2011 Análise de Ponto de Função em Metodologias Ágeis: Pollyanna Ingrid Gonçalves Pimenta

UNIVERSIDADE ESTADUAL DE GOIÁS
UNIDADE UNIVERSITÁRIA DE CIÊNCIAS EXATAS E TECNOLÓGICAS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

POLLYANNA INGRID GONÇALVES PIMENTA

Análise de Ponto de Função em Metodologias Ágeis

Anápolis
Junho,2011

UNIVERSIDADE ESTADUAL DE GOIÁS
UNIDADE UNIVERSITÁRIA DE CIÊNCIAS EXATAS E TECNOLÓGICAS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

POLLYANNA INGRID GONÇALVES PIMENTA

Análise de Ponto de Função em Metodologias Ágeis

Monografia apresentada ao Departamento de Sistemas de Informação da Unidade Universitária de Ciências Exatas e Tecnológicas da Universidade Estadual de Goiás, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.
Orientadora: Prof. Nágela Bitar Lôbo

Anápolis

Junho, 2011

UNIVERSIDADE ESTADUAL DE GOIÁS
UNIDADE UNIVERSITÁRIA DE CIÊNCIAS EXATAS E TECNOLÓGICAS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Análise de Ponto de Função em Metodologias Ágeis

Monografia apresentada ao Departamento de Sistemas de Informação da Unidade Universitária de Ciências Exatas e Tecnológicas da Universidade Estadual de Goiás, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado por:

Nágela Bitar Lôbo

Nágela Bitar Lôbo, Especialista, UEG
(ORIENTADORA)

José Leonardo O. Lima

José Leonardo Oliveira Lima, Mestre, UEG
(EXAMINADOR)

Anápolis, 26 de novembro de 2011.

Universidade Estadual de Goiás
Prof. MsC. Lena Lúcia de Moraes
Coord. Adjunta de TCC
Curso de Sistemas de Informação da UnUCET

UNIVERSIDADE ESTADUAL DE GOIÁS

Unidade Universitária de Ciências Exatas e Tecnológicas

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO DE MONOGRAFIAS ELETRÔNICAS NO BANCO DE MONOGRAFIAS DA UNIDADE UNIVERSITÁRIA DE CIÊNCIAS EXATAS E TECNOLÓGICAS - UNuCET

Eu Pollyanna Ingrid Gonçalves Pimenta, portador do RG nº 4582591 2.A VIA Org. Exp. SSP-GO, inscrito no CPF sob nº 014538281-84, domiciliado no logradouro de nome Rua Pinheiro Chagas Quadra 209 A Lote 20 Sudoeste, na cidade de Goiânia, estado de Goiás.

Na qualidade de titular dos direitos de autor que recaem sobre a minha monografia de conclusão de curso, intitulada Análise de Ponto de Função em Metodologias Ágeis defendida em 26/11/2011, junto a banca examinadora do curso com fundamento nas disposições da lei nº 9.610 de 19 de fevereiro de 1998, autorizo a disponibilizar gratuitamente a obra citada, sem ressarcimento de direitos autorais, para fins de leitura, impressão e *downloading* pela *internet*, a título de divulgação da produção científica gerada pela Universidade Estadual de Goiás / UnUCET de Anápolis, a partir desta data.

[X] autorizo [ ] não autorizo

[X] total [ ] resumo

Assim, **autorizo a liberação total de meu trabalho, estando ciente que o conteúdo disponibilizado é de minha inteira responsabilidade.**

Anápolis, 26 de novembro de 2011.

Pollyanna Ingrid Gonçalves Pimenta.

Universidade Estadual de Goiás
Prof. MsC. Lena Lúcia de Moraes
Coord. Adjunta de TCC
Curso de Sistemas de Informação da UnUCET

FICHA CATALOGRÁFICA

Pimenta, Pollyanna Ingrid.
Análise de Ponto de Função em Metodologias Ágeis[Anápolis] 2011.
(UEG / UnUCET, Bacharelado em Sistemas de Informação, 2011).

Monografia. Universidade Estadual de Goiás, Unidade Universitária de Ciências Exatas e Tecnológicas. Departamento de Sistemas de Informação.

1. Análise de Ponto de função
2. Métricas
3. Metodologias ágeis
4.Scrum

REFERÊNCIA BIBLIOGRÁFICA

Pimenta, Pollyanna Ingrid. **Análise de Ponto de Função em Metodologias Ágeis**. Anápolis, 2011. p. Monografia – Curso de Sistemas de Informação, UnUCET, Universidade Estadual de Goiás.

CESSÃO DE DIREITOS

NOME DO AUTOR: Pollyanna Ingrid Gonçalves Pimenta
TÍTULO DO TRABALHO: Análise de ponto de função emmetodologias ágeis.
GRAU/ANO: Graduação /2011.

_____

Pollyanna Ingrid Gonçalves Pimenta
Rua Pinheiro Chagas Qd. 209 A Lt. 20, Setor Sudoeste
CEP 743030-35 – Goiânia – GO – Brasil

Dedico este trabalho à minha familia.

AGRADECIMENTOS

À minha orientadora Professora Nágela pelo constante apoio, dedicação e incentivo para o desenvolvimento deste trabalho.

À minha familia e os amigos que sempre me apoiaram em minha vida acadêmica.

A todos, os meus sinceros agradecimentos.

RESUMO

A falta de processos e documentações pode gerar um software de qualidade razoável. A documentação excessiva do sistema pode dificulta o processo de desenvolvimento do software ocorrendo atrasos na entrega do produlto já que a modelagem do sistema é longa e estática. As metodologias ágeis são mais dinamicas, como de um projeto desenvolvido sem modelagens, mas possue normas e processos que garante a qualidade do sistema.

As métricas de software garantem à qualidade de um sistema, pois elas medem não só o produto final, mas também o processo que esta sendo utilizado. A falta de padronização dessas métricas gera duvidas para uma equipe de desenvolvimento sobre qual a métrica mais acertiva para medir seu *software*, principalmente num cenário ágil, onde suas métricas são muito informais. No mercado atual a métrica Análise de Ponto de Função se difundiu no mercado se tornando padrão para desenvolver para o governo brasileiro.

Este trabalho teve como objetivo determinar a aplicabilidade da métrica Análise de Ponto de Função em projetos cujo desenvolvimento é baseado em metodologias ágeis.

Através de uma pesquisa bibliográfica e explorátoria os estudos realizados apontaram que tal junção pode ser feita desde que a contagem de pontos, procedimento inerente da técnica de Análise de Ponto de Função, não demande tanto tempo quanto o empregado nos projetos desenvolvidos através das metodologias tradicionais.

PALAVRAS-CHAVE: Desenvolvimento ágil, Scrum, Métricas, Análise de Ponto de Função

ABSTRACT

The lack of process and documentations can generate a software with moderate quality. The excessive documentation of the system can make the process of development of the software difficult, besides occurring delay in product delivery, once the modeling of the system is static and takes much time. The Agile Methodologies are more dynamic, as a project developed without modeling, but have rules and process that guarantee the quality of the system.

The metrics of software guarantee the quality of a system, because they measure the process that is being used, as well the final product. The lack of standardization creates doubts about what metric would be better applied to measure a software, mainly in a agile background, where the metric of the team of development are very informal. In current market, the Function Point Analysis metric have been largely disseminated, becoming a Brazilian Government's standard for development.

The aim of this work was to determine applicability for Function Point Analysis metric in projects whose development is based on agile methodologies.

By means of a bibliographic and exploratory research, this study pointed that this junction can be done, since the point count, procedure that is inherent of the Function Point Analysis technique, do not takes so long as that time spent in projects developed by traditional methodologies.

KEYWORDS: Agile development, Scrum, metrics, Function Point Analysis

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas	**Descrição**
AIE	Arquivo de Interface Externa
ALI	Arquivo Lógico Interno
APF	Análise de Pontos de função
AR	Arquivos Referenciados
BFPUG	*Brazilian Function Point Users Group*
CE	Consulta Externa
CPD	Centro de Processamento de Dados
CPU	Unidade de Controle de Processamento
EDS	Electronic Data Systems
EE	Entrada Externa
FPA	Function Point Analisys
IBM	International Business Machines
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IFPUG	International Function Point Users Group
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
GIT	Grau de Influencia Total
GQM	*Goal Question Metrics*
HH	Homens Horas
PF	Ponto de Função
PFNA	Pontos de Funcao não Ajustados
RAD	*Rapid Application Development*
RUP	*Rational Unified Process*
SE	Saída Externa
SLTI	Secretaria de Logística e Tecnologia da Informação
TDD	Test Driven Development
TCC	Trabalho de Conclusão de Curso
TCU	Tribunal de Contas da União
TD	Tipos de Dados
TI	Tecnologia da Informação
VFA	Valor do Fator de Ajuste

XP Extreme Programming

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

Quando pretendemos adquirir um produto, verificamos sua garantia e, assim, sabemos que foi testado por um órgão responsável pela vigilância. Órgãos como este desenvolvem testes com o produto para proteger o consumidor de possíveis danos. A aquisição de um software não poderia ser diferente. Mas como comprar um projeto que ainda não foi desenvolvido?

Por este motivo, a norma ISO 9000:2000 determina que a organização possua uma política de qualidade Também determina que a organização deve documentar seus processos de qualidade, medir o desempenho da empresa e adotar ações de melhoria continua.

A engenharia de software é a área da computação que propõe especificar, desenvolver e manter um sistema de software, com a aplicação de tecnologias e práticas de gerência de projetos, com vistas à organização, produtividade e qualidade do produto. Alguns riscos encontrados nos processos do desenvolvimento do software, como prazos apertados e a definição de um escopo impreciso, acabam conturbando o canal de comunicação entre o cliente e a equipe de desenvolvimento.

As métricas e a metodologia são disciplinas da engenharia de software que visam garantir essa qualidade tanto no processo de produção quanto no produto final do desenvolvimento de um software. A Análise de Ponto de Função é a métrica tradicionalmente adotada nas metodologias de desenvolvimento clássicas. Porém, no atual contexto da engenharia de software tais metodologias clássicas têm cedido lugar para novas técnicas, destacando-se, dentre elas, as metodologias ágeis.

A proposta deste trabalho é estudar Análise de Ponto de Função num ambiente de desenvolvimento baseado no uso de Metodologias Ágeis. E assim, explicar a contagem de ponto de função em um ambiente ágil.

O presente trabalho está dividido em capítulos. Neste capítulo, são apresentados o tema e o objetivo da dissertação, bem como a forma como o texto foi estruturado.

O Capítulo 1 é constituído pelo Referencial Teórico e em sua primeira sessão é feita uma abordagem das metodologias ágeis, passando pelo surgimento, seus princípios, suas vantagens e desvantagens, os métodos existentes e sua receptividade no mercado de desenvolvimento de software. A seguir, são apresentados os conceitos relacionados a métricas de software; além de uma apreciação generalizada sobre o uso das métricas em planejamento

de software e as classificações dessas medidas. Após, são descritas as métricas voltadas para a metodologia de desenvolvimento ágil e a métrica Análise de Pontos de Função.

Ao final desse Capítulo, aborda-se o uso da análise de ponto de função em metodologias ágeis, problema a ser pesquisado neste trabalho.

No Capítulo 2 é descrita a metodologia que será utilizada na condução da pesquisa, a justificativa de sua escolha e o detalhamento dos procedimentos a serem adotados.

A proposta do Capítulo 3 é explicitar as análises, reflexões e diagnósticos realizados no presente estudo, assim como os resultados observados.

Na conclusão, são relatadas considerações sobre o método de pesquisa, bem como recomendações para trabalhos futuros.

CAPÍTULO 1 – REFERENCIAL TEÓRICO

1.1. Metodologias Ágeis

Uma metodologia de desenvolvimento de software é um conjunto de atividades que auxilia na produção do produto. O resultado da falta de processos na produção do software é a baixa qualidade do sistema.

> As metodologias tradicionais são também chamadas de pesadas ou orientadas a documentação. Essas metodologias surgiram em um contexto de desenvolvimento de *software* muito diferente do atual, baseado apenas em um *mainframe* e terminais burros [Royce (1970)]. Na época, o custo de fazer alterações e correções era muito alto, uma vez que o acesso aos computadores eram limitados e não existiam modernas ferramentas de apoio ao desenvolvimento do software, como depuradores e analisadores de código. Por isso o software era todo planejado e documentado antes de ser implementado. A principal metodologia tradicional e muito utilizada até hoje é o modelo Clássico. (KOSCIANSKI; SOARES, 2005)

Muitas organizações desenvolvem software sem usar nenhum processo. Isso acontece devido às metodologias tradicionais não serem adequadas as suas realidades. As metodologias tradicionais devem ser aplicadas quando os requisitos são estáveis e os requisitos futuros previstos, onde, acrescentar ou mudar algum requisito fora da etapa de levantamentos de requisitos deixa o *software* mais caro e a documentação não fica de acordo com o sistema. O uso de tais metodologias, portanto, torna-se inadequado para o mercado atual de desenvolvimento de *software* que tem se tornado cada vez mais dinâmico.

Darwin (1859 apud VENDRAMEL, 2008) afirmou: “Não são as espécies mais fortes que sobrevivem, nem as mais inteligentes, mas aquelas mais sensíveis a mudanças”. As metodologias ágeis são adequadas para situações em que as mudanças de requisitos são frequentes. Assim, a metodologia deve aceitar as mudanças ao invés de prever o futuro.

1.1.1. Manifesto ágil

As metodologias ágeis tornaram-se conhecidas a partir de 2001, quando profissionais da área de software reuniram-se com o intuito de discutir a eficiência dos processos de desenvolvimento utilizados até aquele momento e publicaram o “Manifesto Ágil”.

> Estamos descobrindo maneiras melhores de desenvolver *software*, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:
>
> - Indivíduos e interações mais que processos e ferramentas

- Software em funcionamento mais que documentação abrangente
- Colaboração com o cliente mais que negociação de contratos
- Responder a mudanças mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda. (BECK et al.,2001).

O Manifesto Ágil não rejeita os processos, ferramentas e documentação, porém, prioriza os indivíduos e o software executável. Podemos considerar que o referido manifesto baseia-se nos seguintes princípios:

"Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de *software* de valor." (BECK et al.,2001). O gerenciamento tradicional de projeto assume que cumprir um plano é igual ao sucesso do projeto que, por sua vez, é igual a demonstrar valor ao cliente. Nas metodologias ágeis o valor ao cliente é frequentemente reavaliado para melhor satisfazê-lo.

"Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas." (BECK et al.,2001). As imprevisibilidades do futuro causam mudanças que podem ser vistas como tragédias a serem evitadas ou como oportunidades a serem abraçadas. Os processos maleáveis aceitam mudanças e, por este motivo, podem acrescentar ou mudar um requisito em qualquer momento do desenvolvimento.

"Entregar software funcionando com freqüencia, na escala de semanas até meses, com preferência aos períodos mais curtos." (BECK et al.,2001).Entregar pequenas interações do software para o cliente num curto intervalo de tempo.

"Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto." (BECK et al.,2001). Em vez de ter conjunto detalhado de requisitos para ser assinado no início do projeto é detalhado o que precisa para desenvolver o software durante a duração do sistema, sendo a lacuna é preenchida com interações freqüentes entre o pessoal de negócio e os desenvolvedores.

"Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho." (BECK et al.,2001). Decisões devem ser tomadas pelas pessoas que mais conhecem a situação. Isto significa que os gerentes devem confiar às suas equipes as decisões desenvolvimento.

"O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara." (BECK et al.,2001).

Conhecimento tático pode ser transferido, movendo ao redor as pessoas que o detém. A razão é que o conhecimento tácito não é somente fatos, mas relacionamentos entre fatos.

"Software funcional é a medida primária de progresso." (BECK et al.,2001). As equipes de projetos que não percebem que estão em perigo de não desenvolver o sistema até um pouco antes da entrega onde qualquer etapa que atrasa pode adiar a entrega ou fazer a entrega na data certa com os esforços de funcionários super-heróis. O desenvolvimento iterativo consegue fornece marcos que não podem ser burlados, os quais transmitem uma medida precisa do progresso.

"Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter, indefinidamente, passos constantes." (BECK et al.,2001). Agilidade depende de pessoas que estão alertas e criativas, e conseguem manter esta atenção e criatividade durante todo o projeto de desenvolvimento de software. Desenvolvimento sustentável significa encontrar um ritmo de trabalho com 40 horas de trabalho por semana.

"Contínua atenção à excelência técnica e bom design, aumenta a agilidade."(BECK et al.,2001). O desenvolvimento ágil é diferente do esforço "rápido e sujo" RAD (*Rapid Application Development*). Enquanto o desenvolvimento ágil é similar ao RAD em termos de velocidade e flexibilidade, há uma grande diferença quando surge a clareza técnica. Abordagens ágeis enfatizam a qualidade do *software* é essencial para manter a agilidade.

"Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito." (BECK et al.,2001).Qualquer tarefa de desenvolvimento de software pode ser abordada por uma variedade de métodos. No projeto ágil, é particularmente importante utilizar abordagens simples, pois elas são mais sensíveis a mudanças. É mais fácil acrescentar alguma coisa num processo muito simples do que tirar alguma coisa de um processo muito complicado.

"As melhores arquiteturas, requisitos e *designs* emergem de times auto-organizáveis." (BECK et al.,2001). A equipe que não tiver comprometimento não irá destacar-se, mesmo que invista em arquitetura e tecnologia.

"Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo."(BECK et al., 2001). As metodologias ágeis não possuem uma documentação abrangente, sendo inexistentes em alguns processos do desenvolvimento do software. Assim, qualquer equipe ágil deve refinar e refletir durante o caminho, a fim de promover a melhora constante de suas práticas para o desenvolvimento.

1.1.2. **Scrum**

É um *framework* iterativo e incremental para o gerenciamento de projetos e desenvolvimento ágil de software.

1.1.2.1. História

O *scrum* foi criado, inicialmente, como um *framework* para gerenciamento de projetos na indústria convencional. Em 1995, Ken Schwaber formalizou o *Scrum* para projetos de desenvolvimento de software.

> O *scrum* teve como inspiração inicial um artigo de Takeuchi e Nonaka, intitulado *"The New New Product Development Game"*, publicado em 1986 na revista *Harvard Business Review*. O nome "s*crum"*, uma jogada de *Rugby*, aparece nesse artigo para descrever uma abordagem ao desenvolvimento de novos produtos utilizado, na época, em diversas empresas no Japão e nos Estados Unidos. (Sabbagh, 2010)

1.1.2.2. Definição

Scrum não é um processo ou uma técnica de desenvolvimento, é um *framework* ágil, dentro do qual pode-se empregar diversos processos e técnicas. Seu papel é deixar transparecer a eficácia relativa das práticas de desenvolvimento para que se possa melhorá-las. *Scrum* é baseado nas melhores práticas aceitas pelo mercado, utilizadas e provadas por décadas.

De acordo com Scawaber e Sutherland (2008),

> *Scrum*, que é fundamentado na teoria de controle de processos empíricos, emprega uma abordagem iterativa e incremental para otimizar a previsibilidade e controlar riscos. Três pilares sustentam cada implementação de controle de processos empíricos. (SCAWABER; SUTHERLAND, 2008)

Os pilares do *scrum* são: transparência, inspeção e adaptação. "A transparência garante que aspectos do processo que afetam o resultado devem ser visíveis para aqueles que gerenciam os resultados" (SCAWABER; SUTHERLAND, 2008). Isto é, quando alguém que inspeciona um processo acredita que algo está pronto, isso deve ser equivalente à sua definição de pronto.

A inspeção, quando realizada com freqüência, garante que as variáveis inaceitáveis no processo possam ser detectadas. "A frequência da inspeção deve levar em consideração que

qualquer processo é modificado pelo próprio ato da inspeção." (SCAWABER; SUTHERLAND, 2008).

> Se o inspetor determinar, a partir da inspeção, que um ou mais aspectos do processo estão fora dos limites aceitáveis e que o produto resultante será inaceitável, ele deverá ajustar o processo ou o material sendo processado. (SCAWABER; SUTHERLAND, 2008).

Por meio da citação acima, observa-se que são características do *Scrum* a maleabilidade e a capacidade de adequação a adaptações e mudanças.

1.1.2.3. Conteúdo

O *scrum* consiste em um conjunto formado por times de *scrum*, *time-boxes* e artefatos e regras. "Times de *scrum* são projetados para otimizar flexibilidade e produtividade. Para esse fim, eles são auto-organizáveis, multidisciplinares e trabalham em iterações"(SCAWABER; SUTHERLAND, 2008).

Os *time-boxes* são eventos com durações fixas para criar uma regularidade entre os elementos do *scrum*. Entre os elementos do *scrum* que possuem (senão, os verbos têm e temos ficariam muito próximos) duração fixa, temos a reunião de planejamento da *release*, a reunião de planejamento da *sprint*, a *sprint*, a *daily scrum*, a revisão da *sprint* e a retrospectiva da *sprint*.

> O coração do *scrum* é a *sprint*, que é uma iteração de um mês ou menos, de duração consistente como esforço de desenvolvimento. Todas as *sprints* utilizam o mesmo modelo de *scrum* e todas as *sprints* têm como resultado um incremento do produto final que é potencialmente entregável. Cada *sprint* começa imediatamente após a anterior. (SCAWABER; SUTHERLAND, 2008)

Os artefatos do *scrum* incluem: o *product backlog*, o *burndown* da *release*, o *sprint backlog* e o *burndown* da *sprint*. As regras fazem o elo entre os *time-boxes*, a equipe e os artefatos do *scrum*.

> Uma regra do scrum diz que somente membros do Time – as pessoas comprometidas em transformar o *product backlog* em um incremento - podem falar durante uma *Daily Scrum*[1]. Modos de implementar *scrum* que não são regras, mas sim sugestões.(SCAWABER; SUTHERLAND, 2008)

[1] *Daily Scrum*: Reunião continua diaria.

Sobre os papéis desempenhados no *scrum,* Scawaber e Sutherland (2008) elucidam que o *ScrumMaster* treina o time para ser mais produtivo e desenvolver os produtos com uma maior qualidade. "O *ScrumMaster* é responsável por garantir que o time de *scrum* esteja aderindo aos valores do *scrum*, às práticas e às regras." (SCAWABER; SUTHERLAND, 2008). Ele protege a equipe de interferências externas e assegura que os *sprints* não contenham itens além do que pode ser realmente entregue. Em alguns lugares, tem-se a visão de que o *ScrumMaster* é um gerente de projetos, contudo, seu papel é de facilitador, alguém que tem a missão de fazer o time usar a auto-organização e a multidisciplinaridade.

O *product owner* é responsável por criar o *product backlog,* priorizando o mais importante para o negócio; também é responsável por fazer as alterações dos itens, quer seja remoção quer seja adição. O seu papel é representar o cliente, o qual conhece o negócio e suas regras de funcionamento.

> O *product owner* é uma pessoa, e não um comitê. Podem existir comitês que aconselhem ou influenciem essa pessoa, masquem quiser mudar a prioridade de um item, terá que convencer o *product owner*. Empresas que adotam *Scrum* podem perceber que isso influencia seus métodos para definir prioridades e requisitos ao longo do tempo. (SCAWABER e SUTHERLAND, 2008)

O Time, também conhecido como *team,* é responsável por transformar itens do *product backlog* em itens do software pronto para ser entregues. As equipes de *scrum* contêm geralmente entre cinco e nove pessoas. Os membros são multifuncionais e podem ser desenvolvedores, *designers*, arquitetos da informação, dentre outros. Outra característica importante é que os times são auto-gerenciáveis, sendo eles responsáveis por controlar as tarefas do desenvolvimento da *sprint*.

> Times também são auto-organizáveis. Ninguém – nem mesmo o *ScrumMaster* – diz ao time como transformar o *product backlog* em incrementos de funcionalidades entregáveis. O Time descobre por si só. Cada membro do Time aplica sua especialidade a todos os problemas. A sinergia que resulta disso melhora a eficiência e eficácia geral do Time como um todo. (SCAWABER; SUTHERLAND, 2008)

Os *time-boxes* são eventos com durações fixas, a fim de haja regularidade entre os elementos do *scrum* que necessitam dessa característica, como: a reunião de planejamento da

release, a reunião de planejamento da *sprint*, a *sprint*, a *daily scrum*, a revisão da *sprint* e a retrospectiva da *sprint*.

O propósito da reunião de planejamento da *release* é estabelecer plano e metas, de modo que o time de *scrum* e o resto da organização possam trocar informações sobre os riscos e as características e funcionalidades do release. Essa reunião é opcional, mas seu grau de relevância é alto, tendo em vista que discussões levantadas durante a mesma podem resolver futuros problemas.

A ideia principal da *sprint* é ter um pequeno ciclo, com duração fixa, planejado e acompanhado por todos os envolvidos, inclusive pelo cliente, com o objetivo de gerir um software funcional no final. O *ScrumMaster* deve garantir que não será feita nenhuma mudança, tanto na concepção do time quanto nas metas de qualidade, que devem permanecer constante durante a *sprint*.

> As *Sprints* podem ser canceladas antes que o prazo fixo da *Sprint* tenha acabado. Somente o *Product Owner* tem a autoridade para cancelar a *Sprint*, embora ele possa fazê-lo sob influência das partes interessadas, do time ou do *ScrumMaster*. Sob que tipo de circunstâncias pode ser necessário cancelar uma *Sprint*? A gerência pode precisar cancelar uma *Sprint* se a meta da *Sprint* se tornar obsoleta. Isso pode ocorrer se a empresa mudar de direção ou se as condições do mercado ou tecnologia mudarem. Em geral, uma *Sprint* deve ser cancelada se ela não fizer mais sentido dadas as circunstâncias atuais. Porém, por causa da curta duração das *Sprints*, raramente isso faz sentido. (SCAWABER; SUTHERLAND, 2008).

Na reunião de planejamento da *sprint,* também conhecida como *sprint planning meeting,* devem estar presentes o *product ower*, o *ScrumMaster* e o time *scrum*; a duração da reunião dura 8 horas para *Sprint* de um mês, podendo diminuir este tempo de acordo com o tamanho da *sprint*. A reunião é dividida em duas etapas: na primeira, decide-se o que fazer na *sprint* e na segunda, como fazer. "O *product owner* apresenta ao time o que é mais prioritário no *product backlog*. Eles trabalham em conjunto para definir qual funcionalidade deverá ser desenvolvida durante a próxima *sprint*." (SCAWABER; SUTHERLAND, 2008).

> As entradas para essa reunião são o *Product Backlog*, o incremento mais recente ao produto, a capacidade do time e o histórico de desempenho do time. Cabe somente ao Time a decisão de quanto do *Backlog* ele deve selecionar. Somente o Time pode avaliar o que ele é capaz de realizar na próxima *Sprint*. (SCAWABER; SUTHERLAND, 2008).

A revisão da *Sprint,* também denominada *sprint review*, é uma reunião realizada ao fim de cada *sprint*, com duração fixa de três horas, que tem como objetivo apresentar o que foi desenvolvido para os representantes do projeto, o *product owner* ou o cliente, em forma de *feedback.*

> A reunião inclui ao menos os seguintes elementos. O *ProductOwner* identifica o que foi feito e o que não foi feito. O Time discute sobre o que correu bem durante a *Sprint* e quais problemas foram enfrentados, além de como esses problemas foram resolvidos. O time então demonstra o trabalho que está pronto e responde a questionamentos. O *ProductOwner* então discute o *ProductBacklog* da maneira como esse se encontra. Ele faz projeções de datas de conclusão prováveis a partir de várias hipóteses de velocidade. Em seguida, o grupo inteiro colabora sobre o que foi visto e o que isso significa com relação ao que fazer em seguida. A Revisão da *Sprint* fornece entradas valiosas para as reuniões de planejamento de *Sprints*seguintes. (SCAWABER; SUTHERLAND, 2008)

A restropectiva da *sprint* é uma reunião de três horas de duração, para *sprint* de um mês, na qual o *ScrumMaster* encoraja o time a revisar as práticas *scrum* e seu processo de desenvolvimento, com a finalidade de tornar a equipe mais eficaz. "A inspeção deve identificar e priorizar os principais itens que correram bem e aqueles que, se feitos de modo diferente, poderiam ter deixado as coisas ainda melhores." (SCAWABER; SUTHERLAND, 2008)

A *dailyscrum* é uma reunião diária de quinze minutos de duração, onde o time responde às perguntas: O que foi feito ontem? O que será feito hoje? Quais os impedimentos? Nela, o *ScrumMaster* avalia as repostas para os impedimentos citados. O *product owner* poderá ou não participar da reunião.

> As *Daily Scrums* melhoram a comunicação, eliminam outras reuniões, identificam e removem impedimentos para o desenvolvimento, ressaltam e promovem a tomada rápida de decisões e melhoram o nível de conhecimento de todos acerca do projeto. (SCAWABER; SUTHERLAND, 2008)

Os artefatos do *scrum* incluem: o *product backlog*, o *burndown*da*release*, o *sprint backlog* e o *burndown* da *sprint*. O *product backlog* possui todos os requisitos para o *software***,** onde o seu conteúdo é priorização das tarefas mostram, uma evolução medida que o *software* está sendo usado em seu ambiente, se tornando mais compreendido. Sendo dinâmico

pela sua constante mudança para melhor se adaptar ao cliente. O *product backlog* existirá enquanto existir o produto.

Para Scawaber e Sutherland (2008):

> O *Product Backlog* representa tudo que é necessário para desenvolver e lançar um produto de sucesso. É uma lista de todas as características, funções, tecnologias, melhorias e correções de defeitos que constituem as mudanças que serão efetuadas no produto para releases futuras. Os itens do *Product Backlog* possuem os atributos de descrição, prioridade e estimativa. A prioridade é determinada por risco, valor e necessidade. Há diversas técnicas para dar valor a esses atributos.

Frequentemente, vários times *scrum* trabalham juntos no mesmo produto. Desta forma, um único *product backlog* é utilizado para todos os times. O gráfico *burdownrealise* registra a soma das estimativas dos esforços restantes do *product backlo*g e este esforço estimado pode estar em qualquer medida de trabalho que a equipe escolher. O *sprint backlog* consiste nas tarefas que são feitas para transformar os itens do *product backlog* em incremento "pronto", espelhando todo o esforço que a equipe depreende para alcançar a meta da *sprint*. São divididos em tarefas menores e podem ser entendidos na *daily scrum*. O *burndown* do *sprint* é um gráfico da quantidade que ainda falta para terminar a *sprint*.

User story ou estórias de usuário é uma pequena descrição que detalha um item do *product backlog.* A estória deve ser compreensível para clientes e desenvolvedores, testável e valiosa para o cliente. São consideradas estorias de usuário como os requisitos do desenvolvimento ágil, a pessoa mais indicada para descrevê-la é o *product owner.* A estória tem três aspectos críticos, os quais devem ser lembrados quando descritos: são escritas em cartões, o que as delimita em tamanhos pequenos, devem ser um lembrete para lembrar a utilidade conversada com os *stakeholders,* as pessoas envolvidas no negócio. Por fim, o cliente ou o *product owner* deve definir uma maneira de validar a estória.

Uma boa prática de escrever uma estória é usando *invest* - um acrônimo em inglês derivado do termo Independent, estórias que devem ser independentes uma das outras. Termos próprios são utilizados na elaboração de uma *user story. Negotiable* não são contratos, mas lembretes para discussões. *Valuable* devem agregar valor para o cliente. *Estimable* são os desenvolvedores que devem ser capazes de estimar os tamanhos das estórias. *Small*, pequena estórias grandes dificultam as estimativas. A estória será quebrada ou agrupada, dependendo do caso, para que tenha um valor para o cliente e seja fácil de estimar. E por fim, Testable, considerado estória se for possível de ser testado.

> Aqui na MCP TECNOLOGIA resolvemos adotar um "padrão" para escrita das estórias, de modo a facilitar o entendimento de nosso cliente. Muitas empresas adotam esse modelo e achamos que também será mais eficiente em nosso "mundo". O modelo ficou definido da seguinte forma: *Como um**<PERFIL>**eu posso/gostaria/devo**<FUNÇÃO>**para**<VALOR AO NEGÓCIO>*Aplicando o modelo em um caso real: *Como um**<TOMADOR DE SERVIÇO>**eu gostaria de**<IMPRIMIR A NOTA FISCAL >**para**<COMPROVAR O SERVIÇO PRESTADO>*Explicando passo a passo o modelo: **QUEM?**A resposta à esta pergunta é representada pelo **<PERFIL>**. Analisando de uma outra forma, podemos dizer que representa o "papel" exercido pelo usuário no fluxo do negócio. Pergunta: Quem deseja imprimir a nota fiscal? Resposta: Tomador de Serviço **O QUE?** Respondendo a esta pergunta encontraremos uma das partes mais importantes da estória, que representa o real desejo do dono do negócio: a **<FUNÇÃO>**. Também para os desenvolvedores, essa é a informação mais valiosa, pois vai determinar o que deve ser feito. Pergunta: Após a nota fiscal ter sido emitida, o que deseja realizar? Resposta: Imprimir a nota fiscal. **POR QUE?** Pode não parecer, mas esta é uma das perguntas mais difíceis de ser respondida. Isso porque na maioria das vezes o motivo não é visto como algo muito importante tornando-se uma tarefa árdua mensurar o **<VALOR AO NEGÓCIO>**.Pergunta: Por que precisa imprimir a nota fiscal? Resposta: Para comprovar o serviço prestado. (MOURA, 2011)

Uma *task* é a menor unidade de trabalho. Também é representada por uma Tarefa indivisível, desde desenvolver o *layout* de alguma tela, escrever uma *procedure,* até testar um caso de teste. Qualquer atividade pode ser executável no *scrum.* Um conjunto de *tasks* implementa uma *feature* ou *user story*.

Há diferença entre tarefas e estórias: As estórias são trabalhos que podem ser entregues para o *product owner,* tendo uma importância para o *product backlog.* As *task* ou tarefas são atividades que não podem ser entregues, mas são importantes para o desenvolvimento do *software.*

O *task board* exibe todo o trabalho que o time está desenvolvendo durante uma *Sprint*. Ele é atualizado continuamente no decorrer da *sprint*. Se alguém pensa em alguma nova tarefa, esta deve ser escrita em um novo cartão e colocada no quadro. Antes ou durante um *daily scrum*, as estimativas são alteradas e os cartões são movidos no quadro.

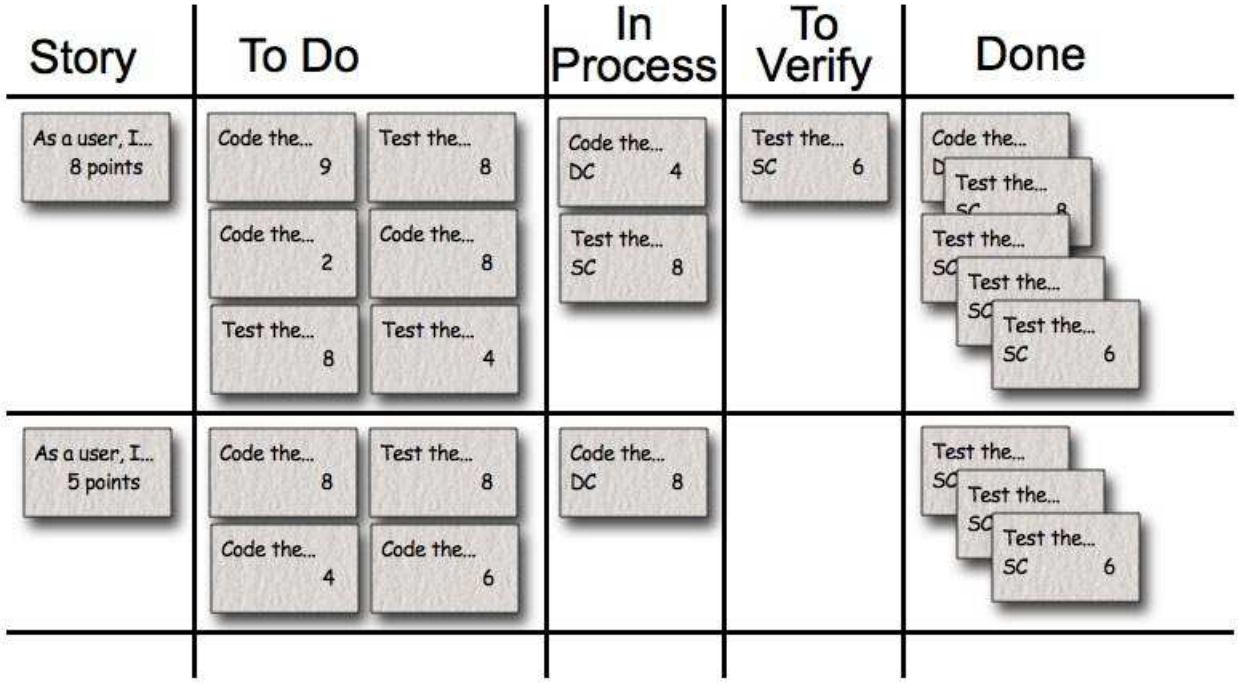

Figura2: Task Board- Quadro que exibe as tarefas de uma *sprint.* Fonte: Mountain Goat Software [s.d.].

Cada linha no quadro de tarefas é um item do *product backlog,* uma estória. Durante a reunião *sprint planning*, o time define os itens do *product backlog* que podem ser concluídos durante o próximo *sprint*. Cada item do *product backlog* é desdobrado em vários itens de *sprint backlog* e cada um deles é representado por um cartão de tarefa que é colocado no *task board* na coluna "A fazer".

As colunas são:

- Estória – As descrições da estória são exibidas nesta linha.
- A fazer - Nesta coluna ficam todos os cartões que não estão finalizados ou em andamento.
- Em andamento – Qualquer cartão que é trabalhado fica nesta coluna.
- A Verificar - Muitas tarefas têm cartões de tarefas de teste correspondentes.
- Concluído - Os cartões são empilhados nesta coluna quando estão concluídos. Eles são removidos no final do Sprint.

1.1.2.4. Escopo Iterativo e Incremental

O único objetivo de um projeto de software é produzir software. Dessa forma, a produção de software não pode ser deixada de lado para satisfazer desejos pessoais ou profissionais dos envolvidos.

Kent Beck (1999) defende o desenvolvimento de um *software* simples. Para isso, o sistema tem que ser:

- Apropriado para o público alvo. Não importa a "elegância" de um *software*; se as pessoas que irão trabalhar com ele (usuários ou desenvolvedores) não o compreendem; então, ele não é simples para elas.
- Comunicativo. Os elementos de um *software* deverão favorecer a boa comunicação com os futuros leitores.
- Fatorado. Duplicação de lógica ou estrutura dificulta o entendimento e a modificação do código. O *software* deverá ter um menor número possível de elementos, pois assim, teremos menos "coisas" a serem testadas, documentadas e comunicadas.

O *scrum* é um *framework* de gerenciamento de projetos extremamente ágil e flexível, que tem por objetivo definir um processo de desenvolvimento iterativo e incremental, podendo ser aplicado a qualquer produto ou no gerenciamento de qualquer atividade complexa.

> Um dos primeiros desafios quando se implanta metodologias ágeis, é nivelar qual o entendimento necessário para iniciar um projeto de desenvolvimento de software e como prover uma gestão de requisitos alinhada com a necessidade de entregas e com a aderência necessária dentro de uma organização. (PIMENTEL, 2009)

A filosofia incremental e iterativa das metodologias ágeis pode ter um escopo de um produto suficientemente informativo para iniciar e conduzir um projeto de desenvolvimento do mesmo.

1.1.2.5. *Scrum* e *eXtreme Programming*

O eXtreme Programming (XP) é uma metodologia de desenvolvimento ágil de software que provê flexibilidade, rapidez e alta qualidade. Foi mostrado que o scrum é focado nas melhores práticas de gerenciamento e o[2]rganização; já o XP centra –se nas tarefas de programação, dentre estas o Test Driven Development (TDD)[2] e o design incremental e contínuo, definido anteriomente. Entretanto é possivel conciliar o uso destas duas técnicas. Essa junção visa a auxiliar uma equipe a construir um software de qualidade.

[2] *Test Driven Development* (TDD) ou Desenvolvimento Orientado a Teste é uma técnica que consiste em se submeter um software a testes automatizados onde ocorre um refatoramento do código que melhore a legibilidade e remova a duplicação.

1.2. Métricas de software

Métricas de software são definidas como qualquer tipo de medição referente a um sistema, processo ou documentação de um software, onde se realiza especificações das funções de coleta de dados para avaliações e desempenho do projeto.

Na engenharia de software as medições permitem melhorar a gerência de projetos, reduzir as frustrações de cronograma e avaliar a produtividade dos processos. Propicia ainda o estudo de benefícios de novos métodos e ferramentas de engenharia de software, a identificação das melhores práticas de desenvolvimento e a garantia da qualidade sobre o produto.

1.2.1 Definição

É importante conhecer as diferenças entre os conceitos de medidas, métricas e indicadores.

"Medição é a ação de medir" (GARCIA, 1986). Consta também em (IEEE, 1983 apud SATO, 2006 p.7) que "Uma medida é uma avaliação em relação a um padrão". Podemos entender então que a medida fornece uma indicação quantitativa da extensão, da dimensão, da capacidade ou do tamanho de algum atributo de um produto ou processo. Um exemplo de medida é 10 metros; metros é o padrão e 10 a medida.

Uma métrica é a medida quantitativa do grau em que um sistema se encontra em relação a um determinado atributo. O número de defeitos encontrados nos testes é um exemplo de métrica.

Um indicador é a variável que pode ser configurada para um determinado estado com base no resultado do processo de uma determinada condição. Segundo definição do IEEE (1983 apud MARIA, 2010 p.19) um indicador é algo que chama a atenção para uma situação particular, estando geralmente relacionado à uma métrica que provê a interpretação desta numa determina situação ou contexto. Assim, pode-se afirmar que alguém que interpreta alguma métrica está considerando algum tipo de indicador. O aumento considerável nos números de defeitos encontrados na última versão é um indicador de queda de qualidade de um *software*.

1.2.2 Categorizações das métricas

Utilizando-se critérios específicos, as métricas podem ser classificadas em: métricas diretas e indiretas; métricas de produto e produtividade; métricas de qualidade e métricas técnicas; métricas privadas e públicas.

1.2.2.1. Métricas diretas e indiretas

As métricas de *software*, do ponto de vista de medição, podem ser divididas em duas categorias: diretas e indiretas.

Nas métricas diretas, são observados atributos como custo, esforço e linhas de códigos. Essas métricas são relativamente fáceis de serem reunidas, desde que as convenções específicas para medições sejam agrupadas.

As métricas indiretas são obtidas por atributos, como: eficiência, confiabilidade, qualidade e funcionalidade. Estas métricas são consideradas mais difíceis de serem avaliadas.

1.2.2.2. Métricas de produto e produtividade

As métricas de produto ocupam-se com as características do software. São divididas entre métricas estáticas e dinâmicas. As primeiras são coletadas por medições feitas das representações do sistema, como projeto, programa ou documentações. Já as segundas são coletadas por medições de um programa em execução.

As métricas de produtividade concentram-se na saída do processo de engenharia de software como números de caso de uso ou iterações.

1.2.2.3 Métricas de qualidade e métricas técnicas (Ou coloca corret

As métricas de qualidade oferecem uma indicação das exigências do cliente em relação ao software, como: erros e fases. As medidas de qualidade de software incluem: corretitude, manutenibilidade, integridade e usabilidade.

Na corretitude, o programa deve funcionar corretamente para ser relevante aos usuários. A corretitude é o grau em que o software executa a função que lhe é exigida.

Manutenibilidade é a facilidade com que um software pode ser corrigido, modificado ou adaptado. Não existe nenhuma forma de medir a manutenibilidade diretamente; mas, caso necessário, utiliza-se medidas indiretas.

Integridade mede a capacidade de um sistema de suportar ataque e ainda assim preservar os componentes do software, quais sejam: programas, dados e documentação.

Usabilidade mede a adaptação do *software* ao usuário.

As métricas técnicas concentram-se nas características do *software* e não no processo em que o mesmo foi desenvolvido. Exemplo disso é a complexidade lógica.

1.2.2.4. Métricas privadas e públicas

As métricas privadas referem-se ao escopo do projeto para a equipe que desenvolveu o sistema. Os erros encontrados durante uma revisão é um exemplo dessa métrica.

As métricas públicas geralmente assimilam informações das métricas privadas de uma equipe e são avaliadas tentando descobrir indicadores.

1.2.3 Por que medir?

As métricas ajudam a entender o processo técnico usado para desenvolver um sistema e até o próprio produto. Sobre a importância das métricas, Vazquez (2010) menciona uma curva do pânico que pode ocorrer no desenvolvimento sem uma métrica. Essa curva é descrita por uma série de eventos interdependentes, que culminam no adiamento da entrega, tais como: menor controle de qualidade, número maior de erros, atraso no projeto, pressão para aumentar produção, pessoal novo no projeto e tempo de aculturamento, mais horas trabalhadas, desgaste da equipe, atrasando ainda mais o projeto.

Para evitarmos esse quadro, temos que aumentar o controle sobre o planejamento e o desenvolvimento do *software.* Segundo Pressman (2006) “Se você não sabe para onde você quer ir, qualquer caminho você pode seguir. Se você não sabe onde você está, um mapa não vai ajudar!”. Então, ao orientarmo-nos, podemos prever onde estamos e como chegaremos ao destino.

As métricas permitem comparar os valores obtidos com padrões estabelecidos para a organização, com o objetivo de obter os indicadores da qualidade. Essas estimativas também auxiliam nas tarefas de entender, avaliar e aperfeiçoar o processo de desenvolvimento.

As métricas acabaram-se tornando uma ferramenta de grande auxílio na gestão de um projeto. Sobre isso, Vasconcelos (2005) afirma: “Não se pode gerenciar o que não se pode medir.” As medidas coletadas dão visibilidade ao estado do projeto, o que permite verificar o se o caminho adotado está correto e para a tomada de decisões. Podem também melhorar o relacionamento com clientes e superiores, já que os riscos podem ser previstos, além de embasarem a solicitações de novas ferramentas.

1.2.4 Os papéis da medição.

De acordo com Humphrey (2008 apud BURGOS, 2009, p.15), a medição de software tem quatro papéis: entender, avaliar, prever e aperfeiçoar. As métricas ajudam a entender o comportamento e o funcionamento de processos, produtos, serviços e recursos de software. Na avaliação, as métricas podem ser utilizadas para tomar decisões e determinar o

estabelecimento de padrões, metas e critérios de aceitação, com o objetivo de controlar processos, produtos e serviços de software. Por fim, podem ser usadas para planejar, extrapolar tendências e prever valores de atributos.

1.2.5 O que medir?

A medida é a quantificação de uma característica. No caso, medições de sofware devem avaliar não só as características do produto final, mas também as dos processos envolvidos em sua construção.

Medir é importante, porém, essa medida é limitada. Por isso, gerentes de projetos precisam saber quando devem encerrar as medições e medir apenas até o momento em que as necessidades do projeto e os objetivos da empresa estejm alinhados. Assim, deve-se estabelecer um programa de métricas adequado para não medir mais do que o necessário; uma vez que é inviável medir o que não se pode controlar, gerenciar, melhorar ou trabalhar.

No artigo "*Software and measurement: The Goal/Question/Metricparadigm*", publicado pelo departamento de Ciência da Computação da Universidade de *Maryland,* foi apresentado um modelo útil na identificação do que medir em um *software*: o modelo GQM - *GoalQuestionMetrics.*

- *Goal* ou Nível Conceitual/Objetivo: Um objetivo que define um produto. Neste nível, deve ser feita a pergunta: Quais são as metas e os objetivos do projeto?
- *Question*/ ou Nível Operacional/ Pergunta: Caracteriza-se por um conjunto de perguntas que norteia a avaliação do objetivo. A pergunta aqui é: Quais as questões que deseja responder?
- *Metrics* ou Nível Quantitativo/Métricas: Um conjunto de dados quantitativos associado a cada pergunta.

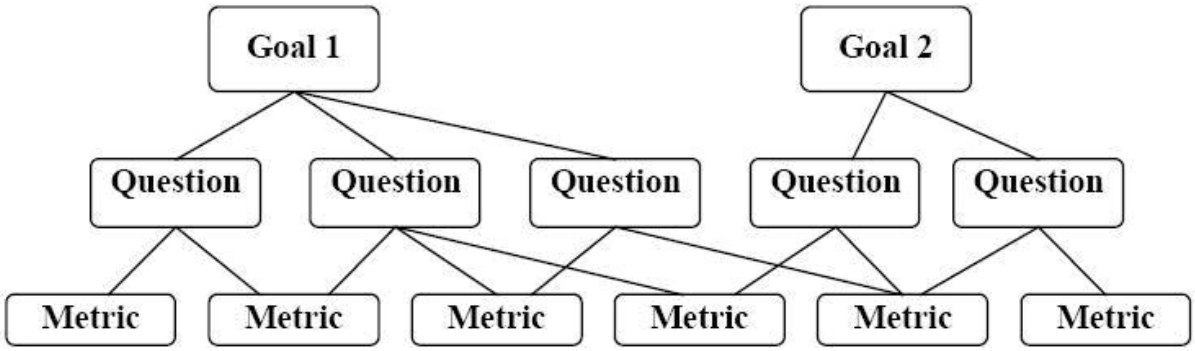

Figura 3: Identificação do que medir em um *software* Modelo GQM. Fonte: Cammarano 2008.

Esse modelo é uma estrutura hierárquica que começa com um objetivo, o qual pode ser uma motivação, preocupação ou tópico. Esse objetivo deriva várias perguntas que, por sua vez, geram uma série de métricas.

1.2.6 Métricas orientadas a tamanho e função

As métricas orientadas a tamanho consideram, como o próprio nome sugere, o tamanho do *software* produzido e referem-se a todas as atividades da engenharia.

As métricas orientadas a função são concentradas na funcionalidade do *software*. Nestas, considera-se o ponto de vista do usuário e são determinados de forma consistente o tamanho e a complexidade de um *software*.

1.2.7 Métricas ágeis

Os métodos ágeis promovem um processo empírico para o desenvolvimento do software, o que exige um ciclo constante de inspeção, adaptação e melhoria. Uma atividade proposta em XP para garantir melhorias é conhecida como *Tracking*. Beck(1999 apud SATO 2006, p. 4) descreve o papel do *tracker* como alguém da equipe responsável por coletar métricas a partir dos dados obtidos e garantir que o grupo compreenda o que está sendo medido.

Hartmann e Dynind (2006 apud SATO, 2006, p. 44) sugerem outra categoria para classificação das métricas: as métricas organizacionais e métricas de diagnóstico, esta última também conhecida como métrica de acompanhamento. A métrica organizacional é responsável por medir a quantidade do valor de negócio que será entregue ao cliente e a métrica de acompanhamento gera informações que ajudam a equipe entender e melhorar o processo.

Lean (2003 apud SATO, 2006, p. 47) distingue bem as métricas organizacionais das métricas de acompanhamento nos princípios de desenvolvimento do software com o objetivo de "*otimizar o todo*" e, assim, propõe usar as métricas na avaliação de desempenho sempre um nível acima; ou seja, não utilizar as métricas para avaliar uma pessoa e sim toda a empresa, incentivando os indivíduos a trabalharem de forma colaborativa para atingir um resultado comum. Também sugere o uso de métricas de acompanhamento para auxiliar a equipe; tais métricas devem ser definidas ocultando o desempenho individual.

No mundo ágil existe a retrospectiva, que encoraja a discussão constante sobre o processo e a forma de trabalho da equipe. O time reúne-se com o objetivo de refletir sobre os processos mais aceitáveis pela equipe para o ciclo de inspeção e adaptação proposto pelos

métodos ágeis. Os resultados são publicados em um pôster, o qual contém os principais pontos de melhoria que serão observados na próxima iteração. A partir dessa reunião, o *tracker* deve escolher qual a melhor métrica de acompanhamento para apontar o progresso nas melhorias levantadas.

1.2.7.1. Características de uma boa métrica ágil

Hartman e Dymond (2006 apud SATO 2006, p. 49) propõem que um *tracker* deve priorizar as métricas ágeis que possuem as seguintes características: reforçar princípios ágeis, como: colaboração com o cliente e entrega de valor; medir resultados e não saídas e, assim, priorizar os resultados obtidos em vez de saídas da atividade do processo; seguir tendências e não números e, dessa forma, considerar que os valores representados por uma métrica são menos importantes que a tendência; preocupar-se mais com a estabilização do que o valor absoluto que ela representa.

Além das características apontadas acima, há outras que merecem destaque, como: responder uma única questão, toda métrica deve mostrar uma informação específica, se surgi outra pergunta é melhor usar outra métrica; pertencer a um conjunto pequeno de métricas e diagnósticos, já que muita informação pode esconder o que realmente importa; ser facilmente coletada nas métricas de acompanhamento - o ideal é ter uma coleta automatizada; revelar seu contexto e suas variáveis - uma métrica deve deixar evidente o fator que a influencia para evitar manipulações e facilitar a melhoria do processo. Incentivar a comunicação - a equipe comentando e aprendendo é um indicativo de uma boa métrica e evita que uma métrica isolada de seu contexto perca o sentido; fornecer *feedback* frequente e regular - essa prática amplifica o aprendizado e acelera o processo de melhoria e, finalmente, encorajar um alto nível de qualidade, sabendo que este deve ser definido pelo cliente e não pela equipe - os métodos ágeis exigem um alto nível de qualidade do *software* desenvolvido.

1.2.7.2. Algumas métricas ágeis.

Entre as métricas utilizadas pela equipe ágil destacam-se a métrica *Ideal Day* e a métrica *planning poker (letra maiúscula?)*. Essas métricas têm como respostas *story points,* medidas para cada ***Sprint.***

1.2.7.2.1.Ideal Day

Ideal Day é utilizada para realizar estimativas de forma ágil, aplicada para planejar o projeto e suas interações. "*Ideal Day* corresponde à quantidade de trabalho que um

profissional da área consegue concluir em um dia de trabalho" (ALVES; ALVES; FONSECA, 2008).

De acordo com Martins (2007 apud GAMBA; BARBOSA, 2010. p.3) este termo é a velocidade calculada a partir do numero de horas que a equipe gasta para programar um trabalho equivalente a um *Ideal Day*. Ainda segundo o mesmo autor, se o trabalho ultrapassar um dia de trabalho, é sugerido dividir esse item em partes menores e assim programar em apenas um dia. Para efetuar o cálculo dos dias estimados usa-se a fórmula:

DE= IED / 1-IED_REAL%

Onde:

DE: quantidade de dias estimados para concluir a tarefa;

IED: prazo necessário para programar o item, que é definido pela equipe;

IED_REAL%: percentual que indica a estimativa de quanto tempo do dia o desenvolvedor dedicará para a implantação do item.

1.2.7.2.2. Planning Poker

Planning poker é uma técnica de estimativa baseada no consenso de toda equipe. Para estimá-la é necessário o *product backlog* (todas as *user story* do projeto) um baralho de cartas específico que contém a sequência de *Fibonacci*, o qual será usado para representar dias, horas e tamanho da estória. Assim, em uma reunião, na qual cada equipe estará munida de seu baralho, é explicada a *user story*. Após isso, questões sobre a complexidade da tarefa a ser executada serão realizadas e respondidas por meio das cartas jogadas por cada membro do time suas opiniões.

1.2.8 Análise de Ponto de Função

Análise de ponto de função é uma técnica de medida da dimensão do software que utiliza a funcionalidade implementada em um sistema, sob o ponto de vista do usuário. O ponto de função é sua unidade de medida e pode ser obtida independente da tecnologia utilizada para a construção do *software*.

> "A Análise de Pontos de Função (APF) é um método-padrão para a medição do desenvolvimento de *software*, visando estabelecer uma medida de tamanho do software em Pontos de Função (PFs), com base na funcionalidade a ser implementada, sob o ponto de vista do usuário." (HAZAN, 2001)

Albert, Simões e Vazquez (2010) afirmam que pontos de função não medem diretamente o esforço, produtividade e custo do produto e sim sua funcionalidade. Entretanto, mesmo sendo uma medida exclusiva do tamanho funcional do software, em conjunto com outras variáveis pode se obter a produtividade, o custo e o esforço.

A Análise de Ponto de Função surgiu em 1979, por intermédio de um projeto desenvolvido por Allan Albrecht, pesquisador da IBM. Seu trabalho envolvia um estudo de produtividade para projetos de software, desenvolvidos por uma unidade de serviços da IBM. Como nem todos os projetos usavam a mesma linguagem de programação, ele buscou elaborar uma medida que observasse apenas aspectos externos do software. Baseando-se na visão do usuário, conseguiu estabelecer uma medida independente da linguagem de programação ou de qualquer outro aspecto relacionado à implementação do software.

> Um artigo publicado por Trevor Crossman em maio de 1979 descreveu uma abordagem funcional similar para medir a produtividade do programador. A sua abordagem define as funcionalidades baseadas na estrutura do programa. Nossa abordagem define as funcionalidades baseadas em atributos externos. Ele se concentrou em tarefas de projeto do programador, código e teste. Como você verá, nós olhamos todo o ciclo de desenvolvimento da aplicação, desde o sistema de projeto até o sistema de teste. Os nossos pontos de vista sobre a necessidade de uma abordagem funcional parecem estar de acordo. A medição baseada em funções tem provado ser um meio eficaz de comparar produtividade entre projetos. Antes dessa definição, nós só podíamos comparar projetos de linguagens e tecnologias semelhantes ou senão, tínhamos que encarar a dificuldade em comparar estimativas de projetos hipotéticos e resultados reais. Nós pretendemos continuar usando e aperfeiçoando a medição por valor da função. (ALBRECHT, 1979)

O BFPUG é a representação brasileira oficial (*Chapter)* do IFPUG – *International Function Point Users Group*, um grupo constituído com o objetivo de estimular e divulgar a utilização de métricas no desenvolvimento de sistemas, em particular a Análise de Pontos de Função – APF ou *Function Point Analysis* - FPA. Destina-se aos profissionais interessados em aprender, praticar e divulgar o uso de métricas e de FPA.

Albert, Simões e Vazquez (2010) relatam que no Brasil pode-se citar empresas como Accenture, Bradesco, Vale, Caixa, Correios, CPM Braxis, Datamec, Totvs, DBA, EDS, IBM, Petrobras, Politec, OI, Unisys, Xerox, dentre outras que utiliza a métrica Análise de Ponto de Função.

O IFPUG possui filiados em mais de 40 países; sendo que o uso da APF é mais intenso na Alemanha, Austrália, Brasil, Canadá, Coréia, Estados Unidos, Índia, Inglaterra,

Itália e Holanda. Exemplos de empresas no mundo que usam a APF são: IBM, Unisys, Xerox, EDS, Citigroup, Tata, Lockheed Martin-EIS, Booz Allen & Hamilton, Nielsen Media Research, Bank of Canada, Ralston Purina Co., Northrop Grumman Corp, Samsung SDS Co Ltd, BASF Corporation, Accenture, Pepsi Co, Compuware, Pricewaterhouse Cooper.

1.2.8.1.Benefícios da Análise de Ponto de Função.

Sobre os benefícios da análise de pontos de função, Dias (2004) observou:

> Dimensionamento dos sistemas, em produção e/ou em desenvolvimento bem como, solicitações de manutenção. Apoio para estimativa de custos e recursos requeridos para o desenvolvimento e manutenção de software. Apoio para gerenciamento da qualidade e produtividade no processo de desenvolvimento do software. Apoio para a tomada de decisão relativa à seleção para aquisição de pacotes, e contratação de serviços.

Albert, Simões e Vazquez (2010) ressaltam os requisitos sólidos e bem especificados facilita a contagem de ponto de função, o que não ocorre no mundo do desenvolvimento ágil. Eles citam como benefícios da APF, quais sejam:

- Uma ferramenta para determinar o tamanho de um pacote adquirido, através da contagem de todas as funções incluídas.
- Suporta a análise de produtividade e qualidade, seja diretamente ou em conjunto com outras métricas como esforço, defeitos e custo.
- Apóia o gerenciamento de escopo de projetos. Ao realizar estimativas e medições dos pontos de função do projeto em cada fase do seu ciclo de vida é possível determinar se os requisitos funcionais cresceram ou diminuíram e se esta variação corresponde a novos requisitos ou a requisitos já existentes e que foram apenas mais detalhados.
- Complementa o gerenciamento dos requisitos ao auxiliar na verificação da solidez e completeza dos requisitos especificados. O processo de contagem de pontos de função favorece uma análise sistemática e estruturada da especificação de requisitos e traz benefícios semelhantes a uma revisão em pares do mesmo.
- Um meio de estimar custo e recursos para o desenvolvimento e manutenção de software. Através da realização de uma contagem ou estimativa de pontos de função no início do ciclo de vida de um projeto de software, é possível

determinar seu tamanho funcional. Esta medida pode ser então utilizada como entrada para diversos modelos de estimativa de esforço, prazo e custo.

- Uma ferramenta para fundamentar a negociação de contratos. Pode-se utilizar pontos de função para gerar diversos indicadores de níveis de serviço em contratos de desenvolvimento e manutenção de sistemas. Além disso, permite o estabelecimento de contratos a preço unitário - pontos de função - onde a unidade representa um bem tangível para o cliente. Possibilitando uma melhor distribuição de riscos entre o cliente e o fornecedor.
- Um fator de normalização para comparação de software ou para a comparação da produtividade na utilização de diferentes técnicas. Diversas organizações, como o ISBSG, disponibilizam um repositório de dados de projetos de software que permitem a realização de benchmarking com projetos similares do mercado.

1.2.8.2.Algumas definições

Para a compreensão melhor da AFP, faz-se necessárias às definições abaixo especificadas:

Escopo da contagem define quais funcionalidades de uma ou mais aplicações serão incluídas em uma determinada contagem. Fronteira da aplicação é a interface conceitual que delimita o *software* que será medido e o mundo exterior.

Processo elementar é considerado a menor unidade de atividade que tem um significado para o usuário. Deve ser completo em si mesmo, independente, e deixar o negócio da aplicação em estado consistente.

Funções do tipo de dados representam as funcionalidades fornecidas pelo sistema aos usuários para atender as necessidades de armazenamento de dados. “A funcionalidade da aplicação é avaliada em termos do quê é fornecido pela mesma, não do como é fornecido. Apenas componentes definidos e solicitados pelo usuário devem ser contados” (GARMUS; HERRON, 2001 apud BARCELLOS, 2008 p.2).

As funções do tipo de dados são classificadas em: Arquivo Lógico Interno (ALI) são os dados ou informações mantidas pela aplicação a ser contada, ou seja, mantido dentro da fronteira da aplicação que está sendo controlada; Arquivo de Interface Externa (AIE) são os dados lidos de outra aplicação sendo identificável pelo usuário, mantido fora da fronteira da aplicação que está sendo controlada. “A diferença básica entre um ALI e um AIE é que o último não é mantido pela aplicação que está sendo contada. Um AIE contado para uma

aplicação sempre será contado como um ALI em sua aplicação de origem.” (BARCELLOS, 2008).

As funções do tipo transação representam as funcionalidades de processamento dos dados fornecidas pelo sistema ao usuário para atender as suas necessidades de processamentos de dados pela aplicação. Essas funções podem ser:

Entrada Externa (EE): processo elementar da aplicação que processam dados ou informações de controle que vêm de fora da fronteira da aplicação que está sendo controlada.

Saída Externa (SE): processo elementar da aplicação que geram dados ou informações de controle que são enviados para fora da fronteira da aplicação que está sendo controlada.

Consulta Externa (CE): processo elementar da aplicação que representa uma combinação de entrada e saída.

Cada entrada externa, saída externa e consulta externa possui dois tipos de elementos que devem ser contados para cada função identificada: Tipos de Dados (TD) e Arquivos Referenciados (AR).

Tipos de dados: campo único, reconhecido pelo usuário, não recursivo. Por exemplo: campos das tabelas.

> As seguintes regras devem ser válidas para a contagem de tipos de dados: Conte um tipo de dado para cada atributo que atravessa a fronteiras da aplicação (entrando e/ou saindo), reconhecido pelo usuário, único, não repetido. Conte um único tipo de dado para a capacidade de envio para fora da fronteira da aplicação de uma mensagem de resposta do sistema, indicando um erro verificando durante o processamento, a confirmação da sua conclusão ou a verificação do seu prosseguimento. Conte um tipo de dado para a capacidade especificar uma ação a ser tomada. Mesmo que haja múltiplos meios de ativar o mesmo processo, deve ser contado ser contado apenas um tipo de dado. (ALBERT; SIMOES; VAZQUEZ, 2010).

Arquivos Referenciados: arquivos lógicos utilizados para processar a entrada e/ou saída. É o total de ALI e AIE utilizados pela transação.

> As seguintes regras são válidas para contagem de um processo referenciado. As duas primeiras que tratam de atualização de arquivos, não são aplicáveis para consultas externas. Conte um arquivo referenciado para cada ALI mantido. Conte apenas um arquivo referenciado para cada ALI que seja tanto mantido quanto lido. Conte um arquivo referenciado para cada ALI ou AIE lido durante o processamento. Mesmo quer o ALI/AIE tenha vários tipos de registro, não deve contá-los mais de uma vez. Não conte arquivos que não são classificados como ALI ou AIE. Não conte o mesmo arquivo mais de

> uma vez, mesmo que a transação faça várias leituras ou atualizações nele. (ALBERT; SIMOES; VAZQUEZ, 2010).

O fator de ajuste influencia os pontos de função não ajustados em +/- 35%, obtendo-se o número de pontos de funções ajustados.

> Hoje o fator de ajuste não é parte do processo de medição submetido pelo IFPUG à ISO para fins de certificação de conformidade com a ISO/IEC 14133(Medição de Tamanho Funcional), contudo ainda é parte do Manual de Práticas de Contagem como um apêndice, e as regras de definições são objeto de questões na prova de certificação como em pontos de função. (ALBERT; SIMOES; VAZQUEZ, 2010).

1.3. Análise de Pontos de Função e Metodologias Ágeis

Oest (2011) descreve como fazer a relação entre pontos de função e metodologias ágeis. Para tanto, o processo de desenvolvimento deve ser dividido em quatro etapas:

Primeira etapa: é descrita a demanda em macro requisitos para que seja estimado prazo e custo de desenvolvimento. O cliente pode tomar decisões com estes dados.

Segunda etapa: a demanda que foi aprovada é detalhada em casos de uso, regras de negócios, glossário, requisitos não funcionais. As estórias do *product backlog* são substituídas por casos de uso. A demanda é recontada e estimado o prazo e o custo do *software.* O prazo define os *time-boxes* do projeto independente da mudança ou acréscimo de requisitos definindo a data do fim do projeto. São construídas todas as funcionalidades que cabem nos *time-boxes* estimado; algumas funcionalidades podem ficar de fora, dependendo da importância do item para o cliente. Por último, deve-se constituir em um projeto de evolução.

Terceira etapa: é a fase de execução, onde aplica-se o *Scrum* e algumas práticas do XP. Na reunião de planejamento da *release* o *product owner* apresenta o *product backlog,* a equipe, composta pelo *ScrumMaster* e o time, onde é definido a prioridade dos casos de usos. Na segunda parte da reunião, são realizadas as estimativas dos *story points* em uma sessão de *planning poker.* São analisados somente os casos de uso que estão contidos na *sprint*. Novas sessões são feitas nas reuniões de planejamento da *sprint* e os casos de usos são analisados de acordo com sua prioridade.

O valor do *story point* é relacionado com o esforço de desenvolvimento e não com o tamanho do *software*. Sendo uma métrica mais adequada para o *ScrumMaster* gerenciar a equipe como o auxílio do *burndown chart* durante as *sprints*.

Os *times-boxes* dos *sprints* possuem duração de trinta dias, onde são utilizadas práticas de integração contínua. As mudanças são permitidas somente no início de cada *sprint*. Nesta fase são realizadas análise, projeto, construção e teste; ou seja, os casos de uso serão desenvolvidos em um *sprint* que são escolhidas pelo product owner a partir de sua prioridade. O esforço de desenvolvimento e a velocidade da equipe são medidos em *story point* com o auxilio da métrica *planning poker*.

Quarta etapa: tem início no final da última *sprint*. A equipe de inspeção e as equipes de teste avaliam o *software* e os documentos gerados necessários. O *software* tem pequenas alterações solicitadas. É instalado o sistema e feita uma nova contagem de pontos de função.

Vazquez (2011), preocupado com as tendências do mercado, percebeu que nem sempre os requisitos podem ser estáveis. Assim, um equilíbrio entre as abordagens de desenvolvimento ágeis e algumas técnicas das metodologias tradicionais.

As mudanças de requisitos que pode existir durante o *time-boxes* do *product baclog* não são consideradas uns problemas na redefinição dos requisitos originais se forem tratadas por aqueles com o poder de tomar essas decisões e com a autoridade para isso em termos da hierarquia na divisão do trabalho e na definição de processos de negócio.

O conjunto mínimo de informações com o *baseline,* padrão oficial básico para os trabalhos subseqüentes, de acordo com os requerimentos para o processo de *software* operar antes de uma elaboração mais detalhada são:

- As necessidades e oportunidades de negócio identificadas.
- As restrições de ordem geral que afetam à entrega da solução proposta como um todo.
- O modelo ambiental com os papéis dos diferentes envolvidos que fornecem ou consomem informações da solução proposta.
- O modelo com os conceitos de negócio sobre os quais a solução proposta mantém dados.
- Os requisitos com regras claras que garantam a sua coesão e unidade de forma que possam ser verificados e validados.

Análise de Pontos de Função se propõe a realizar uma medição numa perspectiva de negócio e que não dependa de tecnologia ou metodologia utilizada no desenvolvimento da solução proposta.

A partir da definição desses requisitos nesse plano do negócio pode ser o insumo para estabelecer um estoque de itens a serem atacados em uma iniciativa que visa entregar

software mesmo que essa iniciativa tenha tempo fixo como o *time-box* e as iterações sejam curtas, planejadas diariamente em janelas de uma ou duas semanas.

Vazquez (2011) responde a pergunta: Como tratar a medição do valor agregado pelo projeto, que se utiliza uma abordagem iterativa e incremental? Ele observou que a resposta mais freqüente é a utilização das fases de acompanhamento gerencial previstas no *Rational Unified Process* (RUP), como: a Iniciação, Elaboração, Construção e Transição.

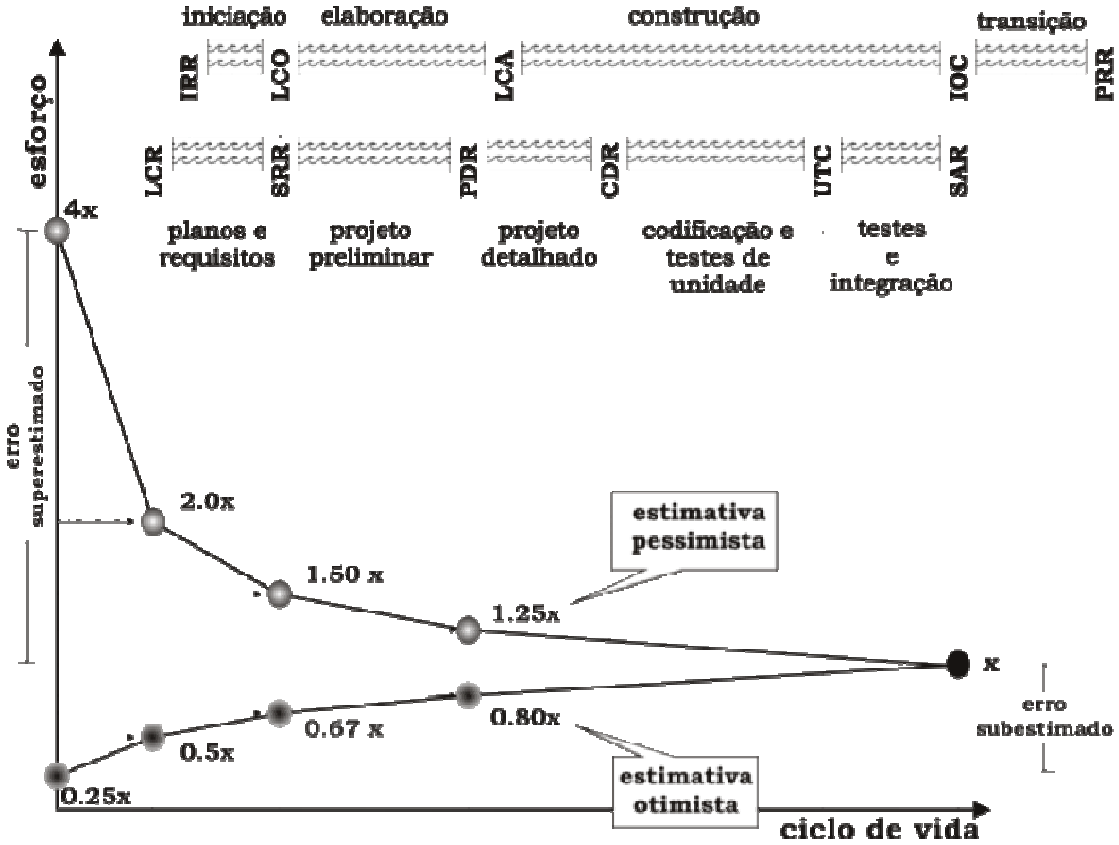

Figura 4: Marcos que defini as fases de acompanhamento gerencial de uma abordagem iterativo e incremental. Fonte: Vazquez, 2010.

O componente central para determinação do valor agregado por um projeto ao final de uma iteração é o Plano de Iteração. Onde uma matriz que possui nas linhas os diferentes pacotes com um conjunto de casos de uso ou estórias de usuário. E nas colunas as disciplinas da engenharia de software.

A idéia desse plano é a iteração em que cada disciplina será inicialmente aplicada em cada pacote. Quando um pacote de uma determinada disciplina não foi totalizado seus itens passam para o próximo pacote mudando o plano de iteração. Cada fase de iteração possui um valor agregado. Para isso, o primeiro passo é distribuir o esforço em termos relativos entre as fases e numa perspectiva do valor agregado por aquele esforço. Nos passos seguintes, vamos determinar o valor agregado enquanto o projeto está nas fases de elaboração e construção, a

fim de evitar que a contratada acabe por transformar o seu projeto originalmente desenhado para usar uma abordagem iterativa e incremental em uma cascata seqüencial.

Segundo passo, definir um *baseline* da estrutura do plano de interações. Uma *baseline* é uma 'imagem' de uma versão de cada artefato no repositório do projeto funcionando como um padrão oficial básico para os trabalhos subseqüentes sendo modificado somente quando autorizado. Deve estabelecer e definir tipos de pacotes de trabalho que estruturam as colunas do plano de iteração. Essas colunas são requisitos, análise e projeto, construção, teste de unidade e integração. Determinar um valor agregado relativo a porcentagem por tipo de pacote de trabalho. E determinar os pacotes de função.

O terceiro passo é apurar o valor agregado ao final de uma iteração e o percentual acumulado pelo projeto como um todo.

Vazquez (2011) descreve de forma estruturada os nomes para calcular suas formulas:

> 1. A matriz com a contribuição de cada fase em relação ao projeto total; chamemos os elementos dessa matriz "% Fase". Chamemos %Agregado Iniciação e %Agregado Transição; para fins de avaliação da contribuição entre as iterações vamos considerar que seja todo o restante necessário à totalização dos 100% do projeto
> 2. Temos o conjunto de entregáveis (documento de visão, lista de requisitos, especificação de casos de uso, protótipo funcionais, especificação complementar de requisitos, especificação de arquitetura, programas fonte, casos de teste, relatórios de teste, etc.). Chamemos essa variável de "Pacote de Trabalho" (as colunas do plano de iteração);
> 3. Temos um conjunto coeso de funções (cadastro de clientes, cadastro de fornecedores, emissão de relatórios de consolidação, etc.). Chamemos essa variável de "Pacote de Funções" (as linhas do plano de iteração);
> 4. Os "Pacotes de Trabalho" devem ser aceitos para que o exercício de determinadas disciplinas (requisitos, análise e projeto, implementação, etc.) sobre um "Pacote de Funções" sejam consideradas concluídas (ainda que possam haver mudanças em função de novas descobertas subseqüentes e dai a importância da reserva como contingência para esse risco). Chamemos essa variável de "%Contingência"
> 5. Funcionalmente dependente de cada "Pacote de Trabalho", temos o percentual agregado por uma iteração. Chamemos essa variável de "$\%Agregado_{\text{pacote de trabalho}}$";
> 6. Funcionalmente dependente de cada "Pacote de Função" , temos a quantidade de pontos de função medidos ou estimados. Chamemos essa variável de "$PF_{\text{pacote de função}}$";
> 7. Indicando em qual iteração houve o aceite de determinado "Pacote de Trabalho" de determinado "Pacote de Função", temos no coração do plano de iteração um número que começa a partir de 01, representando a 1ª iteração; 02, representando a 2ª iteração; e assim por diante. Chamemos esse número de "*#iteração*".

- Para calcular a porcentagem agregada da fase da construção e elaboração que determina o iteração representa utilize a formula:

$$\%Agregado_{\#Iteração} = \frac{\sum_{j=1}^{\#Pacotes\ de\ Função} \sum_{i=1}^{\#Tipos\ de\ Pacote} \%Agregado_i \times PF_j}{\sum_{k=1}^{\#Pacotes\ de\ Função} PF_k}, Planto\ de\ Iteração_{ji} = \#iteração$$

Figura 4: Formula para calcular porcentagem da fase de construção e elaboração da iteração. Fonte:Vazquez, 2011.

- Para calcular a porcentagem agregado acumulado até determinada iteração utilize a formula:

$$\%Agregado\ Acumulado_{\#iteração} = \sum_{i=1}^{\#iteração} \%Agregado_i$$

Figura 5: Formula para calcular o percentual agregado até determinada iteração. Fonte: Vazquez, 2011.

- Para calcular a porcentagem do total do projeto acumulado utilize a formula:

$$\%Agregado\ Acumulado\ Projeto = \%Agregado_{iniciação} + [\%Agregado\ Acumulado_{\#interação} \times (1 - (\%Agregado_{iniciação} + \%Agregado_{transição}))]$$

Figura 6: Formula para calcular a porcentagem do total do projeto acumulado. Fonte: Vazquez, 2011.

Oest (2011 apud Vasquez 2011) apresenta uma visão diferente sobre o como usar processo iterativo e incremental em um contrato de fábrica de *software.* Para determinar o valor do *software*, é calculado o tamanho funcional da demanda do mesmo. É definida a produtividade média de horas por ponto de função (HH/PF) que se pode alcançar no projeto a partir da característica da arquitetura definida para ele e outros requisitos não funcionais. Multiplica-se o tamanho do ponto de função (PF) pela produtividade (HH/PF), obtendo o total de horas (HH) necessárias para o projeto e após o resultado pelo valor contratado.

Nesse capítulo, foi apresentada características da metodologia ágil *scrum*. Essas características mostram uma metodologia versátil, que pode ser considerada um *framework*, o qual permite adaptar suas regras em cada equipe de desenvolvimento, buscando uma qualidade no processo de desenvolvimento do *software*. As métricas também foram abordadas, demonstrando a sua importância para um projeto, diminuindo riscos da entrega do mesmo. Não existe uma métrica padrão quando o assunto é *software*, contudo, a métrica análise de ponto de função é a mais difundida no mercado.

CAPÍTULO 2 - Metodologia da Pesquisa

Primeiramente, tem-se que apresentar a conceituação do objeto deste trabalho: a pesquisa. Lakatos e Marconi (2001) definem a pesquisa como uma atividade voltada à busca de respostas e à solução de problemas para questões propostas, através da utilização de métodos científicos. Portanto, torna-se necessária a delimitação de um problema e, se problema científico, consequente pesquisa científica. Neste capitulo far-se-á uma associação entre o problema pesquisado e o tipo de pesquisa utilizada, apresentando os meios, as etapas e os instrumentos utilizados.

2.1. Tipos da pesquisa

A pesquisa será desenvolvida em torno da seguinte questão: como a métrica análise de ponto de função pode ser utilizada para medir um projeto de desenvolvimento ágil, já que os requisitos são essenciais para contagem. As seguintes perguntas nortearão a pesquisa: O que medir no *software*? Quais são as melhores métricas para um projeto ágil? O que medir em um desenvolvimento ágil? Ponto de função consegue atender as metodologias ágeis?

Na prática, a utilização de Análise de Ponto de Função em Metodologias Ágeis ainda se encontra em estágio inicial e o relato de experiências bem como estudos detalhados do assunto é ainda incipientes, o que justifica a abordagem de tema através de uma pesquisa de caráter exploratório.

Abordaremos o tema através de uma pesquisa de caráter exploratório quantos aos fins e bibliográfico quanto aos meios de acordo com Vergara(2007). Lakatos e Marconi (1985) apresentam a pesquisa exploratória como um grupo componente de pesquisa de campo e citam algumas finalidades da mesma: aumentar a familiaridade do pesquisador com um ambiente. Trata-se, portanto, de uma modalidade de pesquisa usada quando não existe qualquer trabalho científico produzido anteriormente sobre o tema.

A pesquisa exploratória é utilizada para casos em que, por falta de familiaridade com o problema de pesquisa, necessite-se de um estudo que oriente a direção a ser seguida, como em alguns casos específicos dentro de uma organização, embora muitas vezes possam existir teorias e conhecimentos a respeito do tema em questão.

A pesquisa bibliográfica abrange toda a bibliografia já tornada pública em relação ao tema de estudo, desde publicações avulsas, boletins, jornais, revistas, livros, pesquisas, monografias, teses.

2.2. Etapas da pesquisa

As etapas para construção do trabalho foram, primeiramente, a escolha do tema, bem como sua delimitação. Na etapa seguinte procedeu-se à elaboração do Projeto de Conclusão de Curso contemplando o cronograma a ser seguido durante a pesquisa, constante do apêndice A deste trabalho.

Levantamento bibliográfico no primeiro momento vai ser abordado sobre as metodologias ágeis utilizando o *scrum* como modelo estudado. Outro assunto abordado foi as métricas dando ênfase para métrica Análise de Ponto de função.

Também constituiu-se em etapa desta pesquisa a elaboração de um pôster (Apêndice B) exposto no III Simpósio de Tecnologia da Informação e III Semana de Iniciação Científica do Curso Sistema de Informação UNUCET-UEG/2011.

2.3. Instrumentos utilizados

Os instrumentos utilizados serão livros sobre análise de ponto de função e *scrum*, relato da experiência desenvolvida na empresa Petrobrás por Carlos Oest, opiniões de desenvolvedores da proposta de estudo através de listas de discussão e, em particular, a proposta de estudo de Carlos Eduardo Vaquez sobre o uso de análise de ponto de função para gerenciamento de *software* iterativo e incremental.

Com as informações adquiridas com esses estudos será proposta uma contagem de pontos de função através das estórias dos usuários de um *produckt baclog*.

Neste capitulo fala sobre o caminho que será percorrido para desenvolver a monografia. Descrevendo as etapas e a forma que as informações foram obtidas, tratadas e classificadas.

CAPÍTULO 3 – *Análise de Ponto de Função em Metodologias Ágeis*

A estimativa *de software* por Análise de Ponto de Função é uma técnica indepedente da tecnologia e da metodologia utilizada. Seu cálculo é baseado nos requisitos funcionais do sistema. As metodologias tradicionais os requisitos são muito bem detalhados, previsto e estáveis, análise de ponto de função é uma métrica bastante difundida nesse meio.

O Tribunal de Contas da União (TCU) tem se pronunciado, por diversas vezes, apontando o Ponto de Função (PF) como sendo a unidade mais adequada para o entendimento do tamanho de qualquer sistema de informação.

> Diversas instituições, públicas e privadas, têm utilizado a métrica de Pontos de Função (PF) nas estimativas e dimensionamento de tamanho funcional de projetos de software, devido aos diversos benefícios de utilização da métrica, destacando: regras de contagem objetivas e independência da solução tecnológica utilizada. É importante ressaltar que a Instrução Normativa IN04 SLTI/MPOG 2010 recomenda o uso de métricas em soluções de software, restringindo o uso da métrica de esforço homem-hora. Os Acórdãos do Tribunal de Contas da União (TCU) recomendam a utilização da métrica Pontos de Função Não Ajustados em contratos de prestação de serviços de desenvolvimento e manutenção de sistemas. (Roteiro de Métricas de Software do SISP, 2010)

As metodologias ágeis têm um escopo mais dinâmico, permitindo mudanças nos requisitos do sitema. O *Scrum* é descrito por *Ken Schwaber* como um *framework* ágil, o que permite adaptá-lo para cada processo específico. O S*crumMaster* ou o *product owner* utiliza a análise de ponto de função se agregar algum valor ao seu processo. A *Análise de Ponto de Função* a métrica mais difundida no mercado atual, vamos correlacionar essas duas técnicas para conviverem em harmonia.

Na Petrobrás, os pontos de função não são contados a cada iteração, sendo contada somente no início e no final de cada *product backlog.* Já Vazquez (2011) considera que os pontos de funções também têm que ser realizados em cada *sprint* finalizada. Vamos utilizar a contagem de pontos de função no início do *product backlog* quando assumido essa contagem como premissas, que serão corrigidas ao final da vida útil do *product backlog* com uma nova contagem do produto.

3.1. Diferenças entre Análise de Pontos de Função e *Story Points*

Story point é uma métrica relativa onde o cálculo é baseado em razões, não em números absolutos. Através do conhecimento empírico, tem-se informação de velocidade e ou

esforço de equipes, mas sem nenhum parâmetro formal, tornando-se uma medida particular e subjetiva de cada time.

O time *scrum* vai mensurar a quantidade de esforço para implementar uma estória. A medida dessa estória pode ser diferente para cada equipe. Nesse método, cada membro da equipe expõe sua opinião de acordo com seu conhecimento empírico. Contudo, após discutir o assunto com a equipe, poderá mudar sua posição, com o objetivo de chegar a um senso comum.

A análise de ponto de função é uma métrica funcional, onde se mede a funcionalidade do *software* e o esforço não entra no processo de contagem. Ela possui regras e definições comuns para qualquer equipe de desenvolvimento, o que diminui as subjetividades e as interpretações sobre essa medida. Oferece uma medida objetiva e comparável que auxilia na avaliação, no planejamento, na gerência e no controle da produção do *software*.

3.2. Estimando estória do usuário.

Na estimativa de duração de uma estória serão utilizadas técnicas de métricas ágeis. Vazquez (2011) propõe que as estimativas sejam feitas em cada *sprint* em vez de estórias. Oest (2011) estima os casos de uso que vão substituir as estórias do *product backlog*. O valor do *story point* encontrado nesses casos de uso, é relacionado com o esforço de desenvolvimento e não com o tamanho do *software*. São realizadas as estimativas dos *story points* em uma sessão de *planning poker,* na qual são analisados somente os casos de uso que contem na *sprint*, suprimi de acordo com sua prioridade.

A métrica *ideal days* é baseada na duração de tarefas onde dias ou horas é unidade bem definida. Contudo o "tempo ideal" quase nunca é igual ao "tempo real". Esta métrica é mais fácil de estimar, mas pode tornar-se complicada se considerarmos todas as interrupções e variações. A métrica *planning poker* obtém valores relativos e mais abstratos, baseando-se no tamanho da estória, que é influenciada pelo nível de dificuldade e complexidade da mesma e a experiência da equipe.

Dessa forma, proponho que as estórias de usuários sejam contadas por ponto de função, mas, neste caso, o time deve realizar estimativas de duração de *sprints* com a métrica *planning poker.* Além da codificação da estória, outros aspectos também devem ser levados em consideração, como por exemplo: realização do teste unitário, preparação do ambiente de teste e outras necessidades que sejam relevantes para a entrega do *software* funcionando. As estórias do usuário devem herdar o nível de prioridade do *product backlog.*

3.3. Quem deve contar os pontos de função?

Schawaber e Sutherland (2010) sugerem que as medições de uma equipe ágil devem ser realizadas em reuniões com toda a equipe que utiliza a técnica *planning poker* para se estimar a velocidade. Essa reunião deve ser gerenciada pelo *ScrumMaster.* O *ScrumMaster* não é considerado um gerente de Tecnologia da Informação (TI), mas deve garantir que as práticas do *scrum* sejam seguidas e aperfeiçoadas quando preciso, sendo assim um facilitador quando surgir algum empecilho externo. Ele vai gerenciar somente seu time, sendo que poderá existir mais de um time *scrum* para um *product backlog.*

Como Vazquez (2010) um dos beneficios do ponto de função quando utilizados com outras métricas um exemplo é a estimativas de produtividade de uma equipe. Sugiro que o cálculo da velocidade seja feito pelo *ScrumMaster* para avaliar os processos a serem utilizados durante o desenvolvimento. Processos de produção da equipe, o que geralmente faz parte da rotina do *ScrumMaster*, uma vez que este mantém contato direto com o grupo o tempo todo. Lembrando que o que deve ser avaliado sempre é o método e não as pessoas.

Schawaber e Sutherland (2010) escreve como papel do *product owner* o de representar os interesses do cliente, criando o *product backlog* e priorizando os itens com maior valor de interesse para este. Para tanto, ele deve conhecer os processos realizados na empresa e tem que estar presente durante reuniões de *review de sprints* para fornecer *feedback.*

Assim, sugiro que a contagem de ponto de função seja realizada pelas estórias do *product backlog*, o *product owner* responsável pela sua criação, deve realizar a contagem de pontos de função. E para que essa contagem seja maximizada, o valor de seu uso deve ser compartilhado com os *stakeholders* do *software*.

3.4. O procedimento de contagem de Ponto de Função:

Como o objeto dessa pesquisa ainda é insipiente, especialmente no Brasil, considerou-se a importância de analisar um caso completo: a experiência relatada por Oest (2011), a qual realiza a contagem no início e no final de cada *product backlog* de um sistema. Os seus *product backlog* são compostos por casos de uso, substituindo as estórias de usuários, onde são retirados os requisitos funcionais e feita a contagem dos pontos de função.

O escopo do sistema é aberto com *sprint* fixa de trinta dias, sendo implementados os casos de uso com maior valor agregado para o cliente. O que não foi executado na *sprint,* será devolvido no *product backlog,* onde será novamente avaliado o valor de seus itens até a última iteração. Entretanto, a criação de casos de uso e eliminação das estórias do *product backlog* vai contra as técnicas de desenvolvimento ágil. As estórias do usuário devem possuir

uma linguagem sem jargões técnicos e clara, para todos os *stakeholders* do sistema. Além disso, serão escritas com mais detalhes, podendo subtrair os requisitos funcionais para o processo de contagem do ponto de função.

Vazquez (2011) conclui, por estudo detalhado, um enfoque diferente, no sentido de que cada *sprint* entregue para o cliente tenha sua contagem de ponto de função com o valor nas *sprints* agregado, de acordo com seguinte passos: Deve-se agregar porcentagens das fases de elaboração e construção representantes do total do projeto; definir um *baseline* da estrutura de iterações e apurar o valor agregado ao final de uma iteração.

Diante disso, é essencial ressaltar que este processo proposto para contagem de pontos de função, foi realizado em uma época em que as empresas tinham dois tipos de processos para produção de *software*: nas metodologias tradicionais, onde os requisitos são fixos e previstos, ou em empresas com processos de planejamento de produção inexistentes como métricas ou metodologias.

A contagem neste trabalho será realizada através das estórias de usuários, sendo que em algumas etapas do processo serão acrescentados no *product backlog* uma forma de subtração das informações necessárias, reservando o processo original de contagem de pontos de funções, que tem suas definições e regras claras. Assim, a contagem envolverá sete passos bem definidos, os quais estão representados a seguir:

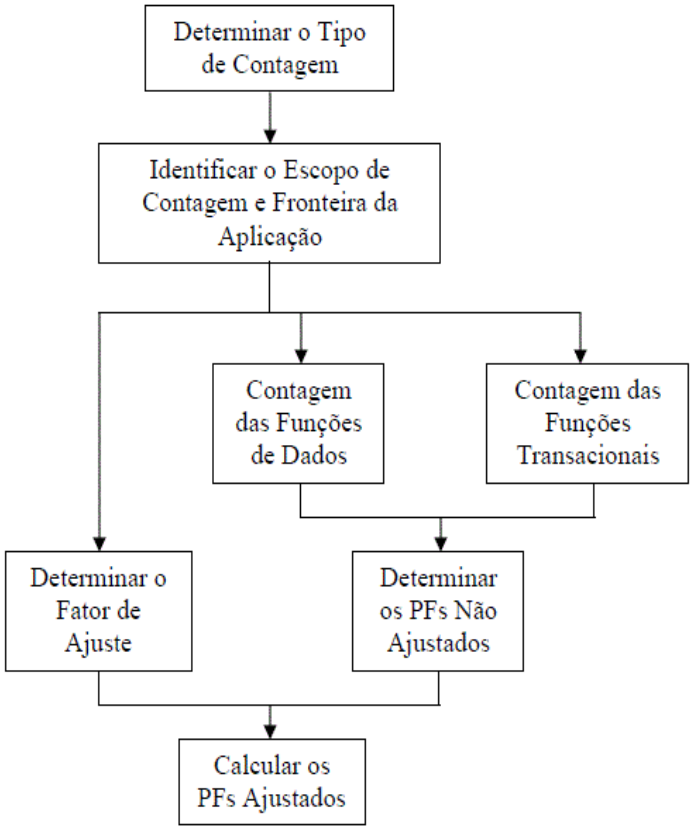

Figura 7: Passos do processo de contagem de ponto de função. Fonte: Hazan 2001.

3.4.1. Determinar o tipo de contagem

Neste passo é estabelecido o tipo de contagem que será usado para medir o projeto de *software*, tanto no processo como no produto, sendo três tipos possíveis:

Projeto de Desenvolvimento: o número de pontos de função de um projeto de desenvolvimento mede a funcionalidade fornecida aos usuários finais quando da primeira instalação do *software* entregue. Este tipo de contagem também abrange as funções de conversão de dados que serão precisas para a implantação do software. Como exemplo de função de conversão de dados pode-se citar a necessidade de importar dados de um sistema.

Projeto de Melhoria ou Manutenção: o número de pontos de função mede as modificações para uma aplicação já existente, ou seja, as funções adicionais, modificadas ou excluídas do sistema pelo projeto e as funções de conversões de dados. Após a conclusão e implantação deste projeto, o número de pontos de função da aplicação deve ser atualizado para refletir as mudanças nas funcionalidades da aplicação.

Projeto de Aplicação: a contagem de pontos de função refere-se a uma aplicação já instalada que mede a funcionalidade fornecida ao usuário, provendo uma medida da atual

funcionalidade ganhada pelo usuário da aplicação. Ela é iniciada ao final da contagem do Projeto de Desenvolvimento e atualizada no final do Projeto de Melhoria.

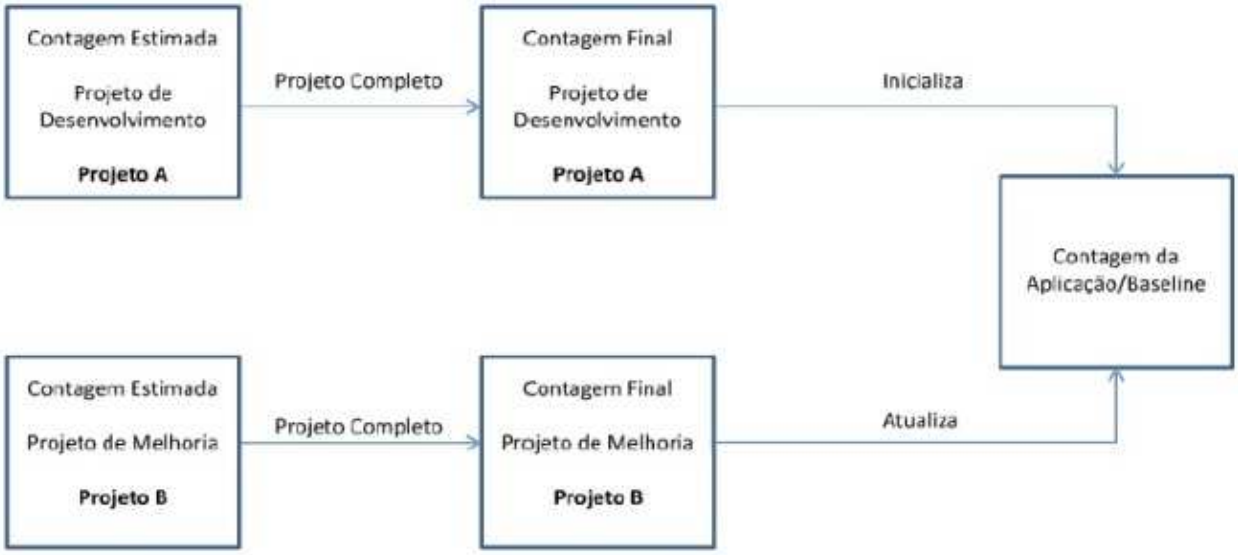

Figura 8: Relacionamento entre os tipos de contagem. Fonte :Albert; Simoes; Vazquez, 2010.

Dessa forma, proponho que o *product owner* realize a contagem de desenvolvimento através das estórias contidas no *product backlog*. Essa contagem vai ser realizada antes da reunião de planejamento da *realise,* e deve levar em consideração a velocidade da equipe em projetos anteriores. Não serão realizadas contagens entre as sprints. O escopo do *software* deve ser aberto, permitindo alterações ou acréscimo das estórias no início de cada *sprint*. No final do ciclo de vida do *product backlog*, ou seja, no final do projeto será realizada a contagem da aplicação.

3.4.2. Identificar o Escopo de Contagem e Fronteira da Aplicação

Neste passo define as funcionalidades que serão incluídas em uma contagem de pontos de função específicas: o escopo e a fronteira da aplicação. A fronteira é definida, estabelecendo um limite lógico entre a aplicação que está sendo medida, o usuário e outras aplicações. O escopo de contagem define a parte do sistema a ser contada.

Os requisitos do *software* são fundamentais para a contagem de pontos de função, pois o processo de medição é baseado exclusivamente neles. Não existem documentos específicos de uso obrigatório para contar pontos de função. Qualquer documento que possa extrair informações dos requisitos do produto pode ser usado. O objetivo da *Análise de Ponto de Função* é medir a funcionalidade do *software* independente da implementação, método ou tecnologia utilizada.

O *product backlog* é uma lista priorizada de tudo que pode ser necessário no produto. Ele é composto por estórias de usuários os requisitos do desenvolvimento ágil. Os documentos que contém os requisitos explícitos em sua linguagem de negócios podem facilitar a contagem dos pontos de função. Assim, com um detalhamento das estórias, pode-se visualizar melhor o requisito.

A fronteira da aplicação separa a aplicação que está sendo contada das aplicações externas, indicando o limite lógico entre elas. Ao identificar a fronteira da aplicação, deve-se estabelecer o relacionamento entre os processos, com indicação de onde eles ocorrem e os limites entre as funções a serem atendidas pelo projeto dimensionado e àquelas pertencentes ao ambiente externo.

Segundo o IFPUG, significado e lista de sigla, para identificar a fronteira do processo de contagem, devem ser adotadas as seguintes regras: considerar o ponto de vista do usuário, ou seja, o que o usuário pode entender e descrever como função da aplicação; a fronteira entre aplicações relacionadas deve considerar a funcionalidade das aplicações em termos das funções de negócio identificadas pelo usuário, e não sob o ponto de vista das interfaces necessárias**.**

A identificação da fronteira da aplicação é um dos passos mais importantes, pois se o posicionamento da mesma for feita de maneira incorreta, poderá haver uma contagem incorreta dos pontos de função, comprometendo o processo.

3.4.3. Contagem das Funções de Dados

As funções de dados representam as funcionalidades relativas aos requisitos de dados internos e externos à aplicação. O termo arquivo não significa um arquivo físico no sentido tradicional de processamento de dados, mas refere-se a um grupo de dados logicamente relacionados e reconhecidos pelo usuário.

Das estórias do usuário, quando especificadas, podemos subtrair informações como as funções de dados, tendo o cuidado de não gerar documentação desnecessária, evitando que a criação do *product backlog* seja estendida, o que iria contra os principios ágeis. As Funções de Dados são classificadas em arquivos lógicos internos (ALI) e arquivos de Interface Externa (AIE).

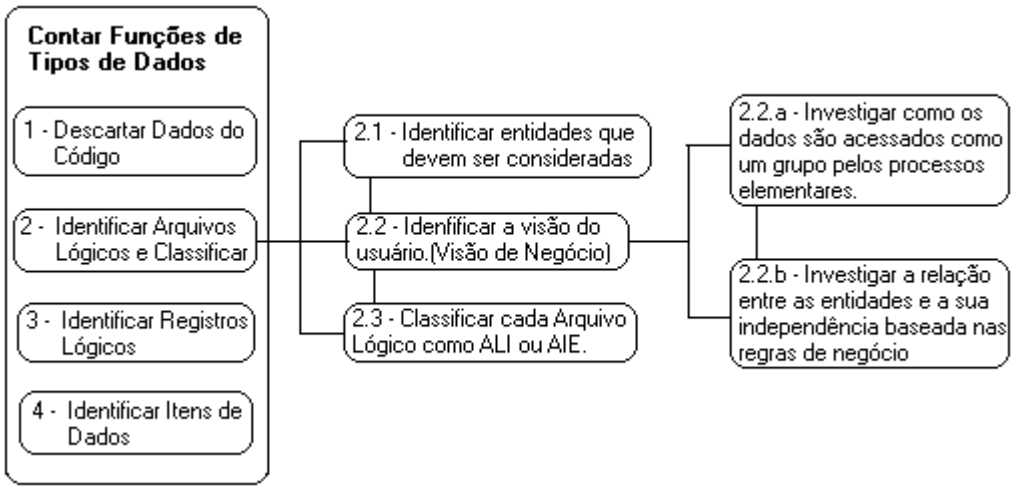

Figura 9: Abordagem prática de contagem ALI e AIE. Fonte: Albert; Simões; Vazquez, 2010.

Arquivo lógico interno tem como sua função principal armazenar dados mantidos dentro da fronteira da aplicação. São exemplos deste tipo de arquivo: 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. Não são considerados como Arquivo Lógico Interno: arquivos temporários; arquivos de trabalho; arquivos de classificação; arquivos de cópia de segurança requerido pelo CPD; arquivos introduzidos somente por causa da tecnologia usada; operações de junção e projeção; arquivos de índices alternativos.

Arquivos de interface externa são os dados lidos de outra aplicação, sendo identificável pelo usuário, mantido fora da fronteira da aplicação que está sendo controlada. Um arquivo de Interface Externa de uma aplicação sempre será contado como um ALI na aplicação de origem. Exemplos deste tipo de arquivo são: arquivos de mensagens de auxílio; arquivos de mensagens de erro. Não são considerados arquivo de interface externa: dados recebidos de outra aplicação usados para adicionar, alterar ou remover dados em um arquivo de lógica interna; dados cuja manutenção é feita pela aplicação que está sendo avaliada, mas que são acessados e utilizados por outra aplicação; dados formatados e processados para uso por outra aplicação.

Cada arquivo lógico interno e arquivo de interface externa possuem dois tipos de elementos que devem ser contados para cada função identificada, quais sejam:

Tipos de dados: é um campo único, reconhecido pelo usuário, não recursivo. Na citação abaixo é possível verificar as regras válidas para que um determinado campo seja contado como um tipo de dado:

> Conte um tipo de dado para campo único reconhecido pelo usuário e não repetido, mantido ou recuperado de um ALI ou AIE por meio da execução de um processo elementar. Quando duas aplicações mantêm ou referenciam o mesmo ALI/AIE, conte apenas os campos utilizados pela aplicação em análise. Conte um tipo de dado para cada campo solicitado pelo usuário para estabelecer um relacionamento com outro arquivo lógico. (Albert; Simões; Vazquez, 2010)

Tipos de registros: é um subgrupo de dados, reconhecido pelo usuário, componente de um arquivo lógico interno e arquivo de interface externa. Existem dois tipos de subgrupo: Opcionais, aqueles em que o usuário tem a opção de não informar no processo elementar que cria ou adiciona dados ao arquivo e Obrigatórios, quando o usuário requer que sejam sempre utilizados pelo processo elementar que cria ou adiciona dados ao arquivo.

> As seguintes regras devem ser utilizadas para determinar o numero de tipos de registro de um Arquivo Lógico Interno ou Arquivo de interface Externa. Conte um TR para cada função de dados (isto é, por default, cada função de dados tem um subgrupo de TD a ser contado com um TR). Conte um TR adicional para cada um dos seguintes subgrupos lógicos de TDs (compreendido na função de dados) que contém mais de um TD: Entidade associativa com atributos não chave; Subtipo (outro que não o primeiro subtipo); Entidade atributiva em um relacionamento que não seja mandatório. (Albert; Simões; Vazquez, 2010)

Tabela1: Tabela de complexidade funcional dos ALI e AIE.

Número de Registro Lógicos	Tipos de Dados		
	< 20	20 – 50	>50
1	Baixa	Baixa	**Média**
2 – 5	Baixa	**Média**	Alta
> 5	**Média**	Alta	Alta

3.4.4. Contagem das Funções Transacionais

As funções transacionais representam as funcionalidades fornecidas ao usuário para atender suas necessidades de processamento de dados pela aplicação. São classificadas em entradas externas, saídas externas e consultas externas.

Entradas externas é o processo elementar que afere dados ou informações de controle recebidos de fora da fronteira da aplicação. Onde processam dados ou informações de controle que entram pela fronteira da aplicação. Esses dados, através de um processo lógico único, atualizam os arquivos lógicos internos. Uma entrada externa é considerada única para uma aplicação se possuir um formato diferente das demais ou se precisar de uma lógica de processamento diferente de outras entradas externas que tenham o mesmo formato. São exemplos de entradas externas: operações de inclusões e alterações de registros em arquivos da aplicação; janela que permite adicionar, excluir e alterar registros em arquivos. Não são exemplos: menus, telas de *login*, telas de filtro de relatórios e consultas; múltiplos métodos de se executar uma mesma lógica de entrada.

Para que o processo elementar identificado seja contado como uma entrada externa, pelo menos umas das três opções a seguir devem ser satisfeitas: a lógica de processamento deve ser única e diferente das demais entradas externas; o conjunto de dados elementares identificados é distinto dos conjuntos identificados por outras entradas externas; os arquivos lógicos internos mantidos e os arquivos de interface externa referenciados são distintos dos utilizados por outras entradas externas.

Saída externa é um processo elementar que envia dados ou informações de controle para fora da fronteira da aplicação e tem o objetivo de exibir informações recuperadas através de processamento lógico, ou seja, processamento que envolva cálculos ou criação de dados derivados e não apenas uma simples recuperação de dados. Uma saída externa pode manter um arquivo lógico interno ou alterar o comportamento do sistema.

O processamento lógico é definido como o conjunto de críticas, cálculos, algoritmos, referência e acesso a arquivos requisitados pelo usuário que visa completar um processo elementar.

Exemplos de saída externa: Relatórios, cada relatório produzido pela aplicação; Para relatórios de formato idênticos mas que necessitam de lógicas de processamento ou cálculos distintos devem ser considerados duas saídas externas; Relatórios *on-line*, saída de dados on-line que não seja a parte de saída de uma consulta externa; Formatos gráficos cada formato gráfico diferente é contado como uma saída externa; Gerador de relatórios. Não devem ser considerados como saídas externas: telas de ajuda, literais, data, hora, controles de paginação; Relatórios múltiplos com a mesma lógica e formato.

Consulta externa é o processamento de consultas da aplicação, sendo uma combinação de entrada e saída de dados. Onde as entradas de dados causa uma recuperação e saída de dados correspondente. A lógica de processamento não deve conter fórmula matemática ou cálculo, nem criar dados derivados ou atualizar nenhum arquivo lógico interno. Exemplos deste tipo de consulta são: telas de *login,* telas de *help*; telas de alteração e remoção, que mostram o que será alterado ou removido antes de sua efetivação; tela de menus, que permitem informar parâmetros para a consulta na tela escolhida. Não são consideras consulta externa: telas de menus que oferecem somente funcionalidade de seleção de telas; dados derivados; documentação on-line; sistema de teste; sistemas tutoriais; relatórios e consultas que contenham cálculo ou gerem dados derivados.

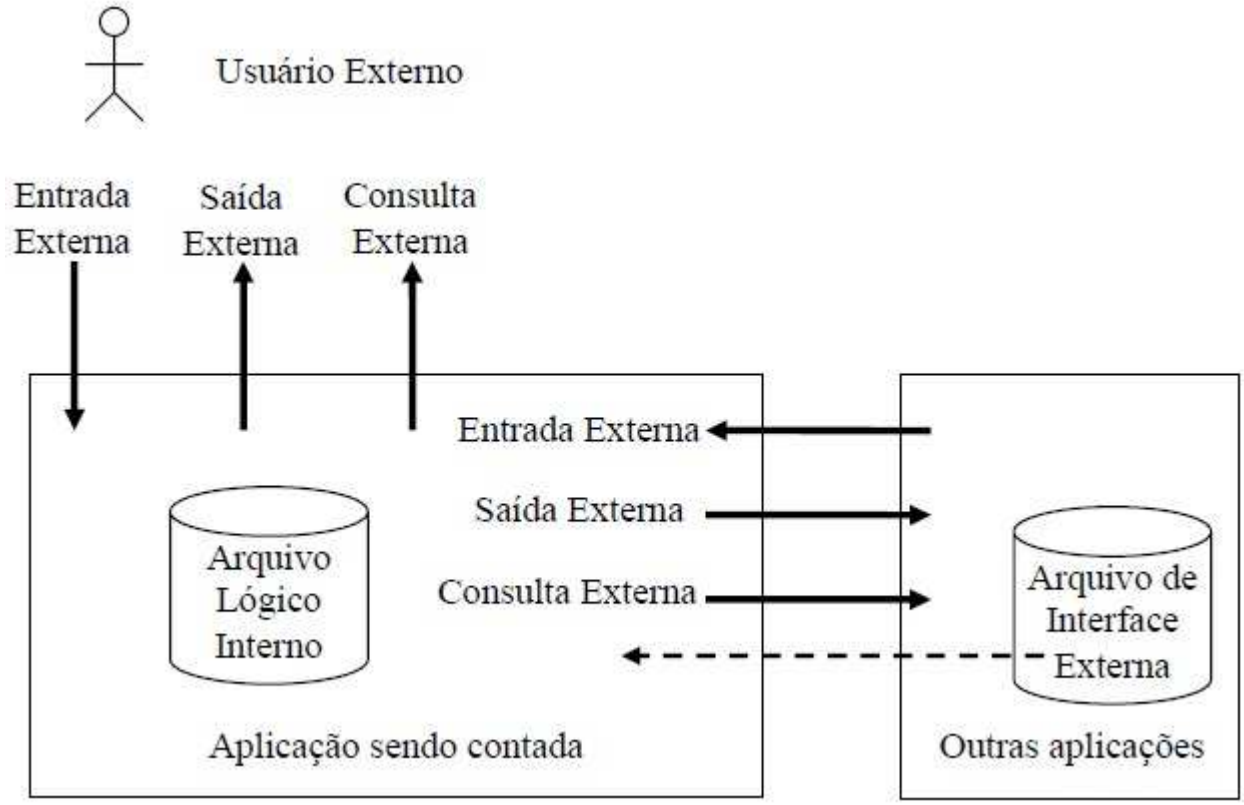

Figura 10: Visão Geral das Funções de transações em uma Aplicação segundo APF Fonte: Hazan, 2001.

Tabela 2: Tabela de complexidade para entradas externas (EE).

Arquivos Referenciados	Tipo de Dados		
	< 5	5 – 15	>15
< 2	Baixa	Baixa	**Média**

2	Baixa	**Média**	Alta
>2	**Média**	Alta	Alta

Tabela 3: Tabela de complexidade para saída externa (SE) e consulta externa (CE).

Arquivos Referenciados	Tipo de Dados		
	< 6	6 – 19	>19
< 2	Baixa	Baixa	**Média**
2 – 3	Baixa	**Média**	Alta
>3	**Média**	Alta	Alta

3.4.5. Determinar os pontos de funções não ajustados

Uma vez contadas as funções de dados e as funções transacionais, é possível calcular os pontos de funções não ajustados de uma aplicação. O número de pontos de função não ajustado de um sistema reflete a funcionalidade que este fornecerá ao usuário, sem considerar suas especificidades.

As funcionalidades seriam as mesmas, o que resultaria na mesma contagem de pontos de função não ajustados, mas quando considera as características do sistema para cada cliente, observa-se que os pontos de função devem ser ajustados para refletir a maior complexidade do sistema na arquitetura cliente servidor.

Após determinar o número de arquivo lógico interno e arquivo de interface externa, calcula-se suas contribuições e respectivas complexidades através da tabela abaixo:

Tabela 4: Contribuição das funções na contagem de pontos de função não ajustados.

	Complexidade		
Função	Baixa	Média	Alta
ALI	7	10	15
AIE	5	7	10
EE	3	4	6
SE	4	5	7
CE	3	4	6

O cálculo dos pontos de função não ajustados é feito da seguinte forma (HAZAN, 2001):

Para cada um dos cinco tipos de função (arquivo lógico interno, arquivo de interface externa, entrada externa, saída externa e consulta externa), são contados os totais de pontos de função (NPF) com a expressão:

$$NPF = \sum_{j=1}^{3} NCi,j * Ci,j$$

Onde:

a) N $C_{i,j}$é o número de funções do tipo i (i variando de 1 a 5, segundo os tipos de função existentes: arquivo lógico interno, arquivo de interface externa, entrada externa, saída externa e consulta externa) que foram classificados na complexidade j (j variando de 1 a 3 , segundo os valores de complexidade: simples , média e complexa)

b) $C_{i,j}$é o valor da contribuição da complexidade j no cálculo dos pontos de função i, conforme a

O total de pontos de função não ajustados (PFNA) é dado pela soma dos pontos das tabelas de função:

$$PFNA = \sum_{i=1}^{5} TPFi$$

Onde: i varia de 1 a 5, conforme os tipos de função existentes (AIL, AIE, EE, SE, CE)

3.4.6. Determinação do Fator de Ajuste

O fator de ajuste influencia os pontos de função não ajustados em +/- 35%, obtendo-se o número de pontos de função ajustados. Para se calcular o fator de ajuste, são usadas 14 características gerais dos sistemas, a saber:

1. Comunicação de dados: os aspectos relacionados aos recursos utilizados para a comunicação de dados do sistema deverão ser descritos de forma global. Descrever se a aplicação utiliza protocolos diferentes para recebimento e envio das informações do sistema.

2. Processamento de Dados Distribuído: Esta característica refere-se a sistemas que utilizam dados ou processamento distribuído, valendo-se de diversas CPUs.

3. Desempenho: Descreve em que considerações sobre o tempo de resposta e a taxa de transações influenciam o desenvolvimento da aplicação.

4. Utilização do Equipamento: Trata-se de observações quanto ao nível de utilização de equipamentos requerido para a execução do sistema. Este aspecto é observado com vista a planejamento de capacidades e custos.

5. Volume de transações: Consiste na avaliação do nível de influência do volume de transações no projeto, desenvolvimento, implantação e manutenção do sistema.

6. Entrada de dados *on-line*: A análise desta característica permite quantificar o nível de influência exercida pela utilização de entrada de dados no modo *on-line* no sistema.

7. Usabilidade: a análise desta característica permite quantificar o grau de influência relativo aos recursos implementados, visando tornar o sistema amigável, permitindo incrementos na eficiência e satisfação do usuário final.

8. Atualizações *on-line*: Mede a influência no desenvolvimento do sistema face à utilização de recursos que visem a atualização dos Arquivos Lógicos Internos, no modo *on-line*.

9. Processamento complexo: a complexidade de processamento influência no dimensionamento do sistema, e, portanto, o seu grau de influência deve ser quantificado.

10. Reusabilidade: a preocupação com o reaproveitamento de parte dos programas de uma aplicação em outras aplicações implica em cuidados com a padronização. O grau de influência no dimensionamento do sistema é quantificado.

11. Facilidade de implantação: a quantificação do grau de influência desta característica é feita observando-se o plano de conversão e implantação e/ou ferramentas utilizadas durante a fase de testes do sistema.

12. Facilidade operacional: a análise desta característica permite quantificar o nível de influência na aplicação, com relação a procedimentos operacionais automáticos que reduzem os procedimentos manuais, bem como mecanismos de inicialização, salvamento e recuperação, verificados durante os testes do sistema.

13. Múltiplos Locais e Organizações do Usuário: consiste na análise da arquitetura do projeto, observando-se a necessidade de instalação do sistema em diversos lugares.

14. Facilidade de mudanças: focaliza a preocupação com a influência da manutenção no desenvolvimento do sistema. Esta influência deve ser quantificada baseando na observação de atributos.

$$GIT = \sum_{i=1}^{14} GIi$$

O valor do fator de ajuste (VFA) é calculado pela seguinte fórmula:

$$VFA = (GIT * 0{,}01) + 0{,}65$$

Se o fator de ajuste de valor é igual a 1,00, 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.

3.4.7. Calcular os pontos de funções ajustados

Uma vez calculados os pontos de funções não ajustados e o fator de ajuste, é possível calcular os pontos de funções ajustados. O cálculo é feito de formas diferentes para cada tipo de contagem.

Para projetos de desenvolvimento, o cálculo é dado por:

$$DFP = (UFP + CFP) * VAF$$

Onde:

DFP: Número de pontos de função de desenvolvimento;

UFP: Número de pontos de função brutos apurados;

CFP: Número de pontos de função adicionados por processos de conversão de dados;

VAF: Valor do fator de ajuste.

Para projetos de melhoria, o cálculo é dado por:

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

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: Valor do fator de ajuste da aplicação depois do projeto de melhoria;

DEL: Número de pontos de função não ajustados das funções excluídas pelo projeto de melhoria;

VAFB: Valor do fator de ajuste da aplicação antes do projeto de melhoria.

Projeto de aplicação.

AFP = ADD * VAF

Onde:

AFP: Número de pontos de função ajustados da aplicação;

ADD: Número de pontos de função não ajustados das funções instaladas;

VAF: Valor do fator de ajuste da aplicação.

3.5. Estimativas

As equipes ágeis não assumem premissas em um projeto e realizam estimativas somente para esclarecer o que cabe em cada *sprint*, usando o cálculo de velocidade.

A maneira mais simples de estimar a velocidade é verificar o histórico do time, respondendo a pergunta: Qual foi a velocidade do time nos últimos *sprints*? Então assumir que a velocidade será a mesma para o último *sprint*. Outra maneira de calcular é através de cálculo de recurso. Fórmula para velocidade estimada do *sprint*:

VELOCIDADE ESTIMADA = DIAS DE RECURSO DISPONIVEL * FATOR FOCO

Fator foco é uma estimativa de como o time está focado no projeto. Um fator foco baixo significa que o time espera encontrar vários inconvenientes. A melhor maneira de determinar um fator foco concreto é analisando a média dos últimos *sprints.* Para tanto, há uma fórmula para calcular o fator foco do último *sprint*:

FATOR FOCO = VELOCIDADE ATUAL / DIAS DE RECURSO DISPONIVEL

Velocidade atual é a soma da estimativa inicial de todas as estórias que foram finalizadas no *sprint* anterior. A velocidade vai ser calculada pelo *ScrumMaster*. Na reunião de planejamento da *sprint* o time *scrum,* o *ScrumMaster* e o *product owner* vão definir uma *sprint* de acordo com a prioridade de cada estória, sendo que o tempo utilizado em cada estória será obtido pela métrica *planning poker e* calculado com o *ScrumMaster* e o seu time, considerando a velocidade da equipe. O *product owner* vai garantir que as estórias com maior valor para o cliente sejam implementadas de forma priorizada, no tempo especificado da *sprint*.

A análise de ponto de função possibilita estimar a dimensão de projetos desde a primeira fase da análise de sistema. A precisão da estimativa do tamanho de uma aplicação varia de acordo com o grau de conhecimento adquirido sobre a mesma, ou em outras palavras, da fase em que se encontra o projeto.

Segundo Meli (1999 apud HAZAN 2009 p.27) “estimar significa utilizar o mínimo de tempo e esforço para se obter um valor aproximado dos Pontos de Função do projeto de *software* investigado.” A estimativa de ponto de função fornece uma avaliação aproximada do tamanho do *software,* utilizando métodos diferentes da contagem de ponto de função. A contagem de pontos de função mede um valor aproximado do tamanho obtido pelas regras do processo de contagem de ponto de função.

Entretanto, para se fazer estimativas antes do final do projeto físico, neste caso a última versão do *product backlog*, com uma margem de erro aceitável, é aconselhável que se apoie em algum método com bases estatísticas sobre os pontos de função para suprir a falta de conhecimento de algumas funções da aplicação.

As estimativas de produtividade serão calculadas pelo *SrumMaster*, o qual utilizará o cálculo da velocidade para auxiliar a métrica *planning poker*. A estimativa de produtividade por ponto de função. Quantidade de unidade de tamanho do software por pontos de função que foi construída em uma unidade de tempo. Fórmula para calcular produtividade de uma equipe por ponto de função:

PRODUTIVIDADE = TEMPO / PONTOS DE FUNÇÃO

A taxa de manutenção de um determinado *software* não vai ser calculada, pois as técnicas ágeis permitem mudanças no escopo do projeto e, portanto, é desnecessário o calculo dessa estimativa.

O esforço para desenvolver um *software* pode ser definido como sendo a quantidade de tempo (horas) de trabalho que serão necessárias para produzir um sistema. Conhecida a produtividade da equipe de desenvolvimento, a fórmula abaixo poderá ser aplicada:

ESFORÇO = PRODUTIVIDADE * TAMANHO DO SOFTWARE

O *product owner* vai calcular esse esforço para poder definir questões de contrato. Não existe um preço único para ponto de função. Deve-se avaliar o conjunto de atividades relativas à disponibilização das funcionalidades medidas em pontos de função e os aspectos não funcionais que são desconsiderados na medição dos pontos de função, a fim de obter uma referência confiável.

O modelo de contrato que ditará a remuneração de um ponto de função sobre a estimativa do preço do *software*.

3.6. Contratos

Segundo Teles (2008), quando um contrato de escopo fixo é realizado, é assumido premissas sobre a previsão do custo, prazo e escopo do projeto, para uma ilusão de previsibilidade e deixar o contrato de escopo físico, utilizado nas metodologias tradicionais, atraente para o cliente. Teoricamente, a empresa contratada sabe o que é preciso ser feito, quanto irá ganhar e terminar o *software.* Mas antes de selecionar essa opção, é preciso responder duas perguntas: O cliente sabe exatamente o que deseja no inicio do projeto? A equipe é capaz de estimar com perfeição e entregar o sistema no dia combinado?

No modelo de contrato tradicional, quando surge alteração no escopo, é cobrado um valor sobre a mesma. No escopo fixo, quando ocorrem mudanças, é considerado um impedimento para a equipe concluir o projeto. A análise de ponto de função usa a taxa de manutenção para achar esse valor.

O escopo não está vinculado ao contrato de escopo negociável, o que diminui os riscos do fornecedor não deixar de cumprir o contrato. Ele permite mudanças nas funcionalidades do *software*, sem cobrar taxas elevadas para isso. Nesse contrato não existe uma previsibilidade do que será feito no projeto, mas existe do custo e do tempo, sendo as variáveis do contrato: tempo, qualidade e custo. Mas como convencer o cliente a utilizar este contrato onde nenhuma premissa é assumida?

A análise de ponto de função antes de terminar o projeto do *software* não consegue responder perfeitamente o tamanho do *software* e, portanto, gera margem de erro pequena sobre as previsões realizadas, que tende a aumentar quando ocorrem mudanças no projeto. Assim, optaremos por utilizar o contrato de escopo negociável. O *product owner* pode utilizar as estimativas da análise de ponto de função para negociar com o cliente.

No contrato de escopo negociável, o cliente pode interrompê-lo caso não esteja de acordo com o trabalho realizado pela equipe durante os primeiros dois meses de trabalho. Para que isso não ocorra, deve-se envolver o cliente no processo de desenvolvimento. Em muitos times *scrum*, o *product owner* é o próprio cliente. É importante que o *product owner* entenda sobre as regras de negócios da empresa do cliente como os processos para o desenvolvimento do *software* o que muitas vezes os clientes não conseguem atender esses requisitos.

Mas o cliente como *stakeholder* é importante que ele juntamente como *product owner* participe dos processos como criação do *product backlog*, com a priorização das estórias do usuário, e o fornecimento do *feedback* para equipe.

Neste capítulo mostra como utilizar a métrica Análise de Ponto de Função em um ambiente ágil destacando: as diferenças entre Pontos de Função e *story points*, quem deve contar os pontos de função, o processo de contagem de pontos de função no ambiente ágil, como é realizada as estimativas, o contrato que deve ser utilizado em uma equipe ágil.

CONCLUSÃO

Existe uma verdadeira polêmica entre desenvolvedores ágeis e tradicionais sobre qual é o melhor método para desenvolver *software***.** Muitos tradicionalistas acreditam que o desenvolvimento ágil não possui documentação necessária e, por este motivo, o consideram como retrocesso.

Este trabalho demonstra que, apesar das equipes ágeis preocuparem-se mais com a implementação do *software* do que com a documentação deste, as mesmas realizam estórias de usuários para que todos os *stakeholders*, pessoas envolvidas ou interessadas no *software*, possam entender o seu desenvolvimento. Para tanto, utilizam uma documentação mais simples, sem jargões técnicos de informática, a fim de gerar transparência do *software* para o cliente.

As métricas foram outro assunto discorrido durante o trabalho, com ressalvas a sua importância para gerenciar e diminuir os riscos do projeto. As métricas ágeis são informais e não têm muito valor para os clientes, principalmente para produção de *software* para o governo onde a métrica *Análise de Ponto de Função* tornou-se padrão na venda de *software* para essas organizações.

Assim, o trabalho propôs conciliar o uso da métrica análise de ponto de função em metodologias ágeis e, para isso, foi utilizada *scrum* que, apesar de ser classificada como uma metodologia, demonstrou ser um *framework* de ideais ágeis que pode ser vinculado com outras metodologias e ferramentas.

A escassez de documentos formais de metodologias ágeis e Análise de Ponto de Função em Metodologias Ágeis dificultou o estudo. A Análise de Ponto de Função em Metodologias Ágeis é um tema atual, logo, existem poucas experiências relatadas, o que pode impossibilitar o estudo de caso.

Como resultado da pesquisa realizada, constatou-se que é possível fazer uma junção dos dois assuntos, dada a versatilidade do *scrum* e a independência da métrica Análise de Ponto de Função de qualquer tecnologia ou metodologia. Essa junção tem causado bastante polêmica, já que para medir-se o ponto de função, os requisitos devem estar bem definidos, o que não acontece no ambiente ágil e pode ferir tanto os princípios ágeis como corromper a contagem de ponto de função, dependendo do objetivo ou como é utilizado.

Assim, como contribuição deste trabalho foi proposta uma contagem de pontos de funções sem corromper os dados obtidos, lembrando sempre que os dados foram tratados como uma estimativa e não uma premissa, onde os resultados são tidos como uma verdade.

Para concretizar tal proposta, o valor agregado na contagem de pontos de função só poderá ocorrer se sua contagem não demandar muito tempo ou dificuldade que atrapalhe no desenvolvimento do sistema.

O estudo de outras métricas para auxiliar o desenvolvimento do software juntamente como ponto de função e o detalhamento da estória de usuários para simplificar a contagem dos pontos de função pode ser material para estudos futuros.

REFERÊNCIAS

ALBERT, Renato; SIMÕES, Guilherme; VARQUEZ, Carlos **Análise de pontos de função Medição, Estimativas e Gerenciamento de Projetos de Software**.10ª Ed. São Paulo: Érika, 2010.

ALBRECHT, Allan J. **Medindo a Produtividade do Desenvolvimento de Aplicativos** Disponível em: < http://www.fattocs.com.br/artigos/MeasuringAplicationDevelopmentProductivity.pdf > Acesso em 03 de dezembro de 2011.

BARBOSA, ANA C; GAMBA Mateus L. **Aplicação de Métricas de Software com Scrum** Disponível em: < http://periodicos.unesc.net/index.php/sulcomp/article/view/242 > Acesso em 03 de dezembro de 2011.

BARCELLOS, Monalessa P. **Realização de Estimativas utilizando Análise de Pontos de Função** Disponível em: <*www.inf.ufes.br/~monalessa/...NEMO/.../**AnalisePontos**Funcao.pdf* > Acesso em 04 de dezembro 2011.

BECK, Kent et al. **Manifesto para o desenvolvimento ágil de software** Disponível em: <http://manifestoagil.com.br/c> Acesso em 29 de Novembro de 2011.

BECK. Kent **Extreme Programming Explained: Embrace Change**. Addison-Wesley Professional, 1999.

BURGOS, Octavio **Modelo de Processo para Medição e Análise em Desenvolvimento de Software Baseado no CMMI** Disponível em: <www.lactec.org.br/mestrado/dissertacoes/arquivos/OctavioBurgos.pdf> Acesso em 03 de dezembro 2011.

CAMMARANO, Raf **Goal Question Metric (GQM) Model** Disponível em:< http://rafcammarano.wordpress.com/2008/04/07/goal-question-metric-gqm-model/> Acesso em 06 de dezembro de 2011.

CLARO, Daniela B. **Métricas de Software.** Disponível em:<http://www.inf.ufsc.br/~danclaro/download/disciplinas/M%E9tricas%20de%20Software.doc> Acesso em 26 de maio de 2011.

DIAS, Raquel **Análise por Pontos de Função: Uma Técnica para Dimensionamento de Sistemas de Informação**,. Disponível em: <www.presidentekennedy.br/resi/edicao03/artigo02.pdf.> Acesso em 08 de novembro de 2011

FONSECA, Isabella **Artigo Engenhariaa de software 7 - Ideal Day e Priorização**. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=11023> Acesso em 28 de maio de 2011.

GAMBA, Mateus; Barbosa, Ana **Aplicação de Métricas de Software com Scrum**. Disponível em: <http://mateusgamba.wordpress.com/2010/11/01/aplicacao-metricas-no-scrum/> Acesso em 28 de maio de 2011.

GARCIA, Hamíliciar **Dicionário Contemporâneo da Língua Portuguesa** 5. ed. Rio de Janeiro: Delta. 1986. 3 v. 1227 p.

HAZAN, Claudia **Análise de Ponto de Função** Disponível: <www.inf.ufes.br/~falbo/download/aulas/es-g/2005-1/**APF**.pdf> Acesso em 08 de novembro de 2011.

HAZAN, Claudia **Análise de Pontos de Função: Uma aplicação nas estimativas de tamanho de Projetos de Software** Disponível em <*www.devmedia.com.br/articles/viewcomp.asp?comp=9146*> Acesso em 04 de dezembro 2011.

KNIBERG, Henrik **SCRUM E XP DIRETO DAS TRINCHEIRAS** Disponível em: <http://www.infoq.com/br/minibooks/scrum-xp-from-the-trenches> Acesso em: 29 de novembro de 2011.

KOSCIANSKI, Andre; SOARES, Michel **Qualidade de Software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software.** 2 ed. Novatec: 2005.

LAKATOS, Eva M.; MARCONI, Marina de A. **Metodologia do trabalho científico**. São Paulo: Atlas, 1995.

MARCORATTI, José C. **Análise de Ponto de Função: O Processo de contagem** Disponível em:<www.macoratti.net/apf_pcta.htm> Acesso em 05 de dezembro de 2011.

MARIA, Jocemar **Roteiro para Elaborar Métricas no Processo de Desenvolvimento de Software Utilizando Métodos DMAIC e GQM** Disponível em: <*tconline.feevale.br/tc/files/0002_2476.pdf*> Acesso em: 29 Nov. 2011.

MOUNTAIN GOAT SOFTWARE. ***Traning for Scrum Task Board Use*** Disponível em: < http://www.mountaingoatsoftware.com/scrum/task-boards> Acesso em 06 de dezembro 2011.

MOURA, Ricardo **SCRUM - Boas práticas para escrever estórias** Disponível em: **<**http://conhecimento.mcptecnologia.com/pages/viewpage.action?pageId=9044504> Acesso em 30 de novembro de 2011.

PIMENTEL, Manoel **Escopo Iterativo e Incremental para o Gerenciamento Ágil de Requisitos** Disponível em: <http://visaoagil.wordpress.com/2009/02/05/escopo-iterativo-e-incremental-para-o-gerenciamento-agil-de-requisitos/> Acesso em 30 Nov. 2011.

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

RAMOS, Carlos R. et al **Roteiro de Métricas de Software do SISP** Disponível em: <http://www.governoeletronico.gov.br/biblioteca/arquivos/roteiro-de-metricas-de-software-do-sisp/view > Acesso:04 Dez. 2011.

RIBEIRO, Camilo **Vocabulário básico para testadores ágeis**. Disponível em: <http://www.bugbang.com.br/?p=1497> Acesso em 27 de maio de 2011.

SABBAGH, Rafael **Um Novo Jogo** Disponível em: < http://scrumemacao.com.br/web/index.php?option=com_content&view=article&id=15&Itemid=15> Acesso em 01 de dezembro de 2011.

SANTOS, Wagner **Estimativas Ágeis – Planning Poker**. Disponível em: <http://netfeijao.blogspot.com/2008/10/estimativas-geis-planning-poker.html> Acesso em: 28 de maio de 2011.

SATO, Danilo **Artigo Métricas de Acompanhamento para Metodologias ágeis**. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=12562> Acesso em : 27 de maio de 2011.

SCHWABER, Ken. e SUTHERLAND, Jeff. 2010 **Scrum Guide** Disponível: <http://www.scrum.org/scrum-guide-for-andriod/> Acesso em 08 de novembro de 2011.

SOMMERVILLE, Ian. **Engenharia de Software**. 6ª Ed. São Paulo: Addison Wesley, 2003.

VAZQUES, Carlos. 2011, **Como gerenciar o desenvolvimento iterativo e incremental em fábricas de software usando a APF,** Disponível: <http://www.fattocs.com.br/blog/?p=155> Acesso em: 08 de novembro de 2011.

TELES, Vinicius **Contrato de Escopo Negociável** Disponível: <http://improveit.com.br/xp/praticas/contrato> Acesso em 08 de novembro de 2011.

VAZQUES, Carlos. 2011, **Gerência de Requisitos, Desenvolvimento Ágil e APF: A Busca do Equilíbrio Perdido,** Disponível: <http://www.fattocs.com.br/blog/?p=392> Acesso em: 08 de novembro de 2011

VENDRAMEL, Wilson **Proposta de um Método de Estimativa para Projetos de manutenção de Software Baseado em Pontos de Caso de Uso** Disponível em: <http://www.unip.br/ensino/pos_graduacao/strictosensu/eng_producao/download/eng_wilson vendramel.swf> Acesso em 29 de novembro de 2011.

VERGARA, Sylvia Constant. **Projetos e Relatórios de Pesquisa em Administração**. São Paulo: Editora Atlas, 2007.

APÊNDICES

Apêndice A – Cronograma de Atividades do Trabalho de Conclusão de Curso

Atividades	2011																					
	Fev		Mar		Abr		Mai		Jun		Jul		Ago		Set		Out		Nov		Dez	
	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
Entrega do termo de Aceite	X																					
Entrega do projeto					X																	
Entrega da Monografia Parcial									X													
Apresentação da Monografia parcial									X													
Entrega do pôster													X									
Exposição do pôster														X								
Entrega da Monografia Final																			X			
Apresentação da Monografia final																				X		
Entrega das correções																					X	
Entrega da versão final																					X	

Apêndice B - PÔSTER APRESENTADO NO III SIMPÓSIO DE TECNOLOGIA DA INFORMAÇÃO E III SEMANA DE INICIAÇÃO CIENTÍFICA DO CURSO DE SISTEMAS DE INFORMAÇÃO UNUCET-UEG/2011.

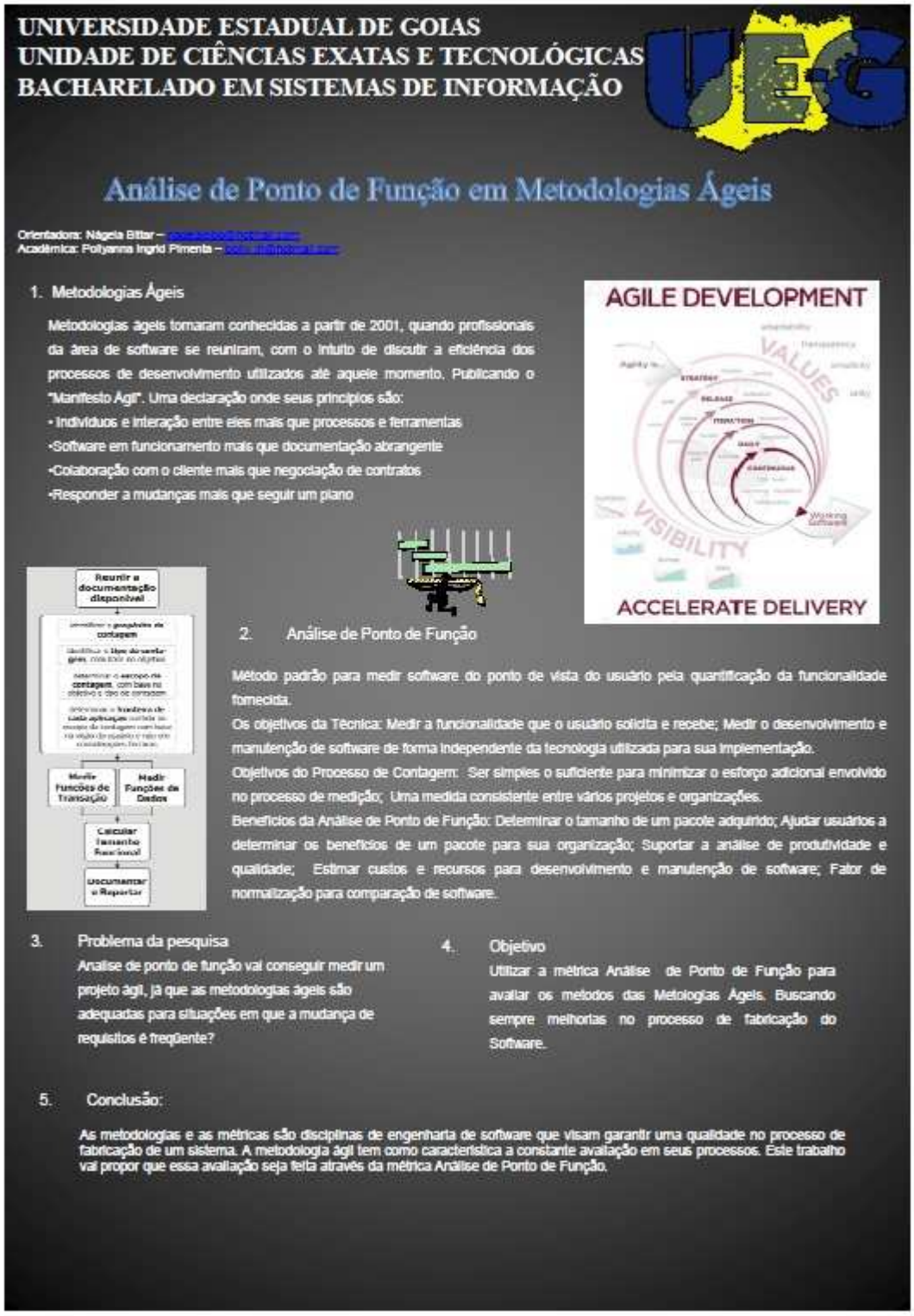

Figura 11: Pôster Análise de Ponto de Função em Metodologias Ágeis.

ANEXOS

ANEXO A: MODELO DE CONTRATO DE ESCOPO NEGOCIÁVEL

MODELO DE CONTRATO DE ESCOPO NEGOCIÁVEL

Contrato de Prestação de Serviços que entre si fazem <razão social da contratante>, inscrita no CNPJ sob o n° <cnpj>, com sede nesta cidade, na Av. <logradouro e complementos>, no <bairro>, doravante denominada, abreviadamente CONTRATANTE, e <razão social da contratada>, inscrita no CNPJ sob o nº <cnpj>, com sede na cidade de <cidade>, no estado de <estado>, na Av. < logradouro e complementos>, no <bairro>, doravante denominada, abreviadamente, CONTRATADA, ambas por seus representantes legais ao final assinados, na forma abaixo:

Para saber mais sobre esse modelo de contrato e seu funcionamento, acesse:

http://www.improveit.com.br/xp/praticas/contrato

CLÁUSULA PRIMEIRA – DO OBJETO

O presente contrato tem por objeto regular as condições mediante as quais a **CONTRATADA** prestará à **CONTRATANTE** os serviços discriminados na cláusula segunda a seguir, com fornecimento de mão-de-obra.

CLÁUSULA SEGUNDA – DA PRESTAÇÃO DOS SERVIÇOS

A prestação de serviços da **CONTRATADA** compreende o desenvolvimento para a **CONTRATANTE** de Sistema de <nome do sistema>, e módulos correlatos, doravante denominado, abreviadamente, **SISTEMA**, com fornecimento de mão-de-obra.

PARÁGRAFO PRIMEIRO – A **CONTRATADA**, para a execução dos serviços ora contratados, utilizará recursos de mão-de-obra próprios, sendo de sua exclusiva responsabilidade todas as despesas com o seu pessoal, inclusive todos os encargos trabalhistas, previdenciários e securitários.

PARÁGRAFO SEGUNDO – A prestação dos serviços ora contratados será feita no horário de 09:00 hs às 18:00 hs, de segunda à sexta-feira, na sede da **CONTRATANTE**, situada nesta cidade, na <endereço onde serão executados os serviços>, ou onde a mesma indicar, através de uma equipe formada por 06

profissionais, a saber

PARÁGRAFO TERCEIRO – A **CONTRATANTE** determinará, no primeiro dia útil de cada semana, durante toda a vigência do presente contrato, as funções a serem desenvolvidas semanalmente pela **CONTRATADA**, que as implementará até o último dia útil daquela mesma semana.

PARÁGRAFO QUARTO – A critério da **CONTRATANTE**, poderá ser aumentado o número de desenvolvedores do **SISTEMA**, podendo a **CONTRATADA** para tanto contratar outras empresas sob sua inteira e exclusiva responsabilidade, desde que previamente autorizada pela **CONTRATANTE**.

PARÁGRAFO QUINTO – A critério da **CONTRATADA**, os serviços poderão ser executados além do horário estabelecido no parágrafo segundo supra, sem que isso acarrete para a **CONTRATANTE** obrigação de pagamento de qualquer remuneração além da estipulada na cláusula oitava deste contrato.

CLÁUSULA TERCEIRA – DAS OBRIGAÇÕES DA CONTRATANTE

Constituem obrigações da **CONTRATANTE,** dentre outras previstas neste contrato:

a) permitir o acesso dos equipamentos e prepostos da **CONTRATADA**, devidamente identificados, ao local onde serão realizados os serviços objeto deste contrato, prestando-lhes todos os esclarecimentos necessários;

b) pagar pontualmente a remuneração da **CONTRATADA**;

c) informar à **CONTRATADA**, semanalmente, as prioridades de funções a serem por ela desenvolvidas;

d) Fornecer à **CONTRATADA** todas as informações relativas às suas normas internas necessárias à prestação dos serviços ora contratados.

CLÁUSULA QUARTA – DAS OBRIGAÇÕES DA CONTRATADA

Constituem obrigações da **CONTRATADA,** dentre outras previstas neste contrato:

a) cumprir rigorosamente os prazos previstos para a execução dos serviços, implementando, semanalmente, as funções estabelecidas pela **CONTRATANTE**;

b) utilizar mão-de-obra qualificada e devidamente treinada para a execução dos serviços ora contratados, substituindo os profissionais que a **CONTRATANTE** considere não atender as necessidades relativas ao desenvolvimento do **SISTEMA**;

c) atender todas as despesas com o pessoal de sua contratação utilizado na prestação dos serviços ora contratados, inclusive os encargos trabalhistas, previdenciários e securitários;

d) prestar à **CONTRATANTE** quaisquer informações e esclarecimentos que se fizerem necessários para o acompanhamento da evolução dos serviços ora contratados;

e) responder por todos os danos e/ou acidentes que seus empregados, prepostos e/ou terceiros sob sua responsabilidade possam ocasionar à **CONTRATANTE** ou à terceiros;

f) cumprir, todas as leis e posturas federais, estaduais e municipais;

g) ressarcir à **CONTRATANTE** todas as despesas e prejuízos que a mesma tiver na hipótese de vir a ser demandada por quaisquer atos ou faltas sua, de seus prepostos e/ou terceiros sob sua responsabilidade, no cumprimento de suas obrigações legais e/ou contratuais;

h) revisar ou corrigir, de forma pronta e imediata, sem qualquer ônus para a **CONTRATANTE**, todas as falhas, deficiências, imperfeições ou defeitos apresentados no **SISTEMA**;

i) fornecer à **CONTRATANTE** manual de utilização quando da validação de cada módulo (release) do SISTEMA.

CLÁUSULA QUINTA – DA FISCALIZAÇÃO EVISTORIA

A **CONTRATANTE** poderá, a qualquer tempo, sempre acompanhada da **CONTRATADA**, fiscalizar e/ou vistoriar a exata e pontual execução dos serviços ora contratados e o cumprimento das demais obrigações previstas no presente contrato, devendo a **CONTRATADA** prestar todos e quaisquer esclarecimentos a ela solicitados.

PARÁGRAFO ÚNICO – A fiscalização e/ou vistoria realizadas pela **CONTRATANTE** e/ou por terceiros por ela prévia e expressamente indicados, não eximirá a **CONTRATADA** das responsabilidades oriundas ou decorrentes da prestação dos serviços ora contratados.

CLÁUSULA SEXTA – DA CONFIDENCIALIDADE E SIGILO

As partes, por seus dirigentes, prepostos ou empregados, comprometem-se, mesmo após o término do presente contrato, a manter completa confidencialidade e sigilo sobre quaisquer dados ou informações obtidas em razão do presente contrato, reconhecendo que não poderão ser divulgados ou fornecidos a terceiros, salvo com expressa autorização, por escrito, da outra parte.

PARÁGRAFO ÚNICO – As partes serão responsáveis, civil e criminalmente, por quaisquer danos causados uma a outra e/ou terceiros em virtude da quebra da confidencialidade e sigilo a que estão obrigadas.

CLÁUSULA SÉTIMA – DA PROPRIEDADE INTELECTUAL

Os estudos, projetos, relatórios e demais dados desenvolvidos pela **CONTRATADA** em razão dos serviços ora contratados, ainda que inacabados, serão de propriedade exclusiva da **CONTRATANTE**, que poderá registrá-los nos órgãos competentes e utilizá-los ou cedê-los sem qualquer restrição ou custo adicional.

PARÁGRAFO ÚNICO – A **CONTRATADA** será a única responsável por infrações a direito de propriedade intelectual de terceiros, inclusive aquelas relacionadas a materiais, equipamentos, programas de computador ou processos de execução protegidos pela legislação em vigor, que tenham sido utilizados na execução dos serviços ora contratados, respondendo diretamente por quaisquer reclamações, indenizações, taxas ou comissões que forem devidas.

CLÁUSULA OITAVA – DA REMUNERAÇÃO DA CONTRATADA

Pelos serviços ora contratados, a **CONTRATANTE** pagará à **CONTRATADA**, mensalmente, o valor de **R$ XXX** (xxx reais), todo dia **primeiro** do mês subsequente ao vencido.

PARÁGRAFO PRIMEIRO – O valor estipulado para a remuneração da **CONTRATADA** inclui todos e quaisquer impostos, taxas e/ou encargos fiscais, sociais, previdenciários e securitários, sendo que se houver alteração na legislação fiscal que importe em alteração dos valores hoje vigentes, as partes poderão negociar a revisão do valor ajustado, para mais ou para menos.

PARÁGRAFO SEGUNDA– O pagamento da remuneração devida à **CONTRATADA** será feito, no respectivo vencimento, na sede da **CONTRATANTE**, ou depósito na conta-corrente bancária da **CONTRATADA** (conta nº **<conta>** do **<nome do banco>**, agência**<número da agência>**), devendo a **CONTRATADA** entregar à **CONTRATANTE**, na sede da mesma, a documentação fiscal correspondente, corretamente preenchida, com 10 (dez) dias de antecedência.

PARÁGRAFO TERCEIRO – A não apresentação pela **CONTRATADA** da documentação fiscal correspondente, corretamente preenchida, no prazo estabelecido no parágrafo segundo supra, acarretará a automática prorrogação do prazo de pagamento da remuneração devida por mais 10 (dez) dias a contar da sua correta apresentação, sem qualquer ônus para a **CONTRATANTE**.

PARÁGRAFO QUARTO – O valor da remuneração da **CONTRATADA** será reajustado anualmente de acordo com a variação do IPCA (Índice de Preços ao Consumidor Ampliado), publicado pelo IBGE.

PARÁGRAFO QUINTO – O não pagamento, nos exatos vencimentos, dos valores devidos à **CONTRATADA**, por culpa exclusiva da **CONTRATANTE**, acarretará a incidência de correção monetária de acordo com a variação do IPCA (Índice de Preços ao Consumidor Ampliado), publicado pelo IBGE, multa de 1% (um por cento) e ainda juros de mora de 0,5% (meio por cento) ao mês, tudo calculado *pro-rata-die* desde a data do vencimento até o efetivo pagamento, devendo tal critério ser aplicado a eventuais créditos a favor da **CONTRATANTE** não liquidados até a data do vencimento.

PARÁGRAFO SEXTO – A **CONTRATANTE** poderá reter ou sustar o pagamento da remuneração devida à **CONTRATADA** por força do presente contrato na hipótese de descumprimento pela mesma de quaisquer das cláusulas ou condições ora pactuadas ou no caso de vir a ser responsabilizada por quaisquer atos ou falhas da **CONTRATADA,** de seus prepostos ou empregados no cumprimento de suas obrigações legais ou contratuais.

CLÁUSULA NONA – DA CESSÃO E TRANSFERÊNCIA

A **CONTRATADA** não poderá ceder ou transferir para terceiros os direitos e obrigações decorrentes do presente contrato sem a expressa concordância por escrito da **CONTRATANTE.**

CLÁUSULA DÉCIMA – DA COMUNICAÇÃO ENTRE AS PARTES

A comunicação entre as partes será feita por carta, entregue contra recibo ou pelo correio com aviso de recepção na sede das mesmas, ou através de e-mail e telefones abaixo especificados:

PARA A CONTRATANTE:

E-MAIL: <email do contato>

TELEFONE: <telefone do contato>

PARA A CONTRATADA:

E-MAIL: <email do contato>

TELEFONE: <telefone do contato>

CLÁUSULA DÉCIMA PRIMEIRA – DA VIGÊNCIA DO CONTRATO

O presente contrato vigerá, a partir desta data, por tempo indeterminado.

PARÁGRAFO PRIMEIRO – Sem prejuízo da satisfação de seus demais direitos, a cada dois meses de aniversário deste contrato, a **CONTRATANTE** poderá dá-lo por rescindido sem que caiba à **CONTRATADA** o direito a qualquer reclamação, indenização ou compensação, desde que o faça, por escrito, até dez dias antes ou após cada aniversário bimestral do contrato, ajustando-se as contas exclusivamente em função dos serviços prestados até a data do distrato.

PARÁGRAFO SEGUNDO – Sem prejuízo da satisfação de seus demais direitos, a cada dois meses de aniversário deste contrato, a **CONTRATADA**poderá dá-lo por rescindido sem que caiba à **CONTRATANTE** o direito a qualquer reclamação, indenização ou compensação, desde que o faça, por escrito, até dez dias antes ou após cada aniversário bimestral do contrato, obrigando-se, entretanto, a concluir as etapas dos serviços que estiverem em andamento.

CLÁUSULA DÉCIMA SEGUNDA – DAS MODIFICAÇÕES OU ALTERAÇÕES

Todas as modificações ou alterações no presente contrato deverão ser feitas por escrito, sendo de nenhum efeito as combinações verbais.

CLÁUSULA DÉCIMA TERCEIRA – DA EXTRAÇÃO DE DUPLICATAS

Fica expressamente proibida a extração de duplicatas ou a emissão de quaisquer documentos pela **CONTRATADA**que possa ensejar protesto contra a **CONTRATANTE.**

CLÁUSULA DÉCIMA QUARTA– DAS PENALIDADES

O inadimplemento de quaisquer das cláusulas ou condições do presente contrato, bem como na hipótese de qualquer das partes se tornar insolvente, sem prejuízo das perdas e danos e demais cominações previstas em lei e neste instrumento, dará direito à parte prejudicada de considerar o presente rescindido de pleno direito, independentemente de qualquer notificação, interpelação ou aviso, judicial ou extrajudicial, acarretando ainda para a parte inadimplente, exceto na hipótese prevista no parágrafo sexto da cláusula oitava, que possui regra própria, o pagamento de multa não compensatória a favor da outra de **R$ XXX** (xxx reais), equivalente a xxx IPCA's/IBGE, devidamente corrigida desde a presente data até o efetivo pagamento.

CLÁUSULA DÉCIMA QUINTA – DO CASO FORTUITO OU FORÇA MAIOR

Nenhuma das partes será responsável perante a outra por qualquer falha ou atraso no cumprimento das obrigações constantes do presente contrato, causados por casos fortuitos ou força maior.

CLÁUSULA DÉCIMA SEXTA – DA TOLERÂNCIA

A tolerância, a não aplicação das penalidades, ou ainda, o não exercício dos direitos que necessariamente defluirão para uma das partes em virtude do inadimplemento da outra, não induzirão novação, precedente ou alteração dos pactos, sendo a ocorrência de qualquer dos fatos supra levada à conta de simples liberabilidade por parte do contratante que tolerou, não aplicou as sanções ou não exerceu o direito.

CLÁUSULA DÉCIMA SÉTIMA – DO FORO

Fica eleito o foro central da Comarca da Capital do Estado do Rio de Janeiro, com renúncia expressa de qualquer outro, por mais privilegiado que seja, para dirimir quaisquer dúvidas decorrentes do presente contrato, sendo o ajuste aqui feito obrigatório para as partes, seus herdeiros ou sucessores.

Assim, por estarem justas e contratadas, firmam as partes o presente em 02 (duas)vias de igual teor e forma para um só fim de direito, o que fazem diante das testemunhas também ao final assinadas.

<razão social da empresa contratante>
Contratante

<razão social da empresa contratada>
Contratada

Testemunhas:

_____
Nome:
CPF:
RG:

_____
Nome:
CPF:
RG: