tipos de dados e quantidade de tipos de registros. Já as funções do tipo transação têm sua complexidade determinado por outros dois parâmetros: quantidade de tipos de dados e quantidade de arquivos referenciados. Após todas essas identificações e classificações, o numero de pontos de função será a soma do peso de cada uma dessas funções.

Uma vez coletados esses dados, é completada a tabela da Figura 20, e associado a um valor da complexidade com cada contagem (PRESSMAN, 2011).

Descrição do	Complexidade		
Tipo Funcional	Simples	Média	Complexa
Arquivo Lógico Interno (ALI)	7 PFs	10 PFs	15 PFs
Arquivo de Interface Externa (AIE)	5 PFs	7 PFs	10 PFs
Entrada Externa (EE)	3 PFs	4 PFs	6 PFs
Saída Externa (SE)	4 PFs	5 PFs	7 PFs
Consulta Externa (CE)	3 PFs	4 PFs	**6 PFs**

Figura 20 – Complexidade dos tipos funcionais.
Fonte: Hazan (2008).

Este passo de contagem de função envolve os três tipos de contagem mensurada na seção 9.3 deste trabalho: projeto de novo desenvolvimento, projeto de melhoria e aplicação.

9.8.1 Cálculo de Pontos de Função não Ajustados ou Brutos

IFPUG (1999 citado por MACORATTI, c2010) afirma que após definir a fronteira da aplicação, o tipo de contagem e reconhecer as funções de dados e de transação pode-se calcular os pontos de função não ajustados, ou brutos, multiplicando-se o total de ALI, AIE, EE, SE, e CE pela respectiva complexidade de cada uma, fazendo uma somatória total.

9.8.2 Projeto de Desenvolvimento

Fórmula para o cálculo (VAZQUEZ, 2012):

$$DFP = (ADD + CFP)$$

Sendo que:

- DFP: tamanho do projeto de desenvolvimento.
- ADD: tamanho das funções entregues.
- CFP: tamanho das funções de conversão.

Para simplificar e exemplificar a fórmula, Vazquez (2012) apresenta uma tabela ilustrada pela Figura 21, onde mostra um sistema de controle de ponto bem simplificado onde todas as funções identificadas estão listadas na tabela, assim como as de conversão de dados.

Nome da Função	Tipo[1]	TD[2]	AR/TR	Complexidade	Contribuição
Pessoa (do Controle de Segurança)	AIE	3	1	Baixa	5
LOGIN	CE	4	1	Baixa	3
APONTAMENTO	ALI	4	1	Baixa	7
Apontamento - Importação (conversão de dados)	EE	4	1	Baixa	3
Registro de Ponto	EE	4	2	Baixa	3
Justificativa	ALI	3	1	Baixa	7
Justificativa - Importação (conversão de dados)	EE	3	1	Baixa	3
Justificativa - Seleção	CE	6	2	Média	4
Justificativa - Inclusão	EE	5	2	Média	4
Justificativa - Alteração	EE	5	2	Média	4
Justificativa - Exclusão	EE	3	1	Baixa	3
Justificativa - Consulta	CE	5	2	Baixa	3
Horário Padrão	ALI	3	1	Baixa	7
Horário Padrão - Importação (conversão de dados)	EE	3	1	Baixa	3
Horário Padrão - Inclusão	EE	4	2	Baixa	3
Horário Padrão - Alteração	EE	4	2	Baixa	3
Horário Padrão - Consulta	CE	4	2	Baixa	3
Relatório de Horas	SE	10	4	Alta	7
Relatório de Justificativas	SE	8	4	Alta	7
[1] Total de Pontos de Função					82

[1] Tipo: ALI, AIE, EE, SE e CE
[2] TD: Número de tipos de dados
[3] AR/TR: Número de arquivos referenciados/tipos de registro

Figura 21 - Exemplo de contagem de pontos de função de um projeto em desenvolvimento.
Fonte: Vazquez (2012).

Aplicando a formula de projeto neste caso, tem-se:

DFP = ADD + CFP

DFP = 73 + 9

DFP = 82 pontos de função.

Lembrando que caso for fazer também com os fatores de ajuste de valor (*value adjustment factors* – VAF), multiplica-se o resultado de DFP com o valor resultante de VAF.

O VAF será explicado e detalhado na seção 9.9 deste trabalho.

9.8.3 Projeto de Melhoria

Não é necessário saber o número de pontos de função da aplicação para determinar o tamanho do projeto de melhoria. O intuito é medir apenas funções que serão afetadas pela manutenção. Porém, caso a contagem da aplicação já esteja disponível, a medição do projeto de melhoria irá ficar mais fácil, pois para funções alteradas, basta apenas trazê-las da contagem do sistema, ajustado uma eventual mudança. Para as funções de exclusões fica mais fácil ainda, onde do mesmo jeito que foram contadas na aplicação, serão contadas na melhoria.

9.8.3.1 Critérios para Avaliar Mudança em Funções de Dados

Segundo Vazquez (2012), a diretriz básica para considerar que uma função do tipo dado foi alterada é que ela seja modificada em sua estrutura, ou seja, campos devem ser acrescentados, excluídos ou terem algum atributo alterado para atender a um requisito de negócio. Neste caso deve-se contar a funcionalidade toda no projeto de melhoria, não somente os campos afetados.

O autor acima mencionado apresenta alguns procedimentos corretos em diversas situações:

- Se a mudança envolve apenas a alteração dos dados armazenados em um arquivo, não pode considerar que o arquivo foi alterado.
- Se um campo é adicionado, alterado ou excluído de um ALI ou AIE que pertencem a várias aplicações, onde as mesmas mantêm o campo, essa alteração é contada para cada uma das aplicações.
- Se a aplicação passa a manter ou referencia um campo já existente onde antes não era utilizado, considera-se uma alteração.

- Caso um campo for adicionado e não é mantido ou referenciado na aplicação, então não é considerada uma alteração.

9.8.3.2 Critérios para Avaliar Mudança em Funções de Transação

Vazquez (2005 citado por MACORATTI, c2010) afirma que uma função do tipo transação é considerada alterada quando há alteração em um dos itens a seguir:

- Tipos de dados – Se houve inclusão, alteração ou exclusão da função.
- Arquivos referenciados – Se foram incluídos, excluídos ou alterados da função.
- Lógica de processamento – Se qualquer lógica for incluída, alterada ou excluída.

Vale lembrar e alertar que as considerações feitas para as funções de dados vale também para as funções de transação (VAZQUEZ, 2012).

9.8.3.3 Fórmula do Projeto de Melhoria

Fórmula para o cálculo (VAZQUEZ, 2012):

$$EFP = (ADD + CHGA + CEP + DEL)$$

Sendo que:

- EFP: número de pontos de função do projeto de melhoria.
- ADD: tamanho das funções incluídas pelo projeto de melhoria.
- CHGA: tamanho das funções modificadas. Reflete as funções depois das modificações.
- CFP: tamanho das funções de conversão.
- DEL: tamanho das funções excluídas pelo projeto de melhoria.

Fórmula para o cálculo com o VAF (MACORATTI, c2010):

$$EFP = [(ADD + CHGA + CFP) * VAFA] + (DEL * VAFB)$$

Onde:

- EFP – Número de pontos de função do projeto de melhoria.
- ADD – Número de pontos de função não ajustados das funções incluídas pelo projeto de melhoria.
- CHGA – Número de pontos de função não ajustados das funções modificadas depois das modificações.
- CFP - Número de pontos de função não ajustados adicionados pela conversão.
- VAFA – Fatores de ajuste de valor da aplicação depois do projeto de melhoria.
- DEL - Número de pontos de função não ajustados das funções excluídas pelo projeto de melhoria;
- VAFB – Fatores de ajuste de valor da aplicação antes do projeto de melhoria.

Utilizando o mesmo exemplo da Figura 21 da seção 9.8.2 deste trabalho, Vazquez (2012) dá um exemplo com o cálculo de projeto de melhoria através da Figura 22. Não haverá funções de conversão de dados e as alterações serão as seguintes:

1. Funções adicionadas: AIE – Senha e SE – Relatório de Ponto
2. Funções alteradas: CE – *Login*
3. Funções excluídas: SE – Relatório de Horas e SE – Relatório de Justificativas.

Descrição da Função	Tipo[1]	Depois do Projeto de Melhoria				Antes do Projeto de Melhoria			
		TD[2]	AR/TR[3]	Complexidade	PF	TD[2]	AR/TR[3]	Complexidade	PF
Senha	AIE	2	1	Baixa	5				
Relatório de Ponto	SE	12	4	Alta	7				
Login	CE	4	1	Baixa	3	4	2	Baixa	3
Relatório de Justificativas	SE					8	4	Alta	7
Relatório de Horas	SE					10	4	Alta	7

[1]Tipo: EE, EI, CE, ALI e AIE
[2]TD: Número de tipos de dados
[3]AR/TR: Número de arquivos referenciados/tipos de registro

Figura 22 - Exemplo de contagem de pontos de função do projeto de melhoria
Fonte: Vazquez (2012).

Aplicando a fórmula para determinação do tamanho do projeto de melhoria, têm-se:

EFP = (ADD + CHGA + CFP + DEL)

EFP = (12 + 3 + 0 + 14)

EFP = 29 pontos de função.

Para complementar, o IFPUG (2010 citado por VAZQUEZ, 2012) afirma que, para o conceito de tamanho do projeto de melhoria, as funções excluídas, alteradas ou adicionas contribuem com a mesma proporção para o tamanho. Por exemplo: se o projeto criar um relatório que vale 6PF, contam-se 6 PF no projeto de melhoria. Se o projeto adicionar um campo no relatório que vale 6 PF, contam-se 6 PF para o projeto de melhoria. O mesmo aconteceria também com a exclusão.

Isso porque, a IFPUG mede se a função foi alterada ou não. Não importa o quanto ela tenha sido alterada.

9.8.4 Aplicação

Vazquez (2005 citado por MACORATTI, c2010) explica que existem duas fórmulas para se calcular pontos de função de uma aplicação:

1. Fórmula para Contagem Inicial: representa todas as funcionalidades requeridas pelo usuário de uma aplicação instalada. As funções da conversão de dados não devem ser computadas no tamanho da aplicação, pois elas existirão somente para o processo de implantação do aplicativo.

AFP = ADD * VAF

Ou, caso não contar os fatores de ajuste de valor:

AFP = ADD (VAZQUEZ, 2012).

Onde:

- AFP – tamanho da aplicação;
- ADD – tamanho das funções entregue;
- VAF - fatores de ajustes de valor.

Verificando a função, dá para observar que como não há conversão de dados, o tamanho do projeto de desenvolvimento será igual ao da aplicação.

Utilizando o mesmo exemplo da tabela ilustrada pela Figura 21 da seção 9.8.2 deste trabalho, resultaria em:

AFP = ADD = 73 pontos de função

2. Fórmula Após o Projeto de Melhoria: após a conclusão de um projeto de melhoria os pontos de função devem ser atualizados para refletir as mudanças na aplicação. Novamente as funções de conversão de dados não devem ser computadas.

AFP = [(UFPB + ADD + CHGA) – (CHGB + DEL)] * VAFA

Ou, caso não contar o valor do fator de ajuste:

AFPA = (AFPB + ADD + CHGA) – (CHGB + DEL) (VAZQUEZ, 2012).

- AFPA – tamanho da aplicação após a melhoria.
- AFPB – tamanho da aplicação antes da melhoria.

- ADD – tamanho das funções incluídas pelo projeto de melhoria.
- CHGA – tamanho das funções alteradas pelo projeto de melhoria depois do seu término.
- CHGB – tamanho das funções alteradas pelo projeto de melhoria antes do seu término.
- DEL – tamanho das funções excluídas pelo projeto de melhoria.

Vale lembrar que quando um projeto de melhoria é concluído, o número de pontos de função da aplicação deve ser atualizado para refletir tais modificações. (VAZQUEZ, 2012).

Aplicando a fórmula à aplicação alterada pelo projeto de melhoria inerente a FIGURA TAL da seção 9.8.3.3, têm-se:

$$AFPA = (AFPB + ADD + CHGA) – (CHGB + DEL)$$

$$AFPA = (73 + 12 + 3) – (3 + 14)$$

$$AFPA = 71 \text{ pontos de função}$$

9.9 FATOR DO AJUSTE

O IFPUG tornou-se o fator de ajuste opcional na aplicação da técnica de análise de pontos de função. O propósito do fator de ajuste é medir requisitos gerais da aplicação (não funcionais), ajustando os pontos de função em mais ou menos 35% de acordo com a influência de 14 características gerais (VAZQUEZ, 2012).

LONGSTREET (2002 citado por PRESSMAN, 2011) apresenta as 14 questões que se levam em conta os fatores de ajustes de valor (FAV), ou *value adjustment factors (VAF).*

- O sistema requer salvamento e recuperação confiável?
- São necessárias comunicações de dados especializadas para transferir informações para aplicação ou da aplicação?
- Há funções de processamento distribuído?
- O desempenho é crítico?

- O sistema rodará em um ambiente operacional existente e intensamente utilizável?
- O sistema requer entradas de dados online?
- A entrada online de dados requer que a transação de entrada seja composta em múltiplas telas ou operações?
- Os ALIs são atualizados online?
- As entradas, saídas, arquivos ou consultas são complexas?
- O processamento interno é complexo?
- O código é projetado para ser reutilizável?
- A conversão e instalação estão incluídas no projeto?
- O sistema é projetado para múltiplas instalações em diferentes organizações?
- A aplicação é projetada para facilitar a troca e o uso pelo usuário?

Vazquez (2012) completa afirmando que se deve responder a essas perguntas por meio de uma escala que varia de 0 a 5:

- Nenhuma Influência
- Influência Mínima
- Influência Moderada
- Influência Média
- Influência Significativa
- Grande Influência

Após apurar-se o nível de influência de cada pergunta, Dekkers (1998) determina o Fator de Ajuste de Valor (FAV) baseado na equação:

(FAV = 0,65 + (Soma das Características Gerais do Sistema x 0,01).

Segundo Macoratti (c2010), se o FAV é igual a 1, a influência total das características gerais do sistema é neutra. Nesta situação, a contagem dos pontos de função ajustados equivale à contagem de pontos de função não ajustados.

9.10 DOCUMENTAR E REPORTAR

Vazquez (2012) explica que o nível de detalhamento da documentação tem haver com o propósito da contagem e o nível deve estar previamente acordado entre as partes interessadas na medição, ponderando-se os custos e benefícios envolvidos. Claro que um nível de documentação alto implica mais tempo e custo, porem, o autor cita algumas facilidades de uma documentação bem feita, tais como: Auditoria da medição, rastrear as funções identificadas até os artefatos do projeto usados na medição, usar os resultados da medição e manter e evoluir a medição.

Documentar e reportar os resultados da medição é o processo final.

10 METODOLOGIA

Este projeto inicialmente é uma pesquisa exploratória, pois visa estudar a Análise de Pontos de Função, na construção de um protótipo específico, no caso, um *software* para estimativa.

10.1 PROCEDIMENTO E *SOFTWARES* UTILIZADOS

O projeto foi desenvolvido em um computador Intel® Core™ i3, 3.07 GHz 3.06GHz, 4GB de memória RAM, disco rígido de 1TB.

Considerando a natureza complexa do assunto e do sistema, primeiramente foi necessário estudar e adquirir um bom conhecimento em engenharia de *software* e aprofundar-se muito no tema de métricas de *software*, juntamente com a análise de pontos de função. Apenas com isso foi possível a construção de um sistema especialista em APF que faça estimativa para projetos.

10.1.1 Modelagem UML

Em primeiro lugar foi feita a modelagem do sistema utilizando a linguagem UML (*Unified Modeling Language*).

A UML é uma linguagem (notação com semântica associada) para visualizar, especificar, construir e documentar os artefatos de um sistema.

Foi modelado o Diagrama de Classes e Diagrama de Casos de Uso com suas devidas descrições.

O Diagrama de Classe é muito útil para o desenvolvimento do sistema, pois representa a estrutura e relações das classes que serão utilizadas, sendo possível detalha-las com seus relacionamentos, definindo também os atributos e as operações.

O Diagrama de Casos de Uso é uma excelente técnica para captura dos requisitos funcionais de um sistema, e pode ser usado como base para estimativas, o que condiz muito com o tema deste projeto.

A ferramenta utilizada para a modelagem dos casos de uso e suas devidas descrições foi o JUDE, pois além de ser gratuito, possui integração com o JAVA, é baseado nos diagramas e na notação da UML e é um *software* de fácil utilização.

10.1.2 Modelagem de Dados

O segundo passo foi a modelagem do banco de dados e a geração de *scripts*. É de suma importância o armazenamento das informações de projetos passados, visando à obtenção de um histórico dessas medições.

Obter dados baseados em estimativas não muda a importância deles. A partir de uma manutenção de uma base de dados com históricos estimados e realizados dos projetos, a organização pode avaliar a adequação de seu processo, e extrair indicadores cada vez mais próximos da realidade, sendo possível realizar com segurança e com confiabilidade o mais cedo possível (VAZQUEZ, 2012).

O Sistema de Gerenciamento do Banco de Dados (SGBD) escolhido foi o MySQL, pois é servidor rápido, de multiprocessamento e multiusuários em SQL (*Structured Query Language*), muito confiável, faz boa integração com a linguagem JAVA. E o melhor de tudo: gratuito.

A ferramenta utilizada para a modelagem dos dados foi o MySQL Workbench 5.2.46. Além de ser gratuita, a ferramenta possibilita trabalhar diretamente com objetos *schema*, além de fazer a separação do modelo lógico do catálogo de banco de dados. Toda a criação dos relacionamentos entre as tabelas pode ser baseada em chaves estrangeiras. Outro recurso que a ferramenta possibilita é realizar a engenharia reversa de esquemas do banco de dados, bem como gerar todos os *scripts* em SQL, facilitando muito o trabalho que teria que ser feito manualmente.

10.1.3 Linguagem de Programação

O protótipo deste trabalho é destinado para *desktop.* A programação do *software* foi realizada na linguagem JAVA, pois é uma linguagem orientada a objeto e gratuita, sendo caracterizada como linguagem portátil, ou seja, não depende de uma plataforma específica.

Além desses motivos, ela tem o quesito segurança, o qual permite executar programas via rede com restrições de execução.

10.2 RESULTADOS ESPERADOS DO *SOFTWARE*

Conforme descrito no capítulo 6, há muitas dificuldades em determinar a exatidão do custo final de um projeto e a data de sua conclusão. Antes disso, o que pode ser feito são estimativas que auxiliam a aproximação, e servem como indicador para próximos projetos.

O protótipo deve calcular os pontos de função, e apresentar as estimativas para o usuário utilizá-las ao longo do projeto.

10.2.1 Estimativa de Tamanho

O primeiro passo para alcançar as estimativas efetivas do projeto de *software* é estimar o seu tamanho.

Hazan (2008) afirma que para totalizar o tamanho em PFs, deve-se proceder conforme a seção 9.8.1 deste trabalho, ou seja, multiplicando-se o total de ALI, AIE, EE, SE e CE pela respectiva complexidade de cada uma, fazendo uma somatória total. Após obter este resultado, multiplica-se com o resultado dos fatores de ajustes de valor (FAV) descrito na seção 9.9 (FAV = 0,65 + (Soma das Características Gerais do Sistema x 0,01)).

Vazquez (2012) afirma que após ter vários projetos similares passados em base de dados, estima-se o tamanho do novo projeto com base em percentual do tamanho

daquele comparado. Pode-se dividir em módulos menores e estimar o percentual proporcional.

Há outras técnicas utilizadas para a estimativa de tamanho, denominadas Métodos Derivados, porém fogem do tema.

10.2.2 Estimativa de Esforço

A melhor forma de chegar à estimativa de esforço a partir da estimativa de tamanho é utilizar dados internos à própria organização, como número de horas de projetos passados, juntamente com a quantidade de pontos de função dimensionados nas diversas fases dos respectivos projetos (VAZQUEZ, 2012).

De posse da estimativa de tamanho, procede-se com a geração da estimativa de esforço. Deve-se obter, primeiramente, um índice de produtividade por meio de análise do banco de dados de histórico de projetos da organização, observando-se os atributos do projeto em questão e o esforço realizado em projetos similares.

O cálculo nada mais é do que o produto entre o valor obtido da estimativa de tamanho com o número de horas para produzir um PF. O resultado será o esforço em HH (homens_hora) (HAZAN, 2008).

10.2.3 Estimativa de Prazo

Neste terceiro passo determina-se o cronograma do projeto. Esta estimativa é uma das atividades das quais exigem mais experiências e conhecimento dos aspectos envolvidos em sua execução (VAZQUEZ, 2012).

Em conformidade com Hazan (2002), a fórmula para a estimativa de prazo consiste em:

Prazo (em dias) = Esforço (horas) / Tamanho da equipe * 6 horas.

Essa constante de 6 horas refere-se à produtividade média diária no Brasil (JONES 1997 citado por HAZAN, 2008).

O resultado é dado em dias úteis. Para transformar em meses, basta apenas dividir o resultado por 22 (média de dias úteis no mês).

10.2.3.1 Estimativa Ideal de Prazo

Caso o engenheiro de *software* não tenha ideia de quantos funcionários ele alocará no projeto, há uma fórmula onde se calcula a estimativa ideal de prazo para um projeto.

Para esta estimativa, foi aplicada a fórmula de Caper Jones (1998 citada por HAZAN, 2008), onde consiste em:

$$Td\ (meses) = V\ ^\wedge\ t$$

Onde:

- Td é o tempo ótimo de desenvolvimento, em meses.
- V é o volume em Pontos de Função
- t é um expoente que depende do ambiente computacional considerado, conforme mostrado na Figura 23.

Ambiente	**Expoente t**
Sistema Comum	0,32-0,35
Sistema Orientado a Objeto	0,36
Sistema Cliente/Servidor	0,37
Sistema Terceirizado	0,38
Sistema de Informações Gerenciais	0,39
Programa Produto Comercial	0,40
Programa de Sistema Operacional	0,41
Software Militar	0,43-0,45

Figura 23 - Índice de aproximação de estimativa de prazo de Caper Jones.
Fonte: Aguiar (2000).

O resultado dessa fórmula é dado em meses considerados “ideais” para término de um projeto com os pontos de funções calculados e deve ser comparado com a

fórmula da seção 10.2.3 deste trabalho para verificar o número ideal de colaboradores no projeto.

10.2.4 Estimativa de Custo

A estimativa de custo deve considerar o valor do salário hora da equipe alocada ao projeto, bem como outros custos de ambiente, ferramentas, deslocamentos, consultoria, etc. O ideal é ter dados históricos de custo por PF de projetos concluídos, possibilitando a derivação direta da estimativa de custo a partir da estimativa de volume em Pontos de Função (HAZAN, 2008).

Além de considerar tudo isso, Vazquez (2012) fornece uma fórmula para calcular o projeto seco, apenas com as PFs:

- Estimar a quantidade de horas necessárias para a conclusão do projeto: (tamanho do projeto (PF) * produtividade (H / PF)).
- (Quantidade de horas do calculo anterior * custo médio da hora da equipe). Esse resultado é a estimativa de custo para o projeto.

11 RESULTADOS

A modelagem de classe foi feita no modelo *Model, View, Controller* (MVC), para facilitar o desenvolvimento de *software* que condiz com esta arquitetura. As modelagens de classe e de caso de uso encontram-se no Apêndice A e B.

A modelagem de dados foi bem representada e de fácil entendimento para o leitor, utilizando de variáveis claras, conforme segue no Apêndice C.

11.1 RESULTADOS DO SOFTWARE

No *Menu* Principal, o usuário se encontrará num ambiente bem dinâmico e de fácil aprendizagem, havendo ainda um botão com um pequeno *tutorial* que auxiliará os primeiros passos de localizações do *software.* O *Menu* Principal encontra-se no Apêndice D.

Há um botão de ajuda em todas as telas do protótipo, conforme ilustram as Figuras 24 e 25.

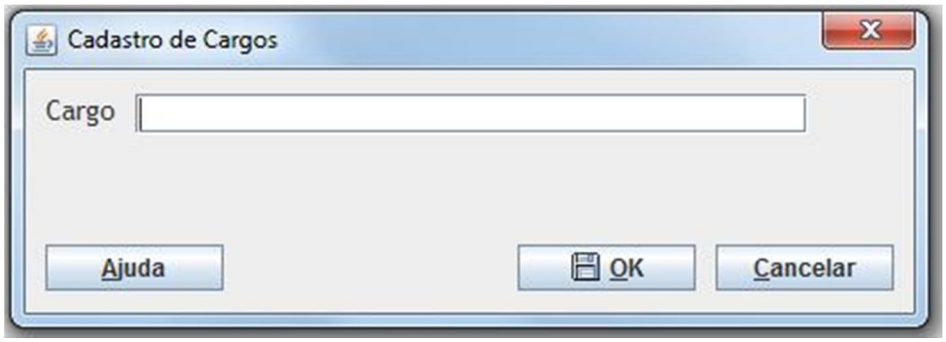


Figura 24 – Cadastro de cargos.
Fonte: Elaborado pelo autor (2013).

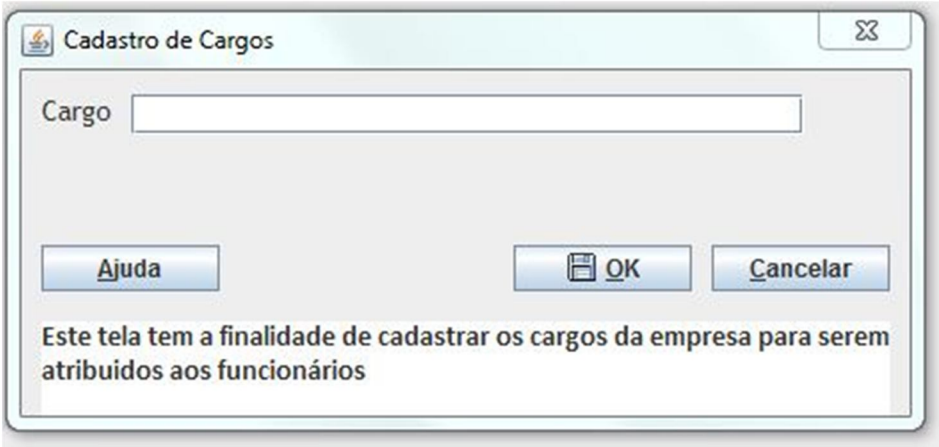


Figura 25 – Amostra do botão ajuda acionado.
Fonte: Elaborado pelo autor (2013).

Além do botão de ajuda, o usuário poderá usufruir do elemento *tooltip* em todas as caixas de textos, *combobox* ou tabelas, focalizando-os com o mouse, conforme ilustra a Figura 26.

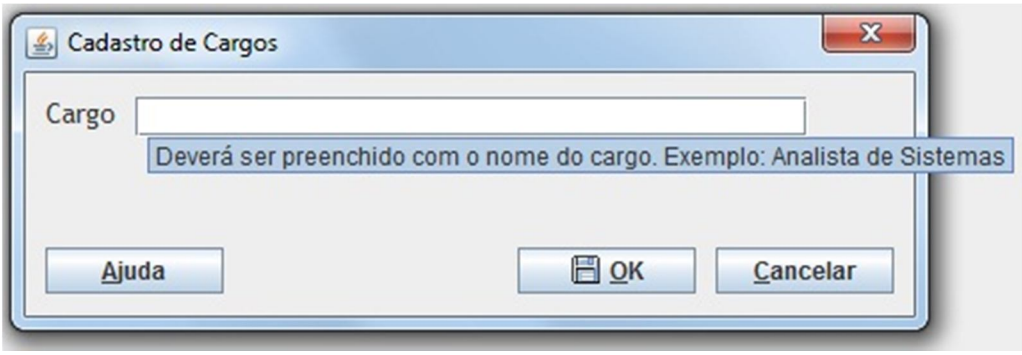


Figura 26 – Amostra do *tooltip* acionado.
Fonte: Elaborado pelo autor (2013).

Além desses benefícios ao usuário, todas as telas do *software* são padronizadas com atalhos, tabelas e *combobox* dinâmicos, no sentido de diminuir o trabalho manual do usuário e assim evitar eventuais erros.

A tela principal onde se realiza as estimativas é simples e tem o atalho das telas para o cálculo do fator de ajuste e cálculo dos pontos de função do projeto a ser estimado. A Figura 27 apresenta conforme a descrição.

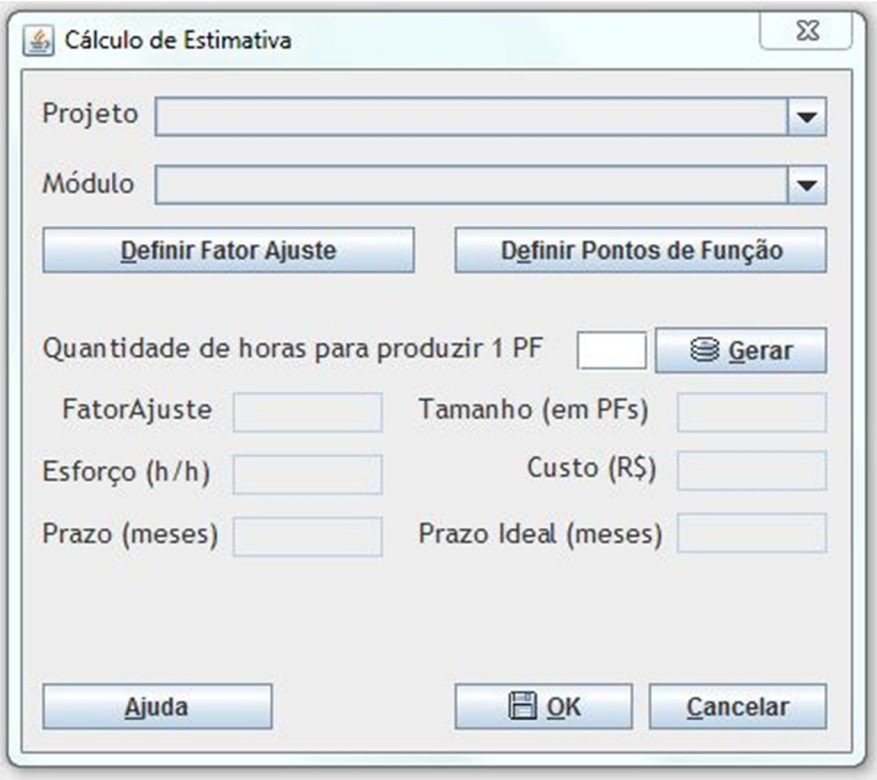


Figura 27 – Tela de cadastro de estimativa.
Fonte: Elaborado pelo autor (2013).

11.2 RESULTADOS DO PROJETO

Foram simuladas as estimativas com o mesmo *software* desenvolvido neste trabalho.

Foi cadastrado um funcionário com salário fictício de R$ 10,00 por hora. Somente ele irá trabalhar no projeto chamado TCC e módulo denominado TCC II, conforme Figura 28 e Figura 29.

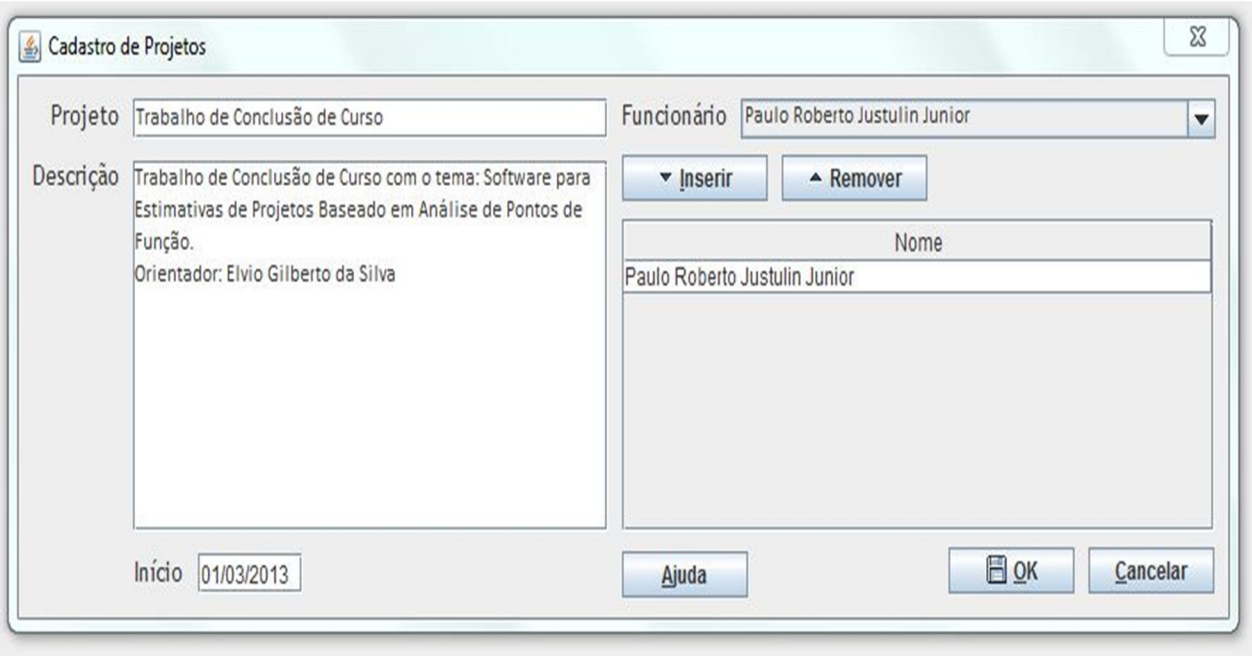


Figura 28 – Cadastro de projetos.
Fonte: Elaborado pelo autor (2013).

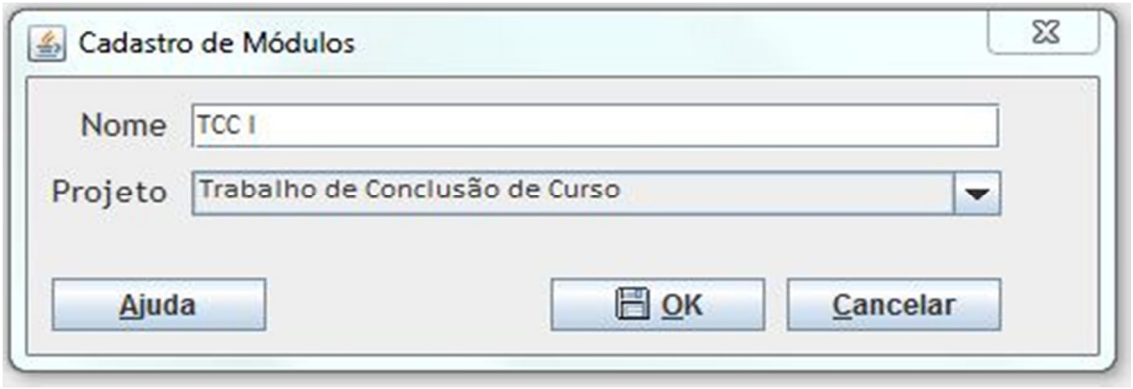


Figura 29 – Cadastro de módulos.
Fonte: Elaborado pelo autor (2013).

De posse dessas informações, já se pode iniciar as métricas. O próximo passo seria obter o valor do Fator de Ajuste de Valor, na qual se devem responder as 14 perguntas com pesos de 0 a 5, onde de acordo com Vazquez (2012):

0 = Nenhuma Influência.

1 = Influência Mínima .

2 = Influência Moderada.

3 = Influência Média.

4 = Influência Significativa.

5 = Grande Influência.

De acordo com o teste realizado, o resultado obtido do *software* é um fator de 0,86, conforme Figura 30.

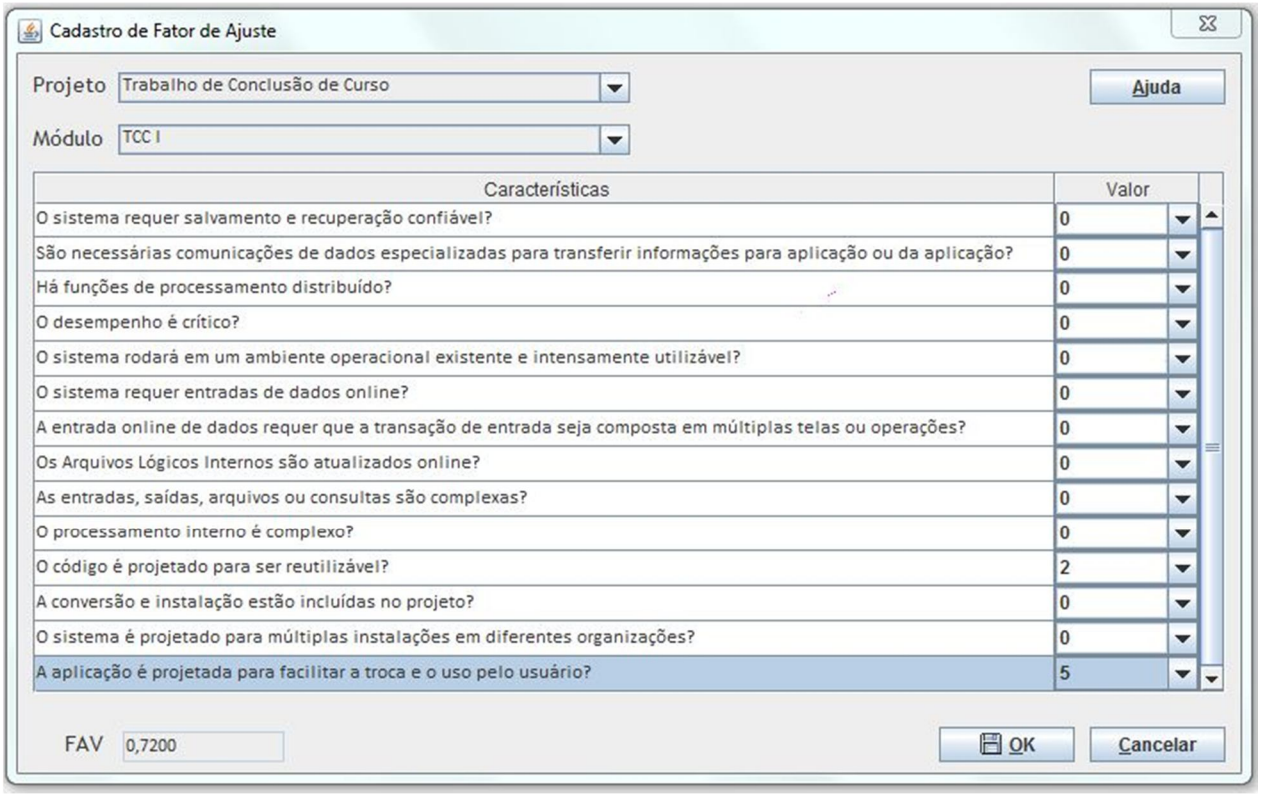


Figura 30 – Cadastro de fator de ajuste de valor
Fonte: Elaborado pelo autor (2013).

Com isso já se pode fornecer as informações para realizar o tamanho do *software* em pontos de função não ajustados, ou seja, ainda sem influência do Fator de Ajuste.

Deve ser descrito a função que será medida, selecionar a funcionalidade e preencher a quantidade de tipos de dados e tipos de registros ou arquivos referenciados.

Após clicar no botão de inclusão, o sistema fará o cálculo de complexidade e contribuição de pontos de função, fazendo um somatório geral no fim da tela, conforme Figura 31.

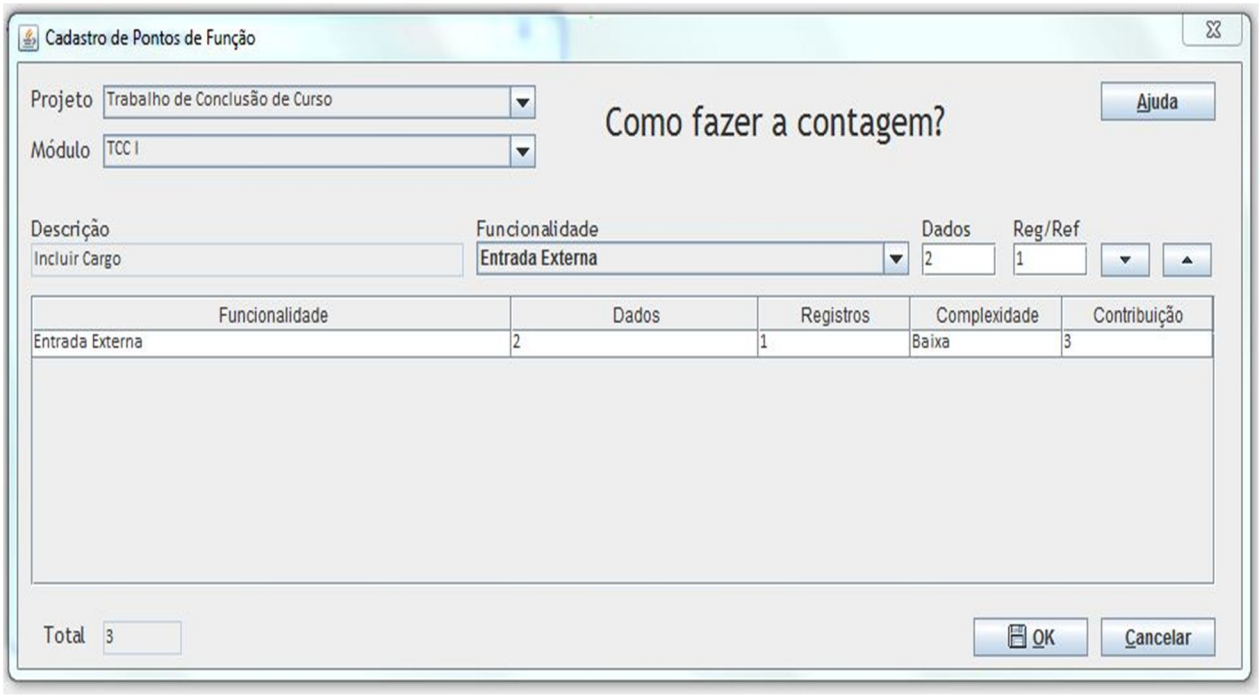


Figura 31 – Cadastro de funcionalidades
Fonte: Elaborado pelo autor (2013).

Fechando esta tela, irá voltar na tela de estimativa do sistema, onde irá realizar os cálculos a partir dos dados preenchidos do usuário.

Deve-se preencher a média de quantidade de horas que a equipe produz um ponto de função. O profissional que preencher esta tela deverá ter uma boa experiência de sua equipe para chegar numa estimativa próxima da realidade. Por isso há a grande valia de manter o histórico de projetos para se ter um indicador.

Segundo Souza (2011), a produtividade varia muito de empresa para outra empresa e conhecer a sua equipe e sua produtividade é fundamental para ter uma estimativa mais próxima da exatidão. Mesmo assim, o autor compilou algumas referências sobre produtividade por Pontos de Função (como IFPUG e outros) para ter uma base e “arriscou” alguns números, conforme Figura 32.

Figura 32 – Produtividade em horas por pontos de função.
Fonte: Souza (2011).

Como será o primeiro projeto e não há bases históricas, será utilizada a produtividade conforme a Figura 32.

Com o preenchimento de 10h para produzir um ponto de função, o *software* apresenta as estimativas ilustradas na Figura 33.

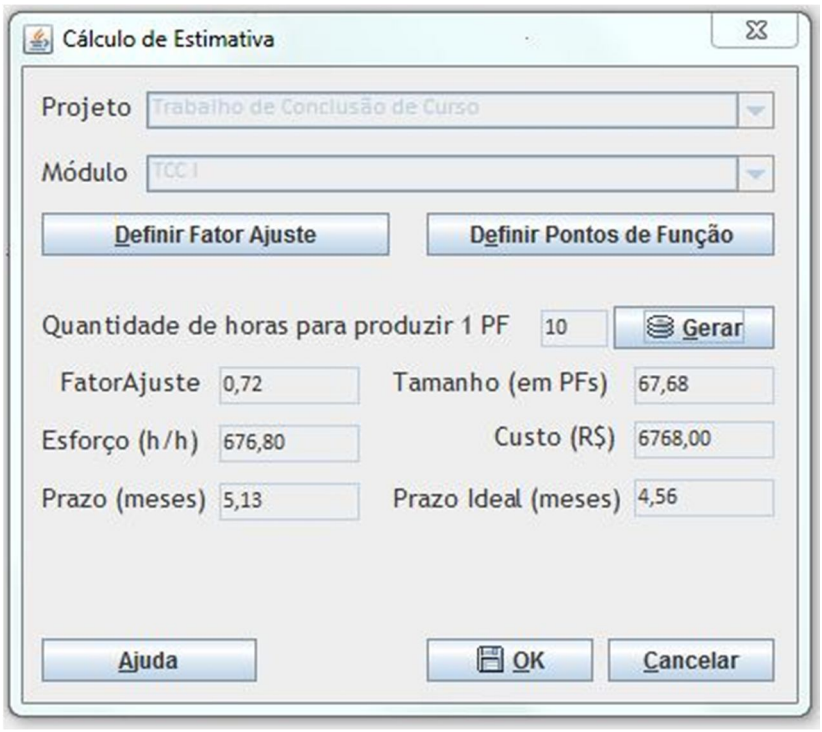


Figura 33 – Resultado da estimativa
Fonte: Elaborado pelo autor (2013).

O valor do fator de ajuste é referente ao cálculo da tela da Figura 30, onde o usuário irá preencher as 14 perguntas.

O tamanho do *software* se dá ao resultado da contagem realizado na tela da Figura 31 multiplicada pelo fator de ajuste. O valor é o tamanho em pontos de função já ajustados (HAZAN, 2008)

O cálculo do esforço se dá ao produto entre o valor obtido da estimativa de tamanho com o número de horas para produzir um PF. O resultado será o esforço em HH (homens_hora) (HAZAN, 2008).

O cálculo do custo depende de muitos fatores. Neste projeto, foi utilizado apenas o salário dos funcionários alocados. Porém, o usuário poderá levar em conta outros gastos, tais como viagens, combustíveis, alimentação, treinamento etc.

Para se chegar ao prazo, utilizou-se a seguinte formula: (Prazo (em dias) = Esforço (horas) / Tamanho da equipe * 6 horas) (HAZAN, 2002). Essa constante de 6 horas refere-se à produtividade média diária no Brasil (JONES 1997 citado por HAZAN, 2008). O resultado é dado em dias úteis. Para transformar em meses, basta apenas dividir o resultado por 22 (média de dias úteis no mês).

E, finalmente, para se chegar ao prazo ideal, utiliza-se fórmula de Caper Jones (1998 citada por HAZAN, 2008), onde consiste em: (Td (meses) = V ^ t).

Onde:

- Td é o tempo ótimo de desenvolvimento, em meses.
- V é o volume em Pontos de Função
- t é um expoente que depende do ambiente computacional considerado, conforme mostrado na Figura 23.

No caso deste projeto, o ambiente escolhido foi o “orientado a objetos”, na qual condiz com a arquitetura do *software.*

Essa fórmula apresenta quantos meses considerados “ideais” se dá ao prazo final, onde o usuário poderá comparar com o seu projeto. Em casos que a fórmula do prazo ideal for muito distante do prazo estimado, poderá haver uma análise da necessidade de alocar mais funcionários neste projeto.

12 CONSIDERAÇÕES FINAIS

A ferramenta desenvolvida cumpriu com o esperado e condiz com tudo o que foi explícito pelos autores, apresentando resultados coerentes.

O prazo foi próximo da realidade, pois o protótipo iniciou-se no final de junho/2013 (com os diagramas de classes UML) com término no fim de novembro/2013. As demais variáveis foram exatas a partir das fórmulas dos autores.

Num próximo projeto que fosse utilizar a mesma equipe, atribuía-se ainda a produtividade em 10h para produzir 1 PF e continuaria com o mesmo critério na análise de pontos de função. Em casos que a equipe se modifique, entre uma pessoa mais experiente ou menos experiente, deve-se também adequar a produtividade proporcionalmente até chegar a um número mais próximo.

Como trabalho futuro, a ferramenta poderia ser implementada com cálculos a partir de um projeto já existente, aplicando-se a fórmula do projeto de melhoria.

Poderia utilizar o aprendizado de máquina (inteligência artificial) para a estimar a produtividade de uma equipe baseando-se em projetos passados.

É de absoluta certeza que só esta ferramenta não deixará o engenheiro de *software* com qualidade e exatidão em seus projetos. Além do histórico de produtividade e experiências passadas, são de suma importância todos os fundamentos científicos da Engenharia de *Software.*

REFERÊNCIAS

ABREU, T. C. L. Métricas de Software: Como utilizá-las no gerenciamento de projetos de software. **DevMedia**, 2013. Disponível em: < http://www.devmedia.com.br/artigo-engenharia-de-software-21-metricas-de-software/15776>. Acesso em: 29 mai.2013.

AGUIAR, M. Uso de métricas na melhoria do processo de software. **BFPUG – Brazilian Function Point Users Group,** 2000. Disponível em: <http://www.bfpug.com.br/Artigos/SPIN-SP-23-10-2000.htm>. Acesso em: 13 mai. 2013.

DEKKERS, C. A. Pontos de função e medidas: o que é um ponto de função?. **BFPUG – Brazilian Function Point Users Group,** 1998. Disponível em: <http://www.bfpug.com.br/Artigos/Dekkers-PontosDeFuncaoEMedidas.htm>. Acesso em: 04 mai. 2013.

ENGHOLM, Jr. H. **Engenharia de Software na Prática.** São Paulo: Novatec, 2010.

HAZAN, C. Análise de Pontos de Função: Uma Aplicação nas Estimativas de Tamanho de Projetos de Software. **Engenharia de Software Magazine**, edição 02, p. 25 – 31, 2008. Disponível em: < http://www.devmedia.com.br/revista-engenharia-de-software-2/9138>. Acesso em: 29 mai. 2013

HAZAN, C. Implementação de um Processo de Medições de Software. **Governo de Pernambuco,** 2002. Disponível em: < http://www.portaisgoverno.pe.gov.br/c/document_library/get_file?uuid=396bec67-5e04-4779-9614-10268e68dc12&groupId=335215>. Acesso em: 25 mai. 2013.

INTERNATIONAL FUNCTION POINT USERS GROUP. About IFPUG. **IFPUG** , 2013. Disponível em: < http://www.ifpug.org/?page_id=6> . Acesso em: 1 jun. 2013.

MACORATTI, J.C. Análise de pontos por função - o processo de contagem. **Macoratti.net**, 2010. Disponível em: <http://www.macoratti.net/apf_pcta.htm>. Acesso em: 27 mai.2013.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

PRESSMAN, R. S. **Engenharia de Software:** Uma Abordagem Profissional. 7. ed. São Paulo: Mc Graw Hill, 2011.

PFLEEGER, S. L. **Engenharia de Software:** Teoria e Prática. 2. ed. São Paulo: Pearson, 2004.

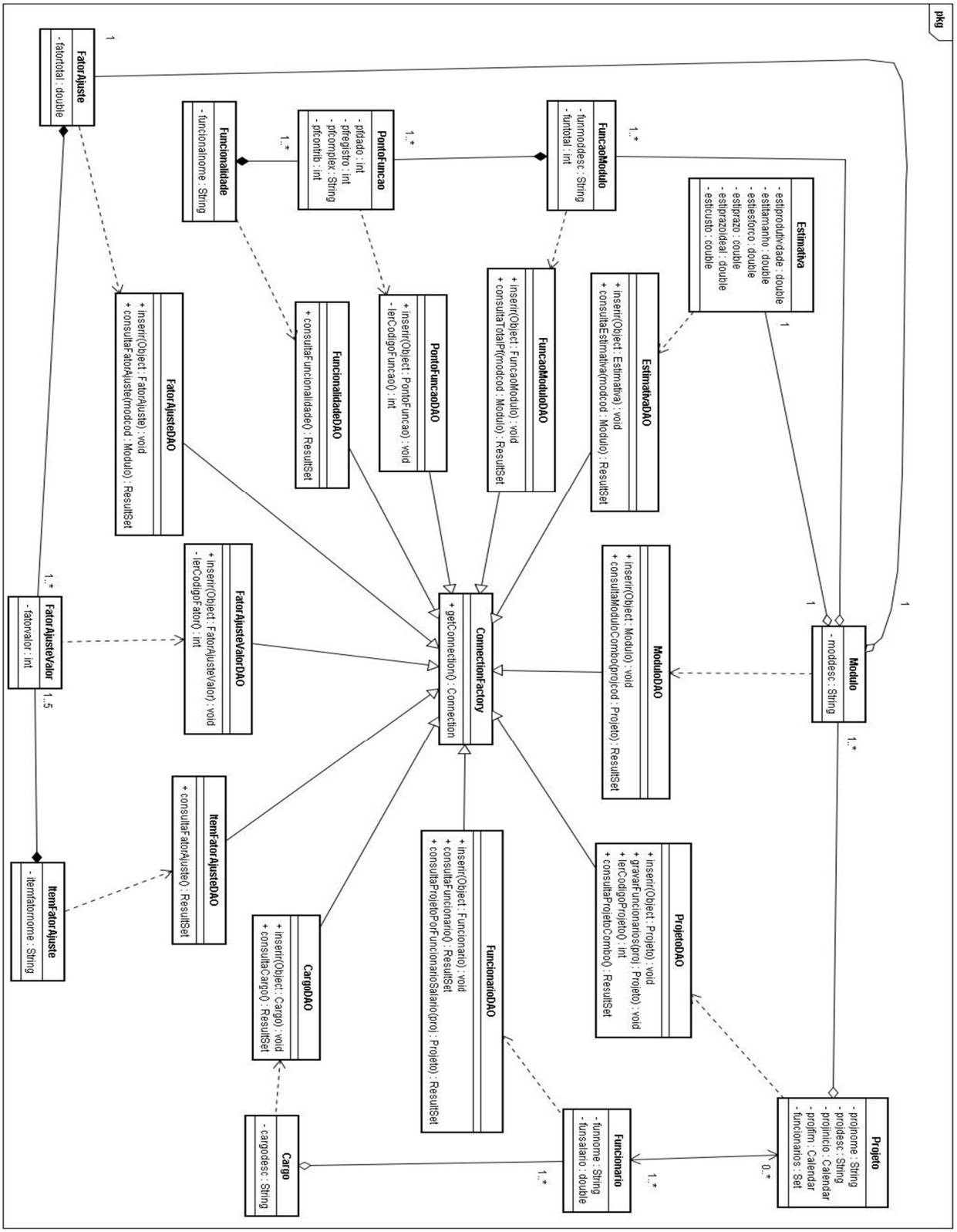
SCHACH, S. R. **Engenharia de Software:** Os Paradigmas Clássico & Orientado a Objetos. 7ª ed. São Paulo: Mc Graw Hill, 2009.

SOMMERVILLE, I. **Engenharia de Software**. Tradução: Selma Shin Shimizu Melnikoff; Reginaldo Arakaki; Edilson de Andrade Barbosa. 9. ed. São Paulo: Pearson Addison Wesley, 2011.

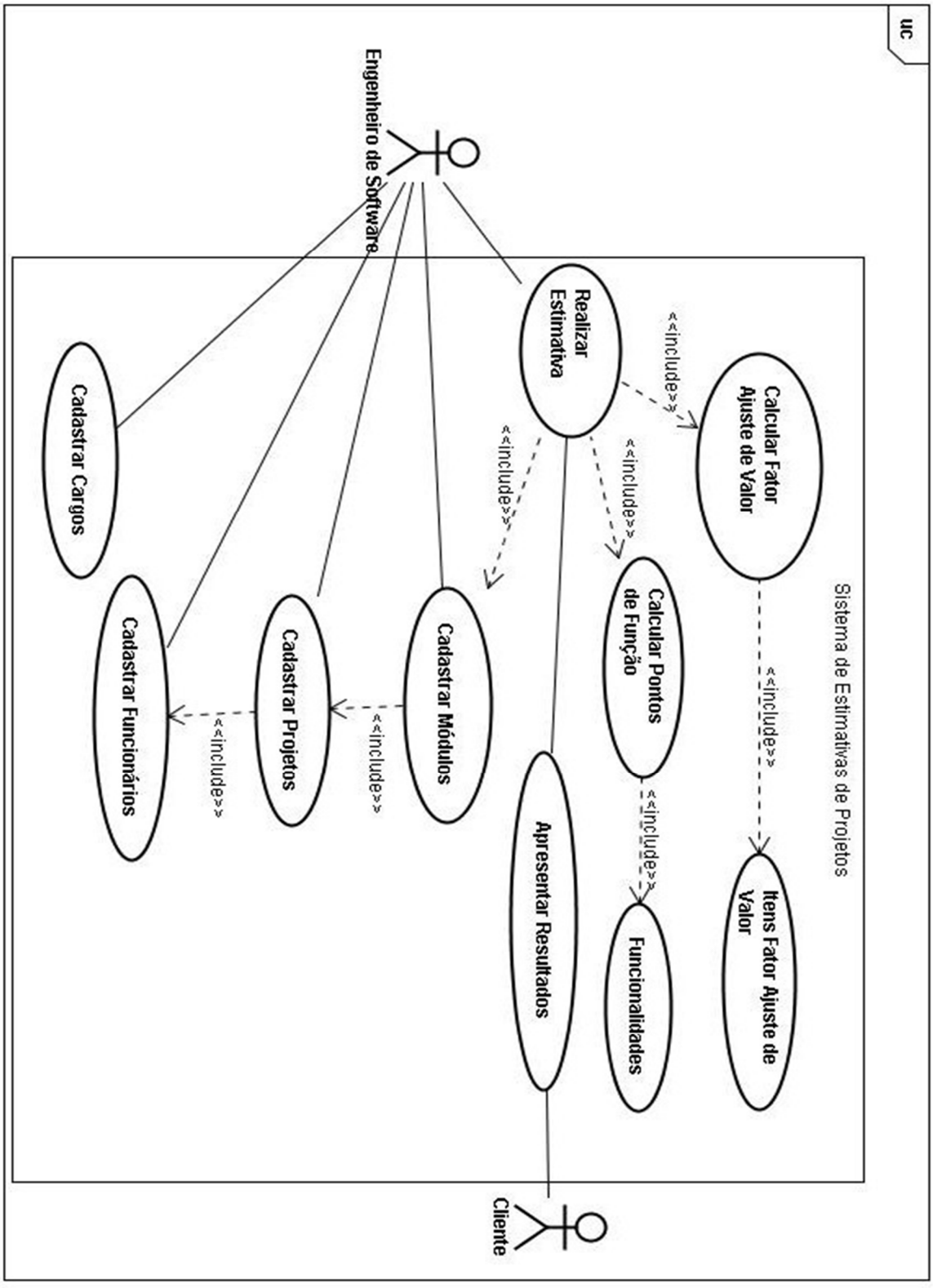
SOUZA, W. Produtividade das linguagens em pontos por função (APF). **Blog CMMI**, 2011. Disponível em: <http://www.blogcmmi.com.br/engenharia/produtividade-das-linguagens-em-pontos-por-funcao-apf>. Acesso em: 24 nov.2013.

VAZQUEZ, C. E.; SIMÕES G. S.; ALBERT R. M. **Análise de Pontos de Função:** Medição, Estimativas e Gerenciamento de Projetos de Software. 12. ed. São Paulo: Érica, 2012.

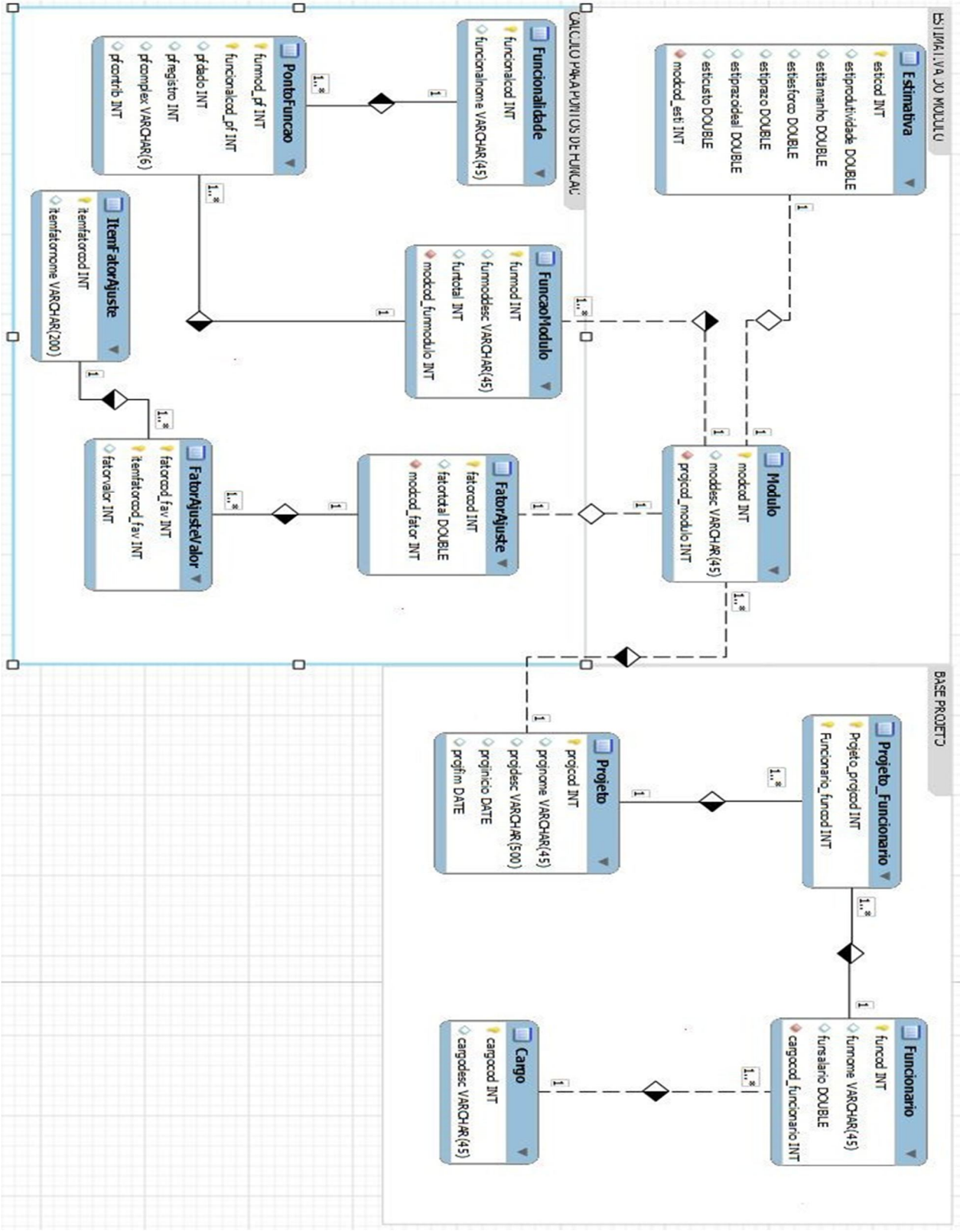
APÊNDICE A – DIAGRAMA DE CLASSES



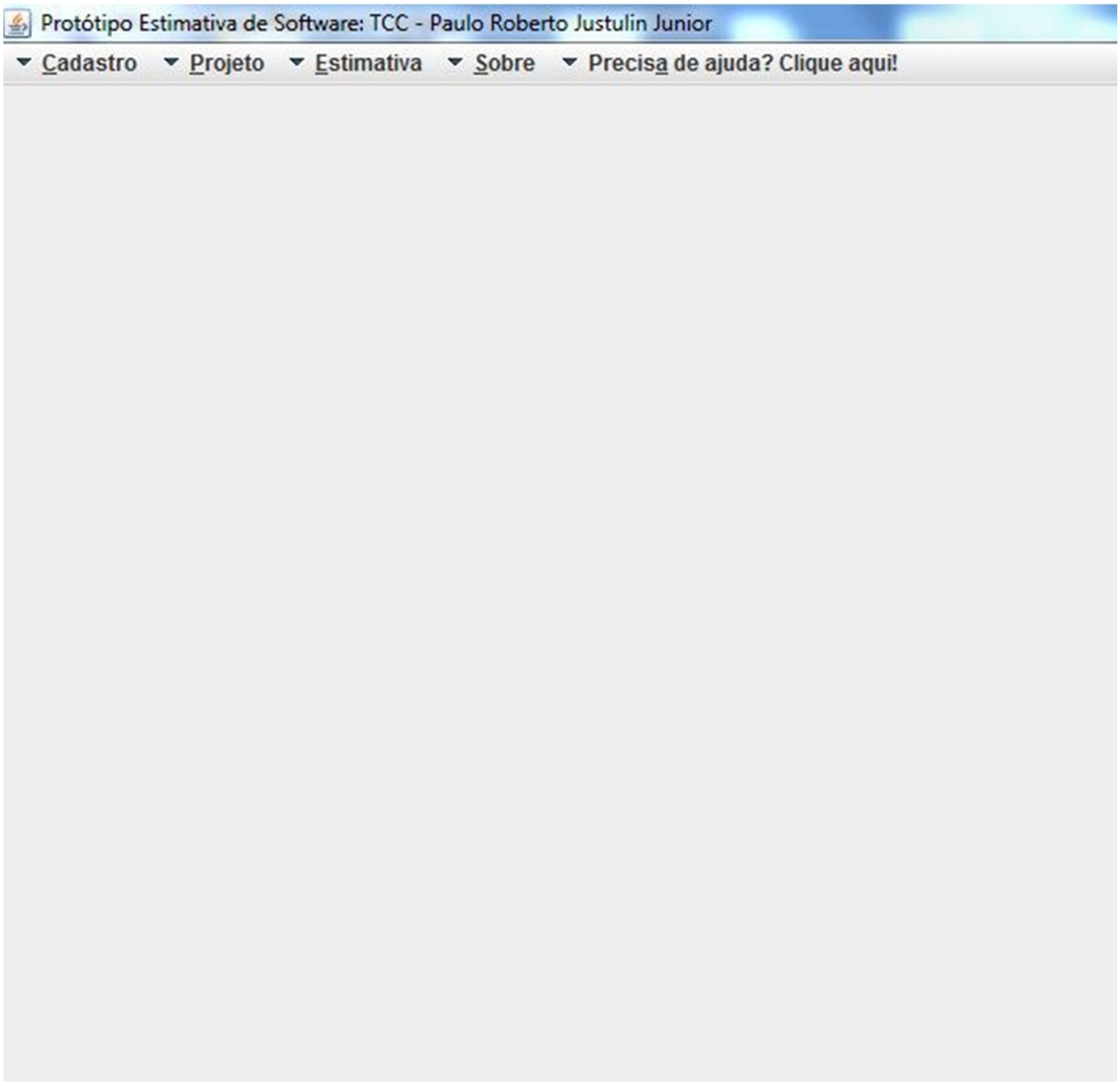
APÊNDICE B – DIAGRAMA DE CASOS DE USO



APÊNDICE C – MODELAGEM DE DADOS



APÊNDICE D – MENU PRINCIPAL



Software para Estimativas de Projetos Baseado em Análise de Pontos de Função

Paulo Roberto Justulin Junior[1], Prof. Dr. Elvio Gilberto da Silva[1], Prof. Ms. Patrick Pedreira da Silva[1], Prof. Esp. Henrique Pachioni Martins[1]

[1]Centro de Ciências Exatas e Sociais Aplicadas – Universidade do Sagrado Coração
Bauru – SP – Brasil

{paulo.justulin, patrickpsilva, henmartins}@gmail.com,
egilberto@uol.com.br

Abstract. *The project consists in creating a prototype software developed in JAVA language, which calculates the size of a project in Function Points, so this result, submit the necessary estimates that the software engineer needs: estimating size, effort, time and cost. The prototype has a simple and clear interface with natural language, so that any user can use, saving projections of manual calculations. Also stores a historical estimates of past projects, so that from these, has become an indicator for future software becoming increasingly accurate estimate. This work also shows the importance of estimating during the initial phase of the project, citing various metrics, with emphasis on analysis of the IFPUG Function Points.*

Resumo. *O projeto consiste na criação de um protótipo de um software desenvolvido na linguagem JAVA, que calcule o tamanho de um projeto em Pontos de Função, para que deste resultado, apresente as estimativas necessárias que o engenheiro de software precisa: estimativa de tamanho, esforço, prazo e custo. O protótipo apresenta uma interface simples e clara, com linguagens naturais, para que qualquer tipo de usuário possa utilizar, poupando-se de projeções de cálculos manuais. Também armazena todo um histórico de estimativas de projetos passados, para que a partir destes, tenha-se um indicador para futuros softwares, tornando-se a estimativa cada vez mais exata. Este trabalho também apresenta a importância de estimar durante a fase de inicial do projeto, citando diversas métricas, com ênfase em Análise de Pontos de Função da IFPUG.*

1. Introdução

A garantia da qualidade é umas das principais preocupações da indústria de desenvolvimento de software, pois a maioria das empresas hoje utiliza esse tipo de aplicação para gerir seus códigos, produtos e relacionamento com clientes. Há diversas medidas de garantia de qualidade para o sucesso de uma aplicação, dentre elas, umas das mais simples e menos custosa, é a medição de software. Através de dados quantitativos, ela auxilia a tomada de decisão e é capaz de informar que aspectos de produto atendem ao padrão de qualidade especificado, além de permitir o entendimento e aperfeiçoamento do processo de produção. Sem contar que irá tornar o gerenciamento de projetos baseados em fatos e não em "achismo" (ABREU, c2013).

Para se medir um software com qualidade, são utilizadas diversas métricas que são como tipos de medições aplicadas a um sistema, documentação ou processo relacionado. Através dessas métricas é possível determinar esforço, tempo, custo e tamanho do produto, por exemplo. Para obter resultados realmente significativos, as métricas devem ser aplicadas em um ciclo constante, envolvendo planejamento, medição, análise de resultados, tomada de decisão e implementação das decisões. Devem, também, ser aplicadas durante as fases de desenvolvimento do software, garantindo maior impacto positivo no final.

2. Objetivo

Desenvolver um protótipo de um software para métricas e estimativas em projeto de sistema, baseado em Análise de Pontos de Função, visando facilidade e confiabilidade para que o engenheiro de software possa apenas executá-lo e gerencia-lo, não necessitando de projeções manuais.

3. Justificativa

Existem poucos *softwares* no mercado voltado para a Análise de Pontos de Função e, dos que existem, usam *interfaces* confusas e de difícil entendimento ao usuário.

Este contexto acabou motivando a escolha deste tema, pois existe uma necessidade de um programa de fácil utilização e de *interface* simples e clara, com linguagens naturais, para que qualquer tipo de usuário possa utilizar, poupando-se de projeções de cálculos manuais, atendendo tanto a profissionais, que terão mais tempo para a gestão do projeto, e focar em outros assuntos pertinentes, quanto a alunos, onde poderão ver na prática os resultados de uma métrica e onde isso poderá ajuda-los no decorrer do projeto.

4. Engenharia de Software

Segundo Sommerville (2011), a engenharia de *software* é uma disciplina de engenharia, cujo foco está na produção do *software*, desde a fase inicial da especificação do sistema até sua manutenção, quando o sistema já esta em uso.

Em geral, os engenheiros de *software* adotam uma sistemática de trabalho, onde costuma ser de maneira mais eficiente produzir um *software* de alta qualidade. No entanto, tem tudo a ver com selecionar o método mais adequado para um conjunto de circunstâncias.

5. Métricas

Para Pressman (2011), a medição é um elemento chave de qualquer processo de engenharia. Através dela que os engenheiros de *software* visualizam o projeto e a construção do *software*, focalizando atributos específicos e mensuráveis dos artefatos da engenharia de *software*. Mas diferentemente de outras disciplinas, a engenharia de *software* não é fundamentada nas leis quantitativas da física. Por serem medidas indiretas, elas estão abertas ao debate, onde ainda alguns membros da comunidade de *software* continuam a argumentar que o *software* é “incomensurável” ou que tentativas de medições deverão ser adiadas até entendermos melhor o *software* e os atributos que deverão ser usados para descrevê-los. Isso é um erro.

Como relata o autor, ainda há muita desconfiança quanto a métricas de *software*, porém, ela é primordial em qualquer projeto. Através delas são tomadas decisões importantes ao caminhar do projeto.

Pressman (1995) diz que essas medidas que indicam aspectos de funcionalidade, qualidade, eficiência, ou seja, as mais difíceis de quantificar são chamadas de medições indiretas. Há ainda as medidas diretas, que são aquelas que são mais fáceis de serem obtidas, por serem mais "físicas". O autor nos fornece também alguns exemplos como: o número de linhas de código produzidas, o tamanho de memória ocupado, a velocidade da execução, o número de erros registrados num dado período de tempo, etc.

5.1. Métricas Orientadas ao Tamanho

Segundo Pressman (1995), métricas de *software* orientadas ao tamanho são medidas diretas do *software* e do processo pelo qual ele é produzido.

De acordo com Jones (1986 citado por PRESSMAN, 1995), as métricas orientadas ao tamanho provocam controvérsias e não são universalmente aceitas como a melhor maneira de se medir o processo de desenvolvimento de software. O motivo principal é o uso de LOC (linhas de código) como medida, já que seu uso em estimativas é complexo, necessitando de um nível de detalhes muito grande e seu uso em estimativas requer um nível de detalhes que pode ser difícil de conseguir, isto é, o planejador deve estimar o número de linhas de código que serão desenvolvidas muito antes que a análise e o projeto tenham sido concluídos.

5.2. Métricas Orientadas à Função

Pressman (1995) afirma que as métricas de *software* orientadas à função são medidas indiretas de *software* e do processo ao qual é desenvolvido. A métrica orientada à função não conta linhas de código, e sim a "funcionalidade" ou "utilidade" do programa.

Dekkers (2003 citado por HAZAN, 2008) completa que a métrica pontos de função é uma medida de tamanho funcional de projetos de *software*, considerando as funcionalidades implementadas, sob o ponto de vista do usuário. A contagem de pontos de função é independentemente da metodologia de desenvolvimento utilizados do andamento *software.* Portanto, a autora recomenda a utilização desta métrica nas estimativas de tamanho de projetos de *software*. Métricas orientadas à função serão melhor discutidas no capítulo a seguir.

6. Análise de Pontos de Função – Visão Geral

A International Function Point Users' Group (IFPUG) é uma organização sem fins lucrativos, membro da organização governada. A missão do IFPUG é ser reconhecida como líder na promoção e incentivo à gestão eficaz de desenvolvimento de software aplicativo e atividades de manutenção através do uso de Análise de Pontos de Função (APF) e outras técnicas de medição de software

Vazquez (2012) diz que é a Análise de Pontos de Função (APF) é uma técnica de medição das funcionalidades fornecidas por um *software* do ponto de vista do usuário. A medição é independente da tecnologia utilizada para o desenvolvimento do sistema, ou seja, a APF busca medir o que o produto faz, e não como ele foi construído.

O processo de medição desta técnica baseia-se em uma avaliação padronizada dos requisitos lógicos do usuário.

Dekkers (1998) explica que pontos de função não medem a produtividade ou o esforço. Pontos de função medem o tamanho do que o *software* faz ao invés de como ele é desenvolvido e implementado. Isto significa que, dado um conjunto de requisitos de usuário, o tamanho funcional do *software* será o mesmo, independente de sua linguagem. Saber o tamanho do *software* é um dos primeiros passos do processo de estimativa de esforço, prazo e custo, porém, ele destaca que pontos de função não medem diretamente essas estimativas, mas sim, mede exclusivamente o tamanho funcional do *software*. Este tamanho com outras variáveis é que pode ser usada para as estimativas citadas acima. A Figura 1 mostra a visão geral do processo de medição funcional do IFPUG:

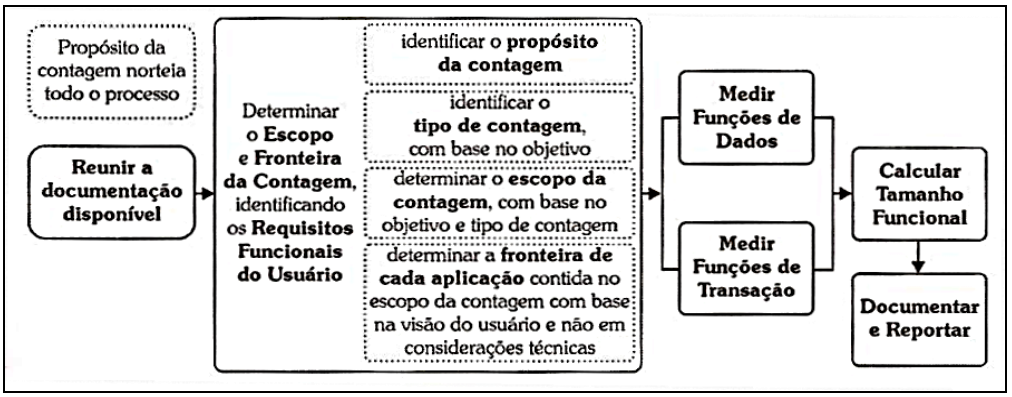


Figura 1. Visão geral do processo de medição funcional do IFPUG.

Basicamente a medição consiste em decompor o projeto em elementos denominados componentes funcionais básicos (ou funções). Vazquez (2012) ainda afirma que o método de medição conceitua abstrações, os tipos de função, nos quais os componentes básicos são classificados. Essa classificação é feita conforme sua função de armazenamento ou transação, sendo necessária a solicitação do usuário, conforme ilustra a Figura 2:

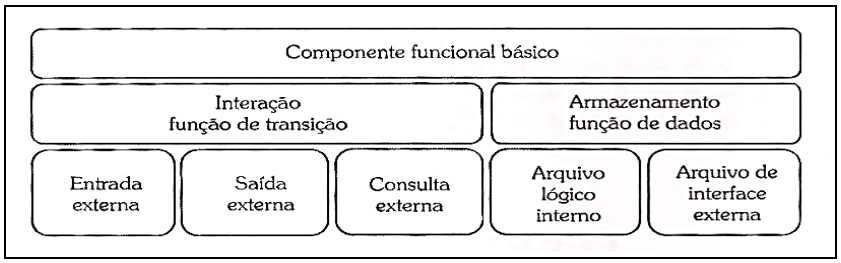


Figura 2. Classificação dos tipos de função.

6.1. Contagem – Funções de Dados

Segundo Vazquez (2012), as funções do tipo dado representam as funcionalidades fornecidas pelo sistema ao usuário com o objetivo de atender as suas necessidades de armazenamento. São classificados como Arquivos Lógicos Internos (ALI) e Arquivos de *Interface* Externa (AIE). Pressman (2011) exemplifica cada um deles:

Arquivo Lógico Interno (ALI): Também conhecido como (*internal logical files* – ILFs), cada arquivo logico interno é um agrupamento lógico de dados que reside dentro das fronteiras do aplicativo e é mantido através de entradas externas. IFPUG (1999 citado MACORATTI, c2010) fornece exemplos de ALIs: Dados da aplicação (arquivos mestres como cadastro de clientes ou funcionários); Arquivos de dados de segurança da aplicação; Arquivos de mensagens de erro.

Arquivo de Interface Externa (AIE): Também conhecido como (*external interface files* – EIFs), cada arquivo de interface externo é um agrupamento lógico de dados que reside fora da aplicação, mas fornece informações que podem ser usadas pela aplicação.

Pode-se notar que a principal diferença entre ALI e AIE é que o Arquivo de *Interface* Externa não é mantido pela aplicação. Ele esta fora da fronteira da aplicação, enquanto o Arquivo Lógico Interno está dentro. O AIE é um ALI de outra aplicação (VAZQUEZ, 2012).

6.2. Contagem – Funções de Transação

Vazquez (2012) explica que as funções de transação representam a funcionalidade fornecida ao usuário para atender às suas necessidades de processamento de dados pela aplicação. São classificados como Entrada Externa, Saída Externa e Consulta Externa. Pressman (2011) apresenta as classificações e descrições das mesmas:

Entrada Externa (EE): Também conhecido como (*external inputs* – EIs), cada entrada externa é originada de um usuário ou transmitida de outra aplicação e fornece dados distintos orientados a aplicação ou informações de controle. As entradas são muitas vezes usadas para atualizar arquivos lógicos internos (ALIs). Vazquez (2012) apresenta alguns exemplos de EE: incluir cliente, alterar cliente e excluir cliente. Processamento em lotes de atualização de bases cadastrais a partir de arquivos de movimento.

Saída Externa: Também conhecido como (*external outputs* – EOs), cada saída externa é formado por dados derivados da aplicação e fornece as informações para o usuário. Pode-se exemplificar que saídas externas se referem a relatórios, telas, mensagens de erros etc.

Consulta Externa: Também conhecido como (*External Inquiries* – EQs), é definida como uma entrada online que resulta na geração de alguma resposta imediata do software na forma de uma saída online, onde muitas vezes é obtida de um Arquivo Lógico Interno. Vazquez (2012) cita alguns exemplos de Consultas Externas, tais como: informações em formato gráfico, consulta de cadastro de clientes, telas de login (sem criptografia).

Vazquez (2012) explica que o conceito de processo elementar é o mais importante para a medição das funções de transação. Se por um lado ele evita que vários processos elementares sejam contados como um único, por outro lado impede que subprocessos componentes de processos elementares também sejam contados. Levando para a parte prática, a validação de CPF não pode ser considerada um processo elementar. De forma análoga, uma tela de cadastro que permite incluir, alterar e excluir deve ser contada como três processos elementares.

6.3. Cálculo do Tamanho Funcional

Cada função do tipo dado e transação possui um peso em PF, que é determinado pela complexidade da função, podendo ser baixa, média ou alta.

A complexidade das funções do tipo dado é determinada por dois parâmetros, sendo quantidade de tipos de dados e quantidade de tipos de registros. Já as funções do tipo transação têm sua complexidade determinado por outros dois parâmetros: quantidade de tipos de dados e quantidade de arquivos referenciados. Após todas essas identificações e classificações, o numero de pontos de função será a soma do peso de cada uma dessas funções.

Uma vez coletados esses dados, é completada a tabela da Figura 3, e associado a um valor da complexidade com cada contagem (PRESSMAN, 2011).

Descrição do	Complexidade		
Tipo Funcional	Simples	Média	Complexa
Arquivo Lógico Interno (ALI)	7 PFs	10 PFs	15 PFs
Arquivo de Interface Externa (AIE)	5 PFs	7 PFs	10 PFs
Entrada Externa (EE)	3 PFs	4 PFs	6 PFs
Saída Externa (SE)	4 PFs	5 PFs	7 PFs
Consulta Externa (CE)	3 PFs	4 PFs	**6 PFs**

Figura 3. Complexidade dos tipos funcionais.

7. Fator de Ajuste

O IFPUG tornou-se o fator de ajuste opcional na aplicação da técnica de análise de pontos de função. O propósito do fator de ajuste é medir requisitos gerais da aplicação (não funcionais), ajustando os pontos de função em mais ou menos 35% de acordo com a influência de 14 características gerais que deve ser respondido com uma escala que varia de 0 a 5 (VAZQUEZ, 2012).

Após apurar-se o nível de influência de cada pergunta, Dekkers (1998) determina o Fator de Ajuste de Valor (FAV) baseado na equação: (FAV = 0,65 + (Soma das Características Gerais do Sistema x 0,01).

8. Metodologia

A UML é uma linguagem (notação com semântica associada) para visualizar, especificar, construir e documentar os artefatos de um sistema. Foi modelado o Diagrama de Classes e Diagrama de Casos de Uso com suas devidas descrições, utilizando a ferramenta JUDE.

Vazquez (2012) explica que é de suma importância o armazenamento das informações de projetos passados, visando à obtenção de um histórico dessas medições. Com isso, foi feita modelagem do banco de dados e a geração de scripts utilizando a ferramenta MySQL Workbench 5.2.46. O Sistema de Gerenciamento de Banco de Dados (SGBD) escolhido foi o MySQL.

O protótipo deste trabalho é destinado para desktop. Sendo assim, o desenvolvimento foi realizado na linguagem JAVA, tendo uma grande integração com o SGBD e com as modelagens, adaptando-se em qualquer plataforma.

8.1. Estimativas

O primeiro passo para alcançar as estimativas efetivas do projeto de software é estimar o seu tamanho em pontos de função (PF). Para totalizar, Hazan (2008) afirma que deve-se multiplicar os ALI, AIE, EE, SE e CE pela respectiva complexidade de cada uma, fazendo um somatório total. Após obter este resultado, multiplica-se com o resultado dos fatores de ajustes de valor (FAV) = (0,65 + (Soma das Características Gerais do Sistema x 0,01)). Vazquez (2012) concluir que após ter vários projetos similares passados em base de dados, estima-se o tamanho do novo projeto com base em percentual do tamanho daquele comparado. Pode-se dividir em módulos menores e estimar o percentual proporcional.

De posse da estimativa de tamanho, procede-se com a geração da estimativa de esforço. Deve-se obter, primeiramente, um índice de produtividade por meio de análise do banco de dados de histórico de projetos da organização, observando-se os atributos do projeto em questão e o esforço realizado em projetos similares. O cálculo nada mais é do que o produto entre o valor obtido da estimativa de tamanho com o número de horas para produzir um PF. O resultado será o esforço em HH (homens_hora) (HAZAN, 2008).

O terceiro passo é determinar o cronograma do projeto. Em conformidade com Hazan (2002) a fórmula para a estimativa de prazo consiste em: Prazo (em dias) = Esforço (horas) / Tamanho da equipe * 6 horas. O resultado é dado em dias úteis. Para transformar em meses, basta apenas dividir o resultado por 22 (média de dias úteis no mês). Essa constante de 6 horas refere-se à produtividade média diária no Brasil (JONES 1997 citado por HAZAN, 2008).

O último passo é a estimativa de custo. Deve-se considerar o valor do salário hora da equipe alocada ao projeto, bem como outros custos de ambiente, ferramentas, deslocamentos, consultoria, etc. O ideal é ter dados históricos de custo por PF de projetos concluídos, possibilitando a derivação direta da estimativa de custo a partir da estimativa de volume em Pontos de Função (HAZAN, 2008).

Por fim, apresentar a estimativa para a equipe e ao cliente.

9. Resultados

A modelagem de classe foi feita no modelo *Model, View, Controller* (MVC), para facilitar o desenvolvimento de software que condiz com esta arquitetura. A modelagem de dados foi bem representada e de fácil entendimento para o leitor, utilizando de variáveis claras.

9.1. Resultados do Software

O usuário se encontrará num ambiente bem dinâmico e de fácil aprendizagem, usufruindo de *combobox* e tabelas dinâmicas, botões de ajuda, elementos *tooltip* e telas

padronizadas. O ambiente onde se realiza as estimativas é simples e já consta com os atalhos para cálculo do fator de ajuste e dos pontos de função

9.2. Resultados do Projeto

Foram simuladas as estimativas com o mesmo *software* desenvolvido neste trabalho. Foi cadastrado um funcionário com salário fictício de R$ 10,00 por hora. Somente ele irá trabalhar no projeto chamado TCC e módulo denominado TCC II.

De posso dessas informações, já se pode iniciar as métricas. O próximo passo seria obter o valor do fator de ajuste de valor. De acordo com o teste realizado, o resultado obtido do software é um fator de 0,72, conforme ilustra a Figura 4.

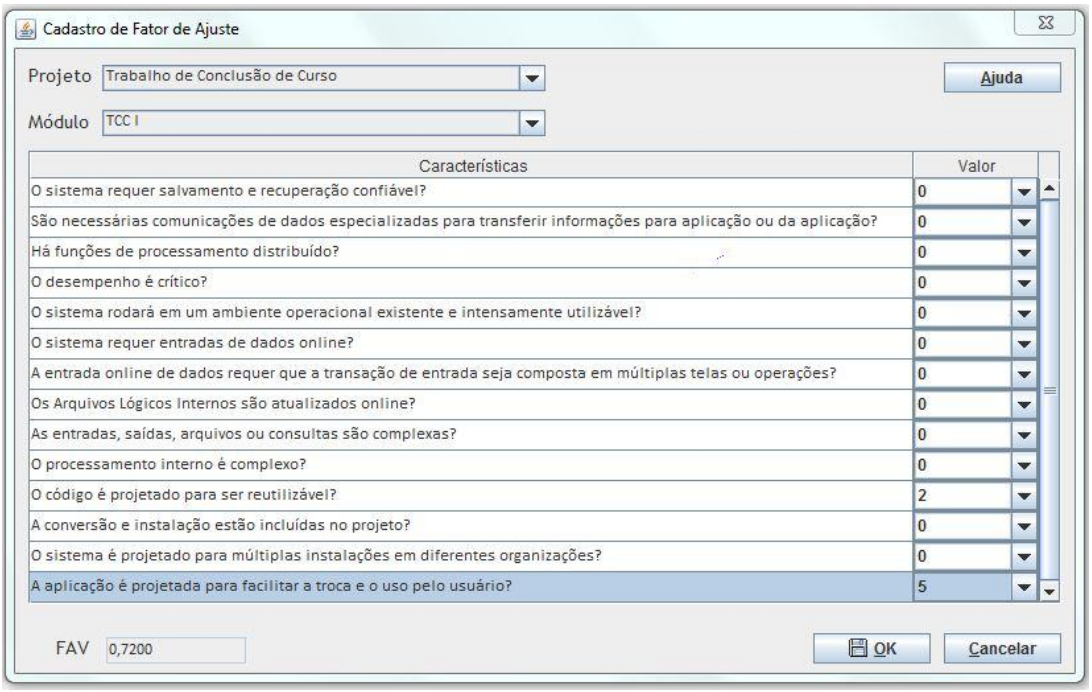


Figura 4. Tela para gerar o fator de ajuste de valor.

Com isso, deve ser descrito a função que será medida, selecionar a funcionalidade e preencher a quantidade de tipos de dados e tipos de registros ou arquivos referenciados. Após clicar no botão de inclusão, o sistema fará o cálculo de complexidade e contribuição de pontos de função, fazendo um somatório geral no fim da tela, conforme Figura 5.

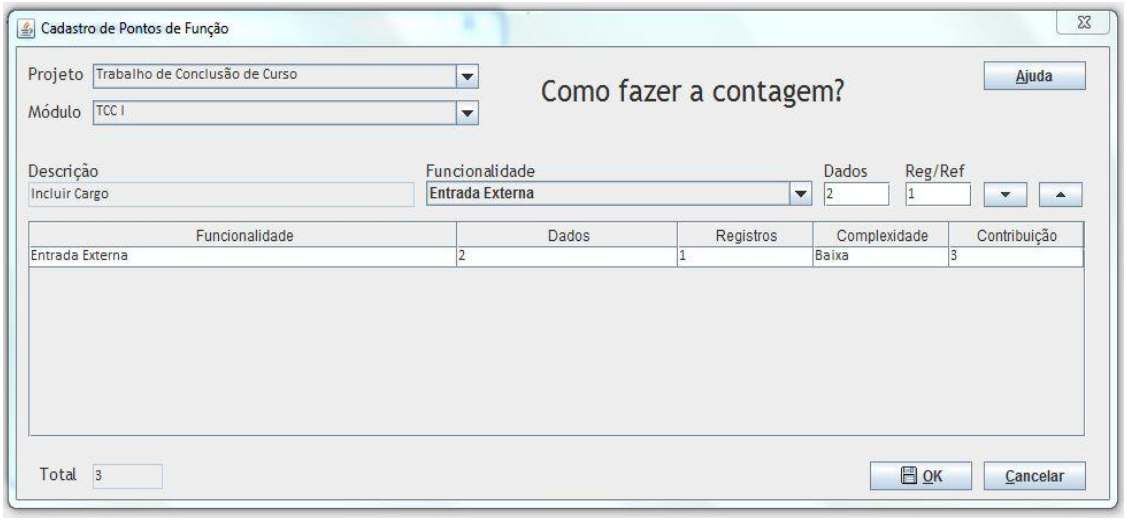


Figura 5. Tela para gerar os pontos de função.

Aberta a tela da Figura 6, deve-se preencher a média de quantidade de horas que a equipe produz um ponto de função. Por isso há a grande valia de manter o histórico de projetos para ter um indicador. Segundo Souza (2011), a produtividade varia muito de empresa para outra empresa e conhecer a sua equipe e sua produtividade é fundamental para ter uma estimativa mais próxima da exatidão. Mesmo assim, o autor compilou algumas referências sobre produtividade por Pontos de Função (como IFPUG e outros) para ter uma base e "arriscou" alguns números, apontando que na linguagem Java, a produtividade seria em média de 10h/PF. Como será o primeiro projeto e não há bases históricas, será utilizada a produtividade apontada pelo autor acima, atingindo um resultado conforme a Figura 6.

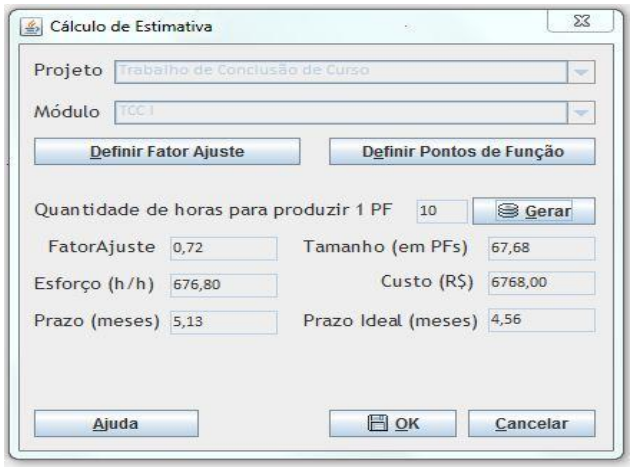


Figura 6. Cálculo da Estimativa

10. Considerações Finais

A ferramenta desenvolvida cumpriu com o esperado e condiz com tudo o que foi explícito pelos autores, apresentando resultados coerentes. O prazo foi próximo da realidade, pois o protótipo iniciou-se no final de junho/2013 (com os diagramas de classes UML) com término no fim de novembro/2013. As demais variáveis foram exatas a partir das fórmulas dos autores.

Num próximo projeto que fosse utilizar a mesma equipe, atribuía-se ainda a produtividade em 10h para produzir 1 PF e continuaria com o mesmo critério na análise de pontos de função. Em casos que a equipe se modifique, entre uma pessoa mais experiente ou menos experiente, deve-se também adequar a produtividade proporcionalmente até chegar a um número mais próximo.

Como trabalho futuro, a ferramenta poderia ser implementada com cálculos a partir de um projeto já existente, aplicando-se a fórmula do projeto de melhoria, ou até utilizar o aprendizado de máquina (inteligência artificial) para estimar a produtividade de uma equipe baseando-se em projetos passados. Porém, é de absoluta certeza que só esta ferramenta não deixará o engenheiro de software com qualidade e exatidão em seus projetos. Além do histórico de produtividade e experiências passadas, são de suma importância todos os fundamentos científicos da Engenharia de Software.

Referências

Abreu, T. C. L. (2013) "Métricas de Software: Como utilizá-las no gerenciamento de projetos de software", http://www.devmedia.com.br/artigo-engenharia-de-software-21-metricas-de-software/15776, Maio.

Dekkers, C. A. (1998) "Pontos de função e medidas: o que é um ponto de função?", http://www.bfpug.com.br/Artigos/Dekkers-PontosDeFuncaoEMedidas.htm, Maio.

Hazan, C. (2008) "Análise de Pontos de Função: Uma Aplicação nas Estimativas de Tamanho de Projetos de Software", http://www.devmedia.com.br/revista-engenharia-de-software-2/9138, 2ª edição, p. 25-31., Maio.

Hazan, C. (2002) "Implementação de um Processo de Medições de Software", http://www.portaisgoverno.pe.gov.br/c/document_library/get_file?uuid=396bec67-5e04-4779-9614-10268e68dc12&groupId=335215, Maio.

International Function Point Users Group. (2013) About IFPUG, http://www.ifpug.org/?page_id=6, Junho.

Macoratti, J. C. (2010) "Análise de pontos por função - o processo de contagem", http://www.macoratti.net/apf_pcta.htm, Maio.

Pressman, R. S. (1995) "Engenharia de Software", Makron Books., Brasil.

Pressman, R. S. (2011) "Engenharia de Software: Uma Abordagem Profissional", Mc Graw Hill, 7ª edição., Brasil.

Sommerville, I. (2011) "Engenharia de Software", Tradução: Selma Shin Shimizu Melnikoff; Reginaldo Arakaki; Edilson de Andrade Barbosa., Pearson Addison Wesley, 9ª edição., Brasil.

Souza, W. (2011) "Produtividade das linguagens em pontos por função (APF)", http://www.blogcmmi.com.br/engenharia/produtividade-das-linguagens-em-pontos-por-funcao-apf, Novembro.

Vazquez, C. E. e Simões G. S. e Albert R. M. (2012) "Análise de Pontos de Função: Medição, Estimativa e Gerenciamento de Projetos de Software" Erica, 12ª edição., Brasil.