



**Uma Nova Proposta para Desenvolvimento de Sistemas**

## **Filosofia Telescope**

**PRD Tecnologia de Gestão Ltda.  
Março/2007**



## Índice

1	Introdução.....	2
2	Origem do Telescope.....	2
3	Telescope e soluções de mercado.....	2
4	Técnicas em uso no Telescope.....	3
5	Sistemas, programas e linguagens de programação.....	3
6	Alguns cuidados ao desenvolver sistemas.....	4
6.1	Valorizar “o que é” e não “o como se faz”.....	5
6.2	Modelagem conceitual, lógica e física.....	5
6.3	Reaproveitamento.....	6
6.4	Implementação e confiabilidade.....	6
6.5	Tecnologia e continuidade.....	7
6.6	Atualização e manutenção.....	7
6.7	Segurança.....	7
6.8	Implantação e migração.....	8
7	Desenvolvendo com o Telescope.....	8
7.1	Valorizar “o que é” e não “o como se faz”.....	9
7.2	Modelagem conceitual, lógica e física.....	9
7.3	Reaproveitamento.....	10
7.4	Implementação e confiabilidade.....	10
7.5	Tecnologia e continuidade.....	10
7.6	Atualização e manutenção.....	11
7.7	Segurança.....	11
7.8	Implantação e migração.....	11
8	Conclusão.....	11



## 1 Introdução

Este documento descreve o Telescope (filosofia e projeto), sua origem, conceitos e justificativas para construir a nossa solução personalizada.

Também discutimos diversos requisitos para projetos de desenvolvimento de sistemas dos quais muitos profissionais da área de tecnologia já devem ter vivenciado, utilizado e/ou estudado. Ao apresentar estes requisitos, sintetizamos o seu significado para o software em construção e problemas decorrentes de seu uso.

Por fim, apresentamos a nossa forma de desenvolver sistemas e como conseguimos produtividade, resolvendo os problemas descritos anteriormente.

## 2 Origem do Telescope

Trabalhamos em desenvolvimento de sistemas de informação e processo de software desde os anos 90. Durante todos estes anos utilizamos muitas tecnologias diferentes (p.ex., Cobol, DMSII, ZIM, Clipper, Oracle, VB, HTML, Java...), passamos por variados ambientes e organizações (p.ex., universidades, indústrias, distribuidoras, comércio, franqueadoras...) e participamos de diversos projetos com equipes heterogêneas, ora contando com muitos recursos (financeiros, técnicos e humanos), ora contando com poucos destes recursos.

Desta larga experiência observamos que tanto a tecnologia – em especial – quanto o próprio negócio mudam constantemente. Mudança, justamente o principal fator desafiador que deu origem ao projeto que chamamos de Telescope.

O Telescope nasceu inspirado na necessidade de maximizar o nosso processo de software, disponibilizando métodos, técnicas e ferramentas capazes de colaborar em todas as fases do desenvolvimento de sistemas. Para isto potencializamos a capacidade de fazer mudanças rápidas para acompanhar os novos rumos estratégicos e de negócios das organizações buscando produtividade, qualidade e independência tecnológica.

Ao falar de processo de software, estamos falando no processo que abrange o mapeamento, desenvolvimento, testes, implementação e - muito importante - na garantia de todo o ciclo decorrente disto, ou seja, na implantação de novas versões e manutenibilidade dos sistemas construídos.

O Telescope não é um software, mas um conjunto completo de componentes que nos permite desenvolver sistemas de forma padronizada, rápida e com qualidade. O Telescope compreende:

- Uma ferramenta I-CASE proprietária com um repositório e interfaces especialistas que permitem o detalhamento conceitual e físico de sistemas de informação;
- Um conjunto inteligente de geradores que transformam as informações armazenadas no repositório em scripts e códigos compiláveis em diversas linguagens de programação e tecnologias;
- Outras ferramentas de apoio integradas ao repositório para controlar projetos, recursos e completar o ambiente do Telescope;
- Um conjunto de documentos que estabelecem os procedimentos a serem seguidos nas etapas de desenvolvimento dos softwares;
- Pessoas competentes e treinadas;

O Telescope é o nosso diferencial competitivo, pois registra e mantém a nossa base de conhecimento, armazenando componentes que podem ser somados a outros, formando sistemas, entidades, regras de negócio, interfaces, eventos, rotinas, menus, regras de acesso, etc.

## 3 Telescope e soluções de mercado

Existem poucos softwares no mercado que atendem as características que consideramos básicas para documentação, modelagem, geração e manutenção de sistemas. No entanto, todos aqueles dos quais tivemos a oportunidade de usar e/ou avaliar são tendenciosos quanto a tecnologia utilizada (p.ex.: Rational Rose, Oracle Designer, OptimalJ...).



O Oracle Designer foi a solução de mercado que mais exploramos e utilizamos. Na época escolhemos esta solução porque adotava uma metodologia formal que valoriza muitos dos conceitos desejados, mas é um software que perdeu seu valor como produto para a Oracle, tanto que foi descontinuado, ou melhor, não recebe mais investimentos.

Embora existam soluções de mercado que atendem à vários dos requisitos para um processo de software bem estruturado, a maioria deles são ferramentas hospedeiras de uma tecnologia e são soluções que vão de encontro aos interesses da própria empresa fornecedora (o que é compreensível, mas não significa que seja o que se quer).

Exemplos destas características tendenciosas:

- Criam aplicações com forte vínculo e dependência tecnológica;
- Implementam conceitos e nomenclaturas próprias;
- Resolvem as associações entre os modelos conceituais e físicos com soluções próprias e “engessadas”;
- Geram códigos personalizados difíceis de customizar. Os geradores mais atuais se utilizam de “frameworks” e “patterns” que colaboram muito com a customização, porém exigem muito conhecimento especializado e grande domínio das técnicas relacionadas à solução.

Com tantos pontos negativos, decidimos construir o nosso próprio ambiente. Para isto utilizamos por completo a experiência adquirida, agregamos conceitos e técnicas sedimentadas no mercado, pesquisamos soluções simples e eficientes e adicionamos soluções mais genéricas porque trabalhamos para isolar a tecnologia, ou seja, queremos soluções simples e com total independentes da tecnologia.

## 4 Técnicas em uso no Telescope

Softwares são intangíveis e representam necessidades reais de pessoas. Para construí-los, precisamos entender as pessoas que têm as necessidades. A compreensão dos requisitos é crítico para o sucesso de um projeto de software, o que é também válido para outras disciplinas, por exemplo, o grau de entendimento de um cirurgião com o seu paciente, o entendimento de um arquiteto, engenheiro, fornecedor...

Mas como validar, medir, quantificar os requisitos? E, como se não bastasse, os requisitos mudam, os governos mudam, os fornecedores mudam, os clientes mudam, as pessoas mudam.

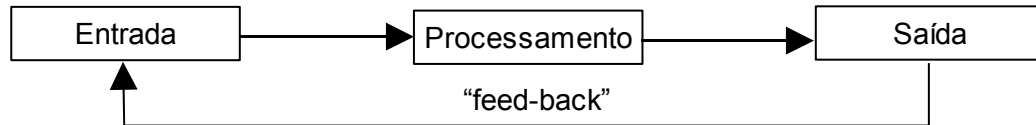
Conscientes da importância de um bom entendimento das reais necessidades das pessoas, precisamos garantir que nossos modelos fossem capazes de serem transformados em conhecimento e ainda permitir resultados rápidos. Para tanto, pesquisamos métodos e técnicas que somamos ao nosso processo de software.

Abaixo algumas das técnicas que adotamos para enriquecer a nossa engenharia de software:

- Maior foco em funcionalidades e usabilidades;
- Simplificar, agilizando o processo (paradigmas, pessoas, métodos...);
- Rapidamente agregar valor (soluções rápidas e objetivas) para alavancar o negócio;
- Reduzir a iteratividade (encolher o tempo do ciclo: requisito, análise, desenho, codificação e testes), pois:
  - Soluções rápidas aumentam a interação e comprometimento de todos (usuários e equipe);
  - Ao utilizar a solução mais cedo, todos entendem melhor a necessidade, abrindo novas perspectivas que enriquecem o software e o projeto;
  - Todos entendem melhor e colaboram mais;
  - A confiança de todos aumenta - assim como a realização - porque o resultado do esforço e o retorno do investimento fica visível.
- Aplicar fundamentos conceituas (produzir código rapidamente, isolar a tecnologia e organizar para rastrear e gerenciar o projeto);
- Ter meios de estabelecer regras e controlar a produtividade (I-CASE implementando a metodologia);
- Trabalhar com pessoas preparadas e qualificadas.

## 5 Sistemas, programas e linguagens de programação

A teoria de sistemas define que um sistema é, teoricamente:



Assim, objetivamente, podemos dizer que um sistema de informação é um conjunto de programas escritos em uma linguagem de programação. Estes arquivos de programas sofrem um processo (compilação) que gera arquivos binários capazes de serem “executados” ou “interpretados” em muitos tipos de equipamentos (p.ex., computadores).

Isoladamente, os programas de computador são - a princípio – muito simples, porque as linguagens de programação nas quais são escritos, podem ser resumidas, sem exceção, em 5 tipos de instruções:

- Definição (define-se variáveis, tipos, tamanhos, rotinas...);
- Atribuição (atribui-se valores, conteúdos, dados aos elementos definidos);
- Condição (testam-se condições que definem o fluxo do programa, p.ex., “se-então”);
- Repetição (loops e execuções repetitivas, p.ex., “repetir até/enquanto”);
- Exceção (tratamento de erros, desvios, interrupções);

Além de serem “simples”, os programas de computador podem ser copiados e alterados para atender funcionalidades e usabilidades semelhantes, por exemplo:

1. Temos um programa para digitação de pedidos de venda que registra dados cadastrais de cliente, datas, valores, itens (produtos), quantidades, etc...
2. Agora temos uma nova demanda que é a necessidade de construir um programa de digitações de pedidos de compra;

Muito bem, o primeiro trata dos dados cadastrais do pedido de venda (cabeçalho) e os itens (produtos de estoque). Já a nova demanda tem como requisito cadastrar pedido de compra que também deve registrar dados cadastrais (cabeçalho) e itens (sem coincidência, produtos de estoque).

Se são similares e terão comportamentos parecidos podemos:

1. Fazer uma cópia do primeiro programa;
2. Dar um novo nome ao novo programa;
3. Modificar e adaptar o novo programa;
4. Salvar e compilar.

Pronto, já se tem mais um programa no sistema de informação que passa a atender mais um requisito do negócio. Agora o sistema registra e mantém pedidos de venda, assim como pedidos de compra.

Por este ângulo, tudo é muito simples...

Mas então, porque vemos tantas soluções, softwares, consultores, metodologias, técnicas, padrões, etc, tudo voltado para melhorar e facilitar a vida do desenvolvedor de sistemas?



Isto ocorre porque desenvolver sistemas é um tanto mais complexo do que descrito acima. Tal complexidade se dá pelo somatório dos elementos envolvidos durante e após o processo da sua construção. Somado a isto, as linguagens de programação - com suas 5 instruções básicas - mudam muito. Linguagens como Cobol ou Clipper que foram largamente exploradas e utilizadas em tantos casos de sucesso são substituídas e perdem seu valor comercial por deficiências tecnológicas e desinteresse comercial.

Por “elementos envolvidos” no processo, entendemos:

- Pessoas (diretores, clientes, fornecedores, funcionários...);
- Governo (leis, normas, convênios...);
- Regras de negócios (novas necessidades);
- Recursos;
- Equipamentos;
- Tecnologias;
- Legado;
- Evolução, mudanças estratégicas, novos mercados;
- etc

São muitas as variáveis relacionadas aos sistemas que desenvolvemos, e, por incrível que pareça, um dos maiores vilões ainda é a tecnologia que muda muito rapidamente. Mesmo para uma organização estável, ajustada e muito bem organizada, a troca de tecnologia é inevitável. Todas precisam acompanhar as novidades, fazer integrações, utilizar novos meios de comunicação, fazer upgrades de hardwares e softwares, substituir versões de banco de dados e sistemas operacionais, utilizar novas linguagens, etc.

## 6 Alguns cuidados ao desenvolver sistemas

As técnicas de desenvolvimento de sistemas (engenharia de software) objetivam facilitar a vida do desenvolvedor auxiliando este profissional durante todo o projeto de construção dos sistemas. O desejo final é obter programas mais confiáveis, reaproveitáveis, íntegros e que sejam fáceis de serem modificados. O problema é que a quantidade de variáveis que afetam os projetos de software aumentam as complexidades dos sistemas.

Abaixo listamos diversos fatores pertinentes a projetos de desenvolvimento de sistemas. São fatores diretamente relacionados à atividades e/ou requisitos (necessidades) de projeto. Indiferente de suas características, são críticos e mais cedo, ou mais tarde - vão aparecer para as equipes de desenvolvimento, seja em discussões e reuniões técnicas de planejamento, seja em reuniões para resolução de problemas ou mesmo durante o desenvolvimento e programação dos sistemas.

### 6.1 Valorizar “o que é” e não “o como se faz”

O desenvolvedor deve ser capaz de abstrair a necessidade do cliente entendendo o problema enquanto que o cliente - por sua vez - deve realmente participar do processo de modelagem e não ser apenas um convidado.

Aqui destacamos 2 desafios:

1. Fazer com que o esforço de abstração resulte em documentação qualificada e suficiente;
2. Dar ao usuário um papel de destaque no processo porque é ele quem detém o conhecimento que está sendo documentado.

Problemas:

- São 2 profissionais envolvidos (o custo é alto) portanto deve ser produtivo;
- A preocupação com a tecnologia aparece muito cedo;



- A experiência do profissional de informática deve servir para agregar ao conceito e não para sobrepor o conhecimento do cliente;
- O resultado da modelagem dos conceitos muitas vezes não é encarado como um patrimônio da organização e fica perdido em arquivos, planilhas e anotações personalizadas e individualizadas;

## 6.2 Modelagem conceitual, lógica e física

Os modelos - que podem ser muitos e diversificados - servem para facilitar o entendimento e para representar relações, restrições, fluxos, contexto, etc. Ao utilizar modelos, o profissional busca entender melhor o problema e formalizar a solução proposta permitindo explicações sobre o desenvolvimento, tanto para usuários, equipe, diretores, etc, quanto para ele mesmo.

Com modelos, indiferente da tecnologia e nomenclaturas adotadas, criam-se ótimas ferramentas que colaboram para representar o que o usuário quer, principalmente quando facilitam a sua participação.

As metodologias de desenvolvimento de sistemas sempre pregam que o profissional deve preocupar-se com “o quê é” antes de pensar no “como fazer”. Ou seja, defendem o uso de modelos que permitam abstrair conceitos para posteriormente fazer uso dos elementos mais ligados a tecnologia (p.ex, utilizar modelos conceituais e modelos físicos, nesta ordem respectivamente).

Adotando estas dicas, o profissional consegue se utilizar de outra importante técnica presente na engenharia de software, o refinamento sucessivo. Isto porque os conceitos mapeados em modelos conceituais vão sofrendo alterações e atualizações gradativas, forçando (naturalmente) um refinamento sucessivo e evolutivo. Melhora-se aos poucos.

Problemas:

- Existem perdas ao transformar (traduzir) conceitos de um modelo para outro, pois esta tradução dá origem a elementos que são adicionados ao novo modelo, mesmo que a tradução seja – de certa forma – simples, p.ex, um símbolo conceitual pode corresponder à um conjunto de códigos específicos de uma linguagem de programação. A maior dificuldade ocorre quando deve-se “retroceder” (voltar atrás) e desfazer o que já foi “traduzido” porque a transformação de um modelo conceitual para o físico deve ser feita para uma determinada tecnologia que conta com seus componentes brilhantes e “milagrosos”. Retornar, a partir destes códigos, não é algo tão simples ou direto.
- Existem os profissionais que defendem maior uso possível de modelos conceituais e aqueles que preferem se utilizar das facilidades tecnológicas existentes. Os interesses e preocupações de ambos profissionais são diferentes podendo vir a ser até mesmo conflitantes, o que resulta em prejuízo para o projeto e organização. Se um erro for identificado em um software, qual destes profissionais vai assumir a culpa, aquele que representou o conceito no seu modelo cheio de semântica com figuras muito parecidas cujo pequeno detalhe indicam significados completamente diferentes. Ou aquele profissional que utilizou a nova versão dos componentes cuja documentação garante total compatibilidade com outras soluções dos mais diversos fornecedores.
- Mesmo se apropriando das técnicas de uma metodologia, o engenheiro de software pode ter dificuldades em modelar o sistema porque o usuário não entende o que significam os símbolos semânticos da técnica escolhida. Aliás, o usuário não tem o dever de conhecer.
- As metodologias procuram fornecer meios para representar aspectos estáticos e dinâmicos dos sistemas durante a modelagem. O problema é que são muitos componentes e símbolos que as técnicas oferecem para serem utilizados, levando o profissional e consumir mais tempo preocupado com o uso correto deles do que com a necessidade do usuário propriamente dita.

## 6.3 Reaproveitamento

Quem desenvolve software precisa obter o máximo reaproveitamento do seu trabalho porque o investimento em software é alto e deve ser o mais duradouro possível.



Reaproveitamento pode ser dado com rotinas genéricas e bibliotecas de programas que são reutilizados constantemente em eventos diferentes. Por exemplo, rotinas de cálculos de metas, rotinas de validação de dados, utilitários de conversões de dados, etc.

Problemas:

- Construir rotinas genéricas exige profissionais mais qualificados;
- O benefício de utilizar genéricas desenvolvidas pode ser de médio prazo porque exige mais testes e maior tempo de planejamento e programação;
- Mudanças de tecnologia podem antecipar a “aposentadoria” de rotinas e bibliotecas que consumiram muitas horas de trabalho. Uma nova tecnologia pode descaracterizar programas inteiros desenvolvidos ou até mesmo invalidar comandos utilizados;

## 6.4 Implementação e confiabilidade

Mesmo adotando técnicas apropriadas para a realidade da organização e contando com excelentes profissionais, não se tem garantia de que o produto corresponda totalmente aos modelos mapeados e documentados.

Organizações procuram metodologias com ferramentas que permitam documentar modelos e com poder de transformá-los até chegarem ao produto final. Pode acontecer de que o processo de transformação não inclua todas as características importantes mapeadas para o negócio. Sendo assim, a solução mais simples é adicioná-las manualmente ao código gerado. Fazendo isto, a confiabilidade cai porque o modelo pode não estar mais representando tudo o que foi incluído manualmente (e vice-versa).

Muitas técnicas de desenvolvimento acelerado (ágil) estão ganhando espaço no mercado. Adotar estas metodologias sem um bom aporte técnico pode conduzir a sérios problemas de confiabilidade quanto ao resultado final (implementação). Métodos ágeis também exigem muito conhecimento a respeito das tecnologia relacionadas. Além disto, se são “ágeis”, podem estar deixando de fazer algo que as técnicas tradicionais propunham, por exemplo, construir modelos conceituais, documentar, etc.

Problemas:

- O processo acelerado encanta o usuário, pelo menos de forma pontual. No entanto, em projetos longos vê-se com frequência que uma solução pontual “apaga um incêndio” mas, pode vir a ser a origem de estragos que não ocorreriam caso houvesse um planejamento mais detalhado. Por exemplo, criar uma nova coluna “desnormalizada” no banco de dados, utilizar um campo de descrição para armazenar um código, implantar uma planilha eletrônica que será substituída por um sistema (“Quando? Não se preocupe, em breve...”), etc;
- O retrabalho aumenta. Este problema reflete diretamente sobre custos. Além do especto financeiro, todo retrabalho causa um impacto muito forte na produtividade porque ninguém gosta de fazer novamente “o que já fez antes”;
- É fácil criar programas isolados e especialistas que resolvem problemas pontuais. E depois, como garantir o legado? Como integrar com sistemas maiores? Como evoluir? etc;
- Quanto a documentação de sistemas, vale a frase (modificada) de Scott W. Ambler, “pior do que não ter documentação, é tê-la desatualizada”. Com ferramentas de geração de código capazes de gerar muitos programas em poucas horas/minutos, será conseguimos uma documentação ajustada que represente as regras de negócio do cliente? Será que utilizando “wizards” estaremos livres do problema da dependência tecnológica e de alcançar maior confiabilidade?

## 6.5 Tecnologia e continuidade

Embora na maioria das vezes um único programa seja simples, sob o aspecto de fluxo (entrada, processamento e saída), existem aqueles programas que são bastante complexos porque formam os “alicerces” construídos para garantir a integridade dos sistemas existentes. Estes programas complexo interagem com diversos recursos, sejam eles de software ou de hardware.



Muitas empresas, tanto de pequeno quanto de médio porte, podem ter sistemas de informação com mais de 1.000.000 de linhas de código. Também é muito comum terem softwares contábeis, de RH e comerciais desenvolvidos em tecnologias diferentes “conversando” entre si por meio de rotinas de integração. As próprias rotinas de integração formam sistemas complexos uma vez que obedecem aos mesmos paradigmas de um sistema administrativo, p.ex, são escritos em uma linguagem, executam um fluxo básico (entrada, processamos e saída), estão amparados por uma tecnologia, etc.

O desenvolvedor deve estar sempre atento a tudo isto, deve estar preparado e ser muito organizado para não desconsiderar detalhes impostos pela integração e evolução dos sistemas. Os cuidados devem ser redobrados quando atua em ambientes com muitos programas e muitas planilhas eletrônicas personalizadas, redundantes e (o pior) desatualizadas e inconsistentes.

Problemas:

- Ambientes complexos que utilizam diferentes tecnologias exigem profissionais mais caros e experientes;
- Um profissional que atua em ambiente com alto grau de complexidade tende a ficar frustrado porque é o pivô para manter tudo funcionando e, conforme as suas aspirações profissionais, pode querer um ambiente mais dinâmico.
- Muita complexidade pode desviar o foco do profissional que volta-se para os problemas tecnológicos antes mesmo de pensar nos requisitos, deixando passar erros básicos, p.ex, não encerrar conexões de banco de dados, não inicializar variáveis, utilizar instruções fixas dentro dos programas, não ter tempo de avaliar novos produtos, ser mais reativo do que pró-ativo, etc;
- A tecnologia muda rapidamente e impacta diretamente sobre as estratégias administrativas das organizações. O uso de uma tecnologia específica pode gerar resultados fantásticos mas que, em médio prazo, acabam tornando-se um pesadelo a ponto de inviabilizar novos investimentos e impedir o crescimento do negócio, p. ex., um empresa com um sistema que monitora equipamentos de uma célula (gargalo) de produção cuja tecnologia usada foi descontinuada e não tem mais suporte técnico. Nesta empresa fictícia os funcionários reúnem-se diariamente para pedir a Murphy que ele não faça um visita.
- A tecnologia pode ser a solução de muitos problemas, no entanto é cara e pode ter vida curta. Além disto, a tecnologia normalmente nos brinda com um “laço de união” tão forte exigindo que acompanhemos o ritmo ditado pelos seus fornecedores, quando mudam temos que mudar também. Nestes casos, os custos normalmente se elevam e novas exigências aparecem, seja com gastos em treinamentos, conversões/adequações às novas “features”, compra de novos equipamentos ou, quem sabe, até mesmo para a troca de tecnologia porque a atual foi descontinuada.

## 6.6 Atualização e manutenção

Para muitos o mais difícil não é construir o sistema, mas sim mantê-lo. Manutenção de sistemas pode vir a ser a correção de “bugs” ou a própria evolução ou adaptação dos sistema, já que os requisitos mudam.

Problemas:

- Sistemas são construídos por pessoas que podem não estar mais presentes ou interessadas. Este problema pode ser amenizado se a documentação está “ok” e atualizada, caso contrário...
- Novas demandas de negócio exigem o uso de tecnologias exclusivas que não existiam quando o sistema foi concebido. Requisitos desta natureza podem elevar demasiadamente o custo de manutenção das aplicações. P.ex, o diretor investe em um software de BI instalado no Windows que depende da extração dos dados do seu ERP que executa no Linux com um banco de dados hospedeiro, ultrapassado e sem “drivers” de acesso;

## 6.7 Segurança

A segurança é um item presente na mente de qualquer empresário. Para um diretor ou gestor, é inconcebível homologar uma solução cujo grau de segurança seja baixo ou pouco confiável.



Embora o conjunto de softwares que dão suporte aos sistemas desenvolvidos já tenham em si soluções de segurança, os sistemas construídos também devem atender este requisito. Segurança diz respeito à restrição de acessos, proteções contra “hackers”, “spans” e criptografias de dados, etc. Os sistemas de informação devem oferecer mecanismos para controle de acessos e garantia da integridade do negócio, restringindo visibilidade e atualizações das informações da organização.

Problemas:

- Desenvolvem-se sistemas inteiros e as rotinas de controle de acesso são criadas apenas ao final do projeto. Esta estratégia não é interessante porque pode gerar um solução incompleta e com falhas de integração com as aplicações;
- A segurança muitas vezes recai para o profissional mais técnico (p.ex, DBAs, suporte...). Este profissional é muito capacitado para tratar de segurança de acessos, mas quanto a segurança relativa aos sistemas de gestão. Esta preocupação é do gestor, do usuário, é ele quem conhece como os dados pode ser processados e quem pode ter acesso as informações;

## 6.8 Implantação e migração

A implantação é uma das etapas mais complexas do processo de desenvolvimento de sistemas. O fracasso de uma implantação pode por em risco todo o investimento e inviabilizar o uso do sistema.

Aqui destacamos alguns desafios:

1. Fazer com que o esforço para implantar/migrar seja mútuo, tanto por parte da equipe de desenvolvimento quanto por parte da organização e usuários. Implantar significa mudar. Se mudamos o que está “funcionando” por algo novo batemos de frente com os interesses pessoais, p.ex., para alguns implantar significa promoção, para outros significa incerteza, dúvida, para outros demissão, etc. O sucesso da implantação vai depender das pessoas, as quais podem normalmente tem interesses muito diferentes e conflitantes.
2. Planejar e fazer testes de integração para migrar os dados de um sistema para outro ainda nas fases iniciais do desenvolvimento. A dificuldade será muito maior quando a migração se dá entre sistemas com tecnologias diferentes. Por exemplo, migrar um sistema de gestão desenvolvido em Cobol para uma nova aplicação web desenvolvida em Java, com banco de dados cuja estrutura são complementemente diferentes, o primeiro com arquivos sequenciais e o segundo apoiado em um banco de dados relacional.

Problemas:

- Utilizar o modelo “waterfall”, ou seja, ao desenvolver sistemas completos esgotando as etapas análise, depois a etapa de desenvolvimento, depois a etapa de testes e documentação, para finalmente implantar o sistema que podem não ser mais o que o usuário quer/precisa;
- Não considerar as dificuldades de transferir dados que uma tecnologia para outra. Muitas vezes faz-se um terceiro sistema que é o conjunto de programas que deve converter os dados do sistema atual para o novo sistema. Com novas tecnologias, os banco de dados passaram a armazenar dados em formatos e estruturas mais complexas, diferente daquelas utilizadas em modelos mais antigos. Este fenômeno ocorre até mesmo nas trocas de versões de um mesmo software, p.ex, maior capacidade de armazenar textos, imagens, arquivos, etc.

## 7 Desenvolvendo com o Telescope

Pregamos que a qualidade deve estar no processo, não apenas no produto. Desta forma estabelecemos que o nosso ambiente deve garantir o nosso processo de software.

Características que buscamos em nosso processo de software:

- Ser verificável
- Permitir priorização de requisitos
- Facilitar modificações e manutenibilidade



- Permitir rastreamento e análise de impacto
- Ser inteligível
- Ser correto
- Ser consistente
- Não ser ambíguo

Também priorizamos a documentação conceitual de um sistema de forma que as suas especificações, objetos e relações sejam mantidas na organização como patrimônio e não como conhecimento de propriedade de poucos.

A forma como conduzimos a documentação no Telescope favorece o mapeamento conceitual dos elementos, permitindo maior compreensão dos requisitos, maior reutilização dos elementos documentados e total independência tecnológica. A documentação tem um peso muito significativo para o Telescope, já que a geração de códigos depende diretamente da documentação, ou seja, sem documentar o elemento no repositório, não é possível gerar código.

As ferramentas do Telescope fazem uso de um repositório planejado onde são armazenados os elementos que serão utilizados pelos sistemas gerados. É por meio deste repositório que conseguimos registrar e armazenar nossa base de conhecimento.

Utilizando o repositório, passamos a mapear os sistemas em um único modelo, já que está armazenado de forma estruturada e organizada, por exemplo:

- Sistemas, “builds”, acessos
- Tipos de dados, entidades, atributos
- Regras do negócio, eventos, métodos
- Interfaces, telas, campos
- Menus, acessos, usuários,
- Logs, setups...

Vantagens para utilizar um modelo único:

- Maior produtividade
- Menor quantidade de erros
- Melhor comunicação entre usuários, analistas e projetistas
- Maior velocidade com melhor resultado
- Melhor qualidade
- Maior flexibilidade (manutenibilidade)

Abaixo apresentamos como o Telescope nos atende e como superamos os problemas descritos no item anterior.

## 7.1 Valorizar “o que é” e não “o como se faz”

O Telescope valoriza o conceito e permite a completa presença do usuário durante as fases de desenvolvimento.

Como:

- As interfaces do I-CASE foram montadas para acelerar o desenvolvimento, facilitando a modelagem conceitual e documentação das regras de negócio sem referenciar a tecnologia na qual será implementada;
- O usuário define o sistema conforme a sua linguagem, explorando naturalmente a técnica do refinamento sucessivo (evoluindo gradativamente);
- O Telescope oferece prototipação “on-line” do modelo de persistência e interfaces do sistema;



- O Telescope permite utilizar sistemas já definidos, permitindo abstrair suas funcionalidades porque já estão modeladas e especificadas no repositório;

## 7.2 Modelagem conceitual, lógica e física

O repositório do Telescope é uma base de conhecimento, estruturada com hierarquias, relacionamentos e associações de elementos. Tudo isto integrado, obedecendo a um modelo único e conceitual.

Como:

- Sendo um modelo único, não existe a perda da tradução de modelos. O que está documentado, associado e relacionado não sofre tradução, pois o código é gerado conforme o que é apresentado na prototipação;
- Sem vínculo com tecnologia, a facilidade de retroceder e revisar os modelos é muito grande. Além disto, a prototipação colabora e enriquece o trabalho do analista de sistema para que – junto com o usuário – problemas sejam identificado com maior velocidade evitando retrabalhos;
- O Telescope permite alto grau de modularização onde o programador adiciona o código em métodos especificados conforme as regras do negócio. Desta forma, o comprometimento do analista e programador aumenta porque os métodos são modelados exclusivamente para atender as regras do negócio, sem que o programador desvie sua atenção para os algoritmos do tipo CRUD (já garantidos pela base de conhecimento armazenada);
- As interfaces do repositório (I-CASE propriamente dito) permitem que a documentação dos elementos do sistemas seja feita com a linguagem do próprio usuário;
- A ferramenta conduz o trabalho de forma objetiva e apresenta protótipos nas interfaces do I-CASE. Esta facilidade aumenta a produtividade pois a crítica é localizada, tudo sem a necessidade de traduzir modelos ou navegar por outras telas.

## 7.3 Reaproveitamento

A base de conhecimento do Telescope concentra os conceitos modelados para as aplicações desenvolvidas. Dela são extraídos os elementos dos modelos documentados no repositório. Estes elementos de sistemas podem e devem ser reutilizados sempre que possível.

Como:

- Os sistemas registrados na base de conhecimento do Telescope se utilizam de outros sistemas já documentados, p.ex., o sistema de pedidos utiliza o sistemas de pessoas e produtos já cadastrados;
- O repositório do Telescope possibilita que elementos sejam reutilizados através da associação de sistemas com outros já prontos ou em desenvolvimento. Isto garante maior produtividade e maior retorno do investimento porque, sempre que um sistema evolui, os demais sistemas que se utilizam dele também evoluem;

## 7.4 Implementação e confiabilidade

A partir do Telescope geramos códigos completos com persistências, validações, transações de banco de dados, captura de erros, etc. A geração sempre inclui códigos necessários que, mesmo para sistemas pequenos, todo desenvolvedor deve “gastar” tempo programando e muito importante testando.

Com a potencialidade da geração, a qualidade e a funcionalidade dos sistemas são altas com códigos gerados conforme a sua documentação e representação conceitual registradas na base de conhecimento. Modificações no seu conceito afetam os códigos gerados, porém mantém a estrutura sem que haja o risco de faltar o tratamento transacional, relacionamentos com outros entidades/interfaces, uso de tipagens incorretas de dados, regras de negócios desatualizadas, etc.

Além da qualidade final, a confiabilidade é muita porque os códigos gerados representam o que está documentado no repositório.

Como:

- O sistema terá o comportamento conforme está documentado no repositório, este comportamento é prontamente apresentado na prototipação. Desta forma, a confiabilidade em relação ao sistema final é muito grande porque a documentação é que define realmente como o sistema funciona. Sem documentar, não é possível gerar o sistema.
- O Telescope, por meio do uso de geradores que extraem dados da base de conhecimento e pela reutilização que potencializa, reduz o retrabalho e aumenta o retorno do investimento;
- A ferramenta I-CASE do Telescope orienta o desenvolvimento de forma objetiva sem fazer referências a tecnologia na qual o sistema será gerado. Se a tecnologia mudar, muda-se o gerador, mas a base de conhecimento permanece.

## 7.5 Tecnologia e continuidade

Para obter maior retorno do investimento e garantir maior ciclo de vida, os sistemas devem ser definidos e modelados sem considerar a tecnologia na qual serão implementados. O retorno aumenta mais ainda quando se consegue modelar o seu comportamento através de regras de negócio, restrições e validações sem que a tecnologia seja restrição ou requisito.

Como:

- O Telescope permite modelar conceitualmente os sistemas, definindo “o quê” e “o como” sem definir a tecnologia de implementação;
- O Telescope permite especificar entidades, atributos, regras de negócio, sequência de eventos, etc, tudo para detalhar o comportamento dos sistemas e seus elementos sem que uma linha de código da linguagem final seja digitada;
- O Telescope permite que o desenvolvimento seja feito focado no requisito. Os códigos gerados são padronizados e representam fielmente o protótipo. Por exemplo, o desenvolver não precisa “perder” tempo com algoritmos de “CRUD”, validações de restrições e tipagem de dados, transações, tratamento de erros, logs, etc;

## 7.6 Atualização e manutenção

O repositório que dá sustentação a base de conhecimento é uma das principais ferramentas do Telescope, porque potencializa a produtividade do nosso processo de software e também mantém a massa de dados com todos os elementos modelados.

Tendo em mão a documentação conceitual das aplicações geradas, mais o uso da capacidade de consultar e executar análises de impacto, maximizamos as atividades de atualização e manutenção dos sistemas.

Como:

- O repositório armazena os elementos dos sistemas. p.ex. entidades, atributos, relacionamentos, regras, interfaces, telas, menus, etc. Todos estes conceitos fazem parte da base de conhecimento do Telescope. Desta forma, não existe a dependência de pessoas, fornecedores, profissionais, etc;
- A base de conhecimento armazena o que foi necessário para gerar os sistemas, porém, pode ser aumentada e melhor detalhada dando origem a novas versões de sistemas mais capazes e completos;

## 7.7 Segurança

Sistemas desenvolvidos que já nascem com a preocupação de segurança desde sua concepção, aumentam a confiabilidade do usuário e do investidor, além de qualificar a solução final.

Como:

- O gerador do Telescope adiciona mecanismos de segurança nos códigos finais gerados, atendendo ao requisito de segurança de acesso;



- O gerador do Telescope também adiciona mecanismos de criação de logs nos códigos finais gerados, atendendo ao requisito de garantir/permitir a rastreabilidade da informação;
- Pelo fato de adicionar códigos tão relevantes nas aplicações, fazendo com que os sistemas sejam implantados naturalmente com estas características, tem-se um ganho imenso de produtividade no processo de software porque elimina esta preocupação durante o desenvolvimento, eliminando digitação de códigos e testes massivos;

## 7.8 Implantação e migração

O Telescope prega (e exige) a presença do especialista (usuário) durante o desenvolvimento do sistema. Primeiro porque o especialista é quem conhece o negócio, segundo porque o I-CASE foi construído para permitir/garantir que seja desta forma.

Como:

- O usuário participa do processo de software do Telescope desde a sua concepção. Para o Telescope, sua presença é fator crítico de sucesso, desta forma, o resultado é construído em conjunto e todos são responsáveis pelo sucesso do sistema em desenvolvimento;
- Com o Telescope, os ciclos de desenvolvimento são curtos e reduzidos. Rapidamente consegue-se especificar, modelar, prototipar e gerar aplicações inteiras. Esta velocidade permite que requisitos estejam funcionais em curtos espaços de tempos. Assim, o usuário logo requisita migrações de dados, tanto para validar as versões já em uso quanto para utilizar os sistemas gerados em produção;
- Com ciclos curtos, antecipamos cargas de dados, problemas de especificação, validações e regras de negócios, reduzimos a ambientação de todos com novas versões de sistemas, etc;

## 8 Conclusão

O Telescope não é a “fada madrinha” ou “a bala de prata” que soluciona tudo, nem mesmo foi criado para ser a única ferramenta capaz de resolver problemas com desenvolvimento de sistemas. Longe disto, o Telescope é um projeto que nasceu, mas continua em andamento. Na verdade não queremos que encerre porque, os requisitos mudam, os governos mudam, os fornecedores mudam, os clientes mudam, as pessoas mudam.

Como filosofia, gera resultados excelentes que atendem as características citadas anteriormente:

- Permite verificação e crítica;
- Permite que os requisitos seja priorizados;
- Facilita modificações e manutenibilidade;
- Permite rastreamento e análise de impacto;
- É inteligível porque tem um I-CASE com interfaces para registro, manutenção e recuperação de dados;
- É correto no sentido de gerar soluções padronizadas com alta funcionalidade e usabilidade;
- É consistente e continua evoluindo;
- É única, porque tem uma base de conhecimento com elementos de sistemas que podem ser reutilizados;

Em relação aos sistemas gerados, é importante destacar que o Telescope utiliza códigos finais digitados (na tecnologia escolhida), mas não necessita deles para gerar os sistemas, ou seja, mesmo sem implementar qualquer código em uma linguagem específica, as aplicações geradas já nascem com comportamento que vão além do “CRUD” padrão. Por exemplo:

- Os conceitos estão em um repositório, portanto podem ser modificados e regenerados constantemente;
- As interfaces são geradas com listas de valores customizadas conforme os modelos de dados;



- As aplicação controlam transações de banco (nas interfaces e nas camadas de persistências);
- As regras de negócios são documentadas e geram códigos de validação e de integridade da base de dados;
- Os sistemas são gerados com camadas de persistências, validações e regras de negócios independentes das interfaces, mas associadas e integradas para captura e tratamento de erros unificados;
- Os sistemas são gerados com regras de acesso especificados já em fase de modelagem;
- Os sistemas são gerados com mecanismos de logs para rastreabilidade e controle de acessos;

Acreditamos no Telescope porque são muitos anos de experiências materializadas neste projeto. O próprio I-CASE foi gerado nele mesmo, ou seja, após a versão inicial, toda a evolução que conseguimos adicionar no I-CASE como sistema foi utilizando ele para documentar os seus conceitos na nossa base de conhecimentos e para gerar novas versões aumentando suas funcionalidades. Tudo como se estivéssemos construindo um sistema comercial.