1
Introdução

1.1.
Motivação

Um sistema de software está sempre evoluindo para se adequar a modificações relacionadas ao negócio de organizações. Após diversas evoluções, tanto o software quanto a sua documentação, quando existentes, tornam-se difíceis de entender. Quanto mais difícil é entender um sistema, mais os desenvolvedores focam na modificação do software, diminuindo a confiabilidade de documentação existente, dificultando mais ainda a tarefa de manutenção. Além da manutenção, a evolução torna-se uma tarefa complexa. Isso acarreta, no final, um aumento de custo da produção ou manutenção de artefatos produzidos a partir do software original, seja este aumento proveniente de um atraso frente a estimativas sobre algo que não se conhece, ou mesmo da necessidade de adicionar mais recursos humanos ao projeto, devido à alta complexidade que acarreta a baixa produtividade do grupo.

Para se efetuar uma manutenção em um software, é importante que o desenvolvedor faça uma análise de impacto da mudança [Sommerville 2007]. Entende-se análise de impacto como sendo a identificação das potenciais conseqüências de uma alteração e a ação de estimar o que se deve ser alterado para realizar esta modificação. Uma análise de impacto envolve verificar se outras partes do software devem ser modificadas por conta de uma manutenção, buscando minimizar os efeitos colaterais de uma mudança [Wilde and Huitt 1992; Queille et al. 1994].

Para se fazer uma análise de impacto é necessário ter uma representação da estrutura do sistema [Queille et al. 1994]. Porém, a documentação existente sobre um sistema legado tende a perder a confiabilidade ao longo do tempo. Para contornar este problema, pode-se recuperar a estrutura do sistema utilizando engenharia reversa, que se preocupa em recuperar a representação arquitetural do sistema a partir do código fonte, permitindo seu entendimento e modernização [Chikosfky and Cross 1990].

Qual a dificuldade em se encontrar uma ferramenta para realizar tal tarefa? Imagine que se trata de uma linguagem antiga, ou de um sistema com pelo menos cinco anos de existência e em evolução constante, e que teve uma alta rotatividade de desenvolvedores. Como fazer uma análise de impacto confiável? Além disso, como gerar automaticamente modelos coerentes com o que está implementado? Esta dissertação visa abordar parte deste problema.

1.2. Definição do Problema

O Laboratório de Engenharia de Software da PUC-Rio (LES) conduz vários projetos na área de engenharia reversa. Um, em especial, foi o motivador do trabalho proposto neste documento. O projeto envolve a documentação e teste de um software (produzido por uma organização externa ao LES) controlador de estoque para os produtos produzidos pela Petrobras, como óleos e Gás Natural Veicular (GNV). Além de um aplicativo, ele pode ser considerado um ambiente de integração para diversos outros artefatos, pois disponibiliza uma interface de comunicação que permite a outros produtos, produzidos por qualquer fábrica de software, consultar e alterar dados de sua base.

Como este software foi iniciado há mais de sete anos, não existia a facilidade atual para criação de serviços distribuídos, como a tecnologia de *web service*. A solução adotada foi o uso de *procedures* no SGBD Oracle. Toda a inteligência do modelo de negócios ficou implementada no banco de dados. Na interface foi definida uma camada simples, que chama as *procedures* e apresenta o resultado retornado.

A meta principal do projeto desenvolvido pelo LES era gerar documentações e casos de testes para essas *procedures*, utilizando técnicas de engenharia reversa. O objetivo primário dessa documentação era facilitar o entendimento do banco de dados para fins de manutenção. Então, duas frentes de trabalho foram iniciadas: de criação de testes e de documentação.

No caso dos testes, tecnologicamente não houve problemas para o seu desenvolvimento. Foram utilizados *frameworks* em Java que possibilitaram a criação de testes de unidades para as *procedures*. O grande problema encontrado era o entendimento do ambiente necessário para a modificação ou execução das *procedures*. Elas possuem uma árvore de chamadas complexas, além de usar tabelas temporárias como meio de comunicação entre si. Isso adicionou um grau extra de complexidade, pois, para cada *procedure* a ser

executada, todas as tabelas temporárias das quais elas dependem deveriam estar corretamente preenchidas. Com o passar do tempo, percebeu-se que as dúvidas que surgiam estavam relacionadas com a necessidade de documentação do sistema.

Para gerar a documentação, foi feita uma análise do que já havia sido documentado e como poderia ser o processo de atualização destes artefatos. Uma categoria de documentos produzidos, porém defasados, chamou a atenção pela capacidade de atender as necessidades dos desenvolvedores e dos testadores. Eram documentos em Word que continham o código das *procedures*. Para cada trecho de código fonte, havia comentários feitos em caixas de texto, como por exemplo: "*Popula com parâmetros: temp_cenarios_rel, temp_produtos_rel*". As dependências e chamadas às outras *procedures*, ou informações desse gênero, eram descritas nesses comentários. Na figura abaixo, é mostrado um trecho do documento original.

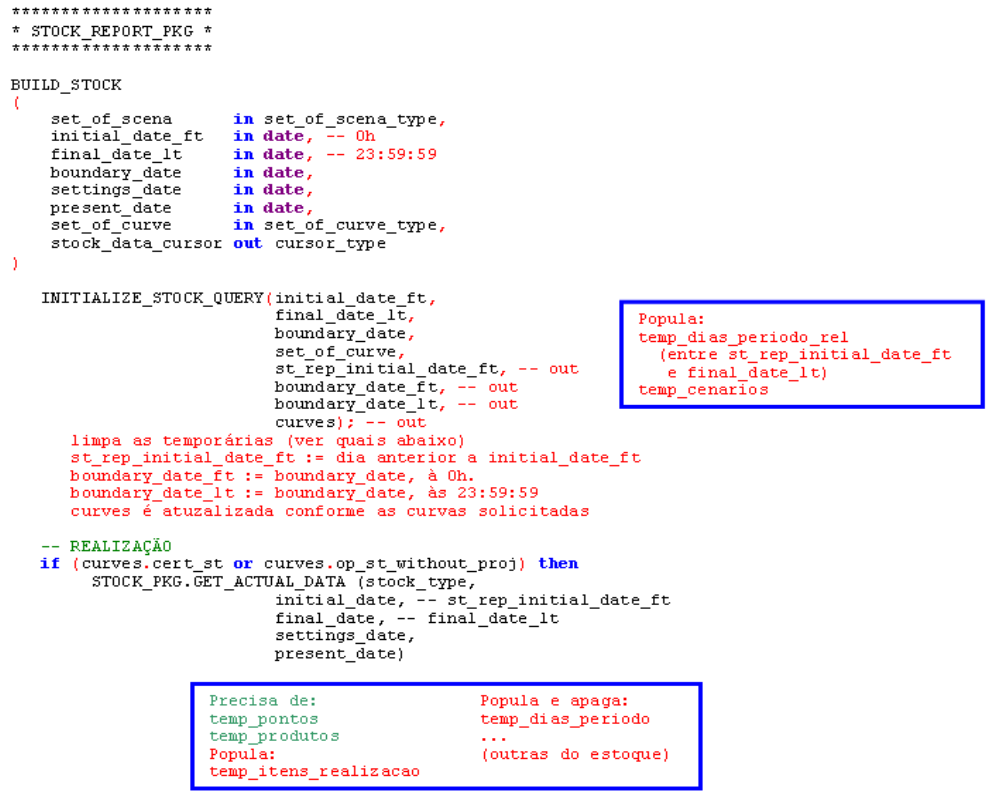

Figura 1: Documento original produzido pelo desenvolvedor

O documento acima quase sempre se encontrava desatualizado, devido ao esforço constante necessário para a manutenção do sistema. Apesar disto, era muito consultado, principalmente por desenvolvedores recém contratados e pelos analistas de testes, que precisavam entender o funcionamento da árvore de chamadas das *procedures* para poder produzir testes eficazes.

O fato acima levantou um questionamento: será que não haveria um diagrama que fosse capaz de informar visualmente o que estava descrito em forma de texto no documento? Além disto, não haveria uma ferramenta capaz de gerá-lo automaticamente?

1.3. Objetivo e Organização do Trabalho

O objetivo inicial desta dissertação era produzir uma ferramenta capaz de gerar uma documentação que satisfizesse as necessidades expostas na seção anterior. São elas: análise de impacto para fins de manutenção; o entendimento do workflow de chamadas das *procedures* para auxiliar o desenvolvimento de testes, manutenção e evolução do sistema.

Com a evolução dos estudos sobre o problema exposto na seção 1.2, o objetivo desta dissertação também evoluiu, pois foi percebida a possibilidade de fazer mais do que simplesmente atender ao problema apresentado. O objetivo se transformou em: prover uma base para geração de diagramas comportamentais de sistemas escritos em qualquer linguagem, além de fornecer um arcabouço de visualização de diagramas baseado em metadados, que minimiza o trabalho de quem deseja desenvolver um artefato para realizar engenharia reversa. Além disso, ela tem como meta documentar a experiência e o conhecimento adquirido de um estudo de caso que partiu de um grande volume de código escrito em uma linguagem de difícil reengenharia para um gerador automático de *workflow* de chamadas.

Foi feito um estudo relativo aos trabalhos existentes na literatura e no mercado que pudessem auxiliar na construção de um artefato que consiga atender o objetivo acima. Este estudo é apresentado no capítulo 2. Ele mostrou a carência de uma ferramenta que conseguisse suprir estas necessidades, não apenas para a linguagem *PLSQL*¸ mas para diversas outras que, por estarem em desuso, ou por terem poucos adeptos, possuem escassos instrumentos para engenharia reversa de sistemas nelas escritos. No capítulo 3 é apresentada uma ferramenta que tem como meta minimizar esse problema. Devido ao fato de ser orientada a metadados, esta ferramenta precisou de uma *API* gráfica com a mesma característica, para que seja possível a geração de diagrama. Devido à complexidade das *APIs* existentes foi desenvolvido um *framework* de visualização gráfica discutido no capítulo 4. No capítulo 5 é apresentado o estudo de caso que motivou o surgimento desta dissertação, e como ele fez uso

dos artefatos desenvolvidos por ela. Além disso, é feita uma avaliação do resultado produzido em comparação com as necessidades práticas do estudo de caso. Por último, no capítulo 6, a conclusão da dissertação, apresenta o comentário final e os trabalhos futuros.