



## **O Processo de Desenvolvimento Telescope**

**PRD Tecnologia de Gestão Ltda.  
Julho/2008**



## Sumário

Introdução.....	3
O desenvolvimento de software tradicional.....	3
O problema da produtividade.....	3
O problema da portabilidade.....	6
O problema da interoperabilidade.....	6
O problema da documentação e manutenção .....	7
O modelo de desenvolvimento Telescope.....	7
O ciclo de vida do desenvolvimento Telescope.....	8
Sistemas e sub-sistemas.....	9
Gestão de requisitos.....	9
Modelo de negócio independente de arquitetura.....	9
Geradores para arquiteturas específicas.....	10
Builds.....	10
Código.....	11
Elevando o nível de abstração.....	11
Automatização das etapas da transformação.....	11
Benefícios do Telescope.....	11
Produtividade .....	12
Portabilidade.....	12
Interoperabilidade.....	13
Documentação e Manutenção.....	13
Outros benefícios.....	14
Os elementos chaves do Telescope.....	14
Conclusão.....	15



## Introdução

Este artigo descreve alguns dos principais problemas no desenvolvimento de softwares e como o processo de desenvolvimento Telescope ajuda a resolver estes problemas.

## O desenvolvimento de software tradicional

O desenvolvimento de software é frequentemente comparado com desenvolvimento de hardware em termos de maturidade. Enquanto no desenvolvimento de hardware tem havido muito progresso, (a velocidade do processador cresceu exponencialmente em trinta anos), os progressos realizados no desenvolvimento de software parecem ser mínimos. Mas na realidade, os progressos realizados no desenvolvimento de software não podem ser medidos em termos de velocidade ou custo no desenvolvimento.

Os progressos no desenvolvimento de software são evidentes a partir do fato de que é possível a construção de sistemas muito maiores e complexos. A 30 anos, as aplicações envolviam apenas sistemas rígidos, sem interfaces gráficas, com funcionalidades limitadas e sem qualquer conexão com outros sistemas. Não existem mais desenvolvimento de sistemas deste tipo, e é por isso que não temos a idéia de valores sólidos dos progressos que foram feitos.

O desenvolvimento de software é uma área que enfrenta uma série de problemas. Escrever software é um trabalho intenso. Por um lado, cada vez que surge uma nova tecnologia, muito trabalho precisa ser feito de novo e de novo. Além disso, os sistemas nunca são construídos utilizando somente uma tecnologia, e sistemas precisam sempre se comunicar com outros sistemas e que provavelmente utilizam outras tecnologias.

Por outro lado, as novas tecnologias estimulam novas possibilidades. Por sua vez, estas novas possibilidades se transformam rapidamente em novas necessidades de um mercado que exige cada vez mais inovações tecnológicas como diferencial competitivo.

Para mostrar como o Telescope pode auxiliar muito na construção de sistemas, nós analisaremos alguns dos mais importantes problemas com o desenvolvimento de software bem como as suas causas.

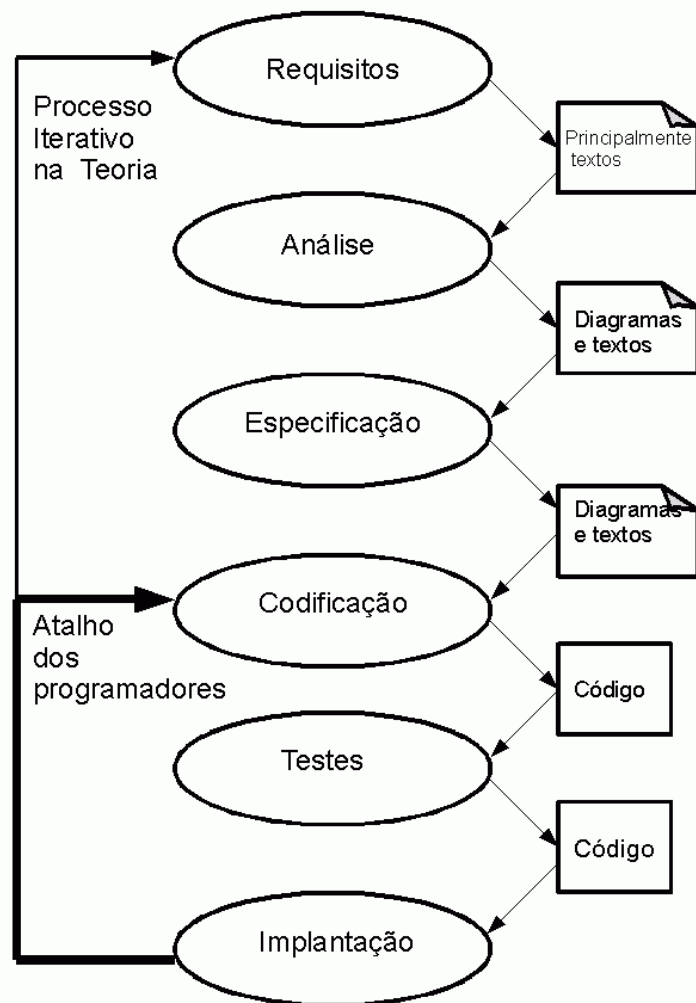
## O problema da produtividade

O processo de desenvolvimento de software tal como conhecemos hoje é frequentemente impulsionado pelo desenvolvimento e codificação de baixo nível feito por analistas de sistemas e programadores.

O processo típico, como ilustrado abaixo, inclui diversas fases:

1. Definição dos requisitos
2. Análise e descrição funcional
3. Projeto
4. Codificação
5. Testes
6. Implantação

Se nós usássemos uma versão incremental e iterativa deste processo, ou o tradicional processo em cascata, documentos e diagramas seriam produzidos durante as fases 1 à 3. Estes incluem descrição de requisitos em textos e imagens, e frequentemente muitos diagramas UML (Unified Modeling Language) de casos de uso, diagramas de classes, diagramas de interação, diagramas de atividade, e assim por diante. A pilha de papel produzida é às vezes impressionante. No entanto, a maior parte dos artefatos resultantes destas fases é apenas papel e nada mais.





Os documentos e diagramas criados nas primeiras três fases perdem rapidamente o seu valor a partir do começo da codificação. A relação entre os diagramas e o código vai desaparecendo à medida que a fase de codificação progride.

Além disso, são muitos tipos de diagramas diferentes e, muitas vezes, nem tão fáceis de interpretar. A própria UML é uma “linguagem” que deve ser devidamente utilizada e que frequentemente causa dúvidas de interpretação.

Na medida que o sistema é alterado ao longo do tempo, a distância entre o código e os textos e diagramas produzidos nas primeiras três fases torna-se maior. Mudanças são frequentemente feitas nos códigos somente, porque o tempo para atualizar os diagramas e outros documentos de alto nível não é disponível. Além disso, as vantagens de manter os diagramas e documentos atualizados são questionáveis, uma vez que qualquer nova mudança acaba sendo sempre iniciada pelo código.

A idéia do Extreme Programming (XP), bem como de diversas práticas “Ágeis” de desenvolvimento, tornou-se moda popular rapidamente. Uma das razões para tal é que ele reconhece o fato de que o código é a força motriz no desenvolvimento de software. As únicas fases no processo de desenvolvimento que são realmente produtiva são codificação e testes.

Como afirma Alistair Cockburn no Agile Software Development (Cockburn 2002), a abordagem XP resolve apenas parte do problema. Enquanto uma mesma equipe trabalha sobre o software, há conhecimento suficiente de alto nível em suas cabeças para compreendê-lo. Durante o desenvolvimento inicial este é frequentemente o caso. Mas após a entrega da primeira versão do software, a equipe é frequentemente desfeita e então surge um grande problema. Outras pessoas precisam manter o software, corrigindo erros, acrescentando funcionalidades, integrando com outros sistemas, melhorando sua usabilidade e assim por diante. Neste momento, aquela documentação produzida nas fases 1 a 3 contribuem muito pouco e buscar o entendimento de como um sistema funciona através de seu código composto de provavelmente milhares de linhas não é tarefa fácil.

Então, ou usamos o nosso tempo nas primeiras fases de desenvolvimento de software construindo altos níveis de documentação e diagramas, ou usamos o nosso tempo na fase de manutenção descobrindo o que o software está realmente fazendo. Em ambos os casos, não estamos sendo produtivos no sentido de produzir código.

Muitos desenvolvedores consideram que escrever código é produtivo e que escrever modelos ou documentação gera uma sobrecarga desnecessária. Mesmo assim, em um projeto de software maduro essas tarefas tem que ser feitas.



## O problema da portabilidade

A indústria de software tem uma característica especial que faz com que seja diferente de outras indústrias. A cada ano, novas tecnologias estão sendo inventadas e tornando-se populares (por exemplo Java, XML, HTML, SOAP, UML, J2EE, .NET, JSP, ASP, Flash, Web Services, e assim por diante). Muitas companhias necessitam adotar essas novas tecnologias por diversas razões:

- A tecnologia é exigida pelos clientes.
- Resolve alguns problemas reais.
- Fornecedores descontinuam suporte a ferramentas que utilizam as velhas tecnologias dando foco para a nova.

As novas tecnologias oferecem benefícios tangíveis para as empresas e muitas delas não podem se dar ao luxo de ficar para trás. Por isso, as pessoas têm de adotar essas novas tecnologias o mais rápido possível. Como consequência, os investimentos feitos em tecnologias anteriores perdem seu valor podendo até se tornar inúteis.

Além disso, as tecnologias já adotadas também mudam. Novas versões são criadas e frequentemente sem garantias de compatibilidade com as versões anteriores.

Como consequência, muitos softwares existentes são portados para uma nova tecnologia, ou para a uma versão mais nova da tecnologia em uso, agregando poucas ou mesmo nenhuma nova funcionalidade real, mas apenas para se manter compatível com uma nova versão de sistema operacional, um novo banco de dados, um novo protocolo de comunicação, etc.

## O problema da interoperabilidade

Sistemas de software raramente vivem isolados. A maior parte dos sistemas precisam se comunicar com outros já existentes. Um exemplo típico é o que vem ocorrendo nos últimos anos onde muitas empresas têm construído novos sistemas baseados na Web mas que precisam buscar informações de seus sistemas legados.

Mesmo quando o sistema é construído totalmente do zero, ele frequentemente mistura múltiplas tecnologias, às vezes velhas e novas. Por exemplo, o sistema usa EJBs mas precisa persistir as informações em um banco de dados relacional.

Ao longo dos anos aprendemos a não criar grandes sistemas monolíticos e complexos. Em vez disso, buscamos construir vários sub-sistemas com escopos específicos e integrados entre si. Isso facilita (e torna possível) as manutenções no sistema como um todo. Cada sub-sistema pode usar a tecnologia mais apropriada para aquele escopo, mas cria o problema da interoperabilidade entre eles.



## O problema da documentação e manutenção

Documentação sempre foi um elo fraco no processo de desenvolvimento de software. A maioria dos desenvolvedores sentem que sua tarefa principal é produzir o código e que escrever a documentação durante o desenvolvimento custa tempo e atrasa o processo. A disponibilidade de documentação ajuda os novos desenvolvedores que irão entrar no processo mais tarde, ou seja, do ponto de vista do desenvolvedor, a documentação reduz o seu vínculo com o seu contratante, e o que ele deseja conscientemente ou não, é justamente o oposto.

Estas são algumas das razões principais pelas quais a documentação não é frequentemente de boa qualidade. As únicas pessoas que podem verificar a qualidade são os colegas de desenvolvimento que odeiam escrever a documentação tanto quanto ele. Da mesma forma, estas também são razões que fazem com que a documentação não seja mantida atualizada. A cada mudança no código a documentação necessita ser atualizada.

Naturalmente, os desenvolvedores estão errados. A sua tarefa é desenvolver sistemas que possam ser mantidos e melhorados. Apesar do sentimento de muitos desenvolvedores, escrever a documentação é parte essencial de suas tarefas.

Uma das soluções propostas para esse problema são os mecanismo de gerar a documentação a partir do código fonte. A documentação seria na verdade uma parte do código, e não uma entidade separada. Esta idéia é apoiada em várias linguagens de programação, como Eiffel e Java. No entanto, esta solução só resolve o problema de baixo nível de documentação. O alto nível (texto e diagramas) ainda necessita ser mantido à mão. Dada a complexidade dos sistemas que são atualmente construídos, a documentação em um nível mais elevado é uma necessidade absoluta.

## O modelo de desenvolvimento Telescope

O modelo de desenvolvimento Telescope é uma estrutura para desenvolvimento de software definida pela PRD que se assemelha em muitos conceitos ao MDA definido pela Object Management Group (OMG). A chave do Telescope é a importância dos modelos no processo de desenvolvimento de software. Dentro do Telescope, o processo de desenvolvimento é conduzido pela atividade de detalhamento de seu sistema de software através da sua descrição semântica.

Ferramentas de desenvolvimento CASE tem sido historicamente vistas de forma negativa porque muitos gerentes de projetos não suportaram a curva de aprendizado necessária para compreender como estas ferramentas trabalham. Devido a muitos destes insucessos iniciais, criou-se um estigma em torno da palavra "CASE" e muitas empresas se afastaram totalmente do conceito.



O Telescope pode ser comparado com uma ferramenta CASE de desenvolvimento de sistemas, porém com diversas características inovadoras com relação às soluções CASE existentes no mercado:

- **Liberdade de tecnologia** – O Telescope não é uma ferramenta que gera em uma tecnologia específica. A transformação do modelo semântico em código fonte pode ser livremente implementada.
- **Colaborativo** – Todos os diagramas e especificações do sistema são mantidos em um repositório centralizado armazenado em uma banco de dados e acessados através de uma interface Web com telas voltadas para a produtividade da equipe.
- **Respeito ao programador** – O Telescope não substitui o programador. Procedimentos com algoritmos complexos continuam a ser programados manualmente. Em muitos casos, um trecho de código escrito em uma linguagem genérica ou em uma DSL, são a melhor maneira de descrever um algoritmo.
- **Geração 100%** – Mesmo existindo trechos codificados manualmente, eles são mantidos no repositório. A geração insere estes códigos nos locais adequados. Os códigos finais jamais são alterados.
- **Integridade** – Como o repositório é armazenado em uma base de dados, a própria integridade relacional do BD garante a integridade entre os artefatos do sistema. É impossível excluir um elemento que possui dependências ou criar uma relação inexistente.
- **Centrado na documentação** – Nenhum programador está apto a criar um novo elemento, quer seja uma entidade, um atributo, uma regra de negócio, um método, uma tela, uma ação, etc, sem primeiro descrevê-lo.

A seguir, explicaremos primeiramente o ciclo de vida básico do desenvolvimento Telescope e em seguida mostraremos como o mesmo ajuda na solução dos problemas mencionados anteriormente.

## O ciclo de vida do desenvolvimento Telescope

O ciclo de vida do desenvolvimento Telescope, não se mostra muito diferente do ciclo de vida tradicional. As mesmas fases são identificadas. Uma das principais diferenças encontram-se na natureza dos artefatos que são criados durante o processo de desenvolvimento. No Telescope, os artefatos são modelos formais e que são compreendidos pelo computador.

O conjunto de modelos é composto na realidade de uma massa de especificação do sistema detalhada a tal ponto que permite geração de grande parte do mesmo.



## **Sistemas e sub-sistemas**

O primeiro nível de divisão dos meta dados que o Telescope usa são chamados de “Sistemas”. Cada sistema pode ser um software completo ou apenas um módulo de um sistema maior. Dentre os inúmeros elementos que compõe um sistema, existe um que informa as dependências de um sistema com relação a outros sistemas (ou sub-sistemas).

Isso permite dividir o escopo de grandes sistemas em diversos escopos menores, muitas vezes genéricos e de fácil reutilização.

Para cada sistema/sub-sistema, é permitido total controle de quais usuários da equipe de desenvolvimento tem acesso para leitura, alteração, geração, etc.

## **Gestão de requisitos**

Para cada sistema, são registrados os seus objetivos, requisitos funcionais, requisitos não funcionais, etc. Cada requisito é documentado através de um caso de uso, bem como possíveis protótipos propostos e plano dos respectivos testes funcionais e/ou unitários.

O plano de testes é parte fundamental para o sistema e sugerimos que seja concebido junto dos requisitos. Mesmo que o teste ainda não esteja implementado, ele já aparecerá nas listas de testes automatizados do respectivo sistema.

A medida que o desenvolvimento progride, os demais artefatos especificados serão vinculados com os requisitos que estão sendo atendidos, criando total rastreabilidade entre os requisitos e os demais elementos do sistema.

## **Modelo de negócio independente de arquitetura**

O modelo de meta dados que especificam um software no Telescope, são de um nível de abstração suficientemente alto para ser independente de tecnologia mas não excessivamente abstrato a ponto de se perder uma relação direta entre os elementos conceituais e suas implementações físicas.

Durante o desenvolvimento, o sistema é definido do ponto de vista de como este suporta melhor o negócio. O fato do sistema ser implementado em um mainframe com um banco de dados relacional ou em uma aplicação de servidor EJB não faz a menor diferença no seu detalhamento conceitual.

Este modelo envolve uma grande quantidade de tipos diferentes de elementos, tais como domínios, entidades, atributos, regras de relacionamento, regras de validação, regras de inferência, eventos, procedimentos, funções, interfaces, telas, formulários, regras de editabilidade/visibilidade, seqüências, ações, etc.

Cada tipo de elemento está relacionado com um formato específico e objetivo, mas sempre exigindo um determinado grau de documentação e integridade.

## Geradores para arquiteturas específicas

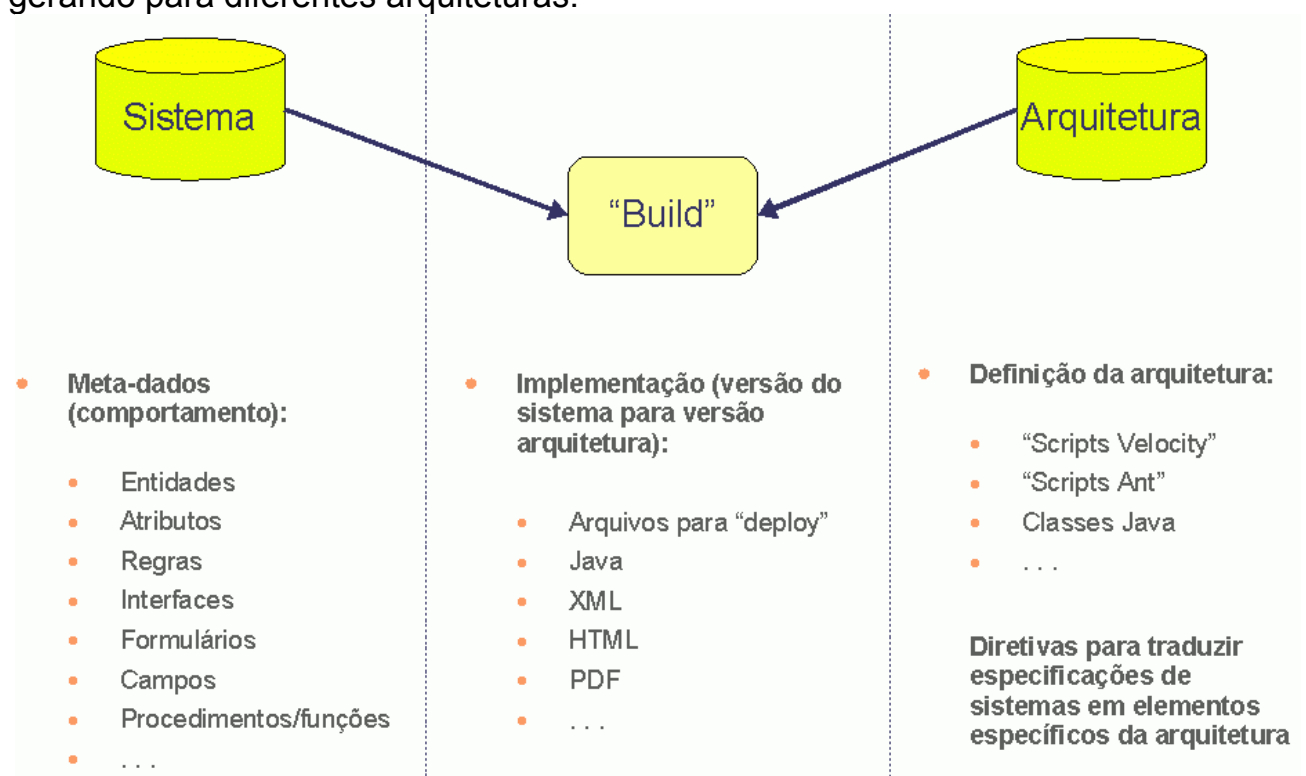
Em paralelo com a especificação do sistema sob o ponto de vista de negócio, uma outra equipe trabalha na especificação da arquitetura para atender o ponto de vista tecnológico. Esta equipe estabelece a melhor combinação tecnológica que deverá ser usada na construção do sistema, definindo os elementos necessários (templates, scripts e outros mecanismos de geração) para implementar o modelo na arquitetura desejada.

Por exemplo, podemos utilizar geradores para criar uma arquitetura EJB onde classes de entidades, ouvintes de entidades, arquivos de configuração da persistências, etc, são gerados a partir das definições do modelo de negócio.

## Builds

No Telescope, quando queremos gerar um determinado modelo de negócio em uma determinada arquitetura, criamos um elemento chamado de Build.

Podemos ter vários Builds diferentes para o mesmo modelo de negócio, cada um gerando para diferentes arquiteturas:





## **Código**

Uma vez geradas as saídas de um build, os códigos fontes podem ser compilados e empacotados conforme definido para a própria tecnologia. Existem várias situações onde o programador se sente atraído em modificar o código resultante. Em geral isso é uma má idéia, mas existem situações onde o ganho da manutenção deste código pode representar um ganho significativo de produtividade, principalmente com as IDEs atuais ricas em facilidades de edição tais como auto complete, verificações de sintaxe, etc. Para estes casos, o Telescope disponibiliza mecanismos de integração entre o código fonte e o repositório através de serviços de captura destas modificações formando uma solução de round-trip.

## **Elevando o nível de abstração**

O modelo de negócio e os componentes de geração para uma arquitetura geram artefatos de diferentes etapas no ciclo de desenvolvimento. Mais importante, eles representam diferentes níveis de abstração na especificação do sistema. A capacidade de transformar um modelo de negócio diretamente em código final aumenta substancialmente o nível de abstração em que um programador pode trabalhar. Isso permite ao desenvolvedor lidar com sistemas mais complexos com menos esforço.

## **Automatização das etapas da transformação**

Tradicionalmente, as transformações de modelo conceitual para modelos físicos e destes para código são feitas, principalmente, a mão. Muitas ferramentas podem gerar código a partir de um modelo mas normalmente atendem a apenas uma parte do mesmo e o restante tem que ser programado manualmente.

Em contraste, no Telescope toda a geração é feita por ele e o código resultante não deve ser editado. No processo tradicional, gasta-se muito esforço em projetos com o trabalho de construir uma base de dados, componentes EJB, interfaces CRUD, etc, a partir de um mesmo modelo de alto nível.

No modelo Telescope, mesmo que o processo de definição da arquitetura não esteja definido, o modelo de negócio pode ser implementado e totalmente validado em uma outra arquitetura, permitindo ter um gabarito imediato do sistema com todas as suas funcionalidades básicas.

## **Benefícios do Telescope**

Vamos dar uma olhada mais atenta o que a aplicação do Telescope nos traz em termos de melhoria no processo de desenvolvimento de software.



## **Produtividade**

O objetivo do Telescope é direcionar o foco do desenvolvedor para o modelo de negócio. A aplicação dos frameworks e da arquitetura definida são preocupações dos técnicos especialistas. Naturalmente, alguém ainda precisará definir os componentes de geração, que é uma tarefa difícil e especializada. Mas tal atividade necessitará ser definida uma única vez e pode então ser aplicada no desenvolvimento de muitos sistemas. O retorno para o esforço no sentido de definir uma geração é grande, podendo ser feito apenas por pessoas altamente qualificadas. A maioria dos desenvolvedores irá focar-se apenas no desenvolvimento dos modelos de negócio. Especificações da plataforma alvo e detalhes técnicos serão adicionados automaticamente pelo Telescope no momento da geração.

Isto melhora a produtividade em duas maneiras. Em primeiro lugar, os desenvolvedores do modelo de negócio tem menos trabalho para fazer por que os detalhes específicos da plataforma não precisam ser projetados e escritos. Há muito menos código a ser escrito, porque uma grande quantidade do código é gerada automaticamente.

A segunda melhoria vem do fato de que os desenvolvedores podem concentrar o foco no modelo de negócio, assim, prestando mais atenção à resolução dos problemas da empresa. Isso resulta em um sistema que se adapta muito melhor com as necessidades dos usuários finais.

Tal produtividade só pode ser alcançada através da utilização de ferramentas totalmente automatizadas. Note que isto implica que uma grande parte da informação sobre a aplicação seja incorporada no modelo de negócio e/ou nos componentes de geração da ferramenta. Como o modelo de alto nível já não é "apenas papel," mas sim um conjunto de meta dados formais e relacionados diretamente como o resultado final, as exigências sobre a completude e consistência do modelo são mais elevados do que no desenvolvimento tradicional. Um humano lendo um modelo no papel pode ser indulgente - uma ferramenta automatizada de transformação não é.

## **Portabilidade**

No Telescope, a portabilidade é alcançada porque foca no desenvolvimento de modelos de negócio que são, por definição, independentes de plataforma. O modelo de negócio pode ser automaticamente implementado em várias plataformas diferentes. Tudo que você especificou no nível de modelo de negócio é, conseqüentemente, portátil.



A extensão que a portabilidade pode alcançar depende das ferramentas automatizadas de geração. Algumas plataformas populares estão disponíveis. Para plataformas mais específicas, ou para novas tecnologias e plataformas que chegarão no futuro a PRD constrói os geradores sob demanda e com o auxílio de especialistas na tecnologia em foco.

### **Interoperabilidade**

Como já falamos, o Telescope pode gerar códigos para diferentes plataformas. Muitas vezes precisamos que os sistemas destas plataformas também se comuniquem com outros sistemas em outras plataformas. Normalmente, para implementar estas comunicações entre elementos de diferentes plataformas, construímos artefatos que são utilizados como pontes entre os dois sistemas traduzindo os conceitos de uma plataforma nos conceitos da outra.

Como os componentes de geração estão sob nosso controle, podemos gerar estes componentes de comunicação da mesma forma com que geramos os demais componentes do sistema.

Se somos capazes de implementar um modelo de negócio em duas plataformas de tecnologias distintas, as informações que necessitamos para fazer a ponte entre eles está disponível.

Por exemplo: Digamos que seja necessário interagir um sistema gerado em uma plataforma Java com um sistema legado construído em uma plataforma Oracle e que usa Table APIs para manter as regras de negócio no banco de dados. Para um elemento Pedido, nós conhecemos a(s) classe(s) que foram geradas para ele, da mesma forma, nós conhecemos a(s) tabelas(s) deste elemento. Construir uma ponte entre o objeto Java que execute corretamente o acesso às operações de insert, update e delete é fácil.

Com o Telescope, você pode lidar muito mais facilmente com as mudanças tecnológicas, preservando o seu investimento no modelo conceitual de negócio.

### **Documentação e Manutenção**

Trabalhando com o ciclo de vida do Telescope, os desenvolvedores focam-se no modelo de negócio, que está em um nível maior de abstração que o código. Este modelo é interpretado e usado para gerar o código. O modelo é uma representação exata do código. Assim, o modelo cumpre a função da documentação de nível elevado que é necessário para todo o sistema de software.

A grande diferença é que é impossível fazer qualquer alteração no sistema sem alterar o modelo, ou seja, ao contrário do desenvolvimento tradicional, o modelo conceitual do negócio não está abandonado após a escrita. O sistema gerado é consequência direta do modelo.



Na abordagem Telescope, a documentação tem um bom nível de abstração e estão naturalmente disponíveis. Mesmo informações adicionais, que não são diretamente necessárias para o modelo podem ser mantidas. Isto inclui, por exemplo, a argumentação de escolhas que foram feitas durante o desenvolvimento.

Além disso, o Telescope possibilita a geração de artefatos de documentação da mesma forma que os fontes, permitindo saídas como manuais de usuários, manuais técnicos, etc.

## Outros benefícios

- Ganhos significativos no retorno do investimento (ROI) – Com as atividades de automação existentes, o Telescope atua diretamente no retorno do investimento, reduzindo os custos ou entregando um maior número de funcionalidades com o mesmo custo.
- O Telescope permite que o foco esteja voltado a elementos que tenham um ROI mais alto sem comprometer os prazos de entrega.
- Time-to-market – O modelo permite uma redução significativa dos ciclos de desenvolvimento melhorando a habilidade de inovação.
- Redução de riscos – O modelo simplificado de desenvolvimento reduz o risco de não entregar o software no prazo.
- Redução da barreira tecnológica – A equipe responsável em detalhar o modelo de negócio não precisa se preocupar com os detalhes tecnológicos.

## Os elementos chaves do Telescope

O Telescope não é uma ferramenta mágica. Ele é o resultado do somatório de diversos elementos:

- Modelos de alto nível, escritos em um padrão, com linguagens bem definidas, que são coerentes, precisos e contêm informações suficientes sobre o sistema a ser desenvolvido.
- Definições do modo como um modelo conceitual é transformado em códigos específicos para uma determinada plataforma, tecnologia ou arquitetura.
- Documentação organizada e formal de todos os conceitos do sistema.
- Pessoas treinadas na ferramenta
- Metodologia do processo de software
- Ferramentas de gestão do projeto (controle, apropriação de horas, ...)



## Conclusão

Como vimos, o Telescope auxilia muito na superação das principais dificuldades do desenvolvimento tradicional de sistema. Entre outras vantagens, podemos citar:

- Produtividade
- Padronização
- Documentação
- Confiabilidade
- Preservação do conhecimento independente de pessoas
- Independência tecnológica
- Portabilidade