Pós-Graduação em Ciência da Computação

Uma abordagem para a priorização de requisitos em ambientes ágeis

Por

Diego Maciel Asfora

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife, Abril de 2009

Diego Maciel Asfora

Uma abordagem para priorização de requisitos em ambientes ágeis

DISSERTAÇÃO APRESENTADA AO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO, COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADORA: *PROFA. DRA. CARINA FROTA ALVES*

Recife, Abril de 2009

Asfora, Diego Maciel

Uma abordagem para priorização de requisitos em ambientes ágeis / Diego Maciel Asfora. - Recife: O autor, 2009.

xii, 110 folhas : il., fig., tab.

Dissertação (mestrado) - Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2009.

Inclui bibliografia e apêndice.

1. Engenharia de software. I. Título.

005.1 CDD (22.ed.) MEI-2010-017

Dissertação de Mestrado Profissional apresentada por **Diego Maciel Asfora** Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título, **"Uma abordagem para a priorização de requisitos em ambientes ágeis"**, orientada pela **Profª. Carina Frota Alves** e aprovada pela Banca Examinadora formada por:

Carina Frota Alves

Profª. Carina Frota Alves
Centro de Informática / UFPE

Rosa Cândida Cavalcanti Pinho

Profª. Rosa Cândida Cavalcanti Pinho
Faculdade Guararapes

Carla Taciana Lima Lourenço Silva

Drª. Carla Taciana Lima Lourenço e Silva
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 08 de maio de 2009.

Francisco de Assis Tenório de Carvalho

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Agradecimentos

Agradeço primeiramente **a Deus** ter chegado até aqui enfrentando e vencendo todas as adversidades que apareceram na minha vida. À **minha família,** por me proporcionar todo o apoio necessário durante essa empreitada. Em especial ao **meu Pai (João Alberto Asfora)** e a **minha Mãe (Ana Isabel Maciel)** por terem me educado de uma forma que só os dois poderiam fazer. Aos meus irmãos, que ao eu ser suas fontes inspiradoras me deixam com mais vontade de fazer cada vez melhor. Gostaria de destacar ainda aos meus avós que passaram para meus pais e conseqüentemente para mim suas experiências de vida proporcionando os meios que eu chegasse até aqui.

A minha orientadora **Profa. Carina Alves**, que além de ter acreditado na proposta foi realmente uma grande orientadora buscando sempre mostrar pela sua experiência como seria esse desafio de um mestrado e como passar por esses caminhos com menos sofrimento. Sua paciência, dedicação, conhecimento único de uma verdadeira orientadora.

Gostaria de agradecer a todos que colaboraram para a realização da pesquisa, as empresas que se dispuseram a ajudar e principalmente **Igor Macaúbas** e **Felipe Furtado** que foram sempre prestativos ajudando na realização desse trabalho.

As professoras Rosa Pinto e Carla Silva que se dispuseram a compor a banca avaliadora dessa dissertação.

A todos os amigos que me apoiaram e me incentivaram durante a confecção desse trabalho. Não poderia citar os nomes para não ser injusto com alguns, uma vez que durante 2 anos e meio de trabalho muitas pessoas me ajudaram. Cada uma ao seu momento. E da forma que conseguiram ajudar.

Resumo

Atualmente, empresas de software precisam desenvolver sistemas de forma rápida e eficiente. A competição crescente tem levado empresas a adotar metodologias de desenvolvimento ágeis para garantir a vantagem competitiva e aumentar a satisfação de clientes. A priorização de requisitos é uma atividade crítica do desenvolvimento de software. Esta atividade envolve a análise da importância de cada requisito por parte dos stakeholders e seleção dos requisitos que irão ser implementados em determinada versão do sistema. A decisão errada sobre quais requisitos priorizar pode afetar a qualidade global do sistema, e conseqüentemente sua aceitação pelos clientes. No desenvolvimento ágil é mais enfatizada a indicação do cliente para realizar a priorização dos requisitos. Nela alguns problemas podem ocorrer, pois o cliente pode acreditar que determinado requisito tem mais importância do que ele realmente possui, assim como ele pode não dar a devida importância a um requisito que ele desejava muito, mas que só sente necessidade quando o sistema estiver pronto e com aquela funcionalidade faltando. Metodologias ágeis com Scrum e Extreme Programming oferecem orientações bastante básicas sobre como conduzir essa priorização. Dentre as principais limitações do processo de priorização descrito pelas metodologias ágeis destacamos a dificuldade de comparar a real importância dada aos requisitos e a falta de análise caso determinados requisitos não sejam selecionados. O objetivo central desta pesquisa é propor um processo de priorização de requisitos para projetos de software ágeis baseado na técnica Kano, que foi originalmente proposta na área da administração. Os diagramas produzidos pela técnica Kano objetivam priorizar requisitos através de uma pergunta funcional e uma pergunta disfuncional para cada requisito. O processo proposto nesta dissertação auxilia a priorização de requisitos e melhora a forma de visualizar os resultados da priorização através de gráficos e quadros mais ilustrativos, facilitando a tomada de decisão por parte dos stakeholders do projeto. Com o objetivo de avaliar a adequação da proposta foram realizados estudos de caso em duas empresas de software em Recife. Os resultados encontrados sugerem que a abordagem proposta facilitou a priorização de requisitos por ser um processo simples e intuitivo.

Palavras-chave

Requisitos, Kano, Priorização, Scrum, Metodologias Ágeis.

Abstract

Nowadays, software companies need to develop systems quickly and efficiently. The growing competition stimulates companies to adopt agile methodologies to ensure their competitive advantage and increase client satisfaction. The requirements prioritization is a very critical task in the software development. This task involves each requirement's significance analysis by the stakeholders as well as the requirements breakdown classification into each of software versions to be released. A wrong decision about which requirements should be prioritized could affect negatively the overall quality of the system and, consequently compromise the client's acceptance.

In agile development, it is recommended to use the client's indication to prioritize the requirements. This could lead to some issues, once the client may believe that the meaning of certain requirement is higher than it actually is, or even pointing a very desired requirement as having lower priority, and only miss it when the system is ready with all other functionalities except that one. Agile methodologies such as Scrum and XP offer very basic orientation on how to conduct this prioritization. Among others, the main limitation of the prioritization process described by the agile methodologies includes the difficulty of comparing the actual importance of the requirements as well as the lack of analysis whenever certain requirements are not selected.

The main goal of this research is to propose a requirements prioritization process to agile software projects based on the Kano technique, which was initially proposed in the administration field. The diagrams produced by Kano aim to prioritize requirements through a functional and a dysfunctional question to each requirement. This dissertation proposes a process to help the prioritization and improve the visualization of the prioritization results with charts and figures to make the decision making process easier for the stakeholders.

In order to evaluate the proposal, case studies were conducted in two software companies at Recife. The outcome suggests that the proposed approach have facilitated the requirements prioritization due to its simplicity and intuitive process.

Keywords

Requirements, Kano, Prioritization, Scrum and Agile.

Índice

Lista de Quadros

Lista de Figuras

Introdução

Neste capitulo serão apresentadas as principais motivações para realização desta dissertação, buscando contextualizar o objetivo principal da pesquisa e descrever as suas contribuições.

1.1 Motivação

Diversos estudos têm ressaltado que a fase de engenharia de requisitos (ER) é uma das etapas mais críticas do processo de desenvolvimento de software [Lamsweerde 2000]. O sucesso de sistemas depende de critérios básicos como atender ao prazo, custo e escopo estabelecidos inicialmente junto com o cliente. Em particular, a satisfação dos requisitos do cliente é um dos fatores críticos para o sucesso de sistemas. Para atingir este objetivo é necessário que o projeto entregue os requisitos que irão trazer maior satisfação para o cliente e aumentem o valor de mercado da solução final. Por isso priorização de requisitos é uma atividade crítica do desenvolvimento de software. Esta atividade envolve a análise da importância de cada requisito por parte dos *stakeholders* a seleção dos requisitos que irão ser implementados em determinada versão do sistema. A decisão errada sobre quais requisitos priorizar pode afetar a qualidade global do sistema e a sua aceitação conseqüentemente pelos clientes.

Atualmente, empresas de software precisam desenvolver sistemas de forma rápida e eficiente. A competição crescente tem levado empresas a adotar metodologias de desenvolvimento ágeis para garantir vantagem competitiva e aumentar a satisfação de clientes [Cockburn 2002]. Nos últimos anos, diversas metodologias ágeis tem sido propostas na literatura, tais como: *Extreme Programming* (XP), Scrum, *Crystal Clear, Feature Driven Development* (FDD), etc.

A partir da experiência pessoal do autor desta pesquisa através de sua atuação profissional em empresas de software, foi observado que estas empresas têm buscado utilizar metodologias ágeis para aumentar a sua competitividade e qualidade de suas soluções. A motivação para a realização desta pesquisa surgiu da experiência obtida trabalhando em uma destas empresas, onde foi observado o início de alguns projetos ágeis. Para garantir a correta assimilação das práticas ágeis em seu ambiente

organizacional, várias modificações foram realizadas nos processos da empresa. Em particular, foi constatado que o maior problema da empresa era durante a fase de engenharia de requisitos que, apesar de todos os processos terem sofrido muitas modificações, a fase de ER ainda sofria de vários problemas (elicitação, gerência, priorização entre outros). Analisando qual área da engenharia de requisitos seria mais interessante explorar nesta pesquisa, foi percebida a dificuldade latente para escolher a ordem de implementação dos requisitos em todos os projetos observados, fossem eles baseados em metodologias ágeis ou tradicionais.

Foi observado ainda que na maioria dos projetos que não utilizavam metodologias ágeis, essa priorização era feita apenas por motivos técnicos sendo desprezada a real satisfação do cliente sobre sua entrega. Enquanto que em projetos que utilizavam metodologias ágeis foi observado que o cliente usava a experiência do gestor para dizer o que era mais prioritário sem ter nenhum embasamento para tomar aquela decisão, nem mesmo a certeza do que ele realmente queria. Desta forma, foi percebido que a priorização de requisitos é um problema real enfrentado por empresas de software, além de ter sido amplamente discutida a sua importância na literatura.

1.2 Descrição do problema

De acordo com a explicação apresentada na seção anterior, a priorização de requisitos é um desafio principalmente em ambientes ágeis, pois tais projetos precisam que versões do sistema sejam entregues com freqüência e cada versão deve selecionar os requisitos mais prioritários para serem implementados. No entanto, a maioria dos projetos não captura de forma adequada o sentimento do cliente para indicar qual requisito deveria ser implementado primeiro. Isto gera inúmeros problemas tais como incluir vários requisitos com a prioridade máxima, não sendo possível diferenciar a real importância de cada requisito. Além disso, é impossível prever qual o impacto para o cliente caso determinado requisito não seja implementado.

Com o objetivo de definir melhor o escopo desta dissertação, foi escolhida a metodologia ágil Scrum[Schwaber 2004] como referência para elaborar a proposta deste trabalho. A justificativa para escolha do Scrum é devido ao enfoque desta metodologia ágil para apoiar as decisões relacionadas à gerência de projetos, estando mais próxima da tomada de decisão sobre quais requisitos serão priorizados para cada versão. As outras metodologias têm o foco em práticas de desenvolvimento, por exemplo, XP como iremos mostrar no capítulo 2 a seguir.

No caso de Scrum, os clientes classificam a importância dos requisitos numa escala de 0 a 1000, e acontece com freqüência ter vários requisitos com o valor 1000 outros com o valor 750 e ainda outros com o valor 500. Geralmente, é desta maneira que a prioridade de todos os requisitos é distribuída. Dessa forma, não se sabia realmente a diferença entre implementar um ou outro requisito com prioridade de 750, ou mesmo se quando o cliente estipulou aquele número ele tomou como importante para o produto era de fato uma necessidade dos usuários e outros *stakeholders*.

Diante destes problemas, torna-se necessário propor formas mais intuitivas e eficazes para capturar a real importância de cada requisito a ser priorizado em projetos ágeis. A melhoria da priorização de requisitos pode trazer benefícios tanto para as equipes de desenvolvimento que irão saber a real prioridade dos requisitos como também garantir que o cliente irá receber um produto com funcionalidades que satisfazem as suas reais necessidades e expectativas.

1.3 Contribuições

Esse trabalho tem como objetivo propor um processo para auxiliar a priorização de requisitos para projetos de software ágeis. Para atingir este objetivo, este trabalho adaptou a abordagem Kano[Cohn 2005], técnica originalmente proposta na área de administração. Neste contexto, esta dissertação contribui para melhorar a forma como as metodologias ágeis podem ser usadas na prática, facilitando a priorização e a seleção de requisitos que irão compor cada versão do sistema.

As principais contribuições desta dissertação são:

- Propor um processo de priorização de requisitos para ambientes ágeis baseado na técnica Kano. Esta abordagem visa uma interpretação baseada em gráficos que mostram de forma diferente os resultados do Kano tradicional. Além de evoluir a técnica buscando novas interpretações para deixar a técnica mais fácil de ser usada, também foi proposto um guia para a sua aplicação e interpretação em projetos ágeis.
- Realização de estudos de caso em duas empresas de software localizadas em Recife. As empresas já possuem experiência na adoção de metodologias ágeis. Em particular, os projetos escolhidos utilizaram a metodologia Scrum. No entanto, até o momento do estudo, as empresas não utilizavam nenhuma técnica especial para priorizar os requisitos de cada versão do sistema.

- Outra contribuição deste trabalho foi colaborar com as empresas estudadas para melhorar seus processos de priorização de requisitos. De acordo com relatos dos participantes do estudo, o processo proposto está sendo atualmente implantado nos projetos ágeis das empresas. Este tipo de iniciativa pode se caracterizar como estímulo para transferência tecnológica a fim de ampliar a colaboração entre projetos de pesquisa e empresas locais.

1.4 Organização da dissertação

Esta dissertação está organizada em seis capítulos. O presente capítulo apresentou uma introdução sobre as motivações, descrição do problema e as contribuições da pesquisa.

No capítulo 2 são introduzidas as metodologias ágeis seus princípios e valores. Além disso, são apresentadas as metodologias XP e Scrum.

No capítulo 3 é apresentado o processo de engenharia de requisitos, além de descrever as principais abordagens para priorização de requisitos existentes na literatura.

No capítulo 4 é apresentada a técnica Kano de priorização de requisitos e sua adaptação para ambientes ágeis.

No capítulo 5 são apresentados os estudos de caso realizados nas empresas Provider e no C.E.S.A.R .

No capítulo 6 são feitas as considerações finais, trabalhos futuros, limitações e contribuições deste trabalho.

Metodologias Ágeis

Este capítulo apresenta as metodologias ágeis e descreve abordagens usadas para gerenciar projetos de software utilizando metodologias ágeis. As abordagens de desenvolvimento ágil foram formalmente apresentadas para a comunidade através do manifesto ágil que ocorreu em 2000 [Agile Manifesto]. O manifesto ágil estabelece um conjunto de valores que são adotados nos projetos ágeis, são eles:

- **Indivíduos e interações** ao invés de **processos e ferramentas**;
- **Software funcionando** ao invés de **documentação abrangente**;
- **Colaboração com o cliente** ao invés **de negociação de contratos**;
- **Responder a mudanças** ao invés de **seguir um plano**.

De acordo com o manifesto, embora exista valor nos itens à direita(acima em negrito), os processos ágeis valorizam mais os itens que estão à esquerda(acima em negrito).

Atualmente, estes são os principais processos ágeis conhecidos: Scrum[Schwaber 2004], Dynamic Systems Development Method (DSDM) [Abrahamsson 2002], Crystal Methods[Cockburn 2004], Feature-Driven Development (FDD)[Palmer 2002], Lean Development (LD)[Poppendieck 2003], Extreme Programming[Johnson 2002] e Adaptative Software Development [Highsmith 2002]. Nesse trabalho serão apresentados os métodos Extreme Programming(XP) e Scrum por serem os métodos mais utilizados nas empresas pesquisadas.

O que torna o processo ágil diferente do processo de desenvolvimento tradicional é o que é chamado de *barely sufficient*, ou seja, mínimo necessário. Enquanto abordagens mais tradicionais como *Rational Unified Process[Rup 2002]* ou Processo Unificado da Rational procuram estabelecer um arcabouço de "melhores práticas", os processos ágeis definem poucas práticas. Isto torna os processos ágeis mais simples de serem adotados sem ter uma necessidade de reduzir o processo para ser utilizado. Situação que, na maioria das vezes, causa desconforto nas áreas em que os processos são reduzidos ou adaptados. A ênfase está em trazer elementos novos para o projeto somente quando os mesmos realmente se mostram necessários [Boehm

2003][Cockburn 2002][Highsmith 2002]. Nas próximas seções deste capítulo serão apresentados os principais valores ágeis e serão descritas as abordagens XP e Scrum.

1.5 Valores Ágeis

1.5.1 Feedback

A psicologia do aprendizado ensina que o tempo entre uma ação e o correspondente *feedback* é crítico para o aprendizado. Segundo [Beck 2000], experimentos com animais mostram que mesmo pequenas diferenças no tempo de *feedback* resultam em enormes diferenças de aprendizado. Portanto, um dos mais importantes princípios ágeis é obter *feedback*, interpretá-lo e colocar o que foi aprendido de volta no processo de desenvolvimento do sistema o mais rapidamente. Os programadores aprendem como fazer o *design*, programar e testar o sistema da melhor forma possível e retornam este aprendizado em segundos ou minutos ao invés de dias, semanas ou meses.

A compreensão das necessidades dos usuários é um processo de aprendizado contínuo em que os desenvolvedores aprendem sobre os problemas do negócio e os usuários tomam conhecimento das dificuldades e limitações técnicas. Segundo [Weinberg 1971] para atingir essa compreensão dos usuários, um princípio psicológico bem conhecido indica que para maximizar a taxa de aprendizado, a pessoa precisa receber *feedback* sobre quão bem ou mal ele está indo.

Usando o princípio de *feedback* contínuo, metodologias ágeis como o Extreme Programming, propõe a organização em ciclos curtos de *feedback* que possibilitam aos usuários solicitar funcionalidades e aprender sobre elas através de software funcionando em prazos curtos. Já o Scrum utiliza o conceito de *Sprints*, ou seja, iterações que funcionam de forma curta, apenas usando um termo diferente. Esse processo envolve a priorização de poucas funcionalidades a serem implementadas de cada vez e a simplificação das mesmas na medida do possível. O objetivo se torna apresentar a funcionalidade ao usuário rapidamente, de modo que ele possa, o mais cedo possível, detectar eventuais falhas enquanto ainda é mais barato corrigi-las. A razão básica para estratégias incrementais e iterativas é permitir que os inevitáveis erros das pessoas sejam descobertos relativamente cedo e reparados de forma metódica [Cockburn 2002].

Através dos ciclos de *feedback* curtos, o Extreme Programming e o Scrum procuram assegurar que pouco trabalho seja efetuado e concluído de cada vez. A equipe verifica através do *feedback* do trabalho realizado as possíveis falhas e as mesmas são corrigidas o mais rápido possível. Caso tudo esteja correto, a equipe continua o desenvolvimento. A utilização de pacotes reduzidos de trabalho assegura que eventuais falhas poderão ser corrigidas com maior rapidez. Exatamente porque o escopo do trabalho é reduzido, o que significa que menos coisas podem dar errado.

1.5.2 Comunicação

Com freqüência, equívocos no processo de comunicação causam desentendimentos ou compreensão incorreta de algum aspecto do projeto. A especificação de requisitos, por exemplo, envolve um processo de comunicação de conhecimentos tácitos, o que explica grande parte da dificuldade do desenvolvimento de software [Melo 2008]. Traduzir conhecimento de um contexto para outro, como traduzir qualquer língua, não envolve apenas gramática básica e regras sintáticas, mas também questões de significado e intenção que são contextuais e subjetivas.

Segundo [Eischen 2002], o desenvolvimento de sistemas de software envolve um exercício de traduzir algoritmos existentes – sobre o ambiente, organizações ou práticas – para a forma digital. Grande parte deste conhecimento sobre o domínio da aplicação é tácito, indefinido, não codificado e desenvolvido ao longo do tempo, freqüentemente sem ser explícito até mesmo para os indivíduos que participaram do processo. Além disso, o conhecimento e as práticas organizacionais são dinâmicos evoluindo constantemente e se transformando.

A transmissão de conhecimento tácito representa um desafio significativo para as equipes de desenvolvimento, o qual pode ser solucionado de forma mais ou menos eficaz dependendo dos mecanismos de comunicação adotados no projeto. Segundo [Teles 2004], a forma de se transmitir uma idéia exerce uma grande influência na compreensão correta da mesma. De acordo com [Jeffries 2001], existem diversos estudos que mostram que a comunicação é reduzida enormemente quando há saída ou trocas entre os membros da equipe. Mover colegas um andar abaixo reduz a comunicação quase tanto quanto se os movêssemos para o outro lado do mundo.

Os projetos ágeis apresentam a comunicação como um grande problema que deve ser bem resolvido através do envolvimento direto dos usuários (ou pelo menos um representante dos mesmos) como parte integrante da equipe de desenvolvimento. Na

prática, isto significa que o usuário (ou seu representante) está presente no mesmo local onde os desenvolvedores trabalham, possibilitando que eles tenham acesso rápido e direto à um ou mais especialistas no domínio do negócio. Segundo [Teles 2005], isso ajuda a acelerar o fluxo de informações e permite que a comunicação seja predominantemente feita através de diálogos presenciais. A transferência de conhecimento tácito torna a comunicação muito mais efetiva possibilitando principalmente que o software seja desenvolvido baseado na necessidade real do cliente.

De acordo com [Cockburn 2002], a rapidez na comunicação é um aspecto relevante para o sucesso de projetos de software, pois o ritmo de progresso de um projeto está ligado ao tempo que se leva para transmitir uma informação de uma pessoa para outra. Além disso, os custos de um projeto crescem na proporção do tempo necessário para as pessoas se compreenderem e entenderem quais atividades devem ser realizadas.

Segundo [Cockburn 2002], outro fator que influencia a qualidade da comunicação é a quantidade de pessoas envolvidas. À medida que a equipe aumenta, torna-se mais difícil para as pessoas saber o que os outros estão fazendo e como não sobrepor, duplicar ou interferir no trabalho do outro. Por esta razão, projetos ágeis procuram contar com um número reduzido de participantes, freqüentemente menor do que uma dúzia de pessoas em cada equipe. No caso de projetos maiores são feitas várias equipes mantendo a comunicação no mesmo nível.

1.5.3 Envolvimento do Cliente

Para que um projeto de software seja concluído com sucesso é necessário que se tenha a participação ativa de todas as partes envolvidas. O cliente e a equipe de desenvolvimento têm um papel fundamental nesse contexto. O cliente pode ser envolvido para explicar exatamente o que ele deseja, quais as suas expectativas e necessidades. O cliente também deve participar do projeto e acompanhar o desenvolvimento, mas se a equipe não desenvolver um bom software ou não tiver capacidade técnica para desenvolver da maneira como o cliente deseja, o projeto não terá sucesso. Da mesma forma se a equipe tiver excelentes desenvolvedores que estão empenhados no sucesso do projeto e fazem sempre o que o cliente pede, mas o cliente não está envolvido e não se preocupa ou não tem tempo para esclarecer as dúvidas de

negócio, a equipe pode correr o risco de elaborar um software que nunca vai ser utilizado por ninguém porque não se encaixa na necessidade do negócio. Isto significa que cliente e equipe de desenvolvimento são interdependentes e que o sucesso do sistema somente é possível através da cooperação entre as duas partes.

De acordo com pesquisas recentes, é importante a participação não somente do cliente (i.e. quem paga pelo sistema), mas também dos usuários (i.e. quem irá utilizar o sistema no seu dia-a-dia). Segundo [Kujala 2005], a falta de envolvimento dos usuários têm sido uma das principais causas de falhas nos projetos. Mesmo quando entregue no prazo e dentro do orçamento, um projeto pode falhar se não tratar das reais necessidades e expectativas dos usuários. [Standish 2001]

Além disso, de acordo com [Teles 2004], essa proximidade ainda traz como benefício a melhoria na relação de confiança entre as partes envolvidas. Estar fisicamente próximo permite que o cliente perceba mais facilmente os esforços da equipe de desenvolvimento e melhore o relacionamento entre as partes. Além disso, compartilhar os sucessos e fracassos ao longo do projeto também ajuda a elevar a satisfação final do cliente.

Apesar de o ganho ser considerável e ser amplamente provado que a presença física do cliente nos projetos é um fator importante para o sucesso do mesmo, nem sempre isso é possível de ser feito na prática. Projetos de desenvolvimento distribuído não permitem que o cliente esteja fisicamente próximo da equipe de desenvolvimento. Neste caso, a comunicação deve ser realizada através de mecanismos e ferramentas de cooperação.

1.5.4 Simplicidade

Durante a terceira Conferência Internacional sobre Extreme Programming, [Johnson 2002] apresentou um estudo revelador sobre a utilização das funcionalidades nos projetos que foram pesquisados pelo Standish Group. Os resultados mostram que 45 % das funcionalidades encontradas em um sistema típico jamais são usadas enquanto 19 % raramente são usadas. Na Figura 2.1 é possível ver o resultado dessa pesquisa em forma gráfica.

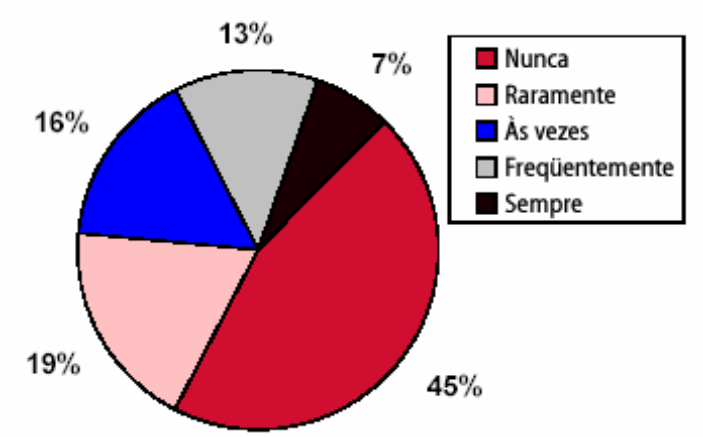


Figura 2.1 Estatística sobre a utilização de funcionalidades de acordo com [Johnson 2002]

Uma das conclusões do estudo é que cerca de 64 % das funcionalidades não necessitariam ter sido desenvolvidas. Em outras palavras, os projetos de software freqüentemente investem grande parte dos recursos (tempo, pessoas e dinheiro) em esforços desnecessários.

De acordo com [Teles 2005], em diversos projetos observa-se a ocorrência de três fenômenos que ajudam a esclarecer as razões dos resultados acima:

1) O escopo é fixado no início e alterações no mesmo são evitadas;
2) Os desenvolvedores criam soluções genéricas para facilitar possíveis alterações que possam ocorrer no escopo;
3) Os desenvolvedores produzem funcionalidades adicionais na tentativa de antecipar o que o usuário certamente irá solicitar no futuro.

No primeiro caso é facilmente justificável na situação em que o cliente exige o escopo fechado com medo de não receber todas as funcionalidades que ele julga indispensáveis, acarretando a decisão de fixar o escopo logo no início do projeto. No segundo caso, o desenvolvedor teme por uma mudança de última hora que possa causar uma grande alteração no sistema que não tinha sido feito bem flexível. Enquanto que terceiro caso ele tenta prever funcionalidades que "provavelmente" serão necessárias (embora o cliente ainda não perceba isso) e as desenvolve de modo que, quando o cliente as solicitar, as funcionalidades já estarão prontas.

Estes problemas poderiam ser sido evitados com iterações curtas com freqüente *feedback* do usuário, garantindo um escopo reduzido para cada iteração e requisitos priorizados de acordo com o real valor de negócio de cada requisito.

Os desenvolvedores de um projeto XP constroem as funcionalidades priorizadas para cada iteração com a maior qualidade possível, porém com foco somente naquilo que é claramente essencial. Funcionalidades que não se provem

imediatamente necessárias são evitadas, pois de acordo com [Jeffries 2001], se o desenvolvedor mantiver o sistema suficientemente simples o tempo todo, qualquer modificação feita nele será inserida facilmente e afetando o mínimo de locais possível.

De acordo com a prática ágil de simplicidade, qualquer funcionalidade a mais colocada sem a devida necessidade pode tornar o sistema mais complexo ou talvez sem sentido, a não ser que a mesma tenha sido solicitada pelo cliente. Procurar prever futuras necessidades de clientes não é uma boa prática, podendo exigir novas implementações sem a devida necessidade.

1.5.5 Gerenciamento de Riscos

Metodologias ágeis, e em particular XP, buscam gerenciar riscos estimulando a equipe a ter coragem e confiança nos processos utilizados no XP e acreditar que serão capazes de resolver esses possíveis problemas. XP parte do princípio que os problemas irão ocorrer, no entanto, a equipe utiliza mecanismos para reduzir ou eliminar as conseqüências desses problemas.

Com relação ao receio do cliente, caso o projeto não atenda ao seu pedido, a metodologia adota iterações curtas (de uma a três semanas) e durações fixas. Desta forma, é analisado ao fim de cada iteração se realmente a equipe desenvolveu o solicitado e se o resultado foi realmente o que o cliente esperava. Caso o que foi implementado não tenha sido correspondente ao que foi pedido, o impacto pode ser de no máximo uma iteração perdida o que é bem menos traumático do que se perder um projeto inteiro.

O desenvolvimento ágil também ajuda a lidar com o medo que o cliente tem de pagar demais recebendo muito pouco. Ao receber funcionalidades com freqüência, em prazos curtos, o cliente passa a ter diversas oportunidades de avaliar o trabalho da equipe com base em *feedback* concreto: software executável. Assim ele pode decidir se continua ou não a empregar aquela equipe ou se é preferível trocar [Teles 2004]. Além disso, o *feedback* constante, produzido ao longo das iterações, faz com que o cliente possa saber exatamente o que está acontecendo no projeto.

Finalmente, o processo de planejamento não é estático. A cada início de iteração o planejamento geral do projeto é revisado e atualizado com base em informações mais recentes. Isto é, o processo de planejamento é contínuo e procura incorporar *feedback* ao longo do tempo. Isso permite a elaboração de planos para cada

iteração que têm maiores chances de acerto. Além disso, no processo de priorização, segundo [Beck 2001], o cliente pode incorporar novas decisões de negócios de forma natural.

Outro risco que é tratado no XP é a de novas implementações afetarem negativamente as entregas já realizadas. Com o objetivo de minimizar este risco foi criado o desenvolvimento orientado a testes. Também chamado de *Test Driven Development* (TDD), ou seja, o projeto é iniciado pelos casos de testes automáticos e se considera que uma funcionalidade foi concluída no momento em que todos os testes passam. Esses testes são executados sempre ao fim de cada funcionalidade para garantir que nada que tinha sido desenvolvido anteriormente foi afetado pelas novas funcionalidades.

Com relação à questão de tempo para desenvolver as solicitações, as práticas recomendadas pelas metodologias ágeis tratam este risco dividindo claramente a responsabilidade por decisões técnicas e de negócio. O cliente é responsável pelas decisões de negócio, ou seja, determina os requisitos e qual a prioridade de suas necessidades. Enquanto que a equipe é responsável por estimar os prazos fazendo com que eles se comprometam e principalmente escolham prazos mais realistas.

Extreme Programming (XP)

O XP teve origem na década de 80, com Kent Beck e Ward Cunningham [Teles 2005]. Suas principais práticas são: refatoração, programação em par, mudanças rápidas, feedback constante do cliente, desenvolvimento iterativo, testes automatizados, entre outras [Teles 2005]. Segundo uma pesquisa realizada no 3º relatório sobre o estado do Desenvolvimento Ágil [Asosad 2009], promovido pela [Versionone 2009] o método XP engloba 79,4% da utilização das abordagens ágeis dentre 3061 empresas estudadas em todo mundo. Essa metodologia é muito utilizada em sistemas com o escopo não definido e em sistemas onde a equipe tem a necessidade de trabalhar de forma ágil. Pode ser utilizado em qualquer tipo de aplicação nessas condições. Algumas práticas do XP são:

1.6.1 Jogo do Planejamento

Decidir o que implementar é uma das atividades mais importantes do desenvolvimento de um sistema. Estudos do Standish Group [Standish 2001] demonstram a importância da priorização de requisitos para o desenvolvimento de software. Segundo [Yourdon 2004], as funcionalidades de um sistema podem ser categorizadas em "tem que ser feita", "deveria ser feita" e "poderia ser feita". Cabe aos membros do projeto assegurar que as funcionalidades com maior prioridade sejam implementadas em primeiro lugar. No capítulo 3, serão apresentadas várias abordagens para priorizar requisitos.

O XP divide o projeto em versão e cada versão em iterações. As versões duram normalmente meses e as iterações duram, em média, duas semanas. Dentro dessas iterações são colocadas algumas funcionalidades que são apresentadas dentro do projeto através de estórias.

Versão → Conjunto de Iterações

Iteração → Conjunto de Funcionalidades

Funcionalidades → Divididas em estórias

Como já foi mencionado, o cliente e a equipe têm suas responsabilidades[Beck 2001]. As responsabilidades do cliente incluem:

- Definir as estórias;
- Decidir qual o valor de negócio de cada estória;
- Decidir que estórias serão construídas no release.

As responsabilidades da equipe envolvem:

- Estimar quanto tempo será necessário para construir cada estória;
- Advertir o cliente sobre riscos técnicos significativos;
- Medir o progresso da equipe para fornecer um orçamento geral para o cliente.

Outra responsabilidade do cliente é de posse da estimativa da equipe priorizar o que será feito essa decisão é feita dependendo do esforço de cada estória e principalmente, avaliando o valor de negócio. Ainda no planejamento quando decidido o que vai entrar naquela iteração é definido a produtividade baseado em iterações passadas e do conhecimento do time.

Segundo [Beck 2001], a velocidade do time representa basicamente a quantidade de trabalho que a equipe é capaz de entregar em uma iteração. No início

de uma iteração, o modelo de planejamento do XP assume que a equipe será capaz de entregar a mesma quantidade de estórias efetivamente desenvolvidas na iteração anterior. Contudo, são selecionadas as estórias mais importantes até que o tempo do projeto acabe ou todas as estórias, deixando assim, as de menor valor para o final.

1.6.2 Reuniões diárias

Com o objetivo de assegurar que as partes trabalhem bem em conjunto, o XP utiliza uma breve reunião diária chamada de *stand up meeting*. O objetivo da reunião é alinhar com os membros da equipe os objetivos daquele dia, informando os resultados obtidos no dia anterior e permitindo que os participantes priorizem as atividades do dia que se inicia.

Essa técnica gera visibilidade de tudo o que ocorre na equipe para todos os seus membros, o que permite identificar rapidamente problemas e soluções que foram encontrados no dia anterior. É uma forma de disseminar conhecimento e assegurar que as pessoas tenham acesso às informações mais recentes à medida que o projeto prossegue.

Segundo [Williams 2003], estas reuniões curtas são excelentes para desenvolver o espírito de equipe e comunicar, bem como para determinar quem irá trabalhar em dupla com quem durante o dia. [Beck 2001] considera que reuniões diárias curtas são inestimáveis para dar a cada pessoa uma idéia do que os outros estão fazendo. Cada pessoa diz brevemente o que fez no dia anterior e o que está fazendo hoje. Problemas e anúncios importantes para a equipe também são passados. O objetivo da reunião é comunicar problemas e não resolvê-los. Tudo que demandar qualquer coisa além de um breve anúncio deve ser reservado para outra sessão onde apenas aqueles interessados no assunto devem estar presentes.

1.6.3 Programação em Par

De acordo com [Williams 2003] a programação em par é um estilo de programação no qual dois programadores trabalham lado a lado em um computador, continuamente colaborando no mesmo projeto, algoritmo, código e teste. A programação em par é utilizada por todos os desenvolvedores durante toda a duração de um projeto XP. Na programação realizada em pares recomenda-se que se coloque um programador mais sênior e outro júnior fazendo com que o sênior acabe

ensinando outras formas mais rápidas ou mais eficientes de desenvolver aquele código. Melhorando o desenvolvimento técnico da equipe.

De forma semelhante, a programação em par também é uma ótima estratégia de gestão do conhecimento – uma excelente maneira de passar conhecimento tácito pela equipe [Williams 2003]. Como observamos anteriormente, a disseminação de conhecimento tácito é essencial para o bom andamento de um projeto. Programação em par funciona para o XP porque encoraja a comunicação entre os membros da equipe.

Embora a programação em par ofereça oportunidades de melhoria nos projetos, de acordo com [Williams 2003], a reação intuitiva de muitas pessoas é rejeitar a idéia porque assumem que haverá um aumento de cem por cento de programador-hora colocando dois programadores para fazer o trabalho que um pode fazer. Entretanto, pesquisas demonstram que isso não ocorre na prática. Williams e Kessler (2003) executaram experimentos, com resultados estatisticamente significativos, demonstrando que a programação em par eleva o número de programador-hora em apenas 15 por cento. Além disso, [Williams 2003] conseguiu validar estatisticamente os relatos que alegam que a programação em par é um meio acessível de produzir software de mais elevada qualidade. Ou seja, apesar da ligeira elevação no consumo de recursos, os resultados mostraram que os desenvolvedores que trabalharam em par geraram um número significativamente menor de defeitos, produziram menos código para solucionar o mesmo problema e tiveram resultados mais consistentes.

Já [Stephens 2003], acredita que essa prática não deve ser adotada durante o tempo todo (como dito na metodologia XP) porque o programador necessita estar isolado para se concentrar. Ele mostra que um ambiente isolado e calmo a criatividade flui mais facilmente, ajudando na programação.

1.6.4 Inspeção de Código

Todo código criado em pares será obrigatoriamente inspecionado por pelo menos um programador exceto o que criou o código que pode identificar várias falhas evitando todo o retrabalho e burocracia que uma detecção tardia do erro poderia causar.

Para [Emam 2003], revisões em pares (também conhecidas como inspeção do código) representam uma das formas mais eficazes para encontrar defeitos em

software. As evidências que dão suporte às revisões em pares remontam a mais de duas décadas de pesquisas na área. Existem indicações de que inspeções de software tradicionais não são tão prevalentes na prática. A inspeção de código é uma forma de institucionalizar as revisões em pares dentro dos projetos de software o que seria uma grande melhoria da qualidade para a maioria dos projetos.

As revisões de código são tão importantes como as outras revisões como a de documentos por exemplo. No entanto, quando elas não são usadas da forma correta se tornam não só um desperdício de tempo como um trabalho que gera um alto grau de insatisfação da equipe. Porque o time não consegue visualizar o ganho de se utilizar essas revisões tornando um trabalho cansativo e sem o ganho esperado.

1.6.5 Código Coletivo

Complementando a programação em par, equipes XP também praticam o conceito de código coletivo de acordo com [Jeffries 2001], todas as classes e métodos pertencem à equipe e qualquer membro da equipe pode melhorar o que for necessário. O desenvolvedor não apenas programa em par com diferentes pessoas ao longo do tempo, como também tem acesso a todas as partes do código, inclusive aquelas que ele não programou. Além disso, tem o direito de fazer qualquer alteração que considerar necessária sem ter que pedir permissão. Naturalmente, seus colegas também podem alterar o seu código sem lhe pedir [Beck 2000].

Esta prática também protege o projeto ajudando a tornar a equipe mais robusta, na medida em que os desenvolvedores se habituam a trabalhar nas mais variadas partes do sistema. Sendo assim, todo o trabalho fica menos suscetível de ser afetado se alguém ficar doente ou faltar. Isso não apenas reduz as variações no cronograma, mas também eleva as chances de ter alguém por perto quando o código tiver que ser modificado [Weinberg 1971].

A propriedade coletiva do código tem a tendência de evitar que códigos complexos entrem no sistema. Se você sabe que outra pessoa irá olhar para o que você está escrevendo dentro de pouco tempo (em algumas horas), você irá pensar duas vezes antes de colocar no sistema uma complexidade que você não conseguirá justificar [Beck 2000].

1.6.6 Código Padronizado

Em projetos XP, os programadores codificam seguindo um padrão de código acordado. De acordo com [Jeffries 2001], não importa muito o formato. O que realmente importa é que todos os membros da equipe adotem o padrão e o utilizem sempre.

Padrões de código levam a uma uniformidade de código que facilita a manutenção uma vez que o código passa a ser mais facilmente entendido pela pessoa que não desenvolveu aquele código e ainda é importante para evitar que alguns erros sejam cometidos quando um desenvolvedor pensa numa nova solução a probabilidade de ter mais erros é muito grande.

A adoção de um padrão ajuda a simplificar a comunicação, a programar em par e a tornar o código coletivo. Da mesma forma, a própria programação em par e a prática de código coletivo ajudam a assegurar que a equipe irá seguir o padrão adotado. Como o código passa por vários níveis de revisão, é fácil detectar e corrigir qualquer código fora do padrão. Porém, para assegurar que o padrão realmente seja usado, qualquer padrão envolvido deve surgir e evoluir no nível das próprias pessoas que realizam o trabalho. A propriedade do padrão deve estar nas mãos daqueles que realizam o trabalho [Demarco 2001].

1.6.7 Design Simples

Um software gera de expectativas para os usuários de que tenha os seguintes atributos: eficiência, eficácia, usabilidade e evolução contínua. As práticas do XP se complementam formando um conjunto que busca atingir estes objetivos. Para assegurar que o sistema esteja fazendo a coisa certa e esteja funcionando, o desenvolvimento é executado em iterações curtas nas quais se implementa um pequeno conjunto de estórias. O feedback gerado ao final de cada iteração permite que os usuários avaliem se o sistema realmente está fazendo a coisa certa e se está funcionando.

Adotar iterações curtas, portanto, é um mecanismo importante para atingir parte destes objetivos, entretanto, impõe um desafio. Como fazer a análise, design, implementação, teste e depuração de um conjunto de funcionalidades em uma ou duas semanas? Isso só é possível se os desenvolvedores mantiverem o design da aplicação suficientemente simples [Beck 2000].

Portanto, embora o desenvolvimento iterativo seja útil para alcançar alguns dos objetivos do projeto, parece ser uma proposta arriscada quando se deseja construir um sistema fácil de ser utilizado e que também possa evoluir ao longo do tempo. Na prática, entretanto, as iterações provêm oportunidades para que a arquitetura seja revisada, aprimorada e amadureça com o tempo. Clientes de um sistema de software geralmente não são capazes de definir o que irão perceber como sendo integridade, assim como eles não sabem descrever de forma precisa o que desejam em um carro. Os clientes sabem quais são seus problemas, mas com freqüência, não conseguem descrever a solução. Eles saberão reconhecer um bom sistema quando o encontrarem, mas eles não conseguem visualizá-lo previamente. Para piorar as coisas, à medida que suas circunstâncias mudarem, também mudará suas percepções sobre a integridade do sistema [Poppendieck 2003]. À medida que novas funcionalidades são adicionadas a um sistema para manter a integridade perceptível, as características subjacentes da arquitetura para acomodar as funcionalidades de maneira coesa também precisam ser modificadas.

As equipes XP procuram adiar decisões de design tanto quanto possível. Ou seja, implementam o design mais simples possível para os problemas atuais. Sofisticações que possam ser necessárias para acomodar necessidades futuras não são implementadas até que se atinja um ponto no desenvolvimento no qual a equipe irá implementar de fato as estórias que demandem tais sofisticações. Desta forma, se reduz as chances de adicionar código desnecessário e elevar a complexidade do sistema cedo demais [Poppendieck 2003]. O problema desta abordagem é que algumas alterações podem acabar se revelando custosas no futuro.

Se o custo de alterar o software se elevar exponencialmente ao longo do tempo, a abordagem iterativa pode ser arriscada e muito custosa. Entretanto, se for possível mantê-lo baixo, tal abordagem pode ser incorporada nos projetos de software com resultados positivos[Teles 2005]. Já [Poppendieck 2003] afirma que, boas práticas arquiteturais podem reduzir significativamente o custo de mudança confinando funcionalidades que são prováveis de sofrerem alterações em áreas pequenas e bem encapsuladas.

1.6.8 Desenvolvimento Orientado a Testes

O desenvolvimento orientado a testes trata-se de uma técnica preventiva utilizada durante todo o projeto e a manutenção do sistema. Consiste em elaborar

testes automáticos para todos os testes unitários do sistema. Os testes automatizados procuram comprovar que as solicitações dos usuários estão sendo atendidas de forma correta. Além disso, os testes verificam se as estórias continuam funcionando ao longo do tempo, pois de acordo com [Hunt 2003] além de assegurar que o código faça o que você quer, você precisa assegurar que o código faça o que você quer o tempo todo. Testes são mais valiosos quando o nível de estresse se eleva, quando as pessoas estão trabalhando muito, quando o julgamento humano começa a falhar. Portanto, os testes têm que ser automatizados.

Se os testes não forem automatizados ou se tomarem muito tempo, eles não serão executados com suficiente freqüência. Grandes lotes de mudanças serão implementados antes de testar, o que tornará muito mais provável a ocorrência de falhas e será bem mais difícil identificar que mudança fez com que os testes falhassem [Poppendieck 2003].

Por maiores que sejam os investimentos em testes unitários e funcionais, equipes XP eventualmente encontram *bugs* que não são detectados pelos testes. Nestes casos, os desenvolvedores criam testes que exponham os defeitos antes de corrigi-los. Desta forma, se protegem do caso em que um mesmo *bug* volte a ocorrer no futuro. Se ocorrer, será identificado rapidamente.

Novas funcionalidades e eventuais correções inseridas em um código têm o potencial de adicionar novos defeitos no mesmo. Por essa razão, segundo [Brooks 1995], após cada correção ou desenvolvimento de nova funcionalidade, deve-se executar a base de casos de teste inteira para assegurar que o sistema não foi danificado de uma forma obscura. Ainda segundo [Brooks 1995], vale a pena construir diversas proteções de depuração e código de teste, talvez até mesmo 50 por cento do produto que está sendo depurado. Isso é particularmente verdadeiro no caso de se desenvolver o software de forma iterativa. Se você desenvolve software em iterações você irá fazer mudanças sérias no código após ele ter sido escrito. Isso é perigoso. Para fazer mudanças de forma segura, é necessário haver uma forma imediata para encontrar e corrigir conseqüências indesejáveis. A forma mais eficaz de facilitar mudanças é ter uma base de testes automatizados. Uma base de testes irá encontrar conseqüências indesejáveis imediatamente e se for boa, também irá indicar a causa do problema [Poppendieck 2003].

Infelizmente, de acordo com [Beck 2000] é impossível testar absolutamente tudo, sem que os testes se tornem tão complicados e propensos a erros quanto o

código de produção. É suicídio não testar nada. Você deve testar coisas que possam vir a quebrar. A idéia de Beck é que ao longo do tempo, e com a prática, os desenvolvedores de um projeto se tornem cada vez mais capazes de identificar que tipo de situações tendem a gerar mais erros. São nelas que eles irão concentrar os maiores esforços de teste.

Escrever um teste antes de cada estória também representa uma forma de aprimorar a análise sobre as características dela. Pois a necessidade de implementar um teste força o desenvolvedor a buscar maiores detalhes sobre o comportamento da funcionalidade [Beck 2000]. O desenvolvimento orientado a testes produz um desenvolvimento de um senso crítico do desenvolvedor que evita muitos erros uma vez que ele ao fazer o código e implementar os testes automáticos ele está sempre pensando em como poderia quebrar aquele código. Sendo assim ele consegue visualizar muitas situações de utilização do código que naturalmente ele não pensaria se estivesse apenas fazendo o código sem nenhum compromisso.

1.6.9 Refatoração

Sistemas mudam ao longo do tempo, especialmente quando são desenvolvidos de forma iterativa. Tradicionalmente, segundo [Brooks 1995], os reparos tendem a destruir a estrutura, elevar a entropia e a desordem de um sistema. Além disso, segundo [Poppendieck 2003], sem melhoria contínua, qualquer sistema de software irá sofrer. As estruturas internas irão se calcificar e se tornar frágeis. Por esta razão, projetos XP investem diariamente no aprimoramento do design e na identificação de pontos da arquitetura que estejam se degradando. À medida que são encontrados, são corrigidos através de uma técnica conhecida como refatoração.

A necessidade de refatoração aparece à medida que a arquitetura evolui, amadurece e novas funcionalidades são solicitadas pelos usuários. Novas funcionalidades podem ser adicionadas ao código uma de cada vez, mas geralmente elas estarão relacionadas umas com as outras e freqüentemente será melhor adicionar um mecanismo arquitetural para dar suporte ao novo conjunto de funcionalidades. Isso comumente acontece de forma natural como uma refatoração para remover duplicação quando você adiciona o segundo ou terceiro de um conjunto de itens relacionados [Poppendieck 2003].

Durante a refatoração de um código, de acordo com [Fowler 2000], cada passo é simples, até mesmo simplista. Apesar disso, o efeito cumulativo destas pequenas

mudanças pode melhorar significativamente o design. Fazendo isso, os desenvolvedores procuram evitar situações nas quais os programas se tornem difíceis de serem modificados.

É importante observar que a integridade conceitual, que de acordo com [Poppendieck 2003], significa que os conceitos centrais do sistema trabalham em conjunto como um todo harmônico e coeso é um fator crítico para a criação da percepção de integridade, sofre à medida que o design se deteriora. Como vimos anteriormente, a integridade conceitual é um fator essencial para tornar o sistema fácil de ser utilizado. Além disso, de acordo com [Brooks 1995], a integridade conceitual de um produto não apenas o torna mais fácil de usar, como também facilita a sua construção e o torna menos sujeito a bugs.

Embora os programadores possam utilizar comentários para documentar o código fonte, os mesmos são evitados. De acordo com [Fowler 2000], quando você sentir a necessidade de escrever um comentário, primeiro tente refatorar o código de modo que qualquer comentário se torne supérfluo. Já [Astels 2003] diz que a maioria dos comentários é desnecessária se o código for escrito de modo que a intenção esteja clara. Se e quando escrevermos comentários, devemos assegurar que eles comuniquem o porquê e não o como. Esses princípios são interessantes, mas difíceis de ser utilizados uma vez que às vezes um código é claro para um programador e ao mesmo tempo é extremamente complexo para outro. Portanto, o cuidado deverá ser grande para saber se é claro para todos realmente.

Apesar dos aparentes benefícios da refatoração, um problema que pode surgir é a velocidade do desenvolvimento. Em princípio, o esforço de refatoração parece representar um re-trabalho e um custo adicional para o projeto, na medida em que seria preferível projetar o design correto desde o início. Como os projetos de software normalmente sofrem com prazos curtos, pode ser que simplesmente não haja tempo disponível para refatorar.

Por maiores que sejam os benefícios aparentes da refatoração, toda mudança no código traz consigo o potencial de que algo deixe de funcionar. Por essa razão, a adoção da prática de refatoração só pode ocorrer em projetos que produzam testes automatizados. De acordo com [Poppendieck 2003], uma ferramenta chave que anda de mãos dadas com a refatoração e de fato a torna viável são os testes automatizados.

1.6.10 Integração Contínua

Equipes XP normalmente são compostas por diversos programadores, trabalhando em pares de acordo com a prática de código coletivo. Isso cria dois problemas práticos. O primeiro é de acordo com [Poppendieck 2003] sempre que diversos indivíduos estão trabalhando na mesma coisa, ocorre uma necessidade de sincronização. O segundo é que os pares precisam ser capazes de evoluir rapidamente sem interferir no trabalho uns dos outros.

Esta questão é resolvida no XP utilizando-se uma prática conhecida como integração contínua. Os pares trabalham de forma isolada, porém integram o que produzem com a versão mais recente do código de produção, diversas vezes ao dia. Isto é, os pares se sincronizam com freqüência à medida que terminam pequenas atividades de codificação [Beck 2000]. Toda vez que um par integra seu código, há um risco de que identifique um erro na integração. Isto é, existe a chance, por exemplo, de que dois pares distintos tenham efetuado alterações conflitantes em uma mesma linha do código.

De um modo geral, os pares procuram descobrir estes eventuais conflitos tão cedo quanto possível, pois de acordo com [Jeffries 2001], quanto mais esperamos, pior as coisas vão ficando. Um bug introduzido ontem é bem fácil de encontrar, enquanto dez ou cem introduzidos semanas atrás podem se tornar quase impossíveis de serem localizadas. Integrando rapidamente, os pares também asseguram que o lote de trabalho a ser integrado será pequeno. Afinal, não se consegue produzir muita coisa em um espaço de tempo curto, como por exemplo, de uma ou duas horas. Desta forma, se houver um erro na integração, o mesmo será referente a um lote pequeno de trabalho, onde menos coisas podem falhar.

1.6.11 Entregas Curtas

Projetos de software sempre são considerados um investimento para o cliente. O cliente investe recursos na expectativa de obter um retorno dentro de certo prazo. O volume de retorno depende do valor de negócio produzido e do tempo em que o mesmo é entregue. O XP procura maximizar o retorno de investimento dos projetos assegurando que as funcionalidades com maior valor de negócio sejam entregues ao final de cada versão e que cada versão tenha uma duração curta. Isto é feito através do processo contínuo de priorização que seleciona sempre as estórias de maior valor para serem implementadas primeiro. Neste sentido, XP propõe a prática de entregas curtas,

que consiste em colocar o sistema em produção com freqüência, em prazos curtos, normalmente de dois ou três meses. Entrega rápida normalmente se traduz em aumento de flexibilidade no negócio.

Trabalhando com entregas curtas e priorização permanente, a equipe procura assegurar que os primeiros *releases* gerem a maior parte do valor do sistema (por incluírem, em princípio, as funcionalidades com maior valor para o negócio). Portanto, espera-se que o maior retorno esteja concentrado mais próximo do início do projeto, quando o cliente normalmente ainda não efetuou a maior parte dos desembolsos.

Colocando versões em produção rapidamente, o cliente tem a oportunidade de receber feedback concreto sobre o real potencial de retorno do projeto. Portanto, tem a chance de decidir cedo, quando ainda não gastou muito, se continua ou não com o projeto e particularmente se continua ou não investindo na mesma equipe de desenvolvimento. Portanto, a adoção de releases curtos funciona como uma estratégia de gestão de risco do projeto [Teles 2004]

Entre outras vantagens, de acordo com [Brooks 1995] entregas curtas e freqüentes provêm benefício cedo para o cliente enquanto fornecem feedback rápido para os programadores. Eles têm a chance de aprender o que a comunidade de usuários, como um todo, pensa a respeito do sistema. Ao longo do desenvolvimento de um projeto XP, é comum os programadores interagirem diariamente com um representante dos usuários.

1.6.12 Metáfora

Uma equipe de desenvolvimento formada por diversos programadores convive com o desafio de manter a integridade conceitual do sistema mesmo havendo diversos projetistas criando estruturas novas na arquitetura ao longo do tempo. Isso pode ser resolvido caso exista um mecanismo capaz de alinhar o pensamento dos diversos projetistas assegurando que todos compartilhem uma visão única de como adicionar e manter as funcionalidades do sistema. O XP utiliza o conceito de metáfora para atingir este objetivo.

Isso nada mais é que o sistema ter uma forma única de lidar com a mesma situação dentro de um software. Por exemplo, as telas de cadastro devem ser projetadas exatamente da mesma forma para que o usuário não pense que tem

sistemas diferentes dentro do mesmo software. O que ocasiona isso é quando as pessoas fazem as partes dos sistemas de forma isolada sem uma interação entre os membros de forma adequada.

De acordo com [Brooks 1995], um exemplo recorrente de uso eficaz de uma metáfora são as interfaces gráficas baseadas em janelas. Elas representam um exemplo sublime de uma interface de uso que possui integridade conceitual, conquistada pela adoção de um modelo mental familiar, a metáfora do desktop.

1.6.13 Ritmo Sustentável

Segundo [Brooks 1995], mais projetos de software saem de curso por falta de tempo do que por qualquer das outras causas combinadas. Quando isso acontece, os projetos normalmente adotam duas estratégias: a utilização de recursos no limite máximo e a adoção de horas-extras de trabalho.

O XP, por sua vez, evita as duas abordagens utilizando a prática de ritmo sustentável. Em síntese, ela recomenda que os membros da equipe de desenvolvimento trabalhem apenas durante o tempo regulamentar, ou seja, oito horas por dia, e evitem horas-extras tanto quanto possível. Além disso, sugere-se que os prazos sejam mantidos fixos e as cargas de trabalho sejam ajustadas (através de priorização) para se adequar aos prazos [Beck 2001]. As organizações precisam incorporar algum nível de folga em suas estruturas para que possam operar de maneira adequada.

Para muitos gerentes de projeto, a adoção de horas-extras é uma solução intuitiva para elevar a produtividade da equipe de desenvolvimento. Entretanto, segundo [Demarco 2001] a hora extra por um longo período é uma técnica de redução de produtividade. Reduz o efeito de cada hora trabalhada.

Como sugerido pela Figura 2.2, a produtividade líquida pode eventualmente aumentar durante as primeiras 20 horas de hora-extra. Mas cedo ou tarde, todo mundo alcança um ponto em que os resultados diminuem. Em algum ponto, a produtividade começa a diminuir devido a erros crescentes e falta de concentração. De fato, existe um ponto em que o membro da equipe se torna um "produtor negativo líquido," porque o esforço de re-trabalho causado por erros e defeitos excede a contribuição positiva de novo software desenvolvido [Yourdon 2004].

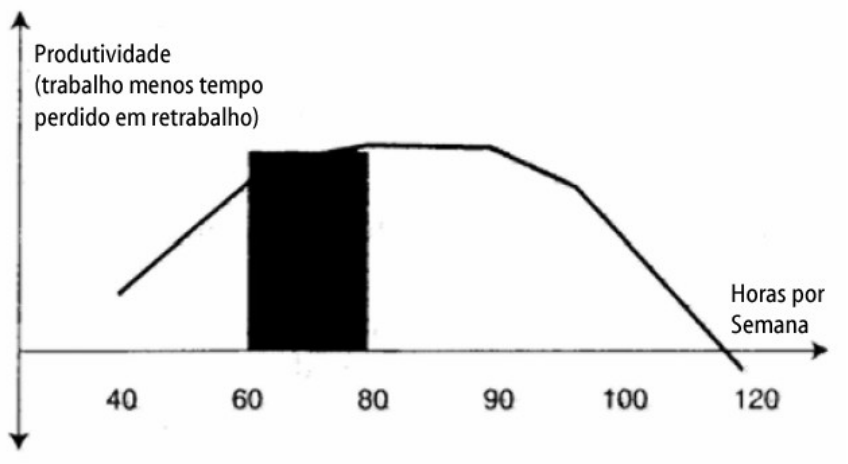


Figura 2. 2 Produtividade líquida versus horas trabalhadas por semana

Por estas razões, o XP procura assegurar que a equipe trabalhe apenas durante as horas regulamentares. As pressões de tempo são tratadas através do processo de priorização e o ajuste do escopo de cada iteração para a real capacidade de entrega da equipe. Como essa capacidade pode variar ao longo do tempo, ela é permanentemente monitorada e ajustada à medida que o projeto avança.

1.7 Scrum

Nesta seção será apresentada a metodologia ágil Scrum. De forma semelhante ao método XP, Scrum tem sido largamente utilizado em projetos de desenvolvimento de sistemas. Diversos estudos têm sido realizados descrevendo a utilização de Scrum em projetos de vários tipos de sistemas, tais como, jogos eletrônicos, aplicações para Web, sistemas de informação, entre outros.

1.7.1 Visão Geral

Scrum foi proposto por Ken Schwaber em 1996 como um método ágil que prevê as mudanças freqüentes durante o desenvolvimento de software. O termo Scrum tem origem no esporte conhecido como Rugby [Rising 2007]. Neste esporte, "Scrum" ocorre quando jogadores de cada time colaboram entre si numa tentativa de avançar juntos pelo campo adversário [Highsmith 2002].

Scrum se destaca das demais metodologias ágeis por dar mais enfoque à área de gerenciamento de projetos. Ele tem atingindo grande sucesso com desenvolvedores e gerentes de projetos. Scrum é um método que descreve com conjunto de práticas, papéis bem definidos e totalmente adaptáveis, seu ciclo de vida se resume em iterações e entregas incrementais. O método permite um melhor acompanhamento do

que está acontecendo durante o projeto, facilitando o ajuste durante o projeto e possibilitando alcançar os objetivos do projeto de forma ágil.

1.7.2 Papéis do Scrum

De acordo com [Schwaber 2004], o Scrum é dividido em 3 papéis: Product Owner (cliente), o time e o ScrumMaster (gerente de projeto). E todas as responsabilidades são divididas entre esses três papeis, são elas:

Product Owner(PO) - é responsável por representar os interesses de todos os usuários do sistema final. Ele é responsável principalmente por saber o retorno do investimento (ROI) de cada requisito do sistema. Ele é o grande responsável e um dos grandes envolvidos pela priorização dos requisitos. De acordo com [Marçal 2007], o PO prioriza o Product Backlog (lista de requisitos) a cada Sprint (iteração), garantindo que as funcionalidades de maior valor sejam construídas prioritariamente.

Time – É a equipe responsável por desenvolver as funcionalidades. Os times para o Scrum são auto-gerenciados, auto-organizados e multifuncionais. Eles são responsáveis por descobrir como tornar os itens de backlog (Product Backlog) em incrementos de funcionalidades dentro de iterações e gerenciando seu próprio trabalho a fazer. Os membros do time são coletivamente responsáveis pelo sucesso de cada iteração e do projeto como um todo.

ScrumMaster – é responsável pelo processo do Scrum, para treinar no Scrum todos os envolvidos no projeto para implementar Scrum, tudo isso alinhado na cultura da organização e ainda entregando os benefícios esperados para garantir que todos irão seguir as regras e práticas descritas no Scrum. O papel do *Scrum Master* seria equivalente ao gerente de projetos.

Os papéis mostram de forma clara a diferença entre XP e SCRUM. O XP tem o foco na engenharia de software enquanto que o SCRUM enfoca na gestão da equipe. A seguir é apresentado o fluxo de um projeto Scrum.

1.7.3 Fluxo do Scrum

Um projeto Scrum inicia com a visão global do sistema que será desenvolvido. Esta visão deve ser bem alto nível, mas isto vai ficando mais claro à medida que o projeto avança. O dono do produto é responsável por produzir esse documento de

forma que seja maximizado seu retorno do investimento (ROI). O *Product* O*wner* elabora o plano para fazer o que será incluso no *Product Backlog* ou Carteira do produto. O *Product Backlog* é a lista de todos os requisitos funcionais e não funcionais. A carteira do produto é priorizada pelos itens mais desejados, os que geram mais valor são os mais prioritários e divididos nas entregas propostas. O *Product Backlog* priorizado é o ponto de partida e o conteúdo, prioridade e os agrupamentos dos itens do *baklog* nos releases normalmente mudam esperado. Mudanças nos itens de *backlog* refletem mudanças nos requisitos de negocio que rapidamente ou lentamente o time poderá transformar numa funcionalidade.

Todo o trabalho é feito em Sprints ou iterações. Cada Sprint é uma iteração de 30 dias consecutivos de acordo com [Schwaber 2004], mas vemos na prática uma flexibilidade grande em relação a esse tamanho uma vez que isso depende muito das características do time e da forma como o esforço para implementar o produto. Cada *Sprint* é iniciado com o *Sprint planning meeting*, onde o *Product Owner* e o time estão juntos para colaborar sobre o que será feito na próxima *Sprint*. São selecionadas as estórias de usuários de maior prioridade, o *Product Owner* fala ao time o que é desejado e o time fala ao *Product Owner* como o que é desejado e esperado para implementar uma funcionalidade durante a próxima *sprint*. A *Sprint planning* não pode ser mais longa que 8 horas. Para as metodologias ágeis de uma forma geral esse controle do tempo é muito importante para a disciplina do time, auto-gerenciamento e principalmente ao foco em todos os momentos.

A *Sprint planning* é dividida em duas partes. A primeira parte é de 4 horas com o PO apresentando os itens de maior prioridade ao time. O time entende cada um dos itens e questiona o conteúdo, a proposta, o significado, as intenções de cada item do *Product Backlog*. Isso no RUP faz parte da atividade de apenas uma pessoa normalmente no levantamento de requisitos. Nessas primeiras 4 horas é o momento que é selecionado o que irá entrar na *Sprint* logo após a definição do time de *Story Points* de cada item de *backlog*. Já na segunda parte é quando o time detalha as tarefas que duram até 8 horas. Todos os dias o time se reúne no *Daily Scrum* ou Reunião Diária que dura no máximo 15 minutos. Nessas reuniões são respondidas por cada integrante do time as seguintes perguntas:

- O que eu fiz para esse projeto desde a última reunião?
- O que eu irei fazer até a próxima reunião?
- Quais os impedimentos que eu tenho para conseguir finalizar meus objetivos da Sprint e das minhas tarefas?

O Scrum aplica conceitos trazidos do XP, normalmente essa reunião é realizada no formato *Stand up meeting* o que facilita ao cumprimento do tempo por todos estarem em pé e um valor que é abordado nesse momento é o da comunicação. Essa prática de falar os problemas e o que ocorreu acaba resultando em um momento rico de troca de conhecimento da equipe e de comunicação resultando uma importante a sincronização do trabalho do time.

No final da *Sprint*, o *sprint review* é realizado. De acordo com [Schwaber 2004] é planejado para durar 4 horas. Nele é feito uma apresentação ao PO de tudo que foi desenvolvido durante aquela *sprint*, Após essa prática é feita a *Sprint Retrospective* onde são feitas as seguintes perguntas pra o time:

- O que aconteceu durante a sprint?
- O que foi bom durante a sprint?
- O que poderia funcionar melhor?

Essa prática é de extrema importância para a equipe se conhecer melhor e melhorar nas próximas *Sprints*. Este processo é necessário para fazer o time ficar mais eficiente na próxima *sprint* com a revisão dos frameworks e dos processos. Juntos a reunião de *Sprint planning*, o *Sprint review* e o *Sprint retrospective* constituem uma inspeção formal e uma adaptação das práticas de Scrum.

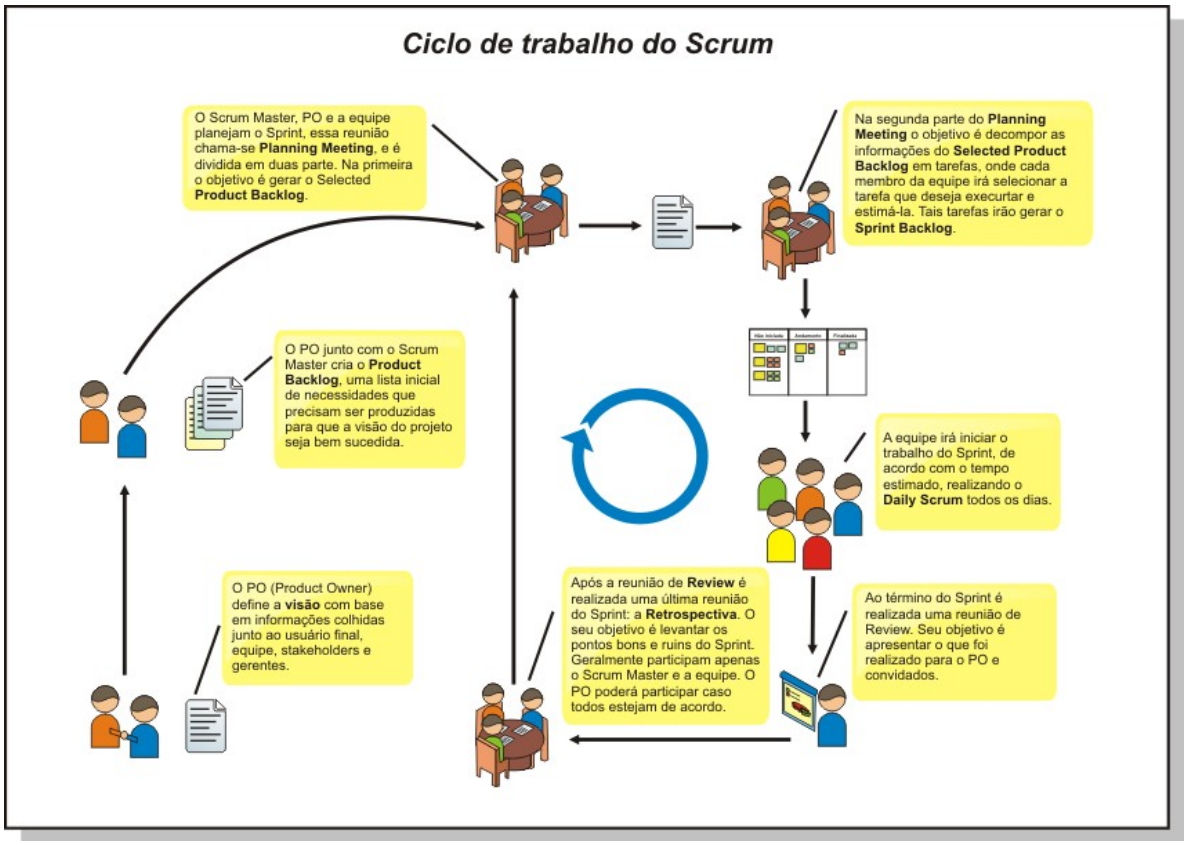


Figura 2. 3 Fases e papeis do Scrum

Na figura 2.3 é possível verificar todas as fases e papeis do Scrum. Ao lado esquerdo em baixo é quando o *PO* define a visão do que será produzido. Acima o Po se une com o *Scrum Master* e cria o *Product Backlog*. Na continuação do ciclo ao topo da figura o *Scrum Master* e a equipe realizam o *Sprint Planning* 1 com o objetivo de selecionar o *backlog*. Ao lado direto é realizada a segunda cerimônia de planejamento o *Sprint Planning* 2 onde os itens escolhidos no *Sprint Planning* 1 são quebrados em tarefas diárias. Abaixo é mostrado o quadro de tarefas que é montado a partir dessa reunião e onde é acompanhada toda a *Sprint* junto com as reuniões diárias. Abaixo é mostrada a reunião de *review* onde são mostrados para o PO os produtos desenvolvidos. Ao lado esquerdo antes de re-iniciar o ciclo é feito finalmente a Retrospectiva onde é feito o processo de melhoria contínuo.

Considerações finais

As metodologias ágeis geraram muita polêmica na academia com o seu surgimento. Como toda abordagem diferente, as metodologias ágeis trouxeram uma quebra de paradigma que causou essa reação negativa inicial. Com o passar dos anos diversos artigos e estudos foram realizados comparando o modelo tradicional com as

metodologias ágeis. Em [Charete 2001] é possível ver essa comparação bem clara e mostra alguns resultados interessantes mostrando que projetos utilizando métodos ágeis obtiveram melhores resultados em termos de cumprimento de prazos, de custos e de padrões de qualidade. Além disso, mostra que o tamanho dos projetos das equipes que utilizam metodologias ágeis têm crescido. Apesar de ser normalmente sugerido que as equipes dos projetos ágeis sejam pequenas e médias (até 12 desenvolvedores), aproximadamente 15% dos projetos que usam metodologias ágeis estão sendo desenvolvidos por equipes de 21 a 50 pessoas e 10% dos projetos são desenvolvidos por equipes com mais de 50 pessoas, considerando um universo de 200 empresas usado no estudo.

As metodologias ágeis não são tão antagônicas em relação às metodologias tradicionais como o RUP. Existiu algum radicalismo no inicio no manifesto ágil[Agile Manifesto], mas com o passar do tempo tanto as pessoas que defendem as metodologias ágeis se tornaram menos radicais como os tradicionalistas passaram a admitir que em alguns projetos as metodologias ágeis se tornam mais adequadas. Alguns autores ainda dizem que metodologia ágil é voltar aos tempos caóticos do desenvolvimento como [Pressman 2001]. Mas a verdade é que os valores devem ser aplicados em qualquer metodologia e que dependendo na natureza do projeto metodologias ágeis podem ser a melhor escolha. Projetos que tem um escopo mais aberto ou que o cliente não sabe exatamente o que quer no início do projeto ou que tem expectativa de mudar muito os requisitos durante o projeto são exemplos de projetos onde metodologias ágeis podem ser bastante adequadas.

Engenharia de Requisitos

1.9 Conceituação

Estudos recentes comprovam que os problemas relacionados aos requisitos do sistema afetam inúmeras organizações que desenvolvem e usam sistemas de software[Sommerville 2007] [Nuseibeh 1997]. Portanto, pode-se considerar que a engenharia de requisitos é uma das fases mais importantes do processo de engenharia de software [Lamsweerde 2000].

A Engenharia de Requisitos – (ER) envolve o como o processo de aquisição, entendimento e refinamento das necessidades do cliente para um projeto de software, cujo objetivo é o de uma correta especificação dos requisitos de software, [IEEE STD830 1984]. Segundo Thayer e Dorfman[Thayer 1997], ER é a primeira das várias etapas do processo de Engenharia de Software. A principal preocupação da Engenharia de Requisitos envolve o entendimento de quais são os reais requisitos do sistema, bem como a documentação, análise, validação e gerenciamento dos requisitos. De acordo com [Sommerville 2007], é um processo que envolve todas as atividades exigidas para criar e manter o documento de requisitos de sistema.

Existem inúmeras definições disponíveis na literatura para o termo requisitos. Segundo Kotonya e Sommerville [Kotonya 1997], um requisito pode descrever:

- Uma facilidade no nível do usuário; por exemplo, um corretor de gramática e ortografia;
- Uma restrição específica no sistema; por exemplo, o tempo de resposta de uma operação;
- Uma propriedade muito geral do sistema; por exemplo, o sigilo de informações sem a devida autorização;
- Uma restrição no desenvolvimento do sistema; por exemplo: a linguagem que deverá ser utilizada para o desenvolvimento do sistema.

Os requisitos podem ser classificados em funcionais, não-funcionais e organizacionais, [Kotonya 1997] e [Loucopoulos 1995], a saber:

- Funcionais - descrevem as funções ou características que o sistema pretende satisfazer, as reações, as entradas e o comportamento;
- Não-funcionais - descrevem aspectos e restrições relacionados às qualidades contempladas pelo sistema, tais como: confiabilidade, desempenho, portabilidade, segurança e usabilidade;
- Organizacionais – descrevem a estrutura da organização, como as metas, filosofia e políticas adotadas, seus funcionários e suas expectativas.

De acordo com[Brooks 1987], a parte mais crítica do desenvolvimento de software é definir precisamente os requisitos do projeto, configurando-se esta como a etapa mais passível de erros no sistema, não sendo superada por nenhuma outra, como também nenhuma outra é tão difícil de ser posteriormente consertada.

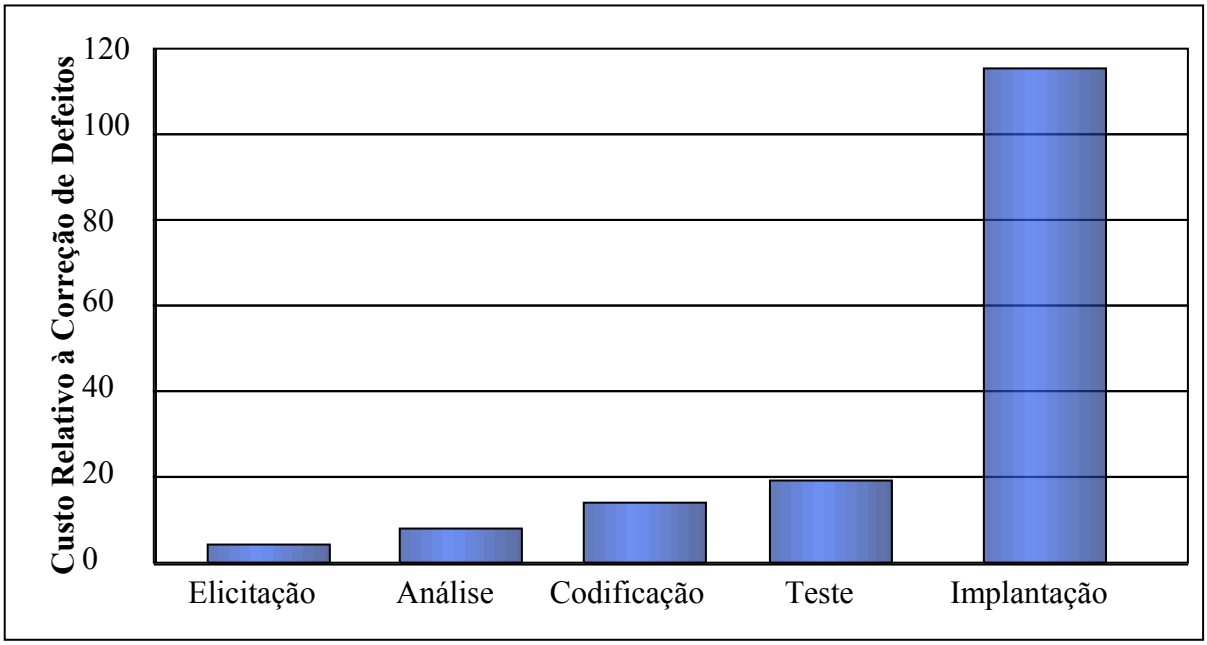


Figura 3.1 Custo Relativo à Correção de Defeitos X Fases do Desenvolvimento – Traduzido de [Grady 1999].

De acordo com a Figura 3.1 quanto mais o tempo passa mais caro fica a correção de um defeito. Portanto é muito importante que os defeitos sejam encontrados nas fases iniciais evitando assim os altos custos de alteração nas fases posteriores.

1.10 Fases da Engenharia de Requisitos

Em [Thayer 1997], as fases da Engenharia de Requisitos são divididas em:

- Elicitação: atividade que envolve a coleta e compreensão dos requisitos junto aos stakeholders.
- Análise: constitui-se do processo em que se analisam as necessidades dos clientes e usuários para se otimizar a definição dos requisitos de software.
- Documentação: é o processo de elaboração documental e diagramas nos quais estão definidos os requisitos.
- Validação: etapa em que se procura assegurar que a especificação de requisitos de software esteja em conformidade com os requisitos elicitados e analisados nas fases anteriores, ou seja, estão de acordo com a solicitação do stakeholder.
- Gerenciamento: planejamento e controle dos requisitos durante toda a sua existência no projeto.

  A Figura 3.2 apresenta uma visão geral das fases do processo de engenharia de requisitos de acordo com o modelo espiral.

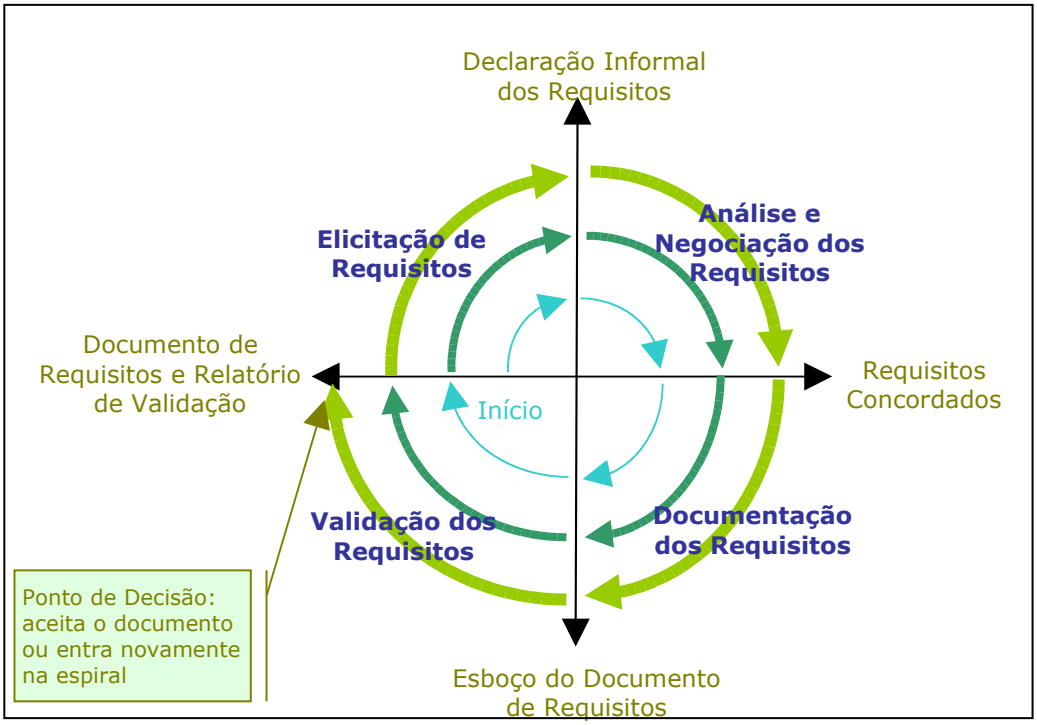


Figura 3.2 Atividades do Processo de Engenharia de Requisitos no modelo espiral[Kotonya 1997].

É possível ver na Figura 3.2 todo o processo espiral com o modelo iterativo incremental. É possível verificar que cada uma das fases é realizada várias vezes até o momento que o documento é considerado aceito. O requisito pode ser priorizado a qualquer momento desde que o mesmo esteja pronto. De acordo com o [Rup 2002], os requisitos irão sendo detalhados com o decorrer do projeto de acordo com a priorização que é efetuada. Uma das críticas ao RUP é a prática de documentar os requisitos minuciosamente no inicio do projeto e de negar as mudanças de requisitos. A mudança dos requisitos faz parte de todos os projetos e isso deve ser tratado com naturalidade e principalmente com um gerenciamento adequado dessas mudanças. Neste contexto, as metodologias ágeis são mais adequadas para tratar tais problemas. Enquanto quando se tem escopo bem definido e fechado às metodologias tradicionais são mais indicadas. Nas seções a seguir são apresentadas cada uma das fases do processo de ER.

1.10.1 Elicitação de Requisitos

Caracteriza-se como a etapa inicial do Processo de Engenharia de Requisitos, cuja finalidade é esclarecer o conhecimento relevante acerca do problema. Em geral, pressupõe uma interação direta com os stakeholders para que se processe a realização

da coleta dos requisitos. Além de identificar as necessidades dos stakeholders, esta atividade requer uma cuidadosa análise da organização, do domínio da aplicação e dos processos organizacionais [Kotonya 1997]. Complementarmente, [Castro 2002] sugere que existe a necessidade de capturar as intenções, desejos e motivações dos stakeholders. Ao final do processo de elicitação de requisitos, o volume de informações resultante costuma ser alto. Essas informações precisam ser criteriosamente organizadas, de forma que permitam sua compreensão e utilização nas demais etapas da Engenharia de Requisitos.

De acordo com [Goguen, 1993], muitas das abordagens utilizadas na elicitação não são oriundas da Ciência da Computação, mas sim das Ciências Sociais como a Sociologia e a Psicologia. Analistas precisam entender e contextualizar tais abordagens para o seu contexto. Dentre as principais técnicas para elicitação de requisitos destacam-se: entrevista, questionário, observação, reuniões em grupo, análise de documentação, etc. Segundo [Patricio 2004], importantes contribuições para a elicitação de requisitos são propostas pela área de marketing, uma vez que ela lida com aspectos de avaliação e aceitação de produtos ou serviços considerando a perspectiva dos consumidores ou usuários.

1.10.2 Análise e Negociação de Requisitos

Durante esta fase são as informações coletadas sobre os requisitos para que estas possam ser checadas, discutidas, melhor compreendidas, priorizadas e negociadas. Esta fase também envolve a discussão dos conflitos existentes entre os requisitos e a busca por uma solução consensual para gerenciar possíveis conflito e atender aos anseios das partes envolvidas. De acordo com [Kotonya 1997], os requisitos finais devem refletir um compromisso entre as necessidades estratégicas da organização, dos requisitos específicos de diferentes partes interessadas, das restrições de projeto, além de limitações de prazo e orçamento. Para tanto, os modelos de negociação geralmente identificam as principais necessidades dos stakeholders, priorizando-as a fim de assegurar que os requisitos mais críticos a elas relacionados sejam atendidos o quanto antes. Segundo [Kotonya 1997], constitui-se como principais tarefas da fase de análise de requisitos:

1. **Verificação da importância dos requisitos -** analisar a real necessidade dos requisitos elicitados, se estes estão alinhados às metas e

objetivos de negócio da organização, bem como, para o problema a ser abordado, esta atividade está diretamente ligada ao foco desse trabalho que será tratado no capítulo seguinte sobre Kano;

2. **Verificação da consistência e completude dos requisitos** – a averiguação da consistência provê a checagem dos requisitos para que estes não estejam contraditórios. Já a checagem da completude, compromete-se examinar se nenhum serviço ou limitação, considerado necessário, foi esquecido;
3. **Verificação da viabilidade dos requisitos** – o resultado da elicitação é uma extensa lista de requisitos. Sendo necessário checar se eles são viáveis em relação ao tempo, orçamento, recursos, entre outros aspectos que dizem respeito ao desenvolvimento do sistema de software.

Para atender as tarefas desta fase de forma eficaz são utilizadas as técnicas apresentas a seguir:

Listas de verificação (checklists) – correspondem às listagens de questões, disponibilizadas ao analista, para auxiliá-lo na avaliação de cada requisito. Checklists são úteis, porque provêem uma referência sobre o que procurar e reduzem as chances de que requisitos importantes deixem de ser checados. Ao final da checagem, pode-se fornecer uma lista de inconsistências encontradas, as quais podem ser solucionadas por meio de negociação ou nova elicitação do requisito;

Matrizes de interação – utiliza-se para evidenciar a interação entre requisitos e facilitar o processo de análise de possíveis conflitos entre estes. A forma mais simples de construção dessas matrizes é o uso de uma tabela e a rotulação de suas linhas e colunas com identificadores de requisitos. Valores numéricos indicam a relação entre cada um dos requisitos mapeados, evidenciando conflitos ou sobreposições. Requisitos com altos valores correlacionados devem ser cuidadosamente analisados para avaliar o impacto que essas relações, e eventuais mudanças nas mesmas, possam ter no desenvolvimento do sistema;

Prototipação – os protótipos criados na etapa de elicitação, podem ser aperfeiçoados na etapa de análise e negociação, possibilitando uma avaliação mais rica dos requisitos do sistema. Além disso, os protótipos contribuem para um maior

envolvimento entre as partes interessadas durante as atividades de elicitação, análise e negociação de requisitos;

Reuniões – é, provavelmente, o meio mais eficiente de negociação e resolução de conflitos entre requisitos. As reuniões de negociação são conduzidas em três etapas:

- Explicação sobre a natureza dos problemas de requisitos;
- Discussão entre as partes acerca do melhor modo de resolução dos problemas e observando as prioridades estabelecidas;
- Resolução dos conflitos mediante tomada de ações corretivas.

1.10.3 Priorização de requisitos

A priorização de requisitos existe para tentar resolver os conflitos existentes entre as necessidades dos usuários sem, todavia impactar na satisfação dos objetivos de cada usuário. Em geral, os modelos de priorização identificam as principais necessidades de cada usuário, ou seja, atribuem prioridades aos requisitos em seguida analisa os resultados para tentar que os requisitos mais críticos sejam atendidos. Dentre as principais vantagens de priorizar os requisitos incluem:

- Evita escolhas entre alguns objetivos conflitantes como qualidade, custo, e time-to-market.
- Priorização dos recursos baseado na importância para o objetivo do projeto.

De acordo com [Karlsson 1997], normalmente a priorização é efetuada por meios informais e pouco efetivos. Outra dificuldade enfrentada é a forma como a priorização é realizada, pois muitas vezes não fica claro para o cliente qual é o impacto real para o seu negócio que a prioridade de determinados requisitos pode trazer. Na prática, muitas empresas de software realizam esse processo de seleção informalmente o que gera softwares que não tem qualidade. Sem as técnicas para fazer essas escolhas importantes, a saída é sempre uma difícil surpresa. Mesmo com todo desenvolvimento da engenharia de requisitos ainda não existe uma forma simples, efetiva e industrialmente testada para priorizar requisitos. Nesse trabalho é proposta uma técnica para fazer essa priorização de forma clara simples e com indicadores bem precisos.

Em [Karlsson 1998], é apresentado um estudo de caso realizado num projeto Ericsson em que duas importantes técnicas para priorização e negociação de requisitos foram comparadas. As técnicas avaliadas foram AHP [Saaty 1990] e QFD [Hauser 1988], os resultados obtidos indicam que a técnica AHP apesar de ter sido considerada complexa, apresentou melhores resultados.

A técnica AHP((Analytical Hierarchy Process) [Saaty 1990] é um método baseado na decomposição hierárquica de um problema de tomada de decisão com múltiplos critérios. Essa técnica tem sido usada com sucesso em diversas áreas, incluindo engenharia de software [Finnie 1995] e seleção de software [Hong 1981] [Ncube 2000] [Min 1992] [Kontio 1996] [Kunda 1999].

O método AHP é suportado por uma ferramenta comercial chamada Expert Choice [Exp 2009] que auxilia nos cálculos e julgamentos necessários. AHP decompõe um problema numa hierarquia de critérios, onde em cada nível a importância relativa de cada fator é avaliada através de comparações aos pares dos critérios. Finalmente, as alternativas são comparadas aos pares em relação a cada critério. Os principais passos para aplicar a técnica AHP são [Saaty 1990]:

i. Definir uma hierarquia de fatores que influenciam a decisão, isso resulta numa estrutura hierárquica de fatores que tem as alternativas como nós finais dessa hierarquia;

ii. Definir a importância dos fatores em cada nível;

iii. Definir as preferências entre as alternativas;

iv. Verificar a consistência do ranking e revisar os resultados;

Apresentar o resultado da avaliação, onde a alternativa com maior prioridade será recomendada como a melhor escolha.

O ranking obtido através da comparação aos pares entre as alternativas é convertido para um ranking normalizado usando o método de autovetores, ou seja, o ranking relativo entre as alternativas é representado em valores cujo total é um. Segundo Saaty [Saaty 1990], o tratamento de um problema com múltiplos atributos através de hierarquias é a forma natural como os seres humanos observam e representam o mundo. Usando essa técnica, as alternativas e critérios são comparados aos pares, o que resulta em resultados mais confiáveis [Kontio 1996]. Dessa forma, o problema de ter que comparar valores absolutos é evitado uma vez que somente sua preferência relativa é comparada.

Um dos principais problemas da técnica AHP é a complexidade dos cálculos necessários para fazer as comparações aos pares e realizar as operações entre as matrizes de valores. Entretanto, com o uso da ferramenta Expert Choice [Exp 2009] esse problema pode ser minimizado. Uma importante limitação de AHP quando aplicada na avaliação de software é o fato dessa abordagem assumir independência entre os atributos dos produtos, ou seja, com o objetivo de fazer as comparações, essa técnica assume que os atributos são independentes [Ncube 2000].

De acordo com [Kotonya 1997], a fase de análise deveria ser bastante simples uma vez que deveria apenas estar preocupada com as necessidades técnicas e organizacionais. Porém, na realidade, são fortemente influenciadas pelas necessidades pessoais dos *stakeholders* que ao observarem várias possibilidades mais detalhadamente, podem mudar suas opiniões, ocasionando a mudança dos requisitos, resultando normalmente em conflitos.

Na verdade, é muito provável que os requisitos mudem, com certa freqüência, ao longo do ciclo de vida do sistema, pois os diversos *stakeholders* possuem visões e objetivos conflitantes. Dessa forma, o analista deve atentar-se à contínua priorização dos requisitos.

1.10.4 Documentação dos Requisitos

Nesta etapa documenta-se num nível apropriado de detalhamento o escopo dos requisitos, servindo como base para o restante do processo de desenvolvimento do software. Segundo [Ryan, 1993] o documento de requisito é o um modo possível de descrição das características do produto e método de desenvolvimento, interface com outras aplicações, descrição sobre o domínio e informações de suporte ao problema.

A documentação dos requisitos do sistema pode ser realizada tanto através de linguagem natural, [Ambriola 1997], quanto da lógica, [Castro 1996]. [Pereira 2007] considera, ainda, que na escolha da melhor forma de documentação dos requisitos, deve-se prezar fundamentalmente a natureza do sistema de software a ser desenvolvido, bem como a freqüência de mudanças e/ou a volatilidade dos requisitos.

O documento de requisitos serve como um contrato entre os stakeholders e os analistas, devendo, portanto, deve ser formatado e estruturado de acordo com padrões organizacionais [Kotonya 1997].

[Toranzo 1999] ressalta que para uma documentação ser considerada completa e consistente, é necessário o rastreamento dos requisitos, que pode ser horizontal ou vertical. O rastreamento é horizontal, quando fizer referências cruzadas entre requisitos; e vertical, quando centrado no relacionamento entre os artefatos de diferentes fases do processo de desenvolvimento.

Observa-se que esta atividade não é trivial, de forma que são necessárias ferramentas específicas de suporte ao gerenciamento de requisitos para acompanhar adequadamente as alterações e evolução dos requisitos. Uma vez produzida uma documentação adequada, esta permite a organização e monitoramento de requisitos, enquanto auxilia no processo de controle de mudanças dos mesmos quando estas forem necessárias.

1.10.5 Validação de Requisitos

A validação de requisitos é definida como o processo de certificação da documentação de requisitos, para garantir que ela esteja em conformidade com as necessidades e intenções dos stakeholders. Essa visão da atividade de validação retrata um processo contínuo, que ocorre, na maior parte do tempo, em paralelo com as fases de elicitação e especificação [Loucopoulos 1995].

Nesta etapa lida-se com a dificuldade de obtenção de consenso entre diferentes stakeholders que podem possuir objetivos conflitantes, [Lawsweerde 1998]. A validação pode ser dificultada quando não existe o comprometimento do stakeholder para realizar a discussão junto ao analista de requisitos [Karlsson 2003]. Este problema é agravando quando o produto está sendo desenvolvido pela primeira vez ou quando não há um método para demonstrar que uma especificação de requisitos esteja correta em relação a outras representações do sistema [Kotonya 1997].

Contudo, deve-se destacar que existe uma variedade de técnicas que podem ser aplicadas para apoiar o processo de validação, são elas:

- **Revisões –** provavelmente, esta é a técnica mais utilizada. Revisões envolvem um grupo de pessoas lendo e analisando a documentação de requisitos à procura de possíveis problemas. A revisão de requisitos se processa da seguinte forma: uma reunião formal, na qual um analista de requisitos apresenta cada um dos requisitos para crítica e identificação de

inconsistências pelo grupo; uma vez identificadas, são registradas para que, posteriormente, sejam colocadas em discussão. O grupo de revisão deve, então, tomar decisões cuja finalidade é a de corrigir os problemas identificados.

- **Prototipação** – se um protótipo foi desenvolvido para fins de elicitação de requisitos, é possível usá-lo posteriormente para validação dos requisitos. Porém, os protótipos para validação devem ser mais completos e conter requisitos suficientes para garantir que facilidades projetadas para o sistema, estejam de acordo com o uso prático esperado por seus stakeholders. Protótipos de elicitação normalmente apresentam funcionalidades ausentes que, muitas vezes, não contemplam mudanças acordadas durante o processo de análise dos requisitos. É, portanto, necessário dar continuidade ao desenvolvimento do protótipo durante a etapa de validação.

- **Testes de Requisitos** – técnicas baseadas em 'cenário', que permitem elicitar e analisar requisitos, enquanto viabilizam a criação de casos de teste para os cenários identificados, [Cysneiros 2002]. A execução dos requisitos implementados pode ser simulada para demonstrar que todos os requisitos do sistema tenham sido considerados e estejam de acordo com o esperado. Na presença de dificuldades, quando analisados casos de teste para um dado requisito, implica que existe algum tipo de problema com a definição do requisito pode existir, [Kotonya 1997].

Ao finalizar a etapa de validação dos requisitos, pode-se dizer que há um conhecimento detalhado do domínio do problema e dos requisitos relevantes do sistema de software a ser desenvolvido, efetuando-se a entrega do Documento de Requisitos do Sistema. Caso não haja a aceitação da entrega do documento um novo ciclo pode ser iniciado e qualquer atividade do Processo de Engenharia de Requisitos poder-se-á realizar mais de uma vez (ver Figura 3.2), dependendo das necessidades de cada projeto. Na prática a validação não é só aplicada ao modelo final de requisitos, mas também a todos os modelos intermediários gerados ao longo do processo de requisitos.

1.10.6 Gerenciamento dos Requisitos

Embora não se contemple o gerenciamento dos requisitos no modelo espiral de Kotonya ([Kotonya 1997]); esta é uma importante atividade no processo de ER e envolve todas as fases já descritas anteriormente. Essa atividade consiste em gerenciar as inevitáveis mudanças dos requisitos propostos que surgem, principalmente, quando são alteradas as prioridades do negócio, quando se identificam erros ou omissões nos requisitos, ou quando novos requisitos são definidos. Uma visão geral do gerenciamento de requisitos envolve os seguintes passos:

- **Identificação de requisitos** – A identificação de cada requisito é importante para que se obtenha a referência cruzada deste e para serem feitas as avaliações de rastreamento;
- **Gerenciamento de mudanças** – Trata-se de um conjunto de atividades que avalia o impacto e o custo das mudanças;
- **Políticas de rastreamento** – Políticas que definem as relações entre os requisitos, assim como entre os requisitos e o projeto;
- **Suporte de ferramenta CASE** – O gerenciamento dos requisitos envolve uma grande quantidade de informações, requerendo o auxílio de uma ferramenta para o bom desempenho desta atividade. Estas ferramentas compreendem desde sistemas especializados, planilhas de cálculo ou bancos de dados.

O gerenciamento dos requisitos envolve também o entendimento e controle das mudanças nos requisitos do sistema, [Sommerville 2007]. Ela não se constitui em uma atividade trivial, utilizando-se de ferramentas específicas para lidar com as informações e alterações de requisitos durante sua evolução, [Sommerville 2007]. Deve-se ressaltar que para execução eficiente do gerenciamento de requisitos é necessário implementar a rastreabilidade.

Considerações finais

A Engenharia de Requisitos é composta de vários processos e de vários passos que dependem muito de todas as pessoas envolvidas no processo. Neste capítulo foram apresentadas as principais fases do processo de requisitos. Em geral, o

processo de priorização de requisitos é considerado crítico para sistemas que precisem ter um curto *time-to-market* e em contextos bastante competitivos.

Como foi discutido no Capítulo 2, os requisitos ágeis são compostos de estórias de usuários que embora tenha a mesma intenção dos requisitos são diferentes nas suas divisões e principalmente no grau de detalhe que ele é composto. No próximo capítulo será apresentada uma proposta para priorizar requisitos em projetos ágeis.

Proposta de Processo de Priorização de Requisitos para Projetos Ágeis

Neste capítulo é apresentada uma proposta de processo de priorização de requisitos para projetos de desenvolvimento de software ágeis. O processo proposto nesta dissertação foi baseado na técnica Kano [Cohn 2005]. As seções a seguir apresentam o Kano e a nossa proposta de adaptação da técnica.

Visão Geral de Kano

A priorização chamada de Kano foi desenvolvida por Noriaki Kano em 1984 [Cohn 2005]. É um modelo utilizado na área de gerência da qualidade e no uso do marketing para saber as necessidades do cliente e satisfazer-las de uma forma mais acentuada.

De acordo com [Info Escola 2009], Kano é uma técnica para o desenvolvimento ou melhoria de produtos baseado na caracterização das necessidades do cliente, sejam elas explicitamente verbalizadas ou não. O diagrama de Kano possibilita aos desenvolvedores de produtos e serviços transformarem as informações obtidas pelas pesquisas de marketing e centrais de atendimento ao cliente em melhorias reais no produto. O objetivo é enfocar não apenas a satisfação do cliente, mas a superação das suas expectativas, ou seja, o obter o encantamento do cliente.

De forma semelhante à maioria das abordagens adotadas pela engenharia de requisitos, técnica Kano não foi originada na área da computação, mas na administração. Ela está alinhada aos princípios das metodologias ágeis enfatizado no manifesto ágil “Colaboração com o cliente ao invés de negociação de contratos”. A forma como Kano utiliza a opinião dos clientes buscando atingir o máximo da particicipação deles. Foi esta característica de Kano que nos levou a adotar e adaptar essa técnica para projetos ágeis.

A pesquisa realizada por Kano mostra uma forma de priorizar as partes dos produtos em:

1. Indispensável ou mandatório;
2. Importantes ou linear;
3. Desejáveis.

De acordo com [Cohn 2005], essa divisão é voltada para priorizar as características que deixam o cliente mais satisfeito. Por exemplo: Um requisito indispensável de um hotel pode ser a limpeza, a cama, o banheiro, uma mesa. Uma vez satisfeita às necessidades do hóspede com o banheiro não importa se ele tem prateleira ou não. Com isso, os requisitos indispensáveis devem estar presentes, embora não estejam diretamente ligados com a satisfação do usuário, uma vez que se ele for feito de uma forma melhor não proporciona um acréscimo em satisfação.

Os requisitos importantes podem ser descritos da seguinte forma: são diretamente relacionados com a satisfação e aumentam de forma progressiva. Exemplo, se um hotel tem ar condicionado, televisão, banheira, ou seja, quanto maior a quantidade desses itens maior a satisfação. Já os requisitos desejáveis fornecem uma grande satisfação quando estão presentes, freqüentemente adicionando um preço ao produto para satisfazer tais requisitos. O ponto principal é que o fato de uma característica não estar presente pode não trazer nenhuma decepção ao cliente muito menos diminuir a satisfação com o produto final. Freqüentemente, estas são chamadas de necessidades desconhecidas, pois os clientes não sabem que precisam delas até possuí-las. A Figura 4.1 mostra a representação dos níveis de satisfação.

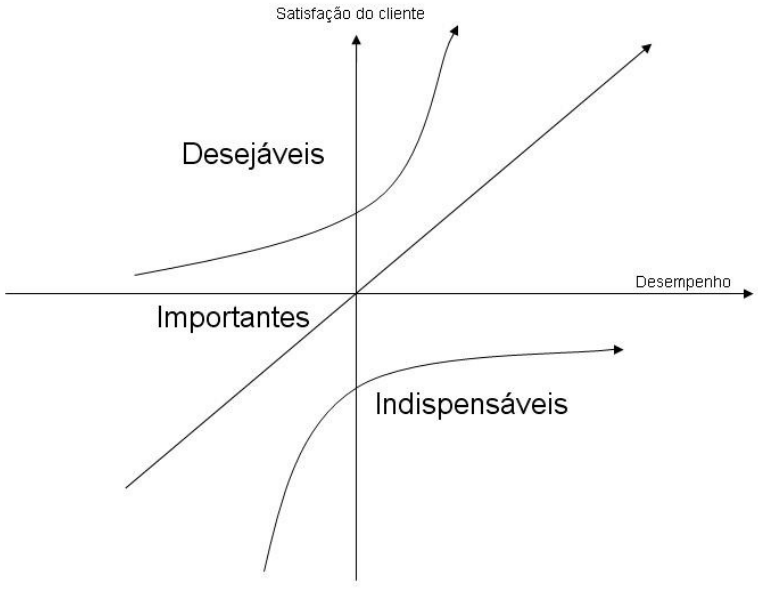


Figura 4.1 Modelo Kano de satisfação do cliente[Cohn 2005]

Ao observar a Figura 4.1 podemos verificar facilmente que o que traz a maior satisfação são os requisitos desejáveis, mas essa satisfação só é alcançada quando no mínimo os requisitos indispensáveis foram satisfeitos. Por causa disto a ordem de priorização dos requisitos é: primeiro são feitos os indispensáveis, depois os importantes e por fim os desejáveis que trariam um valor agregado muito grande se forem atendidos. No entanto, os outros devem ser atendidos antes uma vez que são necessidades básicas e que estão diretamente associadas à valorização do produto.

De acordo com [Cohn 2005], os requisitos indispensáveis não devem ser desenvolvidos obrigatoriamente no início, mas eles devem estar prontos antes do produto final ser entregue uma vez que a satisfação do cliente não seria atingida se na entrega do produto algum item indispensável não fosse entregue. De forma complementar, a ênfase das prioridades é feita desenvolvendo o máximo de requisitos importantes possíveis porque como se observa na Figura 4.1, eles estão diretamente ligados à satisfação do cliente. Finalmente, quando o tempo permite um número mínimo de algumas características desejáveis é atendido.

Ainda de acordo com [Cohn 2005] os requisitos tendem a migrar para baixo no diagrama de Kano com o passar do tempo. Por exemplo, há algum tempo atrás internet sem fio no quarto do hotel era apenas desejável. Hoje é importante e em breve será indispensável.

Figura 4.2 Pirâmide de Maslow [Pirâmide 2009]

Um paralelo interessante feito por [Info Escola 2009] é a comparação do Kano com a pirâmide de Maslow [Pirâmide 2009], na Figura 4.2, e os fatores higiênicos de Herzberg [Herzberg 2009] na Tabela 4.1. Os requisitos mandatórios de Kano são comparáveis aos fatores higiênicos de Herzberg, ou à base da pirâmide de necessidades de Maslow. Na Figura 4.2 é visível que a base da pirâmide é a segurança e as necessidades fisiológicas e os fatores higiênicos tem o mesmo leque de valores como segurança e condições de trabalho, como é possível ver na Tabela 4.1. São os requisitos que o cliente não pede que sejam atendidos e que se atendidos não aumentam muito o grau de satisfação do cliente, porém, se não forem devidamente atendidos podem causar algum descontentamento. Os requisitos importantes são os requisitos do meio da pirâmide e que o cliente sempre pede e quanto mais deles forem atendidos maior será a satisfação do cliente. Já os desejáveis são requisitos que estão além da real necessidade ou do que é esperado pelo cliente. Podemos comparar estes requisitos ao topo da pirâmide de Maslow, como pode ser visto na Figura 4.2 (Auto-realização, Auto-estima) ou aos fatores motivacionais de Herzberg (realização, crescimento e desenvolvimento). Ao atender estes requisitos a empresa certamente estará aumentando drasticamente a satisfação do cliente, mas se eles não forem atendidos, não implicarão em queda de satisfação uma vez que é algo que o cliente não espera. Os requisitos desejáveis podem ser descritos como o "fator Wow" em que o cliente é surpreendido e encantado com a novidade da característica. Eles estão sempre presentes em produtos inovadores e de vanguarda.

Tabela 4.1 Fatores de Herzberg

Fatores Herzberg	
Higiênicos: o ambiente	**Motivadores: o trabalho**
Programa de administração	Realização
Supervisão	Reconhecimento por realização
Condições de trabalho	Trabalho desafiador
Relações interpessoais	Maior responsabilidade
Dinheiro	Crescimento e desenvolvimento
Segurança	

Variações de Kano na Literatura

A motivação principal para a utilização da técnica Kano é a facilidade de priorização que essa técnica fornece para o cliente. Naturalmente, o cliente tem a dificuldade de dizer qual requisito é mais importante para ele do que outro, uma vez que ele acredita que todos os requisitos são importantes.

Em [Robertson 2006], é dito que depois de se fazer a priorização dos requisitos, através das perguntas funcionais e disfuncionais, acaba existindo uma grande quantidade de requisitos que tem a classificação máxima. Esse problema é comumente encontrado na prática de o usuário, no momento de priorizar, simplesmente mostrar que todos aqueles requisitos são importantes que sejam feitos. Por exemplo, como foi descrito no capítulo de metodologias ágeis, Scrum usa a técnica de estimativa utilizando *Business Value* ou valor de negócio. Nesta técnica é muito natural que o cliente coloque vários requisitos senão todos com o valor de negócio máximo fazendo com que não se tenha a real noção do que realmente é mais importante.

A sugestão dada por [Robertson 2006] é de colocar um limite percentual na quantidade de requisitos com o valor 5/5 que no caso da técnica proposta significa a prioridade máxima. A intenção de fazer isso é que o usuário opte por utilizar um requisito ou outro e isso às vezes é uma escolha bastante difícil para o cliente, uma vez que ele não consegue ver o que será analisado para a escolha entre eles. Já em [Cohn 2005], é possível ver uma diferença no momento que o usuário tem uma outra abordagem de resposta. Ele não irá colocar de 1 a 5 seu grau de importância. Ele simplesmente irá dizer o quanto satisfeito ele estará.

A técnica de Kano consiste em fazer 2 perguntas sobre cada requisito. Uma pergunta funcional e outra disfuncional. Existem outros estudos que utilizam essas perguntas, como [Robertson 2006], da satisfação e insatisfação caso o requisito X ou Y seja desenvolvido. No estudo de *Volere* essas respostas são colocadas em valores de 1 a 5, sendo o 1 para satisfação, não preocupado se esse requisito vai ser satisfatoriamente implementado. E 5 para muito feliz se esse requisito for satisfatoriamente implementado. Já para insatisfação, a classificação utilizada é a seguinte: 1 para não liga se o requisito não for satisfatoriamente implementado e 5 para extremamente infeliz se esse requisito não for satisfatoriamente implementado.

O paralelo realizado entre o que deixa o cliente muito satisfeito e o que deixa infeliz se não estiver presente na próxima entrega é muito importante para se ter a real relação de necessidade de cada requisito. É natural que sempre seja respondido que aquele requisito é muito importante que seja feito, pois nós queremos a maior quantidade de atividades possíveis desempenhadas por um software. Nestes casos, é interessante incluir o critério de insatisfação para perceber o que realmente deixa infeliz se não for desenvolvido. Em geral, as variações de Kano propostas na

literatura utilizam os critérios de satisfação e insatisfação que o cliente atribui para cada requisito.

Como Utilizar Kano

O Kano é uma técnica de fácil aplicação que demanda pouco tempo para sua utilização. A técnica Kano funciona da seguinte forma: são selecionados alguns requisitos ou estórias de usuários que se deseja aplicar a técnica e uma vez escolhido os requisitos eles são separados e são listados para montar as perguntas para os usuários. As perguntas são formadas de uma pergunta positiva e outra negativa indicando como o cliente irá se sentir caso o referido requisito ou estória esteja presente na próxima entrega ou não estivesse finalizado qual seria a sua reação.

De acordo com [Cohn 2005], o ideal para aplicar a técnica de Kano seria de ter de 20 a 30 usuários respondendo a pesquisa.

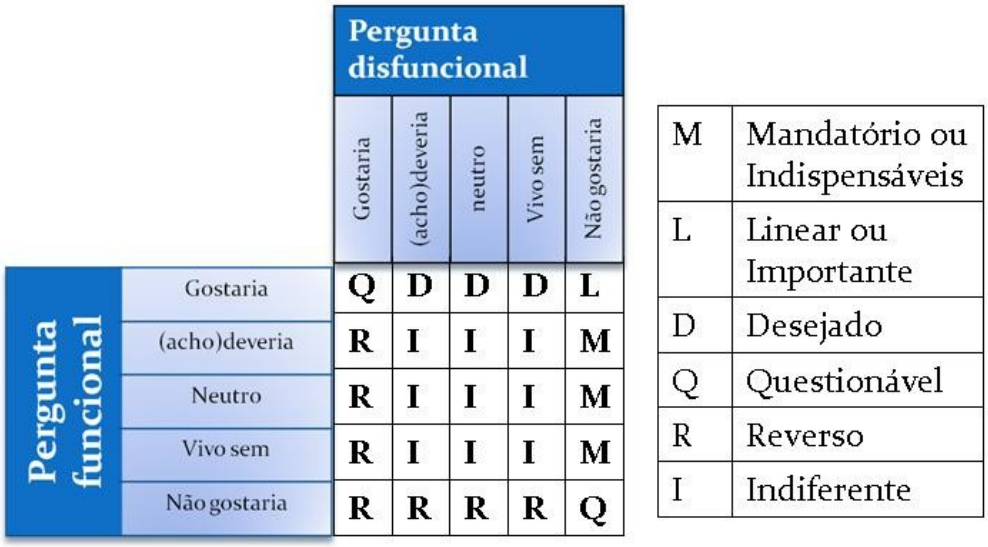
As opções de resposta apresentadas são[Cohn 2005]:

1. Seria melhor dessa forma
2. Eu esperava dessa forma
3. Estou neutro
4. Eu consigo aceitar dessa forma
5. Não gosto dessa forma

A combinação das respostas às perguntas funcionais e disfuncionais gera um resultado para cada requisito, que determina a importância do mesmo. As perguntas seriam formadas da seguinte forma:

- Como você se sentiria caso o Requisito X, estivesse no próximo release?
- Como você se sentiria caso o Requisito X, **NÃO** estivesse no próximo release?

A Figura 4.3, adaptada de [Cohn 2005], ilustra as combinações e os possíveis resultados.



Pergunta funcional \ Pergunta disfuncional	Gostaria	(acho)deveria	neutro	Vivo sem	Não gostaria
Gostaria	Q	D	D	D	L
(acho)deveria	R	I	I	I	M
Neutro	R	I	I	I	M
Vivo sem	R	I	I	I	M
Não gostaria	R	R	R	R	Q

M	Mandatório ou Indispensáveis
L	Linear ou Importante
D	Desejado
Q	Questionável
R	Reverso
I	Indiferente

Figura 4.3 Respostas às perguntas funcionais, disfuncionais e resultados para cada combinação

As categorias ***mandatório/indispensável***, ***importante*** e ***desejado*** foram explicadas na seção 4.1. Já o ***questionável*** se refere quando o cliente coloca respostas antagônicas o que pode significar que ele ou não entendeu as perguntas ou ele não está respondendo com veracidade. O ***reverso*** significa que se aquele requisito for desenvolvido poderá trazer, ao invés de satisfação, uma rejeição ao software ou à determinada funcionalidade. Finalmente, o ***indiferente*** como é visto no quadro acima é o que está presente na maior quantidade de combinações e é utilizado quando o usuário demonstra que não tem necessidade real para esta funcionalidade, ou seja, tanto faz que ela seja satisfeita ou não. A categoria indiferente pode ser originada porque existem outras necessidades mais urgentes ou mesmo porque aquela nova funcionalidade não trará nenhum ganho real.

Após enviar as perguntas a alguns usuários, o resultado deve ser consolidado através dos resultados obtidos na Figura 4.3 formando a Tabela 4.2.

Tabela 4.2 Sumarização dos resultados

Tema	D	L	M	I	R	Q	Classificação
Requisito 1	18.4	**43.8**	22.8	12.8	1.7	0.5	**Linear**
Requisito 2	8.3	30.9	**54.3**	4.2	1.4	0.9	**Mandatório**
Requisito 3	**39.1**	14.8	**36.6**	8.2	0.2	1.1	**Desejado** **Mandatório**

Na tabela 4.2, podemos verificar uma situação hipotética na qual os usuários responderam percentualmente os critérios dos requisitos. Por exemplo, no Requisito 1 18,4 % dos usuários responderam que o consideram Desejado. A dedução de que a

resposta dele foi desejado vem da Figura 4.2 baseado nas respostas em relação às 2 perguntas funcionais e disfuncionais. Ainda nessa tabela é possível identificar pelos percentuais qual a classificação sugerida para cada um dos requisitos.

É possível observar ainda que o Requisito 3, apesar da maioria escolher **desejado,** muitos usuários responderam **mandatório** com isso esse requisito foi classificado como **mandatório e desejado**. Diferentes tipos de usuários e clientes apresentaram diversas expectativas. Isto deve ser visto com atenção porque pode indicar conflitos entre usuários.

Uma vez obtida a Tabela 4.2 o Kano original sugere que já é suficiente fazer a priorização do requisito. No entanto, acreditamos que o resultado apresentado dessa forma ainda é insuficiente para determinar a real importância de cada requisito. Na próxima seção é apresentada uma adaptação desse processo objetivando melhorar sua visualização e conseqüentemente auxiliando a tomada de decisão sobre quais requisitos priorizar.

Processo Adaptado de Kano para Projetos Ágeis

Como foi descrito na seção anterior, Kano é muito interessante no que diz respeito ao embasamento da técnica e nos tipos de dados obtidos. No entanto, em relação à visualização dos resultados ela é bastante insatisfatória, uma vez que é muito abstrato saber quais dos requisitos vão ser priorizados. De acordo com o exemplo da Tabela 4.2, como saber se é melhor implementar o requisito 2 ou 3? Os dois apontam para mandatório. Estes resultados são difíceis de serem interpretados e analisados. Em ambientes ágeis, onde as decisões devem ser tomadas de forma rápida pode ser que a equipe não compreenda de forma correta a essência dos resultados obtidos com a técnica Kano.

Nesta seção apresentamos nossa proposta de adaptação da técnica Kano para melhorar a visualização dos resultados gerados, com o objetivo de auxiliar a tomada de decisão sobre quais requisitos devem ser priorizados para cada versão do produto.

O *Scrum* utiliza para priorizar o valor de negócio definido pelo *Product Owner* e o tamanho definido pelo time durante o *Sprint Planning*. O PO normalmente tem dificuldade de dizer qual o valor de negócio dos itens com apenas sua visão pessoal.

Para solucionar isso, uma melhor visualização desse quadro pode ajudar ao PO e ao *Scrum Master* a entender melhor o que os dados dos usuários querem dizer e

julgar melhor o que deve ser feito. Para dar um suporte maior ao PO, para determinar o valor de negócio, e ao time, para priorizar os itens, o Kano pode ser utilizado com uma melhoria na sua visualização, pois apenas a visualização da tabela, indicando quais os percentuais que foram mais votados pelos usuários, não é suficiente para fazer uma priorização entre os itens.

Com relação às perguntas realizadas para os usuários, para facilitar o entendimento do cliente e aumentar a intuitividade para fornecer as respostas, sugerimos a utilização de ilustrações de faces (*smiles*) indicando a resposta escolhida.

Também é sugerido criar um gráfico de colunas usando uma planilha do Excel, por exemplo, uma vez que podemos visualizar de forma clara algumas distorções entre os dados, levando a decisões importantes. Essas colunas são formadas baseadas no percentual de cada uma das respostas gerando uma visualização em cores das escolhas dos usuários.

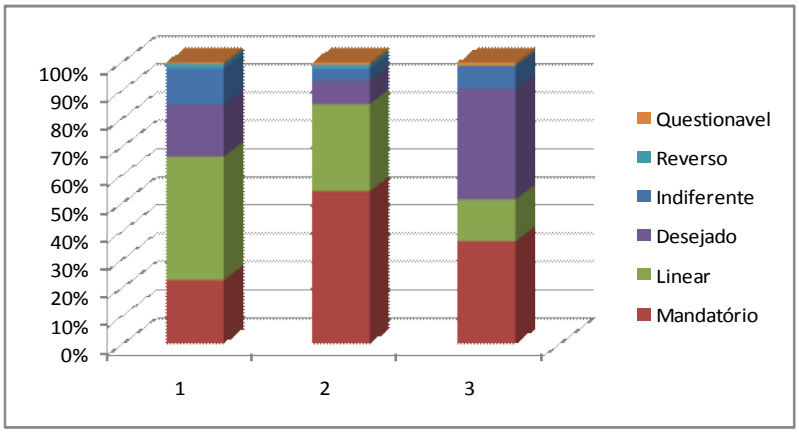


Figura 4.4 Ilustração em forma de colunas da tabela 4.1

Através da figura 4.4 é possível observar com mais clareza quais classificações estão distribuídas para quais requisitos. A partir deste gráfico, várias análises podem ser realizadas. Por exemplo, a quantidade de indiferente, reverso e questionável é totalmente desprezível em relação aos demais itens. Com isso temos que o Requisito 2, com percentual alto de mandatório, é importante tornando-o um grande candidato a ser implementado primeiro, enquanto que o Requisito 3 mostra que muitos usuários escolhera-o como desejável. Dessa forma, o Requisito 1, mesmo tendo a maioria dos usuários determinado-o como linear ou importante, pode ser realizado primeiro do que o requisito 3.

Na figura 4.5 é possível visualizar a alta proporção nesse primeiro requisito de 44% Linear. Na Figura 4.6 é possível visualizar a disparidade de opção por

Mandatório, reforçando mais uma vez o que foi analisado, de que esse deveria ser o primeiro requisito a ser implementado. Ainda é importante destacar que somando Mandatório com Linear esse requisito atinge a impressionante marca de 85% das preferências.

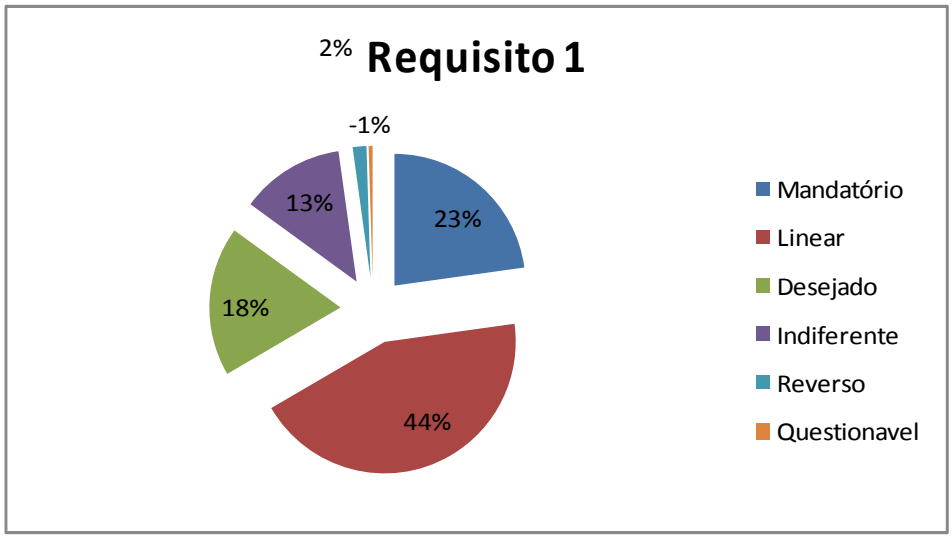


Figura 4.5 Visualização Pizza Requisito 1

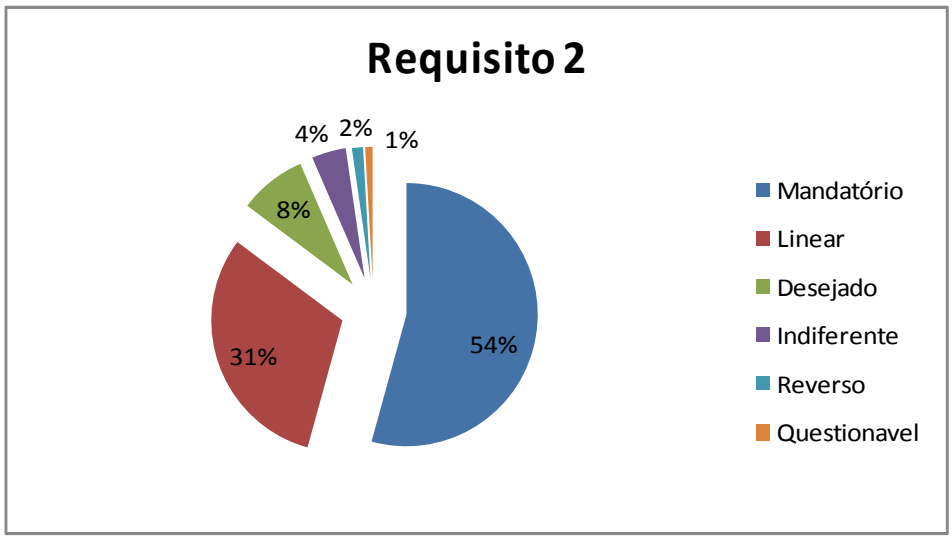


Figura 4.6 Visualização Pizza Requisito 2

Na Figura 4.7 é possível visualizar a proximidade clara entre Desejado e Mandatório e que, como já foi analisado, esse alto grau de desejado fez com que o Requisito 1 fosse considerado mais prioritário, deixando a priorização sugerida da seguinte ordem : Requisito 2, Requisito 1 e finalmente Requisito 3 .

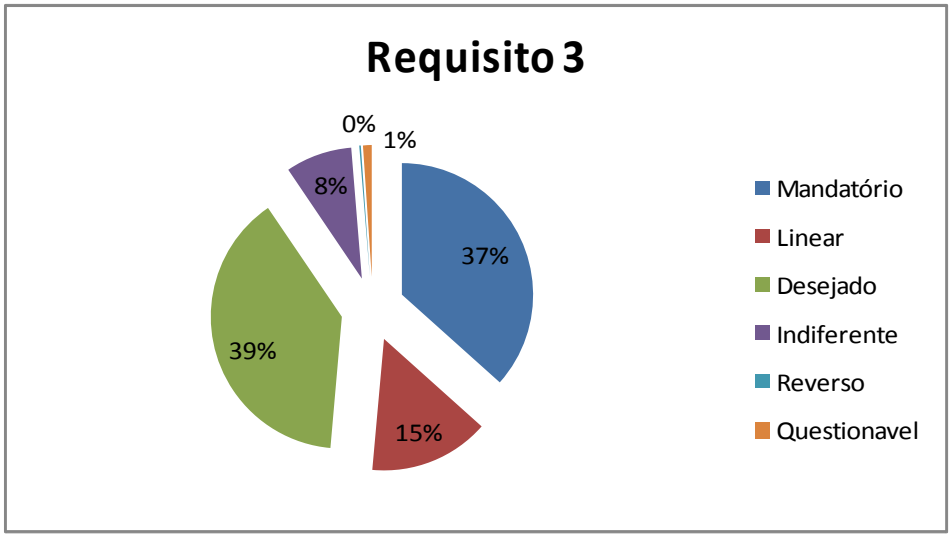


Figura 4.7 Visualização Pizza Requisito 3

Etapas do Processo de Priorização

Nesta seção são apresentadas as etapas do processo de priorização proposto para ambientes ágeis. Com o objetivo de mostrar como o processo poderá ser usado na prática, foi escolhido o método Scrum para detalhar o processo de priorização. Esta escolha é justificada pelo enfoque do Scrum para apoiar as atividades de gerência de projetos. Como a atividade de priorização de requisitos envolve a tomada de decisões estratégicas para o projeto, acreditamos que o Scrum apresenta um nível de detalhamento adequado para fornecer o arcabouço para a nossa proposta de processo de priorização de requisitos.

No momento em que o Scrum recebe as informações do PO ou Cliente dos requisitos com os respectivos valores de negócio, é que foi identificado que eles tinham dificuldade de obter esse número. Esse processo proposto vem para ajudar na escolha desses valores de negócio através de respostas das pessoas que realmente utilizam o software. Trazendo um ROI (Retorno do Investimento) muito maior do que simplesmente seguir os pensamentos do cliente ou PO. A Figura 4.8 apresenta uma visão geral das fases do processo de priorização proposto nesta dissertação.

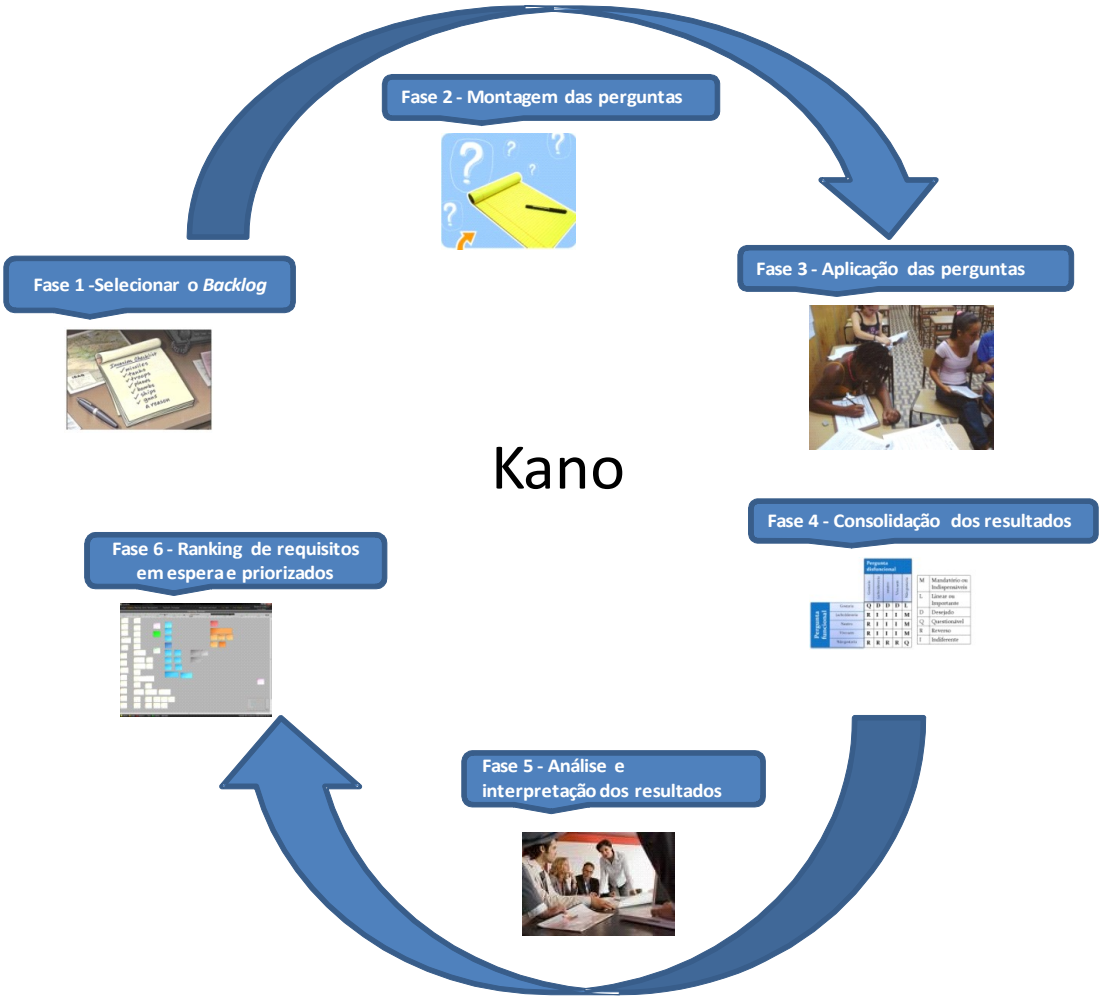


Figura 4.8 Fluxo de fases

Fase 1 - Selecionar o *Backlog* (lista de requisitos)

O primeiro passo para aplicar o processo consiste em selecionar os requisitos que serão priorizados. É recomendado que não seja uma quantidade grande de requisitos para a lista de perguntas não ficar cansativa. O ideal é que seja até 15 requisitos por aplicação. A motivação é não ficar muito cansativo a aplicação da técnica e o usuário acabar por não responder com total atenção as perguntas.

Fase 2 - Montagem das perguntas

Nesta fase os requisitos escolhidos são utilizados para a montagem das perguntas. Aqui já é necessário identificar a abordagem apropriada para a aplicação das perguntas em virtude do contexto do projeto e da distribuição dos usuários.

Por exemplo, quando se tem a equipe de usuários presente é interessante a utilização do questionário em papel uma vez que fica mais fácil para acompanhar todos os usuários além de mostrar mais facilmente o quanto é rápido para responder

os questionários. Outras opções para realizar as perguntas podem ser ferramenta web, planilha Excel ou o próprio papel.

Esta fase deve utilizar os requisitos e montar as perguntas da seguinte forma:

1) Se a próxima versão do SOFTWARE [Nome do software] incluir [a/o] [Nome do requisito ou descrição], como você se sentirá?

Tabela 4.3 Lista de opções de resposta

				
Muito Satisfeito	Satisfeito	Pouco Satisfeito	Indiferente	Insatisfeito

É interessante mostrar essas figuras que deixam a entrevista menos pesada e com uma visualização melhor para os usuários. Após fazer essa pergunta, a mesma é feita com sentido inverso. Como mostrado a seguir:

2) Se a próxima versão do **SOFTWARE [Nome do software] NÃO** incluir [a/o] [Nome do requisito ou descrição], como você se sentirá?

Tabela 4.4 Lista de opções de resposta

				
Muito Satisfeito	Satisfeito	Pouco Satisfeito	Indiferente	Insatisfeito

Lembrando que esse destaque sobre o não é fundamental para que o usuário entenda a diferença entre as perguntas. Essas perguntas devem ser realizadas para cada um dos requisitos escolhidos.

Fase 3 - Aplicação das perguntas

Nessa fase as perguntas montadas são enviadas aos usuários. É importante explicar que aquelas respostas são utilizadas única e exclusivamente para ajudar na escolha dos requisitos que serão incluídos na próxima entrega. E não será usado de forma nenhuma para julgar ou muito menos para dizer se a resposta é correta ou não. Essa é a fase mais trabalhosa do processo uma vez que todos os usuários participantes precisam responder o questionário.

Fase 4 - Consolidação dos resultados

É interessante nessa fase utilizar uma planilha eletrônica para consolidar os resultados e gerar os gráficos com facilidade. O apêndice C tem a cópia dessa planilha. Para a entrada nessa tabela, uma primeira consolidação deve ser feita através da Figura 4.3 que mostra a relação das perguntas com os resultados da combinação.

Nesse momento cada requisito é analisado e baseando-se nas duas respostas do usuário, sua classificação é feita. Caso o usuário responda para a pergunta Funcional, que é a primeira pergunta sobre o requisito, como "Muito Satisfeito" e para a pergunta disfuncional, ou seja, a negativa, que é a segunda pergunta do requisito, como "Indiferente". Demonstrando essa relação na matriz na Figura 4.9.

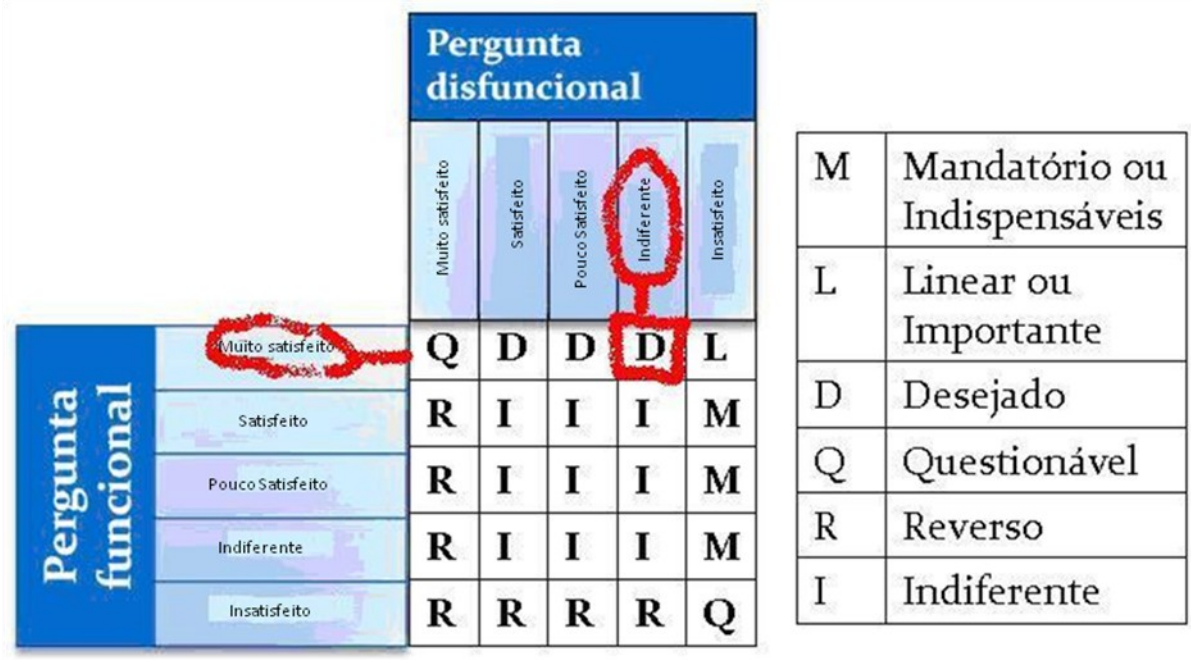


Figura 4.9 Mostrando a utilização do gráfico

Então após fazer essa associação para cada requisito de cada usuário e colocar isso na planilha igual a do apêndice C tem-se os gráficos e os percentuais de cada requisito.

Fase 5 - Análise e interpretação dos resultados

A partir da leitura dos gráficos é possível fazer várias análises. Para realizar a interpretação dos resultados é muito importante que seja usado o bom senso de quem está aplicando a técnica, que deve conhecer as particularidades dos requisitos. Pode-se inclusive priorizar um requisito sabendo que existiria uma rejeição, mas para o gerente aquela funcionalidade pode ser importante, pois envolve uma necessidade mandatória como auditoria, por exemplo.

Deve-se avaliar um requisito pela maioria da classificação, mas devem ser analisados os outros resultados como indicativos importantes.

Alguns indicadores importantes para serem avaliados são:

- O grau de rejeição daquele requisito. Isso indica que além de não deixar o usuário satisfeito, ainda deixa ele descontente, caso o requisito seja implementado.
- O número de indiferente elevado leva a mostrar que aquele requisito não tem uma importância de ser implementado tão grande, o que indica que ele ou não foi bem pensado, ou o usuário não entendeu o que ele faria ou se realmente nesse momento ele deve ficar de lado.
- Um número grande de mandatório aliado com alguma quantidade de Desejado e ou importante, indica que aquele requisito é mandatório para alguns e também tem importância para outros.
- Dentro dessas avaliações deve-se até descartar respostas pouco expressivas que indiquem que o usuário não entendeu a resposta ou até mesmo é apenas contra aquela idéia.

Fase 6 - Ranking de requisitos em espera e priorizados

É interessante, após a análise, fazer um ranking dos requisitos priorizados. E para os requisitos que não atingiram um grau de satisfação esperado, é interessante colocar-los em espera porque as necessidades mudam ao longo do tempo e nesse momento é interessante reavaliar caso seja necessário fazer uma mudança.

Considerações finais

A principal contribuição dessa dissertação está na estruturação da implementação do Kano e principalmente nos gráficos e na análise gráfica que pode ser realizada a partir deles. O modelo Kano proporciona uma boa visão dos requisitos que devem ser priorizados para cada versão do sistema.

A proposta do processo de priorização estudada nessa dissertação é mais bem aproveitada quando se trata de um projeto que tem vários clientes, pois normalmente em projetos desse tipo os clientes que têm mais poder financeiro têm mais poder de barganha, o que faz com que eles consigam que seus requisitos sejam desenvolvidos antes do que alguns requisitos que teriam maior retorno para o produto e, conseqüentemente, para todos os clientes.

A pesquisa Kano deve acontecer durante uma iteração, o que deixa a pesquisa com um tempo razoável e vai servir de entrada para o planejamento da iteração seguinte. No caso do Scrum, seria no *Sprint Planning*.

2 Estudos de Caso

Este capítulo tem como objetivo apresentar e discutir os estudos de caso realizados para mostrar o funcionamento do processo de priorização proposto em projetos reais de duas empresas de software. Será mostrado na próxima seção uma caracterização das empresas onde foi realizada a pesquisa.

Caracterização das empresas

A pesquisa envolveu um estudo de caso em duas empresas de software instaladas no Recife. Foram observados um total de 3 projetos ágeis, sendo 2 projetos de uma empresa e 1 de uma outra.

A primeira empresa estudada foi o C.E.S.A.R - Centro de Estudos e Sistemas Avançados do Recife – que é um instituto privado de inovação que cria produtos, processos, serviços e empresas usando Tecnologia da Informação e Comunicação (TIC). Atuando há mais de 10 anos em âmbito nacional e internacional, o C.E.S.A.R interliga centros de inovação numa rede de conhecimento que realiza projetos de desenvolvimento conectados ao futuro, com qualidade e agilidade.

O C.E.S.A.R também faz parte do Porto Digital, ambiente de empreendedorismo, inovação e negócios de tecnologias da informação e comunicação do estado de Pernambuco que reúne mais de 100 empresas no pólo do Bairro do Recife.

O contexto do estudo foram dois projetos do SEPG – Grupo de melhoria do processo interno de desenvolvimento de software. Esses projetos utilizam Scrum e adotam a técnica para gerenciar o que será melhorado ou desenvolvido pela área de qualidade para ser usado nas diversas áreas dentro da empresa. Os dois projetos são programas do grupo, um é o de gerência de projeto que define como será todo o processo de gerência de projeto no C.E.S.A.R, nesse caso, os usuários são os gerentes de projeto do C.E.S.A.R, estes processos ou programas são aplicados a todos os projetos na área de gerência de projeto. Já o outro projeto foi o de revisão e aprovação. Ele é responsável por definir como será o processo de inspeção, revisão dos documentos e a aprovação de cada projeto. Os usuários do projeto são os SQAs do C.E.S.A.R que usam as informações para definir juntamente com o gerente de projeto o que será utilizado em casa projeto.

A outra empresa escolhida foi a Provider Sistemas, braço de desenvolvimento de sistemas do Grupo Provider. A Provider Sistemas se posiciona no mercado como uma empresa de desenvolvimento de software, na modalidade de produtos próprios e de fábrica de software. A sua linha de produtos tem focos bem definidos, voltado para o segmento de Gestão de Atendimento a Clientes (CRM).

O projeto utilizado foi o eQual Contact Center. É uma plataforma para Gestão de centros de contato ativos e receptivos, atendendo às necessidades operacionais e gerenciais de empresas que utilizam o canal telefônico para atender o seu cliente. O software possui também um Discador. O intuito do sistema é gerenciar todo o relacionamento dos clientes com a empresa cliente, além de monitorar e controlar a performance dos operadores do sistema, gerando relatórios gerenciais dos mais diversos, tais como produtividade e performance.

O eQual Contact Center adotou Scrum há mais de um ano em seu desenvolvimento, e um profissional da área de negócios assumiu o papel de PO do produto, cabendo a ele priorizar relativamente sozinho o que irá ser colocado ou não na próxima Sprint.

2.2 Resultados do Estudo de Caso no CESAR

A pesquisa no Cesar envolveu dois projetos que tem características muito semelhantes, pois se tratam de projetos de melhoria de processo de software. Na pesquisa realizada no Cesar foi utilizada uma planilha como é possível ver no apêndice C para mandar para os usuários os requisitos com as opções. Foi utilizada essa técnica uma vez que os todos os usuários da pesquisa trabalhavam de frente com o computador e em prédios fisicamente diferentes dificultando a entrega de questionários em papel e a consolidação utilizando os mesmos.

A princípio foi pensando em utilizar uma ferramenta que foi desenvolvida em Delphi que já consolidava os resultados instantaneamente, mas depois foi visto que como o usuário precisava instalar na máquina tanto o arquivo executável como a lista de requisitos que deveria ser colocada na mesma pasta. Esse processo apesar de ser simples necessitava de uma explicação detalhada sobre o uso da ferramenta, portanto foi escolhido utilizar o Excel uma vez que é uma ferramenta que é utilizada abundantemente na organização para diversas ações.

2.2.1 Projeto 1 : Gerência de Projeto

Nesta sessão são apresentados os dados da pesquisa realizada no projeto responsável pelos processos utilizados pelos gerentes de projeto na empresa pesquisada. Nos dados mostrados nas tabelas a seguir temos as respostas dos cinco usuários que responderam à pesquisa. Ainda sobre esses usuários o ultimo usuário deixou em branco dois requisitos. Possivelmente por não conhecer bem aquele requisito.

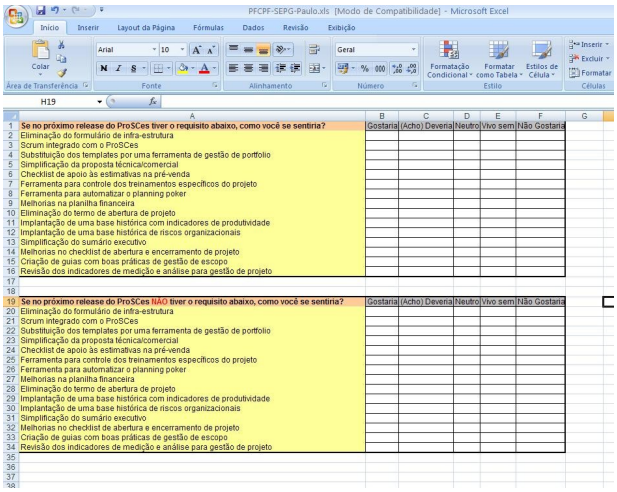
Fase 1: Selecionar o *backlog*

No C.E.S.A.R esse trabalho foi feito pelo P.O. que é a pessoa responsável pela área. Ele observou os itens que estavam no Backlog para selecionar até 15 itens. Foi relatado nesse momento que essa análise foi muito importante para que fossem encontrados requisitos esquecidos na lista e que tinham uma grande importância na visão do Product Owner. Esses itens não vinham sendo priorizados simplesmente porque os novos criados estavam com os pedidos mais urgentes. E assim foram selecionados 15 requisitos.

Fase 2: Montagem das perguntas

Nessa fase foi estudada a possibilidade de aplicar as perguntas utilizando uma ferramenta própria para isso. Foi desenvolvida uma ferramenta em Delphi, mas foi abortada no inicio da sua aplicação por necessitar de uma instalação. Os usuários preferiram uma planilha Excel para pontuar suas respostas para as perguntas funcionais e disfuncionais propostas. Assim foi montada uma planilha com as perguntas como pode ser observado na Figura 5.1.

Assim foi aplicado o questionário onde os usuários marcavam um X em cada resposta para o requisito. Na figura 5.1 existem 2 blocos de perguntas o acima tem “Se no próximo release do ProSCes(nome do processo do C.E.S.A.R) tiver o requisito abaixo, como você se sentiria?” e a lista de requisitos com as opções de resposta nas colunas da direita. No bloco de baixo a pergunta é “Se no próximo release do ProSCes NÃO tiver o requisito abaixo, como você se sentiria?”. Assim foram montadas as perguntas.



	A	B	C	D	E	F	G
1	Se no próximo release do ProSCes tiver o requisito abaixo, como você se sentiria?	Gostaria	(Acho) Deveria	Neutro	Vivo sem	Não Gostaria	
2	Eliminação do formulário de infra-estrutura						
3	Scrum integrado com o ProSCes						
4	Substituição dos templates por uma ferramenta de gestão de portfolio						
5	Simplificação da proposta técnica/comercial						
6	Checklist de apoio às estimativas na pré-venda						
7	Ferramenta para controle dos treinamentos específicos do projeto						
8	Ferramenta para automatizar o planning poker						
9	Melhorias na planilha financeira						
10	Eliminação do termo de abertura de projeto						
11	Implantação de uma base histórica com indicadores de produtividade						
12	Implantação de uma base histórica de riscos organizacionais						
13	Simplificação do sumário executivo						
14	Melhorias no checklist de abertura e encerramento de projeto						
15	Criação de guias com boas práticas de gestão de escopo						
16	Revisão dos indicadores de medição e análise para gestão de projeto						
17							
18							
19	Se no próximo release do ProSCes NÃO tiver o requisito abaixo, como você se sentiria?	Gostaria	(Acho) Deveria	Neutro	Vivo sem	Não Gostaria	
20	Eliminação do formulário de infra-estrutura						
21	Scrum integrado com o ProSCes						
22	Substituição dos templates por uma ferramenta de gestão de portfolio						
23	Simplificação da proposta técnica/comercial						
24	Checklist de apoio às estimativas na pré-venda						
25	Ferramenta para controle dos treinamentos específicos do projeto						
26	Ferramenta para automatizar o planning poker						
27	Melhorias na planilha financeira						
28	Eliminação do termo de abertura de projeto						
29	Implantação de uma base histórica com indicadores de produtividade						
30	Implantação de uma base histórica de riscos organizacionais						
31	Simplificação do sumário executivo						
32	Melhorias no checklist de abertura e encerramento de projeto						
33	Criação de guias com boas práticas de gestão de escopo						
34	Revisão dos indicadores de medição e análise para gestão de projeto						
35							
36							
37							

Figura 5.1 Questionário aplicado no C.E.S.A.R usando uma planilha do Excel

Fase 3: Aplicação das perguntas

Utilizando o questionário montado na Figura 5.1 foi enviado por e-mail para os Gerentes de Projeto que são os usuários do Projeto. Então foram preenchidas com um X as respostas para cada requisito e os resultados foram enviados para o P.O. que iria consolidar todos os dados.

Fase 4: Consolidação dos resultados

Nessa fase foram utilizados os dados das respostas dos usuários para a consolidação dos resultados utilizando o gráfico de matriz Kano. Como pode ser visto na Figura 5.2.

Pergunta funcional \ Pergunta disfuncional	Gostaria	(acho)deveria	neutro	Vivo sem	Não gostaria
Gostaria	Q	D	D	D	L
(acho)deveria	R	I	I	I	M
Neutro	R	I	I	I	M
Vivo sem	R	I	I	I	M
Não gostaria	R	R	R	R	Q

M	Mandatório ou Indispensáveis
L	Linear ou Importante
D	Desejado
Q	Questionável
R	Reverso
I	Indiferente

Figura 5.2 Matriz Kano

Com isso cada resposta dos usuários por requisito foi colocado na matriz gerando assim a Tabela 5.1 que tem já os resultados finais da integração das perguntas funcionais com as disfuncionais.

Tabela 5.1 Respostas dos usuários

Requisito	Usuário1	Usuário2	Usuário3	Usuário4	Usuário5
1	Indiferente	Indiferente	Desejado	Linear	Reverso
2	Indiferente	Linear	Linear	Linear	Linear
3	Indiferente	Linear	Linear	Desejado	Linear
4	Indiferente	Linear	Indiferente	Indiferente	Linear
5	Indiferente	Linear	Linear	Desejado	Linear
6	Indiferente	Indiferente	Desejado	Indiferente	Linear
7	Indiferente	Indiferente	Indiferente	Indiferente	S\ Resp
8	Indiferente	Linear	Mandatório	Linear	Linear
9	Indiferente	Indiferente	Indiferente	Reverso	S\ Resp
10	Indiferente	Mandatório	Linear	Mandatório	Linear
11	Indiferente	Mandatório	Linear	Mandatório	Linear
12	Indiferente	Indiferente	Indiferente	Indiferente	Linear
13	Indiferente	Mandatório	Desejado	Indiferente	Linear
14	Indiferente	Linear	Desejado	Desejado	Linear
15	Indiferente	Mandatório	Linear	Indiferente	Linear

Na Tabela 5.2 foi feito com o somatório das respostas dos usuários de cada requisito. A quantidade de resposta de cada requisito para cada opção de resposta. Por exemplo, no requisito 1 teve 2 usuários que responderam indiferente, 1 linear, 1 desejado e 1 reverso.

Tabela 5.2 Somatório das respostas dos usuários por requisito

Requisito	Mandatório	Linear	Desejado	Indiferente	Reverso	Questionável
1	0	1	1	2	1	0
2	0	4	0	1	0	0
3	0	3	1	1	0	0
4	0	2	0	3	0	0
5	0	3	1	1	0	0
6	0	1	1	3	0	0
7	0	0	0	4	0	0
8	1	3	0	1	0	0
9	0	0	0	3	1	0
10	2	2	0	1	0	0
11	2	2	0	1	0	0
12	0	1	0	4	0	0

13	1	1	1	2	0	0
14	0	2	2	1	0	0
15	1	2	0	2	0	0

Para o próximo passo é calculado o percentual de respostas de cada valor. Da seguinte forma: O requisito 1 obteve 5 respostas das quais 2 foram indiferentes portanto equivalente a 40% das respostas. E assim é montada a Tabela 5.3 com o percentual de resposta de cada requisito por resposta.

Tabela 5.3 Percentual de respostas por requisito

Requisito	Mandatório	Linear	Desejado	Indiferente	Reverso	Questionável
1	0	20	20	40	20	0
2	0	80	0	20	0	0
3	0	60	20	20	0	0
4	0	40	0	60	0	0
5	0	60	20	20	0	0
6	0	20	20	60	0	0
7	0	0	0	100	0	0
8	20	60	0	20	0	0
9	0	0	0	75	25	0
10	40	40	0	20	0	0
11	40	40	0	20	0	0
12	0	20	0	80	0	0
13	20	20	20	40	0	0
14	0	40	40	20	0	0
15	20	40	0	40	0	0

Convertendo esses valores percentuais para um gráfico de barras feito no Excel podemos visualizar a Figura 5.3 abaixo.

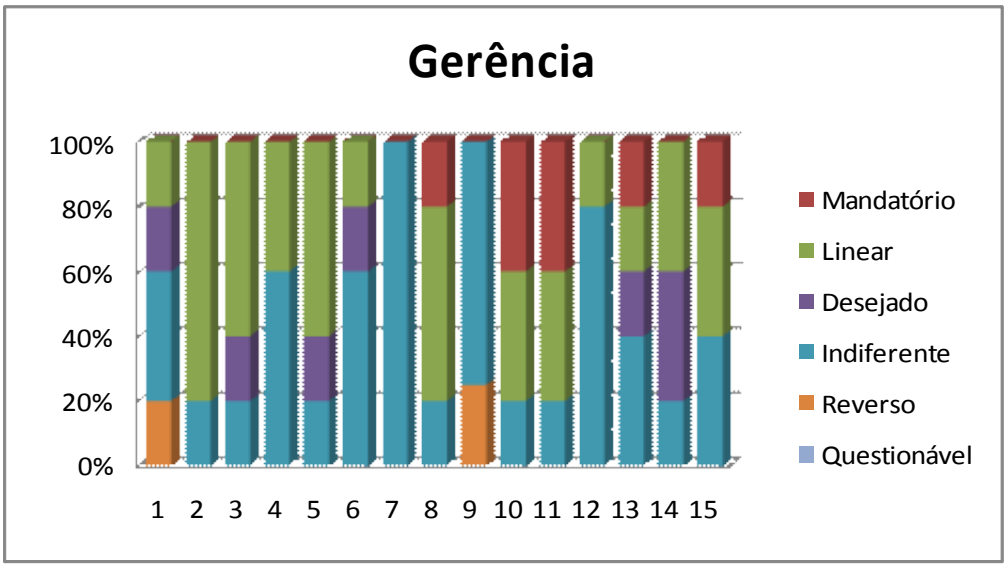


Figura 5.3 Visualização do gráfico do percentual dos resultados por requisito

Fase 5: Análise e interpretação dos resultados

Nessa fase foram utilizados os dados consolidados na fase anterior para realizar as análises. Nos gráficos seguintes é possível verificar isoladamente para cada requisito sua classificação proporcional em relação aos critérios.

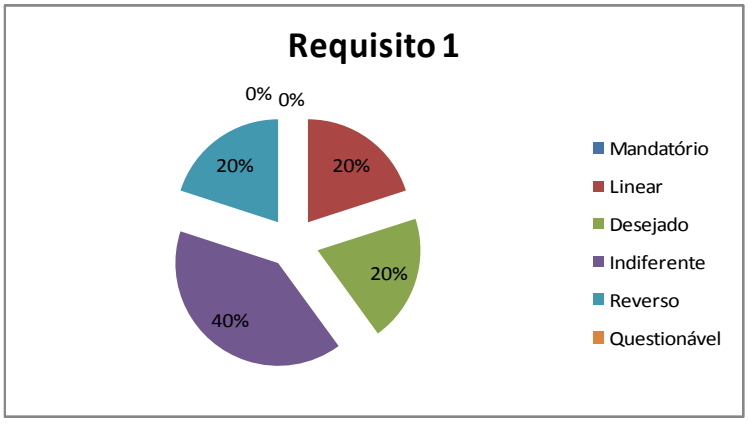


Figura 5.4 Gráfico em Pizza Requisito 1

O requisito 1 é um forte candidato a não ser implementado por enquanto uma vez que tem 40% de pessoas vendo ele como Indiferente e 20% como Reverso. Com isso ele já seria forte candidato a não ser implementado. Mas ele ainda tem 20% de Desejado e 20% de Linear o que não o torna indispensável para nenhum usuário. Com isso não é candidato a ser implementado nesse momento. No entanto, pode ser que ele passe a ser importante com o passar do tempo uma vez que essa classificação muda de acordo com a mudança das necessidades do cliente. Com isso um mesmo requisito pode e deve ser questionado mais de uma vez.

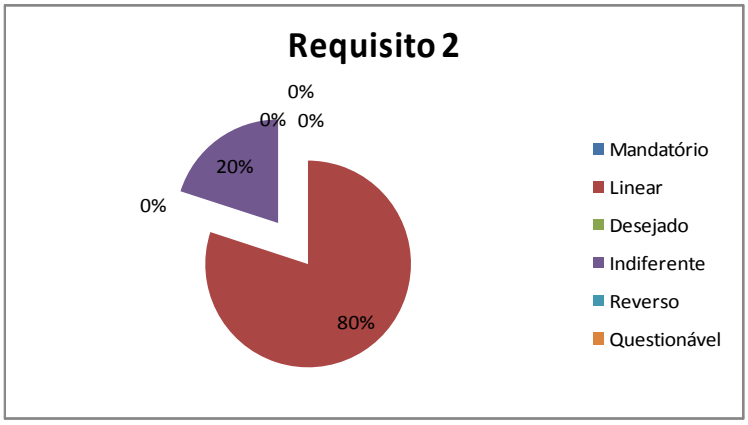


Figura 5.5 Gráfico em Pizza Requisito 2

Já o requisito 2 ele deverá ser implementado uma vez que é possível observar que ele é Linear para 80% dos usuários.

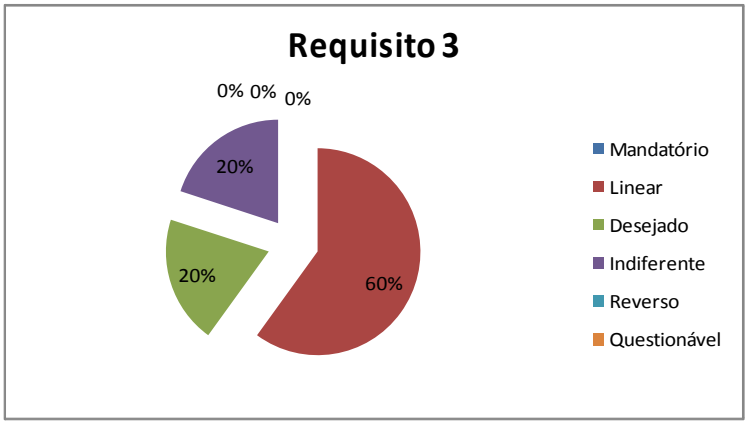


Figura 5.6 Gráfico em Pizza Requisito 3

O requisito 3 apresenta um resultado bem interessante uma vez que 60% dos usuários o classificam como Linear 20% como Desejado e apenas 20% o considera indiferente. Lembrando que ele deverá ser implementado, mas obviamente após o requisito 2.

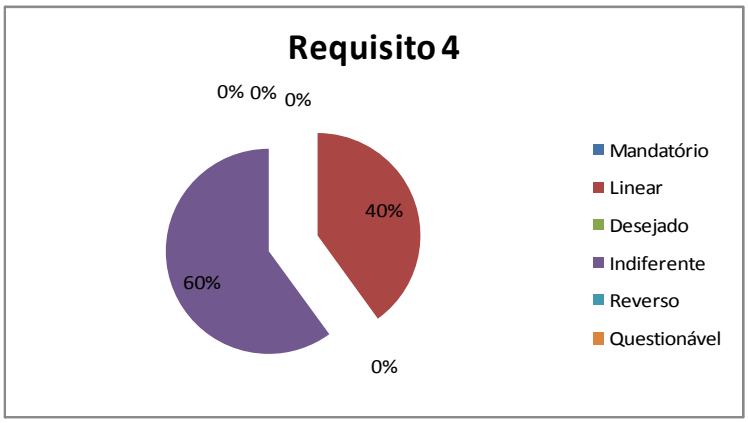


Figura 5.7 Gráfico em Pizza Requisito 4

O requisito 4 apresenta um alto índice de indiferente 60% o que o torna candidato a esperar um pouco mesmo tendo 40% dos usuários que o considera como linear ele não ficaria totalmente isolado por esses 40% mas entraria com certeza com uma prioridade menor que os requisitos 2 e 3.

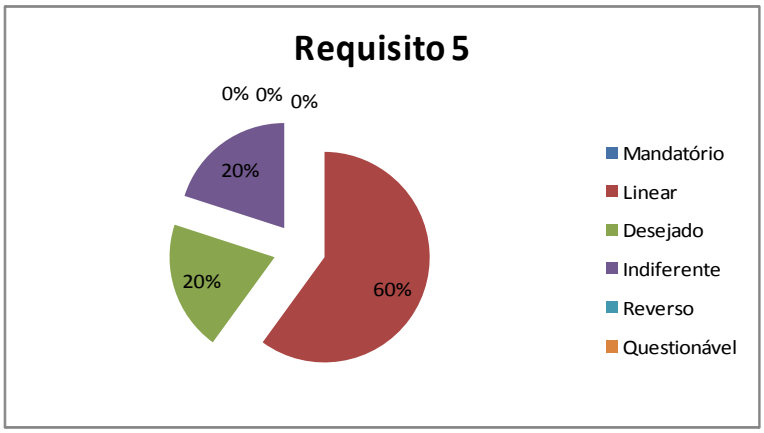


Figura 5.8 Gráfico em Pizza Requisito 5

O requisito 5 tem 60% de linear e 20% de Desejado e 20 % de Indiferente o que torna esse requisito pelo menos desejado por 80% dos usuários tornando assim um forte candidato a ser implementado.

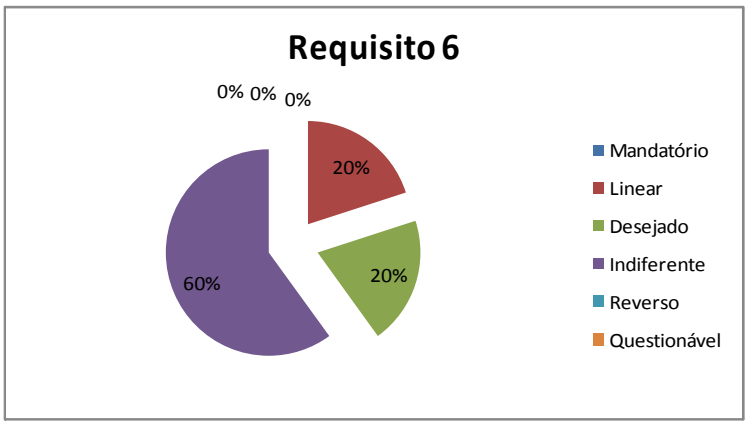


Figura 5.9 Gráfico em Pizza Requisito 6

O requisito 6, assim como o requisito 4, tem 60% de indiferente, 20% de desejado e 20% de linear deixando esse na lista de espera sendo menos prioritário que o requisito 4.

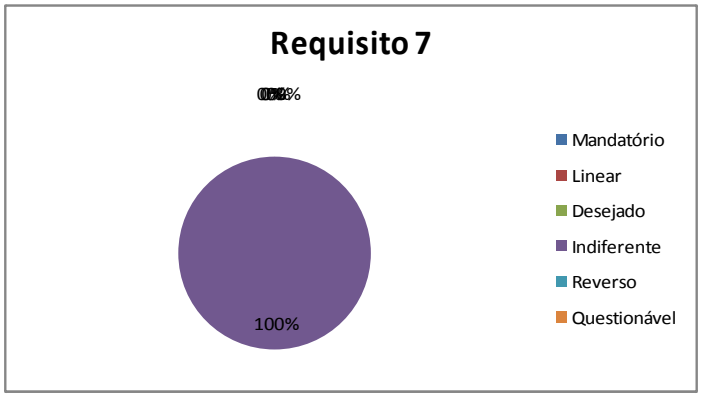


Figura 5.10 Gráfico em Pizza Requisito 7

O requisito 7 é forte candidato a ser excluído da lista de backlog ou de ser colocado em espera por enquanto, com 100% das respostas como indiferente, é bem provável que não seja o momento de ser implementado.

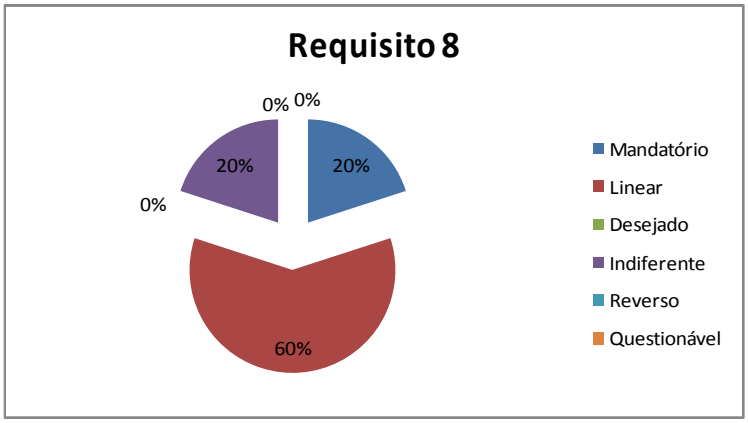


Figura 5.11 Gráfico em Pizza Requisito 8

O requisito 8 foi o mais importante até aqui uma vez que tem 60% de Linear , 20% de Mandatório tornando mais importante que os requisitos 2,3 e 5.

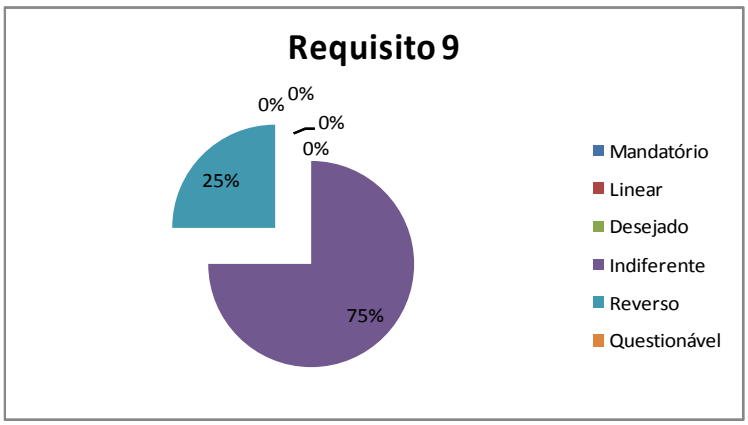


Figura 5.12 Gráfico em Pizza Requisito 9

O requisito 9 é um requisito que deve ser re-analisado uma vez que foi visto 75% de pessoas com indiferença e 25% de rejeição ao requisito o que significa que por ou os usuários não entenderam a proposta do requisito ou vem de uma demanda da gerência que trás alguma insatisfação dos usuários. Mas de qualquer forma ele deve ser repensado.

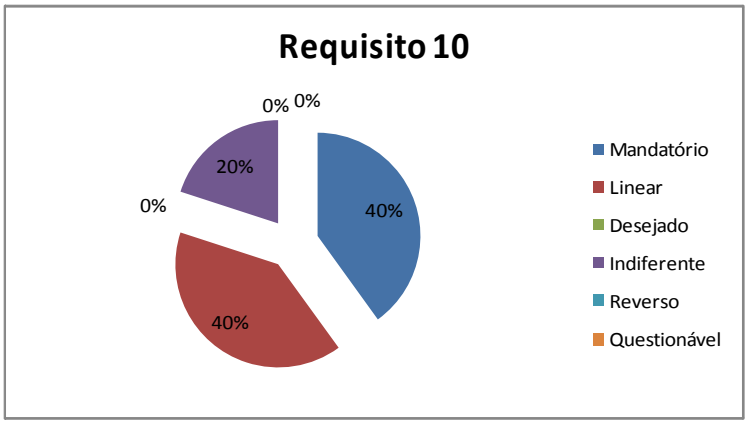


Figura 5.13 Gráfico em Pizza Requisito 10

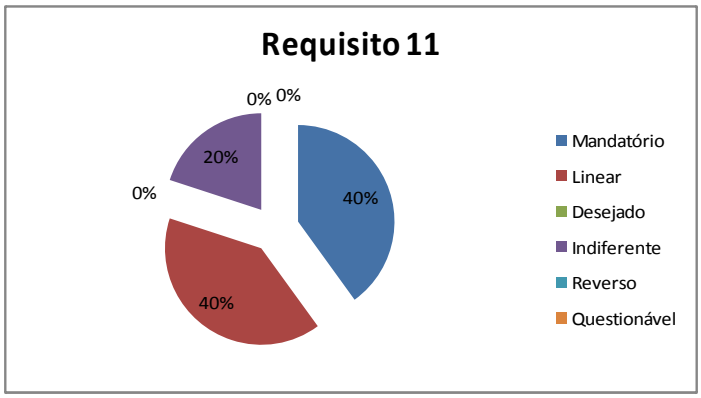


Figura 5.14 Gráfico em Pizza Requisito 11

Os requisitos 10 e 11 gráficos 5.13 e 5.14 tiveram os mesmos percentuais 40% Mandatório, 40% Linear e 20% Indiferente, os tornando os mais importantes até aqui uma vez que tem mais Mandatório que o requisito 8.

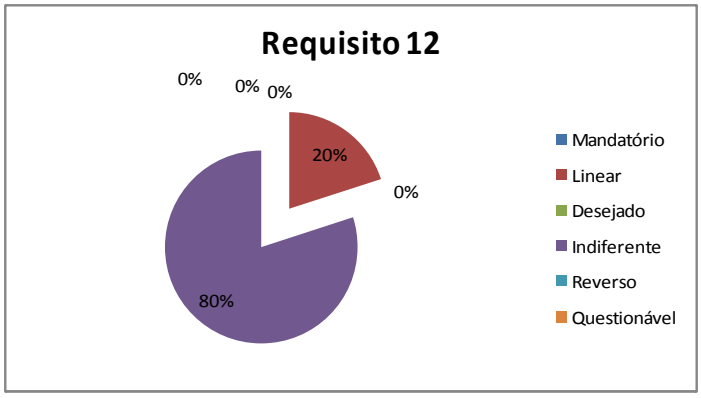


Figura 5.15 Gráfico em Pizza Requisito 12

O requisito 12 deve entrar na lista dos requisitos que irão esperar ou serão estudados novamente no futuro.

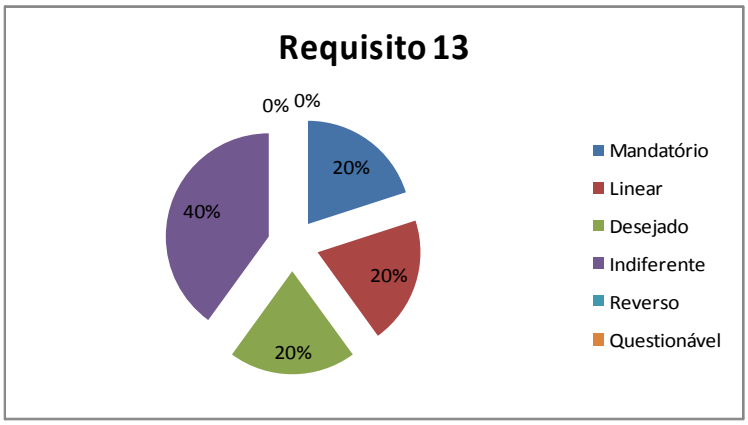


Figura 5.16 Gráfico em Pizza Requisito 13

O requisito 13 tem 40% de indiferente 20% de mandatório, linear e desejado. O que o torna um candidato com pouca prioridade.

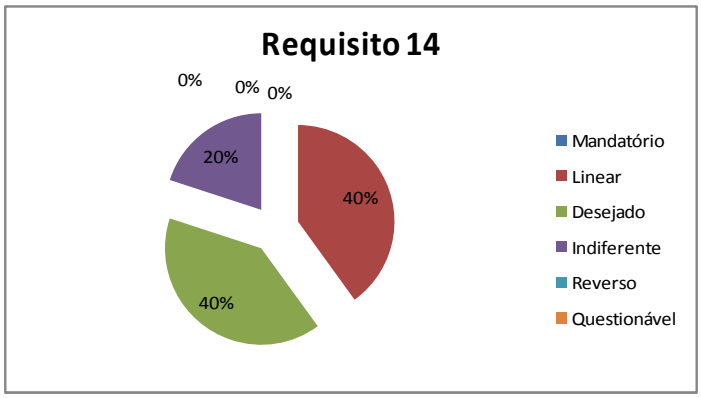


Figura 5.17 Gráfico em Pizza Requisito 14

O requisito 14 tem 40% de linear e 40% de Desejado e 20% de Indiferente. O que o torna candidato a ser feito sim, mas em virtude dos outros já citados existe outros mais prioritários.

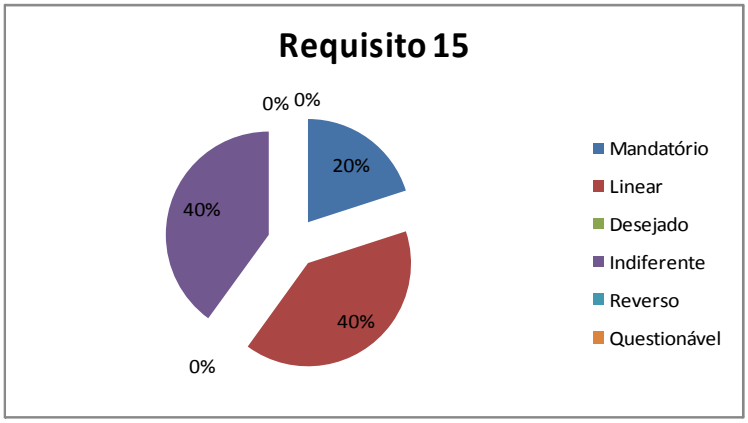


Figura 5.18 Gráfico em Pizza Requisito 15

O requisito 15 tem 40% de Indiferente e Linear e 20% de mandatório o que mostra claramente que é vontade de 60% mas não estaria numa prioridade tão grande quanto os já citados.

Fase 6: Ranking de requisitos em espera e priorizados

Nessa fase com base em todas as análises realizadas é montada uma tabela para visualização do Ranking dos requisitos.

Tabela 5.4 Sugestão de priorização

Prioridade	Priorizados	Em Espera
1	Requisito 10	Requisito 1
2	Requisito 11	Requisito 4
3	Requisito 2	Requisito 6
4	Requisito 3	Requisito 7
5	Requisito 5	Requisito 9
6	Requisito 8	Requisito 12
7	Requisito 14	
8	Requisito 13	
9	Requisito 15	

A tabela 5.4 apresenta a consolidação dos resultados anteriores e mostra a lista de prioridade de quais requisitos devem ser selecionados para serem implementados e quais devem ficar em espera. É importante lembrar que essa priorização utilizada foi formada única e exclusivamente com o pensamento na satisfação dos usuários que responderam a pesquisa. É prudente levar em conta dados como esforço e custo para implementar os requisitos antes de gerar a priorização final.

2.2.2 Projeto 2 : Revisão e Aprovação

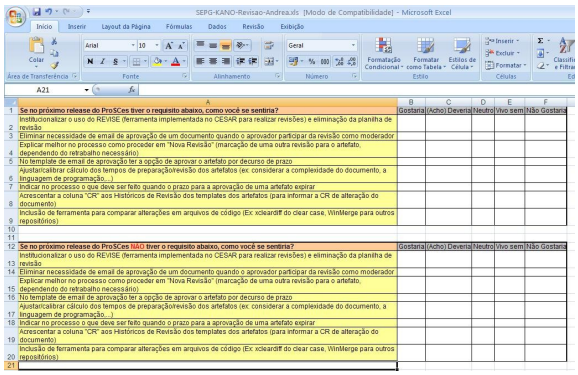
Nesta sessão são apresentados os dados da pesquisa realizada no projeto responsável pelos processos utilizados pelos SQAs(Software Quality Assurance) na empresa pesquisada. Nos dados mostrados a seguir temos as respostas de quatro usuários que responderam à pesquisa.

Fase 1: Selecionar o backlog

No C.E.S.A.R esse trabalho foi feito pelo P.O. de revisão e aprovação que é a pessoa responsável pela área. Ele olhou os itens que estava no Backlog para serem feitos para selecionar até 15 itens. Foram selecionados 8 requisitos.

Fase 2: Montagem das perguntas

Nessa fase foi montada uma planilha Excel com as perguntas como pode ser observado na figura 5.5.



	A	B	C	D	E	F
1	Se no próximo release do ProSCes tiver o requisito abaixo, como você se sentiria?	Gostaria	(Acho) Deveria	Neutro	Vivo sem	Não Gostaria
2	Institucionalizar o uso do REVISE (ferramenta implementada no CESAR para realizar revisões) e eliminação da planilha de revisão					
3	Eliminar necessidade de email de aprovação de um documento quando o aprovador participar da revisão como moderador					
4	Explicar melhor no processo como proceder em "Nova Revisão" (marcação de uma outra revisão para o artefato, dependendo do retrabalho necessário)					
5	No template de email de aprovação ter a opção de aprovar o artefato por decurso de prazo					
6	Ajustar/calibrar cálculo dos tempos de preparação/revisão dos artefatos (ex: considerar a complexidade do documento, a linguagem de programação,...)					
7	Indicar no processo o que deve ser feito quando o prazo para a aprovação de uma artefato expirar					
8	Acrescentar a coluna "CR" aos Históricos de Revisão dos templates dos artefatos (para informar a CR de alteração do documento)					
9	Inclusão de ferramenta para comparar alterações em arquivos de código (Ex: xcleardiff do clear case, WinMerge para outros repositórios)					
10						
11						
12	Se no próximo release do ProSCes NÃO tiver o requisito abaixo, como você se sentiria?	Gostaria	(Acho) Deveria	Neutro	Vivo sem	Não Gostaria
13	Institucionalizar o uso do REVISE (ferramenta implementada no CESAR para realizar revisões) e eliminação da planilha de revisão					
14	Eliminar necessidade de email de aprovação de um documento quando o aprovador participar da revisão como moderador					
15	Explicar melhor no processo como proceder em "Nova Revisão" (marcação de uma outra revisão para o artefato, dependendo do retrabalho necessário)					
16	No template de email de aprovação ter a opção de aprovar o artefato por decurso de prazo					
17	Ajustar/calibrar cálculo dos tempos de preparação/revisão dos artefatos (ex: considerar a complexidade do documento, a linguagem de programação,...)					
18	Indicar no processo o que deve ser feito quando o prazo para a aprovação de uma artefato expirar					
19	Acrescentar a coluna "CR" aos Históricos de Revisão dos templates dos artefatos (para informar a CR de alteração do documento)					
20	Inclusão de ferramenta para comparar alterações em arquivos de código (Ex: xcleardiff do clear case, WinMerge para outros repositórios)					
21						

Figura 5.19 Questionário aplicado no projeto de revisão e aprovação

Assim foi aplicado o questionário onde os usuários marcavam um X em cada resposta para o requisito. Na figura 5.19 existem 2 blocos de perguntas o acima tem "Se no próximo release do ProSCes(nome do processo do C.E.S.A.R) tiver o requisito abaixo, como você se sentiria?" e a lista de requisitos com as opções de resposta nas colunas da direita. No bloco de baixo a pergunta é "Se no próximo release do ProSCes NÃO tiver o requisito abaixo, como você se sentiria?". Assim foram montadas as perguntas.

Fase 3: Aplicação das perguntas

Utilizando o questionário montado na figura 5.5 foi enviado por e-mail para os *Software Quality Assurances* que são os usuários do Projeto. Então foram preenchidas com um X as respostas para cada requisito e foi respondido para o P.O. que iria consolidar todos os dados.

Fase 4: Consolidação dos resultados

Nessa fase foram utilizados os dados das respostas dos usuários para a consolidação dos resultados utilizando o gráfico de matriz Kano. Como pode ser visto na figura 5.2. Com isso cada resposta dos usuários por requisito foi colocado na matriz gerando assim a tabela 5.5 que tem já os resultados finais da junção das perguntas funcionais com as disfuncionais.

Tabela 5.5 Respostas dos usuários

Requisito	Usuário 1	Usuário 2	Usuário 3	Usuário 4
1	Linear	Linear	Linear	Reverso
2	Linear	Desejado	Mandatório	Indiferente
3	Linear	Indiferente	Indiferente	Indiferente
4	Reverso	Reverso	Indiferente	Indiferente
5	Desejado	Indiferente	Reverso	Linear
6	Desejado	Indiferente	Desejado	Linear
7	Linear	Indiferente	Reverso	Indiferente
8	Desejado	Indiferente	Indiferente	Indiferente

Na tabela abaixo foi feito o somatório das respostas dos usuários de cada requisito. A quantidade de resposta de cada requisito para cada opção de resposta. Por exemplo, no requisito 1 teve 3 usuários que responderam linear e 1 reverso. Como pode ser visto na Tabela 5.6.

Tabela 5.6 Somatório das respostas dos usuários por requisito

Requisito	Mandatório	Indiferente	Linear	Desejado	Reverso	Questionável
1	0	0	3	0	1	0
2	1	1	1	1	0	0
3	0	3	1	0	0	0
4	0	2	0	0	2	0
5	0	1	1	1	1	0
6	0	1	1	2	0	0
7	0	2	1	0	1	0
8	0	3	0	1	0	0

Para o próximo passo é calculado o percentual de respostas de cada valor. Da seguinte forma: O requisito 1 obteve 4 respostas das quais 3 foram linear portanto 75% das respostas. E assim é montada a Tabela 5.7 com o percentual de resposta de cada requisito por resposta.

Tabela 5.7 Percentual de respostas por requisito

Requisito	Mandatório	Indiferente	Linear	Desejado	Reverso	Questionável
1	0	0	75	0	25	0
2	25	25	25	25	0	0
3	0	75	25	0	0	0
4	0	50	0	0	50	0
5	0	25	25	25	25	0
6	0	25	25	50	0	0
7	0	50	25	0	25	0

Legenda:
I – Indiferente
L – Linear
M - Mandatório
D- Desejado
R –Reverso
Q - Questionável

8	0	75	0	25	0	0

Convertendo esses valores percentuais para um gráfico de barras feito no Excel podemos visualizar a Figura 5.20.

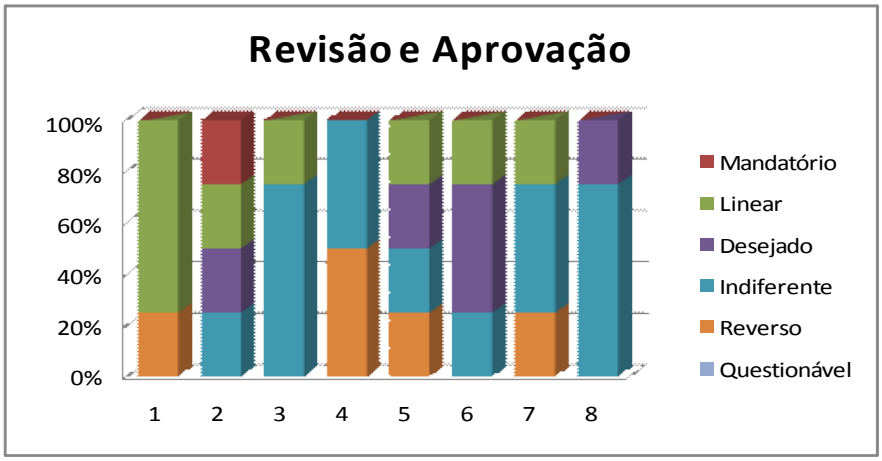


Figura 5.20 Visualização do gráfico do percentual dos resultados por requisito

Fase 5: Análise e interpretação dos resultados

Nessa fase foram utilizados os dados consolidados na fase anterior para realizar as análises. Nos gráficos seguintes é possível verificar isoladamente para cada requisito sua proporção em relação às respostas.

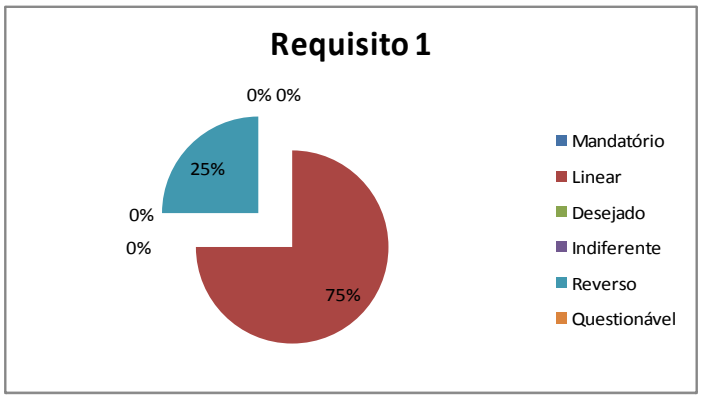


Figura 5.21 Gráfico em Pizza Requisito 1

O requisito 1 é claramente muito importante de ser implementado, pois ele é 75% Linear. Mas o 25% de reverso o faz pensar um pouco se essa implementação pode ser feita sem gerar impacto no trabalho dos usuários que mostraram sua insatisfação caso a implementação do requisito.

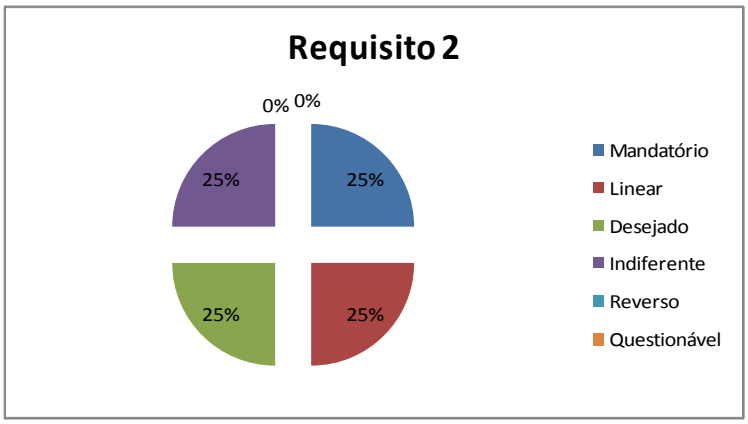


Figura 5.22 Gráfico em Pizza Requisito 2

O requisito 2 é 25% Mandatório, Linear, Indiferente e Desejado como isso indica uma satisfação a 75% dos usuários ele é recomendável.

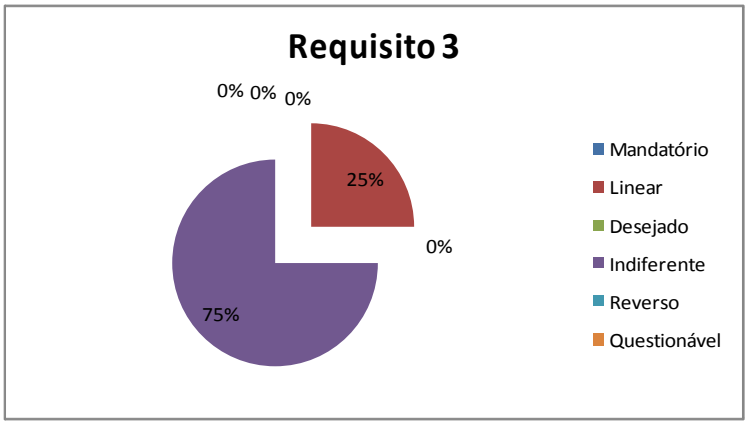


Figura 5.23 Gráfico em Pizza Requisito 3

O requisito 3 apresenta um número alto de 75% de Indiferente o tornando um forte candidato a entrar em espera uma vez que hoje ainda não é vista sua real necessidade.

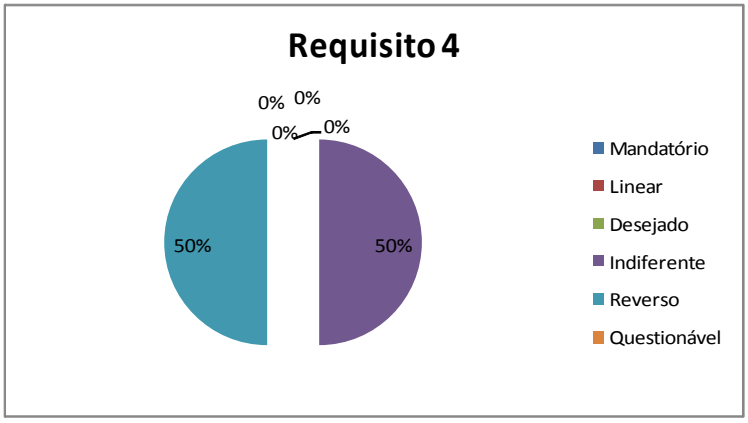


Figura 5.24 Gráfico em Pizza Requisito 4

O requisito 4 é 50% indiferente e 50% reverso tornado ele com mais problemas que o 3 uma vez que além de ter 50% de usuários que o consideram indiferente ainda existe 50% de usuários que demonstraram o desejo de não contar com a sua alteração.

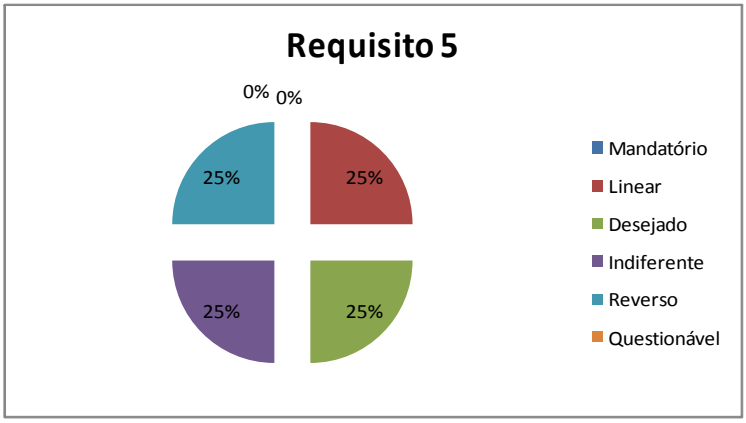


Figura 5.25 Gráfico em Pizza Requisito 5

O requisito 5 25% Linear, Desejado, Indiferente e Reverso. O aconselhado seria colocar ele em espera uma vez que apenas 25 % dos usuários ficariam com certa satisfação caso ele fosse implementado enquanto teria o mesmo percentual que não gostaria da sua implementação talvez com um pouco mais de tempo ou uma reformulação da idéia resolva o problema.

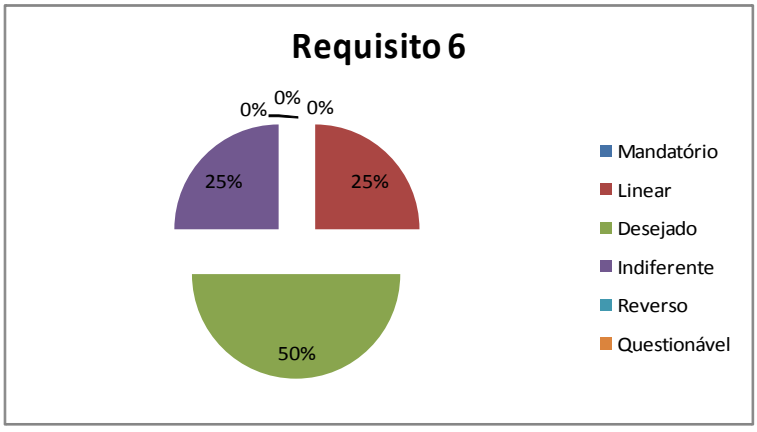


Figura 5.26 Gráfico em Pizza Requisito 6

Requisito 6 tem 50% de Desejado e 25% de Linear e Indiferente o que o torna um requisito na média como Desejado.

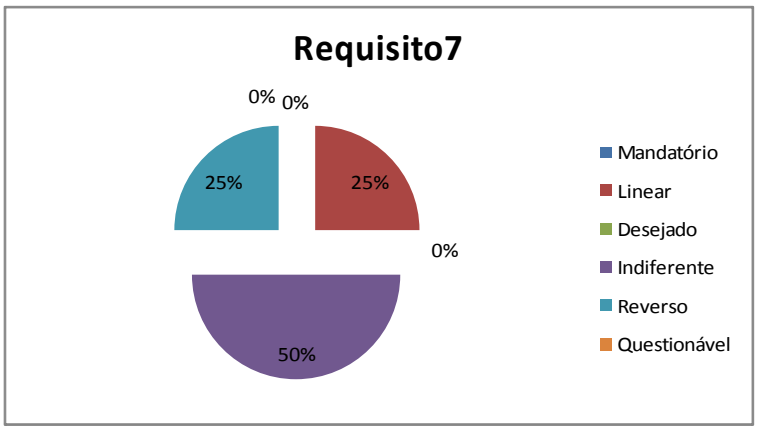


Figura 5.27 Gráfico em Pizza Requisito 7

O requisito 7 com 50% de indiferente e 25% de Reverso e Linear não deveria ser implementado nesse momento ficado na lista de espera.

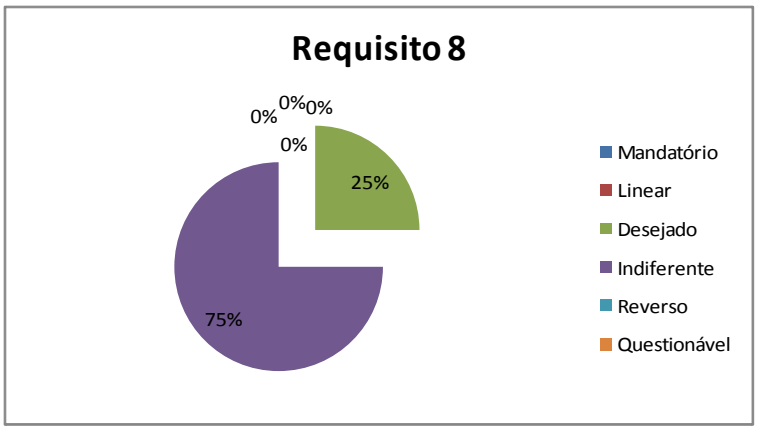


Figura 5.28 Gráfico em Pizza Requisito 8

O requisito 8 já que 75% o consideram indiferente e 25% o consideram Desejado o que é pouco para a implementação de um requisito, a não ser que não exista mais nada na lista para ser implementado.

Fase 6: Ranking de requisitos em espera e priorizados

Nessa fase com base em todas as análises realizadas é montada uma tabela para visualização do Ranking dos requisitos.

Tabela 5.8 Sugestão de priorização

Prioridade	Priorizados	Em Espera
1	Requisito 2	Requisito 3
2	Requisito 1	Requisito 4
3	Requisito 6	Requisito 5

4	Requisito 8	Requisito 7

A tabela 5.8 acima foi feita baseada na análise já demonstrada. Lembrando que essa priorização utilizada foi formada única e exclusivamente com o pensamento na satisfação dos usuários que responderam a pesquisa. É prudente levar em conta dados como esforço dos requisitos antes de gerar a priorização final.

Resultados do Estudo de Caso na Provider

A pesquisa realizada na Provider envolveu um sistema real desenvolvido na empresa. A pesquisa foi feita através de papel utilizando o questionário como mostrado no Apêndice B anexo. Podemos verificar nesse questionário os desenhos que facilitam muito a visualização do questionário convidando os usuários a responder uma vez que ele percebe que são apenas algumas respostas objetivas.

Fase 1: Selecionar o *Backlog*

Na Provider esse trabalho foi feito pelo P.O. que era a pessoa responsável pela priorização dos requisitos. Ele observou os itens que estava no *Backlog* para serem implementados e selecionou 8 requisitos.

Fase 2: Montagem das perguntas

Nessa fase do estudo de caso foram utilizados os questionários em papel que além de trazerem uma maior facilidade, pois o usuário não tem que enviar e-mail com suas respostas. O formato das perguntas e respostas pode ser visto no Apêndice B. Foram montadas as perguntas como no *template* para cada um dos 8 requisitos.

Fase 3: Aplicação das perguntas

Utilizando o questionário montado de acordo com o apêndice B foi entregue impresso aos usuários do sistema desenvolvido que são funcionários da mesma empresa. Então os questionários foram respondidos no papel sem identificação e entregues ao P.O. que havia explicado como se daria a aplicação e que se tratava de uma pesquisa apenas na intenção de entender suas reais necessidades. Que ninguém seria julgado por nenhuma resposta.

Fase 4: Consolidação dos resultados

Nessa fase é utilizado os dados das respostas dos usuários para a consolidação dos resultados utilizando o gráfico de matriz Kano. Como pode ser visto na Figura 5.2. Com isso a resposta dos usuários em relação a cada requisito foi colocada na matriz gerando assim a Tabela 5.9 que tem os resultados finais da integração das perguntas funcionais com as disfuncionais.

Tabela 5.9 Respostas dos usuários

Requisito	U1	U2	U3	U4	U5	U6	U7	U8	U9
1	M	M	I	M	I	M	L	I	I
2	M	L	I	L	L	I	L	M	L
3	I	I	L	I	I	I	I	M	I
4	M	I	L	I	I	I	I	M	L
5	M	I	I	I	I	I	I	I	I
6	I	I	I	I	I	I	I	I	I
7	L	M	R	M	L	L	M	L	M
8	L	I	I	I	M	I	I	I	M

Legenda:
I – Indiferente
L – Linear
M - Mandatório
D- Desejado
R – Reverso
Q – Questionável
U- Usuário

Na tabela 5.10 foi feito um somatório das respostas dos usuários de cada requisito. A quantidade de resposta de cada requisito para cada opção de resposta. Por exemplo, no requisito 1 teve 4 usuários que responderam indiferente,4 indiferente e 1 linear.

Tabela 5.10 Somatório das respostas dos usuários por requisito

Requisito	Mandatório	Indiferente	Linear	Desejado	Reverso	Questionável
1	4	4	1	0	0	0
2	2	2	5	0	0	0
3	1	7	1	0	0	0
4	2	5	2	0	0	0
5	1	8	0	0	0	0
6	0	9	0	0	0	0
7	4	0	4	0	1	0
8	2	6	1	0	0	0

Para o próximo passo é calculado o percentual de respostas de cada valor. Da seguinte forma: O requisito 1 obteve 10 respostas das quais 4 foram indiferentes portanto 44,44% das respostas. E assim é montada a tabela 5.11 com o percentual de resposta de cada requisito por resposta.

Tabela 5.11 Percentual de respostas por requisito

Requisito	Mandatório	Indiferente	Linear	Desejado	Reverso	Questionável
1	44,44	44,44	11,11	0,00	0,00	0,00
2	22,22	22,22	55,56	0,00	0,00	0,00
3	11,11	77,78	11,11	0,00	0,00	0,00
4	22,22	55,56	22,22	0,00	0,00	0,00
5	11,11	88,89	0,00	0,00	0,00	0,00
6	0,00	100,00	0,00	0,00	0,00	0,00
7	44,44	0,00	44,44	0,00	11,11	0,00
8	22,22	66,67	11,11	0,00	0,00	0,00

Convertendo esses valores percentuais para um gráfico de barras feito no Excel podemos visualizar a Figura 5.29.

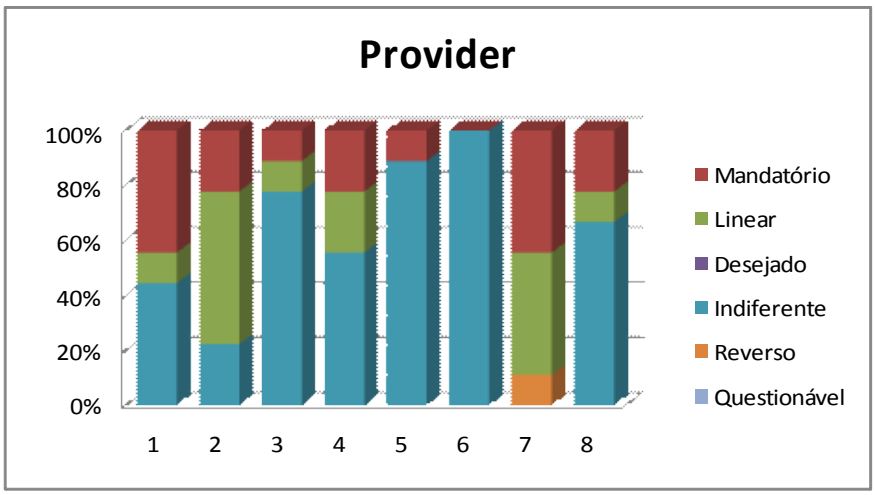


Figura 5.29 Visualização do gráfico do percentual dos resultados por requisito

Fase 5: Análise e interpretação dos resultados

Nessa fase são utilizados os dados consolidados na fase anterior para realizar as análises. Nos gráficos seguintes é possível verificar isoladamente cada requisito sua proporção em relação às respostas.

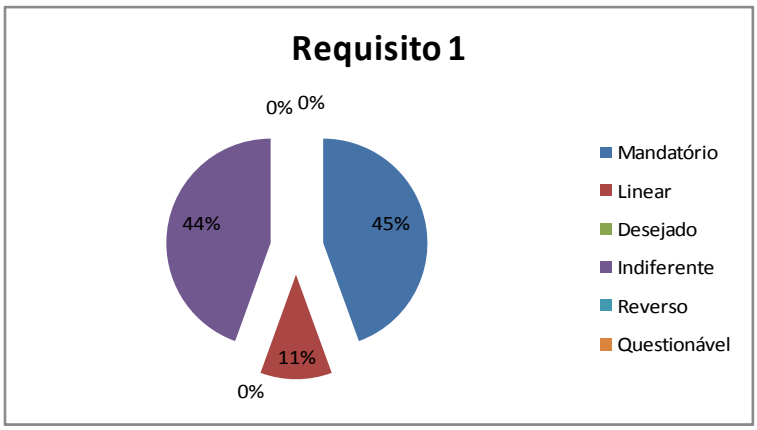


Figura 5.30 Gráfico em Pizza Requisito 1

O requisito 1 com 44% de mandatório e 11% de linear e 44% de indiferente poderia ser implementado pelo alto grau de mandatório.

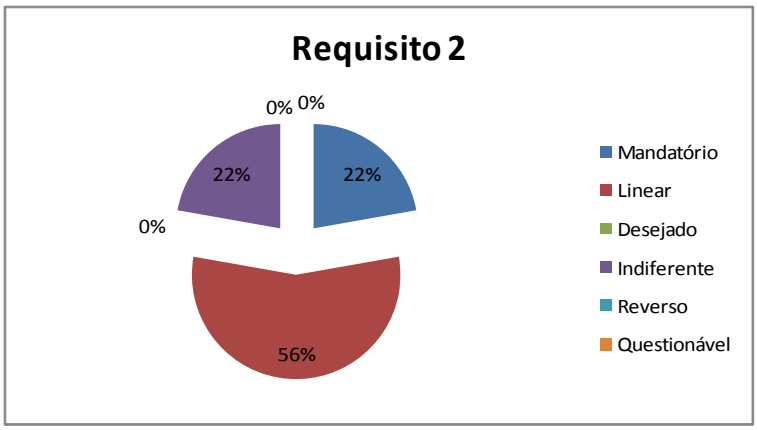


Figura 5.31 Gráfico em Pizza Requisito 2

O requisito 2 tem 55% de Linear e 22% de mandatório e 22% de indiferente. É recomendável que seja feito pelo alto grau de Linear que é possível visualizar.

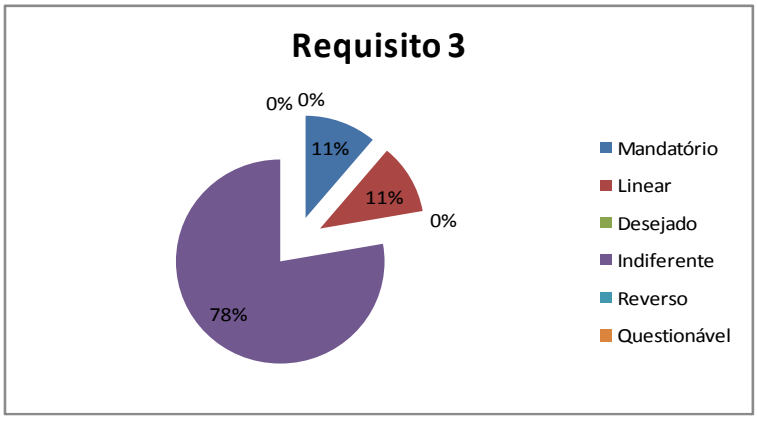


Figura 5.32 Gráfico em Pizza Requisito 3

O requisito 3 tem 78% de indiferente e apenas 11% de linear e mandatório devendo ir para lista de em espera para ser avaliado posteriormente.

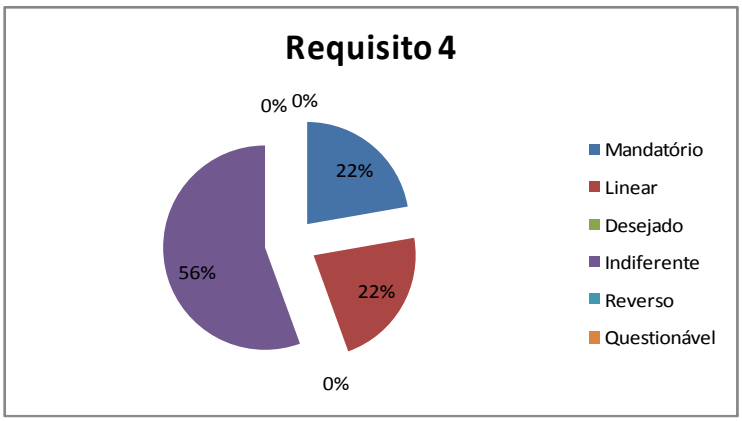


Figura 5.33 Gráfico em Pizza Requisito 4

O requisito 4 tem 55% de indiferente e 22% de Linear e Mandatório, mesmo não sendo tão grave quanto o 3 ele deveria ir para a lista de espera também.

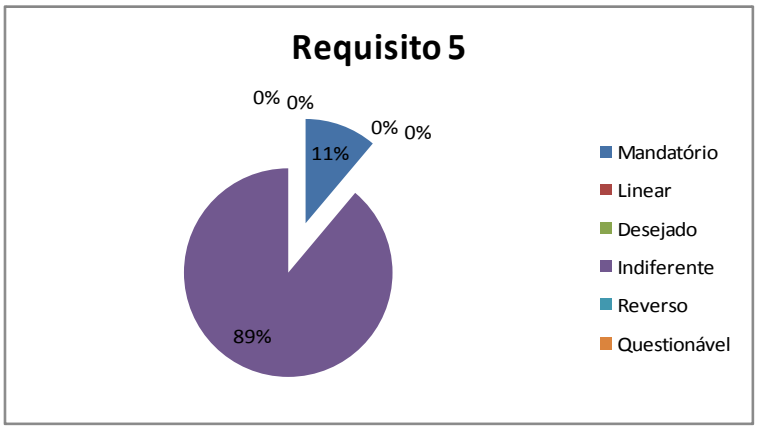


Figura 5.34 Gráfico em Pizza Requisito 5

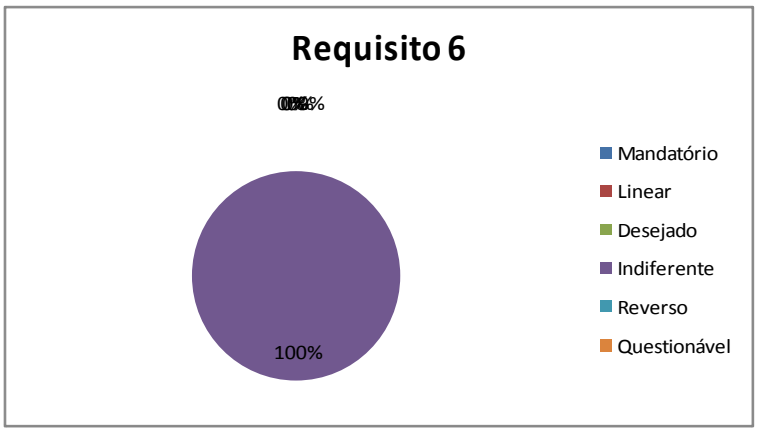


Figura 5.35 Gráfico em Pizza Requisito 6

O requisito 5 teve 89% de Indiferente e 11% de mandatório deve ir pra lista de em espera devendo ser re estudado sua avaliação assim como o principalmente o requisito 6 que teve 100% de Indiferente. Provavelmente não seja o momento certo de se implementar ele e ele possa ser estudado uma forma diferente de implementar-lo.

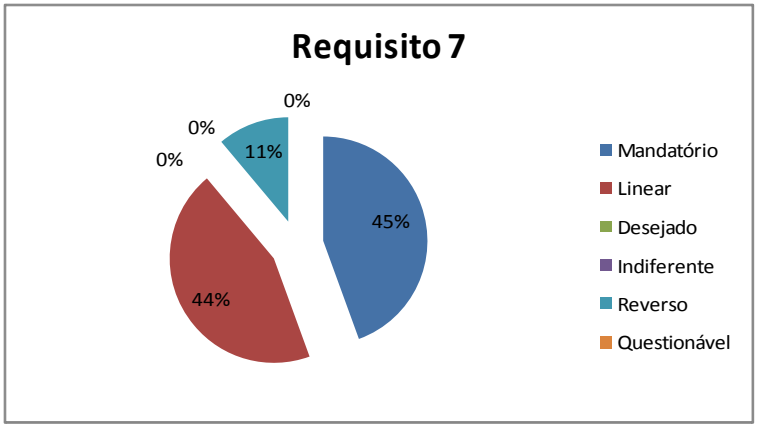


Figura 5.36 Gráfico em Pizza Requisito 7

O requisito 7 deve ser implementado com 44% de mandatório e 44% de linear foi o que obteve a maior satisfação dos usuários mesmo tendo 11% de rejeição que pode ser considerado um caso isolado.

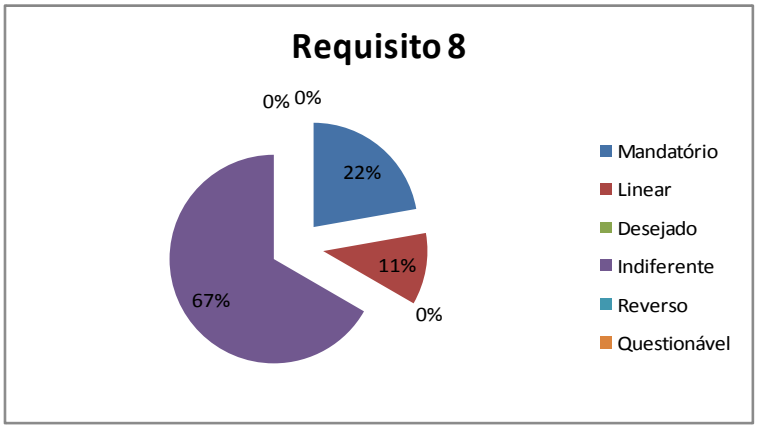


Figura 5.37 Gráfico em Pizza Requisito 8

O requisito 8 tiveram 67% de indiferente, 22% de mandatório e 11 % de linear o tornando outro forte candidato a ir para a lista de espera.

Fase 6: Ranking de requisitos em espera e priorizados

Nessa fase com base em todas as análises realizadas é montada uma tabela para visualização do Ranking dos requisitos.

Tabela 5.12 Sugestão de priorização

Prioridade	Priorizados	Em Espera
1	Requisito 7	Requisito 4

2	Requisito 2	Requisito 8
3	Requisito 1	Requisito 3
4		Requisito 5
5		Requisito 6

A tabela 5.12 foi feita baseada na análise já demonstrada. Lembrando que essa priorização utilizada foi formada única e exclusivamente com o pensamento na satisfação dos usuários que responderam a pesquisa. É prudente levar em conta dados como esforço, necessidade tecnológica ou mesmo dependência dos requisitos antes de gerar a priorização final.

Considerações finais

Durante os estudos de caso muitas dificuldades foram encontradas tais como: convencer os gestores das empresas para aprovar a realização da pesquisa e demora na definição de quais seriam os projetos escolhidos. Foi necessário discutir com os representantes das empresas como seria a melhor forma de aplicar a pesquisa. Também foi necessário avaliar como os usuários poderiam melhor compreender os objetivos da técnica de priorização proposta. O sentimento da importância só acontece com o passar do tempo quando o usuário vê que os requisitos que ele tinha dito como os mais importantes foram justamente os priorizados. De um modo geral, podemos avaliar que o uso da técnica por parte dos usuários foi muito positiva nas duas empresas.

Em conversa com o usuário que ajudou na pesquisa no C.E.S.A.R, foi sugerido, como pontos de melhoria a utilização dos desenhos (*smiles*) que tornam as respostas mais intuitivas. Outra sugestão foi ao invés de adotar papel fornecer uma ferramenta na intranet para realizar essa pesquisa.

Foi destacado que o processo ajudou a priorizar os requisitos da próxima Sprint. E que forçou o Scrum Master a avaliar todo o backlog, uma vez que acontecia de às vezes um requisito ser selecionado sem sequer avaliar todos os outros requisitos que faltavam. Foi dito que essa técnica se adéqua muito bem aos projetos do Cesar que utilizam metodologias ágeis e com uma adaptação pode ser utilizada em outros projetos. Finalmente, foi levantado que isso depende na organização dos gerentes de projeto que são responsáveis por definir o que será implementado dentro do projeto, inclusive a escolha das técnicas de priorização.

Já na Provider o feedback foi mais positivo, uma vez que foi obtido alguns ganhos interessantes com a utilização da técnica. Por exemplo, após o uso da técnica alguns requisitos foram simplesmente removidos do backlog uma vez que 100% dos usuários não viam ganho nenhum nele. Foi destacado que nem estimado esse requisito tinha sido. Esse resultado gerou um grande ganho de tempo na reunião de planejamento. Houve ainda alguns esforços para aplicar a técnica porque foi sentido dificuldades de fazer os usuários responderem os questionários. É muito importante seu resultado até para planejar o que seria mais bem explorado no sistema. Questões relativas à segurança que o PO acreditava ser de extrema importância para o usuário, por exemplo, foi detectado que na verdade não gerava grande satisfação. Com isso foi possível saber qual área da aplicação deve ser utilizada pra a criação de novos requisitos. Ainda foi identificado que seria importante fazer esse estudo de Kano sempre que um requisito novo fosse pensado, para assim só ter o trabalho de estimar os requisitos que já soubessem que existe uma necessidade real do cliente.

Vale salientar que os estudos de caso podem ser considerados um esforço inicial para validar o processo de priorização proposto nesta dissertação. No entanto, é necessária a realização de outros estudos em empresas e projetos com características distintas para comprovar os reais benefícios e adequação do processo proposto nesta pesquisa.

Conclusões e Trabalhos Futuros

Nesse capítulo são apresentadas as contribuições desse trabalho para as empresas participantes na pesquisa. Além disso, são expostas as sugestões de trabalhos futuros, limitações deste trabalho e considerações finais.

Esse trabalho tem como principal contribuição a proposta de um processo de priorização de requisitos para projetos ágeis. O processo de priorização foi baseado na técnica Kano e foi adaptado para as necessidades de projetos ágeis e em destaque para a utilização de Scrum. Foram realizados dois estudos de caso em empresas locais para avaliar a adequação do processo proposto e obter feedback dos participantes da pesquisa com o objetivo de conduzir melhorias futuras.

s Futuros

Dois dos estudos de caso desse trabalho foram realizados em projetos que não tem como produto um software o que pode gerar distorções no resultado da pesquisa. A melhor utilização dessa técnica é quando o projeto tem vários clientes ou usuários e que cada um vive realidades diferentes, pois dificilmente o responsável pelo projeto teria tantas informações para suportar na decisão de priorização. O recomendável seria 20 a 30 usuários o que nos casos pesquisados não seria possível e dificilmente será uma quantidade facilmente atingida. Para atingir isso em projetos onde existe uma grande quantidade de usuários e com um envolvimento maior destacando essa priorização para os usuários é possível alcançar isso. Trazendo um resultado mais confiável a medida que essa quantidade de usuários seja maior. Com relação a trabalhos futuros é sugerido criar uma ferramenta web, realização em outros projetos de software ou não, utilização de técnicas para estimular as pessoas a responder os questionários o mais rápido possível, como por exemplo, prêmios para os usuários

que responderem os questionários ou uma certeza da utilização daquele dado. Um outro trabalho seria a utilização dessa técnica em XP.

A utilização da priorização utilizando Kano nas duas empresas foi muito positiva. As empresas não utilizavam nenhuma técnica para a priorização dos requisitos e através do uso desta técnica tiveram um ganho considerável na qualidade da priorização dos seus requisitos. A partir dessa técnica foi obtida uma forma de envolver os usuários sem deixar o processo pesado.

Esse ganho não se limitou a escolher o que seria feito ou não na próxima Sprint. O ganho foi mais destacado em requisitos que haviam sido planejados ou que se acreditava ser interessante. Através do uso do processo de priorização os usuários mostraram que para eles era indiferente a existência ou não daquele requisito, poupando trabalho de implementação que logicamente é o maior, mas também de estimativa que a equipe teria que fazer para todos os requisitos.

Além disso, foi identificada a possibilidade de se utilizar os resultados obtidos na priorização para saber quais funcionalidades mais atendem as necessidades atuais do cliente, ou seja, os mais importantes.

Referências Bibliográficas

[Abrahamsson 2002] Abrahamsson, Salo, Ronkainen, Warsta Agile Software Development Methods: Review and Analysis, VTT Publications 478, 2002.

[Agile Manifesto] Disponível em http://www.agilemanifesto.org/ Ultimo acesso em 25/10/2008

[Ambriola 1997] Ambriola, V. e Gervasi, V. Processing Natural Language Requirements. 12th International Conference on Automated Software Engineering. Lake Tahoe, EUA Nov. (1997)

[Asosad 2009] 3º ANNUAL SURVEY OF STATE OF AGILE DEVELOPMENT.: disponível em: http://www.versionone.com/pdf/3rdAnnualStateOfAgile_ FullDataReport.pdf . Acesso em : 20/02/2009.

[Astels 2003] ASTELS, David. Test-driven development: a practical guide. 1. ed. Upper Saddle River, NJ: Prentice Hall PTR, 2003.

[Beck 2000] Kent Beck. Extreme Programming explained: embrace change. 1. ed. Reading, MA: Addison-Wesley, 2000.

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

[Beck 2003] BECK, Kent. Test-driven development: by example. 1. ed. Boston: Addison-Wesley, 2003

[Boehm 2003] Boehm, Barry; TURNER, Richard. Balancing agility and discipline: a guide for the perplexed. 1. ed. Boston: Addison-Wesley 2003

[Brooks 1987] Brooks, F., "No Silver Bullet: Essence and Accidents of Software Engineering",Computer, Apr. 1987.

[Brooks 1995] BROOKS, Frederick P. The mythical man-month: essays on software engineering, 20th anniversary edition. 2. ed. Reading, MA: Addison-Wesley, 1995.

[Castro 1996] Castro,J.; Gautreau,J. e Toranzo,M Toll Support for Requirements Formalization. Proceedings of the ACM SIGSOFT Viewpoints 96: International Workshop on Multiple Perspective Software Development – ACM VP 96. San Francisco USA oct. 1996

[Castro 2002] Castro,J.;Kolp,M. e Mylopoulos,J. " Towards Requerements-Driven Information System Engineering: The Tropos Project", Information System, Elsevier, Amsterdam, The Netherlands.

[Charete 2001] Charette, R. "Fair Fight? Agile Versus Heavy Methodologies" Cutter Consortium E-project Management Advisory Service, 2, 13, (2001)

[Clemen 1997] CLEMEN, P. T. Making Hard Decisions: An Introduction to Decision Analysis. Hardcover 2nd edition, 1997.

[Cockburn 2002] Cockburn, Alistair. Agile software development. 1. ed. Boston: Addison-Wesley, 2002.

[Cockburn 2004] Cockburn, Alistair. Crystal Clear : A Human-Powered Methodology for Small Teams (Agile Software Development Series) ed. ADDISON-WESLEY,2004

[Cohn 2004] Cohn, Mike. User Stories Applied: For Agile Software Development. Ed Addison –Wesley, Marco, 2004

[Cohn 2005] Cohn, Mike. Agile estimating and planning. 1 ed. Pearson Education,

2005.

[Cysneiros 2002] Cysneiros, G., 2002. “Ferramenta para o Suporte do Mapeamento da Modelagem Organizacional em i* para UML”. Centro de Informática, Universidade Federal de Pernambuco, Tese de Mestrado, 2002.

[Demarco 2001] DEMARCO, Tom. Slack: getting past burnout, busywork, and the myth of total efficiency. 1. ed. New York: Broadway Books, 2001.

[Eischen 2002] Eischen, Kyle. Software development: an outsider's view. IEEE. Computer, v. 35, n.5, 2002.

[Emam 2003] EMAM, Khaled E. Finding success in small software projects. Cutter Consortium Agile Project Management. Executive Report, v. 4, n. 11, 2003.

[Exp 2009] Disponível em http://www.expertchoice.com último acesso em 24/04/2009

[Finnie 1995] FINNIE, G. R. WITTIG, G. E. e PETKOV, D. I. Prioritizing Software Development Productivity Factors Using the Analytic Hierarchy Process. Journal of Systems and Software, vol 22, 1995.

[Fowler 2000] FOWLER, Martin. Refactoring: improving the design of existing code. Upper Saddle River, NJ: Addison-Wesley, 2000.

[IEEE STD830 1984] IEEE STD. 830. IEEE Guide to Software Requirements Specification. The Institute of Electrical and Electronics Engineers, New York, EUA. (1984)

[Info Escola 2009] Disponível em http://www.infoescola.com/administracao/diagrama-de-kano/

[Jeffries 2001] Jeffries, Ron; Anderson, Ann; Hendrickson, Chet. Extreme Programming installed. 1. ed. Boston: Addison-Wesley, 2001.

[Johnson 2002] Johnson, Jim. "ROI, It's your job". Published Keynote Third International Conference on Extreme Progrmming, Alghero, Italy, Maio 26-29, 2002.

[Goguen, 1993] Goguen, J.; Linde, C. Techniques for Requirements Elicitation. First International Symposium on Requirements Engineering, IEEE Computer Society Press, 1993.

[Grady 1999] R. Grady. An Economic Release Decision Model: Insights into Software Project Management. 1999

[Hauser 1988] HAUSER, J. e CLAUSING, D. The House of Quality. The Harvard Business Review, vol 3. 1988.

[Hong 1981] HONG, S. e NIGAM, R. Analytic Hierarchy Process Applied to Evaluation of Financial Modeling Software. Proceedings of the 1st International Conference on Decision Support Systems. Atlanta, EUA. 1981.

[Herzberg 2009] Frederickhttp://pt.wikipedia.org/wiki/Frederick_Herzberg. Disponível em: http://pt.wikipedia.org/wiki/Frederick_Herzberg. Acesso Abril 2009.

[Hunt 2003] HUNT, Andrew; THOMAS, David. Pragmatic unit testing: in Java with JUnit. 1. ed: The Pragmatic Programmers, 2003.

[Highsmith 2002] James A. Highsmith. Agile software development ecosystems. 1. ed. Boston: Addison-Wesley, 2002

[Karlsson 1997] Karlsson, J. Ryan, K. (1997), “A cost-value approach for prioritizing requirements”, In: IEEE Software.

[Karlsson 1998] KARLSSON, K. A Systematic Approach for Prioritizing Software Requirements.
Department of Computer and Information Science, Linkoping University, Suécia. 1998.

Referencias Bibliográficas

[Karlsson 2003] Karlsson, L. Improving Requirements Selection Quality in Market-Driven Software Development. PhD Thesis. Department of Communication System, Lund Institute of Technology (2003)

[Kontio 1996] KONTIO, J. CALDIERA, G. e BASILI, V. Defining Factors, Goals and Criteria for Reusable Component Evaluation. CASCON'96. Nov. 1996.

[Kotonya 1997] Kotonya, G e Sommerville, I. Requirements Engineering – Processes and Techniques. John Willy & Sons. (1997)

[Kujala 2005] Kujala,Sári; Kauppinen, Marjo; Lehtola,Laura; Kojo,Tero (2005). "The Role of User Involvement in Requirements Quality and Project Success". Proceeding of the 2005 13th IEEE International Conference on Requirements Engineering(RE'05)

[Kunda 1999] KUNDA, D. e BROOKS, L. Applying Social -Technical Approach for COTS Selection. Proceedings of the 4th UKAIS Conference, University of York, Apr.1999.

[Lamsweerde 2000] LAMSWEERDE, A. Requirements Engineering in the Year 00: A Research Perspective. 22nd Proceedings of International Conference on Software Engineering. Limerick, Ireland. Jun. 2000.

[Lawsweerde 1998] Lawsweerde, A.; Darimont, R. e Letier, E. Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Transaction on Software Engineering. Special Issue on Managing Inconsistence in Software Development (1998)

[Loucopoulos 1995] Loucopoulos, P. E Karakostas, V. System Requirements Engineering. McGraw-Hill Book Company. 1995

[Marçal 2007] Ana Sofia Cysneiros, Marçal Bruno Celso Cunha de Freitas, Felipe Santana Furtado Soares, Teresa Maria Medeiros Maciel,Arnaldo Dias Belchior . Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI. CLEI 2007: XXXIII Conferencia Latinoamericana de Informática, San Jose, Costa Rica.

[Melo 2008] Melo, Ricardo Alves de. "Um Estudo Qualitativo do Perfil de Elicitador de Requisitos Orientado a Obtenção do Sucesso de Projetos de Software" Dissertação Mestrado, UFPE, 2008.

[Min 1992] MIN, H. Selection of Software: The Analytic Hierarchy Process. International Journal of Physical Distribution and Logistics Management, vol 22, 1992.

[Ncube 2000] NCUBE, C. A Requirements Engineering Method for COTS-Based Systems Development. PhD thesis . School of Informatics, City University, London. Mai. 2000.

[Nuseibeh 1997] NUSEIBEH, B. Ariane 5: Who Dunnit? IEEE Software, Mai./Jun. 1997.

[Palmer 2002] Palmer, Stephen R. e Felsing, John M, A Practical Guide to Feature-Driven Development,ed. Prentice Hall PTR ,2002

[Patricio 2004] Patricio, L , Cunha, JF, Fisk, RP, e Nunes, NJ. Customer Experience Requirement for Multi-Platform Service Interaction: Bringing Services Marketing to the Elicitation of User Requirements. Proceeding of 12th IEEE International Requirements Engineering Conference. set 2004

[Pereira 2007] Pereira, S. C. Um Estudo Empírico sobre Engenharia de Requisitos em Empresas de Produtos de Software. Dissertação de Mestrado em Ciência da Computação. Centro de Informática, Universidade Federal de Pernambuco 2007

[Pirâmide 2009] Disponível em http://www.brandme.com.br/publico-alvo/ . Acesso em Abril de 2009

[Poppendieck 2003] POPPENDIECK, Mary; POPPENDIECK, Tom. Lean software development: an agile toolkit. 1. ed. Upper Saddle River, NJ: Addison-Wesley, 2003

[Pressman 2001] Pressman, R. "Engenharia de Software" McGraw-Hill, (2001)

[Rising 2007] Rising, L. The Scrum Software Development Process. Disponível

em: http://members.cox.net/risingl1/articles/IEEEScrum.pdf. Acesso em março 2007.

[Robertson 2006] Robertson, Suzanne; Robertson, James. Customer Satisfaction/dissatisfaction – Mastering the Requirements Process (Volere) 2 ed, 2006.

[Rup 2002] Rational Unified Process Tutorial. Versão 2002 05 00.

[Ryan, 1993] Ryan,M. Defaults in Specification. Proceedings of IEEE International Symposium on Requirements Engineering. IEEE Computer Society Press. San Diego,EUA. Jan 1993

[Saaty 1990] SAATY, T. The Analytic Hierarchy Process. New York: McGraw-Hill, 1990

[Schwaber 2001] Ken Schwaber: Agile Software Development with Scrum. Prentice Hall (2001).

[Schwaber 2004] Ken Schwaber. "Agile Project Management with Scrum" Ed. Microsoft Press 2004.

[Sommerville 2007] Sommerville, I. (2007), Engenharia de Software, Pearson Addison-Wesley, 8a. edição. .

[Standish 2001] THE STANDISH GROUP INTERNATIONAL, Inc. Extreme chaos. The Standish Group International, Inc, 2001. Disponível em: http://www.standishgroup.com/sample_research/PDFpages/extreme_chaos .pdf. Acesso em: 23/12/2004.

[Stephens 2003] Matt Stephens, Doug Rosenberg,Extreme Programming Refactored: The Case Against XP. 2003 Ed. Softcover.

[Teles 2004] Teles, Vinícius Manhães. Extreme Programming: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. 1. ed. São Paulo: Novatec, 2004.

[Teles 2005] Teles, Vinícius Manhães, "Um estudo de caso da adoção das práticas e valores do extreme programming", IM-NCE/UFRJ 2005.

[Thayer 1997] Thayer, R. H. e Dorfman, M.; "Introduction to Tutorial Software Requirements Engineering" in Software Requirements Engineering, IEEE-CS Press, 2 . ed. 1997.

[Toranzo 1999] Toranzo, M. e Castro,J. A Comprehensive Traceability Model to Support the Design of Interative System. International Workshop on Interative System Development and Object Models – WISDOM99. Lisboa, Portugam. Jun (1999)

[Versionone 2009] Versionone: HTTP://www.versionone.com/ Acesso em 20/02/2009).

[Weinberg 1971] Gerald M. Weinberg, The psychology of computer programming. 1. ed. New York: Van Nostrand Heinhold Company, 1971. 288 p.

[Williams 2003] WILLIAMS, Laurie; KESSLER, Robert. Pair programming illuminated. 1. ed. Boston, MA: Addison-Wesley, 2003.

[Yourdon 2004] YOURDON, Edward. Death march. 2. ed. Upper Saddle River, NJ: Prentice Hall PTR,2004.

Apêndices

Nos apêndices são apresentados vários artefatos usados no processo de priorização proposto e nos estudos de caso realizados.

2.5 Apêndice A

Nesse apêndice são mostradas as perguntas que foram guias para as entrevistas sobre os resultados dos estudos de caso.

1- Como foi a aplicação da técnica?
2- Quais foram às dificuldades encontradas na implantação do Kano?
3- Qual foi o ganho na aplicação da técnica?
4- A técnica será aplicada em outros projetos?
5- O que poderia ser feito para melhorá-la?
6- A técnica se aplica a todos os projetos da sua organização?
7- Como foi a confiança do PO na utilização da técnica?
8- O que foi identificado com o uso da técnica que não era possível antes durante a priorização?

2.6 Apêndice B

Neste apêndice é mostrado o questionário utilizado pela pesquisa realizada na Provider. Onde foi impresso em papel para que os usuários respondessem a pesquisa.

Questionário Kano

Lembrando que essa pesquisa tem a intenção de verificar qual seu sentimento em relação aos requisitos. Ninguém será julgado ou punido por nenhuma resposta.

1. Se a próxima versão do **SOFTWARE** incluir o requisito X, como você se sentirá?

Muito Satisfeito | Satisfeito | Pouco Satisfeito | Indiferente | Insatisfeito

Tabela 8.13 Resposta dos usuários

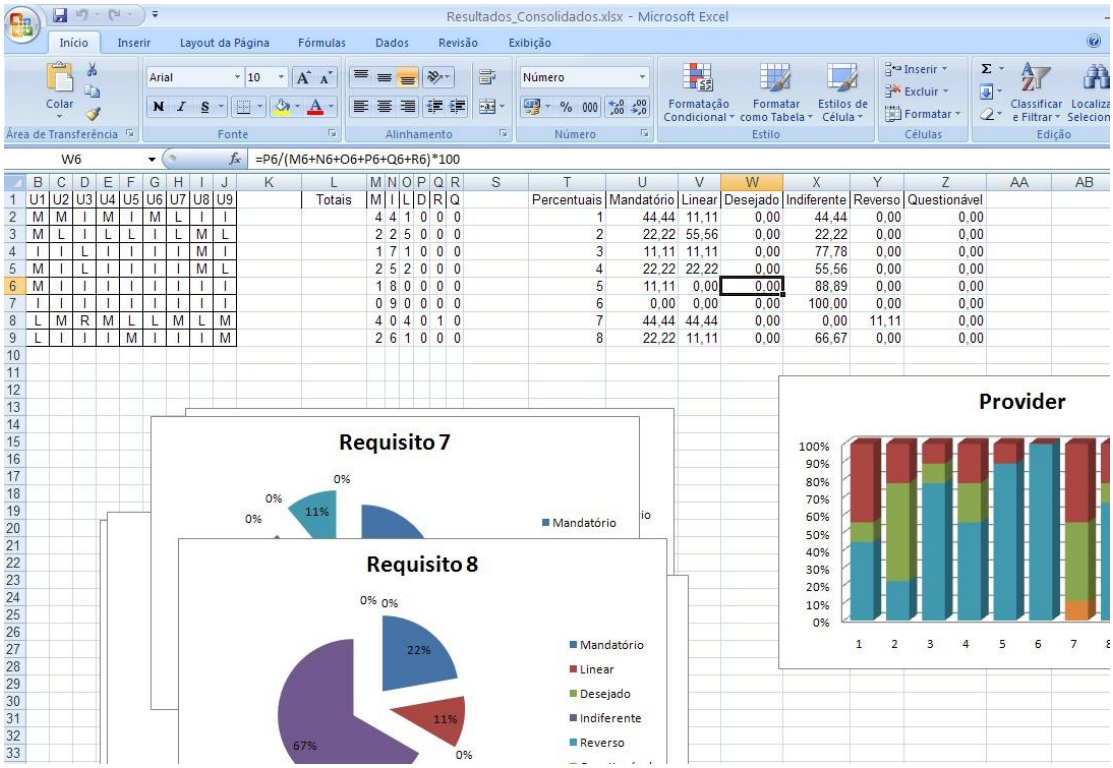
2. Se a próxima versão do **SOFTWARE NÃO** incluir o requisito X, como você se sentirá?

Muito Satisfeito | Satisfeito | Pouco Satisfeito | Indiferente | Insatisfeito

Tabela 8.14 Resposta dos usuários

2.7 Apêndice C

Esse apêndice mostra como foi utilizado o Excel para facilitar na consolidação dos resultados. Abaixo é possível ver a planilha com os resultados do estudo de caso feito na Provider.



	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1	U1	U2	U3	U4	U5	U6	U7	U8	U9		Totais	M	I	L	D	R	Q		Percentuais	Mandatório	Linear	Desejado	Indiferente	Reverso	Questionável		
2	M	M	I	M	I	M	L	I	I			4	4	1	0	0	0		1	44,44	11,11	0,00	44,44	0,00	0,00		
3	M	L	I	L	L	I	L	M	L			2	2	5	0	0	0		2	22,22	55,56	0,00	22,22	0,00	0,00		
4	I	I	L	I	I	I	I	M	I			1	7	1	0	0	0		3	11,11	11,11	0,00	77,78	0,00	0,00		
5	M	I	L	I	I	I	I	M	L			2	5	2	0	0	0		4	22,22	22,22	0,00	55,56	0,00	0,00		
6	M	I	I	I	I	I	I	I	I			1	8	0	0	0	0		5	11,11	0,00	0,00	88,89	0,00	0,00		
7	I	I	I	I	I	I	I	I	I			0	9	0	0	0	0		6	0,00	0,00	0,00	100,00	0,00	0,00		
8	L	M	R	M	L	L	M	L	M			4	0	4	0	1	0		7	44,44	44,44	0,00	0,00	11,11	0,00		
9	L	I	I	I	M	I	I	I	M			2	6	1	0	0	0		8	22,22	11,11	0,00	66,67	0,00	0,00		

Figura 8.38 Consolidação dos dados usando o Excel

Foi criada uma planilha com os dados das respostas e a partir daqui foram utilizados esses dados para chegar às outras planilhas, e conseqüentemente nos gráficos. A primeira planilha mostrada na figura 8.1 no alto do lado esquerdo refere-se à resposta de cada usuário por cada requisito. A tabela do meio dos totais foi utilizada a Formula SE do Excel para facilitar o somatório. Um exemplo da formula foi:

=SE(B2="M";1;0)+SE(C2="M";1;0)+SE(D2="M";1;0)+
+SE(E2="M";1;0)+SE(F2="M";1;0)+SE(G2="M";1;0)+SE(H2="M";1;0)+SE(I2="M";1;0)+SE(J2="M";1;0)

Esse foi o somatório para os requisitos mandatórios por isso o “M” na formula.

Apêndices

A tabela da direita foi utilizada a fórmula:
=N6/(M6+N6+O6+P6+Q6+R6)*100
Que divide total daquele item que havia sido calculado na outra formula pelo total de respostas e multiplica por 100 obtendo assim o valor percentual de cada item.

Após esses passos é apenas utilizada a última tabela de referência para criar o gráfico da direita e os gráficos por requisito de acordo com os gráficos do Excel : Colunas 3D 100% empilhadas para todos os requisitos e Pizza destacada para os requisitos individualmente.