Luiz Marcio Cysneiros

Requisitos Não Funcionais: Da Elicitação ao Modelo Conceitual

Tese apresentada ao Departamento de Informática da PUC/RJ como parte dos requisitos para a obtenção do título de Doutor em Ciências da Computação.

Orientador: Julio Cesar Sampaio do Prado Leite

DEPARTAMENTO DE INFORMÁTICA

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Índice

1 - Introdução

1.1 - Motivação

A crescente complexidade dos sistemas de software e o aumento da exigência de qualidade por parte dos clientes vêm impulsionando o mercado a cada dia produzir mais softwares que atendam, não somente as funcionalidades exigidas, mas também a aspectos não funcionais exigidos pelos clientes tais como: custo, confiabilidade, segurança, manutenabilidade, portabilidade, performance, rastreabilidade de informações entre outros. Estes aspectos não funcionais devem ser tratados como requisitos não funcionais (RNF) do software. Devem ainda ser tratados desde o início do seu desenvolvimento [Chung 95] [Chung 00] [Cysneiros 97] estendendo este tratamento por todo o ciclo de vida do software.

Requisitos não funcionais vêm sendo citados em vários processos de desenvolvimento de software, dentre outras formas como restrições e condições de contorno, porém sempre de maneira quando muito secundária e altamente informal do ponto de vista da elicitação de requisitos. Este tipo de tratamento faz com que, quando tratados, este requisitos sejam freqüentemente contraditórios, difíceis de serem considerados durante o desenvolvimento de software e difíceis de serem validados. O fato destes requisitos terem sido mal elicitados ou não elicitados tem causado uma série de histórias de insucessos, incluindo a desativação de sistemas pouco após terem sido implantados [Finkelstein 96][Lindstrom 93]. Estudos apontam estes requisitos como estando entre os mais caros e difíceis de corrigir [Brooks 87] [Davis 93] [Cysneiros 99].

Por outro lado, requisitos não funcionais têm recebido muito pouca atenção na literatura e são muito menos compreendidos do que os outros fatores menos críticos

ao desenvolvimento de software [Chung 00]. A maior parte dos trabalhos que abordam RNFs, o faz orientado ao produto, ou seja, se situa no campo da quantificação do grau de conformidade de um software para com os RNFs a que ele deveria satisfazer [Keller 90] [Boehm 76] [Boehm 78] [Basili 91] [Fentom 97] [Musa 87] [Lyu 96].

Poucos trabalhos propõem um tratamento explícito para RNFs sob a ótica do processo de desenvolvimento de software. Os trabalhos que seguem esta ótica propõem o desenvolvimento de técnicas para justificar decisões sobre a inclusão ou exclusão de requisitos que se refletirão no desenho do software durante o desenvolvimento deste. Ao contrário da abordagem orientada a produto, preocupada em medir quanto um software satisfaz a RNFs, este tipo de abordagem procura racionalizar o processo de desenvolvimento de software em termos de RNFs. Freqüentemente, quando adicionamos um RNF a uma especificação de requisitos, forçamos tomadas de decisões que poderão afetar positiva ou negativamente outros RNFs, efeito esse visualizado como interdependências entre RNFs. Estas interdependências podem ser positivas ou negativas, ou seja, um RNF pode influenciar positivamente outro RNF contribuindo assim para sua satisfação, bem como pode influenciar negativamente, ou seja, contribuir para que um destes RNFs não seja satisfeito ou satisfeito apenas parcialmente. Dentre os trabalhos existentes, destacamos os seguintes:

- Boehm [Boehm 96] monta uma base de conhecimento com RNFs priorizados pela visão do cliente, mas que entretanto só trata RNFs primários, ficando assim num nível de abstração ainda muito alto para identificação de conflitos entre RNFs ou com requisitos funcionais.

- Kirner [Kirner 96] descreve as propriedades de 6 RNFs no domínio de sistemas de tempo real, performance, confiabilidade, seguro, segurança, manutenabilidade e usabilidade. O trabalho de Kirner descreve como medir estes RNFs e propõe algumas heurísticas de como tratar com estes requisitos durante o desenvolvimento deste tipo de software.
- O Framework proposto por Chung [Chung 93] [Chung 95] [Chung 00] é a abordagem mais completa até o momento e procura tratar de todos os tipos de requisitos não funcionais desde as primeiras etapas do processo de desenvolvimento de software. Apesar de ser um framework bastante completo e eficaz para lidar com requisitos não funcionais durante a elicitação de requisitos, este framework não favorece a integração destes requisitos nas etapas seguintes do desenvolvimento de software, ficando, assim, uma lacuna aberta para que conflitos entre requisitos funcionais e não funcionais passem desapercebidos durante o projeto do software.
- Neto [Neto 00] propõe uma estratégia apoiada pela ferramenta OORNF para o desenvolvimento de software, baseada no léxico ampliado da linguagem estendido para tratar RNFs. A partir do léxico, são gerados os cenários, cartões CRC e um modelo conceitual orientado a objetos. Apesar desta proposta ter apresentado bons resultados nos estudos de casos controlados a que foi submetida, por tratar de RNFs apenas no Léxico Ampliado da Linguagem [Leite 93], ela não favorece o tratamento de interdependências entre RNFs. Outro problema é que o tratamento de RNFs até o desenho do software só é efetivo se utilizarmos a estratégia de derivação de cenários proposta por Hadad [Hadad 97] e a estratégia de derivação de cartões CRCs e diagramas de classes proposta por Leonardi [Leonardi 97].

Da análise da literatura existente é possível observar que o uso dos RNFs durante o processo de desenvolvimento de software é abordado apenas de forma parcial. Pode-se ainda observar, que existe uma lacuna no que tange à existência de propostas que apresentem uma estratégia para o uso de RNFs, não apenas nas primeiras etapas do processo de desenvolvimento de software, mas também na integração dos RNFs elicitados com os modelos conceituais gerados, e na conseqüente avaliação de impactos resultantes no desenho do software.

Trabalhos apresentados por Cysneiros [Cysneiros 99, Cysneiros 01] enfatizam premissas apresentadas por Chung [Chung 00] de que a falta de integração dos RNFs aos requisitos funcionais, e por conseqüência sua integração aos modelos conceituais, pode resultar em tempos maiores para se finalizar um software, assim como maiores custos com manutenção.

1.2 - Visão Geral da Solução Proposta

Apresentaremos nesta tese um processo para se lidar com requisitos não funcionais desde as etapas iniciais do processo de desenvolvimento de software. A integração dos RNFs, elicitados durante as fases iniciais do desenvolvimento do software, com os modelos conceituais gerados a partir dos requisitos funcionais deverá permitir obtermos um modelo conceitual que enfoque os requisitos funcionais e não funcionais ao mesmo tempo, obtendo-se assim modelos mais completos e com os impactos da satisfação dos requisitos não funcionais já analisados, resultando em uma melhoria da qualidade final do produto sob a ótica, tanto do cliente quanto do desenvolvedor, bem como em um menor tempo de entrega de produtos e menores custos de manutenção.

2 - Conceitos Básicos

2.1 - Engenharia de Requisitos

A engenharia de requisitos procura sistematizar o processo de definição de requisitos. Essa sistematização é necessária porque a complexidade dos sistemas exige que se preste mais atenção ao correto entendimento do problema antes do *comprometimento de uma solução.* [Leite 94]. Uma boa definição para requisitos pode ser encontrada em [Leite 94] e é mostrada a seguir.

"***Requisitos:*** *Condição necessária para a obtenção de certo objetivo, ou para o preenchimento de certo objetivo*."

Pesquisas sobre a aquisição (elicitação) de requisitos são valorosas por duas razões: primeiramente, da perspectiva da engenharia de software, a elicitação de requisitos é talvez a mais crucial parte do processo de desenvolvimento de software. Estudos [Boehm 84] indicam que, quando só detectados depois do software implementado, erros em requisitos de software são até 20 vezes mais caros de se corrigir que qualquer outro tipo de erro. Além disso, elicitação de requisitos não é atualmente muito bem apoiada por ferramentas de software.

Para a engenharia de requisitos é fundamental que o engenheiro de software delimite os contornos do macrosistema em que a definição de software ocorrerá, ou seja, delimitar o Universo de Informações do sistema (UdI).

É importante ressaltar que o UdI sempre existe. O UdI independe do modelo que estamos utilizando. Mesmo que o macrosistema não esteja bem definido, sempre podemos, e devemos, estabelecer os limites de nossa atuação. Quanto mais bem delineado um UdI, maiores são as chances de um software bem definido.

A elicitação de requisitos é um processo que está constantemente em evolução, ou seja, toda vez que o UdI muda, o modelo resultante do processo de elicitação de requisitos deve mudar junto. Esta realidade é muitas vezes desconsiderada através do que se procurou determinar como um congelamento dos requisitos. Se por um lado a constante evolução do UdI pode gerar custos altos no desenvolvimento, uma vez que temos de estar atualizando nossos requisitos, por outro lado o congelamento dos requisitos em um determinado momento não deve produzir efeitos contrários, aumentando consideravelmente o custo do produto final em decorrência de alterações que serão demandadas para a aceitação do software. Este é um tema ainda pouco explorado e maiores pesquisas deverão ser realizadas neste campo para que possamos conhecer o ponto ideal de equilíbrio.

Em muitas organizações, os requisitos são escritos como parágrafos em linguagem natural e suplementados por diagramas [Kotonya98][Sommerville98]. A linguagem natural é a única notação que é compreendida por todos os leitores potenciais (ex. clientes, usuários, engenheiros de software, engenheiros de requisitos) dos requisitos. Não obstante, alguns pesquisadores não compartilham desta visão tendo feito pesadas críticas a esta utilização e ressaltando a ambigüidade como um dos problemas inerentes aos requisitos escritos em linguagem natural [Meyer85]. Para nós, os eventuais problemas que advêm do emprego da linguagem natural são fundamentalmente resultantes da má utilização desta linguagem, e não necessariamente problemas inerentes à linguagem natural. Sommerville e Sawyer apresentem cinco orientações para se escrever requisitos em linguagem natural [Sommerville98]. Toro apresenta padrões lingüísticos e padrões de requisitos para representar requisitos em linguagem natural [Toro99]. Leite apresenta uma estratégia de representação de requisitos na qual utiliza listas de requisitos em linguagem

natural, suplementadas por léxicos ampliados da linguagem do UdI [Leite92]. Estas propostas podem minimizar os problemas identificados pelos críticos da linguagem natural.

É interessante ressaltar que diversos pesquisadores que apontavam a inerência da ambigüidade da linguagem natural, recentemente mudaram de opinião. Kotonya [Kotonya 95] afirma que alguns problemas devem ser tratados por todo método da engenharia de requisitos. Entre estes problemas, encontra-se a inerência de ambigüidade da linguagem natural que poderia ocasionar más interpretações . Posteriormente, Kotonya indicou que requisitos devem ser escritos em linguagem natural e suplementados por diagramas [Kotonya98].

Uma vez que dificilmente teremos à disposição recursos que permitam satisfazer todos os requisitos dos usuários [Berry98], requisitos podem ser classificados como desejáveis ou obrigatórios, utilizando-se aqui de um enfoque voltado para a necessidade de priorizar requisitos. Os requisitos podem ser ainda classificados como estáveis, mudam mais lentamente, ou voláteis, mudam mais rapidamente. Esta classificação auxilia a atividade de gerência de requisitos, uma vez que possibilita antecipar mudanças prováveis de requisitos. Por fim, uma outra classificação que tem tido bastante aceitação na comunidade acadêmica [Yeh84] [Roman85] [Brackett90] [Mylopoulos 92] [Sommerville98] [Kotonya98] [Mamani99] [Cysneiros99], divide os requisitos em requisitos funcionais e requisitos não funcionais.

2.2 - Requisitos Não Funcionais

A complexidade de um software é determinada em parte por sua funcionalidade, ou seja, o que o sistema faz, e em parte por requisitos gerais que fazem parte do desenvolvimento do software como custo, performance,

confiabilidade, manutenabilidade, portabilidade, custos operacionais entre outros [Chung 00]. Estes requisitos podem ser chamados de requisitos não funcionais. Esta denominação foi utilizada por Roman [Roman 85], sendo os RNFs também conhecidos como atributos de qualidade [Boehm 78] [Keller 90], restrições [Roman 85], objetivos [Mostow 85] entre outros. Recentemente, o termo requisitos não funcionais vem se firmando como nomenclatura corrente no meio acadêmico, sendo esta a nomenclatura que iremos utilizar.

Os RNFs desempenham um papel crítico durante o desenvolvimento de sistemas, e erros devido a não elicitação ou a elicitação incorreta destes estão entre os mais caros e difíceis de corrigir, uma vez que um sistema tenha sido implementado [Brooks 87] [Davis 93].

Embora sem formalmente definir RNFs, alguns trabalhos propõem classificações destes. Em um relatório do *Rome Air Development Center* [Bowen 85], RNFs são classificados da seguinte maneira:

- orientados ao consumidor (ou critérios de qualidade de software), aqui se referindo a requisitos observáveis pelo cliente como eficiência, correção, amigabilidade e outros; e,
- atributos orientados tecnicamente (ou critérios de qualidade) associados mais a requisitos ligados ao sistema, como tratamento de erros e anomalias, completeza, processamento veloz em pontos críticos de tempo real e outros.

Duas diferentes abordagens são utilizadas em tratamento sistemático de RNFs. Elas podem ser classificadas como orientada a produto e orientada a processo [Chung 93]. A primeira aborda RNFs de forma a classificar o quanto um sistema satisfaz os RNFs que dele são requeridos. Por exemplo, medir a visibilidade de um software pode incluir, entre outras coisas, a medição de quantos desvios (branches) existem em um

software. Isto pode ser obtido utilizando-se um critério como: “Não deve haver mais do que X desvios por 1000 linhas de código”. Quase todos os trabalhos em RNF utilizam esta abordagem, orientada a produto, a qual é bem descrita por Keller [Keller 90].

Boehm [Boehm 78] mostra que a qualidade geral do produto final de um software pode ser melhorada simplesmente tornando os desenvolvedores conscientes de que critérios de qualidade deviam ser cumpridos. Também baseados em uma abordagem quantitativa sob a ótica da qualidade de software, Basili e Musa [Basili 91] propõem modelos e métricas para o processo de engenharia de software de uma perspectiva gerencial.

Já a abordagem orientada a processo, que é por nós utilizada nesta tese, advoga o desenvolvimento de estratégias para justificar decisões de desenho *durante o processo de desenvolvimento de software*. Ao contrário da abordagem orientada a produto, nesta abordagem procura-se racionalizar o processo de desenvolvimento propriamente dito em termos de RNFs [Chung 00]. Freqüentemente, quando adicionamos um RNF a uma especificação de requisitos, forçamos tomadas de decisões que poderão afetar positiva ou negativamente outros RNFs, efeito esse visualizado como interdependências entre RNFs. Estas interdependências podem ser positivas ou negativas, ou seja, um RNF pode influenciar positivamente outro RNF contribuindo assim para sua satisfação, bem como pode influenciar negativamente, ou seja, contribuir para que um destes RNFs não seja satisfeito ou satisfeito apenas parcialmente.

Utilizamos nesta tese a noção de que um RNF raramente pode ser considerado totalmente satisfeito. Em conformidade com o que é utilizado por Mylopoulos [Mylopoulos 92], onde é utilizado o termo “satisfice” ao invés de “satisfy”,

adotaremos daqui por diante a idéia de que um RNF quando dito satisfeito, não necessariamente terá sido totalmente satisfeito, e sim, satisfeito dentro de limites razoáveis de serem utilizados.

Ortogonalmente, podemos classificar o tratamento dispensado a RNFs como divididos em qualitativos e quantitativos. A maioria das abordagens orientada a produto é quantitativa, uma vez que elas propõem métodos quantitativos para medir o quanto um software satisfaz um dado RNF. As abordagens orientadas a processo são, por outro lado, inteiramente voltadas para o aspecto qualitativo, via de regra adotando idéias oriundas do raciocínio qualitativo [AI 84].

RNFs abordam importantes aspectos relacionados à qualidade de softwares. Eles são essenciais para que softwares sejam bem sucedidos. A não observância de RNFs pode resultar em: softwares com inconsistência e de baixa qualidade; clientes e desenvolvedores insatisfeitos; tempo e custo de desenvolvimento além dos previstos devido à necessidade de se consertar softwares que não foram desenvolvidos sob a ótica da utilização de RNFs.

RNFs são geralmente subjetivos, uma vez que podem ser vistos, interpretados e conceituados de forma diferente por diferentes pessoas. É verdade que os requisitos funcionais também sofrem do problema de diferentes conteúdos advindos de diferentes pontos de vista, porém, como os RNFs são por natureza mais abstratos e uma vez que NFRs são comumente relacionados de maneira breve e vaga este problema é potencializado [Chung 00].

Além disso, RNFs freqüentemente interagem entre si, uma vez que a tentativa de satisfazer um RNF pode prejudicar ou ajudar a satisfazer outros RNFs. Como vários RNFs possuem efeitos de natureza global nos softwares, uma solução localizada pode não ser adequada.

Por todas estas razões, RNFs são difíceis de se lidar e vitais de serem tratados para que possamos obter softwares de qualidade.

2.3 - Requisitos Funcionais versus Requisitos Não Funcionais

Os *requisitos funcionais* são requisitos que expressam funções ou serviços que um software deve ou pode ser capaz de executar ou fornecer. As funções ou serviços são, em geral, processos que utilizam entradas para produzir saídas.

Os *requisitos não funcionais (RNFs)* são requisitos que declaram restrições, ou atributos de qualidade para um software e/ou para o processo de desenvolvimento deste sistema. Segurança, precisão, usabilidade, performance e manutenabilidade são exemplos de requisitos não funcionais.

A distinção entre o que é um requisito funcional e o que é um não funcional nem sempre é clara. Parte da razão advém para o fato de que os RNFs estão sempre relacionados a um requisito funcional [EAGLE 95] [Chung 00]. A Grosso modo, partindo da definição acima podemos dizer que um requisito funcional expressa algum tipo de transformação que tem lugar no software, enquanto um RNF expressa como essa transformação irá se comportar ou que qualidades específicas ela deverá possuir.

Um exemplo de diferença entre requisito funcional e RNF pode ser visto a seguir.

Suponhamos que estejamos no domínio de laboratórios de análises clínicas: um requisito funcional desse sistema poderia ser expresso da seguinte forma:

> "O sistema deve fornecer uma entrada de dados que possibilite a designação de resultados a exames admitidos para um paciente por técnicos, supervisores e chefes".

Este mesmo requisito funcional poderá ter associado a ele o seguinte RNF:

> "Alguns exames deverão ter tratamento especial para a entrada de resultados. Para estes exames valores acima ou abaixo de pré-determinados valores só poderão ser digitados por chefes de seção".

É importante ressaltar que esse último parágrafo não exprime uma função do sistema, e sim, uma restrição a uma função existente, entrar resultados de exames. O que se vê aqui é que essa funcionalidade do sistema deverá ser restrita de forma que, quando utilizada por pessoas que não sejam chefes, será restrita à condição de que essas pessoas estejam digitando um valor que se encontre dentro de limites pré-estabelecidos.

É visível que este RNF irá demandar um processo de segurança no tratamento de acesso a módulos do software, bem mais complexo daquele que seria implantado se apenas o requisito funcional tivesse sido especificado. A implementação de um processo de acesso a módulos, que critique não apenas a permissão de acesso ao se entrar em um módulo de software, mas também o conteúdo do que está sendo digitado dentro do módulo, é consideravelmente mais complexa. A tardia ou não elicitação deste requisito certamente implicaria num grande "retrabalho".

2.4 - Classificação de RNFs

Algumas classificações dos RNFs podem ser encontradas na literatura. Mamani propõe a classificação de RNFs apresentada na Figura 2.1 [Mamani 99] enquanto a Figura 2.2 apresenta a classificação de RNFs proposta por Sommerville [Sommerville 92] e a Figura 2.3 mostra uma classificação proposta por Boehm [Boehm 76] na qual ele define uma árvore mínima de padrões de qualidade que um software deveria apresentar.

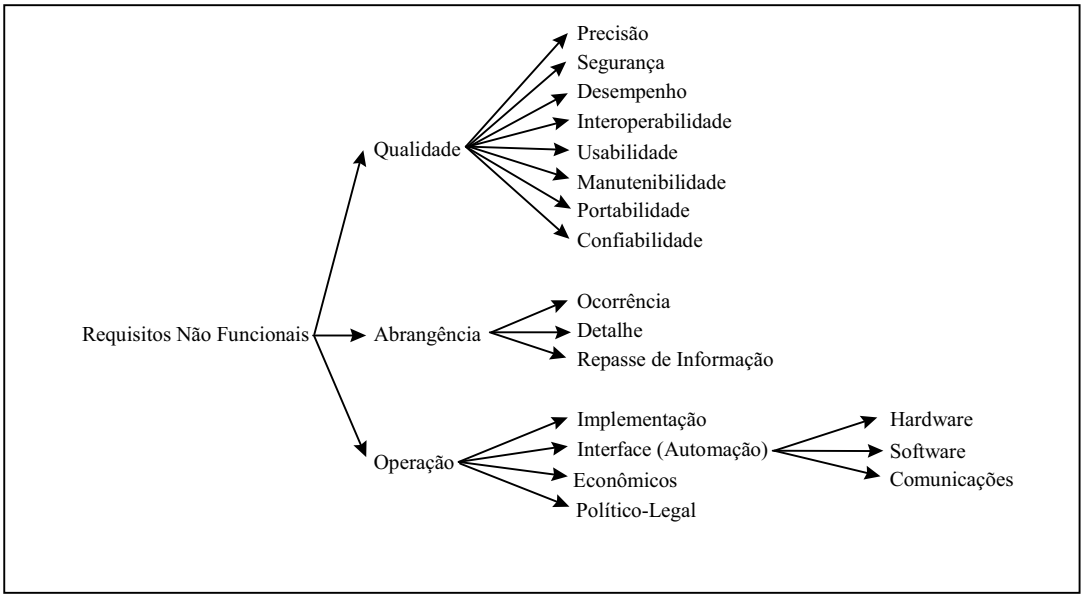

Figura 2.1 - Classificação de RNFs proposta por Mamani

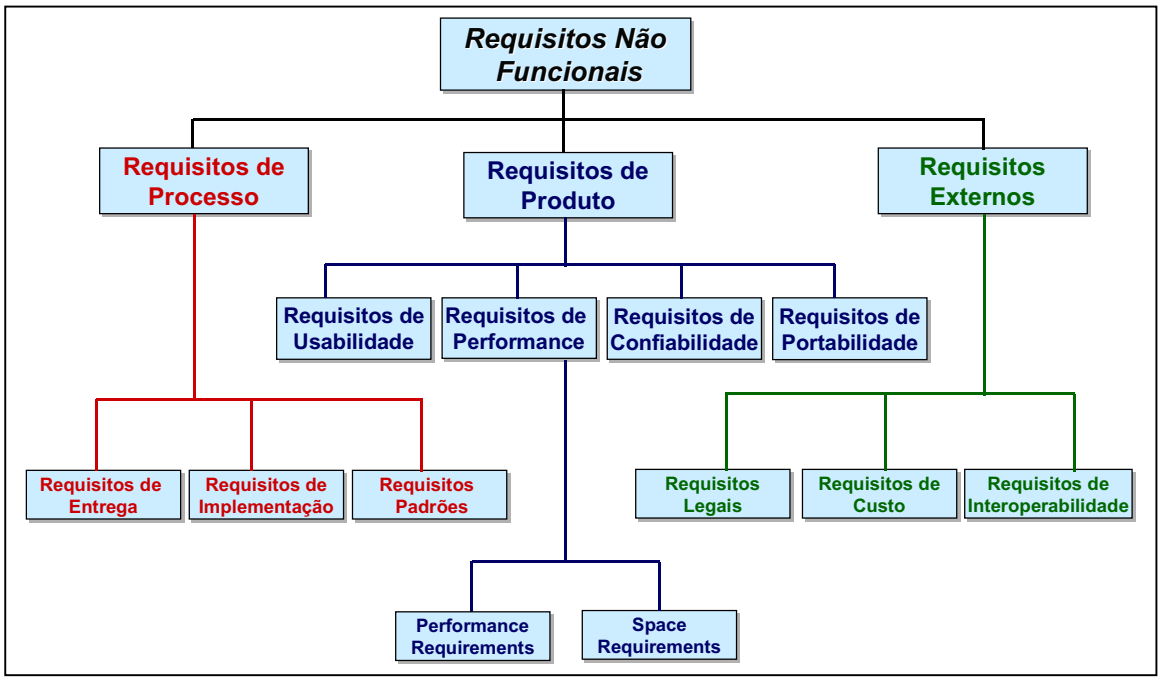

Figura 2.2 - Classificação de RNFs segundo Sommerville

Uma outra fonte que classifica alguns RNFs é o padrão internacional ISO 9126 [ISO9126]. Nesta norma são descritas seis características que definem a qualidade de software: funcionalidade, confiabilidade, usabilidade, eficiência, manutenabilidade e portabilidade. Se observarmos as diversas definições de RNF, facilmente iremos considerar que estes requisitos de qualidade definidos na ISO 9126 são, em geral,

RNFs. O item funcionalidade é, porém, uma exceção que certamente não poderá ser enquadrado como RNF.

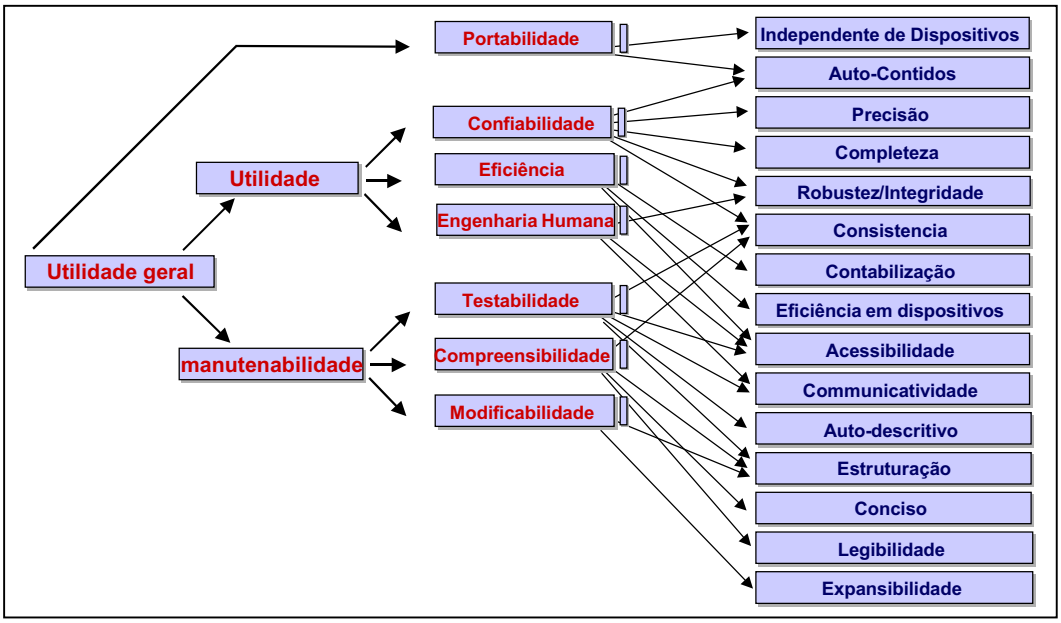

Figura 2.3 - Árvore de Características de Qualidade de Software (Boehm 76)

Proporemos aqui uma taxonomia que classifica os RNFs em primários e específicos. RNFs primários são aqueles mais abstratos que representam propriedades como: Confiabilidade, Rastreabilidade, Precisão, Segurança. Já os RNFs específicos são decomposições que se seguem aos RNFs primários e tendem a diminuir o nível de abstração de um determinado RNF, e portanto, atingir um nível de granularidade no tratamento da informação mais detalhado.

Um exemplo desta classificação seria o RNF Primário *Confiabilidade* que pode ser decomposto em validação, autorização e entrega do software.

RNFs primários podem ser decompostos em outros RNFs secundários até que se chegue ao que Chung [Chung 00] chama de operacionalização dos RNFs. Uma operacionalização de um RNF é uma ação ou informação que irá garantir a satisfação do RNF. Por exemplo, no caso de um sistema de informação para laboratórios de análises clínicas, a entrega do laudo ao paciente possui um RNF de *confidencialidade*

que seria seu RNF primário. Para que possamos detalhar o que isso implica, temos de decompor este RNF em um RNF específico de *segurança operacional* que por sua vez poderia ser decomposto na operacionalização *solicitar carteira de identidade*. Ou seja, para satisfazer o RNF *confidencialidade* teríamos de prever uma ação no sistema, que garantisse que o funcionário que entregou o laudo solicitou a carteira de identidade ao paciente antes de entregar-lhe o laudo.

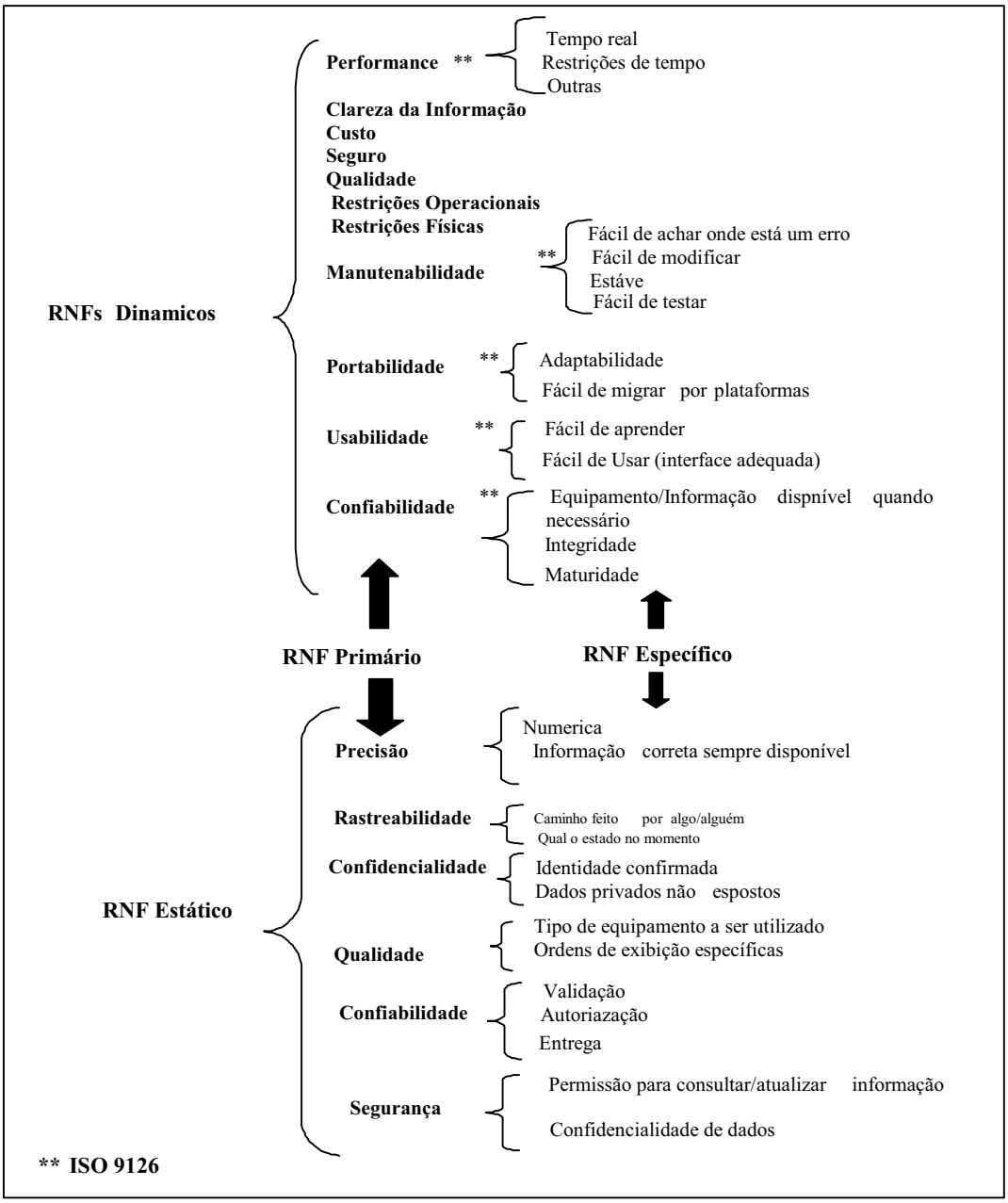

Figura 2.4 – Uma Taxonomia para RNFs

Ortogonalmente, separamos os RNFs em estáticos e dinâmicos. RNFs estáticos são aqueles que, quando presentes, **normalmente requerem o uso de dados** para validá-los como, por exemplo: segurança, precisão e rastreabilidade. RNFs dinâmicos, por outro lado, usualmente representam conceitos mais abstratos, **que normalmente envolvem ações ou critérios de qualidade** como, por exemplo: Qualidade, performance, manutenabilidade, restrições operacionais. Alguns RNFs podem ser tanto estáticos quanto dinâmicos dependendo do contexto do domínio que estejam inseridos. A Figura 2-4 mostra um resumo desta taxonomia que também pode ser utilizada como um checklist para a elicitação de RNFs.

A lista apresentada na figura acima não pretende ser completa, mas sim, um ponto de partida para cada desenvolvedor desenvolver seu próprio conhecimento a respeito de RNFs. Uma lista mais exaustiva de RNFs pode ser encontrada em [Chung 00].

2.5 - O Léxico Ampliado da Linguagem

O principal objetivo do Léxico Ampliado da Linguagem (LAL) é registrar a linguagem utilizada pelos atores do UdI, sem contudo se preocupar com a funcionalidade [Franco92][Leite93]. O LAL do UdI é composto por entradas, onde cada entrada está associada a um símbolo (palavra ou frase) da linguagem do UdI. Cada símbolo pode possuir sinônimos e é descrito através de noções e impactos. As noções descrevem o significado e as relações fundamentais de existência do símbolo com outros símbolos (denotação). Os impactos descrevem os efeitos do uso ou ocorrência do símbolo no UdI ou os efeitos de outras ocorrências de símbolos no UdI sobre o símbolo (conotação). Dependendo do símbolo que descrevem, as entradas podem ser classificadas como sujeito, verbo, objeto e estado (predicativo).

Ao se descrever entradas do LAL, deve-se obedecer aos princípios de vocabulário mínimo e de circularidade. O princípio de vocabulário mínimo demanda que a