ISBN: 1646-8929

IET Working Papers Series
No. WPS08/2009

Inês Simão
(e-mail: ines_simao_@hotmail.com)
Patrícia Varela
(e-mail: patricia_varela_@hotmail.com)

A Engenharia de Requisitos como processo inovador nas organizações

IET
Research Centre on Enterprise and Work Innovation
Centro de Investigação em Inovação Empresarial e do Trabalho
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Monte de Caparica
Portugal

A Engenharia de Requisitos como processo inovador nas organizações [1]

[Requirements Engineering as an innovative process in organisations]

Inês Simão (ines_simao_@hotmail.com), MSc student of Computer Sciences Engineering at Universidade Nova de Lisboa, Faculty of Sciences and Technology

Patrícia Varela (patricia_varela_@hotmail.com), MSc student of Computer Sciences Engineering at Universidade Nova de Lisboa, Faculty of Sciences and Technology

Junho de 2009

RESUMO

A Engenharia de Requisitos (ER) é um processo que actualmente tem vindo a ser muito utilizado. Surgiu como um melhoramento das falhas encontradas aquando do desenvolvimento de software, uma vez que muitos erros surgiam em fases finais do ciclo de vida.

A ER pode ser dividida em categorias, como funcionais, não funcionais e organizacionais. As suas principais actividades são a licitação, análise, especificação, validação e gestão de requisitos.

Para o processo de desenvolvimento é frequente recorrer às actividades do ER, para a produção de um documento de requisitos adequado, que posteriormente resulte num software de qualidade, sendo que este processo tem "entradas" e "saídas". Existem

[1] Trabalho realizado sob orientação do Prof. António Brandão Moniz para a disciplina "Factores Sociais da Inovação" do Mestrado Engenharia Informática realizado na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

várias abordagens para desenvolvimento deste mesmo processo, no entanto este trabalho, destaca o modelo em espiral, que consiste organizar o desenvolvimento do software como processo iterativo.
Tanto os clientes actuais, como futuros, bem como os engenheiros de requisitos, são quem beneficia desta metodologia, uma vez que existe uma diminuição dos custos, por parte dos clientes, havendo um melhor controlo dos requisitos por parte do engenheiro e reutilização da documentação produzida. São apresentadas algumas das ferramentas disponíveis no mercado, que ajudam à construção de software. Existem várias técnicas para a identificação e formulação de requisitos, das quais focamos apenas os métodos de requisitos orientados a "viewpoints", fornecendo um exemplo de aplicação prática.
Pode-se salientar que existe ética na Engenharia de Requisitos dando particular relevância à competência, à confidencialidade, aos direitos de propriedade intelectual e na protecção de dados.

Palavras-chave: Engenharia de Requisitos, indústria de aplicações informáticas

ABSTRACT

Requirements Engineering (ER) is a process that currently is being widely used. It emerged as an improvement of failures during the development of software, since many errors appeared in the final stages of the life cycle. ER can be divided into categories such as functional, non functional and organizational. Its main activities are the elicitation, analysis, specification, validation and management requirements.
In the process of development often rely on the activities of the ER, to produce a document of appropriate requirements, which subsequently results in software quality, and this process has inputs and outputs. There are several approaches to develop this same process, however this work, highlighting the spiral model, organizing software development with iterative process. Both actual and future customers and the requirements of engineers are who benefits of this methodology, since there is a reduction of costs, from customers, with better control of the requirements by the engineer and re-use of documentation produced. We focus some tools available in the market, which help the construction of software.

There are several techniques for the identification and formulation of requirements, which focus only the methods of the viewpoints requirements oriented, providing an example of practical application. There are ethics issues in Requirements Engineering with particular emphasis on competence, confidentiality, and intellectual property rights and data protection.

Key-words: Requirements engineering, software industry

JEL codes: D83; O21

Índice

1. Introdução

Este trabalho vai incidir sobre o tema Engenharia de Requisitos. O tema foi escolhido porque era do nosso interesse, pois incide um pouco nos temas da nossa tese de mestrado de Engenharia Informática, que estamos também a desenvolver. É um tema que faz sentido abordar, uma vez que há cada vez mais avanços tecnológicos neste campo. Sendo a Engenharia de Requisitos uma área vasta, decidimos restringir o nosso trabalho ao tema Engenharia de Requisitos orientada a *viewpoints*. Esta escolha deve-se ao facto de que para além de ser uma área em crescimento, tem um grande interesse para nós, tal com referenciado anteriormente.

O trabalho está dividido nas seguintes secções: Secção 1 (Engenharia de Requisitos, onde se apresenta uma definição, as suas actividades, as categorias em que os requisitos sem dividem e todo o processo); Secção 3 (Apresentada a importância da Engenharia de Requisitos); Secção 4 (Onde se disponibiliza algumas das ferramentas desenvolvidas para a área); Secção 5: (Descrição sobre as técnicas utilizadas) e Secção 6: (Ética na Engenharia de Requisitos). Por fim na Secção 7 são apresentadas as conclusões.

2. Engenharia de Requisitos

2.1 O que é a Engenharia de Requisitos

Engenharia de Requisitos (**ER**) é o processo de descobrir a finalidade para a qual um sistema de software se destina [1] [2] [3] [5], cobrindo todas as actividades envolvidas na descoberta, documentação e manutenção de um conjunto de requisitos para um determinado sistema. É feita através da identificação das necessidades dos *stakeholders*, e documentação dos requisitos levantados, em forma de um formulário, sendo esta técnica boa para a análise, comunicação e implementação.

Durante este processo existem muitas dificuldades. Os s*takeholders* podem ser muitos e podem estar distribuídos em diferentes zonas geográficas, em que os seus objectivos podem variar, podendo ocorrer conflitos, como tal os objectivos podem não ser explícitos. **ER** é, portanto, parte do processo de engenharia, estando envolvida com a ligação de actividades de desenvolvimento, de um problema do mundo real, de modo que a sua adequação e a relação custo/eficácia da solução possa então ser analisada. O reconhecimento de um problema a ser resolvido para uma especificação detalhada faz parte também do processo de **ER**.

O uso do termo ‘engenharia’ implica que técnicas sistemáticas e repetitivas possam ser usadas para assegurar que os requisitos de um sistema são completos, consistentes, relevantes, entre muitos outros.

Os requisitos são definidos durante a fase inicial do desenvolvimento de um sistema como uma especificação que deve ser implementada. Eles descrevem como o sistema se deve comportar, além de propriedades e atributos que o sistema deve apresentar. Os requisitos podem também ser vistos como uma restrição no processo de desenvolvimento do sistema [2]. Reflectem as necessidades dos diferentes *stakeholders* para o desenvolvimento de um dado sistema. Estes *stakeholders* são pessoas que estão interessadas no sistema e que têm influência directa ou indirectamente nos requisitos do sistema.

2.2 Actividades da Engenharia de Requisitos

A engenharia de requisitos inclui um conjunto de actividades, como é mostrado na figura que se segue [1] [5].

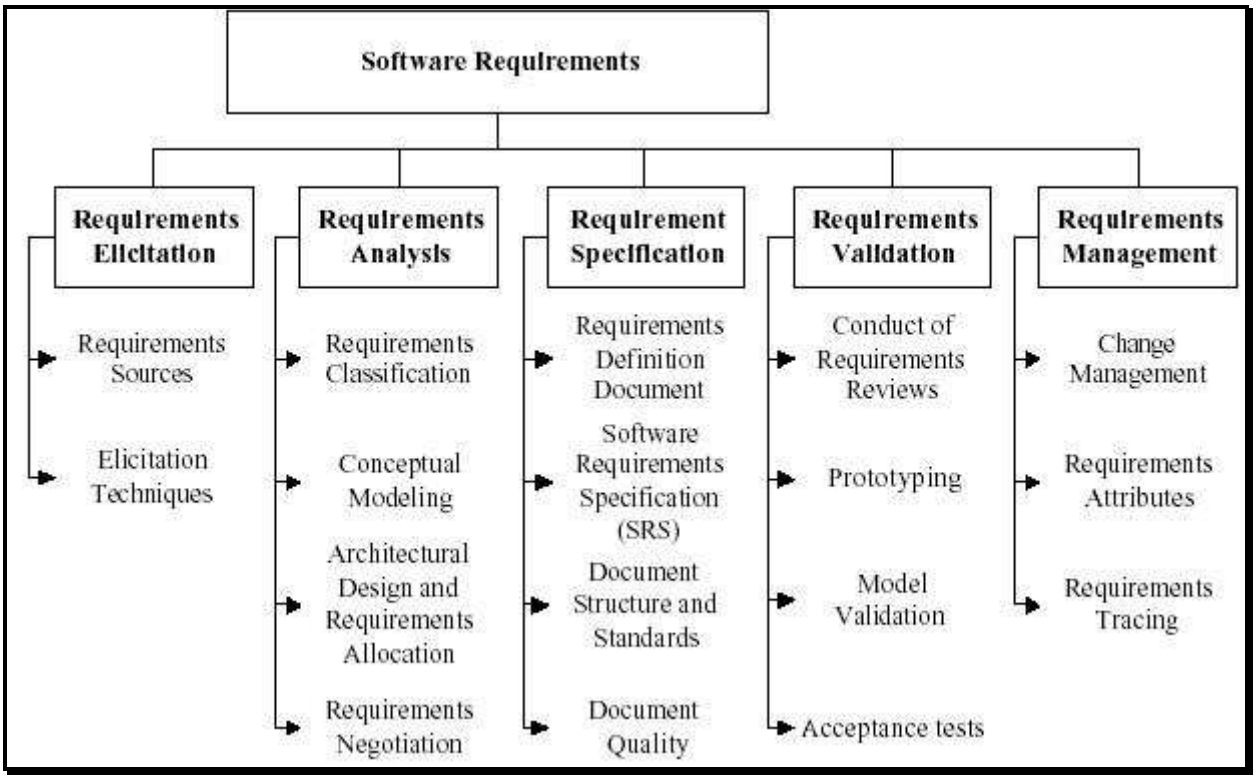

Figura 1 - Actividades de ER

Na primeira secção temos a **identificação de requisitos**. Nesta actividade é efectuado o levantamento dos requisitos de um futuro sistema. O objectivo é identificar, sobre o domínio da aplicação, quais os serviços que o sistema deve proporcionar, qual o desempenho solicitado do sistema e as restrições possíveis. Estes requisitos são capturados através de excelentes fontes de informação, tais como entrevistas, questionários, pesquisas e modelos de processos, documentação de sistemas existentes na organização, entre outros;

Na segunda secção temos a **análise de requisitos**. A análise de requisitos é necessária para resolver os problemas da identificação e alcançar um entendimento nas mudanças

para os requisitos, uma vez que estes requisitos são identificados por diferentes *stakeholders*;

A terceira secção trata da **especificação/documentação dos requisitos**. Nesta actividade são usados diferentes diagramas para descrever os requisitos em vários níveis de detalhe, tais como: Diagramas de Fluxos de Dados (DFD), que servem para modelar dados que são processados por um sistema, em termos de entrada e saída; Diagramas de Entidades e Relações (DER), para relacionar as entidades do sistema; Diagrama de Casos de Uso para modelar os principais serviços de um sistema entre outros;

Na quarta secção processa-se a **validação dos requisitos**. Este processo consiste em demonstrar que os requisitos definem o sistema que o cliente quer, ou seja tem como objectivo verificar o conjunto de requisitos definidos e descobrir os possíveis problemas que ocorram. Este processo envolve *stakeholders* do sistema e os engenheiros de requisitos. Estas validações são realizadas através de algumas técnicas, tais como, revisões de requisitos, prototipagem, geração de casos de teste, análise automatizada de consistência, entre outras;

Por fim a quinta e última secção, trata da **gestão de requisitos**. Permite gerir as mudanças nos requisitos durante o processo de engenharia de requisitos, e o processo de desenvolvimento de sistemas. Esta gestão assegura que a qualidade dos requisitos seja mantida.

2.3 Categorias dos requisitos em Engenharia de Requisitos

Os requisitos em **ER** podem ser categorizados em três tipos: os requisitos funcionais, requisitos não funcionais e requisitos organizacionais [4].

Os requisitos funcionais descrevem, os serviços que o sistema deve oferecer, como o sistema deve reagir a certas entradas, e como o sistema deve comportar-se em determinadas situações [1, 4]; as transformações a serem realizadas nas entradas de um sistema ou em um de seus componentes, a fim de que se produzam saídas; devem ser consistentes e completos.

Os requisitos não funcionais podem ser classificados segundo diferentes categorias [1]:

- **Requisitos de Produto:** definem como o produto se deve comportar. Exemplos de requisitos de produto são os requisitos de eficiência e de confiança.
- **Requisitos Processo:** são consequências das políticas e normas estabelecidas pela organização ou pelo desenvolvedor. Exemplo: requisitos de padrão, de implementação e de entrega.
- **Requisitos Externos:** provêm de factores que são externos ao sistema e a seu processo do desenvolvimento. Exemplo, requisitos legais, que garantem que o sistema está de acordo com a lei vigente.
- **Requisitos organizacionais**: dizem respeito aos objectivos da empresa, às suas políticas estratégicas adoptadas, relacionamento entre os seus actores juntamente com seus respectivos objectivos.

De forma geral, a diferença entre requisitos funcionais e não funcionais está no facto dos primeiros descreverem *o que* o sistema deve fazer, enquanto que os outros fixam restrições sobre *como* os requisitos funcionais serão implementados.
Os requisitos organizacionais dizem respeito às metas da empresa, suas políticas estratégicas adoptadas, os relacionamentos entre os seus intervenientes juntamente com os seus objectivos.

Poderá haver conflitos entre os vários requisitos de qualidade existentes em um sistema, uma vez que podem não estar satisfeitos na sua totalidade, levando a optar pelos mais importantes para o sistema a desenvolver [1, 4].

2.4 Processo da Engenharia de Requisitos

Quando se inicia o desenvolvimento de uma aplicação, é possível que os requisitos iniciais expressos pelos *stakeholders* sejam ambíguos, incompletos e inconsistentes.
Deste modo, é frequente recorrer às actividades do **ER**, para a produção de um documento de requisitos adequado, que posteriormente resulte num software de qualidade.

O processo da engenharia de requisitos é composto por,

- Entradas

    - Informações sobre o domínio;
    - Informações sobre o sistema existente;
    - Necessidades de utilizadores directos e indirectos e demais envolvidos;
    - Leis, normas e padrões;
- Saídas
    - Aceitação de requisito;
    - Especificação do sistema;
    - Modelação do sistema.
- Actividades
    - Análise do domínio: Desenvolver uma compreensão do domínio da aplicação;
    - Recolha de requisitos: Interagir com os *stakeholders* envolvidos;
    - Classificação de requisitos;
    - Resolução de conflitos entre requisitos;
    - Prioridades dos requisitos: Identificar os requisitos mais importantes;
    - Verificação e validação dos requisitos: Verificar se eles são completos, consistentes e de acordo com o que os utilizadores esperam do sistema.

Podemos desenvolver o processo de desenvolvimento através de vários tipos de abordagens, entre as quais está o **modelo em espiral**.

O modelo em espiral foi proposto por Boehm em seu artigo de 1988 *A Spiral Model of Software Development and Enhancement* [6], como forma de integrar os diversos modelos existentes à época, eliminando suas dificuldades e explorando seus pontos fortes. Este modelo foi desenvolvido para abranger as melhores características tanto do ciclo de vida clássico como da prototipagem, acrescentando, ao mesmo tempo, um novo elemento - a análise de riscos - que falta a esses paradigmas.

Entretanto a integração não se dá através da simples incorporação de características dos modelos anteriores. O modelo em espiral assume que o processo de desenvolvimento ocorre em ciclos, cada um contendo fases de avaliação e planeamento, onde a opção de abordagem para a próxima fase (ou ciclo) é determinada. Estas opções podem acomodar características de outros modelos.

O modelo original em espiral organiza o desenvolvimento como um processo iterativo em que vários conjuntos de quatro fases se sucedem até se obter o sistema final. Um ciclo se inicia com a identificação de requisitos, alternativas e restrições. Na segunda tarefa, avaliação de alternativas, identificação e solução de riscos, executa-se uma análise de risco. Prototipagem é uma boa ferramenta para tratar riscos. Se o risco for considerado inaceitável, pode parar o projecto. Na terceira tarefa ocorre o desenvolvimento/documentação do produto. Neste quadrante pode-se considerar o modelo cascata. Na quarta tarefa o produto é avaliado e validado preparando-se para iniciar um novo ciclo.

O modelo espiral é, actualmente a abordagem mais realística para desenvolvimento de software em grande escala, e usa uma abordagem que capacita a empresa que presta o serviço, e o cliente a entender e reagir aos riscos em cada etapa evolutiva.

A figura seguinte ilustra as quatro tarefas importantes anteriormente referidas.

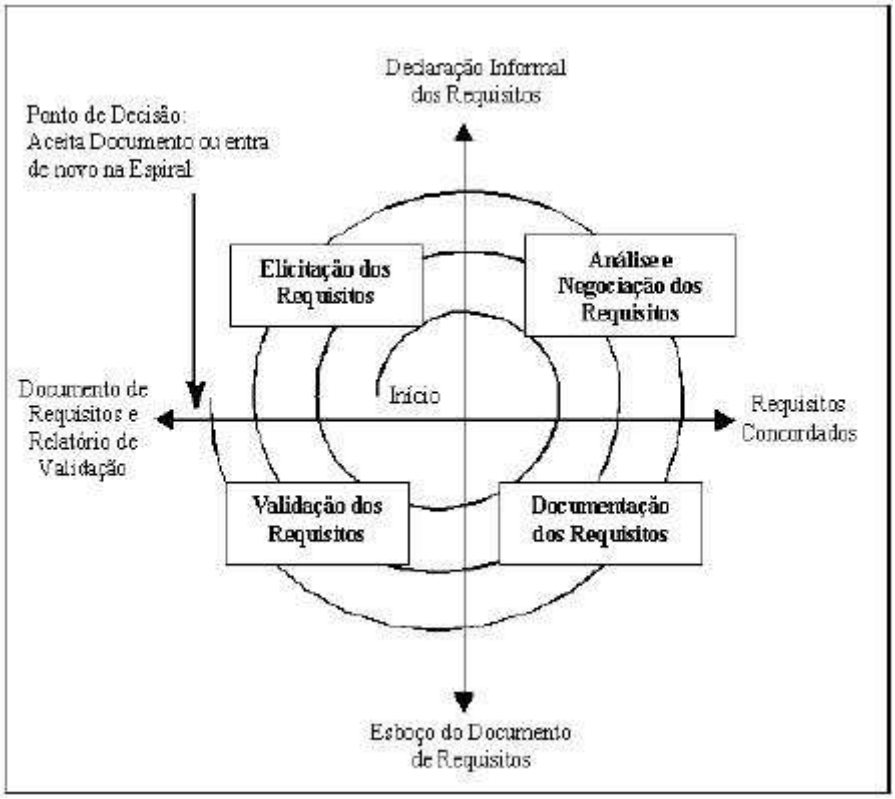

Figura 2 - Modelo Espiral

Algumas das vantagens do modelo em espiral são,

- Permite que ao longo de cada iteração se obtenham versões do sistema cada vez mais completas, recorrendo à prototipagem para reduzir os riscos.
- Este tipo de modelo permite a abordagem do refinamento seguido pelo modelo em cascata, mas que incorpora um enquadramento iterativo que reflecte, de uma forma bastante realística, o processo de desenvolvimento.

Par as desvantagens destaca-se,

- Pode ser difícil convencer grandes clientes (particularmente em situações de contrato) de que a abordagem evolutiva é controlável.
- A abordagem deste tipo de modelo exige considerável experiência na avaliação dos riscos e fia-se nessa experiência para o sucesso. Se um grande risco não for descoberto, poderão ocorrer problemas.
- Este tipo de modelo é relativamente novo e não tem sido amplamente usado.
- É importante ter em conta que podem existir diferenças entre o protótipo e o sistema final. O protótipo pode não cumprir os requisitos de desempenho, pode ser incompleto, e pode reflectir somente algumas facetas do sistema a desenvolver.
- O **modelo em espiral** pode levar ao desenvolvimento em paralelo de múltiplas partes do projecto, cada uma sendo abordada de modo diferenciado, por isso é necessário o uso de técnicas específicas para estimar e sincronizar cronogramas, bem como para determinar os indicadores de custo e progresso mais adequados.

2.5 Técnicas de Validação de Requisitos

Para tornar a validação mais eficaz, existe um conjunto de técnicas que podem ser empregues:

- **Revisões dos requisitos:** Uma equipa de revisores pode analisar sistematicamente a especificação produzida de forma a garantir que esta corresponde ao sistema pretendido; em revisões informais, a equipa de revisores pode simplesmente ter uma conversa, envolvendo o maior número possível de

representantes das partes interessadas, acerca dos requisitos produzidos; em revisões formais, a equipa de revisores deve confirmar junto do cliente um conjunto de critérios que todos os requisitos devem cumprir, nomeadamente: verificação, compreensibilidade (por parte dos utilizadores finais), rastreabilidade (a origem dos requisitos deve ser identificável) e adaptabilidade (capacidade de sofrer alterações sem produzir efeitos em todos os outros requisitos).

- **Prototipagem:** A implementação de um protótipo (por exemplo, da interface do sistema) pode ser útil para os utilizadores finais (e demais interessados), já que se trata do elemento do sistema final com o qual terão mais contacto quando o sistema estiver operacional; esta técnica também é eficaz, embora tenha desvantagens: o tempo gasto na sua implementação pode não justificar o seu uso, pode enviesar os utilizadores (levando a desilusões com a versão final do sistema, no caso de esta ser muito diferente do protótipo) e pode ainda levar os programadores a cair na tentação de usar o protótipo para continuar o desenvolvimento do sistema (pelo que, idealmente, o protótipo deva ser implementado noutra linguagem que não aquela usada pelo sistema, eliminando por completo esta tentação).
- **Geração de casos de teste:** Uma vez que cada requisito deve ser testado, deveria ser possível criar (desenhar) os respectivos testes desde a fase de validação de requisitos; se isto não for possível, é sinónimo de que a implementação deste requisito será difícil e que este poderá ter de ser reconsiderado.
- **Análise de consistência automática:** Através da especificação formal de modelos para o sistema é possível, recorrendo a ferramentas adequadas, testar de forma automática a consistência dos modelos criados; apenas a consistência é testada nesta técnica, pelo que tem de ser complementada com uma das outras técnicas referidas.

3. Importância da Engenharia de Requisitos

A Engenharia de Requisitos surgiu em 1995 [1] por Chaos Report, Standish Group, sendo considerado o "pai" de todos os relatórios sobre falhanços de projectos de software. O Chaos Report mostrava que,

- 31,1% dos projectos são cancelados antes de terminarem;
- 52,7% dos projectos custam mais 89% do que a estimativa inicial;
- Apenas 16% dos projectos são completados a tempo e dentro do orçamento.

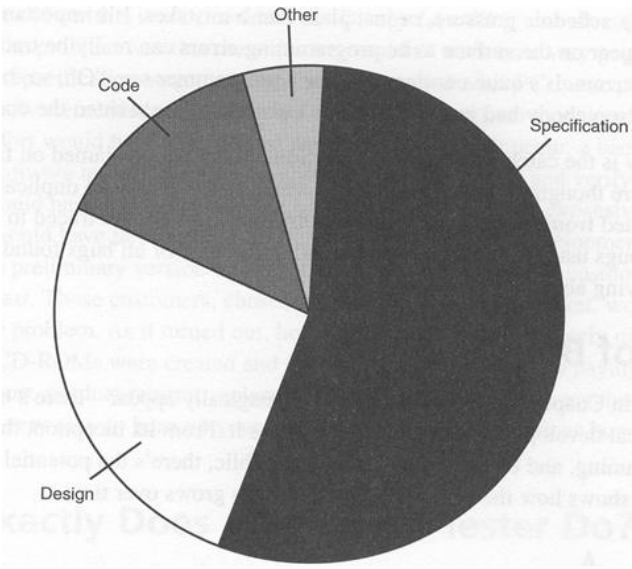

Figura 3 - Bugs são causados por inúmeras razões, mas a causa principal pode ser traçada pela especificação [7]

Este problemas surgiam devido às mais variadas razões, como as seguintes,

- Requisitos incompletos **(13.1%)**;
- Falta de envolvimento com o utilizador **(12.4%)**;
- Falta de recursos **(10.6%)**;
- Expectativas irrealistas **(9.9%)**;
- Falta de suporte executivo **(9.3%)**;
- Alterações aos requisitos e especificações **(8.7%)**;
- Falta de planeamento **(8.1%)**;
- Sistema deixou de ser necessário **(7.5%)**.

Os problemas iam assim surgindo, e não eram solucionados, uma vez que o custo de os poder corrigir de uma forma mais adequada, era bastante elevada, como mostra a figura seguinte.

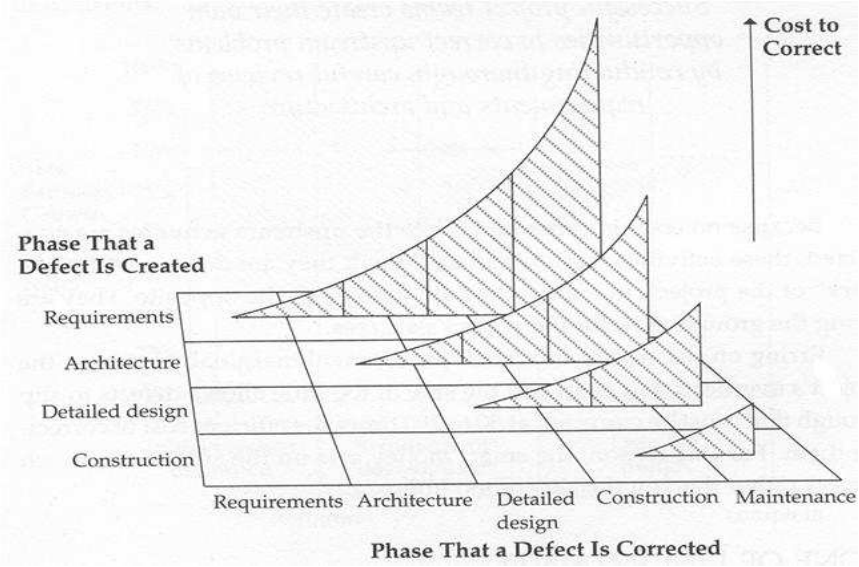

Figura 4 Aumento no custo de defeito como o tempo entre a criação do defeito e aumento da correlação do defeito [8]

Com isto começou a haver a noção de que há um problema a ser resolvido, ou seja, a insatisfação com sistema actual, nova oportunidade de negócio, etc. É aqui que entra o papel do engenheiro de requisitos sendo este considerado um agente de mudança.
O engenheiro de requisitos deve, tornar-se um perito no domínio da aplicação, identificar o problema e a oportunidade, sabendo responder às seguinte perguntas,

- Qual o problema a ser resolvido? (Identificar a Fronteira do problema);
- Onde está o problema? (Compreender o Domínio do problema e o Contexto);
- De quem é o problema? (Identificar os s*takeholders*);
- Porque necessita ser resolvido? (Identificar os Objectivos dos *stakeholders*);
- Como poderia um sistema de software ajudar? (Definir Cenários);
- Quando necessita ser resolvido? (Identificar Restrições ao desenvolvimento);
- O que pode impedir ao engenheiro de resolver o problema? (Identificar Riscos).

Existem processos, métodos e ferramentas usadas em engenharia de requisitos, dos quais se dá ênfase:

- **Notação**: Esquema de representação (ou linguagem) para exprimir conceitos. Exemplos: lógica de 1ª ordem, diagramas de fluxo de dados, UML, entre outros;
- **Técnica**: Descreve como realizar uma actividade técnica particular e, se necessário, como descrever um produto dessa actividade numa dada notação. Exemplos: Diagramas de casos de utilização;
- **Método**: Descrição técnica sobre como realizar uma colecção de actividades, focando-se na integração de técnicas e guias sobre o seu uso.
  Exemplos: OMT, RUP, KAOS, etc;
- **Modelo de processo**: Descrição abstracta de como conduzir uma colecção de actividades, focando-se na utilização de recursos e dependências entre actividades;
- **Processo**: Actuação de um modelo de processo.

Mas nem sempre é fácil decidir onde encaixar os métodos de **ER** nos processos de **ER**, pois muitas vezes não é claro onde cada método "encaixa" e qual é o método apropriado para alguns tipos de domínio de problema.

3.1 Quem se beneficia com a engenharia de requisitos?

Clientes, quem solicita o serviço de análise do problema, por terem um documento que assegure a resolução dos problemas referentes aos requisitos funcionais propostos e terem também a garantia da qualidade através de requisitos não funcionais expressados no mesmo documento.

Engenheiro de Requisitos, fornecem o serviço de análise de problemas, com o documento produzido através da engenharia de software, estes terão uma visão mais precisa e clara dos verdadeiros requisitos funcionais necessários, não perdendo tempo e sendo mais objectivo, e também pode focalizar a qualidade do produto de acordo com os requisitos não funcionais escolhidos.

Futuros clientes e engenheiros de requisitos, que podem beneficiar da reutilização da engenharia de requisitos através de documentos bem elaborados feitos anteriormente para resolução de problemas semelhantes, diminuindo assim o tempo de desenvolvimento e por consequência o custo para o cliente, contextualizando o engenheiro com o problema a ser resolvido e com as peculiaridades do mesmo em relação a conhecimentos específicos do domínio do problema na área proposta.

Concluindo, a engenharia de requisitos traz benefícios a ambas as partes, tanto com diminuição de custos para os clientes, como em uma melhor contextualização e um maior controlo sobre os requisitos para o engenheiro, e também propõe a reutilização dos documentos produzidos, ou seja, incita a reutilização do conhecimento adquirido através a iteração entre as partes ao longo do processo de engenharia de requisitos.

4. Ferramentas de Engenharia de Requisitos

Nenhum projecto de desenvolvimento de software é levado a cabo com sucesso sem uma equipa de desenvolvimento, onde cada membro sabe o seu papel e os seus objectivos individuais e onde todos cooperam para atingir o objectivo final: um software de qualidade, fácil de utilizar e que cumpre os requisitos para os quais foi projectado. É também importante a utilização de ferramentas de colaboração em equipa que permitam uma gestão eficaz do projecto.

De seguida apresentam-se alguns produtos estudados, bem como as suas aplicações práticas.

Argo UML é uma ferramenta *open source*, desenvolvida pela Tigris, líder de modelação de UML, e inclui suporte para todos os diagramas strandard UML 1.4. Corre em qualquer plataforma Java e está disponível em dez línguas. O ArgoUML utiliza a Framework de edição gráfica UCI para editar diagramas UML. **Unified Modeling Language** (UML) permite que os engenheiros de software visualizem os produtos do seu trabalho em diagramas padronizados. Junto com uma notação gráfica, o UML também especifica significados, isto é, semântica

ConceptDraw Office, concebida pela CS Odessa, é uma ferramenta abrangente destinada a ajudar os trabalhadores em empresas de qualquer tamanho para racionalizar o trabalho diário e para aumentar a vantagem competitiva com a abordagem inteligente à organização do fluxo de informação nos processos empresariais. O ConceptDraw permite a modelação de apenas alguns diagramas UML, uma vez que não é uma ferramenta de apenas modelação de software.

MagicDraw, cujo vendedor é No Magic, é uma ferramenta de modelação de sistemas. Desenhado para Analistas de Negócios, Analistas de Software, Programadores, e Elaboradores de Documentação, esta ferramenta de desenvolvimento dinâmico e versátil facilita a análise e desenho de sistemas orientados a objectos (OO) e bases de dados. Proporciona o melhor mecanismo de código da indústria da engenharia (com pleno suporte para Java, C + +, C #, CL (MSIL) e linguagens de programação CORBA IDL), bem como esquemas de modelação de dados, geração de DDL e engenharia revertida.

StarUML é um projecto *open source*, desenvolvido pelo From SourceForge, para o desenvolvimento rápido, flexível, extensível, *featureful* e plataforma UML / MDA de avaliação livre disponível correndo em plataforma Win32. O objectivo do projecto StarUML é construir uma ferramenta de modelação de software e também uma plataforma que obrigar a substituição de ferramentas comerciais UML, tais como Rational Rose e o Together. StarUML é em grande parte escrito em Delphi. No entanto, StarUML é um projecto multilingue específico e não vinculado para uma linguagem de programação específica, por isso qualquer linguagem de programação pode ser usada para desenvolver StarUML, por exemplo, C / C + +, Java, Visual Basic, Delphi, JScript, VBScript, C #, VB.NET, etc.

Smartdraw, produto desenvolvido pela, SmartDraw, permite, um tipo de desenho de software diferente. Em vez de começar com uma folha de desenho em branco, o SmartDraw dá a opção de começar exactamente com modelo que o utilizador precisa. Em seguida, possui comandos simples para adicionar a sua informação. Não há outro software de desenho parecido. O SmartDraw inclui SmartTemplates especializadas para mais de 50 tipos de gráficos, cada uma personalizada e adaptada para o SmartPanel ajudar.

Visio produto produzido pela Microsoft permite aos profissionais de TI e empresariais visualizar, explorar e transmitir informações complexas de forma fácil. Desde texto e tabelas complexas de difícil compreensão a diagramas, o Visio, transmite informações rapidamente. Ao contrário das imagens estáticas, os diagramas do Visio de dados ligados que são apresentados, são fáceis de actualizar e aumentam consideravelmente a produtividade.

Objecteering é uma ferramenta editada pela Objecteering estando disponível em duas versões: (i) Free Edition Objecteering 6, que é uma ferramenta de modelagem UML 2.0, fornecida gratuitamente, mas com algumas restrições. Esta versão tem um nível elevado de modelação, incluindo a geração de documentos, (ii) a edição Enterprise é a edição comercial de Objecteering 6, que fornece acesso à linha completa do produto, permitindo modelação e desenvolvimento de vários modelos, através do uso de um repositório.

De seguida apresenta-se uma tabela com os vários produtos e os modelos de software que cada uma disponibiliza.

Produtos / Diagramas	**ArgoUML**	**ConceptDraw**	**MagicDraw**	**StarUML**	**Visio**	**SmartDraw**	**Objecteering**
Class diagrams	✓	✓	✓	✓	✓	✓	✓
Use case diagrams	✓	✓	✓	✓	✓	✓	✓
Object diagrams						✓	✓
Communication diagrams			✓			✓	✓
Collaborarion diagrams	✓			✓	✓	✓	✓
Deployment diagrams	✓		✓	✓	✓		✓
Activity diagrams	✓		✓	✓	✓	✓	✓
Sequence diagrams	✓	✓	✓	✓	✓	✓	✓
Interaction overview diagrams							✓
State diagrams	✓		✓	✓	✓	✓	✓
BPMN process diagrams							✓
Profile diagrams							✓
Goal diagrams							✓
Requirement diagrams							✓
Dictionary diagrams							✓
Package diagrams						✓	
UML Web App			✓			✓	
Data Flow diagrams						✓	
Entity relationship diagrams						✓	
UML component				✓			
Composite Structure diagrams			✓	✓			
Package diagrams			✓				
Implementation diagrams			✓				
Robustness diagrams			✓				

Class diagrams

Diagramas de classe são os mais utilizados em diagramas UML: representam os conceitos de um sistema (ou apoiados por um sistema), juntamente com as suas propriedades e inter-relações. A partir de um nível elevado (conceptual), as classes representam os conceitos apoiados por um sistema, enquanto que a partir de um baixo nível (físicos) perspectivas que podem representar as classes implementadas por uma linguagem.

Use case diagrams

Diagramas de caso de uso são uma prática muito popular em UML. Contribui para formalizar um pedido que se destina a ser utilizado por utilizadores e que tipo de uso cada utilizador terá de fazer o pedido.

Object diagrams

Este modelo mostra o exemplo de uma família que instancia o modelo apresentado na secção de diagramas de classe. Esta característica é muito útil quando se representa a constituição de elementos montagem outros elementos.

Communication diagrams

Os diagramas de comunicação que expressam mais ou menos a mesma semântica como diagramas de sequência. Esquemas de comunicação têm uma capacidade limitada quando se trata de mostrar sequencialmente escolhas, ou para ser estruturados ou para referenciar-se. Eles são usados para ilustrar a arquitectura e os seus princípios fundamentais.

Deployment diagrams

UML2 artefactos são utilizados para definir a configuração dos elementos produzidos a partir de modelos. Aparecem como elementos ortogonais, e permitem a definição de arquivos fonte, bibliotecas, esquemas, com a designação do modelo de elementos que as compõem.

Activity diagrams

Diagramas de actividade são utilizados para modelar processos de negócio ou parte da dinâmica de um modelo. Podem representar um processo, ou representar o comportamento de uma operação. Partições podem ser representadas na horizontal ou na vertical.

Sequence diagrams

Descreve a maneira como os grupos de objectos colaboram num dado comportamento ao longo do tempo. Registam o comportamento de um único caso de uso e exibe os objectos e as mensagens transferidas entre esses objectos no caso de uso.

Interaction overview diagrams

Diagramas de interacção em UML são um tipo de actividade em que nós representam interacções nos diagramas. São um mecanismo de alto nível estruturação dos diagramas de sequência. Ilustram uma visão panorâmica dos fluxos de uma aplicação.

State diagrams

Diagramas de estados são usados para descrever o comportamento de um sistema. Descrevem todos os possíveis estados de um objecto e como ocorrem nos eventos. Cada diagrama representa normalmente objectos de uma única classe e mostra os diferentes estados dos objectos através do sistema.

BPMN process diagrams

Business Process Modeling Notation é uma notação padronizada pela OMG. O principal objectivo do BPMN é fornecer uma notação que é facilmente compreensível por todos os utilizadores empresariais, analistas de negócios que criam os primeiros processos, e finalmente, para os empresários que vão gerir e acompanhar os processos. Assim estes diagramas criam uma ponte padronizada para o fosso entre o processo de negócio e do processo de concepção de execução.

Profile diagrams

Diagramas de perfil são diagramas de classe simplificados, que representam os conceitos de perfil e de classe.

Goal diagrams

Diagramas de objectivos representam objectivos, as suas dependências e as suas propriedades. Para cada objectivo, é importante expressar como e quando este será verificado, os critérios concretos para o seu sucesso, e para quem o objectivo é atribuído.

Requirement diagrams

Um requisito específico apresenta uma capacidade ou condição que deve ser preenchida. Um requisito pode definir uma função que um sistema deve desempenhar ou uma condição de desempenho que um sistema deve atingir. Um requisito pode aparecer em outros diagramas para mostrar suas relações com outros modelos de elementos. Os requisitos têm propriedades e ligações a outros elementos.

Dictionary diagrams

Um dicionário representa a terminologia usada para um domínio específico. É composto de termos que representam nomes e definições. Um dicionário é a base para identificar qual o modelo, que elementos devem ser criados, a fim de representar noções.

Collaboration diagrams

Os diagramas de colaboração permitem mostrar uma organização espacial dos componentes e interacções, em vez de se concentrarem na sequência das interacções. Um diagrama de colaboração mostra uma interacção organizada em torno dos objectos na interacção e seus vínculos entre si.

Um diagrama de colaboração é um cruzamento entre um diagrama de símbolos e um diagrama de sequência, no qual uma situação específica é descrita por setas numeradas que mostram o movimento das mensagens durante o desenvolvimento de uma situação.

O diagrama de colaboração pode ser usado para:

- Descrever uma situação específica, representando o movimento de mensagens entre os objectos;
- Mostrar uma organização espacial de objectos e suas interacções, em vez da sequência das interacções.

Composite structure diagram

Um *Composite structure diagram* no UML é um tipo de diagrama de estrutura estática, que mostra a estrutura interna de uma classe e as colaborações que esta estrutura torna possível.

Este diagrama pode incluir partes internas, portas através das quais as partes interagem umas com as outras ou através das quais instâncias da classe interagem com as partes e com o mundo exterior. Uma estrutura composta é um conjunto de elementos interligados que colaboram em tempo de execução para conseguir algum efeito. Cada elemento tem um papel definido na colaboração.

Packages diagram

Um diagrama de pacotes mostra pacotes e relações entre pacotes. Na realidade, não existem propriamente diagramas de pacotes em UML; em vez disso, pacotes e relações entre pacotes aparecem noutros diagramas, de acordo com o tipo de pacote:

- Pacotes de classes (pacotes lógicos) - em diagramas de classes;
- Pacotes de componentes – em diagramas de componentes;
- Pacotes de nós – em diagramas de distribuição;
- Pacotes de casos de utilização – em diagramas de casos de utilização.

Implementation diagrams

Estes diagramas mostram aspectos da implementação, incluindo estrutura de código fonte e estrutura de implementação em tempo de execução. A arquitectura física é uma decomposição detalhada do hardware e software que cercam a implementação do sistema.

Web application diagram

Um diagrama Web application é um ficheiro que ajuda o utilizador a visualizar e alterar o fluxo de uma aplicação Web, tais como uma interfaces ou aplicações. O editor de um diagrama Web é um editor gráfico para visualizar diagramas Web.

Robustness diagrams

Diagramas robustness destinam-se a fazer a ponte entre os casos de uso e os diagramas de classes e garantir que o utilizador abrangeu todas as bases.

Data Flow diagrams

Diagramas de fluxos de dados são uma abordagem para visualização de processamento de dados. É forte uma forte ilustração das relações de processos, armazenamento de dados e entidades externas de um dado sistema.

Entity Relationships diagrams

Modelação Entity Relationships é um método de modelação de dados, usado para produzir um tipo de diagrama conceitual ou um modelo de dados semântico de um sistema, muitas vezes uma base de dados relacional.

UML Component

Diagrama de componentes é um tipo especial de diagrama de classes que dá ênfase aos componentes de um sistema.

Qual a melhor ferramenta para a modelação de software necessária na Engenharia de Requisitos?

O MagicDraw destaca-se das outras ferramentas de desenho de software, pelo que apresenta mais modelos em relação às outras, encontrado dez motivos que mais o destacam das outras ferramentas, sendo elas:

1. Promove uma rápida aprendizagem com interface intuitiva
2. Cria diagramas mais rapidamente do que qualquer ferramenta no mercado
3. Deriva modelos de código fonte existente em apenas alguns segundos
4. Permite ao utilizador visualizar o seu modelo em alguns passos rápidos
5. Permite vários utilizadores trabalhar no mesmo modelo em paralelo
6. Distribui o código fonte a partir do modelo UML instantaneamente
7. Elimina a preparação de documentos com geração automática de relatórios
8. Estende capacidades UML além do UML 2 rapidamente
9. Acelera o “tempo de viagem” entre a modelação de domínios
10. Permite a navegação rápida nos diferentes modelos

5. Técnicas da Engenharia de Requisitos

Existem várias técnicas em **ER** e métodos para a identificação e formulação de requisitos, para explicar como estas técnicas podem ser aplicadas durante o processo da engenharia de requisitos.
Essas técnicas são [1],

- **Métodos para a Engenharia de Requisitos**: o processo para identificação, estruturamento e formulação de requisitos de software é normalmente guiado por um método de requisitos. Os métodos de requisitos são formas sistemáticas de produção de modelos de sistema. Os modelos de sistema são baseados em conceitos computacionais, tais como, objectos ou funções em vez de conceitos de aplicação do domínio. São também importantes pontes entre a análise e o desenho do processo. Quatro modelos de sistema usados frequentemente são: Modelação Data-flow, Modelos de dados semânticos, Abordagens Orientadas a Objectos e Métodos formais;
- **Métodos de Requisitos Orientados a *Viewpoints***: *Viewpoints* são um mecanismo explícito que têm em conta o sistema e a perspectiva do problema dos diferentes *stakeholders*. *Viewpoints* implícitos foram introduzidos inicialmente no método SADT e foram feitos explícitos no método CORE. Duas das abordagens orientadas a *viewpoints* são a VOSE, onde um *viewpoint* é um modelo para descrever o modelo do sistema numa representação em particular, e o VORD, onde um *viewpoint* representa um receptor de serviços do sistema.
- **Requisitos não funcionais**: são requisitos que não são especificamente envolvidos com a funcionalidade de um sistema. Colocam restrições no produto a ser desenvolvido e no processo de desenvolvimento, e especificam restrições externas que o produto deve satisfazer. Os requisitos não funcionais incluem a *security*, *safety*, *usability*, *reliability* e *performance*.
- **Especificação de sistemas interactivos**: são sistemas que envolvem a interacção do utilizador formulando os requisitos para esses sistemas colocarem problemas. Devido à sua natureza interactiva, deve-se ter em consideração para formular os seus requisitos externos.

5.1 Métodos de Requisitos Orientados a *Viewpoints*

5.1.1 Noção de Viewpoints

A noção de *viewpoints* [1] para a engenharia de requisitos destinam-se a formalizar que o que pensamos é uma parte natural de qualquer processo de análise.

Um bom engenheiro de requisitos irá recolher informações sobre o que está a ser estudado a partir de um certo número de diferentes fontes e vai reconhecer que estas fontes podem ter diferentes origens, muitas vezes igualmente válidas. Reconhecendo estas perspectivas e conciliar as diferenças entre elas é essencial se a análise é válida. Abordagens orientadas a *viewpoints* são simples de modo a formalizar esta análise intuitiva.

Uma abordagem orientada a *viewpoints* para a engenharia de requisitos reconhece que toda a informação sobre o sistema de requisitos não pode ser descoberta por considerar o sistema a partir de uma única perspectiva. Pelo contrário, é necessário recolher e organizar requisitos de um número de diferentes *viewpoints.* Um *viewpoint* é assim um encapsulamento de informações parciais sobre um sistema de requisitos. Informação a partir de diferentes *viewpoints* deve ser integrada para formar a especificação final do sistema.

5.1.2 Vantagens

As principais vantagens de uma abordagem baseada em engenharia de requisitos com *viewpoints* são [1]:

- A utilização de sistemas é heterogénea - não existe tal coisa como um utilizador típico. V*iewpoints* podem organizar os requisitos do sistema a partir de diferentes classes do sistema do utilizador final e outro sistema dos *stakeholders*;
- Diferentes tipos de informação são necessários para especificar sistemas, incluindo informações sobre o domínio da aplicação, sobre o ambiente do sistema e engenharia de informação sobre o desenvolvimento do sistema.

V*iewpoints* podem ser utilizados para recolher e classificar esta informação;

- V*iewpoints* podem ser utilizados como um meio de estruturar o processo de identificação de requisitos;
- V*iewpoints* podem ser utilizados para encapsular diferentes modelos do sistema de cada um dos quais se prevê especificação algumas informações;
- V*iewpoints* podem ser utilizados para estruturar a descrição dos requisitos e expor conflitos entre diferentes requisitos.

5.1.3 Métodos

Nos métodos baseados em *viewpoints* que foram desenvolvidos, dois tipos diferentes de *viewpoints* foram propostos [1]:

- *Viewpoints associados com stakeholders do sistema*: Informalmente, num sistema, o *stakeholder* é alguém que esteja, directa ou indirectamente, afectado pela existência de um sistema. Assim, os *stakeholders*, podem ser utilizadores finais de um sistema, gestores de organizações onde os sistemas estão instalados, outros seres humanos e sistemas baseados em computador em uma organização, entidades externas que têm algum tipo de interesse no sistema (por exemplo, órgãos reguladores, os clientes de uma organização que tem o sistema instalado) e engenheiros envolvidos no design, desenvolvimento e manutenção do sistema.

- *Viewpoints associados com o conhecimento organizacional e de domínio:* Conhecimento organizacional e de domínio é o conhecimento que restringe os requisitos do sistema. As restrições podem ser físicas (por exemplo, o desempenho da rede), organizacionais (por exemplo, incompatibilidade de hardware utilizado em diferentes divisões de uma empresa), humano (por exemplo, taxa de erro média operador), ou podem reflectir locais, leis nacionais ou internacionais, regulamentos e normas. Este tipo de *viewpoint* não pode estar associado com uma única classe de *stakeholders*, mas inclui informação recolhida de muitas fontes diferentes (pessoas, documentos, outros sistemas, etc.).

5.1.4 Exemplo de aplicação prática de viewpoints

Para ilustrar a quantidade de *viewpoints* que podem ser úteis para descobrir, analisar e documentar os requisitos para um sistema baseado em computador, considera-se um exemplo, relativamente simples, sistema baseado na Internet para fornecer instalações bancárias on-line para os clientes através de uma interface WWW.

Os *viewpoints* que podem ter de ser considerados são:

1. Um ou mais *viewpoints* clientes, depende dos diferentes tipos de clientes, tais como empresas e clientes pessoais (*stakeholder*);
2. Um ou mais *viewpoints* agentes bancários abrangendo a operação e gestão do sistema (*stakeholder*);
3. Um *viewpoint* de segurança (*stakeholder*);
4. Um *viewpoint* de comercialização (*stakeholder*);
5. Um *viewpoint* de base de dados (organizacional);
6. Um *viewpoint* pessoal (organizacional);
7. Um *viewpoint* regulador (domínio - os bancos têm reguladores externos);
8. Um *viewpoint* engenheiro/ rede (domínio);

Na prática, se muitos *viewpoints* são identificados, é difícil gerir a grande quantidade de informação gerada e verificar quais os requisitos prioritários. Portanto, uma etapa essencial da maioria dos métodos baseados em *viewpoints* consiste em seleccionar apenas os *viewpoints* mais críticos a serem utilizados na análise. Os engenheiros de requisitos devem equilibrar as vantagens da ampla cobertura adicional oferecida pelos *viewpoints* com os custos da análise e os problemas da gestão da informação.

Os viewpoints, ilustrados no exemplo anterior, para engenharia de requisitos de sistemas baseados em computador, são muito usados nos dias de hoje, mas o conceito também tem sido encontrado para ser útil em outras áreas.

Alguns exemplos dessas áreas são:

- Em sistemas de ensino inteligentes e distribuídos, as noções de *viewpoints* cognitivos para encapsular crenças foram encontrados para ser úteis (Moyse, 1992; Self, 1992);
- Em comunicações, o modelo de referência ODP define cinco *viewpoints* (empresas, informação, informática, engenharia e tecnologia) a partir do qual um sistema pode ser especificado (Linington, 1995; Bowman et al., 1996);
- Em CSCW, *viewpoints* têm sido utilizados para estabelecer a estrutura de análises organizacionais (Hughes et al., 1995);
- E em engenharia concorrente, *viewpoints* têm sido propostos como uma abordagem sistemática à análise de conflitos (Klein, 1992).

6. Ética na Engenharia de Requisitos

Com o uso da engenharia de requisitos existem vários pontos que podem por em causa a ética da mesma, pelo que existe um Código de ética ACM/IEEE [10], que consiste,

- Público: actuar consistentemente de acordo com o interesse público;
- Cliente e Empregador: actuar de forma a servir os interesses do seu cliente e empregador, consistentemente de acordo com o interesse público;
- Produto: assegurar-se o mais possível de que os produtos e modificações relacionadas vão de encontro às normas profissionais;
- Juízo Profissional: manter integridade e independência no seu juízo profissional;
- Gestão: subscrever e promover uma abordagem ética à gestão do desenvolvimento e manutenção do software;
- Profissão: avançar a integridade e reputação da profissão de acordo com o interesse público;
- Colegas: ser justo e apoiante dos colegas;
- Auto-Ética: participar numa aprendizagem ao longo de toda a vida e promover uma abordagem ética à prática da profissão.

De particular relevância para a Engenharia de Requisitos tem-se,

- **Competência**: nunca exibir erradamente o seu nível de competência;
- **Confidencialidade**: respeitar a confidencialidade de todos os *stakeholders;*
- **Direitos de propriedade intelectual**: respeitar os autores de ideias e designs;
- **Protecção de dados**: ter conhecimento de todas as leis sobre gestão de dados pessoais.

7. Conclusão

A engenharia de requisitos depende muito da interacção entre os clientes e os engenheiros de requisitos, de modo que seja minimizado qualquer problema na definição de requisitos por parte do cliente. Por mais que não se deseje, os requisitos estão sempre em mudança durante o desenvolvimento de um sistema, e quão melhor for o processo de engenharia de requisitos desenvolvido pela empresa, menores serão os problemas encontrados em função de toda a dificuldade que envolve esta importante parte da análise.

Também é notável que o cenário actual na engenharia de requisitos está muito longe do ideal tanto para os clientes como para os engenheiros de requisitos, isto deve-se à falta de padronização principalmente na engenharia de requisitos não funcionais, e à falta de uma real consciencialização da importância da engenharia de requisitos no contexto da engenharia de software.

Há também uma carência conceitual em relação aos profissionais da área de engenharia de software relacionada com a falta ou o pouco entendimento do domínio do problema a ser analisado e documentado, sendo este causado pela transdisciplinaridade da aplicação de engenharia de software na resolução de problemas.

8. Bibliografia

[1] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*: John Wiley, 1998.

[2] S. L. Pfleeger, *Software Engineering - Theory and Practice*, Second Edition ed: Prentice Hall, 2001.

[3] I. Sommerville, *Software Engineering (7th Edition)*: Pearson Addison Wesley, 2004.

[4] Sommerville I, Sawyer P. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, 1997.

[5] Sutcliffe A., *User-Centered Requirements Engineering: Theory and Practice*, first ed: Springer, 2002.

[6] Boehm, B., *A Spiral Model of Software Development and Enhancement*, http://www.cs.usu.edu/~supratik/CS%205370/r5061.pdf.

[7] Patton, R. "Software Testing", SAMS, 2001.

[8] McConnell, S., *Software Project Survival Guide*, 1998, ISBN: 1-57231-621-7.

[9] Database Ansewers, 2002, http://www.databaseanswers.com/modelling_tools.htm.

[10] ACM/IEEE: http://www.acm.org/about/se-code