UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE – UFRN
INSTITUTO METRÓPOLE DIGITAL – IMD
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA – DIMAp
DIM0501 – BOAS PRÁTICAS DE PROGRAMAÇÃO
DOCENTE: HANDERSON BEZERRA
DISCENTES: WESLEY REUEL MARQUES SILVA – MAT: 2013005456

APLICAÇÃO DE BOAS PRÁTICAS DE PROGRAMAÇÃO

Natal-RN, maio de 2016

Sumário

1. Introdução

O seguinte trabalho tem como finalidade por em prática os conhecimentos adquiridos na disciplina de “Boas Práticas de Programação”, mostrando como um código bem escrito e organizado poderá ser de grande ajuda em sua manutenção, escalabilidade e reuso de código.

Após uma grande procura em arquivos próprios, e algumas dúvidas, o projeto escolhido para ser trabalhado foi um módulo de segurança (cadastros de módulos e suas funcionalidades), inicialmente o projeto desenvolvido em Windows Forms e SQL Server, para atender as demandas da disciplina foi necessário a conversão em Console Application, para que assim pudesse se tornar um produto que poderia ser avaliado academicamente.

O programa consiste em cadastro de módulos de uma aplicação, que podem existir sem funcionalidades (em desenvolvimento). Em outro momento temos os cadastros de funcionalidades, que só poderão existir pertencendo a um módulo, assim o que o sistema trata de fazer é um cadastro simples, sua listagem e edição.

2. Desenvolvimento

2.1 Apresentação

Uma vez que o programa não era Console, ele não foi propriamente modificado, em fato ele foi recriado, desconsiderando toda a aplicação existente, para ser criado puramente com efeitos acadêmicos, logo, não teremos exibição de alterações, e sim de um programa novo, criado apenas copiando a ideia de outro programa existente.

Ao verificarmos o código da aplicação é possível encontrar nomes como “SelecionarOperacao”, “ListarModulos”, “MenuPrincipal”, “SelecionaOpcao”, dentre outros, é possível ver que os nomes foram pensados na sua utilização, quando o método ListarModulos é chamado, espera-se que ele liste os módulos na tela para usuário, e assim por diante.

Códigos repetitivos como por exemplo o de selecionar uma opção no menu, foi tratado como um método, um exemplo é o SelecionarOpcao dentro da classe Menu, que seria repetido para cada método dentro do Menu (MenuPrincipal, MenuModulo, MenuFuncionalidade), e dentro dele temos um condição com loop, onde o loop não irá parar de solicitar que o usuário digite um valor válido, até que o valor válido seja digitado.

Assim é possível dizer que os comandos condicionais e loops aparecem em momentos de entrada de usuário, e quando o sistema irá exibir listas de objetos, como por exemplo, no método ListarFuncionalidade dentro da classe FuncionalidadeOperacoes.

Até agora só fora comentado sobre as classes que interagem quase que diretamente com o usuário, mas existem também as classes de Controller e de Entidade, que foram alocadas em outros projetos para facilitar a escalabilidade do projeto em si.

O projeto Seguranca.Entidades, pelo nome já é plausível dizer que trata das classes de entidade, assim suas classes e propriedades foram nomeadas de acordo com o Banco de Dados, que de todo o projeto foi o único que se manteve da ideia original. Esse projeto também possui dois outras classes que foram herdadas e alteradas da implementação do EntityFramework que visa ajudar a comunicação entre a aplicações Banco de Dados, que não será aqui explicado por não fazer parte do escopo do projeto. Assim como acontece com Seguranca.Entidades, aconteceu com Seguranca.Controller, que nada mais é do que um projeto que possui os controles das entidade declaradas no projeto já citado, e do mesmo modo herda classes da EntityFramework para auxilio geral de sua implementação.

É possível que você esteja se perguntando o motivo dos métodos e classes terem sido criados com a primeira letra maiúscula e não seguindo padrão apresentado em aula, mas para tudo no mundo se tem uma explicação, para realização desse trabalho foi usada a linguagem C#, que apesar de ser muito parecida com Java, tem suas peculiaridades e uma delas é o uso do PascalCase que diz que cada método/classe deve ter sua primeira letra em maiúsculo assim como a letra de cada nova palavra que irá compor nome do método/classe.

2.2 Métricas

Ao chegar às métricas, fora descoberto que o VisualStudio apenas apresenta apenas 5 métricas, sendo assim todas as cinco foram utilizadas nesse relatório, e estão anexo em documento Excel para apreciação posterior.

Métricas:

- Maintainability Index: De um modo geral o código se mostrou altamente fácil de fazer manutenção, trazendo um índice de 92 e caindo até 66 (por projeto), o que até o Visual Studio exibiu ser fácil de se manter. A maior parte dos índices baixo, seria devido aos métodos que interagem diretamente com o usuário.
- Cyclomatic Complexity: Nessa métrica pode-se dizer que o programa se manteve na casa dos 50, isso que dizer que para cada projeto existem em torno de 50 modos de realizar suas operações. Uma vez que essa métrica define a quantidade de caminhos lineares independentes que podem ser realizados pelo software.
- Depth of Inheritance: Foi possível manter uma dependência de no máximo 2, e a o projeto principal tem apenas 1. Assim as dependências do software são bem pequenas.
- Class Coupling: Métrica que tenta definir quantas classes únicas, uma classe usa, um blog oficial da Microsoft para desenvolvedores cita que 9 seria o número ideal, vemos que para cada projeto o número passa disso, sendo o seu maior 32 e seu menor 15, mas ao observar classe a classe, esse número cai tendo o seu maior por classe o valor de 29 que é a classe herdada do EntityFramework, e os outros valores que passaram de 9 variaram entre 11 e 13, e são as classes responsáveis por fazer alterações no sistema.

- Line of Codes: Por ultimo temos a linhas de código que contabiliza as linha de código criada pra cada método, projeto e solução, seus valores variaram de 1 no menor método para 182 no maior projeto.

2.3 Programação Defensiva / Testes Unitários

Como parte de programação defensiva, foram colocados repetições e clausulas lógicas (whiles e if) que irão garantir que o usuário não insira valores inválidos para o registro no banco de dados, principalmente quando se fala em registrar os valores de uma funcionalidade que depende de um módulo, o usuário irá ser questionado por um módulo válido. Mas vale lembrar que é possível inserir valores em branco. O tratamento realizado foi apenas na parte de chaves e buscas por chaves, garantido que onde haja necessidade de entrada de um número inteiro, isso será garantido.

Em contra partida, os teste unitários não foram realizados para tal projeto.

3. Conclusão

Assim é possível concluir que a disciplina de Boas Práticas de Programação, mostrou os motivos para se fazer códigos que possam ser lidos não apenas por máquinas mas por seres humanos, que em futuro (próximo ou não) poderão realizar uma manutenção no sistema, e caso esse sistema não possua documentação, com um código padronizado e entendível será mais fácil para o próximo programador (ou até mesmo o que criou o código originalmente) possam entender e compreender o código ali digitado.

4. Referências Bibliográficas

Capitalization Styles. Disponível em: <https://msdn.microsoft.com/en-us/library/x2dbyw72(v=vs.71).aspx>. Acesso em: 24 maio. 2016.

Code Metrics – Class Coupling. Disponível em:
<https://blogs.msdn.microsoft.com/zainnab/2011/05/25/code-metrics-class-coupling/>.
Acesso em: 24 maio. 2016.