Análise do Impacto da Aplicação de Padrões de Projeto na Manutenibilidade de Um Sistema Orientado a Objetos

Isaias Alves Ferreira[1], Antônio Maria P. de Resende[2], Heitor A. Xavier Costa[2]

[1]Laboratório de Estudos e Projetos em Manejo Florestal (LEMAF)
Universidade Federal de Lavras (UFLA) – Lavras, MG – Brazil

[2]Grupo de Pesquisa em Engenharia de Software (PqES)
Departamento de Ciência da Computação
Universidade Federal de Lavras (UFLA) – Lavras, MG – Brazil

isaias.ferreira@lemaf.ufla.br, {tonio,heitor}@dcc.ufla.br

Abstract. *Software design patterns are the result of best practices that help minimize the recurring problems in systems development. Thus, this article examines how the application of design patterns has contributed to improving the maintainability, among other quality factors, a legacy system during it is refactoring. Improving the quality of the system was observed through a comparative analysis of software metrics applied to the legacy system and the system refactored.*

Resumo. *Padrões de projeto de software são resultados de boas práticas que ajudam a minimizar problemas recorrentes no desenvolvimento de sistemas. Nesse sentido, este artigo analisa como a aplicação de padrões de projeto contribui para a melhoria da manutenibilidade, dentre outros fatores de qualidade, de um sistema legado durante sua refatoração. A melhoria da qualidade do sistema foi observada por meio de uma análise comparativa dos resultados de métricas de software aplicadas ao sistema legado e ao sistema refatorado.*

1. Introdução

Manutenibilidade é um atributo de qualidade de software que determina o grau de facilidade que o mesmo pode ser corrigido ou alterado diante de uma necessidade [Brusamolin, 2004 apud IEEE, 1993]. Coleman (et. al.,1995) citado por Brusamolin (2004), afirma que a manutenção de um software pode consumir tanto ou mais recursos que o desenvolvimento do mesmo e, por esse motivo, é extremamente importante que o fator manutenibilidade seja levado em consideração. Atualmente, a manutenção de um software não diz respeito apenas em mantê-lo funcionando. Os sistemas precisam estar em constante evolução para atender às novas funcionalidades que lhes são requeridas. No entanto, identifica-se a necessidade de produzir softwares de forma que a adição de novos requisitos seja facilitada e com maior confiabilidade.

De acordo com Lauder e Kent (1998), padrões de projeto de software são resultados de boas práticas que ajudam a minimizar problemas recorrentes no desenvolvimento de sistemas.

Este trabalho analisa o impacto da aplicação de padrões de projeto em um sistema legado por meio de métricas de software, fazendo uma comparativa dos resultados das métricas aplicadas ao sistema antes e após sua refatoração com a aplicação dos padrões. Além de uma pequena contextualização do tema apresentado nesta introdução, este trabalho apresenta conceitos de manutenibilidade e métricas de software nas seções 2 e 3 respectivamente seguidas do desenvolvimento do estudo de caso na seção 4 e da análise dos resultados na seção 5. Ao final, as conclusões e referências bibliográficas.

2. Manutenibilidade de Software

A norma ISO/IEC 9126-1 apresenta alguns atributos chave para identificar e medir a qualidade de um software. São eles: Funcionalidade, Confiabilidade, Usabilidade, Eficiência, Manutenibilidade e Portabilidade. Pigoski (1996), citado por Brusamolin (2004), sintetiza que "*Manutenção de software é a totalidade de atividades necessárias para prover, minimizando o custo, suporte a um sistema de software*". São três os tipos de manutenção mais comuns. A corretiva que visa unicamente à correção de eventuais defeitos, a adaptativa que corresponde às mudanças que o software sofre para se adaptar em um novo contexto e a perfectiva que corresponde às mudanças efetuadas para atender a novas necessidades do usuário. Segundo Pigoski (1996), 55% dos esforços de manutenção de software são para a perfectiva, 25% para a adaptativa e 20% para a corretiva.

A manutenibilidade é justamente a facilidade encontrada no software para que ele seja modificado [IEEE, 1993]. A norma ISO/IEC 9126-1 ainda define cinco características da manutenibilidade, são elas: a) analisabilidade que evidencia o esforço necessário para diagnosticar deficiências; b) modificabilidade que evidencia o esforço necessário para modificá-lo; c) estabilidade que evidencia o risco de acontecerem efeitos inesperados ocasionados por alguma modificação; d) testabilidade que evidencia o esforço necessário para validar o software mesmo após modificações; e) conformidade que analisa se o software está condizente com padrões que permitirão ao mesmo ser alocado em diferentes ambientes.

De acordo com Martins (2002), alguns fatores que impactam diretamente a manutenibilidade são arquitetura, tecnologia, documentação e compreensibilidade. Do ponto de vista da arquitetura foram elaborados testes de desenvolvimento de um mesmo sistema em diversas arquiteturas diferentes e concluiu-se que o esforço para modificar o software foi menor e menos defeitos foram identificados quando se utilizou unidades de código menores [Martins, 2002]. No contexto da tecnologia, Henry e Humphey (1990 apud BRUSAMOLIN, 2004) observaram que softwares produzidos em linguagens orientadas a objeto possuem alto índice de manutenibilidade. Quanto à documentação, a sua falta pode implicar em um alto gasto de tempo para compreender o produto antes de modificá-lo.

Obviamente, se ao adicionar uma nova funcionalidade, além de implementar o código responsável unicamente pela sua lógica, for necessário modificar diversas partes do software, a manutenção terá um custo muito maior comparado ao custo de alterar apenas pontos específicos e centralizados no código [BRUSAMOLIN, 2004].

3. Métricas de Software

De uma forma geral, métricas de software podem ser descritas como formas encontradas para se mensurar algum atributo de software e para aferir tempo, esforço e recursos necessários na execução dos processos de software. É muito importante que, durante os processos de desenvolvimento, sejam utilizadas medições para se avaliar a qualidade dos produtos de software dentre outros fatores. Os processos de software utilizam de diversas medições para que sejam executados com qualidade. Segundo Pressman (2006), alguns membros da comunidade de software afirmam que um software não pode ser mensurável ou que a tentativa de mensurá-lo deveria ser adiada por ainda não se ter conhecimento suficiente para fazê-lo corretamente. Pressman (2006) ainda afirma que isso é um erro. Existem muitas formas para se medir um software, de modo que a avaliação dos valores obtidos pela aplicação de métricas podem proporcionar resultados significativos no que diz respeito à melhoria dos processos e à qualidade do sistema.

Este trabalho utiliza a métrica *Maintainability Index* (MI) como base para o cálculo do índice de manutenibilidade. Consequentemente, as métricas de Halstead e a Complexidade Ciclomática de McCabe que compõem a MI também são abordadas.

3.1 Métricas de Halstead

A teoria de Halstead [HALSTEAD, 1977], foi a primeira a associar valores quantitativos ou mensuráveis ao desenvolvimento de softwares por meio de um conjunto de medidas que podem ser retiradas do código após o mesmo ser gerado. Estas medidas são: número de operadores distintos (n1); número de operandos distintos (n2); número total de operadores (N1); número total de operandos (N2);

Por meio destas medidas primitivas pode se obter:

a) Extensão do programa	N = N1 + N2
b) Vocabulário do programa	n = n1 + n2
c) Volume	V = N*(Log2n)
d) Dificuldade	D = (n1/2) * (N2/2)
e) Esforço	E = D*V

Tabela 1 – Medidas de Halstead

3.2. Complexidade Ciclomática de McCabe

Criada por Thomas McCabe em 1976 [BRUSAMOLIN, 2004], ela mede o número de caminhos linearmente independentes dentro de um módulo. Ela é largamente utilizada e não depende de uma linguagem. Montando-se o grafo dos possíveis fluxos dentro do programa podem-se obter os dados para se calcular a complexidade ciclomática. Os dados são: número de arestas (E); número de nodos (N) e o número de componentes conectados (p). O cálculo é feito da seguinte forma:

E – N + 2p

Tabela 2 – Fórmula da Complexidade Ciclomática de McCabe

3.3 Índice de Manutenibilidade (MI – *Maintainability Index*)

O índice de manutenibilidade (*Maintainability Index*) desenvolvido por Oman [Oman, 1994] tem como objetivo final diminuir os custos de manutenção podendo calcular o índice de manutenibilidade e apresentá-lo de uma forma conveniente. Coleman (et. al., 1995) explica como o MI foi calibrado validando seu uso para a indústria de software. O índice de manutenibilidade básico de um determinado programa é dado pelo polinômio descrito na Tabela 3, onde: aveV é a média da medida de Halstead por módulo; aveLOC é a média de linhas de código por módulo; aveV(g') é a média extendida da complexidade ciclomática (McCabe) por módulo; perCM é a média do percentual de comentários por linha de código (opcional).

171–(5,2*ln(aveV))–(0,23*aveV(g'))–(16,2*ln(aveLOC))+(50*sin(√2,4perCM))

Tabela 3 – Índice de Manutenibilidade

4. Estudo de Caso

Como estudo de caso, utilizou-se um módulo de um software legado que tem por objetivo automatizar procedimentos padrão relativos à análise ambiental do estado de Minas Gerais para concessão de licenças ambientais. Foi desenvolvido pelo Laboratório de Estudos e Projetos em Manejo Florestal (LEMAF) anexo ao Departamento de Ciências Florestais da Universidade Federal de Lavras utilizando-se a linguagem JAVA J2EE, o padrão MVC, as tecnologias Spring Framework e Hibernate/JPA dentre outras. Mudanças em algumas especificações e a necessidade da adição de novos requisitos ao projeto permitiram que alguns fatores que dificultavam a manutenibilidade do sistema fossem identificados. Diante disso, decidiu-se refatorar um módulo do sistema legado aplicando-se alguns padrões e boas práticas para que sua manutenibilidade fosse melhorada. Os principais problemas levantados foram: a) novas funcionalidades quase sempre implicavam na criação de novas classes; b) redundância de partes do código; c) múltiplas instancias de um mesmo objeto sem necessidade; d) dificuldade para identificar a instancia correta a ser utilizada em determinado caso; e) métodos extensos; f) demora em entender o contexto do código; g) demora em localizar defeitos; h) incerteza sobre o tamanho do escopo, no código, de uma funcionalidade; j) quantidade considerável de classes com apenas um método.

Segundo Quan (et. al., 2008), os padrões e boas práticas de design no paradigma orientado a objetos, são demonstrações objetivas de como utilizar melhor as possibilidades que o paradigma proporciona. As soluções para as deficiências foram a aplicação dos padrões e boas práticas apresentados a seguir durante a refatoração do código.

a) Padrão Factory – consiste basicamente em criar uma fábrica de objetos que são diferentes, porém possuem a mesma abstração [Freeman et. al., 2007].

b) Padrão Template – este padrão ajuda muito a reduzir a duplicidade de partes do código. Utiliza classes e métodos que servem como modelos para realizar determinada tarefa. São classes e métodos abstratos [Freeman et. al., 2007].

c) Padrão Singleton – este é o padrão que garante a existência de apenas uma instância de um objeto em toda a aplicação [Freeman et.al., 2007].

d) Padrão Command – este padrão consiste basicamente em encapsular uma chamada a um método [Freeman et. al., 2007]. Mais detalhadamente, é como se a aplicação fosse como um provedor de serviços a um cliente (entende-se como cliente, outro programa ou sistema que consuma aqueles serviços).

e) Not Anemic Domain Model – esta prática evita que existam "*classes anêmicas*" no sistema. Essas classes, geralmente, são aquelas que possuem apenas seus atributos e métodos "*getters*" e "*setters*". Ou seja, classes sem comportamento.

f) Métodos pouco extensos – Essa prática mantém métodos relativamente pequenos no sistema, ou seja, defende dividir métodos sempre que possível e plausível.

A prática denominada "*Not Anemic Domain Model*" foi a principal dentre as abordadas neste trabalho para o aumento do índice de manutenibilidade do sistema legado. As "*classes anêmicas*", evitadas por essa prática, podem ser grandes vilãs da manutenibilidade de um software. Elas se assemelham aos registros que existem em linguagens procedurais como o Pascal por exemplo. Uma das principais características de um objeto dentro do paradigma Orientado a Objetos, é permitir que ele se movimente, que ele se altere. Quando há a necessidade de manipular um atributo de um objeto por exemplo, há uma tendência em fazer sua manipulação dentro da lógica que está se implementando. Isso pode não ser bom. A prática defende que toda a lógica que diz respeito apenas a um objeto deve estar dentro do próprio objeto. Se a lógica que manipula unicamente atributos do objeto não estiver dentro do objeto, em todas as partes do código que for necessário manipular seus atributos, provavelmente, haverá redundância de código. Evitar classes anêmicas pode aumentar a manutenibilidade do software em vários aspectos e a legibilidade é um deles. É muito mais claro entender a chamada de um método com um nome sugestivo do que entender várias linhas de código para executar uma rotina que não está diretamente ligada ou que não é de responsabilidade da lógica que se está implementando.

Na utilização de métodos muito grandes para uma determinada lógica, pode não haver junção nem comunicação entre as partes simplesmente porque não haverá partes. Métodos relativamente pequenos pode ser um bom fator para o aumento da legibilidade e, consequentemente, da manutenibilidade do código. Quando se consegue implementar uma pequena ação e dar nome a ela, as ações ficam mais legíveis e manuteníveis no sentido de que o provável problema já estará quebrado em problemas menores e obviamente mais fáceis de serem localizados e resolvidos.

5. Análise Comparativa e Discussão dos Resultados

Na seção anterior desenvolveu-se um sistema refatorado a partir da correção de deficiências do sistema legado. Para a observação da ocorrência de melhorias efetivas, aplicaram-se várias métricas no sistema legado e no refatorado, a fim de comparar os resultados, como mostrado nas Tabelas 3 e 4. Para a coleta das métricas necessárias para o cálculo do índice de manutenibilidade utilizou-se um plug-in para o eclipse denominado Google CodePro Analytix. O gráfico de radar apresentado pela figura 1 mostra a diferença dos resultados de uma forma proporcional onde os valores foram normalizados como porcentagem. Por exemplo, o resultado da medida LOC obtida do sistema refatorado foi de aproximadamente 40% do valor da mesma medida obtida do sistema legado.

Descrição	*Fórmula*	*Sistema Legado*	*Sistema Refatorado*
Número de operandos distintos	n1	**20**	**17**
Número de operandos distintos	n2	**215**	**156**
Número total de operadores	N1	**1.723**	**440**
Número total de operandos	N2	**3.085**	**1.079**
Extensão do programa	N = N1 + N2	**4.808**	**1.519**
Vocabulário do programa	n = n1 + n2	**235**	**173**
Volume	V = N*(log2n)	**37.870,29**	**11.293,2**
Dificuldade	D=(n1/2)*(N2/2)	**143,48**	**58,79**
Esforço	E = D*V	**5.443.946,76**	**663.946,06**
Complexidade Ciclomática	CC	**35,55**	**5,42**
Número de linhas de código	LOC	**1.268**	**503**

Tabela 3 - Resultado das métricas abordadas

Descrição	Fórmula	Sistema Legado	Sistema Refatorado
MI (escala de -∞ a 171)	171-(5,2*ln(V))-(0,23*CC)-(16,2*ln(aveLOC))	**-7,72**	**20,48**

Tabela 4 - Resultado do Índice de Manutenibilidade

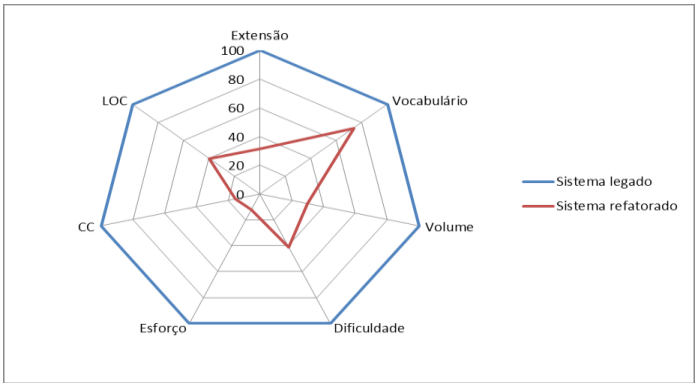

Figura 1 – Gráfico de comparação proporcional

Conclui-se que o sistema refatorado obteve melhorias significativas em relação ao sistema legado pela observação das medidas obtidas com a aplicação das métricas. Nota-se ainda, que todas as medidas do sistema refatorado apresentaram desempenho melhor em relação ao sistema legado. Os tópicos a seguir analisam como os padrões aplicados ao código puderam contribuir para a melhoria da qualidade do sistema.

Em relação à Eficiência, um código menos extenso e menos complexo possibilita ao sistema um aproveitamento maior de seus recursos [Freeman et. al., 2007]. Como observado pelos resultados, houve uma redução da Extensão do código de

4.808 pontos para 1.519 pontos e também uma redução da Complexidade Ciclomática de 35,55 pontos para 5,42 pontos após a refatoração.

Quanto à funcionalidade, como a manutenibilidade em geral melhorou, a implementação de novos requisitos tem uma maior probabilidade de ser efetuada com sucesso. Entretanto, a verificação da funcionalidade de um sistema depende de vários fatores e só pode ser confirmada após a validação do software.

A diminuição na Dificuldade após a aplicação dos padrões de 143,48 pontos para 57,79 pontos indica que uma eventual manutenção será feita com menor dificuldade e, portanto, permitindo que o sistema esteja mais confiável. A diminuição da Complexidade Ciclomática também colabora para a confiabilidade.

Em relação à Manutenibilidade do sistema, a métrica *Maintainability Index* (MI) abordada na seção 3.3, utiliza uma combinação de métricas que mensuram diversos fatores de um sistema como o Volume, o Esforço e a Complexidade Ciclomática por exemplo. Percebe-se pelos resultados que todas as métricas calculadas tiveram um resultado melhor após a refatoração do código aplicando-se os padrões. Padrões como o Factory, o Template e a utilização de *classes não anêmicas*, fazem com que o código seja reduzido evitando redundâncias. São muitas as vantagens de se ter uma menor quantidade de código escrito.

Além de melhorar a legibilidade do código, a redução na Dificuldade de 143,48 pontos para 58,79 pontos, observada pelo resultado dessa métrica, reflete que numa eventual manutenção, o código estará mais fácil de ser alterado e de ser entendido. Alguns fatores que contribuíram para a diminuição do Volume de 37.870,29 pontos para 11.293,2 pontos, do Vocabulário de 235 pontos para 173 pontos e da Extensão do código de 4808 pontos para 1.519 pontos de acordo com os resultados obtidos, foram as reduções dos números de operadores e operandos. O uso de *classes não anêmicas* contribui para a diminuição destes fatores no sentido de que tirar partes da lógica das rotinas principais do sistema na medida do possível, leva para outros locais operadores e operandos que estariam deixando a lógica principal menos legível dificultando, principalmente, a manutenção perfectiva naquele código. Em suma, a reutilização de grande parte do código e a centralização de determinadas implementações foram os grandes responsáveis pela grande melhoria no índice de manutenibilidade do sistema abordado.

6. Conclusão

Para a análise do impacto da aplicação de padrões de projeto em um sistema, utilizou-se um sistema legado desenvolvido na linguagem JAVA J2EE que tem por objetivo automatizar procedimentos padrões no gerenciamento de manejo florestal e análise ambiental. O sistema legado utilizado apresentou deficiências quanto à sua manutenibilidade e foi refatorado aplicando-se alguns padrões e boas práticas de software, a fim de melhorar sua qualidade, especialmente seu índice de manutenibilidade. A observação das melhorias se deu por meio da aplicação de métricas de software.

Os resultados de todas as métricas aplicadas ao sistema refatorado foram satisfatórios em relação aos resultados obtidos pelo sistema legado. Diante disso pôde-se

concluir que os padrões e as boas práticas aplicadas tiveram impacto positivo melhorando a qualidade e em especial a manutenibilidade do sistema legado.

Como trabalhos futuros, pretende-se realizar uma revisão sistemática a procura de valores de referência para as métricas aplicadas neste artigo e outras de interesse do grupo de pesquisa. Havendo valores de referência pretende-se estabelecer protocolos aplicáveis no mercado, as métricas sem valores de referência serão alvos de estudos para estabelecer intervalos para diagnóstico da qualidade do software para os diversos fatores de interesse do grupo de pesquisa PqES.

8. Referências Bibliográficas

BRUSAMOLIN, Valério. Manutenibilidade de Software, Instituto Científico de Ensino Superior e Pesquisa – ICESP, Revista Digital Online vol. 2, 2004.

LAUDER, A., KENT, S. Precise Visual Specification of Design Patterns – ECCOP 98 Anais da 12 a Conferência Europeia sobre Programação Orientada a Objetos – Londres, 1998.

NBR ISO/IEC 12207 – Tecnologia de informação – Processos de ciclo de vida de software. Rio de Janeiro: ABNT, 1998.

ISO/IEC 9126-1. Software engineering – Software product quality – Part 1: Quality Model, 2000.

PRESSMAN, Roger S. Engenharia de Software, 6ª edição, McGraw-Hill, São Paulo, 2006.

HALSTEAD, M. H. Elements of Software Science, Operating and Programming Systems Series. Vol. 7. Elsevier, 1977.

OMAN, P. W., HAGEMEISTER, J. R., Construction and Testing of Polynomials Predicting Software Maintainability. Journal of Systems and Software, pp. 251 – 266, 1994.

COLEMAN, Don. LOWTHER, Bruce. OMAN, Paul. The application of Software Maintainability Models in Industrials Software Systems. J. Systems Software, 1995.

Quan, L. Zongyan, Q. Liu, Z. Formal Use of Design Patterns and Refactoring, T. Margaria and B. Steffen (Eds.): ISoLA 2008, CCIS 17, pp. 323–338, 2008.

PIGOSKI, Thomas M. Pratical Software Maintenance: Best Practices for Managing Your Software Investiment. Wiley Computer Publishing, 1996.

MARTINS, Vidal, VOLPI, Lisiane M. Influência da Arquitetura na Qualidade do Software. Bate Byte, 2002.

HENRY, Sallie M., HUMPHEY, Matthew. A Controlled Experiment to Evaluate Maintainability of Object-Oriented Software. IEEE, 1993.

LI, Wey, HENRY, Sallie. Maintenance Metrics for the Object Oriented Paradigm. IEEE, 1993.

FREEMAN, Eric, FREEMAN, Elizabeth. Use a cabeça! Padrões de Projeto (design Patterns), 2ª edição, Alta Books, 2007.