2
Engenharia de Requisitos

Um dos principais objetivos da engenharia de requisitos é melhorar a modelagem de sistemas e a capacidade de analisá-los, possibilitando maior entendimento de suas características antes da implementação (Robinson 2003). É seu papel realizar a interação entre requisitantes[4] e desenvolvedores, entre "o que" deve ser feito e "como" deve ser feito. É necessário nesta etapa, elicitar, analisar conflitos, validar, priorizar, modificar e reusar requisitos, rastreá-los considerando sua origem, os componentes arquiteturais e o código que os implementa, dentre outras tarefas.

A reutilização, evolução e rastreabilidade de requisitos estão intimamente relacionados à habilidade de gerenciar interações entre requisitos, que por sua vez, está relacionada à habilidade de separar e compor características, representando-as em modelos. Visto que, geralmente, um único tipo de modelo não é suficiente para explicitar todas as características do sistema, diferentes modelos são utilizados, tornando as informações espalhadas e entrelaçadas, também, em diferentes visões. Em engenharia de requisitos o uso de visões é extremamente importante porque com este artifício consegue-se representar e visualizar os requisitos, focando as características de interesse a cada momento do processo. O uso de visões também é uma maneira de abordar a separação de características.

Assim, a engenharia de requisitos tem abordado o princípio de separação de características por meio de métodos de modelagem em conjunto com o uso de visões, com o intuito de prover facilidades para o reuso, rastreabilidade e evolução. Na Seção 2.1, resumimos o processo de definição de requisitos. Na Seção 2.2, apresentamos a modelagem de requisitos e mostramos a presença dos problemas de espalhamento e entrelaçamento. Na Seção 2.3, apresentamos como visões são utilizadas para separação de características. Na Seção 2.4, 2.5 e 2.6 definimos, respectivamente, os conceitos de evolução, rastreabilidade e reuso.

[4] O termo requisitante está sendo utilizado para representar as pessoas que requisitam serviços ou impõem restrições, tais como usuários e clientes.

2.1. Processo

O processo de definição de requisitos pode ser definido, resumidamente, por três atividades: elicitação, modelagem e análise (Leite, 1988), veja Figura 3. Geralmente, estas atividades ocorrem simultânea e incrementalmente, num processo evolutivo que dura todo o processo de desenvolvimento de software (Jacobson, 1999).

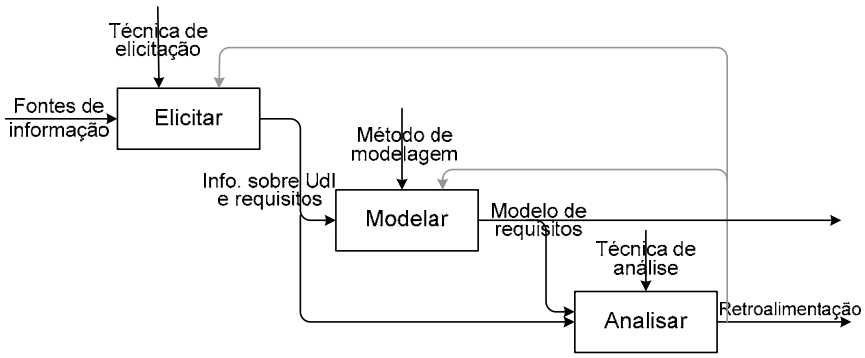


Figura 3. Processo de engenharia de requisitos

- Elicitação – o engenheiro de requisitos utiliza alguma ou um conjunto de técnicas para descobrir (elicitar) os requisitos do sistema a ser desenvolvido, incluindo as informações do UdI (universo de informação) que restringem este sistema. Algumas das técnicas mais conhecidas estão relacionadas a técnicas de leitura de documentos, observação, entrevistas, reuniões, questionários, antropologia, participação ativa dos atores e engenharia reversa (Goguen, 1994).
- Modelagem – as informações obtidas durante a elicitação são registradas e organizadas em modelos de requisitos tais como, casos de uso (Jacobson, 1992), cenários e léxico (Leite, 1997), entidade relacionamento (Chen, 1976), SADT (Ross, 1977), DFD (Marco, 1979), dentre outros. A construção destes modelos exige maior aprofundamento no conhecimento sobre o UdI, sobre as necessidades dos usuários e também informações técnicas. Isto remete à atividade de análise, sendo necessário analisar as informações registradas para descobrir erros e omissões, sendo muitas vezes necessário retornar à elicitação para esclarecer, acrescentar ou corrigir o conhecimento adquirido.
- Análise – além da análise de erros e omissões o processo de definição de requisitos possibilita a análise sob diferentes perspectivas tais como,

viabilidade, custo, tempo, prioridades, reuso, completude, corretude, variabilidade, evolução, dentre outras.

As atividades de elicitação e análise estão fora do escopo desta tese. Porém, no Capítulo 4, mostramos qual o impacto de nossa abordagem nestas atividades.

2.2. Modelagem

A modelagem de requisitos é a atividade central no processo de engenharia de requisitos porque dela resulta o produto principal deste processo: o documento de requisitos, o documento no qual os desenvolvedores devem se basear para construir a arquitetura e o código do sistema (Jacobson, 1999; Sommerville, 2000; Kotonya, 1998b). Vários modelos são necessários para compor o documento de requisitos porque cada um deles enfatiza apenas parte dos aspectos necessários para que os desenvolvedores entendam o que o sistema sendo construído deve prover e o seu contexto.

Requisitos são sentenças que indicam as necessidades dos interessados (Kotonya, 1998b). Eles são, geralmente, categorizados como: requisitos funcionais, i.e, aqueles que representam a funcionalidade do sistema; e não-funcionais, i.e, aqueles que restringem os requisitos funcionais. Eles podem estar associados a diferentes dados, serviços, *features*, restrições, níveis de abstração, representações, dentre outros, veja Figura 4.

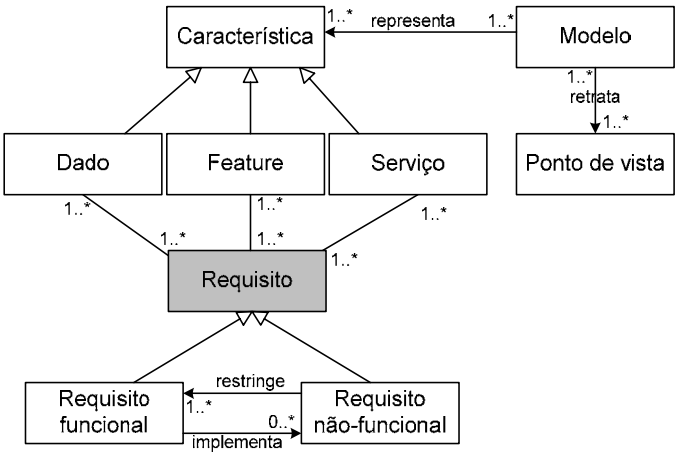


Figura 4. Relação entre requisitos e características

Em (IEEE Std. 830; Sommerville, 2000; Volere, 2006) são definidos *templates* para o documento de requisitos. Eles apresentam estruturas para organizar os requisitos no documento, veja na Figura 5. De maneira geral, um

documento de requisitos deve definir, no mínimo, um glossário de termos do UdI (em cinza claro na Figura 5) e uma lista de sentenças de requisitos, geralmente, em linguagem natural não estruturada e organizada de diferentes maneiras (em cinza escuro na Figura 5).

Volere (2006)	**IEEE Std. 830**
PROJETO 1. O propósito do produto 2. Clientes, fregueses e outros interessados 3. Usuários do produto RESTRIÇÕES DO PROJETO 4. Restrições obrigatórias 5. Convenções para nomes e definições 6. Fatos relevantes e suposições REQUISITOS FUNCIONAIS 7. O escopo do trabalho 8. O escopo do produto 9. Requsitos funcionais e de dados REQUISITOS NÃO-FUNCIONAIS 10. Requisitos de amigabilidade 11. Requisitos de usabilidade 12. Requisitos de performance 13. Requisitos operacionais 14. Requisitos de manutenibilidade e portabilidade 15. Requisitos de segurança 16. Requisitos culturais e políticos 17. Requisitos legais ASSUNTOS DE PROJETO 18. Assuntos em aberto 19. Soluções de prateleira 20. Novos problemas 21. Tarefas 22. Agenda 23. Riscos 24. Custos 25. Documentação para o usuário e treinamento 26. Requisitos para as próximas versões 27. Idéias para soluções	1. Introdução 1.1 Propósito 1.2 Escopo 1.3 Definições, acronimos e abreviações 1.4 Referencias 1.5 Visão geral 2. Descrição global 2.1 Perspectivas do produto 2.2 Funções do produto 2.3 Características dos usuários 2.4 Restrições 2.5 Suposições e dependências 3. Requisitos específicos Apêndices
	Sommerville (2005)
	1. Prefácio 2. Introdução 3. Glossario 4. Requisitos do usuário 5. Arquitetura do sistema 6. Requisitos dos sistema 7. Modelos do sistema 8. Evolução do sistema 9. Apêndices 10. Indice

Figura 5. Exemplos de templates para documentos de requisitos

Espalhamento e Entrelaçamento de Características

Podemos considerar que ambos, glossário e lista, são modelos de requisitos. No primeiro, enfatizam-se dados e restrições a eles, no segundo enfatizam-se serviços, restrições e exceções. Informações sobre os dados estão detalhadas no primeiro modelo e aparecem no segundo modelo como recursos ou atores dos serviços modelados. Enquanto informações sobre os serviços estão detalhadas no segundo modelo que muitas vezes omite detalhes sobre os dados, sendo que, algumas restrições impostas aos serviços são restrições decorrentes de restrições aos dados. Nenhum dos dois modelos traz toda a informação necessária ou dá igual importância às características, eles devem ser utilizados em conjunto. Esta separação é realizada para que o engenheiro de requisitos seja capaz de se

concentrar ora em serviços ora em dados, mas amplia os problemas de espalhamento e entrelaçamento e tem o preço de ter que manter dois modelos atualizados e consistentes.

Consideremos, então, apenas um dos modelos por vez, por exemplo, a lista de sentenças de requisitos. Tomemos duas situações extremas:

1) cada requisito representa apenas um serviço aplicado a um tipo de dado, obedecendo às diretrizes para escrever requisitos (Alexander, 2002) ou as diretrizes de decomposição modular (Staa, 2000). Consideramos que o conceito de modularização em uma lista de requisitos indica que cada requisito deve retratar um forte relacionamento de seus elementos constituintes e representar uma única funcionalidade, sendo pouco dependente de outros requisitos; e

2) cada requisito é escrito livremente, podendo fazer referência às restrições de serviço e dados a que ele se aplica.

Na primeira situação, temos uma lista de requisitos ampla em que os requisitos se encontram entrelaçados apenas por um tipo de serviço ou um tipo de dado. Entretanto, cada tipo de dado e serviço está espalhado por muitos requisitos, veja Figura 6[5]. Por exemplo: operações sobre o dado “evento” estão espalhadas nos requisitos de números 1, 2.8.3, 4.3, 7.1 e 7.3; as operações de “adicionar”, “apagar” e “modificar” estão espalhadas nos requisitos de números 1, 2 e 3; nos requisitos de números 2.8.3 e 2.8.3.1 os dados sobre “eventos” e “categorias” estão entrelaçados, dentre outros.

[5] Este exemplo foi desenvolvido a partir do documento de requisitos do sistema Unical (Unical, 2006), um dos nossos estudos de caso apresentado no Capítulo 5. Estas sentenças de requisitos foram reescritas com base no documento original.

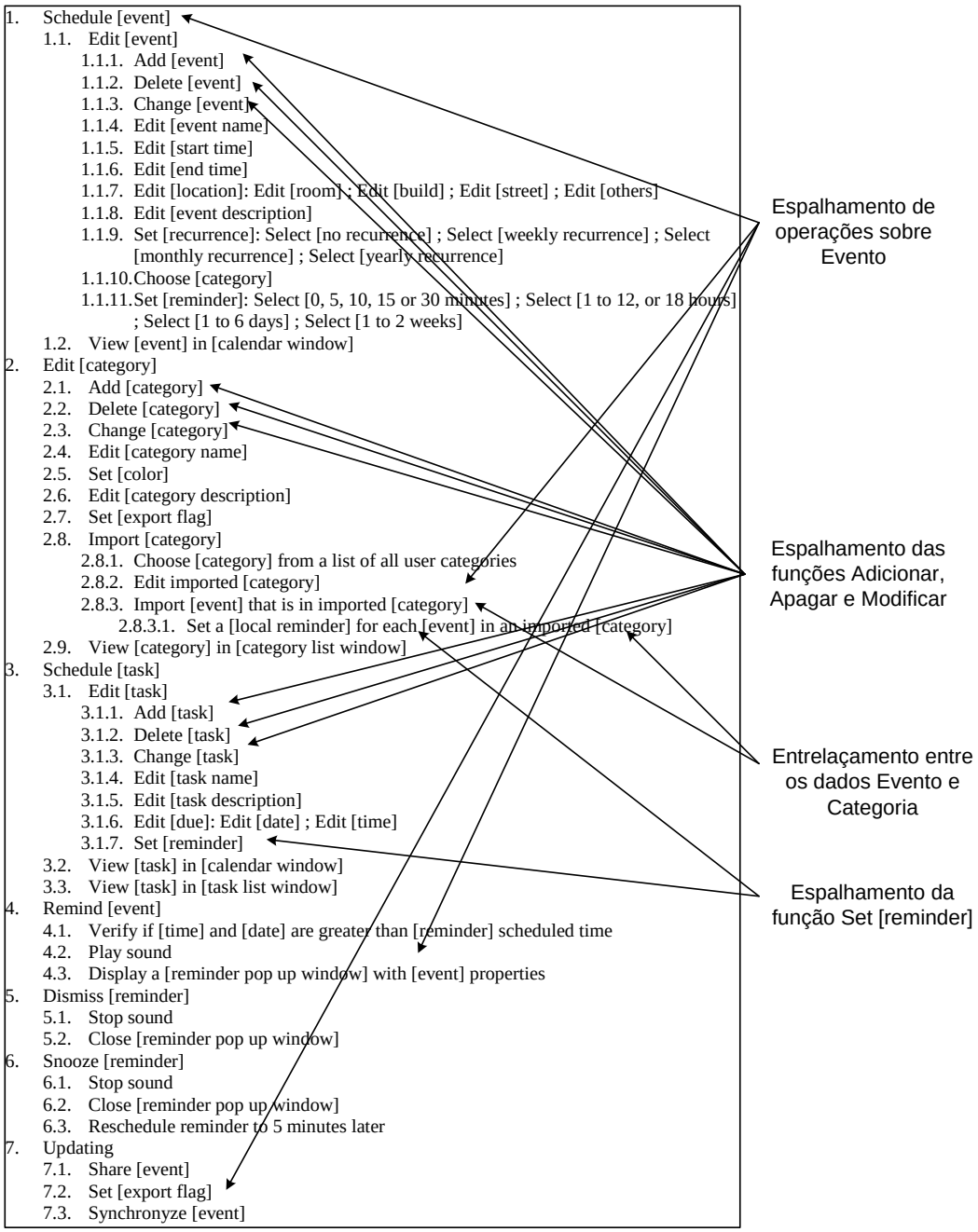


Figura 6. Espalhamento e entrelaçamento em listas de requisitos – 1º exemplo

Por outro lado, na segunda situação, podemos ter uma lista de requisitos menor, e os tipos de dados ou os tipos de serviços não estão muito espalhados, mas cada requisito está tratando de mais de um serviço e um dado, ou seja, as informações estão mais entrelaçadas que no primeiro exemplo, veja Figura 7[6]. Por exemplo: os requisitos de números 4.2.2, 4.2.5 e 4.2.6 contém as operações "adicionar", "apagar" e "modificar" espalhadas e entrelaçadas; os requisitos de

[6] Este exemplo consiste em uma parte do documento de requisitos original do sistema Unical (Unical, 2006).

números 4.2.5.2.3, 4.2.5.2.2 e 4.2.6.2.3 descrevem a mesma restrição ao formato de “tempo”; dentre outros.

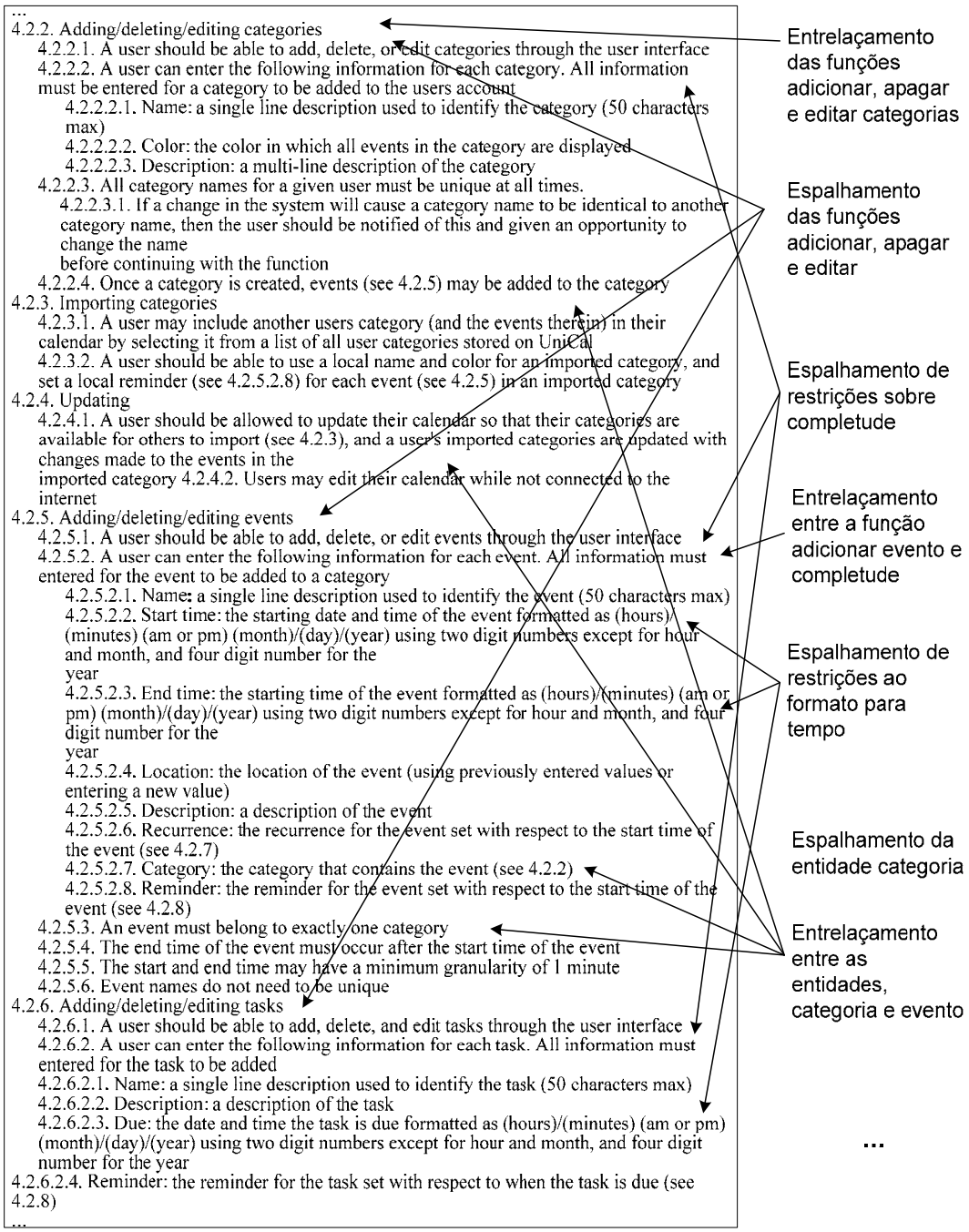


Figura 7. Espalhamento e entrelaçamento em listas de requisitos – 2º exemplo

Em ambos os casos, há os problemas de espalhamento e entrelaçamento. Na segunda situação a dificuldade em manipular o documento pode ser ainda maior porque há maior probabilidade dos requisitos serem confusos e ambíguos. Entretanto, aplicar os conceitos de modularização, apesar de diminuir o problema de entrelaçamento, pode aumentar o espalhamento, como visto no primeiro exemplo. Principalmente porque não há uma única maneira de separar os

requisitos, ora é interessante separá-los como requisitos funcionais, não funcionais e inversos, ora é interessante separá-los por tipo de serviço (por exemplo, tudo que pode ser configurado no sistema), por tipo de dado (por exemplo, tudo que diz respeito a evento), por interessados (por exemplo, tudo que o usuário administrador pode realizar), por relevância, por complexidade, por custo, dentre outros.

Na Figura 6 e Figura 7, exemplificamos os problemas de espalhamento e entrelaçamento em listas de sentenças de requisitos, que são menos estruturadas e modularizadas. Entretanto, estes problemas também ocorrem em modelos de requisitos semi-formais ou estruturados, tais como modelos de metas e cenários. Desta forma, consideramos que é necessário haver uma maneira de realizar não só a separação, mas também o controle das interações entre as partes separadas, sendo possível visualizá-las de maneiras distintas.

Nesta tese, utilizamos modelos de metas (apresentados a seguir) para representar os requisitos do sistema e separar características transversais, e para ter maior controle das interações, utilizamos alguns fundamentos definidos por trabalhos sobre desenvolvimento orientado a aspectos (apresentados no Capítulo 3).

Modelos de Metas

Modelos de metas surgiram com o objetivo de modelar, também, a razão da existência dos requisitos. Desta maneira, o engenheiro consegue identificar a solução mais adequada para o problema em questão. Metas trazem à tona requisitos e regras de negócio normalmente omitidos, e que muitas vezes, levam ao fracasso do produto (Lamsweerde 01; Chung, 2000). Em (Mylopoulos, 1992), *softmetas* são propostas como meio para modelar e analisar RNFs em modelos de metas.

Modelos de metas representam explicitamente requisitos funcionais e não funcionais através da decomposição de metas (Giorgini, 2002; Yu, 2004). Esta decomposição indica como satisfazer uma determinada meta, indicando a razão pela qual as submetas são necessárias. Na literatura há algumas variações de modelos de metas tais como i* (Yu, 95), Kaos (Lamsweerde, 2001) e V-graph (Yu, 2004). Em geral, eles usam árvores *and/or* para representar a decomposição

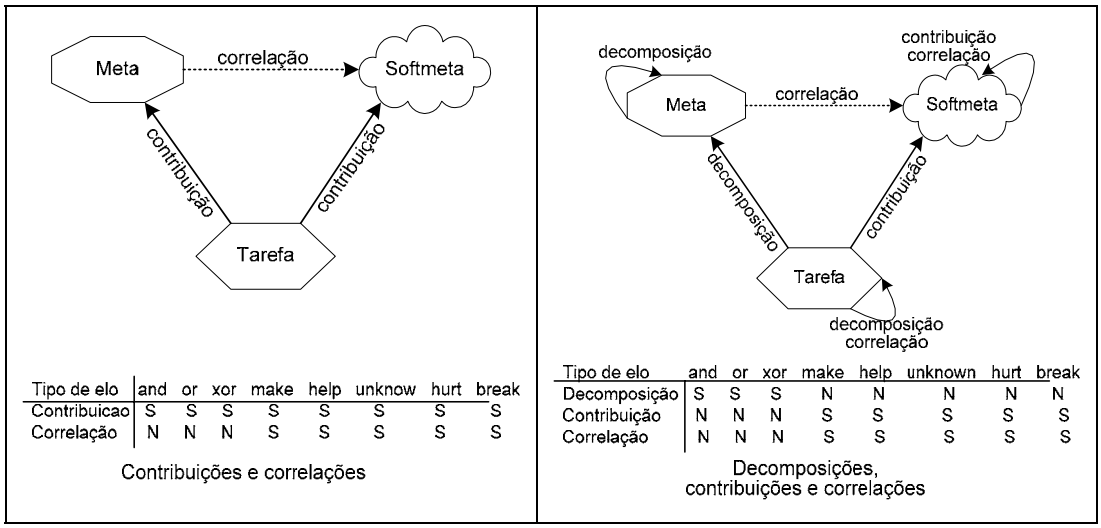
de metas e definir um espaço de soluções alternativas para satisfazer a meta raiz. Na verdade, modelos de metas são grafos e não árvores (Chung, 2000) porque muitos elementos das árvores contribuem para satisfação de elementos em diferentes ramificações.

Como modelos de metas explicitam vários tipos de informação em diferentes níveis de abstração, é possível realizar diferentes tipos de análise, tais como: análise de obstáculos - explora os possíveis obstáculos para a satisfação de uma meta (Lamsweerde, 2000); análise qualitativa de metas - permite atribuir valores qualitativos aos relacionamentos entre metas e ajuda a formalizar e pensar sobre estes relacionamentos (Giorgini, 2002); propagação de rótulo - propaga os rótulos em uma árvore de metas, tornando visível conflitos de interesses e objetivos (Giorgini, 2002); análise de variabilidade (Gonzáles, 2004; Park, 2004)- possibilita a análise de soluções alternativas; análise do *fan-in* e *fan-out* de cada elemento para identificar candidatos a aspectos (Yu, 2004), dentre outros.

V-graph, o modelo de metas que utilizamos nesta tese, utiliza: três tipos de elementos, *softmetas*, metas e tarefas; e dois tipos de relacionamentos, contribuição e correlação. Nós escolhemos este modelo porque seus elementos e relacionamentos nos oferecem maneiras diferentes de separar as características do sistema sem muita dificuldade.

V-graph

V-graph é um tipo de modelo de metas, proposto em (Yu, 2004) como uma simplificação do modelo RNF (Chung, 2000) para demonstrar uma abordagem de identificação de candidatos a aspectos em requisitos. O nome *V-graph* originou-se da disposição entre os seus três elementos constituintes, sendo o elemento tarefa o vértice inferior que operacionaliza metas e *softmetas* correlacionadas. A seguir explicamos os conceitos e relacionamentos do V-graph; para facilitar a explanação, fizemos algumas alterações (Figura 8b) na ilustração original (Figura 8a).



(a) (b)

Figura 8. O modelo de metas *V-graph*: (a) representação original definida em (Yu, 2004); (b) nossa representação, explicitando todos os relacionamentos possíveis entre os elementos do *V-graph*.

V-graph é composto de três elementos: *softmetas*, metas e tarefas. Cada elemento é composto de duas partes: tipo e tópicos. O tipo descreve uma função genérica ou um requisito não-funcional genérico. O tópico captura a informação contextual para o elemento; esta informação contextual é similar ao *subject* definido em (Zave, 1997). Por exemplo, "segurança" e "verificação" são os tipos dos elementos "Segurança na transmissão de dados", "Segurança no armazenamento de dados", "Verificação do modelo de requisitos" e "Verificação das leis trabalhistas", enquanto que "transmissão de dados", "armazenamento de dados", "modelo de requisitos" e "leis trabalhistas" são os tópicos.

Este modelo permite a descrição de nós intencionais (metas e *softmetas*) e nós operacionais (tarefas). Usualmente, *softmetas* representam RNFs, metas representam macro-requisitos, regras ou objetivos do negócio, e tarefas representam RFs que operacionalizam metas e *softmetas*. Metas e *softmetas* se distinguem pelos conceitos de *satisfy* e *satisfice* (Mylopoulos, 1992), respectivamente, i.e.: uma meta é satisfeita totalmente (satisfy) através do conjunto de submetas e tarefas nas quais ela se decompõe; e uma *softmeta* é suficientemente satisfeita (*satisfice*) pelas metas, *softmetas* e tarefas que contribuem ou estão correlacionadas positivamente a ela. Os possíveis relacionamentos entre metas, *softmetas* e tarefas são:

Decomposição – elos de decomposição podem ter rótulo AND, XOR ou OR. AND indica que o subelemento é obrigatório, OR que ele é opcional e XOR

que pode ocorrer apenas um dos subelementos. Eles podem indicar refinamento ou agregação, decompondo os tipos ou tópicos de cada elemento. O sentido da decomposição é do elemento decomposto para o elemento pai, podendo ocorrer entre sub*softmeta*→*softmeta*, submeta→meta, subtarefa→tarefa e tarefa→meta. Normalmente, utilizamos o elo de decomposição quando o subelemento está semântica e diretamente associado ao elemento-pai. Na Figura 9a, ilustramos alguns relacionamentos de decomposição.

Contribuição – elos de contribuição podem ter os rótulos *make(++), help(+), unknown(?), hurt(-)* e *break(--). Make* e *help* indicam contribuição positiva, *unknown* indica que há um relacionamento, mas é desconhecido se positivo ou negativo, e *hurt* e *break* indicam contribuição negativa. A contribuição é utilizada com o mesmo sentido da decomposição, mas ocorre entre sub*softmeta*→*softmeta* e tarefa→*softmeta* devido à natureza *fuzzy* do elemento *softmeta*. Na Figura 9b, ilustramos alguns relacionamentos de contribuição.

Correlação – elos de correlação também podem ter os rótulos *make(++), help(+), unknown(?), hurt(-)* e *break(--).* Entretanto, a correlação é utilizada para representar relacionamentos de contribuição horizontal entre diferentes árvores ou subárvores, indicando menor acoplamento do que os elos de decomposição e contribuição. A correlação é baseada no conceito de "conflito e harmonia" indicando a interferência positiva ou negativa entre árvores de metas aparentemente desconexas. A correlação pode ocorrer entre *softmeta*→*softmeta*, meta→*softmeta*, e tarefa→tarefa. A correlação entre tarefa→tarefa ocorre somente quando existe uma correlação entre os elementos pais de ambas as tarefas. Na Figura 9c ilustramos alguns relacionamentos de correlação.

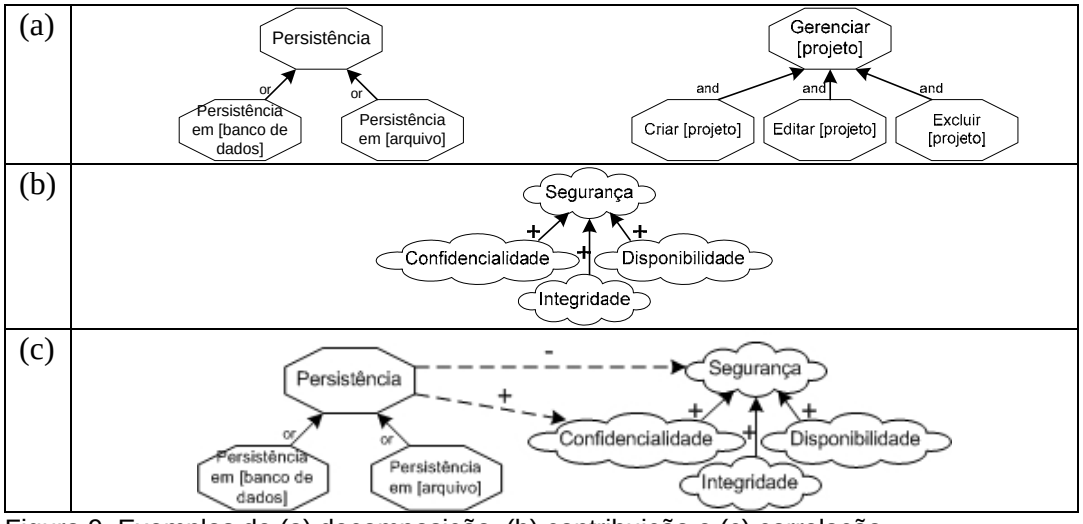


Figura 9. Exemplos de (a) decomposição, (b) contribuição e (c) correlação.

Em (Yu, 2004), os relacionamentos existentes entre tarefas são utilizados na identificação de aspectos-meta (goal aspects). As tarefas que contribuem ou estão correlacionadas a várias *softmetas* e metas indicam espalhamento e entrelaçamento ou que podem se espalhar por vários componentes em fases posteriores, se este comportamento não for percebido a tempo de escolher uma modularização melhor para elas.

Os elementos do V-graph nos possibilitam a extração de várias visões: visão de dados, através dos tópicos; visão de objetos, através dos tópicos e tipos; influência direta ou indireta que uma característica exerce sobre as outras, através dos relacionamentos; visão funcional, através dos tipos; dentre outras.

2.3. Visões

Uma visão é uma representação do sistema inteiro sob a perspectiva de um conjunto de características relacionadas (IEEE St.1471). O uso de visões é uma maneira de separar diferentes características para focar uma por vez. Elas ajudam no entendimento e elaboração de soluções (Sommerville, 2000), sendo, assim, necessárias durante todo o processo de desenvolvimento.

Na literatura, muitas vezes os termos em inglês *viewpoints*, *views* e *point-of-view* são utilizados com significados diferentes ou semelhantes (Leite, 1996). Há problemas, também, quando os traduzimos para o português, pois passam a ser representados pelas expressões “pontos de vista”, “visões” e “perspectivas”. Nesta tese adotamos o termo “visão” num sentido amplo que pode significar todas ou uma das três categorias definidas em (Leite, 1996):

- Visões como opiniões (pontos de vista) – no contexto social, cada um dos interessados tem suas próprias premissas, prioridades e experiências e lidam com os problemas de maneiras diferentes. Assim, é necessário conhecer, comparar e negociar as diferentes opiniões ou diferentes maneiras de ver a mesma coisa, por exemplo: qual a opinião do gerente, do comprador e do vendedor sobre uma compra?
- Visões como serviços (característica) – a idéia de particionar o sistema em um conjunto de serviços que podem ser conectados de diferentes maneiras provê o desenvolvimento baseado em componentes, por exemplo, pagamento de contas, vendas, segurança, dentre outros;

- Visões como modelos (perspectivas) – no contexto de engenharia de software várias técnicas baseadas em linguagens têm sido propostas para retratar o sistema parcialmente, por exemplo, modelo entidade-relacionamento, casos de uso e diagramas de seqüência. Assim, é importante detectar problemas de consistência e completude entre modelos.

Estas categorias não representam conjuntos disjuntos. Normalmente, utilizamos **modelos** (perspectivas) para representar **serviços** de acordo com a **opinião** (ponto de vista) de um ou mais interessados. Além disto, os modelos, por si só, explicitam um tipo de informação ocultando outros, então temos modelos que enfatizam funções, outros, dados, outros enfatizam seqüência de atividades e assim por diante. Desta forma, além dos serviços estarem naturalmente entrelaçados e espalhados entre si, eles ainda encontram-se espalhados e entrelaçados entre as diferentes opiniões e também nos diferentes elementos representativos de cada modelo.

Integração de Visões

Se por um lado, a utilização de visões ajuda a gerenciar a complexidade da modelagem, por outro ela leva ao espalhamento e entrelaçamento de informações. Assim, são necessários mecanismos de integração, seja por meio da integração de serviços, modelos ou de opiniões (resolução de conflitos).

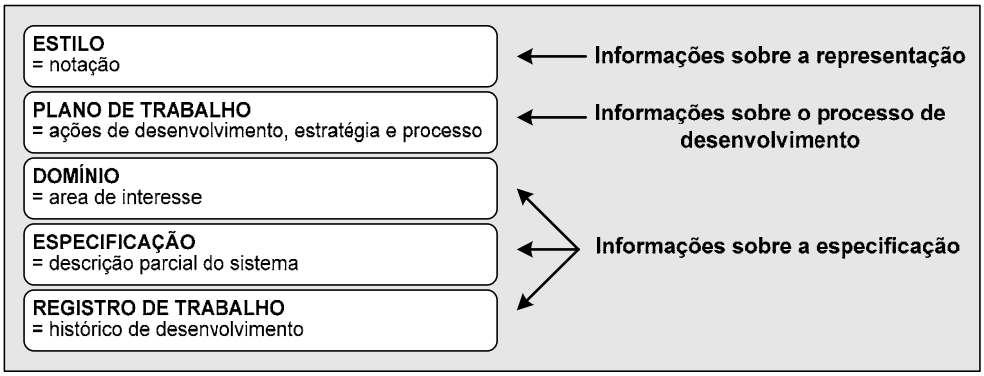


Figura 10. Framework de integração de métodos

Em (Nuseibeh, 2004) é definido um framework para integração de modelos (métodos de modelagem). Este framework determina a especificação das seguintes informações (veja a Figura 10): 1) estilo – indica a notação utilizada; 2) plano de trabalho – indica atividades, estratégias e processos para construir a visão – regras de transformação entre os métodos; 3) domínio – indica a área de

interesse; 4) especificação – é o modelo em si; e 5) registro de trabalho – indica o estado e histórico da especificação em desenvolvimento. As informações descritas em (1) e (2) são informações genéricas, aplicadas a todas as instâncias do mesmo método. As informações em (3), (4) e (5) deste framework são específicas para cada instância do modelo.

Este framework pode ser utilizado para especificação das três categorias de visões (opiniões, modelos e serviços), mas seu foco é a integração de modelos, pois cada instância do framework especifica uma representação. É possível integrar duas instâncias que possuem a mesma notação, e que representam opiniões ou serviços diferentes, mas todo o processo de relacionar os dois serviços tem que ser refeito para cada instância, ou seja, não reutilizado do framework. O principal propósito deste trabalho é dar apoio à checagem de consistência entre diferentes modelos e ao gerenciamento de inconsistências, possibilitando a reutilização das informações sobre a represntação e o processo de desenvolvimento.

Já em (Cysneiros, 2001), é definido um processo para integrar RNFs a RFs (serviços). Ele utiliza os construtos do modelo de entidade-relacionamento (MER), diagrama de classes, e léxico ampliado da linguagem (LAL) (Leite, 1990), para realizar esta composição. O LAL deve conter todos os símbolos representativos de RF e RNFs, com os relacionamentos entre símbolos representados em linguagem natural através do elemento “impacto”. O processo consiste em identificar no LAL todos os símbolos relacionados a cada RNF e para cada um destes símbolos, acrescentar ao modelo entidade-relacionamento ou de classe uma referência para aquele RNF. Depois disto, é necessário verificar se esta adição implica em mudanças maiores nos modelos.

Em resumo, o trabalho definido em (Cysneiros, 2001) foca apenas a adição de RNFs a modelos de RFs sem preocupação em observar os problemas de espalhamento e entrelaçamento naturais a muitos RNFs. Desta forma, uma vez que o RNF é integrado aos vários elementos do MER ou diagrama de classes, não há nenhuma facilidade para que o engenheiro os veja separados ou perceba o entrelaçamento e espalhamento ocasionados.

A sobreposição e os conflitos existentes entre diferentes opiniões (integração de pontos de vista) são abordados por técnicas de análise de conflitos

(Kotonya, 1996; Leite, 1989, 1991), e estão fora do escopo de nosso trabalho. Nesta tese estamos focados em visões como serviços e como modelos:

- considerando-as serviços, queremos prover uma maneira mais eficiente de separá-las sem perder a habilidade de gerenciar a interseção entre elas (relacionamento ou interação); e
- considerando-as modelos, queremos ser hábeis a visualizar os serviços e suas interações por meio de diferentes representações (modelos).

2.4. Evolução

Devido à natureza volátil dos requisitos, é necessário estar preparado para mudanças, ou evolução. A evolução de requisitos ocorre em dois sentidos: no sentido do desenvolvimento de software, mudando do nível alto de abstração para a implementação, i.e., de requisitos para código; e no sentido da melhoria contínua para atender as expectativas dos requisitantes, sofrendo alterações decorrentes de erros/omissões ou para atender novas necessidades (Lehman, 1996; Leite, 1997).

Em ambos os sentidos, conhecer e gerenciar as **interações** entre requisitos é extremamente importante para:

- a decomposição e modularização das características do sistema porque indica o acoplamento e coesão entre as partes envolvidas; e
- para a análise do impacto de mudanças, tanto entre requisitos no mesmo nível de abstração quanto entre requisitos e artefatos de software em níveis de abstração diferentes (Robinson 2003).

Interações entre requisitos são relacionamentos que indicam dependência, decomposição, complementação, conflito, dentre outros (Robinson, 2003). Além de acoplamento e coesão estes relacionamentos podem indicar a dificuldade de modificar requisitos ou parte deles e de manipular (incluir, modificar, excluir) os elementos textuais ou gráficos que os representam em cada modelo de requisitos. Desta forma, se um requisito tem alto acoplamento e é difícil de manipular, consideramos, nesta tese, que ele é complexo. Utilizamos o termo “complexidade” como definido em (Ferreira, 2004): a qualidade de abranger muitos elementos ou partes, de ser observado sob diferentes aspectos, ou de ser composto por

elementos com naturezas distintas. A gerência de interações está associada à rastreabilidade de requisitos.

2.5. Rastreabilidade

Como definido em (Sommerville, 2000), rastreabilidade é a propriedade de uma especificação de requisitos que reflete a habilidade de encontrar requisitos relacionados. Sommerville (2005) define três tipos de informações a serem rastreadas:

- Para a fonte (origem), também chamada de pré-rastreabilidade, liga os requisitos aos interessados que os propuseram e à razão de algumas escolhas *(rationale)* (Dutoit, 2001). Quando uma mudança é proposta, esta informação é utilizada para encontrar e consultar os interessados sobre tais mudanças;
- Entre requisitos, registra a dependência entre requisitos (interações entre requisitos). Esta informação é utilizada para avaliar quais requisitos são afetados por uma mudança e quais as subseqüentes mudanças necessárias;
- Para o projeto, também chamada de pós-rastreabilidade, liga os requisitos aos módulos da arquitetura e implementação. Esta informação é utilizada para avaliar qual o impacto que mudanças nos requisitos exercem nos módulos que os implementam e/ou estão relacionados.

As informações de rastreabilidade são, geralmente, representadas por matrizes de rastreabilidade (Davis, 1990; Sommerville, 2000; Gotel, 1994) ou mesmo pelas linguagens de modelagem de requisitos, tais como: cenários (Leite, 1997); casos de uso (Jacobson, 1992); modelos de entidade relacionamento; modelos de metas (Chung, 2000; Lamsweerde, 2001); dentre outros.

Nesta tese, abordamos, principalmente, a rastreabilidade no sentido de interações entre requisitos. Gerenciando as interações entre requisitos ou entre elementos que os representam é possível analisar, descobrir e agrupar elementos que estão mais coesos e menos acoplados e assim, reutilizar tais agrupamentos.

2.6. Reuso

O reuso durante a definição de requisitos ocorre, principalmente, em relação ao conhecimento sobre requisitos não funcionais (RNFs) ou quando desenvolvendo uma família de produtos. Requisitos não funcionais são características ou restrições impostas ao comportamento de um sistema. Estas características, normalmente, dão origem ou restringem diversas funcionalidades. Por se encontrarem espalhadas e entrelaçadas nos requisitos de um sistema, são consideradas características transversais (Leite, 2004; Rashid, 2003).

Em (Cysneiros, 2003; Sommerville, 2000) é proposta a catalogação de RNFs para que possam ser reutilizados. Em (Sommerville, 2000), relata-se como taxonomias de RNFs podem ajudar na identificação de requisitos ocultos e tácitos. Estas taxonomias são utilizadas para lembrar e guiar os engenheiros na definição de requisitos. Em (Cysneiros, 2003), é proposto um esquema para catalogar e compartilhar RNFs utilizando grafos de metas. Esta abordagem é orientada a domínio, onde cada domínio é definido pelas facetas: tipo, lista de tipos relacionados, lista de operacionalizações e tópico, veja Figura 11.

<*Domain*, Facet **type**, Facet **list of related types**, Facet **list of operationalizations**, Facet **topic**>

Ex.: (traceability, {safety, performance, security, reliability}, {make a historic of places, store the last location, store all the times thing has moved, store time of last move, store all the times thing was changed, …}, things)

Figura 11. Exemplo de template utilizado para catalogar RNFs

```
START := Advice*
Advice := [When] [Who] Why [What]
[Where] [How] [HowMuch]
When:= "(" Expr ")" "=>"
Who:= "<" id ">" "::"
Why:= id
What := "[" id {"," id}* "]"
Where:= "<=" Pointcut {"," Pointcut }*
How:='{' BoolOp Advice* '}'
HowMuch:= "=>" Effect {"," Effect }*
Expr:= "true" | "false" | id | Expr
BoolOp Expr
Effect:= HowMuchOp [ Who "::" ] Why [
"[" What "]" ]
Pointcut:= HowMuchOp [("*"|Who) "::"]
("*" | Why) [ ("[" "*" "]" | What ) ]
BoolOp := "&" | "|"
HowMuchOp:="++"|"+"|"-"|"--" | "?"
```

(a)

```
Persistence { AND
  "Persistence in" [DB] { AND
   "initiate [DB]"
   "verify if is connected"[DB]
   Connect [DB]
   Disconnect [DB]
  } => + mro
  "Make registry operations" { AND
   include [data]
   select [data]
   delete [data]
   update [data]
  }
  "Persistence in" [file] => + "Make
registry operations"
} => + Authentication
```

(b)

Figura 12. a) BNF da linguagem Q7 b) Exemplo utilizando Q7

Em (Leite, 2005), os autores propõem um processo para viabilizar o reuso de RNFs. Esta abordagem define a linguagem Q7, baseada no framework 5W2H (*What, Where, When, While, Who, How* e *How much*), para armazenar e recuperar requisitos de uma biblioteca (veja a Figura 12).

Os elementos desta linguagem registram as informações sobre os RNFs e sobre como relacioná-los a RFs, veja na Figura 13: *Why* – registra a intencionalidade, a razão para utilizar o RNF; ele ajuda na seleção de um grafo parcial; *Who* – registra o requisito destino (*target*) do RNF, a quem ele se aplica; *What* – resgistra as partes do RNF que devem ser aplicadas no requisito destino; *Where* – registra o ponto específico no requisito destino onde o RNF deve ser aplicado; *When* – registra pré-condições que precisam ser atendidas antes que operacionalizações (*How*) possam ser aplicadas em um ponto (*Where*); *How* – registra o refinamento do RNF através de RFs; *How much* – registra o impacto de aplicar as operacionalizações (How).

Who	What	Why	When	Where	How	How much
Função	tópico	Tipo	Claim	*pointcuts*	operacionalizações	elos de contribuição

Figura 13. Relação dos atributos de Q7 e os elementos do V-graph

Este processo utiliza dois V-graphs como entrada, o primeiro recuperado da biblioteca e o segundo representando a aplicação onde o RNF será inserido, e gera um terceiro grafo de metas com as informações integradas. Esta composição é realizada manualmente analisando onde (elemento *where*) na representação funcional o RNF deve ser ligado, e o que (elemento *what*) deve ser incluído. O trabalho sugere que esta tarefa pode ser automatizada, consistindo de um mecanismo similar ao *weaver* de linguagens de programação orientadas a aspectos (Capítulo 3). Este trabalho motivou nossa idéia de definir um mecanismo automatizado de composição de características transversais. No Capítulo 6, apresentamos as diferenças e semelhanças entre essa abordagem e a nossa.

2.7. Resumo

Neste Capítulo descrevemos o contexto e escopo desta tese, a engenharia de requisitos. Mostramos como os problemas de espalhamento e entrelaçamento atrapalham as atividades durante a definição de requisitos e as conseqüências destes problemas para o desenvolvimento de software como um todo. Além disto, apresentamos os conceitos de reuso, rastreabilidade, evolução e como aqueles problemas têm sido tratados pela utilização de diferentes visões.