Questões importantes na implementação de software

Important issues in the software implementation

Lucio Geronimo Valentin, Maria Madalena Dias, Roberto Carlos Santos Pacheco

1 UTFPR - Universidade Tecnológica Federal do Paraná, Campus Campo Mourão. E-mail: lgvalentin@utfpr.edu.br
2 UEM - Universidade Estadual de Maringá, Departamento de Informática. E-mail: mmdias@din.uem.br
3 UFSC – Universidade Federal de Santa Catarina, Departamento de Engenharia do Conhecimento. E-mail: pacheco@egc.ufsc.br

RESUMO

Um software precisa ser considerado de boa qualidade para ser útil e fornecer suporte adequado às atividades a que se propõe. Isto implica que o mesmo deve ter sido bem concebido, bem projetado e bem implementado. A implementação envolve questões importantes que precisam ser tratadas e definidas já na etapa de projeto. Atualmente, existem diferentes soluções, disponíveis no mercado, para facilitar e tornar mais eficiente a realização das atividades que fazem parte da etapa de implementação. Assim, neste artigo, é apresentada uma discussão sobre questões que devem ser tratadas e resolvidas na etapa de projeto de um software, por serem imprescindíveis para sua implementação.

Palavras-chave: Questões de implementação. Boas práticas na implementação. Soluções para implementação.

ABSTRACT

Software needs to be considered of good quality to be useful and provide appropriate support to the activities as proposed, which implies to have been well conceived, well designed and well implemented. Implementation involves important issues that need to be addressed and defined already in the stage of design. Currently, there are different solutions, available in the market, to facilitate and make more efficient implementation of activities that are part of the stage of implementation. Therefore, this paper presents a discussion on issues to be addressed and resolved in the stage of software design, because they are essential to its implementation.

Keywords: Questions of implementation. Good practice in implementation. Solutions for implementation.

INTRODUÇÃO

A implementação demanda grande parte do tempo no processo de desenvolvimento de um software, por ser uma das atividades mais trabalhosas e exigir grandes habilidades do profissional da área de informática. Assim, antes de se iniciar a etapa de implementação de um software, é necessário escolher o ambiente de programação e tratar outras questões que possam influenciar direta ou indiretamente no bom desempenho desta atividade.

Além da escolha do ambiente de programação, existem boas práticas a serem seguidas para facilitar, principalmente, a manutenção do software e, ainda, alguns problemas a serem solucionados relativos à documentação, às rotinas de teste, à integração da equipe de desenvolvimento e à composição de arquivos de configuração da aplicação. No caso de um ambiente orientado a objetos, outros problemas surgem, como, por exemplo, controle de instâncias e relacionamentos entre objetos e persistência de objetos.

Assim, o objetivo deste artigo é apresentar questões importantes que precisam ser consideradas durante o projeto de um software para facilitar a realização da etapa de implementação. Como base para essa discussão, é considerada a implementação de um framework para arquitetura de software proposta por Valentin et al. (2008). Além disso, é apresentada uma aplicação prática das questões abordadas.

QUESTÕES NO DESENVOLVIMENTO DE SOFTWARE

A primeira questão tratada diz respeito à linguagem que seria utilizada para a implementação. Considerando o caso do software ser integrado em ambientes corporativos[1], a primeira decisão é escolher uma tecnologia que ofereça suporte ao desenvolvimento de sistemas corporativos. Foram analisadas as plataformas .Net[2] da Microsoft e a J2EE da Sun. A segunda pode ser definida como sendo a mais indicada por ser um software gratuito e manter um bom relacionamento com a comunidade de software livre[3], além de contar com inúmeras ferramentas gratuitas.

Além da questão da linguagem, a seguir são descritas outras questões que precisam ser abordadas.

MELHORES PRÁTICAS EM DESENVOLVIMENTO DE SOFTWARE

Broemmer (2003) apresenta práticas de desenvolvimento que durante anos têm sido comprovadamente as melhores. Seu foco é na plataforma J2EE, no entanto, as questões abordadas são perfeitamente praticadas em qualquer plataforma ou metodologia de desenvolvimento de software. Aqui são consideradas três práticas que desempenham um importante papel na especificação de um framework.

Tratamento padronizado das mensagens geradas pelos elementos da arquitetura

Durante a execução de uma atividade, alguns erros podem ocorrer. O usuário que iniciou a atividade precisa ser informado sobre o que deu errado. A exibição de uma pilha de execução que mostra os procedimentos que foram interrompidos não será esclarecedora para o usuário. É necessário que o sistema informe, de uma maneira sistemática, quais atividades de negócio foram interrompidas e como o usuário pode proceder. O recurso de tratamento de exceções das linguagens é um avançado mecanismo que auxilia neste controle de erros e de mensagens. No entanto, ele é bastante técnico e é focado em tratar exceções das rotinas do software. É necessário estendê-lo para criar um mecanismo capaz de controlar exceções de negócio.

Erros de variáveis não inicializadas, de tipos incompatíveis de dados, de índice inválido de vetor, entre outros, são erros da linguagem que devem ser separados dos erros de negócio, que seriam: erro ao iniciar um processo; erro ao executar um serviço; erro ao registrar a movimentação na conta; erro de saldo insuficiente para a operação. Geralmente, erros de linguagem revelam bugs da aplicação, enquanto que erros de negócio revelam inconsistência nos dados da aplicação ou dos parâmetros fornecidos para algum processo.

Estendendo o mecanismo de exceções, pode ser criada uma estrutura de mensagens para que cada serviço, processo, visualização ou componente de apoio possa tratar as mensagens de negócio de uma maneira padronizada. Os erros de linguagem são convertidos para esta estrutura. Este mecanismo também é utilizado para tratar mensagens de informações que devem ser exibidas para o usuário.

[1] Ambiente corporativo é referido aqui ao ambiente de instituições que utilizam diversas soluções de software e que integram seus processos de negócio com os seus sistemas de informação. Muitas vezes este ambiente é heterogêneo, com a presença de diversas tecnologias.
[2] Maiores informações sobre a plataforma .NET da Microsoft podem ser conseguidas no sítio http://microsoft.com/net/.
[3] Uma referência de software livre no país é o portal do próprio governo (http://www.softwarelivre.gov.br).

Mensagens armazenadas fora do código, em repositório de mensagens

Os textos das mensagens são mais voláteis que o código da aplicação. Durante o período de implantação, as mensagens tendem a ser alteradas para serem mais bem compreendidas pelo usuário. Uma boa prática é armazenar estas mensagens fora do código fonte da aplicação. Isto permite que os textos sejam alterados sem que a aplicação seja compilada novamente. Além disso, os erros de negócio mostrados para o usuário devem estar na linguagem utilizada pelo usuário. Isto implica que o mecanismo de mensagem precisa armazenar os textos das mensagens em diversas linguagens, o que contribui para a utilização de um repositório que armazena essas mensagens.

Metadados sobre as entidades em arquivos separados

É comum que os programadores implementem rotinas de validação das propriedades das entidades dentro do código da aplicação, como por exemplo: valor máximo, mínimo, valores válidos, entre outras. No entanto, algumas dessas rotinas podem ser automatizadas utilizando um repositório de metadados para descrição das validações que devem ser aplicadas às propriedades das entidades. Isto permite que as validações sejam alteradas sem mudança nas linhas de código da aplicação. Além disso, centraliza a definição das validações utilizadas pela aplicação, evitando a redundância de código e facilitando a manutenção do software.

PROBLEMA DE CONTROLE DE INSTÂNCIAS DE OBJETOS E RELACIONAMENTO ENTRE OS OBJETOS CRIADOS

Uma arquitetura deve manter um controle sobre os objetos por ela instanciados de forma que ela possa definir o relacionamento entre os objetos de uma maneira mais automática. Além disso, é necessário que muitos objetos criados pela arquitetura recebam referências da própria arquitetura em que ele está sendo instanciado. Esta questão é resolvida pelo padrão de projeto chamado fábrica de objetos (Metsker, 2002). Assim, pode ser utilizado um framework de integração chamado Spring[4].

O Spring é um framework de código aberto que foi desenvolvido com o objetivo principal de facilitar a implementação de aplicações empresariais. Sua principal característica é uma fábrica de instâncias de objetos. Esta fábrica possui um mecanismo de IoC (Inversion of Control) que, ao instanciar um novo objeto, verifica os relacionamentos do objeto, instancia outros objetos necessários e faz a ligação entre os objetos. Isto evita que o programador fique controlando os objetos instanciados e os relacionamentos entre eles.

É comum o desenvolvedor de software querer desenvolver seu próprio framework de controle de instâncias de objetos. No entanto, a utilização de um framework como o Spring pode oferecer inúmeros outros benefícios para o projeto, como por exemplo:

- Possibilidade de integração de outras ferramentas como o Ant[5], JSP[6], Hibernate[7] e xDoclet[8]. Isto seria, no mínimo, bastante trabalhoso de se obter com uma fábrica personalizada de objetos;

[4] http://www.springframework.org/
[5] http://ant.apache.org/
[6] http://java.sun.com/products/jsp/
[7] http://www.hibernate.org/
[8] http://xdoclet.sourceforge.net/

- Mecanismo de Programação Orientada a Aspectos (AOP[9], do inglês *Aspect Oriented Programming*). Por ser uma fábrica de objetos, o Spring tem total controle sobre o objeto instanciado e, com isto, é possível definir a injeção de aspectos em tempo de execução da aplicação utilizando a configuração da fábrica;
- Arquivo de configuração centralizado. O Spring utiliza um arquivo XML que descreve a aplicação que ele vai gerenciar. Neste arquivo são definidas as classes, os métodos de instanciação e os relacionamentos entre os objetos instanciados. Vários parâmetros da aplicação podem ser alterados simplesmente editando este arquivo, sem a necessidade de re-compilar a aplicação.

PROBLEMA DE PERSISTÊNCIA DE OBJETOS

A persistência refere-se ao armazenamento não-volátil dos dados, ou seja, uma vez aceitos pelo gerenciador de banco de dados, os dados são mantidos em um dispositivo físico de armazenamento e só podem ser removidos por alguma requisição explícita a esse gerenciador. Na orientação a objetos, a persistência de objetos diz respeito à existência dos objetos mesmo após o término da execução do programa.

O paradigma da orientação a objetos não apresenta uma solução simples para a persistência, raramente existe disponível um banco de dados orientado a objetos e, geralmente, um banco de dados relacional é utilizado para armazenar as características do objeto. Assim, surgem problemas na persistência de objetos.

O problema de armazenamento de objetos em estruturas relacionais já foi bastante pesquisado e apresenta algumas soluções satisfatórias. Uma delas é o framework Hibernate.

O Hibernate é um framework de persistência de objetos sobre bancos de dados relacionais que realiza esta atividade de maneira transparente. É considerado um dos maiores projetos de código aberto desenvolvido em Java. As principais vantagens da utilização do Hibernate em um projeto são (Bauer e King, 2005):

- Transparência do mapeamento Orientado a Objetos vs. Relacional: os objetos e as coleções de objetos são vistos sem a preocupação de referência de esquemas, tabelas e itens de dado do banco de dados. Isto permite que o programador concentre seus esforços na aplicação dos conceitos orientados a objetos nas entidades e nos seus relacionamentos. Utilizando o Hibernate para o controle de persistência, o programador não necessita implementar classes que realizam as operações de inserção, alteração e remoção de uma determinada entidade de negócio.
- Portabilidade de banco de dados: Utilizando o Hibernate, todas as classes da aplicação são mapeadas pelo framework. Este mapeamento é independente do banco de dados a ser utilizado e é responsabilidade do Hibernate realizar as adaptações e traduções do mapeamento para instruções SQL compatíveis a cada sistema gerenciador de banco de dados disponível no mercado.
- Linguagem de consulta de objetos: Outra característica muito importante do Hibernate é fornecer uma linguagem de consulta bastante parecida com a SQL, a HQL (Hibernate Query Language). A HQL permite realizar consulta de objetos persistidos utilizando os conceitos orientados a objetos. Esta linguagem é bastante flexível e suas consultas apresentam um grau de compreensão maior do que a mesma consulta escrita em SQL. Isto porque as relações entre as classes de objetos ficam transparentes para quem escreve a consulta. Por ser bastante semelhante à SQL, a HQL é de fácil aprendizado.

[9] http://aosd.net/

PROBLEMA DE DOCUMENTAÇÃO

A questão aqui tratada é a documentação das interfaces e códigos desenvolvidos. Fazendo a pergunta: quem é que gosta de documentar o que implementa, em uma sala de aula de bacharelandos em informática ou ciência da computação é possível notar que a documentação do software pode se tornar um problema se não abordada logo no início do projeto.

Durante o desenvolvimento de uma rotina, a atenção do programador está voltada à resolução do problema. A documentação geralmente é deixada para um segundo momento, que às vezes não chega nunca. Para auxiliar nesta questão, a integração da documentação com o próprio código é uma proposta que evita que o programador tenha que acessar outra ferramenta para documentar o que está sendo implementado. Segundo Pamplona (2006), a linguagem Java inventou o conceito de comentário de documentação. Este comentário é específico para quem precisa saber o que o código fonte faz sem ver o código, ou seja, é um comentário para documentos. Este padrão de documentação é chamado de JavaDoc[10].

PROBLEMA DE TESTES

Murphy (2005) destaca a importância de estar definindo testes logo no início do processo de desenvolvimento de um software. Ele mostra que é indispensável que cada funcionalidade do sistema seja testada antes de sua integração com os demais elementos da aplicação. As questões de teste abordadas contribuem para a implementação de classes que auxiliam na realização de testes em funcionalidades que se integrarão à arquitetura.

Para a implementação das classes básicas para teste pode ser utilizado o framework de teste unitário JUnit[11]. Com este framework, é possível construir classes de testes que são instanciadas e executadas para automatizar as atividades de teste.

PROBLEMA DE INTEGRAÇÃO

O desenvolvimento de um sistema de grande porte pode envolver diferentes equipes trabalhando em paralelo. Assim, para a sua integração, é necessário o uso de uma ferramenta de controle de versão concorrente (CVS, do inglês Concurrent Version System).

O Eclipse[12] pode ser o ambiente escolhido pelo fato de possuir uma interface ágil, inúmeros recursos que facilitam a produção de software (assistentes e modelos) e consumir menos recurso computacional do equipamento (é mais leve). Nesse ambiente, o sistema de controle de versão já é integrado. Sendo necessário somente configurar um servidor do repositório central.

PROBLEMA DE COMPOSIÇÃO DE ARQUIVOS DE CONFIGURAÇÃO DA APLICAÇÃO

Quando se trata de um ambiente de desenvolvimento que envolve vários frameworks, a atividade de definir os arquivos de configuração desses frameworks, para manipular os componentes da arquitetura, pode se tornar uma tarefa bastante extensa. Esses frameworks são

[10] http://java.sun.com/j2se/javadoc/
[11] http://www.junit.org/
[12] http://www.eclipse.org/

configurados por meio de arquivos XML. A alteração de uma funcionalidade ou a criação de uma nova funcionalidade exige que os arquivos de configuração dos frameworks sejam atualizados. Para agilizar essas atualizações, pode ser usada a ferramenta XDoclet[13].

XDoclet é um framework que possibilita programação orientada a atributos. Utilizando os comentários de documentação JavaDoc, é possível adicionar metadados no código-fonte Java. A ferramenta analisa gramaticamente o código-fonte e gera os metadados em arquivos XML. Esses arquivos são utilizados como arquivo de configuração por outras ferramentas como Spring, Hibernate e Java Faces. Desta forma, a manutenção das configurações é facilitada por que tudo se encontra dentro de um mesmo arquivo, o código-fonte.

Para acionar as tarefas da ferramenta XDoclet, é usada uma ferramenta de compilação chamada Ant[14]. Esta ferramenta permite a criação de arquivos de configuração XML que descrevem todo o processo de compilação e implantação de uma aplicação. Seu principal objetivo é substituir os complicados comandos de console que são necessários para compilar, configurar, copiar e integrar uma aplicação. Estas atividades são abstraídas em tarefas que podem ser facilmente configuradas e executadas.

APLICAÇÃO PRÁTICA DAS QUESTÕES ABORDADAS

Todas essas questões respondidas formaram uma base sólida para a implementação de um framework que foi desenvolvido para validação da arquitetura de software proposta por Valentin et al. (2008). Essa arquitetura faz parte do projeto de um sistema de descoberta de conhecimento em banco de dados (Knowledge Discovery in Database – KDD), mas que pode ser utilizada para outros domínios.

A Figura 1 mostra a área de trabalho do ambiente de desenvolvimento do framework e a integração das ferramentas em um único ambiente. A seguir é descrito cada item que foi destacado na figura.

[13] http://xdoclet.sourceforge.net
[14] http://ant.apache.org

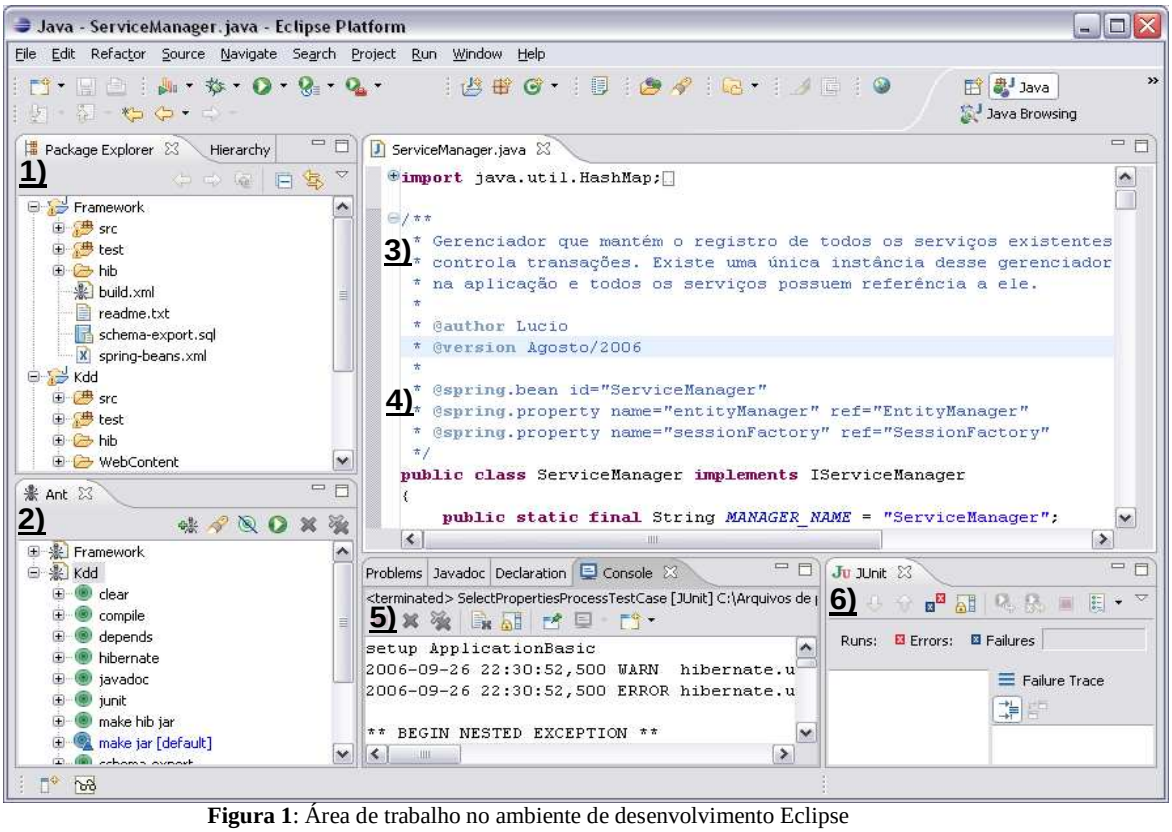

Figura 1: Área de trabalho no ambiente de desenvolvimento Eclipse

O item 1 da Figura 1 destaca a estrutura dos projetos framework e Kdd. Como pode ser visto, cada projeto possui especificações separadas e é implementado independentemente. Porém, todos seguem uma estrutura básica de diretórios que foi definida para melhor organizar os artefatos dos projetos. A seguir são descritos os diretórios e os arquivos em destaque no item 1:

- *src*: diretório onde os pacotes e as classes Java são armazenados;
- *test*: diretório onde os pacotes e as classes que implementam os testes unitários são armazenados. Nesta pasta é seguida a mesma hierarquia de pacotes da pasta src;
- *bin*: este diretório não aparece na figura por ser um diretório oculto, porém ele é utilizado pelo compilador para armazenar as classes compiladas;
- *hib*: este diretório armazena os arquivos de configuração do Hibernate e do Spring;
- *WebContent*: este diretório segue a estrutura J2EE que permite a implantação de uma aplicação em diversos servidores que seguem o padrão J2EE;
- *build.xml*: este é o arquivo de configuração da ferramenta de compilação Ant. O item 2 da figura mostra como este arquivo é organizado em tarefas. Cada tarefa é configurada dentro do arquivo e pode ser acionada com um duplo clique sobre o ícone da mesma;
- *readme.txt*: este arquivo descreve a estrutura de diretório definida;
- *schema-export.sql*: este arquivo é gerado pelo Hibernate, de acordo com o mapeamento dos objetos persistidos que são manipulados pelo sistema, e contém as instruções SQL que criam as tabelas, itens de dado, índices e restrições relacionais em um banco de dados;
- *spring-beans.xml*: este arquivo é utilizado pelo Spring para definir algumas configurações adicionais da aplicação.

O item 3 da Figura 1 destaca a documentação integrada ao código que posteriormente é analisada pela ferramenta JavaDoc para geração de documentos.

O item 4 destaca a inserção de metadados que são utilizados pela ferramenta XDoclet para

geração dos arquivos XML de configuração. Neste item, são mostrados metadados que definem a atual classe ServiceManager como uma unidade controlada pela ferramenta Spring.

O item 5 destaca algumas visualizações do ambiente. Em evidência está a visualização do Console onde são mostradas as mensagens de execução da aplicação, entre outras funcionalidades.

O item 6 da Figura 1 destaca a ferramenta de teste unitário JUnit. Esta ferramenta executa os testes e indica quais foram executados com sucesso ou com falha. Com isto, o ambiente de desenvolvimento está montado.

CONCLUSÃO

As questões no desenvolvimento de um software, apresentadas neste artigo, foram estudadas e analisadas durante o projeto de um framework, por serem consideradas imprescindíveis na sua implementação. Isto ocorre por existir, atualmente disponível no mercado, uma variedade de técnicas e ferramentas que facilitam e tornam possível a implementação de um software com mais rapidez e segurança. O resultado da escolha correta do ambiente de programação e demais ferramentas é um produto de software de boa qualidade. Não basta saber programar em uma linguagem de programação para implementar um software, é necessário, também, conhecer e aplicar boas práticas de programação e usar ferramentas disponíveis para tornar esta atividade mais eficiente e eficaz.

A aplicação prática das questões avaliadas possibilitou a implementação de um framework que, além de validar a arquitetura de software proposta por Valentin et al. (2008), poderá ser utilizado na implementação de software de diferentes domínios.

REFERÊNCIAS

BAUER, C., King, G., **Hibernate in action**. Manning Publications Co., 2005.

BROEMMER, D., J2EE **Best Practices** - Java Design Patterns, Automation, and Performance. Wiley Publish Inc, EUA, 2003.

METSKER S. J., **Design Pattern Java Workbook**. Addison Wesley, 2002.

MURPHY, C., METHODS & TOOLS - **Global knowledge source for software development professionals**. Spring 13(1), ISSN 1023-4918, 2005.

PAMPLONA, V. F., **Tutorial Java**: O que é Java?, disponível no endereço http://www.javafree.org/content/view.jf?idContent=84, acesso em agosto/2009.

VALENTIN, L. G., Dias, M. M., Pacheco, R. C. S. From Reference Architecture towards Software Architecture for Knowledge Discovery in Database Systems. In: **XXXIV CONFERENCIA LATINOAMERICANA DE INFORMÁTICA**, 2008, Santa Fé, Anais... Santa Fé - Argentina, 2008. p.679–688.