UNIVERSIDADE PAULISTA – UNIP

CURSO DE CIÊNCIA DA COMPUTAÇÃO

CRIAÇÃO DE UM MODELO DE PADRONIZAÇÃO DE ALGORITMOS EM ALTO NÍVEL: MODELO MOA

GREISON FERNANDO BASSAN

LIMEIRA-SP, NOVEMBRO DE 2015

GREISON FERNANDO BASSAN

CRIAÇÃO DE UM MODELO DE PADRONIZAÇÃO DE ALGORITMOS EM ALTO NÍVEL: MODELO MOA

Trabalho de conclusão de curso para a obtenção do título de Bacharel em Ciência da Computação pela Universidade Paulista.

Professores orientadores: Marcos Vinícius Gialdi, Antonio Mateus Locci e Sandra Maria Crippa.

LIMEIRA-SP, NOVEMBRO DE 2015

GREISON FERNANDO BASSAN

CRIAÇÃO DE UM MODELO DE PADRONIZAÇÃO DE ALGORITMOS EM ALTO NÍVEL: MODELO MOA

_____

Prof. Marcos Vinícius Gialdi

_____

Prof. Amaury André

_____

Profa. Sandra Maria Crippa

Agradeço primeiramente a Deus por me ajudar a alcançar meus objetivos e a meus pais por me darem o apoio necessário para mais esta etapa concluída em minha vida. Também sou grato aos meus professores por me passarem seus conhecimentos necessários para uma boa formação profissional.

RESUMO

Esta pesquisa tem por objetivo mostrar técnicas de otimização de algoritmos e aplica-las em um modelo chamado de MOA (Modelo de Otimização de Algoritmos), onde esse modelo deve ser seguido para que se tenha um código de qualidade, para evitar futuras falhas de desempenho e um código de difícil legibilidade. A utilização desse modelo deve seguir os padrões exigidos pelo mesmo, para que no final do projeto não tenha que ser realizado um processo extremamente complexo, facilitando a otimização final.

Palavras-chaves: Modelo MOA, Padronização, Otimização, Documentação e Código Legível.

ABSTRACT

This research aims to show algorithms optimization techniques and apply them to a model called MOA (Algorithms Optimization Model), where this model should be followed in order to have a quality code, to avoid future performance failures and a hard code readability. The use of this model should follow the standards required by the same, so in the end the project does not have to be carried out an extremely complex process, facilitating the final optimization.

Keywords: Model MOA, Standardization, Optimization, Documentation and Readable Code.

SUMÁRIO

Figuras

Tabelas

1. INTRODUÇÃO

Durante o desenvolvimento de um projeto de software temos várias fases das quais devemos seguir de maneira correta e cuidadosa, para não comprometer nenhuma delas, resultando em um gasto maior de tempo e um risco de ocorrerem mais falhas. Essas fases são obtidas através da utilização da engenharia de software, que acompanha e organiza todo o projeto do software, desde a análise de requisitos, onde são obtidas todas as informações que o software deve conter, como a sua interface, suas regras de negócios, até a fase de manutenção, onde poderão ser realizadas alterações posteriores de acordo com a necessidade do cliente.

O foco desta pesquisa será voltado mais para a fase de codificação do projeto, onde serão discutidas maneiras de implementar um novo modelo para o auxílio desta fase, de forma que tenhamos uma codificação padronizada.

Quando desenvolvemos um algoritmo, podemos resolver um problema em questão de diversas formas, dependendo da situação. Vários algoritmos podem fazer com que um problema possa ser resolvido, mas nem todos podem ter o melhor fluxo de execução, causando às vezes lentidão na hora da sua execução, devido ao computador do cliente não ser de alto desempenho em comparação ao computador no qual foi desenvolvido. Para que esse problema não ocorra, devemos utilizar métodos de otimização de código, no qual será discutido seu funcionamento com mais detalhes no decorrer da pesquisa.

Os algoritmos devem ser otimizados levando em conta todos os pontos que podem gerar mais processamento e ocupar mais espaço na memória, de maneira desnecessária, para que ele possa funcionar não somente em máquinas top de linha, mas claro, levando sempre em conta o nível do produto, ou seja, o porte da empresa que utilizará o software. Em empresas de pequeno porte pode-se utilizar um software em uma máquina comum, por exemplo, já em uma empresa de grande porte deverá ter computadores mais potentes para que o serviço não pare durante a sua execução.

Apesar de uma boa otimização ser feita no código, isso pode gerar efeitos colaterais também, pois os códigos anteriores às otimizações podem ter passado por diversos testes, enquanto que os otimizados ainda não. Nesse caso, devemos estar

testando eles novamente, para que seja analisada a integridade e consistência do código gerado, para que não ocorram erros posteriormente, como na fase da implementação, por exemplo.

Além disso, vamos discutir técnicas para uma otimização visual no código, para que ele tenha uma melhor legibilidade e para que ele tenha uma documentação adequada, facilitando sua reutilização e alterações necessárias em futuras manutenções.

1.1. Objetivo

O objetivo é juntar técnicas já existentes no processo de desenvolvimento de software e criar uma nova solução, com o intuito de trazer uma melhor organização no processo de codificação, gerando uma maior facilidade na administração desta etapa e reduzindo a probabilidade de gerar erros.

Vamos unir estas técnicas já existentes e introduzi-las em um modelo de projeto de codificação, denominado de modelo MOA (Modelo de Otimização de Algoritmos). Ele tem a função de ajudar na otimização dos algoritmos, sendo que ele é composto por regras que devem ser respeitadas durante a fase de codificação de software, para que no final do projeto ele esteja totalmente otimizado, evitando ter que alterá-lo novamente após a finalização da codificação, apenas passando por um processo de verificação.

Além da otimização, ele também engloba técnicas para uma boa documentação do software, para que se possa realmente entender o que acontece em cada bloco de código, tornando assim um código fonte que qualquer programador, desde tenha uma experiência adequada para entender como funciona determinada linguagem, possa realmente saber o que está acontecendo naquele bloco, sem precisar ficar um longo tempo tentando descobrir como o mesmo está funcionando.

Este modelo também engloba técnicas de legibilidade, para tornar o código escrito de uma forma padronizada, tornando mais fácil a busca por erros decorrentes de sintaxe e erros de lógica.

1.2. Estrutura do trabalho

A introdução está apresentando, de uma maneira geral, as diferenças entre algoritmos desenvolvidos sem a preocupação de como eles estão sendo escritos e os algoritmos que são escritos utilizando técnicas de otimização e legibilidade. Ela também apresenta o objetivo geral do trabalho.

Capítulo 2: descreve como funcionam os processos de otimização de maneira geral.

Capítulo 3: descreve como funciona o processo de documentação e a sua importância.

Capítulo 4: descreve o conceito de reutilização de software e seus benefícios para novos projetos.

Capítulo 5: descreve a importância da padronização de código durante o processo de codificação.

Capítulo 6: descreve como funcionam as reuniões diárias utilizando a metodologia Scrum

Capítulo 7: descreve os tipos de manutenção de software e explica cada um deles.

Capítulo 8: aqui são demonstradas todas as técnicas para se escrever um código melhor em equipes, sendo englobadas todas elas em um modelo para se utilizar durante um projeto de codificação.

2. OTIMIZAÇÃO

Vários são os motivos para a otimização de códigos, sendo que sempre devemos desenvolver algoritmos que ocupem a menor quantidade de memória possível e algoritmos mais rápidos, para que processadores de baixo custo possam executar o algoritmo de forma mais eficaz, não sendo necessário utilizar apenas máquinas de alto desempenho para que a solução seja executada, tanto para que o

custo de uma implementação fique mais barato, quanto para que o gasto com energia seja menor.

Existem três níveis de otimização de códigos, sendo eles:

- Nível de design;
- Nível de compilação;
- Nível de assembly.

2.1. Nível de design

Este é o nível mais alto, onde devem ser aproveitados todos os recursos que a linguagem oferece, como métodos já prontos, para que não seja necessário serem reescritos, entre outros recursos disponíveis para determinada linguagem. Nesse nível, devem ser analisados todos os possíveis métodos para uma boa arquitetura do algoritmo, pois esta é a etapa que mais influenciará no desempenho do algoritmo.

2.2. Nível de compilação

Para garantir que será gerado um código otimizado após sua compilação também devemos sempre utilizar um compilador que garanta isso, para que seja gerado um código de máquina com o melhor desempenho possível. O desempenho entre os compiladores pode variar de acordo com a sua estrutura, então escolher um bom compilador é fundamental na hora de se desenvolver um projeto de software.

2.3. Nível de assembly

Para um algoritmo melhor e mais eficiente, podemos utilizar a linguagem Assembly, que é a linguagem de mais baixo nível de programação, onde obtemos um maior desempenho para solucionar problemas, porém é geralmente utilizado para se

desenvolver quando se conhece a arquitetura de hardware onde será utilizado posteriormente, para que a implementação ocorra da melhor maneira possível. Nesse nível, o programador pode aproveitar todos os recursos de instrução de máquina, sendo uma codificação mais complexa se comparado ao alto nível.

2.4. Procedimentos para otimização

Para que a otimização ocorra da maneira esperada devemos seguir alguns passos para que o efeito não seja inesperado. Devem ser utilizados procedimentos no código, mas os mesmos não podem interferir na finalidade do algoritmo e nem produzir erros. Após o primeiro passo cumprido deve ser analisado se o mesmo está funcionando de forma mais rápida que anteriormente e se realmente ocupa menos espaço na memória e menos processamento na CPU. Por esse motivo deve sempre ser realizado esse procedimento no final de um projeto ou de um módulo do projeto, para que seja testado de forma eficaz.

Devem ser analisadas todas as expressões de um algoritmo para que não sejam reescritas diversas vezes sem necessidade, e sim aproveitar uma que já foi realizada antes. Também devem ser retirados códigos mortos do algoritmo, pois eles estão escritos, mas nunca serão utilizados.

Uma boa análise deve ser feita dentro dos loops do algoritmo, pois os mesmos podem estar executando procedimentos diversas vezes desnecessariamente, sendo que podiam estar sendo declarados fora do loop.

Além desses tipos de otimizações citadas acima, temos diversas outras, que devem ser utilizadas na hora da codificação, de acordo com o problema que deve ser resolvido.

2.5. Vantagens da otimização

As razões para ser aplicar a otimização nos softwares geralmente são geradas devido a problemas complexos, onde o consumo de hardware deve ser levado em conta. O processo de otimização deve se tornar uma prática comum na hora de codificar um software, para que esse software aproveite somente o necessário dos recursos do hardware, gerando assim melhorias no desempenho da execução do programa e reduzindo o uso desnecessário de energia.

Sistemas de grandes empresas, principalmente, consomem muito processamento em seus servidores, então cada processamento a mais que é realizado ou cada espaço a mais de memória que é utilizado, deve ser levado em conta na hora de sua codificação. Além da lentidão que isso pode gerar para realizar as tarefas necessárias no sistema, isso pode gerar custos desnecessários também, já que poderão ser gastos mais com investimentos em hardware para suportar todo o processamento gerado, sendo que se o sistema for otimizado de maneira adequada, ele poderia consumir menos dos recursos disponíveis.

2.6. Otimização x tempo

A otimização no processo de codificação traz grandes melhorias para a qualidade gerada no software, porém, isso também gera um tempo a mais na hora de sua execução. A otimização é uma técnica que deve ser implementada de maneira que venha a ajudar nas melhorias do software no quesito desempenho, mas isso também gera um gasto a mais na hora da implementação deste processo, já que além de codificar para que o software funcione corretamente, deve-se estar sempre atento em como ele está funcionando, como estão sendo realizadas as etapas do bloco de código, e não apenas na preocupação em se ele está sendo executado corretamente e se está funcionando de acordo com o esperado pelo cliente.

Todos esses pontos devem ser levados em conta na hora de se projetar a etapa de codificação do software, para que ele seja desenvolvido de acordo com os prazos desejados e de acordo com o custo-benefício.

Sistemas mais simples que gerem uma menor quantidade de processamento talvez não necessitem de uma otimização muito complexa, apenas deve estar

produzindo um resultado funcional e que não gere um tempo de espera fora do aceitável pelo cliente. Tudo isso deve ser analisado antes do início da etapa de codificação, para que todos os membros da equipe já estejam cientes do grau de complexidade do software.

3. DOCUMENTAÇÃO

A documentação de software é uma peça fundamental no desenvolvimento de um software, pois é através dela que nós saberemos posteriormente como o sistema foi projetado, desde a etapa de análise de requisitos até a etapa final do projeto. Nem sempre essa é uma prática desenvolvida no desenvolvimento de novos sistemas, aplicativos e afins, mas é através dela que conseguiremos saber como tudo está funcionando, e ela funciona como um auxílio na hora de encontrar falhas no sistema sejam elas falhas do programador ou falhas na hora da obtenção dos requisitos que o cliente deseja.

A etapa da documentação é dividida em dois tipos, que são eles a documentação técnica e a documentação de uso. A documentação técnica é voltada para quem participou do desenvolvimento do software e para quem já entende do assunto, sendo que nela são compostas as regras de negócio do software, suas funções, prototipação, modelagem de banco de dados, dentre outras. Através dessa documentação conseguimos saber como tudo está ocorrendo no software, e qualquer futura manutenção será mais fácil, já que através dela, conseguimos saber se alguma alteração irá afetar outra parte do software e gerar outras falhas, entre todos os outros problemas que uma manutenção pode causar.

Esse é o ponto mais importante que devemos sempre lembrar na hora de um novo projeto de desenvolvimento, pois não devemos confiar em nossa memória para saber como está funcionando o sistema, já que essa é uma tática falha, e softwares podem ser prejudicados de uma maneira drástica com isso, já que a empresa que desenvolver o software pode ficar presa a manter os funcionários que o codificaram, já que eles são os únicos que sabem como o sistema está funcionando, e eles são os únicos que conseguem realizar alguma alteração caso haja a necessidade. Isso não

quer dizer que eles vão realizar alterações da melhor maneira possível, pois, como já foi citada, a memória é falha, e pontos importantes do sistema podem ser esquecidos.

Uma documentação bem elaborada de um sistema traz benefícios enormes para quem o desenvolveu e para quem o utiliza, já que a empresa não fica atrelada a manter os funcionários apenas porque somente são eles quem sabe alterar o sistema, e com isso, as alterações tanto simples como complexas, podem ser realizadas em um período mais curto de tempo em relação a alterações que são realizadas sem saber exatamente onde se vai mexer e o que isso vai afetar no sistema. Com isso o cliente vai ter uma maior agilidade na hora de pedir uma manutenção, o que pode gerar créditos para a empresa responsável pelo sistema.

A documentação de uso já é mais voltada para o usuário, pois ela funciona como um manual para ele, onde é explicado detalhadamente como funciona cada parte do sistema, qual a função de cada janela aberta, quais são os relatórios que ele gera, entre as demais funções que um sistema pode gerar. Essa documentação gera uma maior facilidade para o usuário do sistema, já que ele não vai precisar entrar em contato com a empresa que desenvolveu o sistema a cada duvida que for gerada, já que tudo deve estar detalhado na documentação.

A documentação que vai ser utilizada como um dos focos dessa pesquisa vai ser a documentação de código, que nem sempre é realizada como deveria, já que a maioria dos programadores pensa que vão lembrar como cada linha de código funciona e que nunca ninguém mais irá precisar mexer no código a não ser ele.

Essa documentação é constituída de comentários realizados no código em que se está codificando e documentações online geradas através dos comentários. Essas documentações devem ser descritas de forma simples e objetiva, pois a intenção não é escrever linhas e mais linhas de comentário, e sim escrever o que realmente aquela linha ou trecho de código está fazendo. Ela deve descrever o que ocorre no trecho de código que a segue para que no momento em que for necessário corrigir algum erro no sistema ou for necessário realizar alguma alteração, seja mais fácil encontrar onde se deve fazer a alteração. Fazendo isso evitamos que gambiarras ocorram no código, já que programados podem achar melhor adicionar algum trecho de código a mais no sistema e o deixar funcionando como o desejado, do que mexer em alguma parte do código que ele não sabe se poderá prejudicar em alguma funcionalidade do sistema.

Documentações um pouco mais extensas podem e devem existir no começo de cada classe, informando qual é a sua função no sistema e o que contém dentro dela, mas o foco é o mesmo dos demais comentários, ela deve ser escrita sem exceções e de forma objetiva, para que se possa ter um controle total do que cada parte o sistema faz.

4. REUTILIZAÇÃO DE CÓDIGO

A reutilização de código consiste em utilizar códigos já escritos para outros sistemas em sistemas novos, sem precisar escrever tudo novamente. Ela gera uma maior qualidade para o sistema, já que códigos que já foram escritos e estão sendo utilizados passaram por testes e funcionam corretamente. A reutilização de código também gera um menor consumo de tempo na hora de realizar a codificação, já que toda a parte que já está pronta só dever ser implementada no sistema e apenas desenvolver o restante do sistema.

É um conceito muito importante para o processo de codificação, e por isso devemos utilizar ele, pois desenvolver tudo o que já está pronto novamente não traz nenhum benefício para o funcionamento dos sistemas, e sim um atraso nos prazos do projeto.

O conceito de reutilização de código deve ser introduzido na equipe de desenvolvimento desde o início de um novo projeto, onde será escrito pela primeira vez o código que poderá ser reutilizado posteriormente. Para que tudo funcione de forma eficaz, esse código deve ser escrito de uma forma padronizada, assim como todo o restante do código, para que ele possa ser entendido novamente em uma nova implementação, porque se ele estiver bagunçado e não estiver exemplificando corretamente o que cada bloco faz, ele irá gerar um gasto a mais de tempo na hora de sua implementação, o que geraria mais custos e tempo.

5. PADRONIZAÇÃO DE CÓDIGO

Durante a codificação de um sistema podemos escrever códigos de diversas maneiras e deixá-los funcionando da mesma forma. Isso é uma prática bastante comum, principalmente com programadores iniciantes ou empresas que não se preocupam com a padronização dos códigos, porém, isso pode gerar muitos problemas no trabalho em equipe e na hora de realizar manutenções, já que quem vai alterar o código pode não entender como o mesmo está funcionando ou então demorar um maior tempo para entendê-lo, o que gera um maior custo.

Com isso, vemos a grande importância em se padronizar um código durante o processo de codificação, mesmo que não seja estabelecido e exigido pela empresa onde se trabalha, pois isso facilitará o entendimento dos códigos escritos pela mesma equipe e por alguma outra equipe na hora de realizar alguma alteração. Essa prática deve ser constante, respeitando as regras de convenções da linguagem em que se está sendo escrita e em sua documentação, sem exceções.

As linguagens de programação já têm um padrão pré-estabelecido na sua sintaxe, mas na questão de nomes de variáveis, classes, identação, entre outros, cabe ao programador manter sempre um padrão, tanto na hora do desenvolvimento de um novo código, quanto na hora de alterar alguma parte do código.

6. DAILY SCRUM

Dentro da metodologia Scrum, temos reuniões chamadas Daily Scrum, que são reuniões diárias realizadas para saber o que foi feito no dia anterior, identificar se algo está impedindo de realizar o trabalho e souber o que será feito. São reuniões compostas por todas as equipes do desenvolvimento e poder conter pessoas de fora também, contando que elas apenas escutem, para não comprometer o prazo da reunião. Esta reunião deve ser realizada todo dia pela manhã com duração máxima de quinze minutos e ela é conduzia pelo Scrum Master, onde o mesmo geralmente é o gerente do projeto. Ele é responsável por resolver qualquer problema que estiver impedindo o desenvolvimento do projeto.

Questões que forem levantadas, como os impedimentos que estão prejudicando no processo, devem ser discutidos e solucionados em reuniões a parte, com equipes menores, para ver diretamente o que está acontecendo.

Nesta reunião diária, as perguntas que devem ser feitas serão as seguintes:

- O que você fez ontem?
- O que você fará hoje?
- Há algum impedimento no seu caminho?

Com as respostas obtidas através destas perguntas, cada integrante da reunião compromete-se a realizar o que foi dito na frente dos demais e compreende-se totalmente o que está sendo feito por cada um.

7. MANUTENÇÃO

A manutenção de software é uma etapa da vida útil de um software que está presente em todo o seu ciclo de vida, já que um software bem desenvolvido pode ser utilizado por anos e anos, e com isso geralmente são necessárias alterações. Esta etapa é a que consome o maior tempo no desenvolvimento de um software, já que ela vai acompanhar o software durante todo o tempo que ele estiver sendo utilizado. Regras de negócios e funcionalidades podem alterar conforme o tempo e é importante sempre manter o sistema atualizado para que ele não prejudique nos processos da empresa, causando alguma perca de tempo ou até um prejuízo financeiro para a empresa.

Existem quatro de tipos de manutenções que podem ser realizadas em um software, sendo elas:

- Manutenção corretiva,
- Manutenção adaptativa,
- Manutenção evolutiva,
- Manutenção preventiva.

A manutenção corretiva é responsável por corrigir erros no sistema que estão causando falhas no seu funcionamento. Esses erros podem ser ocasionados tanto por

erros na hora da codificação, como na geração de relatórios, contas aritméticas, etc., como também podem ser erros ocasionados durante a análise de requisitos, onde o cliente pode não ter passado de maneira transparente para a equipe de desenvolvimento o que ele realmente queria para determinada funcionalidade do sistema, causando algum problema ou falta de alguma função no sistema.

A manutenção adaptativa é responsável por realizar alterações no sistema para que ele aceite mudanças no ambiente em que se está sendo utilizado para seu funcionamento. Essas mudanças podem ser alterações na regra de negócio da empresa, mudança no hardware em que ele está sendo executado ou mudança no sistema operacional para o qual ele não foi desenvolvido. Enfim, esta etapa deve solucionar todos esses problemas para que não ocorra nenhuma incompatibilidade com essas novas mudanças em seu ambiente de execução.

A manutenção evolutiva é responsável por adicionar funcionalidades ao sistema, funções as quais o sistema não precisava antes, mas que com alguma nova regra na empresa, ele deve se adaptar para acompanhar o processo corretamente ou até mesmo para otimizar o desempenho do sistema.

A manutenção preventiva é responsável por alterar o sistema para que ele tenha uma maior confiabilidade no seu funcionamento. Para passar para o usuário essa confiabilidade é necessária que a equipe responsável pela manutenção esteja sempre atenta com os requisitos de segurança do sistema, com a integridade dos dados, com a escalabilidade, etc. Isso é necessário para evitar algum problema no sistema, como perder dados no sistema, falhas de segurança que possam gerar que terceiros interceptem algum dado importante ou até mesmo que o sistema pare de funcionar, o que pode gerar um prejuízo não apenas de tempo de trabalho, mas um prejuízo financeiro para a empresa também. Essa manutenção nem sempre é feita como deveria, porém, deve-se sempre se preocupar com ela, já que qualquer coisa que possa afetar o funcionamento correto do sistema pode gerar grandes problemas para a empresa.

Sempre se deve realizar um planejamento antes de realizar qualquer alteração em um sistema, para que o mesmo não sofra algum dano em alguma funcionalidade. Por isso que softwares com uma documentação bem elaborada, descrevendo tudo o que ocorre e como foi feito o sistema, tem uma maior vantagem e um menor custo na

hora de realizar uma manutenção, já que se torna mais fácil realizar um planejamento para alterar alguma parte do sistema sem que essa alteração possa afetar outra parte, podendo causar um maior custo para corrigir todos os erros que podem ser gerados durante a manutenção.

Em sistemas que não possuem documentação, o custo para realizar alguma manutenção pode ser dobrado dependendo da complexidade, onde gasta-se um bom tempo tentando entender como o sistema funciona primeiramente, para depois realizar alguma alteração.

8. MODELO MOA

A intenção desse modelo é facilitar o processo de otimização, trazendo os passos que devem ser seguidos para que se possa ter um código mais limpo e com um funcionamento bem melhor, em questão de tempo de execução e em questão de ser mais leve, sem ocupar processamento e espaço de memória desnecessária.

Esse modelo deve ser aplicado durante todo o processo de codificação do software, trazendo melhorias contínuas a cada etapa da codificação. O modelo MOA deve ser implementado juntamente com a metodologia Scrum, que será explicado com mais detalhes no decorrer da pesquisa, para que haja um controle sobre o que está sendo feito e para que haja disciplina entres os membros da equipe de desenvolvimento, pois do contrário ele ficaria à mercê da boa vontade do programador em respeita-lo ou não.

Esse modelo deve seguir duas etapas:

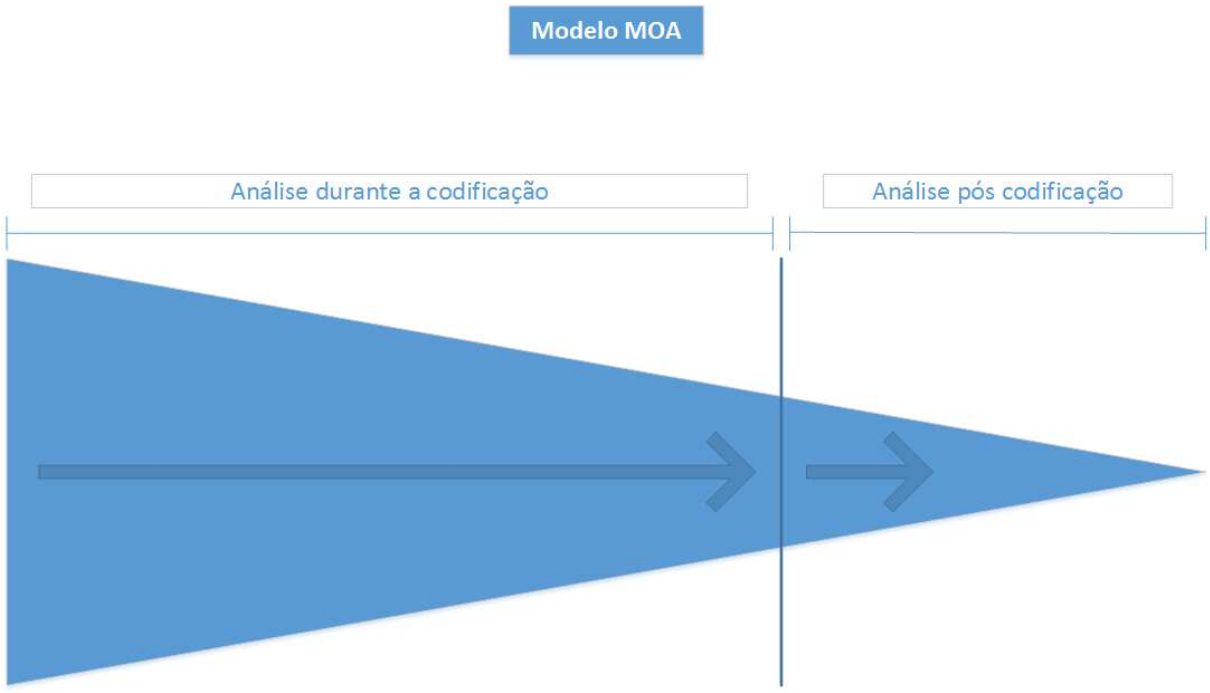
- Análise durante a codificação;
- Análise pós-codificação;

O modelo MOA é representado pelo diagrama da figura 1, logo abaixo, onde ele é representado por um triângulo deitado, sendo que as setas indicam o fluxo da codificação do projeto.

O diagrama é representado desta maneira, de forma que seja associado com o projeto inteiro em si, onde no começo do projeto, temos todo ele para ser codificado

e otimizado, para garantir um resultado com o melhor desempenho e qualidade possível, e conforme o fluxo vai seguindo em frente, é sinal de que o modelo está sendo respeitado e que conforme vai chegando ao final do projeto, a codificação já deve estar funcionando corretamente e otimizada.

Figura 1 – Diagrama do modelo MOA



Fonte: Autor próprio, 2015.

Esse diagrama é dividido em duas partes que representam as etapas do MOA que devem ser seguidas, sendo que a primeira, a análise durante a codificação, é a que consome mais tempo, pois é onde toda a codificação está sendo desenvolvida, e é ali que vamos garantir que o modelo MOA seja aplicado corretamente, pois se o problema não estiver sendo reduzido conforme o diagrama é sinal que o MOA não está sendo aplicado.

Nesta etapa é onde são realizadas as reuniões diárias, utilizando o Scrum, para saber como está sendo aplicado o modelo MOA, se ele está sendo respeitado corretamente e se está ocorrendo algum problema durante a codificação. E é nesta

etapa que todos esses pontos devem ser resolvidos para que não se torne um problema maior depois, desviando assim do objetivo do modelo MOA.

Na segunda parte do diagrama é onde ocorre a análise pós-codificação, que é representada por uma parte menor do triângulo até seu vértice, onde significa que o projeto está pronto. Esta deve ser a etapa que consome o menor tempo na aplicação do modelo MOA, sendo que nela a codificação já deve estar pronta, para que se possa fazer uma revisão de todo o código e realizar as alterações necessárias, sendo que elas devem ser mínimas ou nem existirem.

Assim, passado por estas duas partes do diagrama, temos a garantia de que o código escrito está otimizado, documentado e legível, de forma que ele possa ser reutilizado novamente e que qualquer manutenção necessária poderá ser realizada sem exigir muito esforço do programador.

Esse modelo busca solucionar os problemas que comumente podem ocorrer durante o período da codificação, que são os de produzir um código claro, conciso, legível e otimizado. Todos os programadores e membros envolvidos na equipe já devem saber que um algoritmo pode ser escrito de várias maneiras, e que no final, o resultado visual apresentado para o usuário seja igual, independente de como está escrito, mas se formos analisar isso a fundo, vemos que o desempenho entre esses códigos pode mudar de diversas maneiras e que sua legibilidade seja diferente também. Com isso, vemos que há uma grande necessidade de nos preocuparmos como será codificado o software, e é aí que entre o modelo MOA.

Podemos desenvolver todo o sistema sem nos preocuparmos como ele está escrito, mas apenas se está funcionando corretamente, porém, teríamos que analisar todo ele novamente para aplicar as melhorias, o que seria uma tarefa exaustiva. Mas se utilizarmos o modelo MOA, isso seria mais fácil e ágil, pois faríamos essa análise durante todo o processo de desenvolvimento, acompanhando cada passo do programador. No final apenas nos restaria fazer uma última verificação, para ter certeza que estaria tudo funcionando corretamente e da melhor maneira possível.

8.1. Análise durante a codificação

Essa etapa é a mais importante do modelo MOA, pois ela deve ser seguida respeitando as suas normas e regras para que se tenha um algoritmo de qualidade e para que não seja gasto muito tempo revendo todo o código novamente, gastando assim muito mais tempo na finalização do projeto e correndo o risco de alguma alteração comprometer o objetivo do algoritmo.

O gerente do projeto deve estar atento para analisar o desempenho dos programadores no quesito de respeitar as normas e regras do modelo MOA. Essa análise deve ser realizada sem nenhuma exceção, para que se garanta que no final do projeto apenas pequeno ajustes deverão ser realizados ou nenhum, o que seria preferível, pois se no final do projeto grandes ajustes forem realizados, é sinal de que alguma parte envolvida na implementação do modelo MOA não fez a sua parte de maneira correta.

Para que essa análise ocorra de maneira correta e mais eficaz, deve ser implementado nessa fase a metodologia ágil Scrum. Essa metodologia será responsável por realizar reuniões diárias, chamadas de Daily Scrum, com duração máxima de quinze minutos, onde deverão estar presentes todas as partes envolvidas no desenvolvimento do projeto. Ali deverão ser discutidos os pontos importantes do modelo MOA, onde cada integrante irá falar se está respeitando as regras impostas pelo modelo e se algo o está impedindo de respeitá-las.

O foco dessa reunião diária no modelo MOA é saber se o programador está respeitando os padrões propostos, e não apenas se ele está cumprindo com o seu cronograma de desenvolvimento. Esse deve ser um ponto muito importante que deve ser lembrado toda vez que o modelo MOA for implementado, pois do contrário essa reunião não irá atingir o seu objetivo. Se o programador estiver respeitando os padrões estabelecidos pelo modelo, automaticamente ele estará gerando uma melhor qualidade para o projeto no qual está codificando.

Para que essa reunião aconteça de forma eficaz, todos os envolvidos devem estar cientes das exigências do modelo, para que não ocorram divergências no assunto. O modelo deve ser exemplificado a cada início do projeto de uma forma transparente, facilitando assim o entendimento de todos, e principalmente, dos programadores, pois eles são quem atuarão com mais frequência aplicando essas técnicas.

Antes do início de um novo projeto de codificação devem ser avaliados os perfis dos integrantes da equipe, fazendo assim um levantamento de quem já participou na implementação do modelo em outras ocasiões, e também de quem nunca participou de uma implementação. Com base nos dados que forem obtidos com esse levantamento de perfis devemos avaliar a experiência que cada um supostamente deve ter com o modelo, nos atentando assim em quais pontos cada profissional tem maior facilidade e maior dificuldade na hora da implementação.

Profissionais que já tem experiência com o modelo poderão ficar isentos ou não de uma breve visão sobre o modelo MOA, isso vai depender da análise do gerente do projeto. Demais profissionais deverão receber um treinamento, mesmo que na teoria, de como são as regras e como elas deverão ser aplicadas.

A experiência do programador vai ser levada muito em conta nesta etapa também, pois o programador deverá estar atento aos padrões conforme ele for escrevendo o código, para que ele não precise escrever um bloco de código e depois parar para analisar se o mesmo está correto ou não.

Programadores com experiência conseguirão desenvolver um código com qualidade e com tempo reduzido, pois eles já escreverão o código da maneira correta, sem necessitar de uma revisão a cada bloco de código que escrever.

Nestas reuniões diárias, deverá ser discutido de forma superficial se estão sendo aplicadas corretamente as etapas do modelo MOA, para que se possa ter um controle que durante o processo de codificação não esteja sendo deixado de lado o modelo. Caso algum programador esteja dificuldades em respeitar as regras do modelo MOA, deve ser discutida detalhadamente com ele em uma reunião a parte, para não comprometer o prazo da reunião diária, o que está acontecendo, para que não se torne um problema no final do projeto.

Esse é um ponto muito importante que o Scrum Master deve respeitar sempre, pois se o prazo da reunião for estendido para discutir as dificuldades de todos detalhadamente, a reunião deixará de ser uma metodologia ágil, quebrando a regra do Scrum, e isso pode comprometer os prazos do projeto também, pois a dificuldade de uma pessoa da equipe seria discutida com todos os outros membros juntos, ao invés de eles já estarem focados na codificação do projeto.

Nessa reunião entre o gerente e o programador devem ser discutidos todos os pontos que estão impedindo que o mesmo respeite as regras estabelecidas pelo modelo MOA. O gerente deverá solucionar as dúvidas e problemas relatados pelo programador, para que ele possa voltar ao trabalho sabendo que deve respeitar as normas, resolvendo assim o problema e evitando gasto de tempo desnecessário no final do projeto.

Todos os integrantes do time de desenvolvimento devem trabalhar e produzir de maneira eficaz e interligada, pois no final da primeira etapa, o produto deve possuir um mesmo padrão, independente de quem fez o quê. Caso haja muitos erros no final desta etapa, é sinal de que alguma equipe não fez o que foi previamente proposto pelo modelo MOA.

Os padrões que deverão ser utilizados pelo programador serão chamados de padrões de codificação, que serão listados abaixo. Através do código escrito e desses padrões, deverão ser analisadas todas as etapas dos algoritmos, para que se procurem todas as possíveis melhorias na codificação.

8.1.1. Padrões de codificação

Os padrões de codificação são estabelecidos para que se tenha um controle durante toda a etapa de codificação do projeto. Eles devem ser utilizados tanto na hora de se iniciar um projeto quanto na hora de se realizar uma manutenção, para garantir que todo projeto, seja ele de qual porte for, resulte em um código onde se possa entender realmente o que está sendo feito em cada trecho e para que ele atenda todos os recursos do hardware no quesito desempenho, aproveitando melhor cada utilização do mesmo.

Um dos principais pontos nos padrões de codificação são os padrões de legibilidade, que têm como principal objetivo, como o próprio nome diz, deixar o código o mais legível possível. Esses padrões tratam de toda a formatação do código, o que é aplicável a qualquer linguagem de programação, pois independente dos comandos utilizados, a estrutura visual deve permanecer igual.

Quanto mais qualidade se tiver com esses padrões, mais fácil será achar um erro no algoritmo, principalmente os erros lógicos, o que é mais difícil de resolver, já que a ferramenta que está sendo utilizada para a codificação não acusa esse erro, que pode ocorrer tanto durante sua codificação inicial, quanto durante uma manutenção futura.

Não podemos esquecer os padrões de documentação também, pois tem um alto grau de importância na codificação de um software. Através dele conseguimos entender o que o nosso código está fazendo, ao contrário de um código sem documentação alguma, o que se torna um martírio na hora de realizar alguma alteração posteriormente. Classes bem documentadas explicam seus objetivos antes mesmo de elas iniciarem, para que se possa ter uma visão geral do que está sendo escrito naquele bloco de código, enquanto que os comentários no decorrer do código explicam o que cada trecho de código está fazendo, ficando assim mais fácil de entender o que deve ocorrer na hora da execução do programa.

Os demais padrões visam à questão de desempenho durante a execução do programa. Com eles o algoritmo poderá realizar todos os processos necessários ocupando um menor tempo no processador e ocupando uma quantidade de memória menor, o que deixa o algoritmo mais eficiente durante a sua execução.

Logo abaixo, estamos citando os padrões que devem ser respeitados no momento da codificação, mas esse modelo também pode trazer uma flexibilidade na hora de sua implementação, na qual podem ser implementados outros padrões que ajudem a produzir um código melhor. Vale lembrar que os padrões pré-estabelecidos não podem ser descartados, todos devem ser implementados no momento da codificação, apenas podemos adicionar novos padrões, desde que eles sejam estabelecidos pelos responsáveis pelo projeto e que sejam aprovados pelo gerente.

Isso pode facilitar e ajudar na hora de sua implementação, já que a equipe que deseja implementar o modelo MOA não vai precisar ficar totalmente presa apenas no que está sendo descrito abaixo, mas pode ajustar de acordo com as necessidades e experiência da equipe de desenvolvimento.

Dentre os padrões de codificação, podemos dividi-los em dois conjuntos, sendo eles os padrões de legibilidade e documentação e os de desempenho.

Os padrões de legibilidade e documentação são os seguintes:

- Evitar reescrever códigos: sempre que possível devem ser utilizados métodos para evitar a reescrita de códigos iguais.
- Escrever comentários: comentários devem ser escritos para identificar o que o trecho de código faz, tornando a visualização do código bem melhor, facilitando a manutenção do código e ajudando na hora de encontrar possível falhas de lógica e bugs. Devem ser escritos da seguinte maneira:
    - Comentários sempre devem começar com letra maiúscula e terminar com um ponto final.
    - Não escrever na mesma linha que o código, mas sim em uma linha separada, acima do código que está sendo comentado.
    - Devem respeitar os mesmos padrões de códigos, não excedendo muitos caracteres na mesma linha. Recomenda-se aproximadamente 85 caracteres.
- Utilização de espaços em branco: facilitam a legibilidade do código, porém devemos nos atentar nos mesmos, pois espaços em branco devem ser utilizados apenas onde são permitidos, para que não ocorram erros de compilação. Espaços em branco devem ser utilizados da seguinte maneira:
    - Pular uma linha acima do comentário.
    - Pular uma linha antes da declaração return, exceto se a mesma estiver sozinha no bloco de código.
    - Pular uma linha depois do fechamento de uma chave.
    - Sempre que utilizar o operador “+” para concatenar cadeias de caracteres, deixar um único espaço antes e depois do operador.
- Utilização de identação: garante uma grande legibilidade no código, facilitando a agilidade na hora de fazer alterações e encontrar bugs, e principalmente na abertura e fechamento de chaves.
- Utilizar quebra de linhas: quando uma linha for ficar muito extensa a mesma deve ser quebrada e identeada, para que melhore a visualização do código. Recomenda-se aproximadamente 85 caracteres.
- Respeitar as convenções: devem ser utilizadas para que se torne mais fácil à codificação e uma futura manutenção. Devem ser utilizadas as seguintes convenções:

  - Nomes das classes sempre devem começar com letra maiúscula.
  - Nomes de classes, métodos e variáveis devem ser escritos sem acento e sem caracteres especiais, como, por exemplo, o “ç”.
  - Nomes das variáveis e métodos sempre devem começar com letra minúscula e caso haja mais palavras, as demais deverão ser escritas juntamente e com primeira letra maiúscula, exemplo: primeiroAtributo.
  - Utilizar nomes concisos para as variáveis, métodos e classes, não utilizar, por exemplo, o nome pA e sim primeiroAtributo para variáveis e métodos, e Biblioteca, por exemplo, para classes.
  - Variáveis constantes devem estar com todas as letras maiúsculas e separados os nomes pelo caractere underline (“_”), como, por exemplo, ao invés de escrever primeiroAtributo, escrever PRIMEIRO_ATRIBUTO.
  - Nomes dos métodos devem ser verbos, para explicar melhor o que cada um faz, por exemplo, escrever mostrarResultado, ao invés de resultado.
- Utilizar chaves em estruturas condicionais e loops: utilizar a chave mesmo quando apenas se tem uma linha, facilita uma melhor visualização na hora de uma manutenção e previne erros lógicos.
- Eliminar código morto: procure eliminar trechos do código que nunca serão utilizados.
- Utilizar métodos get e set: sempre utilizar como nome de um método quando métodos vão alterar o valor ou obtiver o valor de uma variável global. Exemplo: getSaldo() e setSaldo().

E os padrões de desempenho são os seguintes:

- Utilização de métodos estáticos: métodos que não dependem da classe onde se encontram devem ser declarados como estáticos, pois somente são carregados na memória durante sua execução, deixando mais espaço livre na memória.
- Evitar atributos estáticos: atributos estáticos são alocados na memória até o fim da execução do programa, ocupando espaço quando não estão sendo utilizados. Somente devem ser usados para casos específicos onde serão utilizados em toda execução do programa.

- Não declarar variáveis dentro do loop: variáveis devem ser declaradas fora do loop, para que não seja alocado um espaço na memória a cada execução do mesmo.
- Evite processamento desnecessário no loop for: não utilizar métodos em sua declaração, utilize os métodos antes de ele iniciar e apenas utilize variáveis e valores constantes em sua declaração.
- Evitar processamento desnecessário dentro de qualquer loop: evitar realizar cálculos em variáveis que sempre serão os mesmos, sendo assim, declarar antes do loop pode ser mais vantajoso.
- Evitar processamento desnecessário em variáveis: quando duas variáveis fazem o mesmo cálculo, pode-se fazer o cálculo em apenas uma e utilizar na outra o cálculo já realizado.
- Sempre utilize ferramentas de linguagens para capturar erros, como o try/catch, evitando assim o máximo possível que o sistema pare por causa de um erro.
- Utilizar modo simplificado para contas aritméticas, por exemplo, ao invés de escrever “primeiro = primeiro + segundo”, escreva “primeiro += segundo”. Isso é mais rápido, mesmo que minimamente, e vale também para divisão, multiplicação e subtração.
- Evitar negação em condições: isso gera dois processamentos, uma para negar o que está sendo descrito e outro para verificar se a condição é falsa ou verdadeira.

8.1.2. Exemplo de uma aplicação prática dos padrões do modelo MOA

Neste exemplo, é utilizado um código fonte de uma calculadora simples, desenvolvida na linguagem de programação C#, que realiza as operações de adição, subtração, multiplicação e divisão, com apenas dois números inteiros, positivos ou negativos.

Este programa é composto por três classes, sendo a classe Program, que é a classe principal, onde ela chama o método para exibir o menu principal, a classe Menu, que exibe o menu na tela com as opções de operações que o usuário pode

escolher, pede para o usuário entrar com os valores desejados e passa para os métodos da classe Calculadora, que é onde são realizadas as operações aritméticas, e após isso a classe Menu, através de um método, exibe o resultado na tela.

Abaixo serão provados que os padrões do modelo MOA foram utilizados na construção desse programa, que apesar de ser pequeno, a teoria se aplica da mesma forma. Não foi utilizado um algoritmo mais extenso para facilitar o aprendizado de quem não está acostumado com os padrões do modelo MOA, pois com pequenos blocos de código, é mais fácil de compreender o que está querendo ser passado para o leitor.

O código está anexado no final desta pesquisa, para que o leitor possa acompanhá-lo durante a explicação do mesmo.

Vamos analisar as classes agora, vemos que todas as classes possuem uma documentação antes de iniciar a classe, o que é essencial para saber o que a classe em questão deve fazer, facilitando assim na hora de uma manutenção. Com isso concluímos que uma das regras já foi respeitada.

As classes possuem comentários, que também servem como uma documentação no programa, e eles são escritos de forma que começam com letra maiúscula, terminem com ponto final, descrevam bem o que o bloco de código abaixo faz, e pulam uma linha antes de escrever o comentário. Eles também respeitam as regras do número aproximado de caracteres que uma linha deve conter, respeitando assim a mais uma regra.

Os nomes de variáveis e objetos foram escritos de maneira que sejam claros e concisos, descrevendo corretamente o que cada um faz. Eles também são escritos sem acentos e sem caracteres especiais, e de acordo com suas convenções, respeitando assim a mais uma regra.

Os nomes dos métodos também representam clareza e concisão, mostrando assim, objetivamente, o que ele cada um faz. Seus primeiros nomes são verbos, e respeitam as convenções também, respeitando assim mais uma regra.

Na classe Calculadora, possuímos dois métodos para alterar e obter o valor da variável resultadoOperacao, que são declarados como getResultado e setResultado, de acordo com mais uma regra.

O operador “+” está seguido de um espaço em branco entre a concatenação de duas cadeias de caracteres, facilitando a visualização e respeitando a mais uma regra.

Em todos os blocos de códigos que estão sujeitos a erros, está sendo feito um tratamento de erros com as cláusulas try/catch, evitando assim que o programa pare caso algum erro ocorrer, como é o caso de dividir um valor por zero ou inserir valores sem serem inteiros, nesse exemplo, respeitando assim a mais uma regra.

Na condição if, que temos assim que apertarmos a opção cinco para sair do loop, temos apenas uma linha de código, mas mesmo assim são utilizadas as chaves para isolar o bloco de código, prevenindo assim erros em manutenções futuras e respeitando mais uma regra.

Uma regra muito importante e que é respeitada em todas as classes é a de que todas as linhas de código estão identadas corretamente, facilitando assim a legibilidade do código.

Outro ponto importante no qual está sendo respeitado é em evitar a reescrita de código, como podemos observar na classe Menu, onde temos o método exibirResultado, o que evita que em cada case do switch tenhamos que escrever o mesmo código, respeitando assim a mais uma regra do MOA.

Vamos mostrar outros exemplos de algoritmos que estão respeitando os padrões de desempenho estabelecidos pelo modelo MOA.

No primeiro exemplo estamos mostrando uma variável sendo declarada dentro do loop while, o que está errado, já que o correto seria declará-la antes de iniciar o loop, como mostrado na figura 2. Isso faz com que a variável não tenha que ser declarada a cada vez que o loop for executado.

Figura 2 – Não declarar variáveis dentro do loop

```
// Exemplo 1: não declarar variáveis dentro do loop.

// Incorreto.
int x = 0;

while (x < 10)
{
    int y = x + 2;

    x++;
}

// Correto.
int x = 0;
int y = 0;

while (x < 10)
{
    y = x + 2;

    x++;
}
```

Fonte: Autor próprio, 2015.

No segundo exemplo está sendo executada uma chamada de método na declaração do loop for, o que está errado, pois a cada laço ele terá que chamar o método novamente. O correto é declarar uma variável antes de iniciar o loop contendo a chamada de método para obter o valor, como está sendo mostrado no trecho de código da figura 3.

Figura 3 – Evitar processamento desnecessário no loop for

```
// Exemplo 2: evite processamento desnecessário no loop for.

// Incorreto.
Conta conta = new Conta();
int i = 0;

for (i = 0; i < conta.getValor(); x++)
{

}

// Correto.
Conta conta = new Conta();
int valor = conta.getValor();
int i = 0;

for (i = 0; i < valor; x++)
{

}
```

Fonte: Autor próprio, 2015.

No terceiro exemplo está sendo feita uma comparação entre uma condição if, onde primeiramente mostramos que a condição está sendo negada, o que gera um maior processamento, já que a condição precisa primeiramente negar para depois testar, ao contrário da outra que ela já testa direto.

Figura 4 – Evitar negações em condições

```
// Exemplo 3: evitar negações em condições.

// Incorreto.
if (!(x > 10))
{

}

// Correto.
if (x <= 10)
{

}
```

Fonte: Autor próprio, 2015.

No quarto exemplo estamos fazendo um comparativo entre uma soma com simplificação e outra sem. Apesar de o processamento ser minimamente diferente, devemos optar pela soma simplificada.

Figura 5 – Utilizar modo simplificado para contas aritméticas

```
// Exemplo 4: utilizar modo simplificado para contas aritméticas.

// Incorreto.
int x = 0;

x = x + 1;

// Correto.
int x = 0;

x += 1;
```

Fonte: Autor próprio, 2015.

No quinto exemplo está sendo mostrado um processamento desnecessário, onde duas variáveis estão realizando a mesma multiplicação. Isso pode gerar um aumento significativo no processamento. Para resolver isso, criamos uma variável chamada de temp para realizar e armazenar essa multiplicação. Com isso as demais variáveis estão apenas recebendo o valor e não calculando eles novamente, como podemos ver no código da figura 6.

Figura 6 – Evitar processamento desnecessário em variáveis

```
// Exemplo 5: evitar processamento desnecessário em variáveis.

// Incorreto.
int x = 0;
int a = 0;
int b = 0;
int exibirPrimeiro = 0;
int exibirSegundo = 0;

while (x < 10)
{
    a = 4 * 10;
    exibirPrimeiro = a + (20 + x);

    Console.WriteLine(exibirPrimeiro);

    b = 4 * 10;
    exibirSegundo = b + (30 + x);

    Console.WriteLine(exibirSegundo);

    x++;
}

// Correto.
int x = 0;
int a = 0;
int b = 0;
int temp = 4 * 10;
int exibirPrimeiro = 0;
int exibirSegundo = 0;

while (x < 10)
{
    a = temp;
    exibirPrimeiro = a + (20 + x);

    Console.WriteLine(exibirPrimeiro);

    b = temp;
    exibirSegundo = b + (30 + x);

    Console.WriteLine(exibirSegundo);

    x++;
}
```

Fonte: Autor próprio, 2015.

No sexto exemplo, estamos mostrando um loop while, onde sempre executamos a mesma multiplicação. Nesse caso, nunca devemos deixar essa multiplicação dentro do loop, pois ele está gerando um processamento desnecessário. O correto é fazer como mostrado no da figura 7, onde essa conta é realizada e armazenada em uma variável fora do loop, apenas chamando ela dentro do loop quando necessário.

Figura 7 – Evitar processamento desnecessário dentro de qualquer loop

```
// Exemplo 6: evitar processamento desnecessário dentro de qualquer loop.

// Incorreto.
int a = 0;
int b = 0;
int x = 0;

while (x < 10)
{
    a = 4 * 10;
    b = a + x;
    x++;
}

// Correto.
int a = 4 * 10;
int b = 0;
int x = 0;

while (x < 10)
{
    b = a + x;
    x++;
}
```

Fonte: Autor próprio, 2015.

8.2. Análise pós-codificação

Essa deve ser a etapa mais rápida do modelo MOA, mas não a menos importante, pois a partir daqui o projeto de codificação estará pronto e se essa etapa não for realizada corretamente, falhas poderão passar despercebidas e ocasionar falta de desempenho na execução do software e gerar um código mais ilegível. Ela tem a função de revisar todo o algoritmo codificado pela equipe de desenvolvimento,

analisando todos os códigos para ver se estão de acordo com os padrões de codificação.

Essa etapa funcionaria basicamente como se fosse um checklist, onde aqui apenas serão realizadas alterações que passaram despercebidas durante as análises anteriores, mas vale lembrar que essas alterações devem ser mínimas ou nem existirem, já que tudo foi feito e analisado seguindo os mesmos padrões anteriormente.

Esta etapa deve ocorrer da seguinte maneira: toda a equipe de desenvolvimento já deve ter acabado completamente a codificação do software e o mesmo já deve estar funcionando corretamente, pois se esta etapa for realizada antes dos testes, o software final corre o risco de ficar com algumas falhas no quesito de seguir os padrões do modelo MOA.

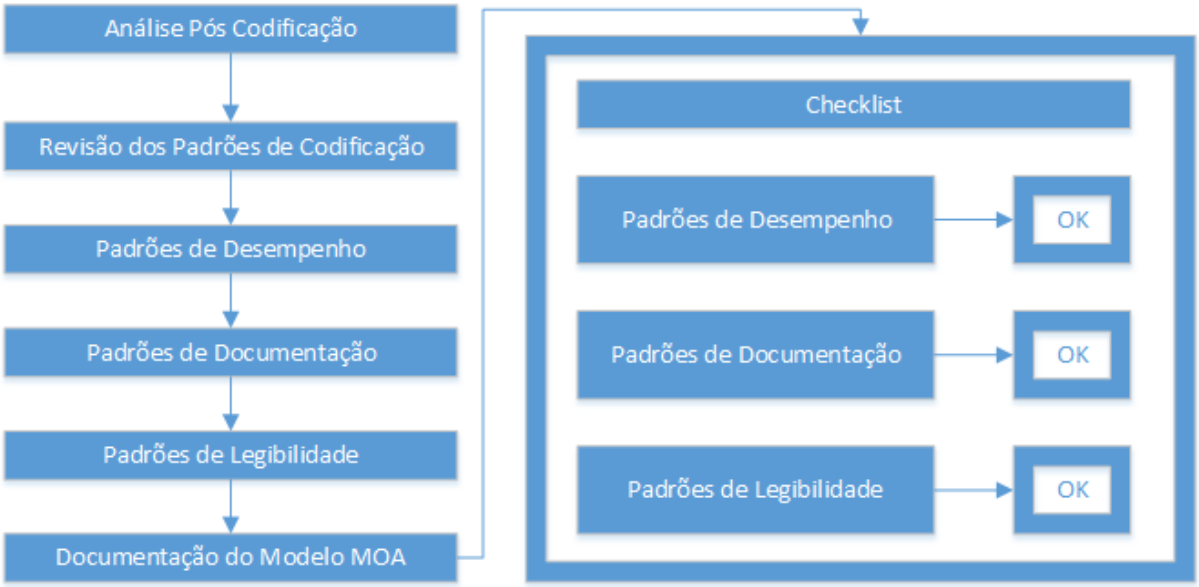
Um exemplo para tornar este ponto mais fácil de entender seria o seguinte, você codifica todo o software e testam as partes ou módulos dele individualmente, aí você realiza a segunda etapa do modelo MOA. Todo o código vai funcionar perfeitamente até o momento e otimizado, mas quando você vai integrar os módulos talvez você precise realizar alguma alteração no código para que ele funcione corretamente e sem falhas. Nesta ocasião dois problemas podem surgir, o primeiro seria que você realizasse novamente a segunda etapa do MOA, fazendo assim o serviço duas vezes, o que não seria viável. O segundo problema seria de você não aplicar mais a segunda etapa do MOA, podendo deixar passar assim alguma falha na hora da alteração do código. Por isso existe a importância de você testar todo o software antes de realizar a segunda etapa.

A segunda etapa funciona como um “selo” para o software, pois depois de realizada, a próxima fase já deve ser da implementação do software, onde tudo já está pronto, verificado e testado.

Para que essa etapa funcione de maneira eficaz, devemos dividir ela em três etapas que a equipe de desenvolvimento deverá seguir, sendo elas:

- Verificação dos padrões de desempenho.
- Verificação dos padrões de documentação;
- Verificação dos padrões de legibilidade;

Figura 8 – Análise pós-codificação



Fonte: Autor próprio, 2015.

Um ponto de extrema importância que deve ser levado em conta na hora de realizar esta etapa é o de que o gerente do projeto deve alternar as equipes de desenvolvimento para que ninguém verifique o próprio código que escreveu, pois isso pode causar falhas nesta etapa, já que os programadores ficam “viciados” na maneira de produzir o código, tornando algo automático e de difícil percepção de erros.

As verificações devem ser realizadas uma por vez e na sequência descrita, para que o foco permaneça apenas na que está sendo realizada, evitando assim possíveis falhas. Todas devem ser feitas na sequência acima, porque se, por exemplo, mudarmos algo nos padrões de desempenho, na hora de verificarmos os padrões de legibilidade, podemos verificar falhas no trecho alterado, o que não ocorreria se fosse feito ao contrário, já que os padrões de legibilidade não seriam aplicados novamente.

Isso pode gerar uma questão como a seguinte: eu estou ciente dos padrões, então por que eu não perceberia que está faltando mexer em algo? Pois bem, respondendo a essa questão, poderíamos perceber que algo está errado, mas de acordo com os prazos do projeto, não podemos ficar indo e voltando toda hora em um

mesmo lugar, devemos fazer isso de maneira padronizada para termos um controle total no que está sendo realizado.

O mesmo ocorre na documentação, devemos primeiramente ver se está tudo documentado de maneira correta, para somente depois passarmos para a fase de legibilidade, porque senão a documentação poderia estar confusa e fora da ordem correta, o que prejudicaria em uma futura manutenção.

No decorrer da análise, devem ser geradas documentações que comprovem quem alterou o código e o que alterou, pois, caso seja necessário realizar alguma alteração, deve ser documentado para que se saiba o que foi necessário mudar no código.

Essa documentação deve ser composta pelos seguintes itens:

- Nome da equipe,
- Nome completo do integrante,
- Nome da classe alterada,
- Número da linha alterada (inicial - final).

Tabela 1 – Planilha de documentação de alterações no código

Nome da equipe	Nome completo do integrante	Nome da classe	Linha (inicial - final)

Fonte: Autor próprio, 2015.

Essas documentações devem ser feitas sem exceções, para que no final elas fiquem registradas junto com as demais documentações do projeto. Elas servem como base para saber tudo o que foi alterado e por quem. Elas devem ser escritas em um formato de planilha, como na figura 3.

Abaixo vamos ver uma aplicação prática de como esta planilha deve ser utilizada nesta etapa do modelo MOA.

8.3. Aplicação do modelo MOA em um projeto

Neste capítulo, vou descrever um case abstrato onde o modelo MOA está sendo aplicado. Apesar de ser apenas uma ilustração do modelo, ao ser aplicado diretamente em um projeto de um software, seu funcionamento deverá ser igual ao mostrado aqui.

Vamos definir então, como funcionaria a aplicação do modelo MOA em um projeto.

Um cliente contratou uma empresa x para desenvolver um sistema para a sua empresa, para que certos departamentos da sua empresa tenham todo o seu processo informatizado. O cliente definiu que quer informatizar os processos de quatro departamentos, sendo eles, o financeiro, o de vendas, o de almoxarifado e o de produção. Após isso, foram definidos quais serão os requisitos que o sistema deve conter, para que ele atenda todas as necessidades do cliente. Para que isso ocorresse de forma adequada foram usadas técnicas de engenharia de software e engenharia de requisitos, evitando que a empresa que está desenvolvendo o sistema não desenvolva um sistema no qual estaria fora das necessidades do cliente.

Após esta etapa, foi iniciada a etapa de projeto do software, onde ele será todo projetado conforme os requisitos e restrições obtidos na etapa anterior. Esta etapa será responsável por projetar desde a interface com o usuário, até suas funcionalidades e acessos ao banco de dados.

Estas etapas não serão discutidas com mais detalhes porque o foco está apenas na etapa de codificação do software, mas, para que esta etapa ocorra de forma adequada, devem ser realizadas todas as etapas anteriores, evitando que problemas no decorrer do projeto possam comprometer o sistema e o prazo de entrega.

Depois de todas estas etapas prontas, vamos iniciar a etapa de codificação do software, que é onde o modelo MOA será aplicado. A partir deste ponto, o gerente do projeto já escolheu ou contratou uma equipe de programadores. Antes de iniciar a etapa de codificação, o gerente irá certificar de que os integrantes dessa equipe

conheçam o funcionamento do modelo MOA, para que assim o mesmo seja aplicado corretamente ao iniciar a codificação. Para que isso ocorra corretamente, este gerente irá fazer uma reunião com todas as equipes, explicando a fundamentação do modelo MOA, quais são seus objetivos e porque ele está sendo implementado no projeto. O gerente irá explicar detalhadamente como as duas etapas do modelo devem ser aplicadas, de forma que nenhum passo seja ignorado na hora de sua aplicação.

Após isso, irão ser esclarecidas todas as dúvidas dos integrantes da equipe, para certificar que nada foi entendido errado e que nada será ignorado. Suponhamos que um dos programadores está com dúvida de quando deve ser implementado as técnicas do modelo MOA, se elas devem ser implementadas ao final de uma parte do código ou devem ser aplicadas enquanto já está sendo escrito o código. Esta dúvida deverá ser esclarecida imediatamente para ele, evitando futuros erros.

Inicia-se então a etapa de codificação do sistema, onde a equipe será dividida em quatro subequipes, uma para cada módulo do sistema. Ao iniciar esta etapa, inicia-se a primeira etapa do modelo MOA, a etapa de análise durante a codificação.

Antes que cada integrante comece a fazer o seu trabalho, será realizada uma reunião, com duração máxima de quinze minutos, para ser discutido o que cada um vai fazer. Nesta primeira reunião não haverá muito que ser discutido, já que cada um praticamente já sabe o que deve fazer. Na reunião do dia seguinte e de todos os dias do projeto, essa reunião será realizada sem nenhuma falta, como definido pelo modelo MOA, sendo discutidos como está o andamento do projeto e tirando algumas dúvidas. Vamos citar um exemplo: um programador de uma equipe disse que no dia anterior realizou todo o seu cronograma corretamente, mas disse que está com dificuldades em aplicar os padrões do modelo MOA na documentação. Neste caso, o gerente do projeto, para não comprometer a reunião, pediu para que o mesmo aguardasse o final da reunião para que ele pudesse esclarecer sua dúvida com mais detalhes.

Terminada a reunião, como não surgiu mais nenhuma dúvida complexa, todos voltaram para seu setor e começaram a realizar o seu trabalho normalmente. Então foi chamado o programador que estava com dúvidas para uma reunião particular, onde foram discutidos os pontos que levantaram dúvidas pelo programador. Com isso, todas as dúvidas foram esclarecidas para que ele pudesse voltar normalmente ao seu trabalho.

Após ser codificado todo o software, sempre sendo certificado de que tudo está sendo feito corretamente, através das reuniões diárias, entramos na segunda etapa do modelo MOA. Para que esta etapa seja realizada corretamente, as equipes devem ser alternadas, para que nenhuma equipe fique responsável em verificar o próprio código que escreveu. As equipes deverão começar a analisar o código escrito na ordem descrita pelo modelo MOA, então elas analisarão primeiramente os padrões de desempenho, para ver se tudo que foi desenvolvido pela equipe anterior está correto. Caso seja analisado que algo está errado, e que possa causar perda de desempenho ou um consumo maior na hora de sua execução, devem ser alterados os trechos de códigos, porém essa alteração deverá ser documentada pela equipe responsável.

Na figura 4 estamos exemplificando como deve ser preenchida a planilha de documentação, onde o integrante Greison Fernando Bassan faz parte da equipe denominada de equipe 2, e ele constatou que foi necessário realizar algumas alterações nos padrões de desempenho do código na classe Financeiro, começando a partir da linha 301 até a linha 305.

Tabela 2 – Utilização da planilha de documentação de alterações no código

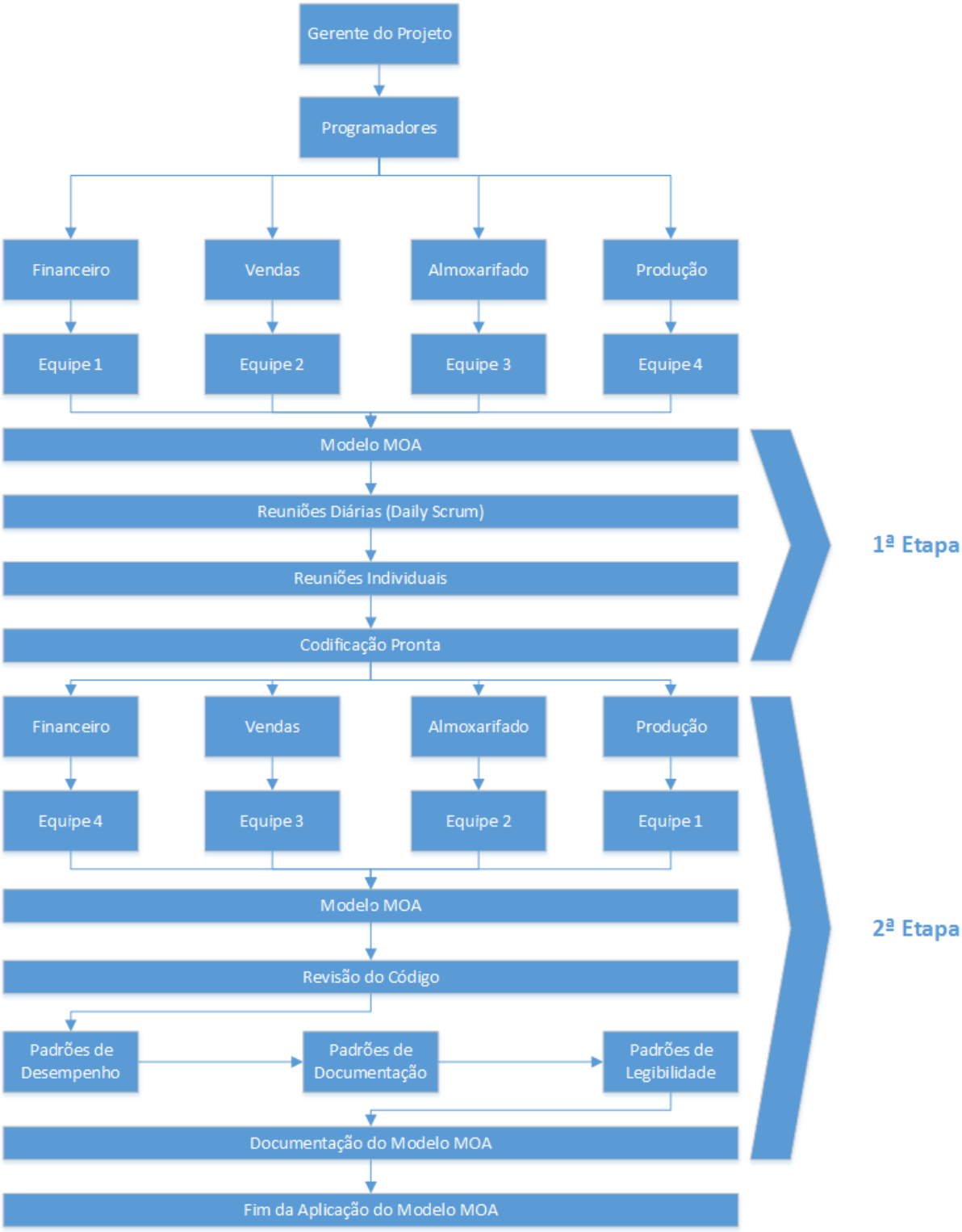
Nome da equipe	**Nome completo do integrante**	**Nome da classe**	**Linha (inicial-final)**
Equipe 2	Greison Fernando Bassan	Financeiro	301-305
Equipe 1	Yan Minatel Silva	Vendas	605-615
Equipe 4	Bruno Passos	Produção	2110-2120
Equipe 3	Pablo Luiz Oliveira Cunha	Almoxarifado	951-953

Fonte: Autor próprio, 2015.

Isso deverá realizado pela mesma equipe nos padrões de documentação e nos padrões de legibilidade, seguindo os mesmos padrões.

Com isso, o modelo MOA torna este sistema mais confiável e com um melhor desempenho, vendo que se ele não fosse aplicado no desenvolvimento do software, não teríamos um padrão pré-estabelecido para codificar o sistema de maneira que controlasse como o mesmo estava sendo escrito, e se ele estaria sendo escrito com os melhores recursos disponíveis para que se tenha um desempenho melhor durante a sua execução e com um código bem melhor para se trabalhar.

Figura 9 – Fluxograma da aplicação prática utilizando o modelo MOA



Fonte: Autor próprio, 2015

8.4. Vantagens na reutilização de algoritmos desenvolvidos com modelo MOA

Utilizando o modelo MOA, conseguimos ter algoritmos padronizados, de forma que tenham um maior desempenho e que tenham uma maior legibilidade, facilitando assim o seu entendimento. Com isso, qualquer equipe que já tenha participado de um projeto utilizando o modelo MOA, já vai estar adaptada em como são os padrões que devem ser respeitados na hora da codificação, e já saberá que os códigos produzidos com esses padrões já foram revisados duas vezes, no quesito em respeitar as regras do modelo MOA, uma vez na hora da codificação, que é onde o programador fica focado em tentar produzir um código totalmente otimizado e legível, e depois que o código já está pronto, onde são revisados todos os códigos escritos.

Como vocês já podem ver, está apenas sendo tratado da implementação do modelo MOA, e não dos testes de como o código está funcionando, se ele está obtendo o resultado esperado para solucionar determinado problema, ou se falta alguma lógica que funcionaria melhor, mas fica claro que se já temos um código pronto e que já foi utilizado em outro sistema, é sinal de que já passou por diversos testes e que está funcionando conforme o esperado.

Com isso, sabemos que o código está funcionando normalmente, e que temos um padrão estabelecido para ele, então podemos reutilizar esse código para desenvolvimento de novos algoritmos, que utilizem as mesmas funções que já foram codificadas anteriormente.

Durante um novo processo de desenvolvimento, seja ele com a mesma equipe que desenvolveu algum algoritmo anterior, que poderá ser utilizado novamente, ou com uma nova equipe, poderá ser implementado com mais facilidade o algoritmo já existente, pois ele conta com uma estrutura legível, documentada e otimizada. O programador não terá que sofrer para entender como funciona o código, tendo que analisar linha por linha para saber como ele funciona, basta ele ter uma experiência na linguagem utilizada que ele já conseguirá interpretar como funciona, ou seja, possuindo um algoritmo legível e com documentação, qualquer sistema que necessitar utilizar um mesmo algoritmo, poderá implementá-lo sem muito custo.

Isso já não ocorreria se o algoritmo tivesse sido escrito da maneira como o programador quisesse, pois não existiria um padrão a ser seguido. A complexidade

em entender como o algoritmo funcionaria, às vezes não compensaria a sua reutilização, pois o tempo que seria gasto tentando entender seu funcionamento, poderia ser gasto desenvolvendo novamente a partir do zero. Claro que não podemos generalizar isso como sendo um fator que ocorre em todos os casos, já que programadores experientes podem desenvolver algoritmos que possam ser entendidos por outros programadores com o mesmo nível de experiência, mas não é a mesma coisa de se ter um padrão que todos utilizam.

8.5. Vantagens na manutenção de softwares desenvolvidos com modelo MOA

Com a utilização do modelo MOA, quando precisarmos realizar uma manutenção, seja ela de caráter corretiva, adaptativa, evolutiva ou preventiva, poderemos fazer isso com uma maior facilidade também, pois o código poderá ser interpretado por qualquer programador, para que não se torne um código com um estilo pessoal, onde somente quem escreveu irá entender. O código irá fornecer a documentação necessária, para que o programador consiga entender com mais agilidade como está funcionando aquele trecho código, pois o mesmo estará legível e documentado.

Em outras ocasiões o programador teria que perder muito mais tempo para tentar entender como funciona aquele código, sendo que muitos códigos podem não possuir documentação, o que dificulta e muito o entendimento.

No momento em que o programador está fazendo uma manutenção em um código, seja ele um código já padronizado com o modelo MOA, ou um código não padronizado, é recomendado que se aplique o modelo MOA, da mesma forma que ele é aplicado na hora de se desenvolver um novo algoritmo, facilitando assim qualquer alteração futura.

Durante a manutenção de um código, o correto seria aplicar as técnicas do modelo MOA em todo o código e não somente onde está sendo escrito um novo trecho, para que não se torne um código igual a qualquer outro código não padronizado, pois se apenas em pequenos trechos estiverem sendo aplicado o modelo MOA, não resolverá o problema como um todo. Claro que isso deve ocorrer

de acordo com o projeto de manutenção em que se está trabalhando, pois, o custo de se alterar todo um código já escrito, dependerá do que foi combinado previamente com cliente.

9. CONCLUSÃO

Com esta pesquisa, concluímos que durante a etapa de codificação de um projeto de software podemos ter diversos problemas se o processo não for padronizado, o que pode comprometer os prazos de entrega do software e gerar um código com uma qualidade inferior ao esperado.

Durante este processo então, esta pesquisa está sugerindo a implementação de um modelo, chamado de modelo MOA, que busca solucionar estes problemas gerados, onde todo o processo será controlado para que tenhamos um software padronizado e otimizado.

Além de este modelo controlar todo o processo de codificação de um novo software, ele pode ser implementado em uma manutenção, mesmo se o software não foi desenvolvido utilizando este modelo. Isso traz uma maior facilidade na hora de codificar e alterar novos softwares e também quando se deseja alterar softwares legados.

Este modelo também soluciona problemas de reutilização de códigos, já que códigos escritos com o modelo MOA já estarão padronizados, tornando assim mais fácil a reutilização dos mesmos em novos projetos de softwares, o que não ocorreria se eles tivessem sido codificados sem nenhum padrão a ser seguido.

Concluímos então que o modelo MOA resolve os problemas de padronização na etapa de codificação de software, sendo que ele tem como objetivo tornar o software otimizado, documentado e legível, através de um processo rigoroso onde ele busca abranger todos os pontos que podem gerar falhas no final do projeto e tratá-los de maneira com que tudo possa ser coordenado. Além disso, ele pode ser implementado em qualquer projeto de software, desde que os integrantes do projeto sigam corretamente as etapas do modelo, para garantir que os problemas sejam resolvidos de maneira eficaz.

9.1. Propostas futuras

Todos os problemas citados nesta pesquisa podem abrir diversas discussões e novos temas para que esta pesquisa possa ter continuação e para que este modelo sugerido possa ter um aperfeiçoamento ainda melhor. Abaixo serão apresentadas sugestões para que a pesquisa possa ter continuidade e para que ela possa cada vez se torna melhor e possuir um case mais forte.

- Aplicação prática deste modelo em um projeto de software: durante esta pesquisa foram discutidas e apresentadas técnicas para que a etapa de codificação se torne padronizada e melhor, porém isto foi apenas exemplificado na teoria. Sugiro que em um futuro trabalho isto seja apresentado de uma maneira prática durante o desenvolvimento de um novo projeto.
- Comparação entre o modelo MOA e modelos já existentes: realizar uma comparação através de resultados obtidos em aplicações práticas do modelo MOA e de outros modelos já existentes, como o modelo espiral, por exemplo, para que possamos testar a qualidade do modelo MOA quando se comparado a modelos já utilizados atualmente.
- Coletar mais dados sobre padrões de codificação: realizar uma pesquisa para coletar uma maior quantidade de padrões de codificação a serem pré-estabelecidos pelo modelo MOA.
- Definir mais modelos de documentação: fazer uma análise sobre tudo o que poderia ser documentado durante a aplicação do modelo MOA e gerar novos modelos de relatório para que possa ser registrado uma maior quantidade de dados durante todo o processo.
- Comparação entre reutilização de códigos escritos com o modelo MOA e códigos escritos sem a aplicação do modelo: coletar dados para realizar uma comparação entre projetos que reutilizam códigos escritos com o modelo MOA e com os demais, e verificar quais são as vantagens e desvantagens com isso.

REFERÊNCIAS

Alencar, W. Coding standards (Padrões de codificação). Disponível em: <http://blog.walkeralencar.com/archives/233> Acesso em 2 de agosto de 2015.

Bruno Nardini. 5 boas práticas para deixar o código otimizado. Disponível em: <http://www.brunonardini.com.br/desenv-back-end/5-boas-praticas-para-deixar-o-codigo-otimizado> Acesso em 13 de julho de 2015.

DesenvolvimentoAgil.com.br. Daily Scrum. Disponível em: <http://www.desenvolvimentoagil.com.br/scrum/daily_scrum/> Acesso em 20 de novembro de 2015.

DevMedia. Artigo Java Magazine 35 - Otimizando Código Java. Disponível em: <http://www.devmedia.com.br/artigo-java-magazine-35-otimizando-codigo-java/8958> Acesso em 20 de julho de 2015.

DevMedia. Manutenção de Software: Definições e Dificuldades - Artigo Revista SQL Magazine 86. Disponível em: <http://www.devmedia.com.br/manutencao-de-software-definicoes-e-dificuldades-artigo-revista-sql-magazine-86/20402> Acesso em 20 de outubro de 2015.

DevMedia. Otimização de aplicações Java ME. Disponível em: <http://www.devmedia.com.br/otimizacao-de-aplicacoes-java-me/6788> Acesso em 13 de outubro de 2015.

DevMedia. Padrões de Codificação. Disponível em: <http://www.devmedia.com.br/padroes-de-codificacao/16529> Acesso em 31 de julho de 2015.

Fabri, J. A. A qualidade do software passa pela padronização do código fonte. Disponível em: <https://engenhariasoftware.wordpress.com/2008/09/30/a-qualidade-do-software-passa-pela-padronizacao-do-codigo-fonte/> Acesso em 18 de outubro de 2015

LeandroMTR. Tipos de manutenção de software. Disponível em: <http://www.leandromtr.com/tipos-de-manutencao-de-software/> Acesso em 20 de outubro de 2015.

Macoratti, J. C. Padronização de código. Disponível em: <http://www.macoratti.net/13/09/net_pcod1.htm> Acesso em 18 de outubro de 2015

Michelazzo, P. A documentação de software. Disponível em: <http://imasters.com.br/artigo/4371/gerencia-de-ti/a-documentacao-de-software/> Acesso em 15 de outubro de 2015.

Microsoft. Convenções de codificação em C# (Guia de Programação em C#). Disponível em: <https://msdn.microsoft.com/pt-br/library/ff926074.aspx?f=255&MSPPError=-2147217396> Acesso em 1 de agosto de 2015.

Pedreiras, P. Sistemas de Tempo-real. Disponível em: <http://ppedreiras.av.it.pt/resources/str1112/docs/STR-10.pdf> Acesso em 10 de julho de 2015.

Pichiliani, M. Manutenção de código legado: por onde começar? Disponível em: <http://imasters.com.br/artigo/25067/desenvolvimento/manutencao-em-codigo-legado-por-onde-comecar/> Acesso em 22 de agosto de 2015.

Rangel, J. L. Compiladores. Disponível em: <http://www.cin.ufpe.br/~mvpm/Compiladores/otim.pdf> Acesso em 21 de julho de 2015.

Schwaber, K e Sutherland, J. Guia do Scrum. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf> Acesso em 20 de setembro de 2015.

Silva, R. P. M. Kernel Linux: Padrão de codificação. Disponível em: <http://www.embarcados.com.br/padrao-de-codificacao-kernel-linux/> Acesso em 14 de agosto de 2015.

SlideShare. Compiladores - Otimização de código Intermediário. Disponível em: <http://pt.slideshare.net/edsonmoretti/compiladores-otimizao-de-cdigo-intermedirio> Acesso em 11 de julho de 2015.

Souza, L. G. S. Dicas de boas práticas de codificação. Disponível em: <https://luizgustavoss.wordpress.com/2010/09/09/dicas-de-boas-praticas-de-codificacao/> Acesso em 7 de agosto de 2015

Symfony. Padrões de Codificação. Disponível em: <http://andreiabohner.org/symfony2docs/contributing/code/standards.html> Acesso em 1 de agosto de 2015.

WIKILIVROS, Construção de compiladores/Otimização de código. Disponível em: <http://pt.wikibooks.org/wiki/Constru%C3%A7%C3%A3o_de_compiladores/Otimiza%C3%A7%C3%A3o_de_c%C3%B3digo> Acesso em 11 de julho de 2015.

ANEXO

```
namespace TCC
{
    /// <summary>
    /// Está é a classe principal, na qual é instanciada a classe Menu para realizar
    /// a exibição do menu da calculadora simples.
    /// </summary>
    class Program
    {
        static void Main(string[] args)
        {
            Menu menu = new Menu();

            // Chama o método para exibir o menu na tela.
            menu.exibirMenu();
        }
    }
}
```

```
namespace TCC
{
    /// <summary>
    /// Essa classe exibe na tela o menu e pede para o usuário digitar dois valores
    /// que serão utilizados para fazer a operação. Essa classe também instancia a
    /// classe Calculadora, e passa os valores obtidos aqui para que ela realize as
    /// operações necessárias.
    /// </summary>
    class Menu
    {
        private Calculadora calculadora = new Calculadora();

        // Método que exibe o menu na tela.
        public void exibirMenu()
        {
            byte opcao = 0;

            int primeiroNumero = 0;
            int segundoNumero = 0;

            do
            {
                Console.Clear();

                Console.WriteLine("Operação com dois valores inteiros");

                // Pede para o usuário inserir um valor inteiro e caso ele
                // insira um valor incorreto, exibe uma mensagem de erro.
                try
                {
                    Console.Write("\n" + "Digite o primeiro número: ");
                    primeiroNumero = int.Parse(Console.ReadLine());
                }

                catch (FormatException e)
                {
                    Console.WriteLine("\n" + "Erro: " + e.Message);
                }
```

```
        // Pede para o usuário inserir outro valor inteiro e caso ele
        // insira um valor incorreto, exibe uma mensagem de erro.
        try
        {
            Console.Write("Digite o segundo número: ");
            segundoNumero = int.Parse(Console.ReadLine());
        }

        catch (FormatException e)
        {
            Console.WriteLine("\n" + "Erro: " + e.Message);
        }

        // Menu.
        // Fica na tela até o usuário pressionar o número 5 para sair.
        Console.WriteLine("\n" + "1 - Somar");
        Console.WriteLine("2 - Subtrair");
        Console.WriteLine("3 - Multiplicar");
        Console.WriteLine("4 - Dividir");
        Console.WriteLine("5 - Sair");

        // Pede para o usuário digitar o número da operação a ser realizada
        Console.Write("\n" + "Digite a operação que você deseja: ");
        opcao = byte.Parse(Console.ReadLine());

        // Chama os métodos da classe Calculadora para realizar a operação e
        // também chama o método exibirResultado para exibir o resultado.
        switch (opcao)
        {
            case 1:
                {
                    calculadora.somar(primeiroNumero, segundoNumero);
                    exibirResultado();
                    break;
                }
            case 2:
                {
                    calculadora.subtrair(primeiroNumero, segundoNumero);
                    exibirResultado();
                    break;
                }
            case 3:
                {
                    calculadora.multiplicar(primeiroNumero, segundoNumero);
                    exibirResultado();
                    break;
                }
            case 4:
                {
                    calculadora.dividir(primeiroNumero, segundoNumero);
                    exibirResultado();
                    break;
                }
            default:
                {
                    break;
                }
        }
    }
    while (opcao != 5);

    Console.Clear();
```

```
            if (opcao == 5)
            {
                Console.WriteLine("Pressione qualquer tecla para finalizar...");
            }

            Console.ReadKey();
        }

        // Método para exibir o resultado na tela e aguardar até o usuário
        // pressionar uma tecla para voltar ao menu inicial.
        private void exibirResultado()
        {
            Console.WriteLine("\n" + "O resultado da operação é: " +
calculadora.getResultado());

            Console.WriteLine("\n" + "Pressione qualquer tecla para continuar...");
            Console.ReadLine();
        }
    }
}


namespace TCC
{
    /// <summary>
    /// Essa classe é responsável por fazer as operações de soma, subtração,
multiplicação
    /// e divisão. Ela também possui um método para retornar o valor da variável onde
é
    /// armazenado o resultado da operação.
    /// </summary>
    class Calculadora
    {
        private int resultadoOperacao = 0;

        // Método para somar os dois valores de entrada e armazenar na
        // variável resultadoOperacao.
        public void somar(int primeiroNumero, int segundoNumero)
        {
            resultadoOperacao = primeiroNumero + segundoNumero;
        }

        // Método para subtrair os dois valores de entrada e armazenar na
        // variável resultadoOperacao.
        public void subtrair(int primeiroNumero, int segundoNumero)
        {
            resultadoOperacao = primeiroNumero - segundoNumero;
        }

        // Método para multiplicar os dois valores de entrada e armazenar na
        // variável resultadoOperacao.
        public void multiplicar(int primeiroNumero, int segundoNumero)
        {
            resultadoOperacao = primeiroNumero * segundoNumero;
        }

        // Método para dividir os dois valores de entrada e armazenar na
        // variável resultadoOperacao.
        // Exibe um erro caso a divisão for feita por 0.
        public void dividir(int primeiroNumero, int segundoNumero)
        {
```

```
            try
            {
                resultadoOperacao = primeiroNumero / segundoNumero;
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("\n" + "Erro: " + e.Message);
            }
        }

        // Método para retornar o valor da variável resultadoOperacao.
        public int getResultado()
        {
            return resultadoOperacao;
        }

        // Método para alterar o valor da variável resultadoOperacao.
        public void setResultado(int resultado)
        {
            resultadoOperacao = resultado;
        }
    }
}
```