Migração de software: considerações e experiências

Fernando Henrique Canto
Centro de Processamento de Dados
Universidade Federal do Rio Grande do Sul
fernando@cpd.ufrgs.br

Resumo

Este artigo relata brevemente os resultados de algumas experiencias vividas pelas equipes de desenvolvimento de software do CPD-UFRGS na migração de sistemas desenvolvidos em linguagens e plataformas legadas para plataformas mais recentes nos últimos anos, os principais problemas observados nesse tipo de processo e algumas recomendações de melhorias e soluções para esses problemas. O artigo analisa principalmente problemas que ocorrem quando o processo de migração se baseia no código-fonte legado, e não em uma especificação técnica de alto nível, forçando que desenvolvedores realizem um trabalho de engenharia reversa e tradução de código, ao invés de utilizarem os recursos e ferramentas da linguagem da melhor forma possível. O objetivo do trabalho é evidenciar que desenvolver o sistema novamente, a partir de uma especificação adequada, ao invés de uma tradução direta entre linguagens, reduz consideravelmente a incidência de problemas e o tempo de manutenção, economizando tempo e esforço das equipes de desenvolvimento envolvidas. Dessa forma, o processo de migração tende a agregar benefícios suficientes para justificar o tempo investido nessa atividade.

Palavras-chave: migração de software, desenvolvimento de software, engenharia de software, frameworks, sistemas legados

Abstract

This paper briefly tells the results of some experiences encountered by the software development teams in the CPD-UFRGS in the modernization old systems and legacy code to more modern software platforms in the recent years, the problems noticed in that kind of process, as well as some recommendations of improvements and solutions for those problems. The paper analyses mostly the problems that happen when the modernization process is based on the legacy code itself, instead of a high level technical specification, forcing programmers to work with reverse engineering and translation between different language, instead of employing the resources and functionalities of the language as effectively as possible. The goal of this work is to show that developing the system from the ground up, based on adequate specification, instead of doing a direct translation, significantly reduces the incidence of bugs and maintenance time, saving time and effort from the development teams. This way, the modernization process tends to bring sufficient benefits to justify the time invested in such activity.

Keywords: software modernization, software development, software engineering, frameworks, legacy code.

1. Introdução

Ao longo dos últimos anos no desenvolvimento de software no Centro de Processamento de Dados, fez-se necessário migrar sistemas antigos para novas linguagens e plataformas de desenvolvimento. A experiência prática nesses projetos e a pesquisa por técnicas e metodologias fez surgir uma certa base de conhecimento sobre o processo, a qual será exposta neste trabalho.

2. Escopo do trabalho

Os exemplos recolhidos para compor este trabalho contemplam as migrações realizadas aproximadamente nos últimos dez anos. As aplicações mais antigas ainda em uso nesse período foram desenvolvidas na plataforma Delphi, da Borland, usando a linguagem Object Pascal, com algumas sendo desenvolvidas também na plataforma PowerBuilder, da Sybase.

Após isso, foram feitas as primeiras aplicações baseadas na Web, na linguagem ASP, da Microsoft. Pouco depois, a linguagem PHP foi adotada como linguagem institucional de sistemas Web na UFRGS, e as aplicações existentes em ASP foram migradas para a nova linguagem, a fim de descontinuar o serviço ASP.

Posteriormente, o CPD selecionou o Yii Framework como plataforma padrão de desenvolvimento para aplicações PHP, com a criação de uma biblioteca de componentes e classes compartilhadas pelos diversos sistemas. Foi decidido que todos os sistemas novos seriam desenvolvidos nessa plataforma, com aplicações legadas sendo gradualmente migradas. Nesse ínterim, algumas aplicações já foram totalmente ou parcialmente migradas.

3. O problema da migração como tradução de código

Um dos aspectos que afetam o processo de migração é a carência, ou a ausência, de documentação técnica dos sistemas legados. Muitas vezes, pensa-se que a existência do código-fonte do sistema torna redundante a existência de uma documentação, e que migrar o sistema para outra plataforma se trata de uma "tradução" de código; isto é, telas ou rotinas inteiras são lidas em sua linguagem original, e reescritas na nova linguagem. Dessa forma, não seria necessário especificar o funcionamento do sistema em uma modelagem de alto nível, pois ele "já está pronto", e a migração depende de um trabalho de engenharia reversa por parte dos desenvolvedores.

Esse tipo de prática tende a gerar problemas quando malconduzido. O principal é que a migração pode acabar acumulando vícios e más práticas presentes no código anterior com o uso ineficaz das ferramentas da nova linguagem, aumentando assim a quantidade de problemas em potencial. Isso tende a acontecer principalmente quando o sistema é migrado em blocos, por exemplo, uma interface de cada vez, sem haver primeiro um estudo de suas funcionalidades fundamentais. Dessa forma, trechos de código repetidos acabam mantendo-se, quando não se proliferam. Por exemplo, na migração de um código puramente procedural para uma plataforma orientada a objetos, ao invés da equipe de desenvolvimento explorar o potencial de reuso de código e modularidade, com o uso de conceitos como herança e polimorfismo, bem como padrões de desenvolvimento, o novo código torna-se um híbrido sem sentido de paradigmas.

Citando um exemplo prático, um dos recursos mais úteis trazidos pelo Yii Framework, assim como diversos outros *frameworks* de desenvolvimento Web, é o mapeamento objeto-relacional (ORM, na sigla em inglês), que permite que todo o acesso a bancos de dados relacionais seja feito sem o uso de consultas SQL, e com recursos adicionais próprios de linguagens orientadas a objetos. O Yii Framework especificamente inclui uma implementação própria do padrão *active record*, proposto por Martin Fowler. É quase sempre útil e recomendável escrever comandos de acesso a banco utilizando o *active record* do Yii Framwork, pois isso permite o uso de recursos úteis como geração de formulários, validação automática, geração de *grid views* e assim por diante.

Porém, nem sempre é fácil "traduzir" uma consulta SQL para seu respectivo comando com *active record*, portanto muitos desenvolvedores acabam copiando a consulta original para dentro do sistema novo. Isso obriga que certas funcionalidades do sistema sejam implementados manualmente, sem o uso dos recursos que o *framework* oferece, ou utilizando recursos menos práticos. Isso tende a acontecer quando se entrega o projeto a um desenvolvedor pouco experiente, ou com prazos restritos.

Esse tipo de migração pode também gerar problemas mesmo havendo um esforço ativo para utilizar os recursos da nova plataforma da melhor maneira. Nem sempre é possível compreender adequadamente o funcionamento de um sistema lendo código-fonte legado, principalmente aquele que foi submetido a repetidos ciclos de manutenção e correção. Trechos de código podem tornar-se confusos, e obscurecer o verdadeiro significado do código que está sendo lido. Em outro exemplo prático, encontrou-se em um sistema recentemente migrado para o Yii Framework, um trecho de código que realizava duas consultas diferentes, uma após a outra, para recuperar o mesmo registro. Isso é um provável sintoma da falta de familiaridade com as regras de negócio do sistema original, ou ainda uma tentativa de corrigir um erro anterior sem compreender o que de fato está sendo feito.

4. O problema da migração somada à mudança

Muitas vezes, existe a intenção de aproveitar a migração do software para realizar, ao mesmo tempo, uma melhoria na funcionalidade, ou na interface; por exemplo, implantar novos padrões de navegação, novos recursos para o usuário, interfaces responsivas, etc.

Em muitos casos, mudanças como essa são inevitáveis, quando os recursos usados anteriormente são defasados ou incompatíveis com a nova plataforma. Assim, nem sempre se trata em satisfazer demandas vindas do usuário. No caso dos sistemas da UFRGS, após a adoção do Yii Framework, desenvolveu-se um estilo visual padrão para todos os sistemas. Dessa forma, sistemas legados, ao serem migrados para esta plataforma, devem ser adequados ao novo padrão, o que muitas vezes exige uma adaptação das interfaces.

O problema que isso acarreta é que, em casos como este, o projeto deixa de ser apenas a migração de uma plataforma para a outra, e passa a ser a construção de uma versão totalmente nova de um sistema, as prioridades se tornam diferentes, e os desafios aumentam. Não se pode mais apenas comparar o novo sistema com sua versão antiga quando a forma de interação do usuário e a maneira como o sistema reage. Ao invés disso, o novo sistema deve ser avaliado em função de seus requisitos básicos e das necessidades dos usuários, sendo que essas necessidades devem ser também adaptadas às novas tecnologias utilizadas. Realizar essa adaptação é muito mais simples em uma modelagem de alto nível do que em código-fonte legado.

Um desses casos ocorreu no CPD, na migração de um antigo sistema desenvolvido em ASP para a linguagem PHP. Neste processo, foi decidido mudar radicalmente a interface do sistema, com ampla utilização de AJAX para a navegação entre as diferentes telas. Esse tipo de projeto exige uma completa reavaliação tanto da maneira como o sistema é implementado internamente, quanto da forma como o usuário interage com o sistema como um todo e com cada tela individualmente. Uma análise deficiente desse tipo de mudança pode resultar em um sistema de difícil manutenção, com código legado inserido inadequadamente em um paradigma para o qual ele não foi concebido.

Nesse caso, o resultado foi um sistema cujas regras de negócio, lógica de validação e navegação estão fragmentadas entre códigos de diferentes linguagens, somados a práticas de desenvolvimento defasadas, que foram mantidas no novo sistema. Por consequência disso, correções e alterações nesse sistema tendem a ser difíceis e custosas, e com menos garantia de sucesso

5. Sugestões de melhoria de processo

O primeiro aspecto importante em um projeto de migração é a consciência de que trata-se, antes de tudo, do desenvolvimento de um novo sistema. Portanto, o projeto deve ser realizado com base em qualquer especificação técnica disponível, e não no código-fonte original. No caso da inexistência de especificação, deve-se investir esforço na compreensão do sistema e de suas regras,

e na elaboração de uma especificação adequada para a nova plataforma onde o sistema será reimplementado.

O objetivo é permitir que os desenvolvedores fiquem focados apenas em escrever código-fonte na nova plataforma, partindo de uma compreensão adequada do sistema, evitando assim repetir falhas e vícios ruins de linguagem. Quanto menos contato os desenvolvedores tiverem com o código-fonte original, melhor.

Dentro os projetos realizados pelo CPD, encontram-se vários casos de migração bem sucedida, e praticamente todos eles passam por um processo semelhante a esse. De fato, os casos problemáticos de migração devem-se em grande parte à dependência do código-fonte legado. E para resolver esse problema, nem sempre é necessário uma documentação extremamente sofisticada e extensa, e sim definições de tarefas claras e suficientes para que os desenvolvedores realizem seu trabalho.

Além do mais, deve-se projetar sempre uma plataforma de desenvolvimento que permita a melhoria contínua do sistema. A experiência prática mostrou que apenas utilizar uma ferramenta pronta, como o Yii Framework, não é garantia absoluta de sucesso. Ao invés disso, os recursos da ferramenta devem ser aplicados da melhor forma possível, e isso só acontece quando a equipe de desenvolvimento compreende o sistema em função de suas funcionalidades e regras, e não da maneira como ele foi previamente implementado. Com a utilização correta dos recursos, é possível dar uma garantia maior de qualidade aos sistemas utilizando, por exemplo, ciclos de refatoração de código, automatização de testes, integração contínua e assim por diante.

6. Conclusão

Assim como qualquer tipo de projeto, nem sempre é possível prever de antemão os riscos e problemas que serão encontrados, e os defeitos na metodologia utilizada antes que ela seja posta em prática. Mesmo assim, é sempre importante adquirir aprendizado com as falhas para evitá-las no futuro, e compartilhar o aprendizado sempre que possível.

Migrações de sistemas, em muitos casos, são inevitáveis, pelo fato de que tecnologias antigas tornam-se defasadas e podem até impedir a evolução da infraestrutura de sistemas. Fora isso, novas tecnologias trazem novos recursos e benefícios, que podem ser empregados na melhoria dos serviços oferecidos, e prolongar a vida útil dos sistemas desenvolvidos.

Referências

GARDNER, D. *Not just a nip and tuck, application modernization extends the lifecycle of legacy code assets.* 24/10/2006. Disponível em <http://www.zdnet.com/article/not-just-a-nip-and-tuck-application-modernization-extends-the-lifecycle-of-legacy-code-assets/>. Acesso em 26/03/2016.

FOWLER, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003.

FOWLER, M. *Refactoring. Improving the Design of Existing Code.* Addison Wesley Longman, 1999.

The Definitive Guide to Yii. Disponível em <http://www.yiiframework.com/doc/guide/>