RBC - REVISTA BRASILEIRA DE COMPUTAÇÃO OUT./DEZ. 1990 RIO DE JANEIRO V.6 N. 2. P. 39 - 52

Validação de requisitos: o uso de pontos de vista

JULIO CESAR SAMPAIO DO PRADO LEITE
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

RESUMO

Apesar do crescente interesse sobre a definição de requisitos, observa-se que a maioria dos trabalhos e métodos apresentados dirigem sua atenção para o problema da modelagem dos requisitos, deixando sempre em aberto o problema fundamental que é exatamente **o que** modelar. Nosso artigo apresenta os conceitos usados e resultados obtidos na aplicação de uma estratégia para validação de requisitos. Nossa estratégia, baseada na análise de diferentes pontos de vista, é uma proposta para uma validação antecipada do *entendimento* do software a ser construído

ABSTRACT

Requirements modeling demands that the knowledge of what should be modeled be available. Acquiring this knowledge or eliciting the necessary requirements is recognized to be a very hard problem. Different suggestions have been made to alleviate this problem. We present an approach centered on the very early validation of requirements, exploring the existence of multi viewpoints in describing a given situation.

PALAVRAS-CHAVE:
- Análise de requisitos
- Definição de requisitos
- Elicitação
- Pontos de vista
- Validação

RECEBIDO EM: 25 /10 / 90

ENDEREÇO DO AUTOR PARA CORRESPONDÊNCIA:

Departamento de Informática
Pontifícia Universidade Católica
do Rio de Janeiro
R. Marquês de S. Vicente, 225
22453 - Rio de Janeiro - RJ
e.mail: leit@lncc.bitnet

1 — INTRODUÇÃO

Análise de requisitos é um termo estabelecido em engenharia de software, para identificar a primeira das fases na construção de software. Consideramos, não obstante, que o melhor termo para identificar esta fase é DEFINIÇÃO DE REQUISITOS. Apesar do crescente interesse acerca da definição de requisitos, observa-se que a maioria dos trabalhos e métodos apresentados dirigem sua atenção para o problema da modelagem dos requisitos, deixando sempre em aberto o problema fundamental que é exatamente **o que** modelar.

Em particular cresce o interesse pela tarefa de elicitação dos requisitos, ou seja, de tornar explícito os desejos, as intenções e necessidades dos clientes em relação ao software a ser construído.

O termo Elicitação de Requisitos refere-se ao processo de *elicitar* [1]. Este neologismo se justifica porque se trata de um subprocesso da definição de requisitos em que a preocupação fundamental não é a de modelar, mas a de coletar, validar e tornar explícito os requisitos do software a ser construído.

Por longo tempo, vem a área de engenharia de software concentrando seus esforços na representação de requisitos, especificações e projetos. Várias propostas de formalização das mais formais a menos formais vêm sendo apresentadas pela comunidade de pesquisa em engenharia de software [10]. Atualmente alguns grupos de pesquisa em várias universidades e centros de pesquisa [8] vêm estudando o problema da elicitação de requisitos, isto é a absorção das reais necessidades dos futuros usuários do software a ser produzido. A contribuição do nosso trabalho é a de apresentar conceitos e resultados de pesquisa que acrescentam conhecimento sobre esta difícil tarefa de elicitar requisitos.

O grande problema que o engenheiro de software encontra na tarefa de elicitar requisitos é a falta de métodos que forneçam um tratamento mais preciso para esta árdua tarefa.

[1] ***Elicitar** : [Var. eliciar + clarear + extrair.] V. t. d.1. descobrir, tornar explícito, obter o máximo de informações para o conhecimento do objeto em questão.*

Novamente vale lembrar que na maioria dos casos, a elicitação de requisitos é deixada a cargo do "bom analista de sistemas"[2], que segundo as listas de qualidades apresentadas na literatura [26][5] mais parecem super-homens. O âmago deste problema reside na reconhecida dificuldade de modelar a realidade. Acreditamos que, conforme bem expressou Fred Brooks [4], não há soluções mágicas para o problema da definição de requisitos. Não obstante, há necessidade de que novas estratégias, métodos e ferramentas estejam à disposição dos engenheiros de software para amenizar as dificuldades desta tarefa. Nosso trabalho centra sua atenção nos aspectos da validação de requisitos.

Recentemente, vários pesquisadores [2] têm proposto linguagens de especificação executáveis, de forma a trazer o processo de validação para mais perto da fase de definição. Partimos de uma proposta em que a validação é feita antecipadamente à especificação. Na nossa proposta a validação é feita no próprio processo de definição de requisitos, mais especificamente durante a elicitação de requisitos.

Nossa pesquisa investigou o uso de diferentes pontos de vista como uma forma de validação dentro do próprio processo de elicitação. Através de um software dirigido por heurísticas, implementamos um analisador de pontos de vista que identifica e classifica diferenças entre pontos de vista. Por exemplo, um engenheiro de software no processo de elicitação de requisitos sobre uma biblioteca coleta fatos com a bibliotecária chefe e com uma das bibliotecárias operadoras. O conjunto de fatos coletados da bibliotecária chefe reflete o seu ponto de vista e o outro conjunto de fatos reflete o ponto de vista da bibliotecária atendente. O analisador de pontos de vista, com base neste dois conjuntos de dados, fornece uma lista, classificada, de discrepâncias entre os pontos de vista. Desta forma provemos uma agenda sobre a qual a reconciliação desses pontos de vista pode ser negociada. Através de experimentos empíricos, verificamos a utilidade dessa reconciliação como um instrumento para um melhor entendimento do assunto em questão.

O artigo apresentará uma caracterização da definição de requisitos orientada a processos, e em particular detalhará o aspecto de validação, Seção 2. Em seguida apresentaremos o conceito de pontos de vista, a definição de resolução de pontos de vista, e detalharemos os problemas atacados pela nossa pesquisa, Seção 3. Na Seção 4 esboçaremos a estratégia de validação baseada em pontos de vista, e detalharemos a representação utilizada. Na Seção 5, apresentaremos o analisador de pontos de vista e descreveremos as heurísticas utilizadas. Na Seção 6 reportamos os casos em que avaliamos empiricamente nossa estratégia. Concluímos, Seção 7, comparando nossa estratégia com outras propostas e relatamos sobre futuros trabalhos em validação por pontos de vista.

[2] *No decorrer deste trabalho usamos a palavra engenheiro de software como aquele profissional com educação adequada para trabalhar na construção de software. Na literatura de sistemas de informação ou análise de sistemas este profissional é referido como analista de sistemas.*

2 — DEFINIÇÃO DE REQUISITOS

A definição de requisitos não se processa no vazio. Nossa perspectiva de sistemas [9], identifica a definição de requisitos como uma parte do sistema de construção de software, cujo produto, o software, é um subsistema de um sistema maior. Este sistema maior, onde o software vai atuar define o que chamamos de universo de informações.

> Universo de informações é o contexto geral no qual o software deverá ser desenvolvido. O universo de informações inclui todas as fontes de informação e todas as pessoas relacionadas ao software. Estas pessoas são também conhecidas como os atores desse universo. O universo de informações é a realidade circunstanciada pelo conjunto de objetivos definidos pelos que demandam o software.

O uso da teoria geral de sistemas, como arcabouço básico para o entendimento do contexto da definição de requisitos, se fundamenta na proposta, dessa teoria, de servir como um fator integrador de diversas áreas do conhecimento. O software é sempre um subsistema. Dependendo do caso, o software pode ser: um subsistema de um sistema de computação (hardware + software), um subsistema de uma determinada aplicação num sistema de informação numa organização, um subsistema do controle de frenagem em um automóvel, ou um subsistema do processo de produção de software. O universo de informações é exatamente o resultado da aplicação dessa visão sistêmica, com o objetivo de facilitar o trabalho de compreensão e detalhamento do **que** deve ser feito.

O universo de informações (Udi), já delimitado pelo sistema mais abrangente, não implica que a informação necessária para o **que** deve ser feito pelo software já esteja disponível. No entanto, o Udi restringe as fontes de onde estas informações serão elicitadas. Esta visão da definição de requisitos, ocorrendo *a posteriori* da definição do universo de informações, restringe seu campo de atuação. Esta restrição é equivalente ao que Davis [6] chama de segundo nível na determinação de requisitos para sistemas de informação. Davis entende que a determinação de requisitos ocorre em dois níveis: requisitos organizacionais e requisitos da aplicação.

Outro ponto importante é o reconhecimento da necessidade de elicitação. Isto é baseado no senso

comum de que antes de fazer algo, o primeiro passo é conhecer e entender **o que** dever ser feito. Como dizia Von Neumann:

> *There is no sense in being precise about something when you do not even know what you are talking about.*

Afirmar o óbvio nem sempre é tarefa simples. Muito da confusão reinante na literatura de engenharia de software sobre os termos "requisito" e "especificação" [19] provêm justamente da falta de entendimento da óbvia frase citada acima. Sem entrar em maiores detalhes citaremos as definições de cada palavra segundo o dicionário:

> Requisito: Condição necessária para a obtenção de certo objetivo, ou para o preenchimento de certo objetivo.
>
> Especificação: Descrição minuciosa das características que um material, uma obra ou um serviço deverão apresentar.

Levando em consideração estes pontos, isto é, que requisitos pertencem a um sistema maior, e da necessidade de elicitá-los, propomos a seguinte caracterização da definição de requisitos.

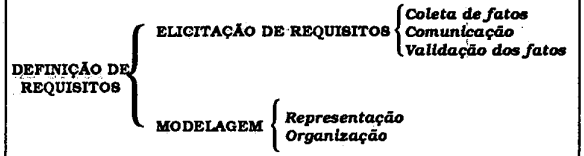
> Definição de requisitos é um processo em que o que deve ser feito é elicitado e modelado. Este processo deve lidar com diferentes pontos de vista, e usa uma combinação de métodos, ferramentas e atores. O produto desse processo é um modelo, do qual um documento, chamado requisitos, pode ser extraído.

Em seguida detalharemos esta visão da definição de requisitos como um processo, bem como descreveremos em mais detalhe o subprocesso de validação.

2.1 — O PROCESSO DA DEFINIÇÃO DE REQUISITOS

Conforme a caracterização apresentada, o processo de elicitação é diferenciado do processo de modelagem dos requisitos. O processo de elicitação compreende os processos de: coleta de fatos, validação dos fatos e comunicação. O processo de modelagem compreende os processos de: organização e representação. Apesar da dificuldade de se separar estes processos, já que a definição de requisitos é um emaranhado desses processos, esta classificação nos auxilia na tentativa de melhor compreender esta tão difícil tarefa, vide Figura 1.

FIGURA 1
O PROCESSO DA DEFINIÇÃO DE REQUISITOS



É durante a elicitação que o conhecimento do universo de informação de uma determinada aplicação é adquirido, para que mais tarde possa ser modelado. Os subprocessos, coleta, de fatos, validação de fatos e comunicação estão bastante interligados. A coleta de fatos procura enumerar fatos que fielmente traduzam o mundo real, evitando ao máximo interpretações. A validação é a maneira com que se confirma ou não a acuidade do processo de coleta. O processo de comunicação permeia a coleta e a validação, já que é fundamental o diálogo entre os vários atores presentes na elicitação.

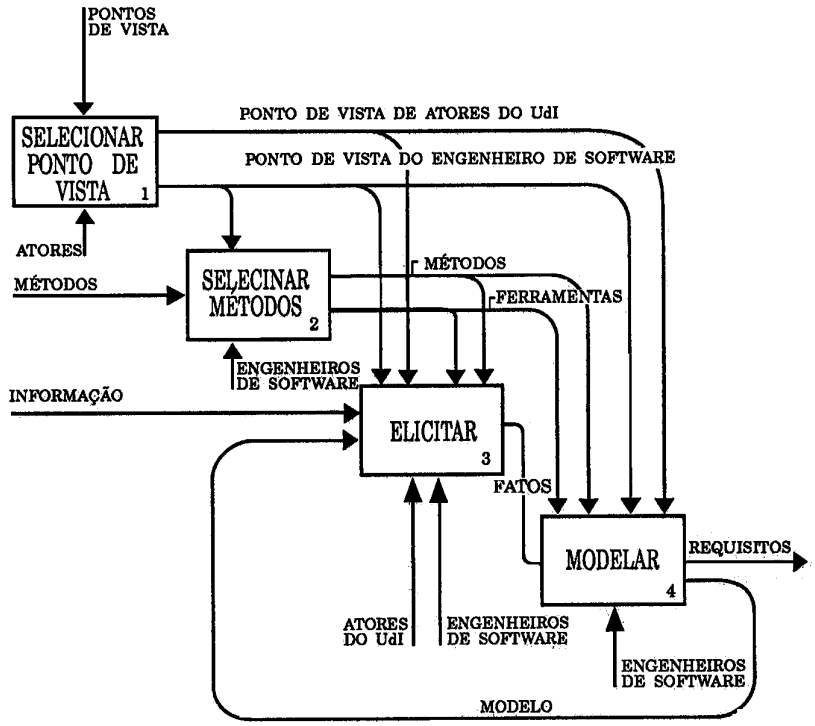
Modelagem é o processo responsável pela caracterização dos resultados obtidos pela elicitação de requisitos. Modelagem, não só produz o modelo final, mas também sua forma formatada, isto é, os requisitos.

O processo de modelagem é composto de duas partes: representação e organização. Para que seja produzido um modelo dos fatos elicitados, os atores usam métodos e ferramentas. Um modelo é composto de um esquema de representação, organizado segundo as políticas organizacionais do modelo. Um modelo deve ser capaz de capturar pontos de vista diferentes.

Pontos de vista refletem diferentes *backgrounds*: quer dos atores responsáveis pela definição de requisitos, como também por aqueles atores pertencentes ao UdI e que estão envolvidos de alguma forma com a aplicação. Método é um conjunto de técnicas e procedimentos. Ferramentas fornecem apoio no uso de um determinado método.

O modelo SADT [30] apresentado na Figura 2, reflete esta caracterização e procura clarificar os diferentes relacionamentos entre as partes e os subprocessos da definição de requisitos. O ponto de vista usado neste modelo SADT é o de identificar relacionamentos que podem existir no processo de definição de requisitos, ele não aborda a operação desse processo.

FIGURA 2
ELICITAÇÃO DE REQUISITOS



Devemos ressaltar, mais uma vez, que o processo de produção de requisitos é bastante entrelaçado, isto é, é difícil realmente separar cada um desses subprocessos, que interagem continuamente. Por causa disso, várias interações ocorrem antes que a versão final do modelo e os requisitos sejam produzidos. A taxonomia apresentada tem por objetivo explicitar conceitos e identificar componentes, que apesar do interrelacionamento muito estreito têm características únicas. A seguir detalharemos aspectos relacionados ao subprocesso de validação.

2.2 — VALIDAÇÃO NA DEFINIÇÃO DE REQUISITOS

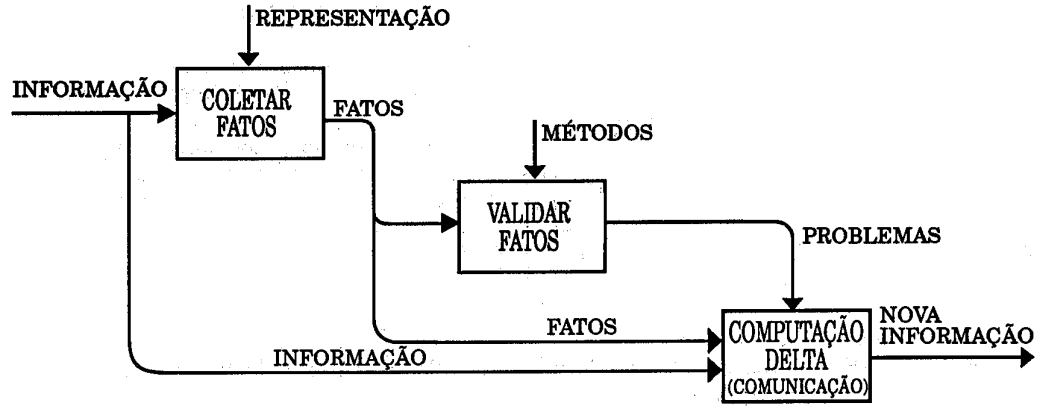
A validação de software, ou seja a confirmação de que o produto é aquele desejado pelo usuário, ocorre, normalmente, no fim do ciclo de vida. O teste do sistema, como é comumente conhecida esta validação, é o teste integrado dos programas do sistema pelo usuário. No nosso caso a validação é feita no próprio processo de elicitação de requisitos, ou seja uma validação anterior à própria especificação.

Várias técnicas de validação de software têm sido propostas pela literatura. Se por hipótese, definirmos que algumas dessas técnicas também podem ser aplicadas à elicitação de requisitos, teremos um conjunto, do qual se sobressaem as seguintes técnicas:

- Comprovação informal.
- Prototipagem.
- Uso de formalismos.
- Reusando domínios.

Todas estas técnicas são capazes de achar alguma diferença (Delta Δ) entre os fatos coletados e o universo de informações. A identificação destes Delta, chamada de computação Delta, é parte do processo de comunicação. Portanto. no processo de comunicação entre atores do UdI e engenheiros de software há oportunidade para se encontrar diferenças entre os fatos coletados e o universo de informações. A coleta de fatos fornece as entradas para a computação Delta. As diferenças entre as técnicas de validação são determinadas pelo tipo e qualidade das entradas fornecidas para a computação Delta, Figura 3.

FIGURA 3
COMPUTAÇÃO DELTA



Na comprovação informal [25] a validação ou a revisão dos requisitos é basicamente uma tarefa de leitura de descrições em linguagem natural e do uso de *check-lists* para detectar problemas na expressão dos requisitos. As estratégias para a comprovação informal são várias, mas elas têm em comum a falta de um apoio automatizado e a excessiva dependência das habilidades analíticas de um "bom" engenheiro de software. A computação Delta é controlada por um lista de problemas.

Δ = **lista de problemas** (informação, fatos)

Em prototipação vários tipos de protótipos têm sido propostos como um meio de obter uma retroalimentação do universo de informações. Alguns deles usam linguagens de alto nível (linguagens do tipo geradores de aplicação) [14], enquanto outros usam linguagens de especificação executáveis [31]. A idéia básica é que pela prototipação é possível validar os requisitos/especificação contra as expectativas do usuário. A computação Delta, neste caso, é controlada pelo comportamento dos fatos:

Δ = **comportamento dos fatos** (expectativas do usuário, fatos)

No uso de formalismos [15], onde o engenheiro de software faz o papel de um provador de teoremas, a validação ocorre dada a possibilidade de se identificar inconsistências, as quais conseqüentemente controlam a computação Delta:

Δ = **inconsistências** (fatos, informação)

Usando-se a técnica de reutilização de domínios [27], [7], que é uma técnica na qual estratégias e heurísticas de Inteligência Artificial são usadas, pretende-se criticar os requisitos cotejando-os contra um domínio previamente codificado.

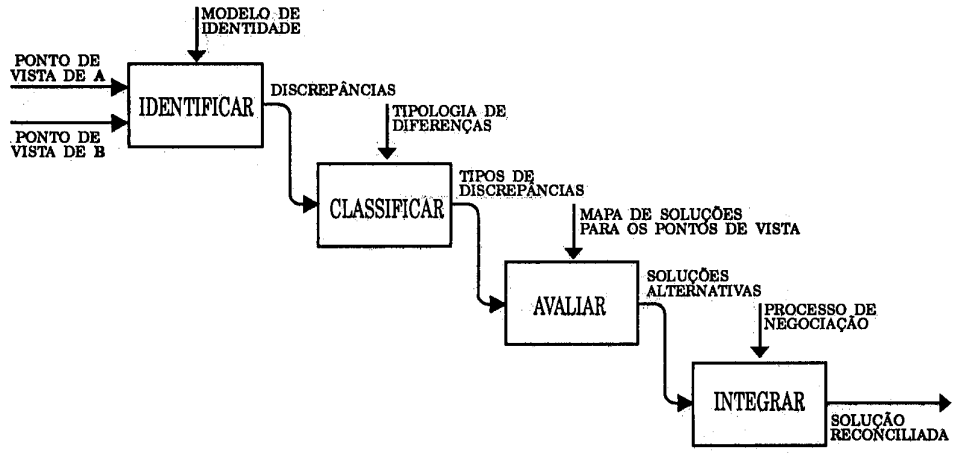
Portanto a formação desse domínio seria baseado em fatos de sistemas similares que já tenham sido elicitados. O uso dessa técnica permitiria que ferramentas do tipo assistentes inteligentes fornecessem críticas sobre os requisitos preparados pelo engenheiro de software. Dada a disponibilidade de um domínio, é possível determinar `fatos errados` e *fatos faltantes*. No caso do uso da técnica de reutilização de domínios, a computação Delta é controlada por fatos errados e fatos faltantes:

$$\Delta = \begin{cases} \textbf{fatos errados} \text{ (fatos do domínio, fatos)} \\ \textbf{fatos faltantes} \text{ (fatos do domínio, fatos)} \end{cases}$$

O uso de domínios para a validação possibilita que seja feita uma diferenciação entre problemas de correteza e completeza, facilidade não presente nas atuais técnicas para validação. O problema com o uso de domínios é, não só o seu alto custo, como também a complexidade de sua construção [3].

A seguir descrevemos o conceito de pontos de vista e sua aplicação na tarefa de elicitação de requisitos, em especial detalharemos os problemas encontrados para a aplicação de pontos de vista como uma estratégia de validação. Chamamos de *resolução de pontos de vista* a esta estratégia de validação.

FIGURA 4
RESOLUÇÃO DE PONTOS DE VISTA



3 — PONTOS DE VISTA NA ELICITAÇÃO DE REQUISITOS

Na tarefa de modelar as expectativas dos usuários dentro de um universo de informações, o engenheiro de software pode encontrar, e usualmente encontra, opiniões diferentes sobre o problema em questão. Diferentes engenheiros de software, quando modelando as expectativas de usuários num **mesmo** universo de informações, produzem diferentes modelos. O mesmo engenheiro de software, quando modelando o mesmo universo de informações pode fazê-lo usando diferentes perspectivas (por exemplo, um modelo de dados *x* um modelo de processos). Tudo isso é conhecido. O ponto importante é que alguns métodos de engenharia de software usam pontos de vista com o objetivo de produzir um modelo que "melhor" espelhe as expectativas dos usuários no universo de informações. Um exemplo de tal método é CORE [24].

O princípio de que mais fontes de informação proporcionam um melhor entendimento de um caso tem sido usado por séculos em tribunais. Testemunhas diferentes podem ter lembranças diferentes ou complementares. Usando o mesmo princípio no processo de elicitação, as possibilidades de detectar problemas de correteza e completeza serão maiores. No entanto, para se lucrar com este princípio é necessário comparar e analisar diferentes pontos de vista.

A análise e comparação de pontos de vista como proposto por Ross (SADT) e Mullery (CORE) são tarefas informais. Por serem tarefas informais dependem basicamente de um "bom" engenheiro de software. Apesar de advogarem o uso de pontos de vista, nem CORE nem SADT têm um modelo estruturado que possa realmente tirar proveito desse enfoque. A falta de uma representação própria torna difícil a existência de procedimentos para comparação e análise de visões oriundas de diferentes pontos de vista.

3.1 — RESOLUÇÃO DE PONTOS DE VISTA E SEUS PROBLEMAS

Resolução de pontos de vista é entendido como um processo composto de quatro subprocessos: identificação, classificação, avaliação e integração (Figura 4). De acordo com nossa definição de elicitação, identificação e classificação fazem parte da validação de fatos, enquanto avaliação e integração fazem parte da comunicação.

A resolução de pontos de vista é utilizada no processo de validação dos fatos. A aquisição desses fatos supõe uma orientação baseada na busca de palavras-chave da aplicação e numa representação própria. Portanto, um dos problemas básicos abordados pela nossa pesquisa é: como ter uma representação que possibilite a aplicação de um modelo de resolução de pontos de vista (Figura 4).

Apesar da validação ser um processo que é intrinsicamente ligado ao processo de construir um modelo, é importante ressaltar que a representação usada no processo de validação por pontos de vista **não** pretende ser a mesma usada para o modelo final produzido pela definição de requisitos. A resolução de pontos de vista e a representação aqui proposta têm o objetivo precípuo de ajudar o entendimento do problema, a modelagem é um processo posterior a este entendimento. Portanto a representação usada

na resolução de pontos de vista é limitada. Para se modelar o entendimento da fase de definição de requisitos é necessário um esquema de modelagem conceitual bem mais elaborado do que a representação usada na resolução de pontos de vista.

Antes de melhor definirmos o problema convém estabelecer algumas definições.

Pontos de Vista — Um ponto de vista é uma posição mental usada por uma pessoa quando examinando ou observando um universo de informações. Um ponto de vista é identificado por uma pessoa (e. g., seu nome) e seu papel no universo de informações (e. g., um engenheiro de software, um funcionário, um gerente).

Perspectiva — Uma perspectiva é o conjunto de fatos observados de acordo com um ponto de vista e modelados com um tipo especial de modelagem. Um exemplo de um tipo especial de modelagem é a modelagem conhecida como modelo de dados. Em nosso método, usamos três tipos: a perspectiva de dado, a perspectiva de ator e a perspectiva de processo.

Visão — Uma visão é uma integração de perspectivas. Esta integração é obtida por um processo chamado "construção de uma visão".

VWP1 — A linguagem de pontos de vista é a representação usada para que os fatos e seus relacionamentos sejam descritos dentro do esquema de resolução de pontos de vista. Esta linguagem é baseada em sistemas de produção e usa uma definição de tipos aliada a árvores de hierarquias.

Hierarquias — Hierarquias são usadas como parte da linguagem de pontos de vista com o objetivo de prover uma extensão semântica aos termos definidos na linguagem. Utiliza-se uma hierarquia de especializações (é uma) e uma hierarquia de decomposições (partes de) para aumentar a carga semântica provida pela tipagem das regras de produção.

Nosso trabalho não lida com os problemas de comunicação na resolução de pontos de vista, já que estes envolvem aspectos de negociação. Os problemas relacionados com a negociação entre atores do universo de informações têm sido estudados em autores preocupados com o impacto social dos sistemas de informação [11] [16]. Nossa pesquisa se dedicou aos problemas de identificação de discrepâncias e a classificação dessas discrepâncias. Identificando e classificando discrepâncias entre diferentes pontos de vista estamos criando uma **agenda** que pode orientar a avaliação e a integração desses pontos de vista.

3.1.1 — PROBLEMAS ABORDADOS

Para que o modelo citado (Figura 4) seja usado é indispensável que haja uma estruturação para pontos de vista. Este fato caracteriza o primeiro problema:

Como formalizar pontos de vista?

Na Figura 4, retângulo 1, é claramente dependente do formalismo proposto e levanta a seguinte questão.

Como formalmente comparar pontos de vista?

O retângulo 2 da citada Figura coloca em questão o seguinte problema.

O quanto se pode diferenciar entre problemas de conflito e falta de fatos entre pontos de vista, e como classificar os tipos de diferenças entre estes pontos de vista?

O principal problema abordado por nossa pesquisa é o seguinte:
No processo de definição de requisitos, o problema reside em como diferenciar entre informações incorretas e falta de informações, sem contar com um domínio já previamente codificado.

Para que este problema possa ser estudado é necessário que as primeiras duas perguntas sejam respondidas. A próxima seção dá uma explicação global sobre a representação proposta bem como sua utilização no contexto da validação de fatos.

4 — A ESTRATÉGIA PROPOSTA

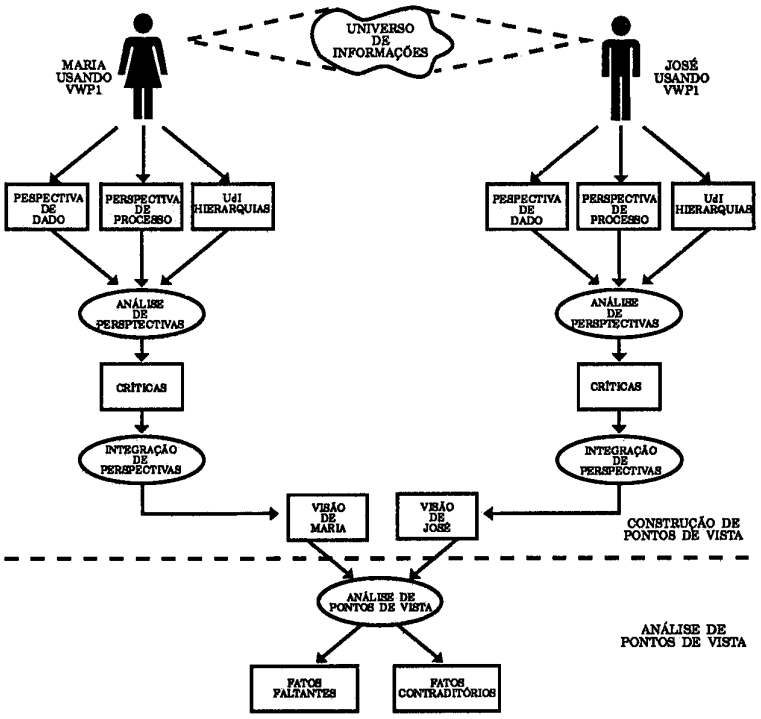
A estratégia proposta é composta de um método e uma linguagem, VWPl, de representação de pontos de vista. O método tem procedimentos para a formalização de pontos de vista, bem como procedimentos para a análise desse formalismo. A linguagem é o meio que codifica o formalismo e torna possível sua análise.

A linguagem é derivada de PRISM [17], uma arquitetura de sistemas de regras. Portanto, nossa estratégia é basicamente um processo que compara duas bases de regras, cada uma representando um ponto de vista diferente.

Conforme observado na Figura 3, o processo de validação de fatos é dependente do processo da coleta de fatos. Supõe-se que os fatos (palavras-chave) estão à nossa disposição antes da aplicação da resolução de pontos de vista. Maior detalhe sobre coleta de fatos, bem como a codificação de pontos de vista estão descritos em [21].

Em seguida, apresentamos uma sucinta descrição do método para a produção de pontos de vista, uma descrição da linguagem VWPl e uma descrição dos procedimentos que analisam diferentes pontos de vista.

FIGURA 5
A ESTRATÉGIA PROPOSTA



4.1 — MÉTODO

Na Figura 5 há um descrição geral da estratégia. José e Maria, ambos engenheiros de software, modelam as intenções dos usuários. Ambos usam VWPI para expressar o universo de informações. Eles usam diferentes perspectivas (processo, dado e ator) e diferentes hierarquias (partes-de, é-uma) com o objetivo de criar uma visão. Uma vez de posse de críticas, cada analista resolve os conflitos e integra sua percepção final em uma visão. Esta visão é expressa usando a perspectiva processo em conjunto com as hierarquias. Depois disso, os pontos de vista de José e Maria são comparados e analisados.

Portanto, de modo a identificar e classificar discrepâncias entre pontos de vista diferentes, visões devem ser extraídas de cada ponto de vista. Visões são produzidas por um processo chamado construção de visões. A construção de uma visão é baseada no seguinte:

- a disponibilidade de um método de coleta de fatos,
- um engenheiro de software de posse de um determinado ponto de vista, usa perspectivas e hierarquias para modelar um ponto de vista,
- perspectivas e hierarquias são analisadas por um analisador estático, e
- uma visão é um modelo integrando as diferentes perspectivas e hierarquias derivadas de um mesmo ponto de vista.

Com a disponibilidade de duas visões, torna-se possível comparar diferentes pontos de vista.

Conforme observado antes, é de conhecimento geral que engenheiros de software quando modelando o universo de informações, o fazem usando diferentes perspectivas. Um exemplo disso é o uso de modelos de dados e modelos de processos. Nossa pesquisa além desses usuais modelos põe em evidência os atores envolvidos no universo de informações [20]. A idéia de se explicitar os atores é de usar a perspectiva daqueles que são responsáveis pelos processos, isto é agentes humanos e agentes físicos.

O objetivo do uso de hierarquias [32] é a de tentar embutir alguma descrição semântica na informação codificada em VWPl. As hierarquias são compostas de relações de especialização entre palavras-chave e de relações de decomposição entre palavras-chave.

Para construir uma visão, o engenheiro de software descreve o problema usando as três perspectivas e duas hierarquias. As perspectivas e hierarquias são comparadas e são produzidas: uma "lista de discrepâncias" e os "tipos de discrepâncias". Uma visão é a integração de perspectivas e de hierarquias conforme um ponto de vista e com o auxílio de uma "agenda", produzida pela análise de perspectivas.

A construção de uma perspectiva é um processo no qual se assume o uso do conceito de **vocabulário da aplicação** [10], de tal forma que as palavras-chave do universo de informações são utilizadas na representação de pontos de vista.

A idéia principal é a de que o engenheiro de software primeiro analisa o problema e anota suas observações usando a perspectiva de ator em VWPl. As observações descritas usando as outras perspectivas são feitas independentemente e em tempos diferentes. Do mesmo modo, as observações relativas às hierarquias são codificadas.

Supõe-se que a comparação dessas perspectivas e hierarquias produzidas por um mesmo engenheiro de software possibilitará que este produza uma perspectiva (processo) e hierarquias finais de melhor qualidade. Esta perspectiva e hierarquias são então consideradas a representação do ponto de vista selecionado.

São as seguintes as diretrizes para a aquisição de perspectivas e hierarquias.

- A produção das hierarquias "é-uma" e "partes-de" constantes do universo de informações.
- Para cada perspectiva:
  - — achar os fatos;
  - — expressar os fatos usando as palavras-chave do domínio de aplicação;
  - — classificar os fatos em: fatos objetos, fatos ações e fatos agentes; e
  - — definir funcionalmente os fatos usando o formalismo de produções.
- Objetos representam tanto os objetos como os estados desses objetos.
- Não há imposições para a produção de hierarquias, o detentor do ponto de vista é quem decide o que deve ou não estar presente nas hierarquias.
- O intuito é a de deixar estas linhas gerais um pouco frouxas de modo a facilitar ao engenheiro de software, a expressão do ponto de vista em questão.

É importante ressaltar o meta uso da estratégia de resolução de pontos de vista. Ela é aplicada na própria construção de um ponto de vista. Isto é, a estratégia para validação de pontos de vista é usada para checar os modelos de perspectiva (dado, processo, ator). A seguir apresentaremos o suporte representacional ao método proposto.

4.2 — A LINGUAGEM DE PONTOS DE VISTA

Embora não seja nosso objetivo lidar com a modelagem de requisitos[3], é mister que um esquema representacional esteja disponível. No nosso caso utilizamos um modelo conceitual muito simples, composto de:

- três tipos de entidades: objeto, ação e agente;
- atributos;
- hierarquias: é-uma e partes-de;
- relações. As relações são de dois tipos:
  - — < entidade, atributo>, as quais constituem fatos, e
  - — a< fatos, fatos>, as quais definem funcionalidade.

Este modelo conceitual é instanciado através de uma linguagem especial chamada VWPl. VWPl é uma linguagem derivada de PRISM [17], uma arquitetura para sistemas de produção. Não sendo uma linguagem de requisitos, sua utilidade é restrita ao processo de validação de fatos.

A idéia básica atrás de VWP é a de ter uma estrutura predefinida para a construção de regras. A imposição de maior restrição no schema usual de lado direito (RHS) e lado esquerdo (LHS) de uma regra, torna possível que tenhamos alguma informação de ordem semântica na estrutura das regras. O enfoque usado em VWPl é semelhante aos usados em *case grammars* [28], onde as estruturas produzidas pelas regras da gramática correspondem a relações semânticas ao invés de somente relações sintáticas.

No caso de VWPl, as relações semânticas são expressas pelo uso do que chamamos restrições de *tipos* e de *classes*. Tipos são os diferentes fatos usados, isto é, fatos **objeto**, fatos **açao** e fatos **agentes**. Classes são os diferentes papeis que cada fato pode ter numa regra. Numa regra um fato pode:

[3] *Outros autores têm centrado sua atenção neste tema, em particular a linguagem RML [12], é um dos mais completos modelos conceituais aplicados à modelagem de requisitos.*

- ser retirado da memória de trabalho (nós chamamos a isto de *classe de entrada*),
- ser adicionado à memória de trabalho (nós chamamos a isto de *classe de saída*), ou
- permanecer na memória de trabalho (isto é chamado de *classe invariante*).

Um fato é uma relação entre palavras-chave. As palavras-chave para expressar fatos em VWPl são checadas contra uma lista de tipos antes de serem analisadas sintaticamente. Em outras palavras, um teste de pertinência semântica é efetuado nas palavras-chave disponíveis para a descrição de uma visão[4]. Um fato é composto de uma palavra-chave-fato e um atributo-fato. Um exemplo é "(livro =ident-livro =autor =titulo)", onde **livro** é a palavra-chave-fato, e *ident-livro, autor e titulo* são os atributos-fato.

Para cada perspectiva há uma combinação especial de tipos e de classes. Neste artigo somente usamos a perspectiva de dado e de ator. Para a PERSPECTIVA DE DADO usamos a seguinte estrutura de regras:

- LHS (lado esquerdo) — a *entrada* é um **objeto**, e a *invariante* pode ser um **agente** ou um **objeto**.
- RHS (lado direito) — a *saída* é uma **ação**.
- Para a PERSPECTIVA DE ATOR usamos a seguinte estrutura.
- LHS (lado esquerdo) — a *entrada* é um **agente** e a *invariante* pode ser um **agente** ou um **objeto**.
- RHS (lado direito) — a *saída* é uma **ação**.

Hierarquias são codificadas como listas. As listas são organizadas pelo tipo de hierarquia (é uma, ou partes-de). Para cada tipo a raiz da hierarquia é a cabeça de uma lista seguida dos ramos daquela hierarquia.

Na Figura 6 damos um exemplo do uso da linguagem VWPl. Este exemplo mostra como uma simples sentença foi codificada de acordo com a perspectiva usada.[5]

FIGURA 6
EXEMPLO

```
"Um oficial do navio tem, como um de seus
deveres, a tarefa de limpar o deck do navio, e
ele pode usar diferentes tipos de vassouras"

REGRAS:
                        Perspectiva de Ator

(10  ( (oficial =patente =nome) (iate =numero-de-registro))
     ->
     ( ($delete-from wm (oficial =patente =nome))
      ($add-to wm (limpar-o-deck =numero-de-registro))))

; O agente "oficial" (entrada) executa a acao "limpar-o-deck" (saida)
; no objeto "iate" (invariante).

                        Perspectiva de Dado

(20  ((oficial =nome) (barco =numero-de-registro)
      (vassoura =tipo))
     ->
     ( ($delete-from wm (vassoura =tipo))
      ($add-to wm (limpar-o-deck-com-vassoura =tipo =numero-de-registro))))

; O objeto "vassoura" (entrada) e usado pelo agente "oficial"
; (invariante) para executar a acao "limpar-o-deck-com-vassoura" (saida)
  no objeto "barco" (invariante)

                        Hierarquias
(is-a (2 (navio iate barco)))
(parts-of (2 (navio deck)))
```

5 — O ANALISADOR ESTÁTICO

A análise de diferentes perspectivas e de diferentes visões é efetuada por uma série de heurísticas, que fazem uma análise de dois conjuntos de perspectivas ou visões. Conforme já observado esta análise é realmente uma análise entre dois conjuntos de regras (Figura 7).

Comparar dois grupos de regras só faz sentido quando há similaridade entre eles. No nosso caso, existe uma série de fatores que nos levam a esta similaridade; abaixo listamos os mais importantes.

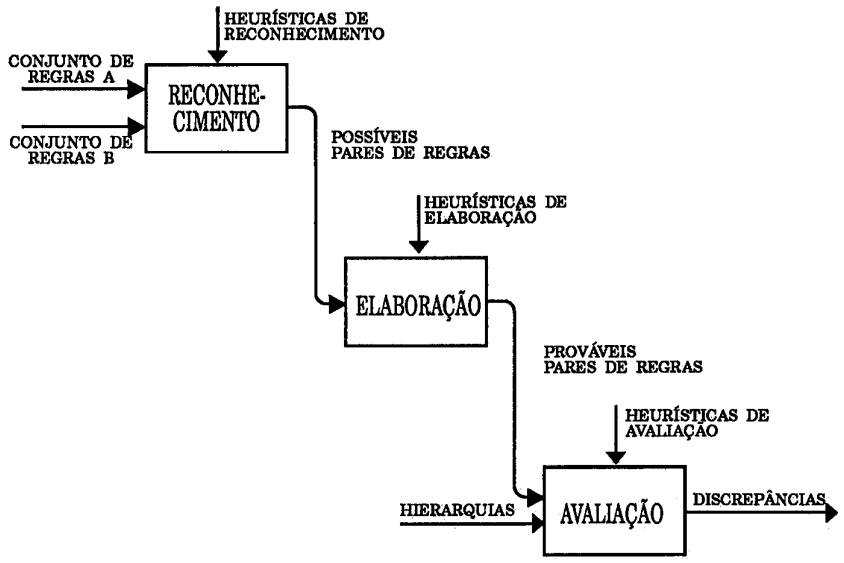
- Os detentores de pontos de vista estão se baseando no mesmo universo de informações.
- O uso de um método em que o uso do conceito de "vocabulário de aplicação" norteia a coleta de fatos.
- O uso de uma linguagem especial que restringe como as regras são expressas.

O analisador estático proposto e implementado em **Scheme** [1] tem duas importantes tarefas: achar quais as regras quem têm similaridade entre si, e uma vez que regras dos diferentes conjunto de regras são identificadas como similares, identificar e classificar as discrepâncias existentes entre elas. Regras que não encontram similares são classificadas como falta de informação. Apesar do analisador ser centra-

[4]A gramática completa de VWP1 pode ser encontrada em [21].

[5]Este exemplo "de brinquedo" tem somente o objetivo de dar uma idéia da linguagem VWPl. Em nosso estudo de casos [21], exemplos com até 20 regras para cada perspectiva ou ponto de vista foram utilizados.

FIGURA 7
ANALISADOR ESTÁTICO



do em uma análise sintática há o uso de descritores de semântica tais como: hierarquias e *case grammars*.

Sendo baseado na representação sintática de termos, o analisador utiliza-se de reconhecedores de padrões e reconhecedores parciais. Estes procedimentos de reconhecimento de padrões são aplicados entre fatos de dois conjuntos de regras diferentes, e têm diferentes algoritmos de *score* dependendo da informação semântica disponível sobre tipos e classes em cada fato. No projeto do analisador estático usamos várias idéias emprestadas do trabalho de Inteligência Artificial no campo da analogia [13].

Hall [13]propõe o seguinte esquema para examinar propostas computacionais para a analogia:

- reconhecimento de um problema análogo,
- elaboração de um mapeamento entre fonte e destino,
- avaliação da analogia elaborada, e consolidação da informação gerada enquanto usando a analogia.

Em particular nosso método lida com três desses processos (Figura 7), sendo que cada um se caracteriza como um subproblema da análise de pontos de vista. São os seguintes os problemas.

Reconhecimento — Dado dois conjuntos de regras em VWPI, produzir os pares mais parecidos entre si.

Elaboração — Dado os pares mais parecidos entre si, identificar os pares mais prováveis, bem como as regras sem par.

Avaliação — Dado os pares mais prováveis, identificar os fatos como contraditórios, ou como faltantes, ou como inconsistentes.

No processo de construir um mapa entre dois conjuntos de regras, a solução de cada subproblema diminui o espaço de busca. Nesta redução do espaço de busca, os pares mais prováveis são identificados de tal maneira que uma análise mais detalhada torna-se possível. Para cada par, os fatos são comparados com objetivo de encontrar discrepâncias. A classificação de discrepâncias, isto é, determinar quais as discrepâncias relativas à falta de informação e quais as discrepâncias relativas a informações contraditórias, é feita baseada em scores e nas hierarquias. É óbvio que o analisador estático não pode afirmar nada sobre duas regras que não têm discrepâncias, mas que na realidade não estão corretas com respeito ao universo da informação.

Se tomarmos o exemplo dado anteriormente, o analisador estático nos fornecerá as seguintes mensagens:

Fatos Contraditórios:

```
(20 (1020 (((30 limpar-o-deck =numero-de-registro)
            30  limpar-o-deck-com-vassoura
                =tipo =numero-de-registro . 0.5))))

mensagem 20: "as regras 10 e 20 tem
              diferentes fatos para representar
              uma acao considerada similar"

 (1 (1020 1 ((oficial) oficial) (=patente =nome) =nome)))

mensagem 1: "as regras 10 e 20 tem atributos diferentes para o mesmo
             agente "officar""

    (4 (1020 (((22 iate =numero-de-registro) 22 barco
                                        =numero-de-registro) . 0.66)))

mensagem 4:"de acordo com a hierarquia
            e-uma os respectivos objetos
            estao em contradicao"
```

Fatos Faltantes:

```
          (16 (1020((vassoura) .21)))

mensagem 16: ''objeto ''vassoura"
             esta faltando na regra 10"
```

6 — ESTUDO DE CASOS

Experimentos realizados demonstraram a eficácia da estratégia proposta. Estes estudos estão detalhados em nossa tese [21] e num artigo apresentado no quinto IWSSD *International Workshop on Software and Specification* [22]. Estes estudos confirmam a eficácia do analisador estático em apontar e classificar discrepâncias entre pontos de vista. Nossos resultados ajudam a mostrar que a comparação de diferentes percepções de um problema ajuda no entendimento deste problema.

Em [21] dois tipos de problemas e quatro pessoas fizeram parte do estudo de caso. Em [22], o aparentemente simples problema da biblioteca (IWSSD) foi usado para mostrar como um esquema antecipado de validação baseado em pontos de vista foi capaz de identificar e classificar problemas relacionados a correteza e completeza. Em seguida, examinaremos o estudo de caso baseado no problema da biblioteca.

6.1 — O PROBLEMA DA BIBLIOTECA

O estudo de caso reportado em [22] é baseado em quatro artigos apresentados no 4º IWSSD: *A Larch Specification of the Library Problem* [33], *What Does it Mean to Say that a Specification is Complete* [35], *Toward a Requirements Apprentice* [29], e *An Executable Specification Language* [18]. Cada um desses artigos tem uma abordagem diferente; o trabalho de Wing usa uma linguagem formal, Yue se baseia nas noções de causação e condicionais em lógica, Rich usa noções de domínios de conhecimento e Lee se baseia numa linguagem executável. Todos os artigos lidam com o problema da biblioteca. Três análises foram feitas. A primeira comparando Rich com Yue, a segundo comparando Yue com Wing e a terceira comparando Wing com Lee. Esta comparação foi feita usando-se apenas uma parte da especificação de cada autor, devidamente traduzida, por nós, para a notação VWPl.

A comparação automática obteve resultados bastante semelhantes aos observados por [34] que usou uma comparação manual. Wing [34] fez um estudo detalhado de 12 artigos, apresentados no 4º IWSSD, que procuravam especificar o problema da biblioteca. Este estudo comparou as diferentes abordagens e como elas revelam problemas na descrição do exemplo da biblioteca. A comparação feita por Wing produziu uma lista de problemas de ambigüidade e completeza na descrição da biblioteca. Ela apropriadamente indicou que: *the interesting result of the specification exercise is not the specification itself but the insight gained about the specificand*, isto é, que no trabalho de especificação, o resultado importante é o conhecimento sobre o problema. Na nossa opinião Wing fez uma resolução de pontos de vista sobre os 12 casos analisados. Nossa análise automática de 4 desses artigos produziram resultados muito parecidos com os de Wing, o que fortalece nossa hipótese sobre a eficácia da comparação de pontos de vista para um melhor entendimento do problema.

7 — CONCLUSÃO

Nosso trabalho deve ser entendido no contexto de uma validação antecipada dentro do processo da elicitação de requisitos. Nossa tese postula que a resolução de pontos de vista é uma estratégia eficaz para suportar a validação durante o processo de elicitação. Nesta seção fazemos considerações sobre a limitação de nossa estratégia e quais as direções promissoras para futuras pesquisas.

7.1 — COMPARAÇÃO COM OUTRAS ESTRATÉGIAS DE VALIDAÇÃO

Como vimos na Seção 2, várias estratégias de validação podem ser usadas na fase da definição de requisitos. Usando o mesmo esquema utilizado anteriormente, podemos distinguir a resolução de pontos de vista como sendo capaz de distinguir entre inconsistências, fatos errados e fatos faltantes.

$$\Delta = \begin{cases} \textbf{inconsistências } (fatos_a, fatos_b) \\ \textbf{fatos errados } (fatos_a, fatos_b) \\ \textbf{fatos faltantes } (fatos_a, fatos_b) \end{cases}$$

Se compararmos com as estratégias já vistas, podemos apontar as seguintes vantagens (+) e desvantagens (-) da resolução de pontos de vista.

(+) automação.
(+) capacidade de diferenciar entre inconsistências, fatos errados e fatos faltantes.
(+) independente de análise de domínios.
(-) dependente na qualidade dos pontos de vista.
(-) o custo da redundância.

O ponto importante dessa estratégia é a qualidade da entrada na computação do Delta. Uma vez possuindo um conjunto organizado de críticas classificadas segundo a correteza, a completeza e a inconsistência, as chances de expor conhecimento implícito são melhores, porque há uma menor dependência em observadores e leitores.

7.2 — LIMITAÇÕES DE NOSSA PROPOSTA E FUTUROS TRABALHOS

Nossa proposta se comparada com as apresentadas na Seção 2, tem, em nosso entender, duas características que a diferenciam daquelas. Primeira, nossa estratégia é voltada para a fase de aquisição de requisitos. Segunda, nossa estratégia foi efetivamente operacionalizada através de um protótipo e usada por outras pessoas em experimentos controlados. Principalmente por esta segunda característica acreditamos que nossa estratégia poderia ser transferida para a prática, desde de que esforços fossem dispensados em adequa-la para uso prático.

A adequação de nossa estratégia para uso prático, deixaria de lado alguns pontos que ainda precisam de mais pesquisa e se concentraria na parte de engenharia da nossa proposta, com o objetivo de prover a qualidade necessária para torná-la utilizável. Acreditamos que o grande trunfo de nosso esquema é a sua simplicidade, facilitando, portanto, o seu uso. Certamente um dos primeiros pontos a serem aperfeiçoados seria a interface com o usuário, que hoje praticamente inexiste. A simplicidade do esquema conceitual, que é muito semelhante a tabelas de decisão, precisa, no entanto, de uma melhor estrutura para suportar problemas mais complexos.

Outros pontos poderiam ser aprimorados, principalmente as heurísticas do analisador, que contando com melhores reconhecedores de padrão evitariam algumas mensagens não relevantes. Estas mensagens em sua maioria são decorrentes de pares falsos entre conjunto de regras. Em nossa tese [21] detalhamos vários possíveis melhoramentos. Recentemente, um trabalho de iniciação científica [23] implementou um *prettyprinter* para a tabela de mensagens geradas pelo analisador, cuidando também, em parte, de um dos problemas mencionados em [21], isto é, o de mensagens redundantes.

Nossa expectativa é a evolução do analisador para ser uma ferramenta integrante de um ambiente de apoio a aquisição de requisitos. Uma de nossas linhas de pesquisa é justamente a de prover ao engenheiro de software métodos e ferramentas que suportem a tarefa de definição de requisitos, onde a ênfase é na elicitação e não na modelagem.

A seguir concluímos fazendo comentários finais enfatizando a contribuição de nosso trabalho e ressaltando o papel da negociação nas fases de avaliação e integração que se seguem à análise de pontos de vista.

7.3 — COMENTÁRIOS FINAIS

Neste trabalho apresentamos uma caracterização da tarefa de definição de requisitos, em que a ênfase maior é na noção de processo. Nesta caracterização, ressaltamos a importância da validação como uma tarefa do processo de elicitação dos requisitos. Nossa tese central é de que um esquema de validação baseado em pontos de vista é eficaz na tarefa de entendimento dos requisitos. Apresentamos um método para validar requisitos com base na comparação automática de pontos de vistas diferentes sobre o mesmo problema, e comentamos sobre experimentos empíricos usados para a validação de nossa tese. Esses experimentos apontaram que o uso de uma **agenda** para a reconciliação de pontos de vista é uma maneira eficaz de sanar, logo cedo, problemas relativos a consistência e completeza de um conjunto de requisitos. Examinando diferentes visões sobre um mesmo problema, estamos obtendo um meta entendimento sobre o que Wing chama de *specificand*.

O resultado básico da aplicação de nossa estratégia é a produção de uma agenda que norteará a reconciliação de pontos de vista. A necessidade de um consenso é importante para a consistência dos requisitos, bem como é fundamental para um melhor entendimento do problema. Observamos em nossos experimentos, que a tarefa de negociação entre diversos atores do UdI não é trivial, mas vale lembrar que estes problemas recaem na área de impactos sociais [16]. Portanto, apesar de fornecer uma base para a tarefa de resolução de pontos de vista, não nos propusemos a atacar os problemas de negociação que afetam este processo social entre vários atores do universo de informações.

8 — AGRADECIMENTOS

Este trabalho é baseado em minha tese de doutorado e em um artigo em co-autoria com o Professor Peter Freeman.

O doutorado foi feito com o apoio do CNPq.

REFERÊNCIAS BIBLIOGRÁFICAS

1— Abelson, H., Sussman, G., and Sussman, J. *Structure and Interpretation of Computer Programs.* MIT Press, Cambridge, Massachusetts, 1985.

2— Agresti, W. *In New Paradigms for Software Development,* W. Agresti, Ed., IEEE Computer Society, Long Beach, CA 1987.

3— Arango, G. Domain Analysis: From Art Form to Engineering Discipline. *In 5th International Workshop on Software Specification and Design* (Pittsburgh, PA, 1989), IEEE Computer Society Press, pp. 152-159.

4— Brooks, F. No Silver Bullet, Essence and Accidents to Software Engineering. *IEEE Computer 4,* 3 (Apr.1987),10-19.

5— Daniels, A. and Yeates, D. *Formação Básica em Análise de Sistemas.* Série LTC/LTD. 1974.

6— Davis, G. *Strategies for Information Requirements Determination,* IBM Systems Journal 21, 1 (Jan. 1982), 4-30.

7— Fickas, S. Automating the Analysis Process: An Example. *In 4th International Workshop on Software Specification and Design* (Monterey, CA, 1987), IEEE Computer Society Press, pp. 58-67.

8— Finkelstein, A. and Waters, R. Summary of the Requirements Elicitation, Analysis and Formalization Track. *ACMSIGSOFT,* Vol. 14, Nº 5, (Jul 1989), pp. 40.

9— Freeman P. *Software Perspectives, The System is the Message,* Addison Wesley, Reading Mass., 1987.

10— Freeman P. and Leite J., Requirements Techniques and Languages: A Survey. *Dept. of* Computer *Science, University of California Irvine,* (1988).

11— Gerson,E. and Star, S. Analyzing Due Process in the Workplace. *ACM Trans. on Office Inf. systems* 4, 3 (Jul.1986), 257-270.

12— Greenspan, S. *Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition.* PhD thesis Computer

13— Research Group, University of Toronto, Mar. 1984. Hall, R. Computational Approaches to Analogical Reasoning: A Comparative Analysis *Artificial*

14— *Intelligence 21,* 1 (Jan. 1988), 241-250. Horowitz, E., Kemper, A. and Narasimhan, B. A

15— Survey of Application Generators. *IEEE Computer 18, 1* (Jan. 1985), 40-54. Jones, C. *Systematic Software Development Using VDM.* Prentice-Hall International, 1986.

16— Kling, R. Social Analysis of Computing:Theorectical Perspective in Recent Empirical Research. *Computer Surveys,* pp.61-110 March, 1980.

17— S. Ohlsson and P. Langley; *PRISM Tutorial and Manual.* Tech. Rep. 86-02, University of California, Irvine - Dept. of Computer Science, Feb. 1986.

18— Lee S. and Sluizer S. SXL: An Executable Language. *In 4th International Workshop on Software Specification and Design* (Monterey, CA, 1987), IEEE Computer Society Press, pp. 231-235.

19— Leite, J. *A Survey on Requirements Analysis,* Tech. Rep. RTP 070, Dept. of Comp. Science, Univ. of California, Irvine, Jul.1987.

20— Leite, J. *The Agent Viewpoint.* Tech. Rep. RTP 070, Dept. of Comp. Science, Univ. of California, Irvine, Mar. 1987.

21— Leite, J. *Viewpoint Resolution in Requirements Elicitation.* PHD thesis, Dept. of Comp. Science, Univ. of California, Irvine, 1988.

22— Leite, J. Viewpoint Analysis: A Case Study. *In 5th International Workshop on Software Specification and Design* (Pittsburgh, PA, 1989), IEEE Computer Society Press, pp.111-119.

23— Leite, J. e Aguilar, K. Implementação de um Pretty-printer: O Uso da Linguagem Scheme. *Departamento de Informática, PUC-RIO,* em preparação.

24— Mullery, G. CORE - A Method for Controlled Requirement Specification. In *Proc. 4th Int. Conf. on Software Engineering* (1979), IEEE Computer Society Press, pp. 126-135.

25— Ould, M. and Vuwin, C. *Testing in Software Development.* British Computer Society and Cambridge University Press, Cambridge, UK, 1986.

26— Pressman, R. *Software Engineering: A Practitioner's Approach.* Mc Graw-Hill, New York, 1988.

27— Reubenstein, H. and Waters, R. The Requirements Apprentice: An Initial Scenario. *In 5th International Workshop on Software Specification and Design* (Pittsburgh, PA, 1989), IEEE Computer Society Press, pp.211-218.

28— Rich E. *Artificial Intelligence.* Mc Graw-Hill, New York, 1983.

29—Rich, C., Waters, R., and Reubenstein, H. Toward a Requirements Apprentice. *In 4th International Workshop on Software Specification and Design* (Monterey, CA, 1987), IEEE Computer Society Press, pp. 79-86

30— Ross, D. Structured Analysis (SA): A Language for Communicating Ideas. In *Tutorial on Design Techniques,* Freeman and Wasserman, Eds., IEEE Computer Society Press, Long Beach, CA 1980, pp. 107-125.

31— ACM Sigsoft. Special Issue on Rapid Prototyping. Dec. 1982. Several Authors.

32— Tanimoto, S. *The Elements of Artificial Intelligence.* Computer Science Press, Potomac, Maryland, 1987.

33— Wing J. A Larch Specification of the Library Problem. *In 4th International Workshop on Software Specification and Design*) (Monterey, CA, 1987), IEEE Computer Society Press, pp. 34-41.

34— Wing, J. A Study of 12 Specification of the Library Problem. *in IEEE Software 5,* 4 (Jul./1988), 66-76.

35— Yue K. What Does It Mean to Say that a Specification Is Complete? *In 4th International Workshop on Software Specification and Design* (Monterey, CA, 1987), IEEE Computer Society Press, pp. 34-41