Iniciação Científica **CESUMAR**
Jul./Dez. 2007, v. 09, n.02, p. 143-151

ESTUDO DAS METODOLOGIAS DE ESPECIFICAÇÃO DE REQUISITOS E SUA ABORDAGEM QUANTO AOS REQUISITOS NÃO-FUNCIONAIS DE SISTEMAS

Everton Luckesi de Souza*
Wilson Bissi**
Márcia Cristina Dadalto Pascutti***

RESUMO: A Engenharia de *Software* é uma abordagem para o desenvolvimento padronizado de *software* de qualquer natureza. Ela define um conjunto de metodologias, técnicas e ferramentas que, quando bem empregadas, auxiliam efetivamente no processo de desenvolvimento de *softwar*es. Uma das maiores dificuldades da especificação de requisitos é a obtenção dos requisitos não funcionais do sistema, que dizem respeito a desempenho, *performance*, usabilidade, manutenibilidade, portabilidade. Dessa forma, o objetivo deste trabalho é estudar as metodologias que abordam os requisitos não funcionais do sistema e metodologias de desenvolvimento ágil. Para isso se farão análises e comparações entre elas, demonstrando suas principais características, com o intuito de obter maior conhecimento nesta área, tornando-a mais difundida na comunidade, de modo que futuramente sirva como base para outros artigos. Neste trabalho as metodologias de desenvolvimento de *software* adotadas para estudos foram a MERUSA, XP (*Extreme Programming)* e SCRUM, das quais as duas últimas são consideradas metodologia de desenvolvimento ágil. Os resultados obtidos são de grande valia, pois é escasso o material que aborda essas metodologias, principalmente sobre a MERUSA, que trata de forma muito eficaz os requisitos não funcionais de sistema, que muitas vezes são omitidos em outras metodologias.

PALAVRAS-CHAVES: Metodologias de desenvolvimento de *software*; Requisitos não- Funcionais; Sistemas de informação; Engenharia de *software*.

STUDY ABOUT METHODOLOGIES OF SPECIFICATION OF REQUIREMENTS AND THEIR APPROACH REFERING TO NON-FUNCTIONAL REQUIREMENTS OF SYSTEMS

ABSTRACT: The Engineering Software is an approaching for the standardized development of software of any nature. Defines a set of methodologies, techniques and tools that, when well employed aid effectively the process of software development. One of the biggest difficulties of the specification of requirements is the acquirement of the non-functional requirements of the system, the ones that is about accomplishment, performance, and usability, maintainability and portability. Inside of this context, the aim of this work is to study the methodologies that approach the requirements non-functional of the system and methodologies of agile development. Thus, we'll have analysis and comparisons between them, demonstrating its main characteristics, intending to get greater knowledge in this area, turning it spread out in the community, in a way that in the future may serves as support base for other articles. In this study, the methodologies of development of software adopted for studies had been the MERUSA (Methodology of Specification of Requirements of Usability and Safety Guided for the Architecture of the System), XP (*Extreme Programming)* and SCRUM, which the last two mentioned are considered agile methodologies. The obtained results are of great value, therefore the material that approaches these methodologies is scarce, mainly about MERUSA, that deals in an efficient form the requirements non-functional of the system, which many times are omitted in other methodologies.

KEYWORDS: Methodologies of development of software; Requirements non-functional; Systems of information; Engineering software.

INTRODUÇÃO

O desenvolvimento de *software* impõe cada dia mais a necessidade de se trabalhar com requisitos não funcionais, os quais dizem respeito a aspectos de segurança, definição de padrões, confiabilidade do sistema, portabilidade do sistema, *performance* e manutenção. Várias metodologias de desenvolvimento abordam esse tema, mas a dificuldade é grande em se distinguir qual a melhor a ser utilizada.

* Acadêmico do curso de Processamento de Dados do Centro Universitário de Maringá – CESUMAR. E-mail: evertonluckesi@hotmail.com
** Acadêmico do curso de Processamento de Dados do Centro Universitário de Maringá – CESUMAR. E-mail: wbissi@gmail.com
*** Docente do curso de Processamento de Dados do Centro Universitário de Maringá – CESUMAR. E-mail: pascutti@cesumar.br

Este trabalho aborda três metodologias: MERUSA (Metodologia de Especificação de Requisitos de Usabilidade e Segurança orientada para a Arquitetura do sistema), XP (*Extreme Programming*) e SCRUM. Foram traçados parâmetros entre as metodologias, levantando como cada uma trata os requisitos não funcionais e a suas formas de aplicação.

Devido ao grande número de metodologias existente e ao grau de complexidade de cada uma delas, este trabalho descreve, de maneira clara, as três metodologias citadas e como cada uma delas enfoca os requisitos não funcionais no desenvolvimento de *software*. Uma tabela demonstrativa facilita ao leitor o seu entendimento e permite uma análise clara de suas informações.

Foi traçada uma metodologia de desenvolvimento que consiste em levantamentos bibliográficos, teórico-metodológicos e técnicos para o desenvolvimento e conclusão deste trabalho.

Cada uma das três metodologias foi descrita detalhadamente, mostrando-se seus pontos fortes e fracos, e com elas, a forma de tratar os requisitos não funcionais. Este trabalho espera contribuir para a disseminação do assunto, que é difícil encontrar em nosso meio.

2 ENGENHARIA DE *SOFTWARE*

A Engenharia de *Software* (ES) surgiu em meados dos anos 70, numa tentativa de contornar a crise do *software* e dar um tratamento de engenharia (mais sistemático e controlado) ao desenvolvimento de sistemas de *software* complexos. Um sistema de *software* complexo se caracteriza por um conjunto de componentes abstratos de *software* (estrutura de dados e algoritmos), encapsulados na forma de procedimentos, funções, módulos, objetos ou agentes interconectados entre si, componentes da arquitetura do *software,* que deverão ser executados em sistemas computacionais (SOMMERVILLE, 2003).

De acordo com Pressman (1995), embora muitas definições abran-gentes tenham sido propostas para a ES, todas elas reforçam a exigência da disciplina de engenharia no desenvolvimento de *software*, abrangendo um conjunto de três elementos fundamentais: métodos, ferramentas e procedimentos. Os métodos detalham "como fazer" para se construir o *software*, as ferramentas proporcionam aos métodos apoio automatizado ou semi-automatizado, e os procedimentos constituem o elo de ligação que mantém juntos os métodos e as suas ferramentas e possibilita um processo de desenvolvimento claro, eficiente, visando garantir ao desenvolvedor e seus clientes a produção de um *software* de qualidade.

A engenharia de software se concentra nos aspectos práticos da produção de um sistema de *software*, enquanto a ciência da computação estuda os fundamentos teóricos. Os fundamentos científicos da engenharia de *software* envolvem o uso de modelos abstratos e precisos, que permitam ao engenheiro especificar, projetar, implementar e manter sistemas de *software*, avaliando e garantido suas qualidades. Além disto, a enge-nharia de *software* deve oferecer mecanismos para se planejar e gerenciar o processo de desenvolvimento. Empresas desenvolvedoras de *software* passaram a empregar os conceitos de engenharia de *software*, sobretudo, para orientar suas áreas de desenvolvimento, muitas delas organizadas sob a forma de fábrica de *software* (SOMMERVILLE, 2003).

Um modelo de processo de *software*, ou simplesmente modelo de processo, pode ser visto como uma representação ou abstração dos objetos e atividades nele envolvidos. Oferece uma forma mais abrangente e fácil de representar o gerenciamento de processo de *software* e, conseqüentemente, o progresso do projeto - por exemplo, Cascata, Evolucional, Espiral, RAD e outros.

Segundo a Enciclopédia Livre Wikipédia, metodologia envolve princípios filosóficos que guiam uma gama de métodos que utilizam ferramentas e práticas diferenciadas para realizar algo.São exemplos de metodologias da ES: metodologias ágeis (*Scrum, Crystal*), programação extrema (XP), *Rational Unified Process* (RUP), *Microsoft Solution Framework* (MSF), etc.

Este trabalho tem o foco voltado para as metodologias de desenvolvimento de *software*, mais especificamente para as metodologias Scrum, XP e MERUSA, que serão abordadas logo em seguida, a começar pela MERUSA.

2.1 REQUISITOS NÃO-FUNCIONAIS (RNF)

A necessidade de se trabalhar com requisitos não funcionais torna-se fator de importância cada vez maior no desenvolvimento de software. Ao contrário dos requisitos funcionais, os não funcionais não expressam nenhuma função (transformação) a ser implementada em um sistema de informações, eles expressam condições de comportamento e restrições que devem prevalecer.

> O grande desafio da análise de requisitos não-funcio-nais é a possibilidade de vencer os impactos causados por restrições operacionais ao software e permitir que ele seja desenvolvido em tempo hábil. Casos comprovados mostram que a falta de um tratamento adequado dos requisitos não- funcionais pode levar a resultados desastrosos (CYSNEIROS, 1998).

Os requisitos não funcionais são aqueles que dizem respeito a aspectos de segurança, definição de padrões, confiabilidade do sistema, portabilidade do sistema, *performance* e manutenção.

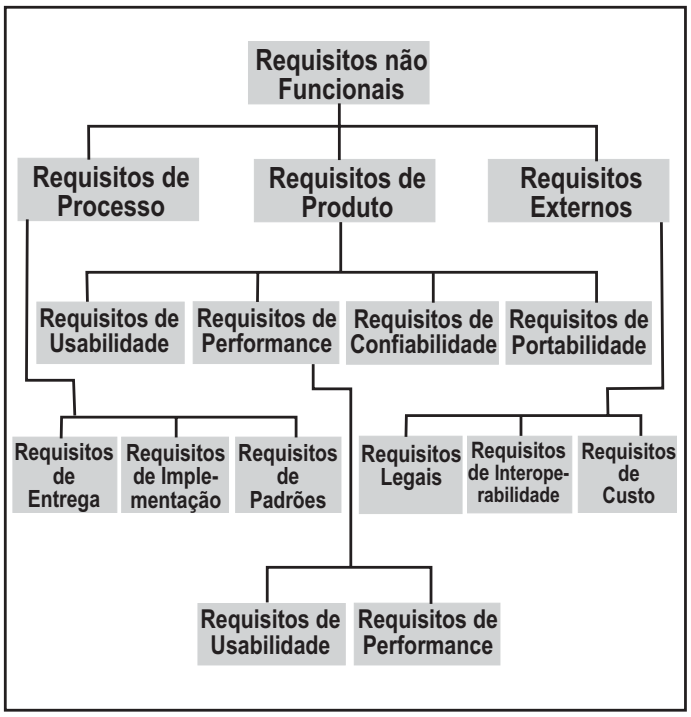

Figura 1: Classificação dos RNFs segundo SOMMERVILLE
Fonte: Cysneiros (2001)

Como se pode observar na Figura 1, existem diversos tipos de RNF, cada um dos quais deve ter a devida atenção na hora da especificação. Fontes históricas têm confirmado que o problema de definição de requisitos atinge grande quantidade de sistemas. Um levantamento de cerca de 8.000 projetos, em 350 companhias norte-americanas, revelou que cerca de um terço destes projetos não foi completado ou o foi com sucesso parcial, apresentando funcionalidade parcial, custo excessivo ou atrasos significativos. Essa pesquisa mostrou que 47% dos executivos responsáveis por esses projetos atribuíram essas falhas a problemas com requisitos, mais especificamente: falta de envolvimento do usuário (13%), requisitos incompletos (12%), mudança de requisitos (11%), expectativas não realistas (6%) e objetivos mal-definidos (5%) (STANDISH, 1995).

Dentre os requisitos demonstrados na Figura 1, os que serão mais abordados por este trabalho são os de usabilidade, segurança e confiabilidade. Os requisitos de usabilidade dizem respeito a: interface simples; documentação completa e consistente; mensagens de erros precisas e construtivas; ajuda completa; facilidade de instalação, etc. Por sua vez, os requisitos de segurança demonstram: integridade (o sistema deve manter os dados sempre consistentes); completude (o sistema deve trabalhar sempre com todas as informações necessárias, não permitindo a entrada de dados incompletos pelo usuário); precisão (o sistema não pode fornecer informações erradas); confidencialidade (as informações geradas e mantidas pelo sistema somente estarão disponíveis para os usuários autorizados).

3 MERUSA (METODOLOGIA DE ESPECIFICAÇÃO DE REQUISITOS DE USABILIDADE E SEGURANÇA ORIENTADA PARA A ARQUITETURA DO SISTEMA)

A MERUSA (Metodologia de Especificação de Requisitos de Usabilidade e Segurança orientada para a Arquitetura do sistema) foi desenvolvida por Valter Fernandes Avelino e apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Doutor em Engenharia, no ano de 2005. A MERUSA tem por objetivo, além do desenvolvimento de uma nova metodologia, demonstrar a relevância da elicitação de requisitos, com foco nos requisitos não-funcionais do sistema. Concentra-se no processo inicial do desenvolvimento do sistema, abrangendo a identificação dos requisitos de usabilidade e segurança e dos critérios para sua análise e validação, em relação à arquitetura dos sistemas (AVELINO, 2005).

3.1. DEFINIÇÃO

Os princípios básicos da MERUSA são os mesmos adotados para as metodologias relativas aos requisitos de segurança e de usabilidade, porém abaixo serão mostrados somente os princípios que dizem respeito a este trabalho, segundo Avelino (2005):

- processo de elicitação e análise dos requisitos, considerando um ciclo de desenvolvimento em espiral, centrado no desenvolvimento da arquitetura do sistema, com refinamentos sucessivos e análise de custo/benefício baseada em análises quantitativas e qualitativas;
- atividade iterativa de elicitação de requisitos ao longo do processo de desenvolvimento, considerando a sincronização sistemática dos diver-sos tipos de requisitos funcionais e não funcionais, que têm um ciclo de vida próprio, para identificação de eventuais conflitos entre estes;
- modelagem dos requisitos de usabilidade e segurança elicitados, utilizando- se os modelos de requisitos por objetivos (GRL– *Goal-ori-ented Requirement Language*), juntamente com os mapas de casos de uso (UCM – *Use Case Maps*), para representar os requisitos funcionais e os cenários causais, de modo a permitir uma verificação cruzada entre objetivos e

cenários, visando à melhoria da completeza e robustez do processo de elicitação de requisitos como um todo;

• atividade de avaliação da satisfação de cada tipo de RNF, de modo individual, realizada por especialistas; essa atividade é independente do processo principal de avaliação de requisitos funcionais e utiliza métodos de avaliação próprios para cada tipo de RNF, de modo que o especialista de cada área possa maximizar a cobertura da sua análise específica;

• atividade de análise integrada de todos os requisitos, baseada na verificação da sensibilidade dos requisitos às mudanças de arquitetura, utilizando métodos de avaliação específicos para cada tipo de RNF; a análise é realizada por especialistas, mas considerando as possíveis interferências dos demais requisitos nessa análise (análise de risco); dessa forma, fornece diretivas para a comparação entre opções de arquitetura em relação ao atendimento de todos os requisitos em conjunto, bem como para o processo de análise de eventuais compromissos entre requisitos;

• atividade de análise de compromissos baseada nas análises de risco, processo no qual eventuais conflitos entre requisitos são verificados; como resultado dessa crítica, os modelos de arquitetura podem ser refinados, modificados, revisados, ou eventualmente alguma mudança nos requisitos pode ser negociada em função do balanço custo/risco/benefício, resolvendo eventuais compromissos e levando ao refinamento sucessivo desses requisitos, juntamente com a definição da arquitetura do sistema.

Especialistas participam em conjunto das atividades relacionadas com a tomada das principais decisões do projeto, tanto nas atividades iniciais de cada ciclo iterativo (elicitação de requisitos). como nas atividades finais de cada iteração (análise integrada e análise de compromissos). Desse modo, facilita-se a comunicação e o entendimento entre as várias áreas envolvidas com o projeto, maximizando o grau de cobertura de cada ferramenta de análise, promovendo a análise da inter-relação entre os requisitos para identificação dos compromissos e garantindo a consistência do conjunto da especificação de requisitos.

As etapas de desenvolvimento da MERUSA são aplicadas por duas frentes (técnicas de avaliação de segurança e de usabilidade), verificando sua adequação através do modelo RM-ODP (*Reference Model Of Open Distributed Processing*) ou modelo de referência para o processamento distribuído e aberto. Após esta etapa é realizada a especificação dos respectivos requisitos e constituída a metodologia. Para a avaliação da MERUSA é utilizada a Metaarquitetura do SIN (Sistema Interligado Nacional), conforme representa a Figura 2.

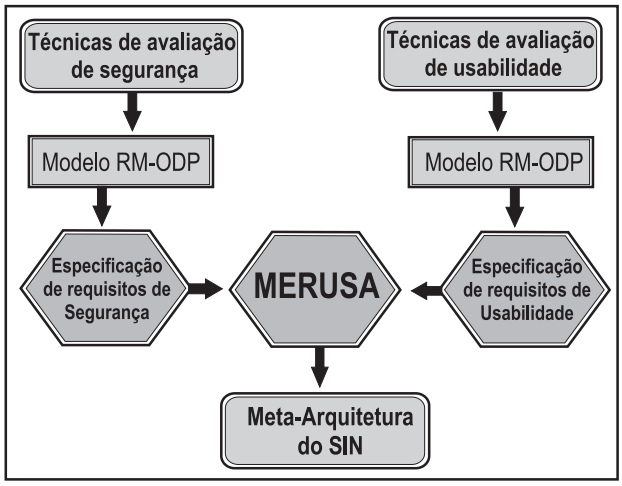

Figura 2: Etapas de desenvolvimento da MERUSA
Fonte: Avelino (2005)

3.2 FASES

De modo análogo aos processos descritos acima, a MERUSA pode ser dividida em quatro fases principais, como mostra a Figura 3. São elas:

• FASE I: atividade de identificação de cenários e elicitação de requisitos do domínio (segundo cada uma das cinco visões ODP): em paralelo, para cada tipo de RNF, existe a identificação das características específicas do ambiente (identificação dos perigos para segurança, identificação do usuário para usabilidade), bem como a identificação dos objetivos e cenários específicos de cada tipo de RNF;

• FASE II - atividade de descrição da arquitetura e construção dos cenários (baseados em casos de uso): em paralelo, para cada tipo de RNF, realizam- se as atividades de elicitação dos requisitos específicos (referentes a cada uma das cinco visões do modelo ODP) e o registro das diretivas de projeto e de análise específicas (construção dos cenários de segurança, elaboração do guia de estilo de usabilidade, reengenharia do trabalho);

• FASE III - atividade de construção do modelo das arquiteturas e avaliação individualizada da sensibilidade dos seus requisitos (determinação da sensibilidade de cada tipo de requisito em relação às alternativas de arquitetura). Em paralelo, para cada tipo de RNF realizam-se as atividades de modelagem, prototipação e análise específicas (identificação e análise de riscos, análise de segurança, modelagem e prototipação da IHC (Interface Homem Computador), análise de usabilidade para cada ponto de vista da arquitetura;

• FASE IV - atividade de análise integrada de todos os tipos de requisitos para a identificação das caraterísticas conflitantes e a subseqüente análise dos compromissos e resolução de conflitos:

nessa fase os grupos de análise de cada RNF específico realizam uma análise conjunta para verificação das interferências entre os diferentes modelos utilizados nas análises, visando à identificação de potenciais conflitos; a solução desses conflitos ou a identificação de requisitos importantes não atendidos podem levar à modificação dos requisitos originais de cada ponto de vista, à criação de novos requisitos ou à necessidade de modificações na arquitetura do sistema, tendo como conseqüência a necessidade de nova iteração do ciclo de elicitação e análise de requisitos, até que seja verificada a adequação de todos os RNF's do sistema.

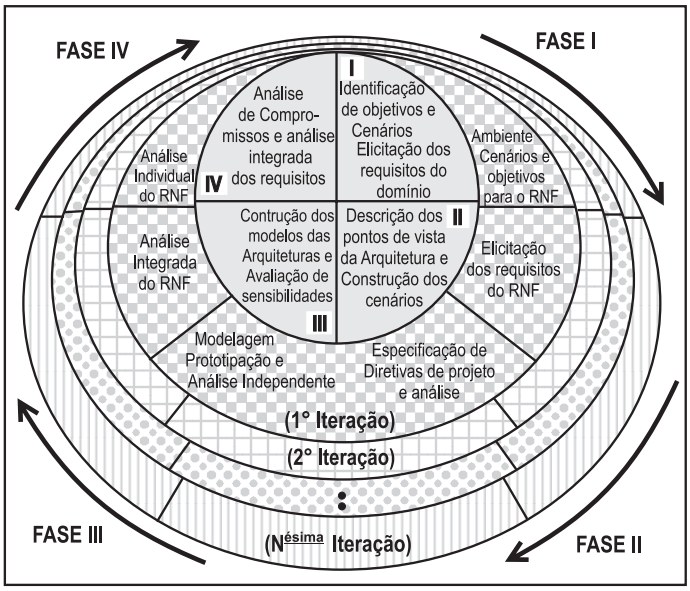

Figura 3: Etapas da MERUSA considerando vários ciclos de iteração
Fonte: AVELINO(2005)

4 METODOLOGIA SCRUM

Ken Schwaber e Mike Beedle desenvolveram esta metodologia na década de 90, baseando-se em sua própria experiência no desenvolvimento de sistemas e processos. O Scrum assume-se como uma metodologia extremamente ágil e flexível. Tem por objetivo definir um processo de desenvolvimento iterativo e incremental que pode ser aplicado a qualquer produto ou no gerenciamento de qualquer atividade complexa. Baseia-se no desenvolvimento incremental das aplicações centrado na equipe, permitindo um controle maior do processo, pelo fato de ter ciclos de iteração muito curtos (FERREIRA et al., 2005).

Scrum aplica-se a projetos tanto pequenos como grandes. Esforçando-se para liberar o processo de quaisquer barreiras, o seu principal objetivo é conseguir uma avaliação correta do ambiente em evolução, adaptando-se constantemente ao caos de interesses e necessidades.

A Metodologia Scrum apenas estabelece conjuntos de regras e práticas de gestão que devem ser adotadas para garantir o sucesso de um projeto. Centrada no trabalho em equipe, melhora a comunicação e maximiza a cooperação, permitindo que cada um faça melhor o seu e se sinta bem com o que faz, o que mais tarde se reflete num aumento de produtividade. Englobando processos de engenharia, este método não requer nem fornece qualquer técnica ou método específico para a fase de desenvolvimento de *software*.

Segundo Ferreira e colaboradores (2005), as principais características do Scrum são:

- é um processo ágil para gerenciar e controlar o desenvolvimento de projetos;
- é um *wrapper* para outras práticas de engenharia de software. Como XP, por exemplo;
- é um processo que controla o caos resultante de necessidades e interesses conflitantes;
- é uma forma de aumentar a comunicação e maximizar a cooperação;
- é uma forma de detectar e remover qualquer impedimento que atrapalhe o desenvolvimento de um produto;
- é escalável desde projetos pequenos até grandes projetos em toda empresa.

Vocabulário utilizado no Scrum:

- **Backlog:** lista de todas as funcionalidades a serem desenvolvidas durante o projeto completo, sendo bem definido e detalhado no início do trabalho; deve ser listado e ordenado por prioridade de execução;
- **Sprint:** período não superior a 30 dias, em que o projeto (ou apenas algumas funcionalidades) é desenvolvido;
- **Sprint Backlog:** trabalho a ser desenvolvido num Sprint de modo a criar um produto a apresentar ao cliente; deve ser desenvolvido de forma incremental, relativa ao Backlog anterior (se existir);
- **Scrum:** reunião diária onde são avaliados os progressos do projeto e as barreiras encontradas durante o desenvolvimento;
- **Scrum Meeting:** protocolo a seguir de modo a realizar uma reunião Scrum;
- **Scrum Team:** a equipe de desenvolvimento de um Sprint;
- **Scrum Master:** elemento da equipe responsável pela gestão do projeto e liderar as Scrum Meetings, são normalmente engenheiros de software ou da área de sistemas; apesar de ser gestor, não tem propriamente autoridade sobre os demais membros da equipe.

4.1 PROCESSO SCRUM

Segundo Cruz (2006), existem dois tipos de processo: definidos e empíricos. Processos definidos são aqueles que determinam o que deve ser feito, quando e como. Para um mesmo conjunto de variáveis de entrada, pode-se esperar sempre o mesmo resultado. Um exemplo

bem conhecido de processo definido é o RUP (*Rational Unified Process*) da IBM *(Rational*). Os processos empíricos devem ser utilizados sempre que os processos definidos não forem adequados devido à complexidade do projeto, ou seja, sempre que não se conheçam todas as variáveis de entrada para que se possa estabelecer um processo repetível (com a mesma saída sempre). O Scrum é um exemplo deste processo.

Para iniciar o processo Scrum, a primeira coisa a definir é a equipe, constituída pelas pessoas designadas para trabalhar que irão compor a equipe Scrum. Esta equipe deve ter de 6 a 9 membros no máximo. Se houver mais membros do que é possível gerir, separam-se várias equipes Scrum e cada equipe focar-se-á numa área específica do trabalho, envolvendo toda a equipe para trabalhar nesta área específica.

A próxima coisa a fazer é apontar o *Scrum Master*, uma vez que é essa pessoa que conduz as *Scrum Meetings*, mede o progresso empiricamente, toma decisões e remove os obstáculos do caminho para não desacelerar ou mesmo parar o trabalho em pontos críticos. O *Scrum Master* fica encarregado - como referido anteriormente de perguntar a todos os membros da equipe as três questões mencionadas. É o *Scrum Master* que deve ser capaz de tomar decisões imediatas e resolver todos os impedimentos rapidamente, de modo a não estender o tempo da reunião.

É o *Scrum Master* que identifica o *backlog* inicial, que é todo o trabalho preeminente para uma área do produto, tanto imediato e bem-definido como de longo prazo e indefinido. Para identificar o *backlog*, a primeira coisa a fazer é listar todo o trabalho conhecido que precisa ser feito e agrupá-lo em incrementos que não devem ter duração superior a 30 dias. Se houver áreas de trabalho voláteis ou que não possam ser completamente definidas para 30 dias, deve ser estabelecido um incremento para um tempo conhecido.

Depois disto, é preciso listar todo o trabalho a ser feito e definir prioridades para todos os elementos listados. Uma vez terminado, o *backlog* deve ser assinado pelos membros das equipes, e a partir daí, só o *backlog* criado deverá ser cumprido durante este *Sprint* para cada área. Para que o processo funcione, é vital executar rigorosamente os trabalhos com base nos pontos restantes do *Sprint Backlog*. Para isso, é preciso estabelecer e conduzir as reuniões diárias Scrum, onde as equipes se encontram e se atualizam sobre o que se vai fazendo.

Isto fornece um foco diário no trabalho em desenvolvimento. Certifique-se de que as reuniões se realizam sempre na mesma hora e no mesmo local, evitando gastos na procura diária de um lugar, assim como evita as equipes terem que diariamente saber onde e quando será a reunião desse dia. Cada reunião não deve ultrapassar 30 minutos.

Durante este tempo o *Scrum Master* cumpre o seu papel de colocar as referidas questões e tomar todas as decisões necessárias. Qualquer questão a que não se responda deverá ser adiada para posteriores reuniões. No fim de cada *Sprint*, deve ser feita uma reunião para revisão e demonstração do *Sprint*. Para conduzir estas reuniões deve ser eleito um porta-voz para guiá-la e conduzi-la de forma a obter algumas questões resolvidas e registrar a retrospectiva do grupo do *Sprint*:

1. Qual o valor acrescentado neste incremento (demonstração)?
2. O que foi completado do nosso *Sprint Backlog?*
3. Qual o *feedback* por parte do cliente do produto?
4. O que aconteceu de relevante no grupo durante o *Sprint*?
5. Como é que cada um se sentiu?
6. O que podemos concluir disso?
7. O que pode ser aplicado para melhorar o próximo processo *Sprint*?

Para que o processo flua melhor, o porta-voz deve explicar perfeitamente as regras de trabalho conjunto sa equipe, e como toda a equipe deve trabalhar no *Sprint*. Cada equipe deve demonstrar algo no fim de cada *Sprint*, uma vez que o objetivo é que sigam regras de auto-organização.

A Figura 4 (página seguinte) mostra o desenvolvimento *Scrum*, retratando as fases e processos desta metodologia.

5 METODOLOGIA *EXTREME PROGRAMMING* (XP)

A metodologia *Extreme Programming* (XP) surgiu através da idéia de novos caminhos para o desenvolvimento de *software*. Esta metodologia, considerada leve, eficiente e eficaz, permite o desenvolvimento de *software* com requisitos dinâmicos ou em constantes mudanças.

XP é uma disciplina de desenvolvimento de *software* baseada em valores de simplicidade, comunicação, *feedback* e coragem. XP envolve o time inteiro para um trabalho de equipe com práticas simples, com *feedback* suficiente para capacitar o time a ver onde eles estão e a convergir para práticas em uma solução única (BECK, 2000).

Fatores que distinguem o XP de outras metodologias:

- apresentar *feedback* (retornos) contínuo e concreto em ciclos curtos;
- abordar planejamento incremental, apresentando rapidamente um plano global, que evolui durante o ciclo de vida do projeto;
- ter habilidade flexível de programar implementação de funcionalidade, respondendo às mudanças das regras de negócio;
- confiar no processo de evolução do projeto, que dura tanto quanto o sistema;
- acreditar nas práticas que trabalham tanto com as aptidões, a curto prazo, dos programadores, quanto os interesses, a longo prazo, do projeto.

O XP utiliza-se praticamente de quatro princípios básicos para alcançar eficiência no processo: comunicação, simplicidade, *feedback* e coragem. A partir desses princípios foram definidas as doze práticas básicas adotadas pelo XP.

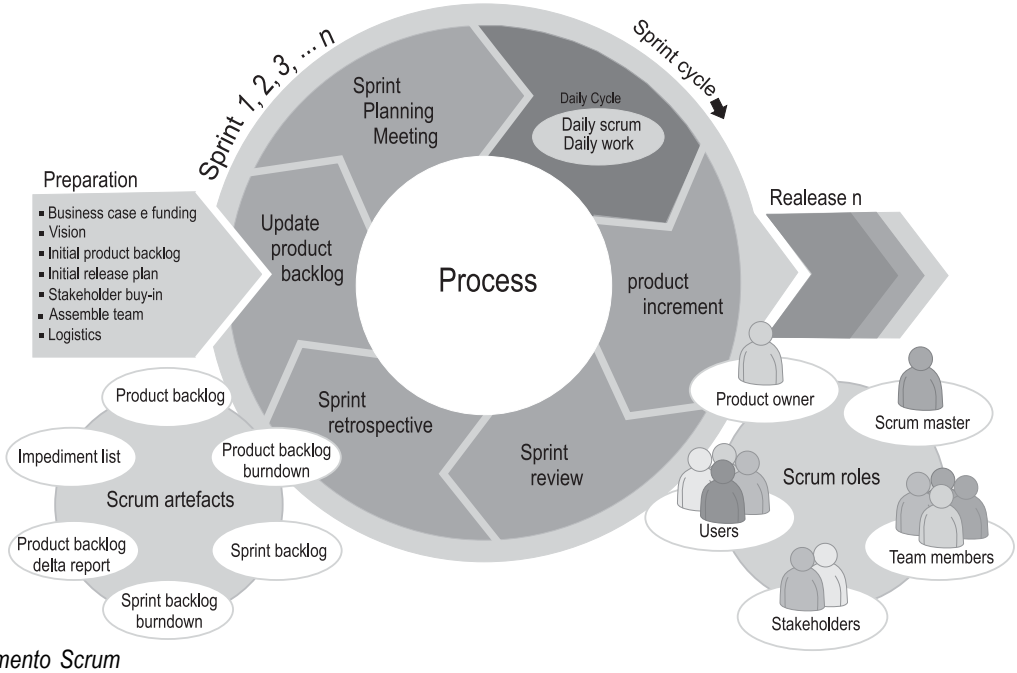

Figura 4: Desenvolvimento Scrum
Fonte:http://scrumforteamsystem.com ProcessGuidanceScrumScrum.html

5.1 AS DOZE PRÁTICAS DO XP

Abaixo serão apresentadas as doze práticas do XP, segundo Bona (2002).

1. Jogo de planejamento: determina o escopo das próximas versões, abrangendo tanto as regras de negócios quanto às considerações técnicas. Ele irá predizer o que será realizado até determinada data, e determinar qual será a próxima tarefa a ser realizada.

2. Pequenas versões: colocam o sistema em produção e o atualizam freqüentemente. O XP trabalha com dois módulos de pequenas versões: o *software* é entregue ao cliente, para que ele possa usá-lo da forma que desejar, tanto para testes quanto para liberá-lo aos usuários, mantendo o projeto aberto. São entregues versões diárias aos clientes, as quais devem conter os requisitos mais importantes.

3. Metáforas: guiam o desenvolvimento e a comunicação com o cliente. Mantêm todos os desenvolvedores em sintonia com o projeto e ajudam a definir nomes para objetos e classes.

4. Projeto simples: desenvolvimento simples, abrangendo os requisitos básicos que o sistema deve conter. No início é desenvolvido a partir de um projeto simples, mantendo as funcionalidades existentes, deixando o *software* sempre pronto para as próximas iterações.

5. Teste: são escritos testes para validar se os requisitos estão sendo atendidos. São criados testes de unidades pelos programadores e testes funcionais pelos clientes. As atividades de testes são criadas em todo o processo de desenvolvimento;

***6. Refactoring*:** aperfeiçoamento do projeto, mantendo a clareza. É o processo de melhorar o projeto de um código que já está funcionando. Os testes de unidade oferecem segurança para que aquilo que já estava funcionando continue a funcionar.

7. Programação em pares: todo o código produzido é feito em pares de programadores. Esta prática compreende dois programadores sentados lado a lado na mesma máquina, o que acrescenta qualidade ao código. Esse tipo de programação necessita de prática para funcionar corretamente.

8. Propriedade coletiva: qualquer um pode alterar qualquer código em qualquer tempo. A qualquer momento o programa pode ser alterado, por qualquer pessoa que faça parte do time de desenvolvimento, o que permite espalhar o conhecimento de todo o sistema para o time.

9. Integração contínua: integra o sistema várias vezes por dia, a cada momento uma tarefa é completada. O sistema é integrado o tempo todo. A integração contínua evita ou descobre problemas de compatibilidade cedo. O código é integrado e testado depois de algumas horas, no máximo depois de um dia de desenvolvimento.

10. Semana de 40 horas: não se deve trabalhar mais que 40 horas por semana. O time XP trabalha intensamente de modo a maximizar a produtividade. Um programador, quando está cansado, raciocina mais lentamente e fica mais distraído. Um programador cansado é ótimo para inserir erros (*bugs*) em um programa, erros que vão dar trabalho para corrigir e exigir mais horas extras.

11. Cliente dedicado: possuir clientes exclusivos a todo o momento, para determinar atributos e propriedades. Um cliente deve estar sempre

disponível, fazendo parte da equipe, e este cliente deverá ser aquele que realmente irá utilizar o sistema desenvolvido. A vantagem é que o cliente poderá fornecer detalhes do sistema, quando surgirem dúvidas;

12. Código-padrão: todo o código é escrito da mesma forma por programadores diferentes. Todo o código segue um padrão entre os programadores, permitindo que o *software* seja consistente e fácil para o time entender.

A maioria das regras XP causam polêmica ou não fazem sentido se aplicadas isoladamente. É a sinergia de seu conjunto que sustenta o processo XP, encabeçando uma verdadeira revolução de metodologias ágeis.

5.2 FASES DO PROCESSO XP

O ciclo de vida do processo XP é curto e difere dos padrões, pois nestes processos os requisitos mudam freqüentemente para atender às funcionalidades do sistema. Quando não mais existirem novas histórias, é o momento de finalizar o projeto. É o momento de escrever algumas páginas sobre a funcionalidade do sistema, um documento que, no futuro, ajude a saber como realizar alguma alteração no sistema. Uma boa razão para finalizar o projeto é o cliente estar satisfeito com o sistema e não ter mais nada que consiga prever para o futuro. Toda a equipe que trabalhou no sistema deve ser reunida para reavaliação. Deve aproveitar a oportunidade para analisar o que pode ter causado queda no sistema e o que fez o projeto avançar. Assim, o time saberá melhor o que fazer no futuro e executará tarefas de formas diferentes da próxima vez.

O XP é um processo que guia um projeto e sua equipe de desenvolvimento. O projeto deve ser ágil e incremental. O XP descarta a documentação, pois o objetivo é ganhar tempo.

6 RESULTADOS OBTIDOS

Inicialmente foram levantadas informações sobre as metodologias expostas neste trabalho, tendo sido realizadas pesquisas bibliográficas pelos integrantes do projeto.

Após a obtenção destas informações foi elaborada a Tabela 1, que compara a relevância com que cada metodologia trata os requisitos não funcionais. Esta tabela consiste na comparação dessas metodologias entre si de acordo com o material pesquisado e suas características, de maneira que se possam observar com mais clareza os pontos fortes e fracos de cada metodologia quanto aos RNFs, retratando assim a relevância de cada uma em seu contexto.

Os critérios abordados na Tabela 1 são requisitos relevantes para qualquer metodologia de desenvolvimento de *software*. As metodologias citadas neste trabalho atendem a todos esses requisitos, algumas dando mais ênfase, outras menos. Seguindo essa linha de pensamento, foi ela-borada uma forma de avaliar como cada metodologia aborda esses re-quisitos, classificando sua aplicação em cada metodologia como: *pouco* (requisitos abordados com pouca relevância); *muito* (requisitos abordados com muita relevância) e *indefinido* (requisitos não tratados nas meto-dologias). Essa classificação foi elaborada com base nas particularidades de cada metodologia, fazendo um comparativo entre elas e observando quais destas tinham vantagens/desvantagens em relação às outras.

Tabela 1. Comparação das metodologias entre si.

CRITÉRIOS	MERUSA	XP	SCRUM
Usabilidade	Muito	Muito	Muito
Segurança	Muito	Pouco	Pouco
Performance	Muito	Indefinido	Indefinido
Portabilidade	Pouco	Muito	Muito
Agilidade	Pouco	Muito	Muito

7 CONCLUSÕES

Este trabalho apresentou a importância das metodologias de desenvolvimento de *software* na fase inicial do desenvolvimento do projeto, bem como no decorrer deste. Três metodologias em particular foram abordadas, sendo elas: MERUSA, XP, SCRUM.

Foram traçados comparativos em suas formas de abordagem de desenvolvimento de *software* e feita a avaliação de seus requisitos em suas fases de desenvolvimento, dentro de cada metodologia.

A MERUSA (Metodologia de Especificação de Requisitos de Usabilida-de e Segurança orientada para a Arquitetura do sistema), por ser uma metodologia a ser aplicada a longo prazo, dentro dos requisitos estudados neste trabalho, dedica uma maior atenção aos requisitos relativos à usabilida-de, segurança e *performance* do sistema, é uma metodologia muito utilizada para o desenvolvimento de sistema integrados e distribuídos. Por outro lado, dedica menor atenção aos relativos à portabilidade e agilidade.

Já as metodologias XP (*Extreme Programming*) e SCRUM dedicam mais atenção aos requisitos relativos à usabilidade, portabilidade e agilidade. Elas estão entre uma nova geração de metodologias que priorizam a diminuição de tempo para a elaboração de sistema. São consideradas metodologias ágeis, e devido a isso dedicam menor atenção aos requisitos de segurança e *performance* do sistema. São utilizadas no desenvolvimento de sistemas de qualquer porte, independentemente da quantidade de integrantes das equipes de desenvolvimento, análise e especificação.

REFERÊNCIAS

AVELINO, Valter Fernandes. **MERUSA:** Metodologia de Especificação de Requisitos Usabilidade e Segurança Orientada para Arquitetura. 2005. 277 fls. Tese (Doutorado em Engenharia) – Escola Politécnica da Universidade de São Paulo, São Paulo, 2005.

BECK, Kent. **Extreme Programming Explained**. São Paulo: Addison-Wesley, 2000.

BONA, Cristina. **Avaliação de Processos de Software:** Um estudo de caso em XP e ICONIX. 2002. 122 fls. Tese (Mestrado em Engenharia de Produção) – Universidade Federal de Santa Catarina, Florianópolis, 2002.

CRUZ, R. L. S. **Metodologia Scrum.** Disponível em: <http:// scrum.squarespace. com/display/ShowJournal?moduleId= 711917&categoryId=66671 >. Acesso em: 01 ago. 2006.

CYSNEIROS, L. M. **Requisitos Não Funcionais:** Da Elicitação ao Modelo Conceitual. 2001. 224 fls. Tese (Doutorado em Ciência da Computação) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2001.

_____; LEITE, J. C. S. P. Utilizando Requisitos Não Funcionais para Validar Modelos de Dados. In: WORKSHOP DE ENGENHARIA DE REQUISITOS, 1, 1998. **Anais...** Maringá: WER, 12 out. 1998, p. 1-8.

FERREIRA, Décio et al. **SCRUM -** Um Modelo Ágil para Gestão de Projetos de Software. Disponível em: <http://paginas.fe.up.pt/~aaguiar/es/artigos%20finais/es_ final_19.pdf>. Acesso em: 15 set. 2006.

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

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

STANDISH, T. A. Standish Group. **Software Chaos**, Massachusetts, Jan.1995. Disponível em: <http:// www.standishgroup.com>. Acesso em: 10 nov. 2006.

WIKIPÉDIA. **Engenharia de Software.** Disponível em: <http://pt.wikipedia.org/wiki/Engenharia_de_software>. Acesso em: 05 nov. 2006.