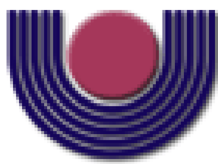


UNIOESTE – Universidade Estadual do Oeste do Paraná

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

Colegiado de Ciência da Computação

Curso de Bacharelado em Ciência da Computação

ANÁLISE PARA SELEÇÃO DE COMPONENTES BASEADA EM REQUISITOS NÃO-FUNCIONAIS COM FOCO EM APLICAÇÕES EMPRESARIAIS

Daniel Bordignon Cassanelli

CASCAVEL

2010

DANIEL BORDIGNON CASSANELLI

ANÁLISE PARA SELEÇÃO DE COMPONENTES BASEADA EM REQUISITOS NÃO-FUNCIONAIS COM FOCO EM APLICAÇÕES EMPRESARIAIS

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Victor Francisco Araya Santander

CASCAVEL

2010

DANIEL BORDIGNON CASSANELLI

ANÁLISE PARA SELEÇÃO DE COMPONENTES BASEADA EM REQUISITOS NÃO-FUNCIONAIS COM FOCO EM APLICAÇÕES EMPRESARIAIS

Monografia apresentada como requisito parcial para obtenção do Título de *Bacharel em Ciência da Computação*, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Victor Francisco Araya Santander
(Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Aníbal Mantovani Diniz
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Carlos José Maria Olguin
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 04 de novembro de 2010

DEDICATÓRIA

Dedico este trabalho a minha família, especialmente minha mãe, por sempre ter acreditado em mim. Aos meus amigos, que sempre foram fonte de boas ideias e momentos de descontração. Ao Prof. Victor por ter ajudado no trabalho de lapidar a ideia e pela paciência na correção do trabalho.

EPÍGRAFE

"Nesta longa estrada da vida,
vou correndo e não posso parar.

Na esperança de ser campeão,
alcançando o primeiro lugar."

Milionário & *José Rico* - *Estrada da Vida* - *1977*

Lista de Figuras

Figura 2.1: Elementos de um cenário …..17

Figura 2.2: Cenário geral de disponibilidade …... 17

Figura 2.3: Cenário específico de disponibilidade …..17

Figura 2.4: Exemplo de Gráfico SIG (Softgoal Interdependency Graph) …..............................22

Figura 2.5: Tipos de satisfação presentes no *NFR-Framework* ....…..............23

Figura 3.1: Conceito de Objeto/Componente .... …........................30

Figura 3.2: Ideia de composição entre frameworks 30

Figura 3.3: Visão de mapeamento entre o repositório e o SUD 32

Figura 3.4: Processo para a utilização de um pacote COTS em um SUD 33

Figura 3.5: Taxonomia de Componentes proposta no CARE 33

Figura 4.1: Taxonomia estendida para o SIGTAX 36

Figura 4.2: Catálogo de atributos ISO 9126 representado através de um gráfico SIG …......... 37

Figura 5.1 Catálogo SIG Inicial específico para componentes 41

A.1: Gráfico SIG dos critérios analisados no componente BIRT 53

A.2: Gráfico SIG dos critérios analisados no componente JasperReport 60

Lista de Abreviaturas e Siglas

COTS	Commercial, Off-The-Shelf
CARE	COTS-Aware Requirements Engineering
SIG	Softgoal Interdependency Graphs
RNF	Requisito Não-Funcional
ISO	International Standards Organization
ESBC	Engenharia de Software Baseada em Componentes
SUD	Software em Desenvolvimento (Software under development)
ACM	Association for Computing Machinery
IDE	Integrated Development Environment

Sumário

LISTA DE FIGURAS......VI

LISTA DE ABREVIATURAS E SIGLAS......VII

SUMÁRIO......VIII

RESUMO......XI

CAPÍTULO 1......1

Introdução......1

1.1 Contexto......1

1.2 Motivação......6

1.3 Proposta......7

1.4 Contribuição Esperada......9

1.5 Estrutura Do Trabalho......9

CAPÍTULO 2......10

Qualidade Na Engenharia De Software......10

Introdução......10

2.2 Qualidade sob a Perspectiva de Pressman......12

2.3 Qualidade pela perspectiva de Sommerville......13

2.4 Qualidade do Ponto de Vista de Kazman......15

2.4.1 Cenários de Atributos de Qualidade......16

2.4.2 Outros Atributos de Qualidade de um Sistema......18

2.4.2.1 Qualidade para Aplicações Comerciais......19

2.4.2.2 Qualidades Arquiteturais......20

2.5 Qualidade Sob o Ponto de Vista de Chung......20

2.6 Escolha do Método Para o Tratamento de Requisitos Não-Funcionais......22

CAPÍTULO 3......24

Seleção de Componentes na Engenharia de Software Baseada em Componentes (ESBC)......24

Introdução......25

3.2 Metodologias Específicas para a Seleção de Componentes........27
3.3 O Framework Catalysis........29
3.4 O Framework CARE........30
3.5 Conclusões........33
CAPÍTULO 4........34
4.1 Definição da Proposta........34
CAPÍTULO 5........39
5.1 Avaliação da Proposta........39
CAPÍTULO 6........42
6.1 Conclusões........42
6.2 Trabalhos Futuros........42
APÊNDICE A........43
A.1 Artefatos........43
REFERÊNCIAS BIBLIOGRÁFICAS........57

Resumo

A busca por melhores critérios e formas para a seleção de componentes arquiteturais tem sido fonte de inúmeros trabalhos no âmbito acadêmico e empresarial. Através do reuso desses componentes arquiteturais, procura-se possibilitar a entrega de um produto com qualidade, preço adequado e dentro de um tempo determinado. Reutilizar esses componentes pode reduzir o tempo de desenvolvimento, possíveis erros e melhora a maturidade e completude dos produtos gerados. Em um mercado tão abrangente e com componentes desempenhando funcionalidades tão similares, encontrar maneiras de melhorar os critérios de seleção desses componentes se torna um diferencial competitivo para a produção e validação de produtos de software. Estudar formas de análise e possibilitar a seleção de componentes com base em Requisitos Não-Funcionais voltados para aplicações empresariais é o foco deste trabalho.

Palavras-chave: Requisitos Não-Funcionais, Seleção de Componentes Arquiteturais, CARE, NFR Framework

Capítulo 1

Introdução

Neste capítulo, serão desenvolvidos os conceitos preliminares do trabalho. Na seção 1.1, contextualiza-se o escopo do trabalho. Na seção 1.2, relatam-se as motivações que guiaram o surgimento da proposta. Na seção 1.3, define-se a proposta e os elementos essenciais aplicados no trabalho. Na seção 1.4. são apresentadas as contribuições esperadas e na seção 1.5 descreve-se a organização estrutural do trabalho.

1.1 Contexto

A Engenharia de Software evolui constantemente na busca de melhores métodos e ferramentas. Uma das alternativas encontradas é a do reuso de artefatos no processo de desenvolvimento. Das sub-rotinas em 1960, módulos em 1970, objetos em 1980 e componentes em 1990, o desenvolvimento de software tem sido uma história de contínuo crescimento em capacidade e complexidade [Almeida et al, 2007]. Utilizando a definição proposta em [Rational Software, 1998], entendemos por artefato algum tipo de informação criada, modificada ou usada por um processo de desenvolvimento, sendo subprodutos gerados no decorrer do projeto, até sua fase final. Entre os vários tipos de artefatos, estão:

- um modelo como um caso de uso ou modelo de projeto;
- um elemento de modelo tal como uma classe, um caso de uso ou um subsistema;
- um documento, tal como um documento de negócio ou um projeto arquitetural;
- código-fonte,componentes ou código executável (binário).

Tais artefatos, gerados durante o processo de desenvolvimento, têm grande importância para a manutenção do produto final (documentação do projeto), assim como para futuros projetos, já que guardam informações referentes ao desenvolvimento, que são de grande valia para o desenvolvedor ou a equipe de projetistas na evolução de processos, além de código e outros artefatos que podem ser reutilizados, evitando-se assim o retrabalho.

Das diversas metodologias de desenvolvimento vistas em [Pressman, 2001][Sommerville, 2007], todas, de alguma forma, podem gerar artefatos. Por exemplo, a metodologia RUP[Rational Software, 1998], bastante conhecida e utilizada na atualidade, define claramente em seu processo os artefatos produzidos durante o processo de desenvolvimento de software. Já outras metodologias como a XP[Beck, 2005] e SCRUM[Rising e Janoff, 2000], devido a sua natureza ágil, não estabelecem ou obrigam o desenvolvedor a criar artefatos específicos para documentar as fases de desenvolvimento.

Contudo, possibilitar a documentação do processo de desenvolvimento, registrando os quesitos utilizados para tomada de decisão em um nível arquitetural, pode contribuir para a análise e evolução do sistema, pois registram os motivos que levaram o projetista a tomar determinada decisão na época do desenvolvimento do projeto. Durante uma possível revisão do sistema, em tempo futuro, pode-se rever os critério de decisão e aplicar uma nova tecnologia, um novo componente ou um modelo de projeto diferenciado, auxiliando assim no processo evolutivo do software previamente produzido. A possibilidade de melhorar as características do produto de software através da troca de componentes é que propõe o trabalho [Sametinger, 1997]. Segundo Sametinger, um sistema deve possibilitar a evolução contínua, através da troca de componentes, de modo a se adaptar constantemente as mudanças emergentes e não ser totalmente reescrito a cada 10 ou 20 anos.

Seguindo essa tendência natural de evolução, surge um ramo na engenharia de software voltado ao projeto baseado em componentes, a Engenharia de Software Baseada em Componentes (ESBC)[Pressman, 2001][Sommerville, 2007][Crnkovic e Larsson, 2002], que pretende prover aos engenheiros de software modelos para a produção de software baseado em componentes, focando na integração de componentes em detrimento do desenvolvimento integral da aplicação, partindo do zero. De forma breve, o desenvolvimento proposto pela ESBC[Pressman,2001] tem as etapas de (1) seleção de componentes potenciais para reuso, (2) quantificação dos componentes de forma a atender a requisitos arquiteturais, (3) se necessárias, serão aplicadas adaptações arquiteturais para adequação e (4) integração dos componentes em subsistemas de modo a criar a aplicação final. Cabe destacar que devido à grande variedade de definições de componente encontradas na literatura [Almeida et al, 2007] [Ochs el al, 2000][Chung, Cooper e Ramapur, 2005][Sametinger, 1997][Sommerville, 2007], será adotada a definição de componente proposto em [Sametinger, 1997] no qual estabelece-se que “componentes são auto-contidos, sendo artefatos claramente identificáveis, que

descrevem ou realizam uma função específica e tem interfaces claras, documentação apropriada e um status de reuso bem definido.”

A ideia do reuso de componentes está presente na história da computação há muitas décadas. Em 1968, durante uma Conferência da OTAN sobre Engenharia de Software, tem-se o que é considerado o nascimento do campo do reuso de software. O foco foi a Crise do Software da década de 70 e o objetivo desse novo ramo da engenharia de software era possibilitar a criação de softwares de grande escala, confiáveis, funcionais e com tempo e preço bem determinados [Almeida et al, 2007]. Neste contexto, pode-se destacar a definição de reuso na Engenharia de Software apresentada em [Basili e Rombach, 1991], no qual define-se reuso de software “como o uso de tudo que está associado ao projeto de software, incluindo conhecimento”. Essa definição apresenta-se mais coerente com o propósito deste trabalho, pois abrange a proposta de criação de conhecimento para posterior reutilização.

O reuso se mostra atrativo do ponto de vista de projeto por possibilitar, entre outros aspectos [Almeida et al, 2007]:

- produtos de melhor qualidade, por utilizar produtos testados e com algum grau de maturidade;
- produtividade, por reaproveitamento de código presente no componente;
- tempo de entrega reduzido, graças ao reaproveitamento de esforço;
- redução do time de desenvolvimento, por evitar retrabalho;

Reduzir retrabalho, com melhora na qualidade com um cronograma menor são vantagens estratégicas desejáveis por qualquer engenheiro de software. Produtos que se beneficiam da maturidade e da robustez do conjunto de componentes, contribuem positivamente em contextos de ciclos de desenvolvimento cada vez mais curtos.

No entanto, é importante ter em mente os custo associado a reutilização de um componente devido a diversos fatores, tais como: as necessidades de generalidade do componente, projeto arquitetural modular, documentação completa e clara, realização de testes mais abrangentes, necessidade de se mudar a maneira de produzir e arquitetar os projetos de software, custos com treinamento para reuso, entre outros [Almeida et al, 2007]. Outro aspecto também a ser considerado é a necessidade de se abordar o reuso de componentes de forma racional, sendo um fator complexo já que não existe solução universal e as métricas aplicadas não garantem precisão sobre o efetivo ganho do reuso no projeto[Almeida et al, 2007].

Mesmo assim, o reuso no âmbito de componentes aplicado de forma **sistemática** pode atingir os benefícios citados anteriormente [Almeida et al, 2007]. Na ESBC, a escolha de um componente está diretamente relacionada à funcionalidade que o mesmo desempenha. Durante a fase de projeto, o engenheiro de software se depara com funcionalidades mais gerais, tais como persistência de dados, geração de telas, criptografia, transmissão de dados por uma rede de computadores; ou mais restritas ao domínio da aplicação, tais como análise de crédito, geração de pedidos, controle de mercadorias, que podem possuir estrutura dependente do modelo de negócio [Pressman,2001]. No caso de uma funcionalidade mais específica, nem sempre teremos alguma opção disponível na forma de um componente. Porém, no caso de funcionalidades mais gerais, pode haver diversos componentes que desempenham a mesma função. Em um meio com varias opções, se destaca o componente que tenha melhor qualidade. Mas como definir qualidade de um componente? Segundo [Pressman,2001], todos tem uma noção do que nos referimos ao falarmos em qualidade. Porém, como visto na literatura em obras como [Pressman,2001][Sommerville, 2007][Chung et al, 2000][SWEBOK,2004], definir de uma maneira precisa o que é qualidade não é uma tarefa trivial. Na literatura, há diversas maneiras de se ver e representar a qualidade. No trabalho de [Pressman,2001], um software tem qualidade se atende as funcionalidades específicas dentro dos padrões determinados através do teste baseados em métricas. Já no trabalho de [Chung et al, 2000], qualidade é definida não como o que o software faz, mas a maneira como ele realiza suas operações, sendo atribuído a qualidade a designação *Softgoal* e um nível de satisfação através da correlação entre critérios de qualidade. Em [Chung et al, 2000], se propõe uma maneira de representar visualmente essas correlações através de gráficos SIG (*Softgoal Interdependency Graphs*). Em um gráfico SIG, temos as distribuição dos *softgoals* em uma estrutura lógica de árvore, possibilitando a visualização das derivações dos subníveis desses *softgoals*.

Ainda com relação a qualidade, tem-se uma norma produzida pela *International Standards Organization* (ISO) voltada às qualidades que devem ser analisadas em um produto de software. Denominada ISO 9126[ABNT, 2003], essa norma possui um conjunto de qualidades a serem avaliadas em um software. A norma também trás métricas e um modelo de avaliação para a sua aplicação em um processo de avaliação de um produto de software.

No âmbito da avaliação de software, existe uma iniciativa nacional baseada na ISO 9126 e ISO 12119, proposta pelo Centro de Pesquisas Renato Archer (CENPRA), conhecida por

MEDE-PROS. Como explanado no trabalho de [Anjos e Moura, 2005] o propósito da metodologia MEDE-PROS é proporcionar, aos avaliadores, meios para apoiar a avaliação de produtos de software, do ponto de vista do usuário desse produto de software, de acordo com as Normas ISO/IEC 9126 e ISO/IEC 12119, com relação a características de qualidade, documentação e teste dos pacotes de software. Nesse método, o software é avaliado com base em um questionário de quesitos, sendo o quesito avaliado atendido ou não, portanto, sem um grau para determinar a satisfação de um item de qualidade.

Trabalhos acadêmicos, tais como [Alves, Alencar e Castro, 1998][Chung e Cooper, 2004][Colombo, 2004][Anjos e Moura, 2005][Guerra, Colombo e Villalobos, 2005], propõem maneiras ou relatam experiências na avaliação de produtos de software visando o produto final em si, ou realizando a seleção de componentes focado em suas especificidades funcionais. Modelos para a criação de componentes, tais como Entity Java Beans, CORBA e COM+, focam em padrões arquiteturais a serem seguidos na criação de componentes para diversas plataformas, mas não abordam qualidades para esses componentes.

Portanto, há literatura que aborda o desenvolvimento ESBC[Pressman, 2001][Sommerville, 2007][Crnkovic, Larsson, 2002], reuso[Almeida et al, 2007], projeto baseado em componentes[Crnkovic, Larsson, 2002][D'Souza, Wills, 2002], padrões arquiteturais para componentes[Crnkovic, Larsson, 2002], mas não temos um método que nos permita documentar os requisitos funcionais e não-funcionais, de modo a utilizar essa documentação como referência futura, para uma seleção de componentes baseados em critérios levantados pelo engenheiro de software durante a fase de análise de componentes.

Entre os diversos trabalhos estudados[Crnkovic, Larsson, 2002][D'Souza, Wills, 1999][Chung, Cooper e Ramapur, 2005], o que possui maior alinhamento com a é o *COTS- Aware Requeriment Engineering* (CARE) [Chung, Cooper e Ramapur, 2005]. Trata-se de um *Framework* orientado a objetivos e agentes, focado nas fases inicias do projeto, que leva em consideração o uso de componentes durante a fase de projeto. O produto em desenvolvimento é denominado *Software under development* (SUD) e após a análise de requisitos é realizado o processo de seleção em um repositório dos componentes, utilizando-se como critério de seleção o melhor atendimento aos requisitos funcionais e não-funcionais presentes no SUD. Infelizmente, não temos um modelo para a seleção **entre** os componentes, e sim, um modelo guiado pelos requisitos funcionais de um determinado componente e seu efeito no contexto qualitativo geral do SUD. Em um caso onde temos um repositório de

componentes e encontramos componentes que atendam ao mesmo tipo de funcionalidade, teremos que utilizar algum tipo de critério de escolha, podendo ser o tamanho do componente, facilidade de uso, portabilidade, etc.

1.2 Motivação

As fases de desenvolvimento de software encontram-se consolidadas e descritas em trabalhos como [Pressman, 2001][Sommerville, 2007][D'Souza, Wills, 2002][Crnkovic, Larsson, 2002], sendo abordadas por diferentes autores de maneiras diversas. Em um cenário de desenvolvimento, focando na fase de projeto, na qual o projetista se depara com a necessidade de selecionar entre componentes que atendam à uma mesma funcionalidade, como realizar essa seleção de forma racional? Devemos ter em mente a necessidade de documentar os critérios de avaliação para futura referência. Como o reuso desses trabalho pode ser utilizado em uma futura seleção através da confrontação de componentes?

Na literatura, o trabalho mais próximo desse objetivo é o CARE[Chung, Cooper e Ramapur,2005][Chung e Cooper,2004]. Porém, o CARE se limita a realizar uma seleção justamente baseada nas funcionalidades de um suposto Software em Desenvolvimento (SUD), utilizando requisitos não-funcionais muito restritos na análise do componente, não sendo, portanto, uma solução plenamente adequada quando focamos em aplicações empresáriais, já que não avalia os não-funcionais do componente que será utilizado de forma mais completa. Se os componentes se propõe a realizar a mesma tarefa, temos que utilizar critérios de qualidade para a análise da melhor opção. Aplicar a ISO 9126 de forma crua criaria o problema de não definir um mecanismo taxonômico[1] adequado para a documentação do componente, dificultando o reuso dessa documentação, além de não possibilitar nos mostrar os efeitos que a correlação entre os atributos de qualidade presentes em um componentes em particular.

Há a necessidade de se criar um artefato que possibilite documentar o estudo das funcionalidades de um componente, relacionando essas funcionalidades com as qualidades presentes no mesmo, deixando clara as correlações entre os itens de qualidade e como estes são atendidos pelo componente. Com esse tipo de conhecimento, pode-se realizar um estudo entre opções de componentes de modo a verificar qual deles atende melhor os critérios de qualidade necessários pela aplicação de forma mais racional.

[1] **Taxonomia** é aplicada em um sentido de classificação de coisas ou aos princípios subjacentes da classificação.

Em um contexto empresarial, os critérios de avaliação das qualidades em um produto de software se voltam para a Norma ISO 9126 [ABNT,2003], a qual possuindo um conjunto de atributos de qualidade completo, robusto e com credibilidade creditada por uma organização de nível internacional como a ISO. Portanto, a ISO 9126 se torna uma escolha natural para o catálogo de não-funcionais a serem utilizados no estudo de um componente que tenha vocação para ser utilizado em aplicação de cunho empresarial.

1.3 Proposta

A proposta aqui apresentada é uma contribuição para a criação de um artefato que relaciona requisitos funcionais e não-funcionais, de forma a ser usado para a documentação de um componente e posterior seleção de componentes arquiteturais, com funcionalidades muito semelhantes, através da confrontação entre os artefatos de documentação gerados por esta proposta. O artefato gerado é composto da taxonomia funcional estendida para componentes, presente no CARE [Chung, Cooper e Ramapur, 2005] e de um gráfico SIG descrito em [Chung et al, 2000] contendo um catálogo de não-funcionais presentes na Norma ISO 9126 [ABNT, 2003].

Com base nas informações presentes na taxonomia do componente, podemos verificar se o componente atende aos requisitos funcionais especificados na fase de análise de requisitos. Com o gráfico SIG, verificamos quais são os elementos qualitativos presentes no componente de acordo com os atributos de qualidade que a ISO 9126 propõe. Com base nisso, podemos confrontar as informações funcionais e não-funcionais e realizar a seleção de acordo com as necessidades do projeto.

O processo consiste da análise documental do componente para o levantamento das características funcionais e da suas características operacionalizantes com base nos itens da ISO 9126, produzindo-se um catálogo de NFRs, de forma a obter um conjunto robusto e coeso de elementos de qualidade. Usamos a diagramação SIG [Chung et al, 2000], parte integrante do *NFR-Framework* [Chung et al, 2000], de modo a criar um catálogo que possibilite a visualização das correlações entre esses elementos de qualidade e suas operacionalizações, sendo este um dos diferenciais desta proposta para com as demais estudadas. Através dessa abordagem, temos a possibilidade de criação e refinamento de

requisitos para componentes, permitindo ao analista incluir novos quesitos e visualizar de forma mais clara os relacionamentos entre eles.

A abordagem também difere do princípio do próprio *NFR-Framework*, que usualmente analisa o software resultante, gerado pelo processo de desenvolvimento, ou seja, do todo [Chung et al, 2000]. Aqui usamos o método na análise de uma parte delimitada e definida no domínio dos componentes. Mantemos a base da taxonomia do componente arquitetural definida no CARE, possibilitando a interoperabilidade com esse *framework*. Isso possibilita utilizar o CARE para realização da análise de requisitos e projeto de software durante a fase inicial. A taxonomia proposta no CARE, foram adicionados os quesitos de qualidade analisados no componente estudado, e em cada atributo, se relata quais foram os critérios analisados. Essa extensão possibilita uma melhor descrição dos elementos presentes no gráfico SIG, licitando os critérios que o analista utilizou durante a fase de racionalização, documentando de forma mais completa as razões das relações estipuladas. Portanto, pode-se relacionar a análise funcional possibilitada pela taxonomia do CARE, com os atributos qualitativos presentes na ISO 9126, mantendo a interoperabilidade com um método de projeto maduro e bem definido como o CARE. Ao final da análise do componente, temos dois documentos correlatos: um que define os quesitos funcionais e descreve as características qualitativas e o outro um gráfico SIG que nos mostra as correlações entre os *softgoals* aplicados na análise do componente, provenientes da ISO 9126.

Com isso, um documento complementa o outro, gerando o artefato final. Outra vantagem colateral é a possibilidade do uso das regras de correlação num possível processo de métrica para a definição do elemento mais adequado aos anseios do projetista, proposta essa delegada a um possível trabalho futuro. A Figura 1.1 apresenta uma visão geral do processo de criação do artefato.

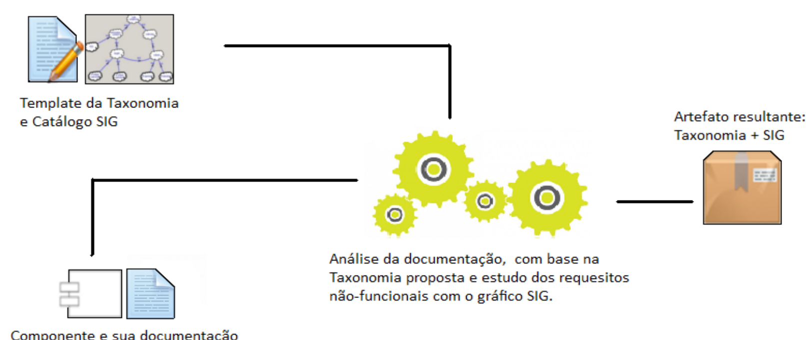


Figura 1.1: Visão resumida da criação do artefato

1.4 Contribuição Esperada

Partindo da carência de artefatos que realizem uma descrição mais completa de um componente, procura-se, através desta proposta, proporcionar ao engenheiro de software um modelo para a documentação dos requisitos funcionais e não-funcionais de um componente, possibilitando a criação de um catálogo de conhecimento que possa auxiliar no estudo de outros componentes, podendo evoluir com novos critérios e atributos, além de prover um artefato para a análise entre componentes de modo racional através da confrontação das características apresentadas nos artefatos produzido durante a análise destes componentes.

1.5 Estrutura Do Trabalho

No Capítulo 2, apresenta-se o conceito de qualidade do ponto de vista da Engenharia de Software. Mostrar-se-á as visões e métodos utilizados no estudo do tema qualidade, com o intuito de apresentar a natureza subjetiva da mesma. No Capítulo 3, estudar-se-á os componentes na Engenharia de Software, apresentando-se como os mesmos são vistos, as metodologias utilizadas para a utilização destes e as propostas de avaliação de componentes presentes na literatura. No Capítulo 4, se definirá a proposta deste trabalho. No Capítulo 5, realizar-se-á uma avaliação da proposta, afim de produzir resultados com relação ao modelo aqui apresentado. No Capítulo 6, conclui-se o trabalho aqui apresentado, evidenciando os pontos positivos e negativos do mesmo, além de apresentação de trabalhos futuros e algumas considerações.

Capítulo 2

Qualidade Na Engenharia De Software

Nesse capítulo, faremos uma revisão com relação a literatura da área, apresentando o conceito de qualidade sob a perspectiva de vários autores. Iniciamos o capítulo com uma introdução (seção 2.1), na qual abordamos de modo geral as diversas formas de tratamento da qualidade presentes na literatura. Na seção 2.2, apresenta-se a visão de qualidade de [Pressman, 2001], na qual qualidade é medida utilizando métricas específicas e testes para a avaliação das funcionalidades esperadas. Na seção 2.3, apresentaremos as ideias sugeridas por [Sommerville, 2007] e como o mesmo diferencia os elementos de qualidade de um produto de software. Na seção 2.4, abordamos a proposta apresentada em [Kazman, Clements e Bass, 2003], sendo essa uma obra das primeiras obras a tratar somente do tema qualidade em nível arquitetural. Na seção 2.5, apresentamos a proposta de [Chung et al, 2000], abordando a qualidade como um tema de natureza subjetiva. Na seção 2.6, concluímos o capítulo com uma breve discussão entre as características da proposta de Kazman e de Chung e a razão pela escolha do *NFR-Framework* para a avaliação dos atributos de qualidade de um componente.

Introdução

Inicialmente, há a necessidade de definir-se de maneira clara, como o tema qualidade é visto na literatura de engenharia de software. Para isso, realizou-se a seleção de trabalhos que abordam o tema, com o objetivo de analisar a proposta mais adequada as necessidades do estudo dos atributos de qualidade de um componente de software. Analisou-se o modo como esses atributos são definidos e apresentados. Também levou-se em consideração se a abordagem proposta, nos trabalhos estudados, possibilita a documentação desses atributos, afim de se reaproveitar essa documentação em posteriores análises de componentes.

A diversidade de visões e as maneiras para se descrever e documentar esses atributos, contribuiu para a discussão do tema qualidade no decorrer deste trabalho. Entre os autores estudados, pode-se verificar que as visões são bastante distintas.

Pressman defende que um software terá qualidade somente se atender os requisitos funcionais e os padrões de projeto definidos por padrões ou preestabelecidos por algum método. Porém, o mesmo autor, também afirma que o não atendimento aos requisitos implícitos torna a qualidade do software duvidosa. Nesse ponto temos um problema, afinal, quais são exatamente esse requisitos implícitos ? A defesa da necessidade da aplicação de métricas é outro ponto controverso. Pressman deixa claro que a engenharia de software deve usar o recurso das métricas para fundamentar a evolução do processo de desenvolvimento e avaliar o produto de software. Porém, esse mesmo autor, também levanta a problemática referente a subjetividade da qualidade e a dificuldade ou impossibilidade de se aplicar métricas de forma efetiva para o mensuramento dos atributos qualitativos.

Sommerville, por outro lado, trata os atributos de qualidade através da designação "Requisitos Não-Funcionais" (RNF). O autor também sugere um conjunto de não-funcionais que, segundo ele, são atributos básicos em um software, dividindo os não-funcionais de modo semelhante ao visto na norma ISO 9126, ou seja, atributos inerentes ao software e atributos emergentes durante a execução, ou internos e externos. Tanto o trabalho [Pressman,2001] quanto [Sommerville, 2007] não descrevem métodos para se trabalhar com RNFs. A contribuição desses trabalhos está na maneira singular de cada um no trato dos RNFs, mostrando que o tema não possui uma visão singular.

As conclusões até esse ponto apontam os RNFs, abordados em obras como [Pressman, 2001][Sommerville, 2007][Chung et al, 2000][Crnkovic, Larsson, 2002][Kazman, Clements e Bass, 2003], como elementos importantes a serem considerados no processo de desenvolvimento e apesar dos modos diversos de tratamento, os autores relacionam o sucesso do software ao atendimento de seus atributos de qualidade. Também deixam a ideia, em alguns casos implicitamente, a natureza da subjetividade dos RNFs, levantando questões importantes relacionadas aos objetivos do trabalho aqui apresentado.

Durante a busca de trabalhos que fornecessem ferramentas para o tratamento de RNFs, dois trabalhos se destacaram: o trabalho realizado por [Chung et al, 2000] e o realizado por [Kazman, Clements e Bass, 2003]. Em ambos os casos, o estudo dos RNF se dá em um contexto arquitetural, estudando maneiras de atender RNFs através de recursos arquiteturais. Em ambos os casos, os autores definem um conjunto de RNFs que devem ser atendidos e sua finalidade.

Apesar da semelhança no propósito, os trabalhos propõe modos distintos de documentação e tratamento dos RNFs. Kazman utiliza os cenários, onde o sistema sofre um estimulo e o resultado desse estimulo é mensurado na tentativa de se avaliar o atendimento ou não do RNF apresentado no cenário em questão. [Chung et al, 2000] utiliza graficos denominados *Softgoal Interdependence graph (SIG),* possibilitando, através de regras de correlação aplicadas aos RNFs e seus elementos operacionalizantes. A seguir, descreve-se o estudo dos trabalhos e, em seguida, realiza-se a justificativa com relação ao modelo escolhido para ser utilizado na proposta aqui apresentada.

2.2 Qualidade sob a Perspectiva de Pressman

No trabalho de [Pressman, 2001], um software tem qualidade se atender plenamente os requisitos funcionais, requisitos de desempenho, padrões de desenvolvimento adequados e as características implícitas esperadas de todo produto de software profissional.

A qualidade de software parte da criação de um conjunto de atividades que ajudarão a garantir que todos os produtos resultantes do processo de engenharia de software atendam requisitos de qualidade bem definidos e explícitos, garantidos através do uso de métricas, desenvolvidas de modo estratégico para melhorar e avaliar o processo de produção e, consequentemente, a qualidade do produto final. A falta de conformidade com os requisitos é definida como falta de qualidade. Após uma análise, os resultados são interpretados afim de se obter informações sobre a qualidade do software e os resultados da interpretação podem levar à modificação do produto, do modelo arquitetural ou do padrão de testes do produto final.

Interessante notar no trabalho do referido autor, a apresentação do conceito de requisitos implícitos de qualidade. No entendimento do autor, há um conjunto de requisitos implícitos que frequentemente não são mencionados (por exemplo, o desejo de facilidade de uso). Se o

software está em conformidade com seus requisitos explícitos, mas não cumpre os requisitos implícitos, a qualidade do software é definida como duvidosa. Ainda de acordo com esse autor, a qualidade do software é uma mistura complexa de fatores que vão variar em diferentes aplicações e clientes que o solicitem.

Durante a discussão relacioanda a qualidade, Pressman cita "Os Fatores de Qualidade de McCall", características que afetam a qualidade do software e podem ser categorizadas em dois grandes grupos: (1) fatores que podem ser medidos diretamente (por exemplo, defeitos por ponto de função) e (2) fatores que podem ser medidos apenas indiretamente (por exemplo, a usabilidade, ou manutenção). Ao concluir, o autor diz ser difícil e, em alguns casos, impossível, se desenvolver medidas diretas destes fatores de qualidade. Como resultado, muitas das métricas definidas no trabalho de McCall et al. são de avaliação subjetiva.

A norma ISO 9126 também é abordada brevemente no trabalho de Pressman. Ele a define como um conjunto de atributos desenvolvidos na tentativa de identificar as principais características de qualidade de um software, não sendo fatores necessariamente utilizados em uma medição direta, mas que no entanto, fornecem uma base válida para as ações indiretas e uma excelente lista de atributos na avaliação da qualidade de um sistema.

2.3 Qualidade pela perspectiva de Sommerville

Sommerville inicia a explanação de tema qualidade questionando: "quais são os atributos de um bom software?". Segundo o autor, o software deve entregar as funcionalidades esperadas com desempenho adequado e deve possibilitar a sua manutenção, bem como ser confiável e fácil de utilizar.

Além das funcionalidades esperadas, um software possui um conjunto de atributos relacionados a qualidade de software. Esses atributos não estão relacionados ao que o software faz e sim com o seu comportamento durante sua execução, assim como a estruturação do código e a documentação relacionada ao mesmo. Um exemplo desses atributos, referenciados na pelo autor como requisitos não-funcionais, é o tempo de resposta resultante de uma pesquisa realizada pelo usuário na base de dados de um programa.

O conjunto esperado desses atributos para um software varia de acordo com o tipo da aplicação. O sistema para uma entidade bancária valoriza a segurança, enquanto jogos de computador tendem a focar no tempo de resposta do programa.

De acordo com o autor, o conjunto das características essenciais de um software bem desenvolvido são as seguintes:

- **Manutenção:** o software deve ser projetado com as mudanças das regras de negócio do cliente em mente. Essa é uma característica crítica, já que as regras de negócio do cliente tendem a mudar com o tempo. Possibilitar a entrega do produto atualizado no menor tempo para o mercado é crucial.
- **Credibilidade:** a credibilidade do software inclui um subconjunto de características tais como confiabilidade, proteção e segurança. No caso de uma falha, um software não pode causar danos a informação ou prejuízo financeiro.
- **Eficiência:** um software não deve utilizar incorretamente memória ou ciclos de processamento em um processo. A eficiência possui o subconjunto de características formado por tempo de resposta, utilização de memória, etc.
- **Usabilidade:** um software deve possibilitar sua utilização sem maiores problemas ou esforços pelo usuário. Isso implica em produzir uma interface gráfica adequada e boa documentação.

Dentre esses RNFs definidos como fundamentais, o autor define alguns requisitos como de natureza emergente. Requisitos não-funcionais emergentes são referentes ao comportamento do sistema em seu ambiente operacional. Exemplos desses RNFs emergentes são confiabilidade, performance, segurança e proteção. Esses são requisitos frequentes para sistemas computacionais. O não atendimento mínimo dessas características pode tornar o sistema impróprio para a utilização do usuário final.

Um sistema que não atenda a todas as funcionalidades esperadas ainda pode ser utilizado por alguns usuários. Porém um sistema com tempo de resposta pobre e de baixa confiabilidade não terá sucesso entre os usuários finais.

Entre os requisitos não-funcionais emergentes, podemos destacar:

- **Tamanho do sistema:** depende diretamente da maneira que os componentes do sistema foram montados.
- **Confiabilidade:** depende diretamente da qualidade dos componentes. A interação entre esses componentes pode gerar falhas, ferindo a confiabilidade do sistema como um todo.
- **Segurança:** a segurança do sistema contra ataques de terceiros não pode ser facilmente aferida. Atacantes podem procurar falhas no sistema que não foram tratadas por mecanismos de segurança.

- **Reparabilidade:** reflete a possibilidade de encontrar um problema do sistema através de seus mecanismos de notificação. Através destes mecanismos é possível localizar o componentes que ocasionou o problema e substituir o mesmo.
- **Usabilidade:** essa propriedade está relacionada à facilidade de utilização do sistema. Bom tempo de resposta e boa interface fazem parte desse item.

Interessante notar o aparecimento do termo "componente". O autor acaba deixando implícito que a qualidade dos componentes afeta a qualidade do produto final. Porém, no trabalho estudado, não temos parâmetro para analisar a qualidade de componentes, temos apenas atributos de qualidade do produto final.

2.4 Qualidade do Ponto de Vista de Kazman

No trabalho de [Kazman, Clements e Bass, 2003], encontra-se um fator de grande importância. Segundo o autor, sistemas são frequentemente redesenhados não porque carecem de funcionalidade, mas sim porque são difíceis de manter, portar, ampliar, estão lentos ou porque foram vitimas de algum ataque que comprometeu a segurança. Nenhum atributo de qualidade é inteiramente dependente do projeto arquitetural, nem da implementação ou da implantação. A arquitetura é um item crítico na satisfação de muitas características de qualidade em um sistema e essas características devem ser analisadas e estudadas no nível arquitetural. Porém, a arquitetura por si só não é capaz de satisfazer essas qualidades. A arquitetura prove a fundação para a satisfação dessas qualidades, mas somente se houver atenção aos detalhes.

Internamente, em sistemas complexos, atributos de qualidade nunca serão satisfeitos isoladamente. A satisfação de qualquer um desses atributos, normalmente, terá efeito positivo ou negativo na satisfação de algum outro atributo. Por exemplo, temos atributos que atuam negativamente em relação ao atributo desempenho. Se o sistema precisa de segurança, haverá o *overhead* do uso de criptografia e validação de transações. Por sua vez, segurança fere a usabilidade, já que o usuário precisa validar uma quantidade maior de dados e operações envolvendo a criptografia de dados aumentam o tempo necessário para acesso aos dados.

Na perspectiva do arquiteto, há três problemas com os atributos de qualidade abordados anteriormente:

1. A definição provida para um atributo não é operacional. É irrelevante dizer que um sistema é modificável. Todo sistema é modificável com relação a um conjunto de mudanças e não é em relação a outro.

2. A subdivisão dos atributos de qualidade não é simples. Uma falha do sistema é um aspecto da disponibilidade, segurança ou usabilidade? Ambos os atributos possuem algum tipo de relação com esse elemento.

3. Comunidades relacionadas ao estudo dos atributos de qualidade possuem seus próprios modos de ver esses atributos. Enquanto a comunidade referente a desempenho visualiza "eventos" ocorrendo no sistema, a de segurança visualiza "ataques" ocorrendo no sistema. Já a comunidade de disponibilidade visualiza "entradas do usuário" no sistema. Todas essas ocorrências são vistas diferentemente pelas diferentes comunidades, mas se referem ao mesmo evento.

Segundo o autor, uma solução para os dois primeiros problemas é o uso de cenários de atributos de qualidade como maneira de caracterizar esses atributos. Uma solução para o terceiro problema é a utilização de uma breve descrição sobre o atributo, focando nas características relevantes para a comunidade desse atributo.

2.4.1 Cenários de Atributos de Qualidade

Um cenário de atributo de qualidade é específico à um determinado atributo de qualidade. Este consiste de seis partes:

1. Fonte de estímulo: o agente que produz o estímulo;

2. Estímulo: o estímulo que chega ao sistema;

3. Ambiente: o estímulo acorre dentro de certas condições. O sistema pode estar sobrecarregado, rodando normalmente ou qualquer outra condição possível;

4. Artefato: o artefato estimulado, podendo ser o sistema inteiro ou uma parte dele;

5. Resposta: a resposta após a chegada do estímulo;

6. Medida da resposta: quando a resposta ao estímulo chega, ela deve possibilitar sua análise para entendimento do resultado do estímulo.

Faz-se a distinção de um cenário de atributos de qualidade geral (cenário geral), por ser um conjunto de atributos que podem pertencer a qualquer sistema computacional Apresenta-se a caracterização dos atributos como uma coleção geral de cenários. Porém, para transcrever um cenário em um item operacionalizante, deve-se fazer uma contextualização específica para o sistema em desenvolvimento. De modo a exemplificar a ideia de cenário, tem-se um exemplos dos elementos que compõe um cenários na Figura 2.1. Na Figura 2.2, tem-se um exemplo de cenário para uso geral e na Figura 2.3, um exemplo de cenário específico.

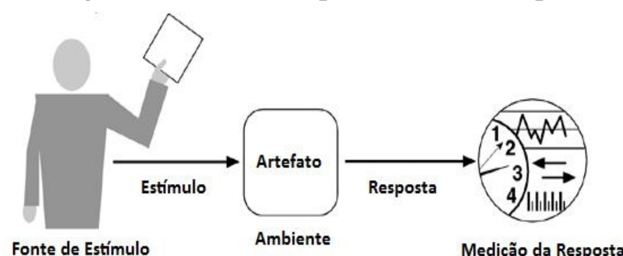


Figura 2.1: Elementos de um cenário

Figura 2.2: Cenário geral de disponibilidade

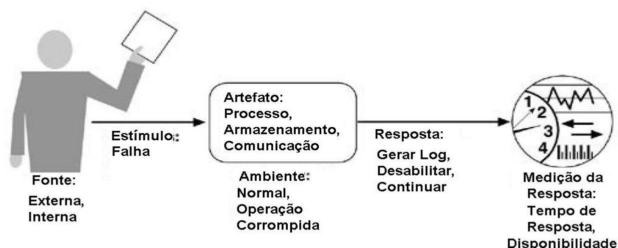


Figura 2.3: Cenário específico de disponibilidade

Lista-se abaixo os seis atributos de qualidade mais comuns e importantes para um sistema, visando possibilitar o entendimento do ponto de vista da comunidade e a criação de um cenário específico. Os seis atributos são:

1. Disponibilidade: diz respeito as falhas do sistema e suas consequências. Uma falha do sistema ocorre quando o sistema não realiza as tarefas especificadas adequadamente. Essa falha é verificável tanto por um usuário humano quanto por um outro sistema computacional;

2. Modificabilidade: diz respeito ao fator custo de uma modificação do sistema;

3. Desempenho: tem relação com o tempo de execução. Eventos (interrupções, mensagens, requisições do usuário) ocorrem durante a execução e o sistema deve responder adequadamente. Existe uma variedade de caracterizações da chegada de um evento e sua resposta e a performance se preocupa com o tempo decorrente da chegada do evento até sua resposta;

4. Segurança: está relacionada a capacidade do sistema resistir a acesso não autorizado enquanto ainda prove seus serviços. Uma tentativa de romper a segurança é denominada de ataque e pode ter diversas formas: varia da tentativa de acesso a um serviço restrito até a indisponibilidade do sistema por parte do atacante;

5. Testabilidade: refere-se a facilidade de se aplicar testes a um sistema e da interpretação dos resultados. Cerca de 40% do custo de um projeto de software estão em testes. Portanto, se um engenheiro de software conseguir reduzir a quantidade de tempo em testes, temos um produto pronto para o mercado em menos tempo;

6. Usabilidade: se refere a facilidade de utilização de um sistema de software pelo usuário para a realização de suas tarefas.

2.4.2 Outros Atributos de Qualidade de um Sistema

Kazman apresenta a descrição de outros atributos em contextos de desenvolvimento distintos. Por exemplo, a escalabilidade é frequentemente um atributo encontrado em sistemas computacionais. Na proposta citada, se discute escalabilidade em termos da capacidade de modificabilidade do sistema. Portabilidade é vista, portanto como a adaptação do sistema a uma diferente plataforma. Se para um determinado projeto, há a necessidade se se tratar uma atributo não presente na obra do autor em forma de cenário, caberá ao projetista realizar essa atividade.

2.4.2.1 Qualidade para Aplicações Comerciais

Kazman define um conjunto de atributos que frequentemente se aplicam a projetos de software comercial. Esse atributos focam em custo, agenda, mercado e quesitos de marketing. Todos os atributos sofrem das ambiguidades que os itens de qualidade geralmente sofrem e devem ser analisados com referência a um projeto de modo a se testar sua aplicabilidade. Abaixo, apresenta-se os atributos de forma geral.

- **Tempo para o mercado:** se o sistema faz parte da estratégia de um parceiro e sua entrega afeta as operações do mesmo, o tempo de mercado se torna fundamental, levando a pressionar o projetista a reutilizar código ou adquirir componentes prontos. O tempo para o mercado geralmente é reduzido com o uso de componentes "Commercial Off-The-Shelf" (COTS) ou elementos reutilizáveis de projetos anteriores. A possibilidade de adaptação depende da possibilidade de reduzir o sistema em partes de acordo com sua funcionalidade.

- **Custo-benefício:** o projeto tem um orçamento que deve ser respeitado. Diferentes arquiteturas levam a diferentes tipos de custo. Por exemplo, existe um trade-off em uma arquitetura baseada em componentes: por um lado, ela se torna mais fácil de manter e atualizar, porém é mais demorada e difícil de se criar.

- **Tempo de vida do sistema:** quando pretende-se criar um sistema com tempo de vida longo, modificabilidade, escalabilidade e portabilidade se tornam importantes. Montar tal estrutura podem comprometer o tempo de mercado, já que arquitetar de modo a atender esses atributos implica em aplicar arquitetura modular e definição de interfaces. Por outro lado, os atributos citados tornam o sistema mais robusto e apto a evoluir com maior facilidade, tornando possível mantê-lo no mercado por mais tempo.

- **Mercado alvo:** para software de uso geral, a plataforma em que o mesmo opera determinará a abrangência de mercado onde o software poderá atuar. Portanto, portabilidade e funcionalidade são pontos chave nesse segmento e atributos como usabilidade, confiabilidade e segurança também tem sua importância.

- **Cronograma de implementação:** a possibilidade de atualizar o sistema com novas funcionalidades lançadas no decorrer do ciclo de vida do software.

- **Integração com sistemas legado:** a possibilidade de integração entre sistemas antigos do cliente é um diferencial importantíssimo por manter a consistência do modelo de negócio do cliente e possibilitar o acesso do cliente a novas funcionalidades através da integração.

2.4.2.2 Qualidade Arquiteturais

Também tem-se um conjunto de atributos de qualidade diretamente ligado a arquitetura do software que são importantes. Descreve-se três delas abaixo:

- **Integridade Conceitual** é um tema subjacente que visa unificar os diversos níveis arquiteturais do sistema. Em nível arquitetural, atividades semelhantes devem ser realizadas de modo semelhante.

- **Corretude e completude** são quesitos correlatos que se direcionam à boa execução do sistema, de modo a atender aos requisitos especificados em fase de projeto.

- **Montagem** diz respeito a capacidade de se produzir o sistema pelo time de desenvolvimento em um tempo determinado e com possibilidade se alteração com base em requisitos emergentes. Refere-se a facilidade de integração entre os módulos para atender necessidades arquiteturais, delegando módulos a times distintos de desenvolvedores, tentando limitar a dependência entre estes módulos. O objetivo é maximizar o desenvolvimento em paralelo da aplicação, tendo em vista que o custo de desenvolvimento tem grande relação com o tempo de produção. No entanto, o atributo "montagem" tem uma natureza complexa. A criação de partes distintas do sistema pode depender de artefatos que são produzidos por ferramentas de produção diversas. Por exemplo, a criação da interface gráfica é realizada em um "Widget Creator" ou ferramenta correlata, que por sua vez é manipulado por uma gerenciador de janelas da aplicação. Portanto o desafio é manter a integração entre o padrão gerado pela ferramenta e o recurso arquitetural da aplicação.

2.5 Qualidade Sob o Ponto de Vista de Chung

Na obra de [Chung et al, 2000], verifica-se que o autor denomina os atributos de qualidade como Requisitos Não-Funcionais. O trabalho em questão define uma metodologia

denominada *NRF-Framework*, possibilitando a criação de catálogos de conhecimento através de gráficos SIG, permitindo o reaproveitamento dessas informações. A obra estudada fornece aos desenvolvedores, através dos gráficos SIG, uma maneira de licitár sistematicamente os RNFs e posteriormente usá-los para guiar o processo de desenvolvimento de software racionalmente. Com uma abordagem orientada a procedimentos, os RNFs podem ser representados como metas a serem atingidas. Uma característica importante é que as metas contribuirão positivamente, negativamente e, muitas vezes, apenas parcialmente, na satisfação de uma determinada meta. Portanto, metas, ou *softgolas,* podem ser satisfeitas, negadas, fracamente satisfeitas, fracamente negadas, conflitantes ou pendentes.

Assim, o termo satisfação da meta sugere o que é esperado do software dentro de limites aceitáveis para os RNFs. Essa abordagem permite que os aspectos de subjetividade, relatividade e interatividade, inerentes aos requisitos não-funcionais, sejam mais compreensíveis. O *NFR-Framework* consiste de cinco componentes principais: *Softgoals*, interdependências, procedimento de avaliação, métodos de refinamento e regras de correlação.

- **Softgoals** são as unidades básicas para representar requisitos não-funcionais e são três os tipos de *softgoals*: *Softgoals* de Requisitos Não-Funcionais representam RNFs a serem satisfeitos, agindo como restrições gerais no sistema;
- **Softgoals de Operacionalização** são técnicas de desenvolvimento que ajudam a satisfazer RNFs, fornecendo mecanismos mais concretos no sistema, operações, processos, representações de dados, estruturação, limitações e agentes no sistema para satisfazer as necessidades declaradas na *softgoals*;
- **Softgoals Declarativas** ajudam a justificar decisões. Justifica e explica a lógica do contexto de uma *softgoal* ou a ligação de interdependência;
- **Interdependências** são interrelações entre os refinamentos de softgoals no sentido da satisfação de *softgoals* relacionadas, existindo três tipos de refinamentos:
- Decomposição é quando uma *softgoal* é refinada em outra *softgoal* da mesma natureza.
- Operacionalização ocorre quando o RNF é refinado em softgoals de operacionalização; isto corresponde a uma transição crucial, onde nos movemos de RNFs (que serão alcançados) para o desenvolvimento de técnicas que podem satisfazer os RNFs.

- Argumentação - Razões de projeto são anotadas através de argumentação. Elas são rastreadas por refinamento, geralmente envolvendo *softgoals*.

Tipos de contribuições referem-se, através do E ou OU (contribuir para suas metas correlatas). Um único descendente pode contribuir para o seu "pai" nos seguintes níveis:

- totalmente positiva ("++" ou *MAKE*);
- parcialmente positiva ("+" ou *HELP*);
- parcialmente negativa (" - " ou *HURT*);
- totalmente negativa ("--" ou *BREAK*);

Um exemplo da diagramação usando um gráfico SIG é apresentado na Figura 2.4. Através das regras de correlação, pode-se determinar o grau de satisfação de um determinado RNF atinge. Os nós da folha do gráfico de interdependência de *softgoals* são rotulados de acordo com seu grau de satisfação: satisfeito, negado, conflitante, fracamente negado, fracamente satisfeito ou pendente, visto na Figura 2.5.

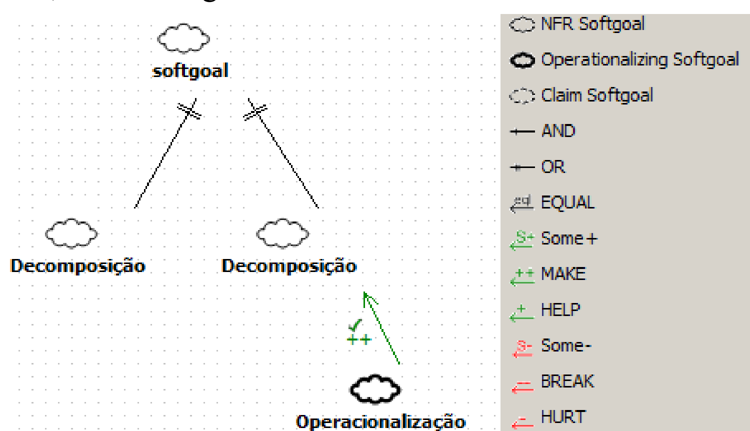


Figura 2.4: Exemplo de Gráfico SIG (S*oftgoal Interdependency Graph*)

Figura 2.5: Tipos de satisfação presentes no *NFR-Framework*: satisfeito, negado, fracamente satisfeito, fracamente negado, conflitante, ou pendente.

2.6 Escolha do Método Para o Tratamento de Requisitos Não-Funcionais

O estudo dos trabalhos de [Pressman,2001] e [Sommerville, 2007], com relação ao seu ponto de vista associado ao tema qualidade, mostra maneiras diversas de tratar o tema na

engenharia de software. A escolha de um método, passa pela racionalização do que se espera como resultado desse processo. Neste sentido, começamos a analisar de que forma estas abordagens poderiam nos auxiliar a estabelecer meios mais efetivos de satisfazer requisitos não-funcionais. Como visto em obras como [Crnkovic, Larsson, 2002][Kazman, Clements e Bass, 2003][Chung el al, 2000], a escolha de um método adequado para auxiliar um engenheiro de software, na satisfação de RNFs, é importante para o bom resultado esperado no produto final. A visão de [Pressman,2001] é importante neste contexto porque aborda a necessidade de se mensurar os resultados, afim de encontrar pontos a serem melhorados através dos resultados obtidos pelas métricas. Porém, [Pressman,2001] cita requisitos implícitos e a importância em atendê-los. O fato de ter-se a necessidade de atendimento de requisitos implícitos cria um problema: se um software só tem qualidade atendendo aos requisitos funcionais e requisitos implícitos, e isso só se verifica através da aplicação de métricas, como aplicar métricas em requisitos que são implícitos e, portanto, não presentes na documentação de requisitos? Se a qualidade tem natureza subjetiva, como aplicar métricas com resultados precisos?

A escolha do método passa pela característica do trabalho aqui apresentado, com forte ligação com o estudo da natureza qualitativa do software a ser analisado. Portanto, o método deve possibilitar a visualização dos atributos de qualidade da maneira mais ampla possível, representando esses atributos, suas derivações, correlações e as operacionalizações entre esses atributos.

Os métodos de [Kazman, Clements e Bass, 2003] e [Chung el al, 2000] tem características semelhantes, tais como a visualização da subjetividade da qualidade, subconjuntos correlatos desses atributos e a geração de catálogos de conhecimento para documentação e posterior reutilização. Porém, o trabalho de Kazman não possibilita visualizar as correlações entre atributos como um todo, restringindo-se aos cenários. A visão do todo é um ponto importante durante uma faze de racionalização devido as inter-relações entre os atributos, como abordado anteriormente nessa seção. Portanto, optou-se por utilizar a proposta de [Chung el al, 2000], que possibilita documentar os RNFs, suas operacionalizações e correlações através de gráficos *SIG*.

Capítulo 3

Seleção de Componentes na Engenharia de Software Baseada em Componentes (ESBC)

Neste capítulo, apresenta-se as opções de seleção de componentes encontradas na literatura de Engenharia de Software. Serão relatados alguns métodos de desenvolvimento que se baseiam no uso de componentes e como esses métodos realizam a seleção de componentes arquiteturais. Na seção 3.1, realiza-se uma breve contextualização relacionada ao desenvolvimento de software baseado em componentes, mostrando as diferenças entre a abordagem tradicional e a proposta pela engenharia de software baseada em componentes (ESBC). Na seção 3.2, apresenta-se um apanhado das metodologias específicas para a seleção de componentes encontradas em trabalhos na área de engenharia de software. A fim de estudar o tema seleção de componentes em metodologias mais completas e consolidadas na área, na seção 3.3, descreve-se o *framework* denominado *Catalysis*, uma metodologia de desenvolvimento que foca na utilização de componentes e *frameworks,* a fim de propiciar o reuso e o rápido desenvolvimento de software com ênfase na documentação do processo. Na seção 3.4, aborda-se o *framework COTS-Aware Requirements Engineering* (CARE), uma metodologia voltada para a documentação das fases de análise de requisito e do projeto de software baseado em componentes. Na seção 3.5, conclui-se com a discussão dos métodos estudados e como a proposta aqui apresentada se diferencia dessas demais.

Introdução

Com o advento das linguagens de Programação Orientadas a Objetos, surge a possibilidade, através de características tais como polimorfismo e herança, de planejar o software visando o reaproveitamento do código de forma mais modularizada. O encapsulamento possibilitou planejar objetos e organizá-los de forma modularizada, permitindo o acesso a métodos e atributos específicos e ocultando os mecanismos de operação do objeto.

O desenvolvimento tradicional atravessa um processo de evolução e agora se depara com a componentização, possível através do programação orientada a objetos, que possibilita o reuso massivo de software e a atualização de aplicações através da troca desses componentes. De acordo com [Stefanuto e Filho, 2007], o desenvolvimento de software, dos sistemas mais simples aos mais complexos, tradicionalmente segue as etapas:

a) Análise dos requisitos do cliente;

b) Especificação do sistema a ser desenvolvido;

c) Aprovação da especificação pelo cliente;

d) Projeto da arquitetura do software, onde alguns desenhos de projeto podem ser reutilizados;

e) Implementação do programa, buscando reutilizar funções já desenvolvidas para outros projetos;

f) Testes;

g) Implantação do sistema no cliente.

Também de acordo com [Stefanuto e Filho, 2007], em um método de desenvolvimento baseado em componentes, temos os seguintes passos:

a) Análise dos requisitos do cliente;

b) Especificação do sistema a ser desenvolvido;

c) Aprovação da especificação pelo cliente;

d) Busca e seleção dos componentes que serão utilizados;

e) Desenvolvimento das partes que não foram atendidas por componentes já existentes;

f) Integração;

g) Testes de integração;

h) Implantação do sistema no cliente.

As diferenças, apesar de sutis, têm impacto marcante no processo de projeto. O primeiro modelo foca na criação da aplicação do zero, com o reaproveitamento ocasional de código. O segundo modelo é fruto da evolução da engenharia de software, a Engenharia de Software Baseada em Componentes (ESBC). Esse segmento da engenharia de software enfatiza, durante o processo de desenvolvimento, a aplicação de componentes previamente desenvolvidos ou a aquisição de componentes produzidos por terceiros, para o atendimento de requisitos levantados da aplicação em desenvolvimento. Na ESBC, se não existir um componente que atenda as funcionalidades especificadas, propõe-se o estudo para o desenvolvimento de um componente que atenda estes requisitos. Esse tipo de abordagem nos direciona a uma maneira diferente de arquitetar a aplicação e exige a reeducação do projetista, sendo esse um dos problemas encontrados no campo do reuso [Almeida et al, 2007].

A incompatibilidade entre componentes é um dos desafios da ESBC. São diversos os modelos para a criação de componentes e diversas as tecnologias nas quais estes se aplicam. Mary Shaw, em uma apresentação no Simpósio de Reusabilidade de Software, em 1995, cita o desenvolvimento com componentes da seguinte maneira: *"é como ter uma banheira cheia de peças da Tinkertoy, Lego, Erector, Lincoln Log, Block City, e outros seis tipos de kits de peças diferentes, os quais atendem funções específicas e deles espera-se que sejam compatíveis*".

Na literatura, obras como [Pressman,2001] e [Sommerville, 2007] trazem um apanhado geral das ideias referentes a ESBC. Trabalhos mais específicos, tais como [Kazman, Clements e Bass, 2003], [Crnkovic e Larsson, 2002], [D'Souza e Wills, 1999] e [Chung et al, 2000], abordam o desenvolvimento baseado em componentes de formas diversas, possibilitando um estudo de confrontação entre as propostas e ideias apresentadas.

O trabalho de [Kazman, Clements e Bass, 2003] dá enfase na busca de elementos arquiteturais que ajudem no atendimento de atributos de qualidade extraídos através de cenários, mas não possibilita um ferramental mais completo para o desenvolvimento de software. Já o trabalho de [Crnkovic e Larsson, 2002] é mais um compêndio referente à ESBC, abordando tópicos referentes a componentes vistos em artigos apresentados a *Association for Computing Machinery* (ACM).

Das obras estudas, [Chung et al, 2000] e [D'Souza e Wills, 1999] trazem um conjunto mais completo de ferramentas de documentação para o desenvolvimento baseado em componentes. Destas, [D'Souza e Wills, 1999] possibilita uma cobertura mais completa do que a proposta

de [Chung et al, 2000]. Porém, [Chung et al, 2000] usa recursos para a análise de requisitos distintos aos de [D'Souza e Wills, 1999], tornando o estudo entre elas de grande interesse. Por serem duas metodologias baseadas em componentes e bem documentadas, realiza-se o estudo delas nas seções 3.3 e 3.4, com o objetivo de entender como ambas tratam a escolha dos componentes.

A seguir, realiza-se a apresentação de metodologias especificas para a seleção de componentes COTS, procurando com isso, até a conclusão do capítulo, deixar mais clara a diferença da proposta aqui apresentada com as demais relatadas.

3.2 Metodologias Específicas para a Seleção de Componentes

Durante o estudo das alternativas para a seleção de componentes, o trabalho que mais se destacou foi o realizado por [George, Fleurquin e Sadou, 2008]. O trabalho refere-se a proposta de um *framework* para a seleção de componentes COTS de forma automatizada e apresenta um estudo das metodologias mais conhecidas para a seleção de componentes COTS mais conhecidas.

Inicialmente, apresenta-se o trabalho na área de seleção de componentes COTS produzido por C. G. En e H. Baraçli. Esse estudo analisou métodos de seleção de componentes diversos e concluiu que a maioria dos processos de seleção segue, no mínimo, as três seguintes fases: definição de critérios de avaliação, priorização dos critérios e avaliação de candidatos COTS de acordo com esses critérios. Normalmente, os processos de seleção usam a Técnica de tomada de decisão baseada em multiplos critérios, ou do inglês, *multi-criteria decision making techniques* (MCDM). As técnicas mais utilizadas baseadas na MCDM são as de *Score* Ponderado, ou WSM e o Processo de Análise Hierárquica, ou AHP, onde se atribui um peso ao critério de acordo com sua possição na árvore de análise.

Dos modelos estudados, temos o OTSO, considerado como um dos primeiros processos de seleção dedicada aos componentes COTS. Além das três fases citadas inicialmente, este adiciona novas etapas, como a pré-seleção de COTS para identificar possíveis candidatos e limitar o número de opções, deixando claro o problema de análise e seleção de muitos candidatos de forma manual. O método PORE é um processo de seleção que defende uma

seleção progressiva. Os candidatos são filtrados e seu número diminui conforme a descrição das necessidades torna-se mais precisa. O médoto DEER destina-se a seleção dos componentes individuais ou conjuntos de componentes, que atendam os requisitos, com foco na minimização dos custos. O método PECA acrescenta uma fase extra: o planeamento da avaliação. Consiste em escolher responsáveis pela avaliação dos componentes e as técnicas que iram utilizar no processo. O método STACE propõe a consideração de critérios "técnico-sociais". Tais critérios podem ser, por exemplo: qualidade do produto, tecnologia do produto, os aspectos do negócio (por exemplo, a reputação do fornecedor, no mercado). O método BAREMO se adapta ao AHP, definindo um conjunto de critérios específicos e sub-critérios dedicada a componentes COTS. O Método COTSRE propõe a criação de catálogos de critérios reutilizáveis. O último método relatado é o PAC. Este propõe critérios específicos não-funcionais inspirado nos atributos de qualidade citados na norma ISO-9126.

Os trabalhos que utilizam a norma ISO 9126, aplicam métricas de algum tipo, ou até mesmo as propostas na norma, na tentativa de realizar a seleção entre opções de componentes COTS. Neste contexto, cabe ressaltar que nosso trabalho não aborda o uso de métricas, por entender que a natureza subjetiva da avaliação dos atributos de qualidade traria problemas de consistência nos resultados gerados por diferentes analistas. Em detrimento das métricas, procurou-se possibilitar a análise e catalogação dos requisitos funcionais através da taxonomia proposta pelo método CARE e os requisitos não-funcionais através dos gráficos SIG propostos pelo *NFR-Framework*, auxiliando a documentação das características do componente e a racionalização no processo de seleção através da comparação entre os requisitos funcionais apresentados na taxonomia e não-funcionais analisados no gráfico SIG. Outro diferencial marcante entre nossa proposta e os trabalhos listados acima é a possibilidade de entendimento das correlações entre os requisitos não-funcionais em vez do estudo isolado de um atributo de qualidade com um critério de checagem métrico ou booleano.

A partir dos métodos vistos, passa-se ao estudo das maneiras de seleção de componentes em metodologias mais completas de desenvolvimento. Na próxima seção, aborda-se como os componentes são escolhidos/selecionados nos *frameworks* de desenvolvimento baseado em componentes *Catalysis* e CARE. Novamente é importante destacar que a escolha destes dois frameworks foi realizada considerando a completude e maturidade dos mesmos comprovada pelos inúmeros casos de utilização apresentados na literatura [Crnkovic, Larsson, 2002] [Sommerville, 2007] [Szyperski, 2002].

3.3 O Framework Catalysis

O *Framework* Catalysis disponibiliza um conjunto de técnicas baseadas em UML para análise do modelo de negócio e desenvolvimento da aplicação baseadas no paradigma da orientação a objetos, possibilitado a análise e o *design* da aplicação. Há a possibilidade de adaptação do grau de detalhes utilizado na produção de artefatos, tornando a utilização do Catalysis tanto em pequenos quanto grandes projetos. Outra característica do Catalysis é o foco no desenvolvimento de aplicações baseado em componentes (DBC), no qual famílias de produtos são criadas utilizando-se kits de componentes, possibilitando o reuso dos artefatos utilizados no processo de *design.* Uma visão dos conceitos pode ser vista nas Figuras 3.1 e 3.2.

A utilização do Catalysis também propicia a especificação de interfaces eliminando ambiguidades, permitindo ao arquiteto da aplicação definir estas interfaces para desenvolvedores terceirizados. Há também técnicas para a definição de conectores entre componentes previamente desenvolvidos e código legado.

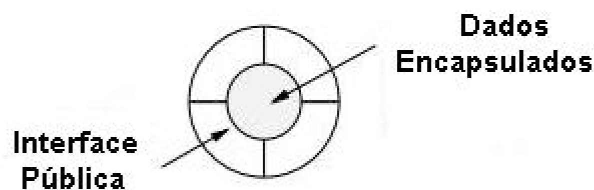


Figura 3.1: Conceito de Objeto/Componente

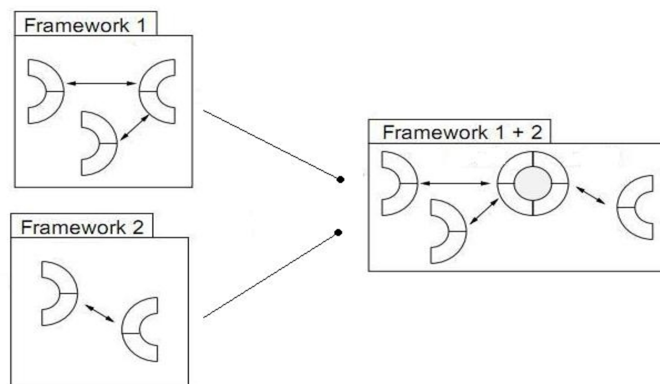


Figura 3.2: Ideia de composição entre *frameworks*

O foco do reuso no Catalysis são os *Frameworks.* No contexto de desenvolvimento DBC, os autores afirmam que *frameworks* são mais reutilizáveis do que objetos, sendo a união entre objetos o meio para a criação de frameworks que atendam as funcionalidades requisitadas na aplicação. Os Padrões de Projeto também são abordados na metodologia, a qual prove mecanismos para sua utilização. Com famílias de produtos baseadas em kits de desenvolvimento, temos dois cenários de desenvolvimento aplicáveis ao modelo DBC. No cenário 1, planeja-se uma série de produtos correlatos, utilizando-se de kits previamente utilizados em outros projetos e que foram produzidos com a finalidade do reuso em mente. No cenário 2, utiliza-se uma arquitetura distribuída, possibilitando rápida readaptação do modelo de negócio. Em suma, a metodologia Catalysis utiliza UML para a documentação das fases de desenvolvimento do projeto, permitindo o reaproveitamento de artefatos de documentação de natureza arquitetural no formato de *templates*, com foco na reutilização e criação de *frameworks* através do acoplamento entre objetos, utilizando a aplicação de padrões de projeto. Porém, ao estudar-se os mecanismos propostos pelo Catalysis, não há um modelo explicitamente definido para a seleção dos componentes que formarão o framework. Fica implícito que o arquiteto sabe quais objetos ele deve utilizar e o Catalysis possibilitará o sucesso no processo de montagem dos *frameworks* e as posteriores ligações entre os mesmos no decorrer do desenvolvimento.

3.4 O Framework CARE

O CARE utiliza um conjunto de metodologias disponíveis atualmente na Engenharia de requisitos, incluindo o RUP[RUP, 1998] , MBASE [Chung, 2005], and ACRE/PORE[George, Fleurquin e Sadou, 2008]. O objetivo deste *Framework* é complementar e estender essas metodologias de modo a atender o desenvolvimento baseado em componentes. O escopo abordado pelo CARE se restringe somente as fases iniciais do desenvolvimento de software, não abordando detalhadamente as fases de *design*, implementação, teste e manutenção.

O *Framework* CARE pode ser caracterizado como orientado a metas/requisitos, além de orientado a agentes, com a utilização de base de conhecimento e com uma método/processo bem definido. Na fase inicial do desenvolvimento do sistema é realizada a definição dos agentes (*stakeholders*) e os requisitos necessários ao sistema em desenvolvimento. A

diagramação utilizada para racionalizar sobre as ações e requisitos de um agente é realizada pelo *Framework* i*.

O mesmo disponibiliza um conjunto de técnicas de modo a auxiliar tanto o Engenheiro de Requisitos, quanto o Arquiteto de software e o engenheiro de componentes, na tarefa de elicitar requisitos, encontrar, avaliar e selecionar componentes, considerando a fase de negociação de mudança de requisitos durante a fase inicial de desenvolvimento. Por exemplo, o framework define que após o levantamento dos requisitos do software em desenvolvimento (SUD - Software under Development) ocorre o mapeamento dos requisitos com os componentes COTS presentes no repositório de componentes. Cabe ao engenheiro de requisitos a tarefa de selecionar e avaliar o componentes usando como critério o grau de satisfação do atendimento aos requisitos presentes no SUD por cada componente. A negociação dos requisitos para adequação dos componentes também cabe ao engenheiro de requisitos. Essa definição de papeis, aliada a integração entre metodologias, torna o CARE uma opção a ser levada em conta para elicitação e documentação da fase inicial do projeto.

O CARE, diferente do framework Catalysis, possui uma abordagem para a seleção entre componentes COTS, baseada no método PORE. O modelo é bem definido e documentado, incluindo as diversas fases na escolha de um componentes de modo a atender os requisitos do SUD. Uma ilustração do processo pode ser vista na Figura 3.3. Observando a referida figura, podemos notar que o repositorio de componentes contém um conjunto de componentes, os quais são formados por dois níveis de abstração: objetivo e especificação do produto. O objetivo fornece uma descrição de alto nível dos requisitos funcionais e não-funcionais do componente. A especificação do produto prove requisitos funcionais e não-funcionais com relação ao sistema em desenvolvimento. Na Figura 3.4 tem-se os passos para a utilização de um pacote COTS em um SUD.

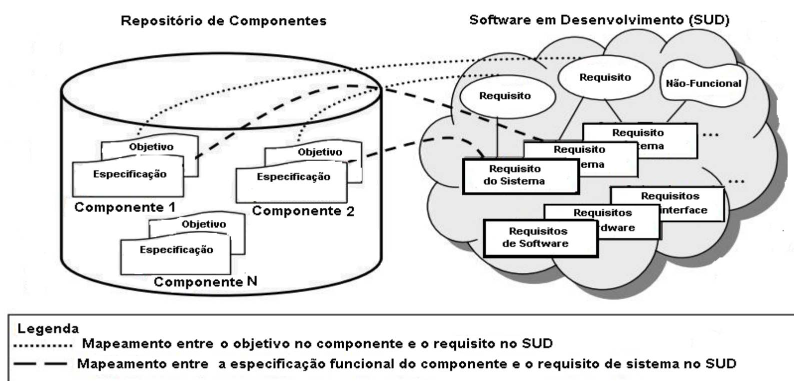


Figura 3.3: Visão de mapeamento entre o repositório e o SUD

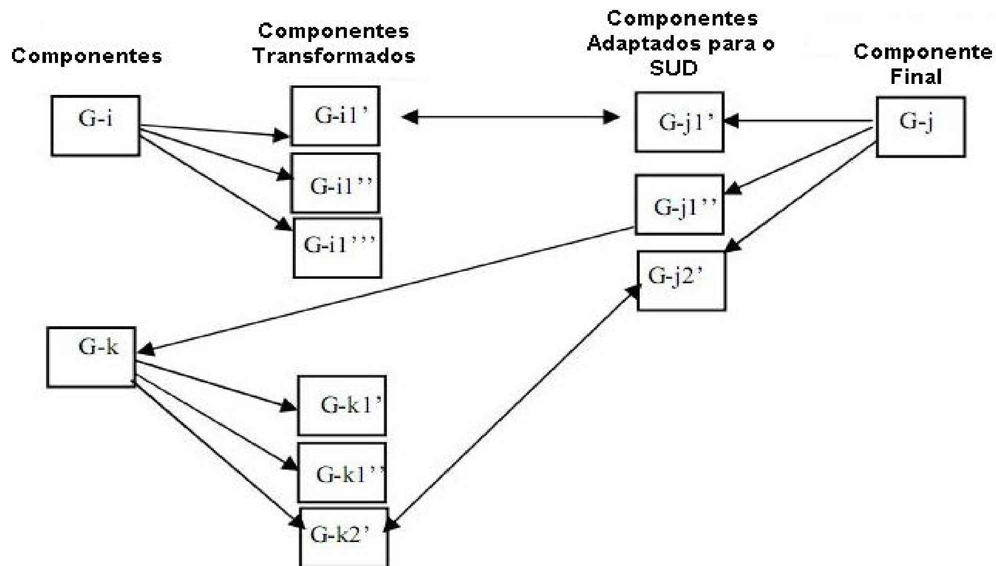


Figura 3.4: Processo para a utilização de um pacote COTS em um SUD

Na descrição do método, leva-se em consideração a análise de requisitos não-funcionais, porém esta não é tão abrangente. Cita-se alguns atributos de qualidade aplicados na taxonomia do componente, vista na figura 3.5, não possibilitando uma clara visualização das relações entre esses atributos. A aplicação do *NFR-Framework* se dá em nível arquitetural, ou seja, um componente atendendo a um requisito não-funcional definido na arquitetura através de um requisito do *stakeholder.*

Component	
Name	
Type	
Name	
Keyword	
Functional Overview	
Domain	
Vendor	
VendorEvaluation	
Version Number	
Functional Specification	
Interface	
OS	
Standards Compliance	
Performance	
Security	
Availability	
Reliability	
Memory	
Disk Space	
Lessons Learned, Experience	
Subcomponents	
Assumptions	
Notes	

Figura 3.5: Taxonomia de Componentes proposta no CARE

3.5 Conclusões

A partir do estudo dos métodos de seleção, verifica-se alguns fatores semelhantes nos métodos vistos. O uso de métricas se mostra importante na busca de um critério mais preciso no processo de seleção. Porém, os métodos estudados aplicam índices ou valores booleanos para a análise das características de pacotes. Em um contexto onde foca-se na qualidade como critério de análise, índices (tais como valores percentuais ou qualquer outro número) não se mostram a melhor alternativa devido a natureza subjetiva da qualidade. Aplicar um valor booleano (tal como verdadeiro/falso ou atende/não atende) acaba sendo muito limitado, por não trazer o grau de satisfação do atributo.

Já no caso dos frameworks vistos para o desenvolvimento a partir de componentes, o Catalysis tem uma visão e objetivo distinto a do CARE. O Catalysis abrange o ciclo de desenvolvimento de maneira mais completa que a do CARE, além de enfatizar a reutilização de frameworks, montados a partir de objetos, como unidades de reuso. Em contra-partida, o CARE tem definido uma fase e método de seleção de componentes baseado tanto em requisitos funcionais quanto em não-funcionais. No caso do CARE, a abordagem dos não funcionais presentes no documento que define o *framework* não é definida na taxonomia proposta na forma de atributos a serem analisados. Infelizmente, falta um catálogo mais completo e robusto, além de um artefato mais adequado para a racionalização referente a esses atributos de qualidade.

Ao final do estudo, verificou-se a necessidade da criação de um artefato para análise de componentes, que permiti-se o estudo dos requisitos não-funcionais do componente, viabilizando visualizar as relações entre os requisitos não-funcionais levantados. Além disso, o processo de criação do artefato deveria possibilitar o reaproveitamento de conhecimento em outras análises de componentes, evitando o retrabalho no levantamento de requisitos no domínio dos componentes.

Com esse critérios em mente, propõe-se na próxima seção, um método para a análise de componentes, visando atender as necessidades encontradas durante a fase de estudo.

Capítulo 4

4.1 Definição da Proposta

A partir do estudo realizado na literatura, pode-se verificar a importância dos requisitos não-funcionais para o sucesso de um software [Pressman, 2001][Sommerville, 2007] [Kazman, Clements e Bass, 2003][Chung et al, 2000]. O estudo também deixa claro a dificuldade em se aplicar métricas em RNFs devido à sua natureza subjetiva. A avaliação de componentes se mostrou diversificada [Chung e Cooper, 2004][George, Fleurquin e Sadou, 2008][Chung, Cooper e Ramapur, 2005]. Das diversas estudadas, a CAP [Ochs et al, 2000] pareceu ser a mais próxima às expectativas iniciais do estudo. Infelizmente, o método CAP tem um alinhamento mais próximo à visão de avaliação da qualidade presente na norma ISO 9126, e como a essa, não apresenta um método para a visualização das relações entre os RNFs em um contexto geral.

Das metodologias de desenvolvimento estudadas, o CARE [Chung, Cooper e Ramapur, 2005] e o Catalysis [D'Souza e Wills, 1999], apenas o CARE apresenta um processo de seleção de componentes bem definido. Em suma, o *framework* CARE possibilita, através da adaptação de sua taxonomia, a criação de um artefato aplicável a uma metodologia de desenvolvimento já documentada e fundamentada. Com base nessa observação, surge a proposta do artefato SIGTAX, formado pela união entre uma versão estendida da taxonomia do *framework* CARE e um gráfico SIG, definido no *NFR-framework*, de modo a aliar requisitos funcionais e RNfs em um artefato. Como catálogo de conhecimento, utiliza-se os atributos presentes na norma ISO 9126, por ser um conjunto de atributos bastante completo e maduro.

A criação do artefato se dá, inicialmente, através do estudo documental do componente a ser avaliado. A análise funcional é licitada através do template de taxonomia estendida do CARE, vista na Figura 4.1. Utilizando-se do catálogo presente na ISO 9126, aplicado ao *NFR-Framework*, temos um template, visto na Figura 4.2, dos atributos a serem avaliados, que possibilita a visualização das relações entre os atributos de qualidade.

1. DEFINIÇÃO DAS FUNCIONALIDADES E CARACTERÍSTICAS DO COMPONENTE
- NOME:
- TIPO:
- KEYWORDS:
- DESCRIÇÃO:
- DOMÍNIO:
- VENDOR:
- TIPO DA LICENÇA:
- NÚMERO DA VERSÃO:
- ESPECIFICAÇÃO FUNCIONAL:
- INTERFACE:
- SISTEMA OPERACIONAL:
- PADRÕES ATENDIDOS:
- DEPENDÊNCIAS:

2. CONCORDÂNCIA COM REQUISITOS DE QUALIDADE

FUNCIONALIDADE
- ADEQUAÇÃO:
- ACURÁCIA:
- INTEROPERABILIDADE:
- SEGURANÇA:

CONFIABILIDADE
- MATURIDADE:
- TOLERÂNCIA A FALHAS:
- RECUPERABILIDADE:

USABILIDADE
- INTELIGIBILIDADE:
- APREENSIBILIDADE:
- OPERACIONALIDADE:
- ATRATIVIDADE:

EFICIÊNCIA
- TEMPO:
- RECURSOS:

MANUTENIBILIDADE
- ANALISABILIDADE:
- MODIFICABILIDADE:
- ESTABILIDADE:
- TESTABILIDADE:

PORTABILIDADE
- ADAPTABILIDADE:
- INSTALAÇÃO:
- COEXISTÊNCIA:
- SUBSTITUIÇÃO:

3. HISTÓRICO DE UTILIZAÇÃO

00/00/0000	

Figura 4.1: Taxonomia estendida para o SIGTAX

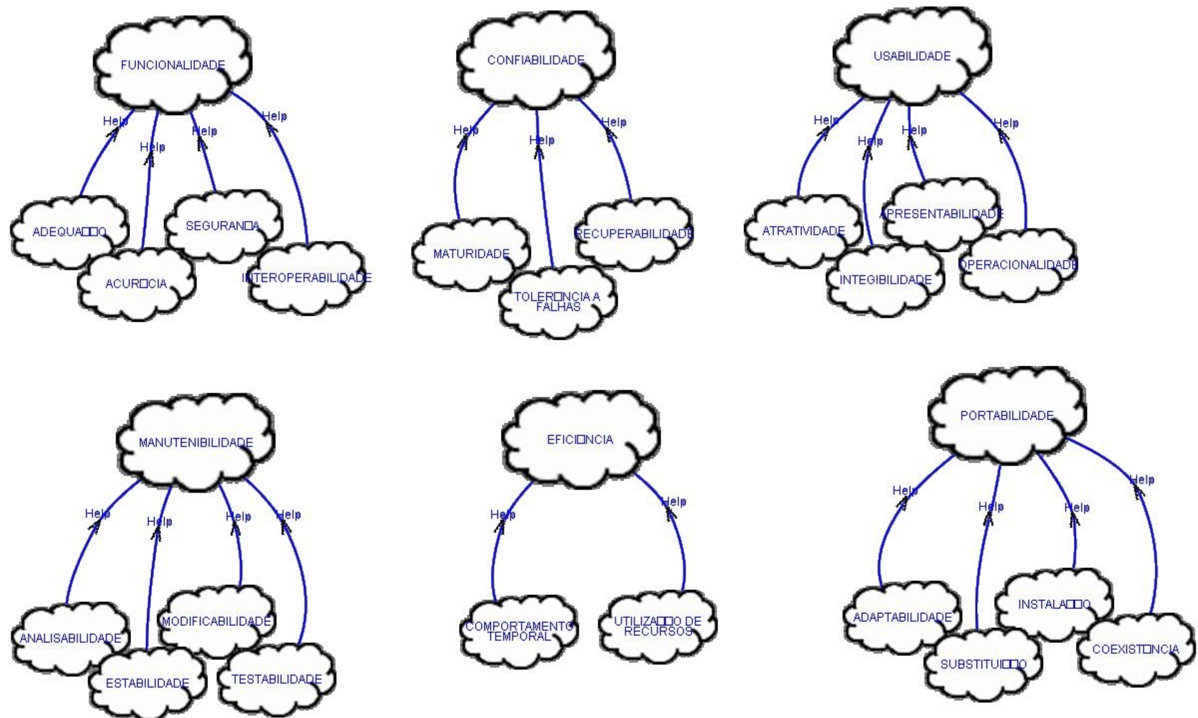


Figura 4.2: Catálogo de atributos ISO 9126 representado através de um gráfico SIG

Importante notar que a escolha da ISO 9126 passou por um processo de racionalização que leva em conta a completude dos requisitos não-funcionais presentes nesse catálogo. Os atributos decisivos na escolha desta norma foram a credibilidade no meio acadêmico e empresarial, a maturidade do catálogo de atributos (devido a revisão realizada em 2003) e a completude do catálogo. Como visto anteriormente, autores como Chung e Kazman propõe catálogos em seus trabalho, porém a norma ISO tem o peso e a credibilidade de uma instituição de normalização internacional com presença em mais de 170 países.

Durante o estudo do componente, o engenheiro de componentes descreve os requisitos funcionais e as características requeridas na taxonomia (o papel de engenheiro de componentes é visto na documentação do *framework* CARE). Conjuntamente, o mesmo realiza a racionalização através do gráfico SIG, aplicando elementos operacionalizantes extraídos da documentação. Ao realiza esse processo, o engenheiro de componentes descreve as relações dos elementos operacionalizantes relacionados ao atributo na seção referente a RNFs presente na taxonomia.

Podemos então definir o seguintes passos para a realização da análise do componente:

1) Levantamento documental do componente: consiste da procura de informações referentes ao componente. Bibliografia, documentação oficial do produtor do componente e fóruns referentes a informações de utilização do componente a ser analisado;

2) Análise da documentação através dos itens dos *templates* de taxonomia e do catálogo de RNFs apresentado no gráfico SIG: consiste no preenchimento dos itens presentes na taxonomia e do processo de relacionamento dos elementos operacionalizantes encontrados durante esse processo de análise documental;

Um fator a ser destacado é a natureza dos RNFs durante a análise. Alguns RNFs tais como desempenho e compatibilidade só poderão ser avaliados de forma efetiva em tempo de execução. Portanto, um piloto deve ser realizado com o componente como forma de validar ou avaliar esses atributos e validar outras características vistas na documentação. Durante esse processo, o engenheiro de componentes, utilizando o gráfico SIG, pode negar alguma característica apresentada na documentação. Importante verificar que graças aos conceitos presentes no *NFR-Framework*, pode-se ter um atributo satisfeito, negado, fracamente satisfeito, fracamente negado, conflitante, ou pendente. Caberá ao arquiteto da aplicação avaliar se esse componente possui elementos suficientes para a utilização na aplicação.

Nesse ponto, emerge um problema referente a precisão dos resultados. Como a avaliação é baseada em atributos qualitativos, e estes têm natureza subjetiva, como garantir que a análise do engenheiro de componentes foi adequada? A experiência do engenheiro de componentes é importante no resultado final da análise. A utilização do catálogo da norma ISO pode contribuir para uma análise mais completa devido a completude do conjunto de atributos. Porém, os elementos operacionalizantes serão levantados pelo engenheiro de componentes. Nesse ponto, a qualidade de análise está diretamente ligada a experiência do engenheiro responsável pela análise. Outro problema está diretamente ligado aos atributos em tempo de execução. O piloto usado na análise pode não atender a toda a gama de casos de desenvolvimento. A própria interação entre os componentes pode gerar casos não previstos durante o piloto, podendo resultar na mudança dos resultados.

Apesar destes problemas, a proposta possibilita a catalogação destes elementos emergentes. A taxonomia e o gráfico SIG fornecem elementos para a documentação destes fatores, através dos relacionamentos entre os atributos e do histórico de utilização do

componente presente na taxonomia. Ao final do processo de análise, o SIGTAX pode ser usado em um processo de seleção entre opções de componentes através da confrontação das características funcionais e não-funcionais. Com as especificações do projeto e as características arquiteturais definidas, a análise dos artefatos SIGTAX dos componentes candidatos é realizada e o que melhor atender aos requisitos da aplicação será escolhido.

A processo de levantamento de requisitos e documentação do processo de desenvolvimento será delegado à metodologia escolhida para esse propósito. A proposta aqui apresentada não entra no mérito do projeto e documentação de projetos de software, se restringindo à documentação de componentes para possibilitar a seleção entre opções de arquiteturais.

Portanto, a proposta não se refere a um *framework* ou um método exclusivo para a seleção de componentes. O objetivo é possibilitar a documentação das características funcionais e não-funcionais de componentes e realizar a seleção entre componentes arquiteturais através da confrontação dos artefatos SIGTAX das opções levantadas. A utilização do catálogo de atributos da norma ISO 9126 agrega um conjunto de requisitos não-funcionais a serem avaliados em aplicações empresariais, sendo um dos diferencias do critério de avaliação desta proposta para com os critérios apresentados no *framework* CARE. Os critérios a serem avaliados no processo de escolha serão provenientes da análise dos requisitos funcionais e não-funcionais da aplicação presentes na aplicação a ser desenvolvida. Os resultados, no entanto, estão diretamente relacionados com a experiência do responsável pelo processo de análise, sendo esse a principal deficiência desta proposta.

Capítulo 5

5.1 Avaliação da Proposta

Para a avaliação da proposta, realizou-se o estudo de três componentes voltados à criação de relatórios. A escolha dos componentes estudados foi guiada pela necessidade da apresentação de informação presente em aplicações empresariais. Para isso, realizou-se a análise de dois componentes: JasperReport 3.7.4 e o BIRT 2.6.1. Os artefatos SigTax resultantes estão disponíveis nos Anexos (Anexo A). Como o esperado, foi possível reutilizar o conhecimento aplicado na taxonomia e no catálogo SIG.

Tanto o componente JasperReport quanto o BIRT possuem funcionalidade e propósitos muito semelhantes. Ambos, porém, possuem uma deficiência que deve ser levada em conta pelo analista: são destinados à uma IDE específica. O BIRT é voltado à IDE Eclipse, em forma de plugin, sendo possível encontra uma versão desta IDE já configurada com o componente BIRT, pronta para a utilização por parte de desenvolvedor. O JasperReport possui um plugin para a IDE NetBeans. Esse fator é fundamental durante a fase de seleção devido a questões de desenvolvimento, implicando numa mudança de IDE por parte do desenvolvedor.

A primeira vista, equipes que utilizam-se da IDE Eclipse utilizarão o BIRT e os que utilizam a IDE NetBeans utilizarão o JasperReport. Essa pode ser uma conclusão rápida, simples e incorreta. Ambos os componentes, apesar de suas funcionalidades semelhantes possuem diferenças que devem ser avaliadas. Se o resultado implicar na utilização de uma IDE distinta da utilizada na atualidade pelos desenvolvedores, esse fator deve ser considerado levando-se em conta a curva de aprendizado necessária para a utilização fluente em cada IDE.

Ao término da análise, o componente BIRT se mostrou mais apto a realizar as tarefas necessárias para a geração de relatórios. Os fatores que mais pesaram a favor do componente BIRT foram a possibilidade de geração dinâmica dos modelos de relatórios e a plena integração e suporte por parte da IDE destinada e este. A análise mais completa pode ser vista no artefato SigTex referente a este componente.

A contribuição do estudo realizado, está na criação de um catálogo SIG inicial, que contém elementos operacionalizantes importantes ao se estudar as relações e características dos componentes, possibilitando a evolução dos critérios na avaliação de componentes baseando-se em experiências anteriores, de forma a garantir uma maior completude da análise realizada. O catálogo SIG inicial pode ser visto abaixo, na Figura 5.1.

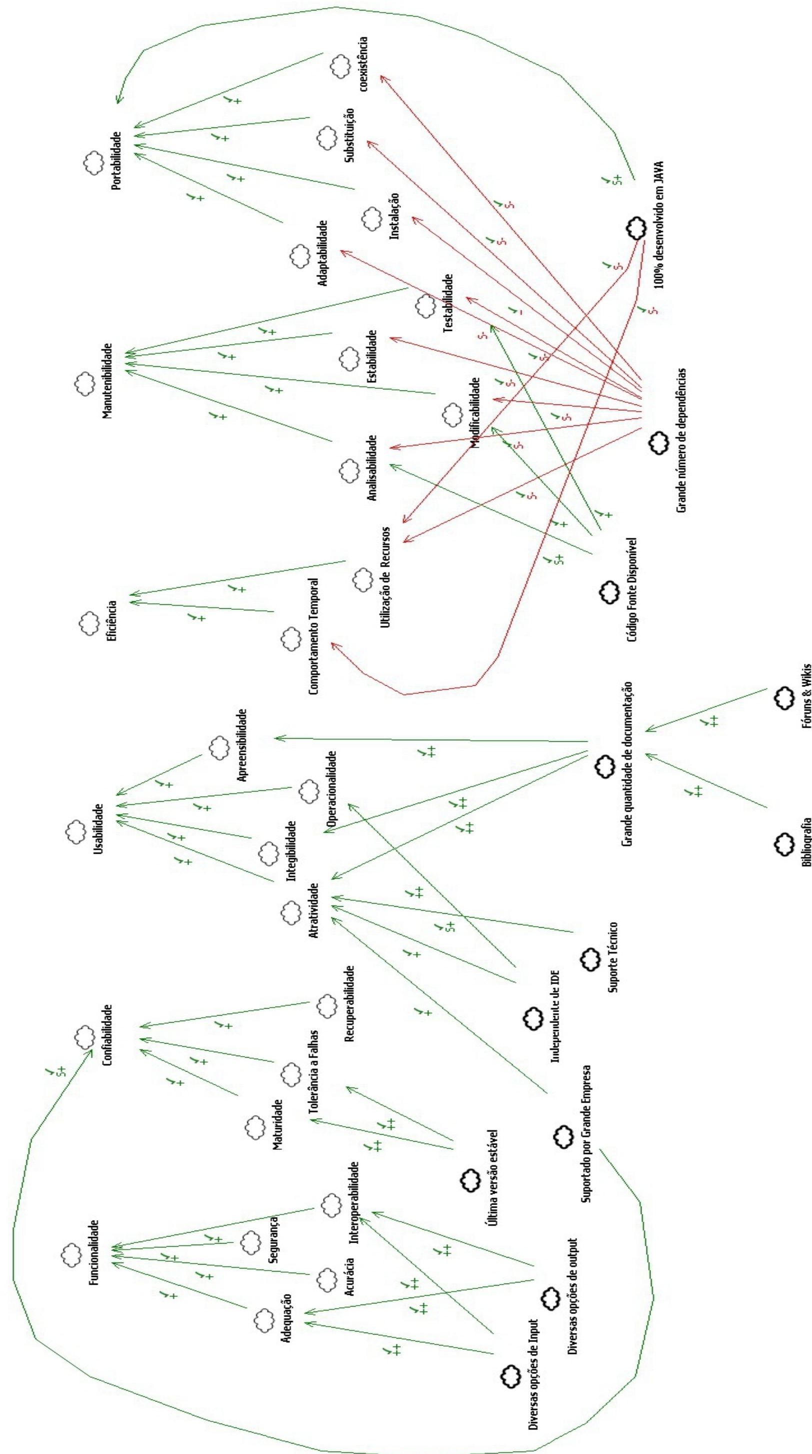


Figura 5.1 Catálogo SIG Inicial específico para componentes

A flexibilidade do *NFR-Framework* também foi importante no decorrer da análise dos componentes, sendo possível incluir novos elementos emergentes da análise documental dos mesmo.

Portanto, a proposta atende aos anseios iniciais de criação de um artefato que relaciona requisitos funcionais e requisitos não-funcionais, possibilitando o reaproveitamento de conhecimento no processo de análise de componentes. O catálogo de atributos presentes na norma ISO 9126 se mostrou bastante completo, levantando questões relevantes no processo de análise de componentes.

Com relação à taxonomia, os elementos listados se mostraram adequados no âmbito dos componentes, não sendo necessário, durante os estudos, a alteração da taxonomia proposta para este trabalho.

Além dos pontos positivos, os pontos negativos também puderam ser avaliados. Ficou evidente o problema gerado pela grande quantidade de atributos em estudo em um gráfico SIG. A quantidade de ligações torna o processo de análise, as vezes, confuso, devido ao grande número de relacionamentos entre os atributos. Esse fato, no entanto, pode ser contornado com criatividade e uma reordenação dos elementos. O maior problema para a aplicação desta proposta fica na grande dificuldade de se encontrar uma boa opção para a criação de gráficos SIG. A aplicação StarUML se mostrou bastante completa, intuitiva e com alguma documentação presente. Porém, a quantidade de *bugs* presentes na aplicação, tornaram a atividade de aplicar as regras de correlação entre os elementos um processo tedioso, com excesso de retrabalho (devido a terminações abruptas da aplicação) e pouco produtivas. A segunda opção encontrada, a OME pecava em um critério básico: internacionalização. Palavras acentuadas eram convertidas em símbolos devido a falta de suporte.

Portanto, há a necessidade de, ao menos, um amadurecimento da aplicação StarUML para que seja possível a utilização dos graficos SIG de forma mais plena (infelizmente o projeto se mostra abandonado, sendo a última atualização realizada em 2008). A simples tarefa de copiar e colar já os elementos do catálogo já seria um grande avanço.

Para o encerramento deste trabalho, apresenta-se no Capítulo 6, as considerações em relação ao trabalho realizado e o que se pretende realizar em trabalhos futuros.

Capítulo 6

6.1 Conclusões

Ao término dos estudos e ao analisar os resultados obtidos, fica claro o quão importante é o estudo de métodos e ferramentas para a análise de componentes. Novos desafios apontam no horizonte com relação ao amadurecimento do processo aqui apresentado, tais como a falta de ferramentas para o trabalho com requisitos não-funcionais, permitindo o uso eficiente do conhecimento produzido. Um maior consenso na abordagem de RNFs e novos estudos para o tratamento dos mesmos também trariam maiores possibilidades na análise, melhorado os resultados obtidos.

Algumas questões ficam sem uma resposta satisfatória, devido à própria natureza do projeto de software baseado em componentes. Como avaliar os requisitos não-funcionais em tempo de execução? A aplicação de um projeto piloto pode não contemplar a plenitude necessária para resultados mais reais. A própria interação entre os componentes pode gerar efeitos adversos na aplicação.

A proposta se mostrou coerente com o que foi encontrado na literatura e a utilização de trabalhos mais maduros teve como objetivo o reaproveitamento de conhecimento, foco da proposta aqui apresentada.

O processo de seleção através da confrontação dos requisitos funcionais e requisito não-funcionais resultou em um método mais completo, a fim de fornecer ao engenheiro de componentes a possibilidade de ver a opção arquitetural com maior completude.

A maior contribuição deste estudo, para o autor deste trabalho, fica nas lições aprendidas quanto ao estudo da qualidade no contexto dos componentes e o papel importante destes no campo do reuso. Como a subjetividade da qualidade torna o processo de análise algo difícil de se mensurar e o quão vasto é a área da qualidade na Engenharia de Software.

6.2 Trabalhos Futuros

Entre os trabalhos futuros pretende-se aplicar a proposta a outros componentes no âmbito de geração de relatórios e outros quesitos de qualidade críticos e comuns a aplicações empresarias, e a abordagem de métricas. Também a partir destes trabalhos pretende-se melhorar a proposta e elaborar um suporte computacional para a mesma.

Apêndice A

A.1 Artefatos

Lista-se abaixo, os artefatos gerados durante a fase de estudo de caso aplicado à proposta. O propósito da apresentação dos artefatos é a de esclarecimento do modelo proposto neste trabalho. Os SigTax resultantes dos componentes BIRT 2.6.1 e JasperReport 3.7.4. sendo primeiramente apresentadas as análises funcionais seguidas dos catálogos de requisitos não-funcionais representados por gráficos SIG.

1. DEFINIÇÃO DAS FUNCIONALIDADES E CARACTERÍSTICAS DO COMPONENTE

- **NOME: Eclipse Business Intelligence and Reporting Tools Project (BIRT)**
- **NÚMERO DA VERSÃO:** 2.6.1
- **TIPO:** Relatórios
- **KEYWORDS:** Geração de Relatórios, BI, ERP
- **DESCRIÇÃO:** O BIRT é um projeto open source baseado no Eclipse, voltado para BI e geração de relatórios, desenvolvido para atender aplicações baseadas na tecnologia Java e JEE. O componente possui uma ferramenta para a modelagem de relatórios utilizando o Eclipse, além de uma engine para a criação de gráficos e códigos de barra.
- **DOMÍNIO:** ERP, BI
- **VENDOR:** Eclipse Foundation
- **TIPO DA LICENÇA:** Eclipse Public License v1.0
- **ESPECIFICAÇÃO FUNCIONAL:**

Principais caracteristicas:

Report Designer Components

- ✔ Common Report Designer Components: Report Editor Palette Data Explorer Property Editor Outline view report structure Report Preview Expression Builder Report Problems Chart Builder Script Editor
- ✔ Sub-reports
- ✔ Side-by-side report components
- ✔ Tables
- ✔ Cross-tabs
- ✔ Sorting Horizontal Panningside
- ✔ Hyperlinks within a report
- ✔ Actionable charts
- ✔ drill-down, hyperlinks, mouse-overs Partial hyperlinks Cascading Style Sheets (CSS controlled format)
- ✔ Conditional Formatting

Formato de Dados:

- Multiple data sources and queries per report and support for joining them
- Support for joining multiple data sources in the Designer
- Report can further re-sort, filter, or group data returned from a query
  - Data Sources Types

    - Database JDBC
    - XML File
    - Web Services
    - Flat files: CSV,SSV,PSV,TSV
    - Custom data source
    - OLAP MDX
- Scripted Data Source: POJO,EJB,Hibernate,XML Stream
- Query Designer
- Graphical Query Designer por prototipagem

Tipos de OutPut:

- Paginated HTML
- Unpaginated HTM
- PDF
- Excel (XLS)
- XML
- Plain Text
- RTF
- Powerpoint (PPT)
- CSV
- Postscript

Tipos de Gráficos:

- Custom Formats
- Geometric shapes & lines
- Barcodes
- Charts
- Chart Wizard
- Chart Interactivity
- mouse-over, tool tips, hyperlinks, etc.
- Chart themes
- Precise control over format of all control elements

- Charts types that all have: 2D 3D Pie Multi-pie Bar Stacked Bar Bar XY Line Line XY Area Area XY Stacked Area Bar Line Bubble Scatter Plot Multi-Axis
- Study Charts
- Tube chart
- Cone chart
- Pyramid
- Time Series
- Meter / Gauge
- Difference
- Radar
- Candlestick / Stock Chart (High/Low)

- **INTERFACES DE INTEROPERABILIDADE:** API, XML, (?);
- **SISTEMA OPERACIONAL:** MS Windows 32bits, Linux 32 bits e Mac OS X 32 bits;
- **PADRÕES ATENDIDOS PELO COMPONENTE :** JEE
- **DEPENDÊNCIAS OPERACIONAIS:**

org.eclipse.birt
org.eclipse.birt.chart.cshelp
org.eclipse.birt.cshelp
org.eclipse.birt.doc
org.eclipse.birt.example
...

2. CONCORDÂNCIA COM REQUISITOS DE QUALIDADE

USABILIDADE

APREENSIBILIDADE

Grande quantidade de documentação: o componente possui boa documentação, apresentando:

- Bibliografia: há alguns livros referentes a utilização do componente;
- Fóruns & Wikis: há fóruns e wikis referentes a utilização do componente por usuários pertencentes a comunidade;

OPERACIONALIDADE

Ferramenta de Modelagem: a produção de relatórios é facilitada por esse recurso;

Modelos de relatórios podem ser alterados em tempo de execução: o componente não possibilita a criação de modelos em tempo de execução, sendo necessário a criação de um
xml através de compilação;
Customização de templates: o componente permite a crianção de novos modelos a partir dos templates presentes por padrão.

ATRATIVIDADE

Grande quantidade de documentação: um componente bem documentado e com opções de referência tanto na literatura quanto na documentação informal criada pela comunidade, possibilita um maior entendimento das funcionalidade e suporte durante o desenvolvimento;

Independente de IDE: o componente em questão possui suporte, como plugin, para a IDE NetBeans, não tendo extensão para Eclipse ou outra IDE conhecida;

Customização de Templates: possibilitar reaproveitamento dos templates para a geração de novos modelos evita o retrabalho;

Suporte Técnico: obter suporte técnico durante a fase de desenvolvimento contribui para a redução no tempo de desenvolvimento (menos tempo gasto na procura de uma solução);

MANUTENIBILIDADE

MODIFICABILIDADE

Modelos de relatórios podem ser alterados em tempo de execução: não possibilitar a criação dinâmica de modelos de relatórios é um dos principais pontos negativos para atingir a modificabilidade;
Código fonte disponível: se necessário for, há a possibilidade de estudo e alteração de características do componente através da alteração do código fonte disponibilizado.

PORTABILIDADE

100% desenvolvido em JAVA: portabilidade possível devido a natureza da tecnologia JAVA;

ADAPTABILIDADE

Código fonte disponível: possibilita a adaptação do componente através da alteração de suas funcionalidades;

Diversos tipos de input: com um grande número de inputs, pode-se alterar um outro componente no projeto, mantendo-se o componente atual;

Diversos tipos de output: com um grande número de inputs, pode-se alterar um outro componente no projeto, mantendo-se o componente atual;

INSTALAÇÃO

Grande número de dependências: dependências podem dificultar a fácil instalação e utilização do componente;

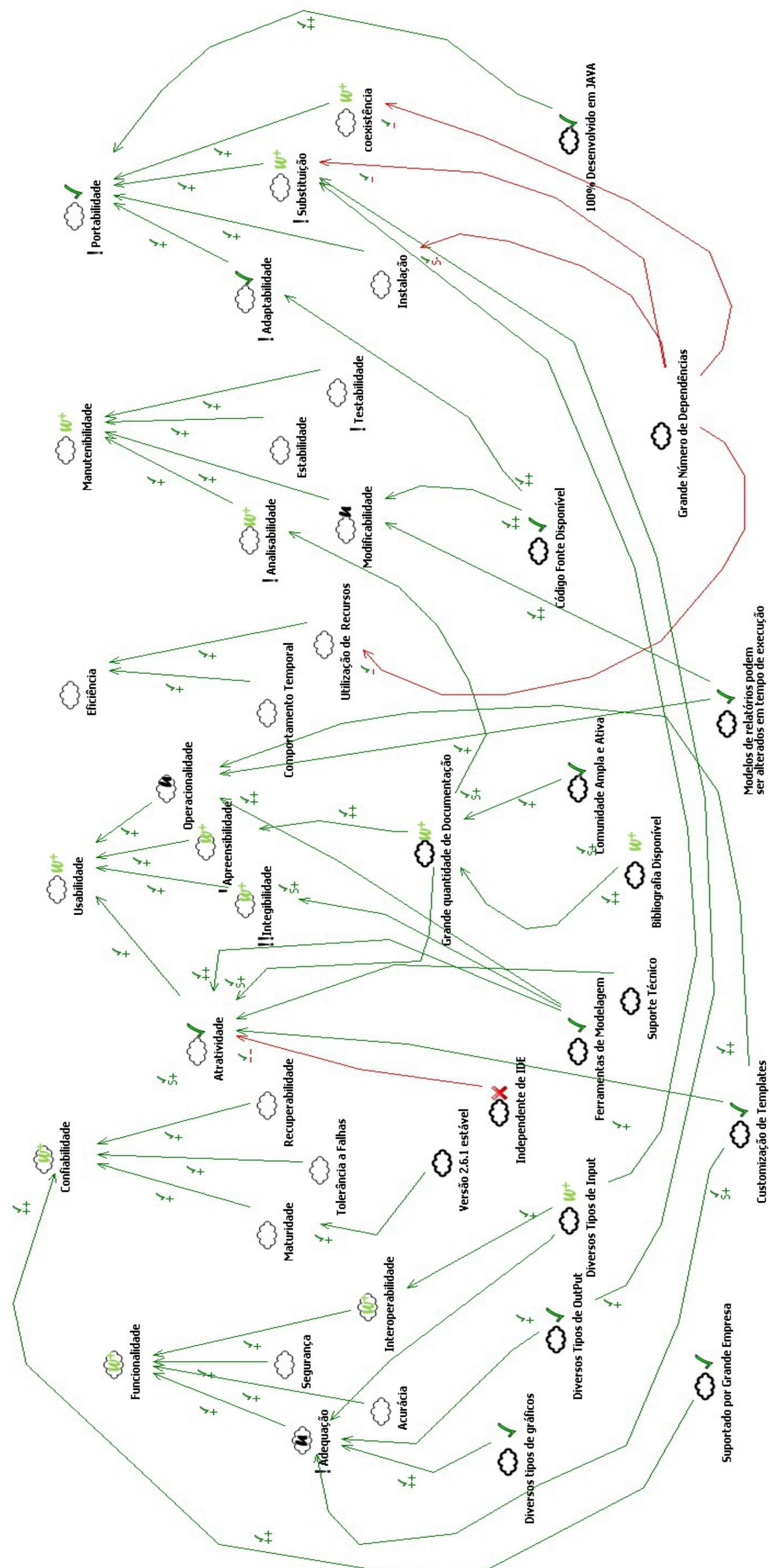
COEXISTÊNCIA

Grande número de dependências: há a possibilidade de ocorrer erros internos devido a conversões de dados, erros referentes a tipos semelhantes, etc ...

SUBSTITUIÇÃO

Grande número de dependências: substituir componentes com um número grande de dependências pode afetar a arquitetura do projeto.

3. HISTÓRICO DE UTILIZAÇÃO



A.1: Gráfico SIG dos critérios analisados

1. DEFINIÇÃO DAS FUNCIONALIDADES E CARACTERÍSTICAS DO COMPONENTE

- **NOME:** JasperReports
- **NÚMERO DA VERSÃO:** 3.7.4
- **TIPO:** Relatórios
- **KEYWORDS:** Geração de Relatórios, BI, ERP
- **DESCRIÇÃO: O** JasperReports é um componente utilizado na geração de relatórios customizados que foi inteiramente escrito em JAVA. O componente possibilita a geração de relatórios através de diversas fontes de dados, produzindo documentos "pixel-perfect" que podem ser vistos, impressos ou exportados em uma variedade de formatos de arquivos, tais como HTML, PDF, Excel, OpenOffice and Word.
- **DOMÍNIO:** ERP, BI
- **VENDOR:** Jaspersoft Corporation
- **TIPO DA LICENÇA:** LGPL(Community) ou Comercial (Professional)
- **ESPECIFICAÇÃO FUNCIONAL:**

Principais caracteristicas:

- ✔ Dashboards, tables, crosstabs, charts and gauges;
- ✔ relatórios com a caracteristicas "pixel-perfect";
- ✔ Arquivos de saída no formato PDF, XML, HTML, CSV, XLS, RTF, TXT;
- ✔ Sub-relatórios facilmente gerados e manipulados;
- ✔ Suporte a código de barras nativo;
- ✔ Rotação visual de texto;
- ✔ Bibliotecas de Estilos para relatórios;
- ✔ Drill-through / hypertext links, incluindo suporte a bookmarks em PDF;
- ✔ Sem limite para o tamanho de relatórios;
- ✔ Impressão condicional;
- ✔ Diversos tipos de base de dados para um relatório;
- ✔ Suporte a internacionalização.

Tipos de Dados Suportados:

- ✔ Database JDBC connection;
- ✔ XML file data source;
- ✔ JavaBeans set data source;
- ✔ Custom JRDataSource;
- ✔ File CSV data source;
- ✔ JRDataSourceProvider;

- ✔ Hibernate connection;
- ✔ Spring loaded Hibernate connection;
- ✔ EJBQL connection;
- ✔ Mondrian OLAP connection;
- ✔ Query Executor mode;
- ✔ Empty data source;
- ✔ Custom iReport connection;
- ✔ XMLA Server Connection.

- **INTERFACES DE INTEROPERABILIDADE:** API, XML, (?);
- **SISTEMA OPERACIONAL:** MS Windows 32bits, Linux 32 bits e Mac OS X 32 bits;
- **PADRÕES ATENDIDOS PELO COMPONENTE :** Dado não fornecido;
- **DEPENDÊNCIAS OPERACIONAIS:**

JRE

- Java Runtime Environment 1.4 or higher

Commons

- Jakarta Commons BeanUtils Component (version 1.7 or later) http://jakarta.apache.org/commons/beanutils/
- Jakarta Commons Collections Component (version 2.1 or later) http://jakarta.apache.org/commons/collections/
- Jakarta Commons Javaflow (Sandbox version) http://jakarta.apache.org/commons/sandbox/javaflow/
- Jakarta Commons Logging Component (version 1.0 or later) http://jakarta.apache.org/commons/logging/

JRXML

- JAXP 1.1 XML Parser
  - Jakarta Commons Digester Component (version 1.7 or later) http://jakarta.apache.org/commons/digester/
  - One of the following for report compilation, depending on the report compiler used:
    - Eclipse JDT Java Compiler (recommended) http://www.eclipse.org/jdt/
    - JDK 1.4 or higher
    - Jikes Compiler http://jikes.sourceforge.net
    - Groovy (version 1.5.5 or later) http://groovy.codehaus.org

- BeanShell (version 2.0 beta 4 or later) http://www.beanshell.org

JDBC

- JDBC 2.0 Driver

PDF

- iText - Free Java-PDF library by Bruno Lowagie and Paulo Soares (version 1.3.1 or later) http://www.lowagie.com/iText/

XLS

- Jakarta POI (version 3.0.1 or later) http://jakarta.apache.org/poi/
- JExcelApi (version 2.6 or later) http://jexcelapi.sourceforge.net/

Charts

- JFreeChart (1.0.0 or later) http://www.jfree.org/jfreechart/
- JCommon - required by JFreeChart http://www.jfree.org/jcommon/
- Batik SVG Toolkit (1.7 or later) - required when rendering charts as SVG http://xmlgraphics.apache.org/batik/

2. CONCORDÂNCIA COM REQUISITOS DE QUALIDADE

1.1.1 FUNCIONALIDADE

ADEQUAÇÃO:

Diversos Tipos de Input: possibilitar adaptar-se com em um número maior de projetos;

Diversos Tipos de OutPut: o componente apresenta diversas opções de formatos de arquivos de saída, tais como HTML, PDF, podendo atender a um maior número de requisitos de output em um sistema.

Diversos Tipos de Modelos de Gráficos: o componente fornece uma grande gama de modelos de gráficos, tais como o de barras, pizza, pontos …

Customização de Templates: há a opção de adaptar templates presentes na biblioteca de estilos pra a criação de novos modelos de relatório.

ACURÁCIA

Não há informações documentais referentes ao atributo;

INTEROPERABILIDADE

Diversos Tipos de Input: com essa característica, é possível utilizar o componentes com uma gama maior de opções arquiteturais.

SEGURANÇA

Não há informações documentais referentes ao atributo;

CONFIABILIDADE

Suportado por Grande Empresa: o componente é suportado pela Eclipse Foundation, a qual é parte de um consórcio de empresas tais como a IBM.

MATURIDADE

Versão 2.6.1 estável

TOLERÂNCIA A FALHAS

Necessita versão piloto para testes;

RECUPERABILIDADE

Necessita versão piloto para testes;

1.1.2 USABILIDADE

INTELIGIBILIDADE

Ferramentas de Modelagem: o componente disponibiliza a possibilidade de utilização de uma ferramenta independente de IDE para a modelagem. Porém, a maneira mais adequada de se trabalhar com este componente é através do plugin específico para a IDE Eclipse;

APREENSIBILIDADE

Grande quantidade de documentação: o componente possui boa documentação, apresentando:

- Bibliografia: há alguns livros referentes a utilização do componente;
- Fóruns & Wikis: há fóruns e wikis referentes a utilização do componente por usuários pertencentes a comunidade;

OPERACIONALIDADE

Ferramenta de Modelagem: a produção de relatórios é facilitada por esse recurso;

Modelos de relatórios podem ser alterados em tempo de execução: o componente permite a criação de modelos em tempo de execução.

Customização de templates: o componente permite a crianção de novos modelos a partir dos templates presentes por padrão.

ATRATIVIDADE

Grande quantidade de documentação: um componente bem documentado e com opções de referência tanto na literatura quanto na documentação informal criada pela comunidade, possibilita um maior entendimento das funcionalidade e suporte durante o desenvolvimento;

Independente de IDE: o componente em questão possui suporte, como plugin, para a IDE Eclipse, não existindo extensão para NetBeans ou outra IDE conhecida;

Customização de Templates: possibilitar reaproveitamento dos templates para a geração de novos modelos evita o retrabalho;

Suporte Técnico: obter suporte técnico durante a fase de desenvolvimento contribui para a redução no tempo de desenvolvimento (menos tempo gasto na procura de uma solução);

EFICIÊNCIA

TEMPO

Necessita versão piloto para testes;

RECURSOS

Grande número de dependências: para a utilização das funcionalidades presentes no componente, o mesmo utiliza-se de outros componentes, que por sua vez, podem não ser necessários para a aplicação em si.

MANUTENIBILIDADE

ANALISABILIDADE

Grande quantidade de documentação: através da documentação, fóruns e wikis do componente, pode-se obter uma melhor compreensão do que é necessário para atender determinada funcionalidade ou solucionar um problema;

MODIFICABILIDADE

Modelos de relatórios podem ser alterados em tempo de execução: possibilitar a criação dinâmica de modelos de relatórios é um dos principais pontos positivos para atingir a modificabilidade;

Código fonte disponível: se necessário for, há a possibilidade de estudo e alteração de características do componente através da alteração do código fonte disponibilizado.

ESTABILIDADE

Necessita versão piloto para testes;

TESTABILIDADE

Necessita versão piloto para testes;

PORTABILIDADE

100% desenvolvido em JAVA: portabilidade possível devido a natureza da tecnologia JAVA;

ADAPTABILIDADE

Código fonte disponível: possibilita a adaptação do componente através da alteração de suas funcionalidades;

Diversos tipos de input: com um grande número de inputs, pode-se alterar um outro componente no projeto, mantendo-se o componente atual;

Diversos tipos de output: com um grande número de inputs, pode-se alterar um outro componente no projeto, mantendo-se o componente atual;

INSTALAÇÃO

Grande número de dependências: dependências podem dificultar a fácil instalação e utilização do componente;

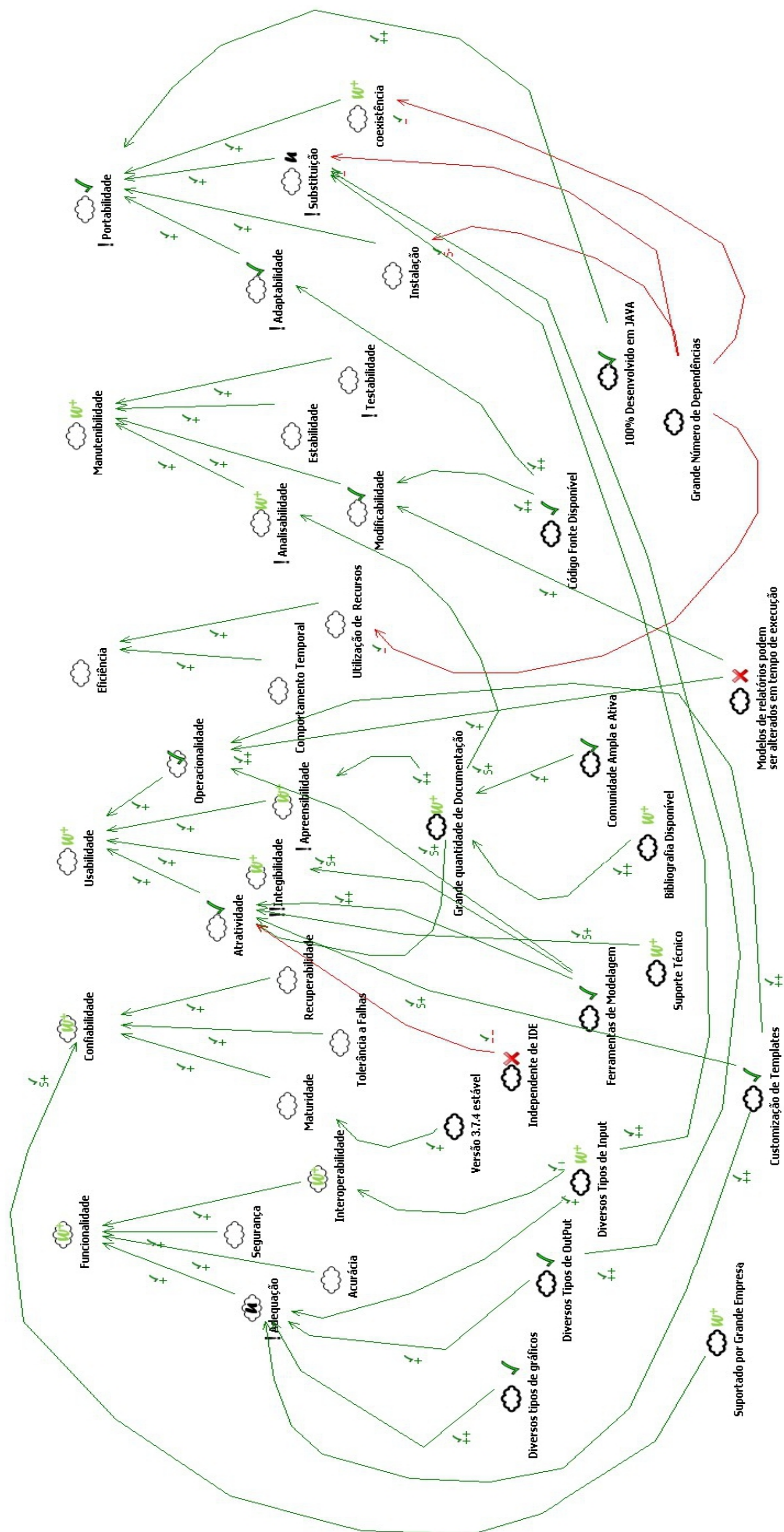
COEXISTÊNCIA

Grande número de dependências: há a possibilidade de ocorrer erros internos devido a conversões de dados, erros referentes a tipos semelhantes, etc ...

SUBSTITUIÇÃO

Grande número de dependências: substituir componentes com um número grande de dependências pode afetar a arquitetura do projeto.

3. HISTÓRICO DE UTILIZAÇÃO



A.2: Gráfico SIG dos critérios analisados

Referências Bibliográficas

[1] ABNT, NBR ISO/IEC 9126-1 Engenharia de software - Qualidade de produto Parte 1: Modelo de Qualidade, ABNT – Associação Brasileira de Normas Técnicas (2003).

[2] ALMEIDA, E.S., et. al. “C.R.U.I.S.E: Component Reuse in Software Engineering”, C.E.S.A.R e-book, Brazil, (2007).

[3] ALVES,C.F., ALENCAR,F.M.R., CASTRO,J.F.B., Requirements Engineering for COTS Selection.”(1998).

[4] ANJOS, L. A., MOURA, H. P.,”Um Modelo para Avaliação de Produtos de Software”, 2005,php.-cin.ufpe.br/~laps/laps/arquivo/arquivo_13.pdf, acessado em 05/10/2010

[5] IEEE, “Guide to the Software Engineering Body of Knowledge - SWEBOK”, Wiley-IEEE Computer Society Press,2004

[6] Basili,V. R., Rombach, H. D., Support for Comprehensive Reuse, SOFTWARE ENGINEERING JOURNAL, British Computer Society, Julho/1991

[7] Beck, K., Extreme Programming Explained: Embrace Change. Addison-Wesley, 2nd edition (2005)

[8] Chung, L., Nixon, B. A., Yu, E., Mylopoulos, J., “Non-Functional Requirements in Software Engineering”, Kluwer Publishing, (2000).

[9] Chung,L., Cooper,K., "Matching, Ranking, and Selecting Components: A COTSAware Re-quirements Engineering and Software Architecting Approach", Int. Workshop on Models and Processes for the Evaluation of COTS Components, co-located with ICSE, (2004).

[10] Chung, L., Cooper, K., Ramapur, C., “A COTS-Aware Requirements Engineering and Architecting Approach: Defining System Level Agents, Goals, Requirements and Architecture Ver. IV” Department of Computer Science ofThe University of Texas at Dallas, Dezembro, (2005).

[11] Colombo,R.M., “Processo de Avaliação da Qualidade de Pacotes de Software”, Monografia, FEM/UNICAMP, Campinas, 17 de Maio de 2004.

[12] Crnkovic, I., Larsson, M., "Building reliable component-based software systems" ARTECH HOUSE, INC. 2002

[13] D'Souza, D. F., Wills A. C., "Objects, components, and frameworks with UML : the Catalysis approach" Addison Wesley Longman 1999

[14] Garland, J., Anthony R., A "Large-Scale Software Architecture - Practical Guide using UML", John Wiley & Sons Ltd, 2003

[15] George, B., Fleurquin, R., Sadou, S., A Component Selection Framework for COTS. In: 11th International Symposium on Component-Based Software Engineering, Karlsruhe, Germany (2008) p. 286-301

[16] Guerra, A. C., Colombo, R. T., Villalobos, M. T., "Processo De Avaliação De Produtos De Software", Conferência IADIS Ibero-Americana WWW/Internet (2005).

[17] Kazman, R., Clements, P., Bass, L., "Software Architecture in Practice", Segunda Edição, Ed. Addison Wesley Abril 11, (2003).

[18] Pressman, R. S. Software engineering: a practitioner's approach, 5th ed. McGraw-Hill, 2001

[19] Ochs, M., Pfahl, D., Chrobok-Diening, G., Nothelfer-Kolb, B.: A COTS acquisition process: Definition and application experience. In: Proceedings of the 11th European Software Control and Metrics Conference (ESCOM), 2000, pg. 335–343

[20] Rational Unified Process Best Practices for Software Development Teams , Rational Software, Cupertino, CA, Rev. 11/98

[21] Rising, L., Janoff, N. The Scrum Software Development Process for Small Teams, IEEE SOFTWARE JOURNAL, Volume: 17 Issue:4, July/August 2000, p 26 – 32, IEEE Computer Society

[22] Sametinger, J., "Software Engineering with Reusable Components", Springer-Verlag, Março, 1997

[23] Sommerville, I., "Engenharia de Software", 8a Edição, Addison Wesley/Pearson, 2007

[24] Stefanuto, G., Filho, S., "Perspectivas de desenvolvimento e uso de componentes na Indústria Brasileira de Software e Serviços", SOFTEX, Campinas: SOFTEX, 2007

[25] Szyperski, C., Murer, G., "Component Software: Beyond Object-Oriented Programming" , Second Edition, Addison-Wesley, 2002