Submetido em 18.05.2018. Aprovado em 17.07.2018

Arquitetura em Camadas com uso do Paradigma MVC e Processo Unificado na Programação de *Software* Orientado a Objetos

Layered Architecture using the MVC Paradigm and Unified Process in Object Oriented Software Programming

José Gladistone Rocha

Resumo

O objetivo desse artigo é apresentar fundamentos que apontem para que o uso das tecnologias a seguir apresentadas facilitem o desenvolvimento de projetos de SI. Isso possibilitará que o produto de *software* tenha melhor qualidade, desde a fase de projeto, seu gerenciamento, como na melhoria do sistema de TI, já na sua fase de produção. Desenvolvedores têm utilizado a abordagem Orientação a Objetos para construção do *software*. A necessidade de se utilizar outras tecnologias se torna imperiosa para efetivamente resultar em um sistema de Tecnologia da Informação com qualidade já que muitos sistemas têm tido problemas nesse quesito. Uma dessas tecnologias é o padrão arquitetural em camadas, o *Model-View-Control*, e para gerenciamento e metodologia do processo de *software* o *Rational Unified Process*. Ou seja, dois importantes assuntos para a programação em si e para o gerenciamento de *software* por meio de metodologias para esse fim. O que se percebe é que muitos desenvolvedores simplesmente não adotam minimamente essas tecnologias, sem falar de outras tão importantes como além das já citadas, como por exemplo a arquitetura de sistemas com *Extensible Markup Language*. O uso desse compêndio de tecnologias apontando os pontos mais importantes delas auxiliará os desenvolvedores pois tirarão os melhores proveitos dessas tecnologias.

Palavras-chave—Arquitetura de *Software*; MVC; Camadas; Engenharia de *Software*; Orientação a Objetos.

Abreviações—Engenharia de *Software* (ES); Orientação a Objetos (OO); Sistema de Informações (SI); *Model-View-Controller* (MVC); Processo Unificado (PU); Tecnologia da Informação (TI); *Unified Modeling Language* (UML).

Abstract

Submetido em 18.05.2018. Aprovado em 17.07.2018

The goal of this paper is to present fundamentals that point out the usage of the following technologies in order to facilitate the development of IS projects. This will enable the software product to have better quality, from the design phase, its management, and in the improvement of the IT system, since the production phase. Developers have used the Object Orientation approach to building software. The need to use other technologies becomes imperative to effectively result in a quality Information Technology system as many systems have had problems in this area. One of these technologies is the layered architectural standard, Model-View-Control, and for Rational Unified Process software process management and methodology. That is, two important subjects for programming itself and for software management through methodologies for this purpose. What is perceived is that many developers simply do not adopt these technologies minimally, not to mention others as important as those already mentioned, such as the systems architecture with Extensible Markup Language. The use of this compendium of technologies pointing out the most important topics that will help the developers because they will take the best advantage of these technologies.

Keywords: *Software Architecture;* MVC*; Software Engineering; Unified Process.*

1 INTRODUÇÃO

A Evolução da ES possibilitou a indústria de *software* entrar em um período de relativa maturidade (SOMMERVILLE, 2011). Além disso, o *software* tem se tornado um componente decisivo para muitos produtos de nossos dias, particularmente àqueles integrantes de um SI.

Essa evolução da ES intrinsecamente abriu espaço para que a arquitetura de *software* referente ao paradigma da OO também evoluísse com a disponibilização de uma série de *Design Pattners* como, por exemplo, os modelos em camadas, como o *Model-View-Controller* (MVC), *Facade*, *Singleton*, e muitos outros a serem utilizados na etapa de programação de um SI. Muitos dos padrões de projetos inseridos na arquitetura do *software* contribuem no desempenho do SI e facilitam manutenções futuras (ZÁRATE, HERNÁNDEZ, GONZÁLEZ, 2012).

Existem poucas referências na literatura quanto a utilização dos três fundamentos: da programação OO, com adoção da metodologia RUP e a utilização do padrão de projeto MVC.

Submetido em 18.05.2018. Aprovado em 17.07.2018

O objetivo desse trabalho é apresentar fundamentos que apontem para que o uso das tecnologias a seguir apresentadas facilitem o desenvolvimento de projetos de SI. Isso possibilitará que o produto de *software* tenha melhor qualidade.

Esse texto está organizado da seguinte forma: a Seção 2 trata dos principais conceitos utilizados nesse estudo; a Seção 3 apresenta a fundamentação teórica relacionada ao trabalho; a Seção 4 trata da utilização do padrão de projeto MVC no desenvolvimento de sistemas de TI; a Seção 5 é dedicado a comentar o paradigma de desenvolvimento orientando a objeto; a Seção 6 discorre sobre a utilização do Processo Unificado aplicado a programão OO e por fim a Seção 7 apresenta as conclusões e apontamentos para trabalhos futuros.

2 CONCEITOS UTILIZADOS

Para facilidade de compreensão do conteúdo deste artigo foram adotados os seguintes conceitos extraídos da literatura:

a) **Arquitetura**: trata destacadamente da organização do espaço e de seus elementos: em última instância, a arquitetura lidaria com qualquer problema de agenciamento, organização, estética e ordenamento de componentes em qualquer situação de arranjo espacial (WIKIPEDIA, 2017);

b) **Arquitetura de *software***: é um conjunto de decisões significativas sobre a organização de um sistema de *software*, a seleção dos elementos estruturais e suas interfaces pelas quais o sistema é composto, com seu comportamento, como especificado nas colaborações entre esses elementos, a composição desses elementos estruturais e comportamentais em subsistemas progressivamente maiores, e o estilo arquitetural que dirige essa organização – esses elementos e suas interfaces, suas colaborações e sua composição Booch e Jacobson apud Larman (1999).

c) **Arquitetura lógica**: é a organização em larga escala das classes de *software* em pacotes, subsistemas e camadas. É chamada de arquitetura lógica porque não há

decisão sobre como esses elementos são implantados pelos diferentes processos do sistema operacional ou pelos computadores físicos em uma rede.

d) **Camadas**: é um agrupamento de granularidade muito grossa de classes, pacotes ou subsistemas que tem responsabilidade coesiva sobre um tópico importante do sistema. Também são organizadas de modo que as “mais altas” solicitem serviços da “mais baixas” mas normalmente não vice-versa (LARMAN, 2007).

e) **Engenharia de *software***: aborda técnicas e processos para auxiliar desenvolvedores na atividade de construção de *software* Sharman,Sharman e Mehta apud Rocha (2012).

f) **Padrões de designer**: normalmente obrigatórias, são convenções, de designers personalizados, cuja função é permitir que se predeterminem, de modo consistente, as características do design da solução no suporte aos objetivos organizacionais otimizados para ambientes corporativos específicos.

g) **Padrões de projetos**: é um repertório tanto de princípios gerais quanto de soluções idiomáticas que guiam os desenvolvedores de TI na criação do *software*. Esses princípios e idiomas, se codificados em um formato estruturado, descrevendo o problema e a solução, além de serem devidamente denominados (LARMAN, 2007).

h) **Processo de *software***: é um conjunto de atividades, tarefas e ações realizadas na criação de algum produto de trabalho (PRESSMAN, 2011).

i) ***Rational Unified Process* (RUP):** é mais do que um *software* para auxiliar no desenvolvimento é uma metodologia de desenvolvimento, com uma estrutura formal e bem definida. Como qualquer metodologia, é composta de conceitos, práticas e regras (PISKE, 2003).

j) **Requisito**: é uma condição ou capacidade que um sistema tem que encontrar. (KRUCHTEN, 2003).

3 TRABALHOS RELACIONADOS

Submetido em 18.05.2018. Aprovado em 17.07.2018

À medida que o mundo real continua mudando, o sistema de *software* que o representa precisa ser continuamente mantido e evoluído. Desenvolver e manter um sistema de *software* tão evolutivo é obviamente difícil. Um processo bem disciplinado e uma boa notação de modelagem são essenciais para controlar as atividades na construção e documentação dos diferentes modelos obtidos em diferentes estágios do desenvolvimento de *software*. O *Rational Unified Process* (RUP) emergiu como um processo popular de desenvolvimento de *software*. A notação de modelagem, o RUP usa a *Unified Modeling Language* (UML), que é a linguagem de modelagem padrão de fato para o desenvolvimento de *software* em uma ampla gama de aplicativos (KRUCHTEN, 2003). O RUP promove várias práticas recomendadas, mas uma posição acima dos outros é a ideia de desenvolvimento orientado a casos de uso e iterativo. Na abordagem de RUP orientada por mecanismo de uso e iterativa, um desenvolvimento de sistema é organizado como uma série de miniprojetos curtos e de comprimento fixo chamados iterações, cada um para um pequeno número de casos de uso. Cada iteração inclui suas próprias atividades de análise, projeto, implementação e teste/verificação de requisitos, descritas na Subseção a seguir. Embora o RUP e a UML sejam praticamente populares, eles não são bem fundamentados com um método formal, tornando difícil analisar a consistência das especificações UML. Este trabalho é para a integração de um método formal com RUP e UML (LIU; JIFENG; LIU, 2004).

A UML é uma notação amplamente utilizada para especificação e design orientado a objetos, e também é um padrão internacional. Juntamente com a Linguagem de Restrição de Objetos, representa uma fusão de linguagens de especificação formal e gráfica que possui alto potencial para introduzir os benefícios das técnicas de especificação formal no desenvolvimento de *software* convencional (LANO; ANDROUTSOPOLOUS; CLARK, 2005).

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de *software*. Cada classe determina o comportamento, definido nos métodos, e estados possíveis, os atributos, de seus objetos, assim como o relacionamento com outros objetos (WIKIPEDIA, 2017).

Submetido em 18.05.2018. Aprovado em 17.07.2018

Para Sommerville (2011) um modelo de processo de *software* é uma representação simplificada de um processo de *software*. Cada modelo, como o próprio RUP, representa uma perspectiva particular de um processo, portanto fornece informações parciais sobre ele.

Muitos atores tentaram criar tecnologias diferentes e exóticas para melhorar principalmente a experiência do usuário e ajudar os desenvolvedores a criarem aplicativos web mais rápidos e mais potentes. Algumas dessas tecnologias têm desempenhado um papel importante no desenvolvimento da web, mas viram uma queda importante no uso nos últimos anos, como Applets Java e Microsoft Silverlight. Por outro lado, muitas tecnologias evoluíram de simples tecnologia para partes poderosas e importantes do ecossistema web de hoje, como JavaScript, Flash e XML (POP; ALTAR, 2014).

Análise e *design* são os estágios de modelagem existentes desde há muito tempo. O design é o passo da estruturação do *software* para módulos e sub-módulos. No nível de modelo de plataforma independente é visto apenas o design abstrato, que é feito sem conhecimento das técnicas de implementação. O objetivo dos modelos de análise e *design* e modelos de plataforma independente, deve ser sustentável e para fornecer o *link* entre modelos de negócios, modelos computacional independente, modelos de código e modelos de plataforma específica. Os modelos UML são apresentados da seguinte forma: o modelo de diagrama de estado interpreta a visão dinâmica, os modelos de diagramas de classe e pacote mostram a visão estática e o modelo do caso de uso apresenta uma visão funcional. Finalmente, estrutura-se todas as classes de acordo com o padrão de design MVC (RHAZALI; HADI; MOULOUDI, 2016).

4 PARADIGMA MVC

Antes mesmo de se iniciar esse assunto é importante destacar os dois princípios que norteiam a ES: a) princípios que orientam o processo; e b) princípios que norteiam a parte prática. Além desses princípios da ES é conveniente também ressaltar os

princípios que regem as atividades metodológicas: comunicação; planejamento; modelagem; construção; e disponibilização (PRESSMAN, 2011).

Em relação aos temas tratados nesse trabalho, Pressman (2011) deixa claro que uma série de tecnologias e fundamentos são essenciais para se produzir *software* com qualidade, tanto no processo como no produto.

O padrão de design MVC foi inicialmente visualizado por Trygve Reenskaug nos anos 70 no Xerox Parc. Segundo ele, “o propósito essencial do MVC é fazer a ponte entre o modelo mental do usuário humano e o modelo digital que existe no computador” (POP; ALTAR, 2014).

O conceito do paradigma arquitetural MVC descrito por Reenskaug em 1979, desde aquela época até o momento atual existem duas variedades de MVC que têm comportamentos que notificam *Views* sobre suas alterações, no entanto a camada *View* foi idealizada para obter notificações sobre mudanças na camada *Model*, por isso há uma ligação entre elas. Isso permite manter a independência da camada *Model* em relação as camada *Controller* e *View* (PROKOFYEVA; BOLTUNOVA, 2017).

Os desenvolvedores combinam o código HTML com as linguagens de programação do lado do servidor para criar páginas e aplicativos dinâmicos e isso leva a um código altamente emaranhado e de difícil manutenção. Outro problema surgiu do fato de que as tecnologias web são cada vez mais utilizadas para construir todos os tipos de aplicações complexas.

Existem enormes benefícios a serem obtidos se alguém construir aplicativos com modularidade em mente. “Isolar as unidades funcionais um do outro, tanto quanto possível, torna mais fácil para o *designer* do aplicativo entender e modificar cada unidade específica sem ter que saber tudo sobre as outras unidades”. Uma aplicação é dividida em três categorias principais: o modelo do domínio principal do aplicativo, a apresentação de dados nesse modelo e a interação do usuário (POP; ALTAR, 2014).

As tecnologias *front-end* e *back-end* se uniram para construir aplicações web, mas como a *World Wide Web* (www) evoluiu a um ritmo muito rápido e porque os desenvolvedores precisam usar um número bastante grande de tecnologias para

construir apenas uma única aplicação web, o resultado do seu trabalho é muitas vezes difícil de manter e corrigir (POP; ALTAR, 2014).

Rodríguez e Acuña (2013) retrataram que inicialmente, a usabilidade foi vista como um requisito não funcional relacionado exclusivamente à interface do usuário (UI), o que significa que foi abordado no final do processo de desenvolvimento e foi implementado usando abordagens que separaram a funcionalidade de UI da funcionalidade do sistema central. Um exemplo desta abordagem é o padrão do modelo, controle e visualização (MVC), que é amplamente utilizado para desenvolver *software* e especialmente aplicativos da web.

A análise de projetos de *sofware* durante sua construção em larga escala de sistemas Orientado a Objetos típico é baseado em várias camadas arquiteturais (LARMAN, 2007). Dito isso, há uma variedade de configurações quando ao uso de camadas aplicadas à programação OO, contendo de 2 a 4 camadas, podendo existir ainda mais, conforme o grau de interdependência idealizado para o modelo arquitetural do *software* a ser desenvolvido.

Usar arquitetura baseada em componentes oferece várias soluções para as causas de origem de problemas no desenvolvimento de *software*:

a) capacidade de recuperação rápida;

b) separação clara de relações entre elementos de um sistema sujeito a mudanças;

c) reutilização facilitada por influência de estruturas padronizadas;

d) facilidade no gerenciamento de configuração; e

e) ferramentas de modelagem visual fornecem automação para o desenvolvimento baseado em componentes.

As três camadas previstas no padrão MVC serão discorridas aqui para que se possa entender as responsabilidades de cada uma delas segundo a visão de Pop e Altar (2014):

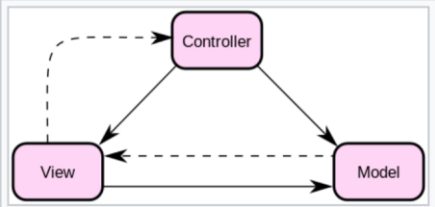
a) **Camada *Model***: é a parte do sistema que gerência todas as tarefas relacionadas aos dados: validação, estado de sessão e controle, estrutura de fonte de

dados (banco de dados). O Modelo reduz consideravelmente a complexidade do código que o desenvolvedor precisa para escrever;

b) **Camada View**: é responsável pelo gerenciamento gráfico da interface do usuário. Isso significa dizer que todos os formulários, botões, elementos gráficos e todos os outros elementos HTML que estão dentro do aplicativo. As visualizações também podem ser usadas para gerar conteúdo RSS, para agregadores ou apresentações em Flash. Ao separar o *design* do aplicativo, sua lógica reduz muito o risco de erros ocorrendo quando o designer decide alterar a interface do aplicativo alterando, por exemplo, um logo-tipo ou uma tabela; e

c) **Camada Controller**: é responsável pelo manuseio de eventos. Esses eventos podem ser acionados por um usuário interagindo com o aplicativo ou por um processo do sistema. O controlador aceita solicitações e prepara os dados para uma resposta. É também responsável por estabelecer o formato dessa resposta. Ele ainda interage com o *Model* para recuperar os dados necessários e gera a Visualização.

A Figura 1, apresenta um esquema das três camadas do MVC.



Um diagrama simples exemplificando a relação entre *Model*, *View* e *Controller*. As linhas sólidas indicam associação direta e as tracejadas indicam associação indireta.

Figura 1: Estrutura MVC – três camadas
Fonte: Wikipédia (2017)

Os desenvolvedores enfatizam que existem enormes benefícios a serem obtidos se tiverem em mente um desenvolvimento de aplicativos com modularidade. “O isolamento de unidades funcionais umas das outras, tanto quanto possível, torna mais fácil para o designer de aplicativos entender e modificar cada unidade em particular, sem ter que saber tudo sobre as outras unidades” (POP; ALTAR, 2014).

Por terem camadas isoladas, no MVC, a manutenção de sistemas que empregam essa tecnologia tem um ganho expressivo uma vez que problemas enfrentados na camada visão são tratadas nela sem ter que modificar as outras duas camadas. Em resumo, cada camada é tratada em termos de manutenção de forma isolada mas sempre considerando-se a integração entre elas. Esses recursos são considerados equivocadamente como uma grande vantagem em se trabalhar com essa tecnologia.

Outro grande benefício é a rápida prototipação, onde em aplicações web muitas partes são usadas repetidas vezes. Por exemplo: formulários web, tabelas de listas, são usadas com muita frequência. Mas construir esses elementos a partir do zero muitas vezes se tornam bastante difíceis, porque envolver escrever várias marcações complexas. Assim cada sistema pode se beneficiar de um módulo de prototipagem rápida que não apenas gera essas ferramentas rapidamente, mas também fornece formas de validar dados, para formulários. Essa técnica é baseada em um mecanismo de modelo flexível e totalmente transparente, que beneficia tanto os desenvolvedores de *back-end* quanto os de *front-end*.

O modelo MVC nem sempre é simples de se implementar ou de fácil entendimento. Por isso existem no mercado diversas ferramentas que implementam o modelo atribuindo inclusive mais camadas do que o do própria MVC. Isso poder ser um problema pois o sistema desenvolvido fica ”escravo” ou dependente da ferramenta para manutenções, principalmente.

5 A PROGRAMAÇÃO ORIENTADA A OBJETOS

Ao final da década de 70 muito se estudou para fazer uma mudança radical na forma de programar sistema, quando foi criado o paradigma de Programação Orientada a Objetos (POO). No início sua utilização mostrou-se muito acanhada mas quando os programadores perceberam a grande potencialidade dessa técnica foi utilizada maciçamente.

Submetido em 18.05.2018. Aprovado em 17.07.2018

A programação orientada a objeto está intimamente ligada a linguagem unificada de modelagem tendo como acrônimo em inglês (UML). No entanto, não significa que ambas tecnologias não estejam atreladas entre si, pois a primeira trata-se de técnicas de programação e a segunda trata-se de projeto de *software*.

Análise e design são os estágios de modelagem existentes desde há muito tempo; O design é o passo da estruturação do *software* para módulos e sub-módulos. No nível do Modelo de Plataforma Independente, observa-se apenas o design abstrato, que é feito sem conhecimento de técnicas de implementação (RHAZALI; HADI; MOULOUDI, 2016).

Até a metade de década de 70, a técnica de programação era estruturada, que auxiliou muitos sistemas legados e que existem até hoje como é o caso de sistemas programados em cobol, dita como uma linguagem morta, mas muito utilizada em sistemas de grande porte, mainframes, particularmente pelos bancos mais antigos.

Um sistema de mapeamento relacional de objeto acrônimo em inglês (ORM) é definido como sendo uma ferramenta que fornece uma metodologia e um mecanismo para sistemas orientados a objetos para armazenar dados de maneira segura e por um longo período de tempo em um banco de dados, tendo controle transacional sobre eles, mas sendo expressos, se necessário, como objetos dentro do aplicativo (POP; ALTAR, 2014)(LIN; YANG; LIN, 2012).

Existem uma infinidade de frameworks, como phpcake e outros que implementam em várias linguagens de programação orientada a objeto. Isso dá um ganho substancial à programação porém gera um grande problema que é a inexistência de documentação de sistemas para sua manutenção. Considerando-se que a maior parte do ciclo de vida de um sistema é a fase de manutenção, pode-se considerar isso como um sério problema (ZÁRATE, HERNÁNDEZ, GONZÁLEZ, 2012).

O conceitos aplicados a OO são os seguintes:

a) **Objeto**: é um tipo abstrato que contém dados mais os procedimentos que manipulam esses dados.

b) **Mensagens**: É uma informação enviada ao objeto para que ele se comporte de uma determinada maneira. Um programa orientado a objetos em execução consiste em envios, interpretações e respostas às mensagens.

c) **Métodos**: São procedimentos residentes nos objetos que determinam como eles atuarão ao receber as mensagens.

d) **Variáveis de Instância**: São variáveis que armazenam informações ou dados do próprio objeto. Podem também ser chamadas de propriedades do objeto.

e) **Classe**: Representa um tipo específico de objeto. Uma classe é composta pela sua descrição, que identifica tanto as variáveis de classe (propriedades da classe) quanto os métodos.

f) **Subclasse**: Uma nova classe originada de uma classe maior (classe-pai).

g) **Instância**: São os objetos de uma classe. Cada objeto utilizado em uma aplicação pertencente a uma classe é uma instância dessa classe.

h) **Hereditariedade**: É um mecanismo que permite o compartilhamento de métodos e dados entre classes, subclasses e objetos. A hereditariedade permite a criação de novas classes programando somente as diferenças entre a nova classe e a classe-pai.

i) **Encapsulamento**: É um mecanismo que permite o acesso aos dados do objeto somente através dos métodos desse objeto. Nenhuma outra parte do programa pode operar sobre os dados de nenhum objeto. A comunicação entre os objetos é feita apenas através de mensagens.

j) **Polimorfismo**: Uma mesma mensagem pode provocar respostas diferentes quando recebidas por objetos diferentes. Com o polimorfismo, pode-se enviar uma mensagem genérica e deixar a implementação a cargo do objeto receptor da mensagem.

k) **Persistência**: É a permanência de um objeto na memória. Quando um objeto é necessário, ele é criado na memória (métodos construtores). Quando ele não for mais necessário, é destruído da memória (métodos destrutores). Quando um objeto é destruído, seu espaço na memória é liberado automaticamente. Este processo recebe o nome de “coleta de lixo” (*garbagge collection*)

Submetido em 18.05.2018. Aprovado em 17.07.2018

A POO tem vantagens em se trabalhar com essa abordagem como, por exemplo, tem alcançado tanta popularidade devido às vantagens que ela traz. Entre elas podemos citar: a) Reusabilidade de código; b) Escalabilidade de aplicações; c) Manutenibilidade e a Apropriação.

A POO não é tão trivial aplicar todos os conceitos que a ela são incorporados, assim para se tornar em um bom programador exige-se um tempo considerável de estudo e aplicação de casos, envolvendo todos os seus conceitos.

6 O PROCESSO UNIFICADO DA RATIONAL

Esse capítulo não tem a pretensão de abranger toda a tecnologia que vem embutida no RUP, mas apresentar pontos importantes de interesse desse estudo.

Como é sabido no mundo da tecnologia que sistemas de TI, tanto nas fases de desenvolvimento quanto na produção apresentam problemas obviamente em circunstâncias diferentes para cada um dois casos. Kruchten (2003) afirma que diferentes projetos de desenvolvimento de *software* falham de formas diferentes – e infelizmente, muitas dessas falhas – mas é possível identificar vários sintomas comuns que caracterizam esses tipos de projetos, aqui serão apresentadas quatro, das nove proposições, para não tornar o texto do artigo muito extenso:

a) **incompreensão das necessidades do usuário final**. Este item deveria ter os analistas de requisitos maiores habilidades para se capturar com completeza e maior definição na fase de levantamento de requisitos, particularmente nas fases iniciais do projeto de desenvolvimento de *software;*

b) **inabilidade para lidar com requisitos variáveis**. É muito comum ocorrer nas diversas fases de desenvolvimento, particularmente nas fases iniciais. Quando se observa mudanças de requisitos nas fases finais ou até mesmo já em produção os custos passam a serem muito mais altos, sem contar que requerem mais tempo para o desenvolvimento do sistema ou para a sua entrada em produção, dependendo de cada um dos dois casos;

c) **um processo de construção e lançamento indigno de confiança**. Neste sentido muitos desenvolvedores simplesmente ignoram metodologias consagradas para o desenvolvimento de um sistema de TI, como o RUP, Scrum e XP, dentre outras. Isso deixa a arquitetura do sistema inconsistente e fragilizada e assim torna o *software* desenvolvido como uma verdadeira "colcha de retalhos" onde os módulos do sistema não se integram e com isso resultando em baixa qualidade do produto de *software; e*

d) **baixa qualidade de *software***. Este talvez seja o mais importante pois de certa forma congrega com quase todo as outras proposições, pois perpassa pela fragilidade do levantamento de requisitos, requisitos mau interpretados, dificuldade na integração dos módulos do sistema, falta de aplicação mais apropriada ou mesmo esquecimento pela equipe desenvolvedora em considerar o maior número de requisitos não funcionais, a não aplicação e falta de realização de testes de *software* com qualidade.

O RUP, segundo Kruchten (2003), é um processo de engenharia de *software*. Ele fornece uma abordagem disciplinar para assumir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a produção de *software* de alta qualidade que satisfaça as necessidades de seus usuários finais dentro de prazo e orçamento previsíveis. Kruchten (2003) foi feliz ao mencionar da satisfação dos usuários finais pois de nada adiantaria se achar pela equipe desenvolvedora que o sistema ficou "redondo" e não atender a quem vai utilizá-lo.

O RUP foi criado pela *Rational Software Corporation*, adquirida pela IBM, ganhando um novo nome **IRUP** que agora é uma abreviação de **IBM Rational Unified Process** e tornando-se uma marca na área de *Software*. Ele em suas versões iniciais é um *software* livre mas agora a IBM cobra por sua utilização, além de outros frameworks utilizados para o desenvolvimento (WIKIPÉDIA, 2017).

Para Piske (2017) um dos principais pilares do RUP é o conceito de melhores práticas, que são regras/práticas que visam reduzir o risco, que existente em qualquer projeto de *software*, e tornar o desenvolvimento mais eficiente. O RUP define essas melhores práticas sendo elas:

a) Desenvolver iterativamente;

b) Gerenciar requerimentos;

c) Utilizar arquiteturas baseadas em componentes;
d) MODELAR visualmente;
e) Verificar contínua de qualidade; e
f) Controlar as mudanças.

O RUP reconhece, segundo Sommerville (2011) que os modelos de processo convencionais apresentam uma visão única do processo. Em contrapartida, o RUP é normalmente descrito em três perspectivas:

a) **dinâmica**: mostra as fases do modelo ao longo do tempo;
b) **estática**: mostras as atividades realizada no processo; e
c) **prática**: que sugere boas práticas a serem usadas durante o processo

O RUP não define a qualidade como responsabilidade de apenas alguns papéis envolvidos, muito pelo contrário a responsabilidade recai a todos os integrantes envolvidos no projeto de TI.

A qualidade do processo RUP é focada especialmente em duas áreas:

a) **Qualidade de produto**: a qualidade do produto sendo desenvolvido (*software* ou sistema) e todos as partes envolvidas (componentes, subsistemas, arquitetura, etc); e

b) **Qualidade de processo**: a qualidade dos processos dentro do projeto de desenvolvimento

Para os profissionais de *software* o desenvolvimento de sistemas de TI cresce a cada dia em virtude da economia mundial depender cada vez mais do *software*. Os tipos de sistemas de *software* intensivo que a tecnologia torna possível e a demanda da sociedade estão aumentando em tamanho, complexidade, distribuição e importância (KRUCHTEN, 2003).

A metodologia RUP por não ser muito trivial segui-la é aconselhável fazer um treinamento com a equipe desenvolvedora para tirar o maior proveito dela e compreender os vários subprocessos nele envolvidos e conhece ainda os possíveis artefato que

podem ser criados como documentação do projeto de desenvolvimento de *software* e servir futuramente na realização de manutenções do sistema.

Por ser bem flexível quanto a sua utilização ele pode ser personalizado para cada tipo de projeto que deseje ser desenvolvido, pois apresenta duas principais versões a *Large Project* para o desenvolvimento de sistemas mais robustos e a versão a *Small Project* para sistemas de menor grau de complexidade. Desenvolvedores defensores de metologias ágeis criticam o RUP por ele produzir uma gama considerável de documentação, mas como o RUP pode ser reconfigurado, caso o desenvolvedor não deseja produzir muitos artefatos de projeto sem comprometê-lo é perfeitamente factível.

O RUP trabalha em duas dimensões, a primeira trato do aspecto estático e a segunda de aspectos dinâmicos. A estrutura do processo RUP está organizado em quatro fases e dez disciplinas conforme a Figura 2.

Nela observa-se a divisão por fazes e em cada fase as disciplinas são atuadas na medida necessária para cada fase do projeto. Por exemplo: A disciplina implantação só começa a ser vista somente na elaboração, já a disciplina Modelagem de Negócio tem seu forte na fase de Iniciação e um pouco mais diluída na elaboração. E assim se comportam as disciplinas ao longo da execução do projeto de TI.

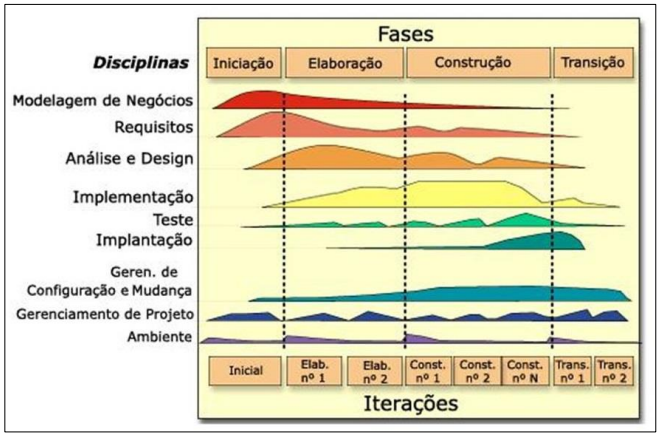


Figura 2: Estrutura do processo – duas dimensões
Fonte: RUP

O RUP não é adequado para todos os tipos de desenvolvimento, como, por exemplo, desenvolvimento de *software* embutido. No entanto, ele representa uma abordagem que potencialmente combina os três modelos, em cascata, incremental e

Submetido em 18.05.2018. Aprovado em 17.07.2018

Engenharia de *Software* orientada a reúso. As inovações mais importantes do RUP são a separação de fases e *workflows* e o reconhecimento de que a implantação de *software* em um ambiente do usuário é parte do processo (SOMMERVILLE, 2011).

7 CONCLUSÃO E TRABALHOS FUTUROS

Pode-se constatar que o RUP é uma metologia madura para se gerenciar projeto e conduzi-lo ao sucesso com qualidade. A programação OO possui bastantes recursos que a torna bem flexível na utilização de suas caraterísticas. O Modelo MVC aplicado na POO traz um ganho considerável pois na fase de manutenção o sistema ficará basante disciplinado pois como as camadas o MVC são separadas para que se interligam pela POO, torna fácil sua manutenção ao longo do ciclo de vida do sistema.

As tecnologias apresentadas têm uma curva de aprendizado não tão favorável pois há necessidade de após treinamentos realizar-se exercícios de aprendizagem. Isso para um desenvolvedor principiante. Já para um desenvolvedor experiente o aprendizado dessas tecnologias é relativamente fácil.

O RUP como metodologia, canaliza ao desenvolvimento dos diversos Casos de Uso e a implementação por meio de OO (SOMMERVILLE, 2011). Com o emprego do padrão MVC será de grande utilidade para o sistema de TI a ser construído pois traz o melhor de cada uma dessas tecnologias

Apesar das dificuldades apontadas, equipes de desenvolvimento em projetos de TI possuem um certo tempo no projeto que podem amadurecer seus conhecimentos e tentar reduzir a curva aprendizagem.

Nos dias atuais programadores vem utilizando essas e outras tecnologias para prover qualidade nos sistemas de TI que vem desenvolvendo. Assim, sugere-se que adote-se esses três tecnologias em comum, ou seja, a programação OO com uso de UML, o modelo arquitetural MVC e a metodologia de desenvolvimento de sistemas RUP. Certamente os resultados serão surpreendentes se utilizados corretamente e na medida certa.

Submetido em 18.05.2018. Aprovado em 17.07.2018

Como trabalhos futuros poder-se-ia monitorar um desenvolvimento de sistema que uso dessas tecnologias para se apontar os aspectos positivos e negativos.

Referências

KRUCHTEN, Philippe. “Introdução ao RUP – Rational Unified Process”.Rio de Janeiro-RJ, Editora Ciência Moderna Ltda, edição revisada, 2003.

LANO, K; ANDROUTSOPOLOUS, K; CLARK, D. “Refinement Patterns for UML”, Electronic Notes in Theoretical Computer Science 137 (2005) 131–149

LARMAH, Craig. “Utilizando UML e Padrões de projetos”. 3ª edição. Porto Alegre. Bookman, 2007. 695p.

LIN, Lendy; YANG, Weipang; LIN, Jyhjong. “A layer-based method for rapid software development”. Computers and Mathematics with Applications. 2012 Elsevier Ltd.

LIU, Zhiming; JIFENG, He; LIU, Jing. “Unifying Views of UML”. Electronic Notes in Theoretical Computer Science 101 (2004) 95–127

PROKOFYEVA, Natalya; BOLTUNOVA, Victoria. “Analysis and Pratical Application of PHP Frameworks in Development of Web Information Systems”. Riga Technical University, LV-1658. Published by Elsevier, 2017.

PISKE, Otávio Rodolfo. RUP – "Rational Unified Process". URL: http://www.angusyoung.org/arquivos/artigos/trabalho_rup.pdf. Acesso em Maio 2017.

POP, Dragos-Paul; ALTAR, Adam. "Designing an MVC Model for Rapid Web Application Development". Published by Elsevier Ltd., 2014.

Submetido em 18.05.2018. Aprovado em 17.07.2018

RHAZALI, Yassine; HADI, Youssef; MOULOUDI, Abdelaziz. “Model Transformation with ATL into MDA from CIM to PIM Structured through MVC”. The 6th International Symposium on Frontiers in Ambient and Mobile Systems (FAMS 2016)

ROCHA, José Gladistone da. Trabalho de conclusão de curso de graduação, “Empresas e Produtos Estratégicos de Defesa do Brasil”. Centro Universitário do Sul de Minas (UNIS) Varginha-MG. 2014. 163p.

SOMMERVILLE, Ian. “Engenharia de Software”. 9ª ed., São Paulo-SP, Pearson Prentice Hall, 2011.

WIKIPEDIA, site. URL:https://pt.wikipedia.org/wiki/IBM_Rational_Unified_Process. Acesso em Maio de 2017.

ZÁRATE, María del Pilar Salas-; HERNÁNDEZ, Giner Alor-; González, Alejandro; Rodríguez. “Developing Lift-based Web Applications Using Best Practices”. The 2012 Iberoamerican Conference on Electronics Engineering and Computer Science