A IMPORTÂNCIA DO CICLO DE VIDA NO DESENVOLVIMENTO DE SOF*TWARE*

Durvalino Batista Godoi Neto
Discente do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas
Faculdades Integrada de Três Lagoas (AEMS)

Samuel Fernandes Pires Siqueira
Discente do Curso de Tecnologia em Análise e Desenvolvimento de Sistema
Faculdades Integrada de Três Lagoas (AEMS)

Alan Pinheiro de Souza
Docente do Curso Tecnologia em Análise e Desenvolvimento de Sistemas
Faculdades Integradas de Três Lagoas (AEMS)
Mestre em Informática pela Universidade Federal do Rio de Janeiro (UFRJ)

Rodrigo de Carvalho Ribeiro
Docente do Curso Tecnologia em Análise e Desenvolvimento de Sistemas
Faculdades Integradas de Três Lagoas (AEMS)

RESUMO
O objetivo deste trabalho é mostrar uma ampla visão com conceito acadêmico e profissional do ciclo de vida de um *software*, as suas variedades e suas aplicações, a fim de exemplificar, facilitar e ajudar leitores e pesquisadores nesta mesma área. Sendo assim, existem diferentes visões de escritores distintos sobre este assunto. Realizou-se um levantamento bibliográfico, mostrando a importância do ciclo de vida no projeto e como deve ser adotado para melhor desenvolvimento de *software*. Pode-se concluir que, cada vez mais, empresas de desenvolvimento de sistemas necessitam adotar processos de *software* adequados para cada tipo de projeto. A definição do ciclo de vida de um *software* é importante para se ter a visão completa do desenvolvimento em si. Com isto, foi possível definir etapas que abrangem desde a análise dos requisitos até a entrega do *software* ao cliente, mostrando sempre a importância do ciclo de vida no planejamento e no desenvolvimento do *software*.

PALAVRAS-CHAVE: Ciclo de Vida; Engenharia de *Software*; Qualidade *Software;* Processos de *Software*; Requisitos.

INTRODUÇÃO

O *software* é a parte programável de um sistema computacional. Ele é um elemento central, porém outros componentes são essenciais: as plataformas de *hardware*, os recursos de comunicação de informação, os documentos de diversas naturezas, as bases de dados e até os procedimentos manuais que se integram aos automatizados (PAULA FILHO, 2000, p.11). Em um contexto generalizado, os projetos de *software* são divididos em etapas para facilitar o controle gerencial e permitir uma melhor sintonia com os processos gerenciais. Essas etapas são

denominadas ciclo de vida do projeto. Quando um projeto se inicia, o gerente de projeto, juntamente com sua equipe de desenvolvimento, deve decidir quais e quantas etapas serão desempenhadas. No caso de um projeto para desenvolver-se um novo tipo produto, pode iniciar com um estudo de viabilidade ou também com a construção de um protótipo, ou direto com uma construção de um produto final (MARTINS, 2010, p.05-06).

Na engenharia de *software* se preocupa com o *software* enquanto produto. Estão fora de seu escopo programas que são feitos unicamente para diversão do programador. Estão fora de seu escopo também pequenos programas descartáveis, feitos por alguém exclusivamente como meio para resolver um problema, e que não serão utilizados por outros (PAULA FILHO, 2008, p.5).

Sendo assim, a engenharia de *software* é utilizada em âmbito comercial, e como todo produto comercializado, no *software* também existe um ciclo de vida, sendo este concebido a partir da percepção de uma necessidade; quando desenvolvido, transforma-se em um conjunto de itens entregue a um cliente; ao fim de todo seu processo de análise, desenvolvimento e teste, o *software* entra em operação, sendo usado dentro de um algum processo de negócio, e sujeito a atividades de manutenção, quando necessário. Quando o mesmo ficar subtilizado, é retirado de operação, ao final de sua vida útil (PAULA FILHO, 2000, p.12).

Este artigo propõe-se a apresentar um levantamento bibliográfico embasado em livros. O trabalho está dividido em três seções, sendo proposta na primeira seção uma visão conceitual sobre ciclo de vida de *software*. A segunda seção cita várias visões de como adotar o melhor ciclo de vida no projeto de um *software*. E na terceira seção são citados vários modelos de ciclo de vida, suas aplicações, seus conceitos e possíveis vantagens e desvantagens.

1. CONCEITUALIZAÇÃO DE UM CICLO DE VIDA

Nesta etapa será apresentada uma visão geral e conceitual do ciclo de vida, na percepção de autores distintos, onde se mostra várias reflexões sobre o conceito e as aplicações dos ciclos de vida nos projetos de *software*, além de uma breve explicação ilustrativa, sobre as etapas de desenvolvimento e produção de *software.*

O ciclo de vida nada mais é do que o desenvolvimento de um *software* dividido em etapas, dentre estas etapas estão: estudo da viabilidade; análise do sistema; projeto; implementação; geração do teste do aceite; garantia de qualidade; descrição de procedimentos; conversão de banco de dados; e instalação (YOURDON, 1989 *apud* REZENDE, 2005, p.42).

Um ciclo de vida de sistema existe para cada projeto, independente da metodologia utilizada. Todo ciclo deve ser ajustado na medida de sua utilização. No entanto, uma metodologia que tem por base um ciclo de vida deve ser a mais abrangente possível para contemplar tanto um sistema complexo desenvolvido internamente, quanto um simples sistema ou aquisição de um pacote de *software* (REZENDE, 2005, p.47). Na visão de Paula Filho (2000, p.12):

> Cada fase do ciclo de vida tem divisões e subdivisões. É interessante observar, que a codificação, que representa a escrita final de um programa em forma inteligível para um computador, é apenas uma pequena parte do ciclo de vida. Para a maioria das pessoas, inclusive muitos profissionais da informática, esta parece ser a única tarefa de um programador, ou seja, um produtor de *software*.

A Figura 1 esquematiza um modelo de ciclo de vida de *software,* que se inicia a partir de uma percepção da necessidade, e passa a ter uma fase de desenvolvimento com etapas de concepção e elaboração. Ainda nessa fase, inicia-se a construção que acomoda uma etapa de desenho inicial. Dentro da fase de construção existe a liberação que conta com o desenho detalhado, a codificação e os testes de unidade. Após isso, na fase de construção e se inicia a etapa do teste alfa, e logo depois na fase de desenvolvimento será realizada a transição, tendo fim todos esses procedimentos o sistema entra em operação, ao fim quando o *software* é retirado de operação ou descontinuado pela empresa, o ciclo de vida é encerrado.

Figura 1: Ciclo de vida de *software*.

<table>
<tr><td rowspan="10">Ciclo de vida</td><td colspan="4">Percepção da necessidade</td></tr>
<tr><td rowspan="9">Desenvolvimento</td><td colspan="3">Concepção</td></tr>
<tr><td colspan="3">Elaboração</td></tr>
<tr><td rowspan="5">Construção</td><td colspan="2">Desenho inicial</td></tr>
<tr><td rowspan="3">Liberação</td><td>Desenho detalhado</td></tr>
<tr><td>Codificação</td></tr>
<tr><td>Testes de unidade</td></tr>
<tr><td colspan="2">Testes alfa</td></tr>
<tr><td colspan="3">Transição</td></tr>
<tr><td colspan="4">Operação</td></tr>
<tr><td colspan="4">Retirada</td></tr>
</table>

Fonte: Retirado de Paula Filho (2000, p.13).

2. A ESCOLHA DE UM CICLO DE VIDA

Umas das partes cruciais de um projeto de *software* é a boa escolha de seu ciclo de vida, pois não sabendo escolhê-lo bem, o projeto pode acabar em decadência. Portanto, esta etapa do levantamento bibliográfico é dedicada a esse assunto, lembrando que para a escolha do ciclo de vida, ele deve atender no mínimo as características principais do projeto.

No desenvolvimento de um *software*, o ponto de partida para a arquitetura de um processo é a eleição de um modelo de ciclo de vida. Um dos modelos mais caóticos é aquele que é chamado de “Codifica-remenda”. Partindo apenas de uma especificação (ou nem isto), os desenvolvedores começam instantaneamente a codificar, remendando à medida que os erros vão sendo descobertos. Nenhum passo é definido e seguido, infelizmente, esse é um dos modelos mais usados. Para alguns desenvolvedores, este modelo é atraente porque não exige nenhuma sofisticação técnica ou gerencial. Por outro lado, ele é de alto risco, difícil de gerir e não permite assumir compromissos confiáveis (PAULA FILHO, 2000, p.24).

Considerando modelos de desenvolvimento de projetos sistemáticos, um dos ciclos de vida mais utilizados na produção de *software* é o cascata, por ser simples e de fácil manipulação, sendo também um dos mais conhecidos na área. Entretanto, na visão de Sommerville (2003, p.39):

> O modelo de ciclo de vida cascata só deve ser utilizado quando os requisitos forem bem compreendidos. Contudo, ele reflete a prática de engenharia. Conseqüentemente, os processos de software com base nessa abordagem ainda são utilizados no desenvolvimento de software, em particular quando fazem parte de um projeto maior de engenharia de sistemas.

Segundo Schach (2010, p.65), empresas de desenvolvimento de *software* devem decidir sobre o modelo de ciclo de vida que seja pertinente a sua organização, sua direção, seus funcionários e seu processo de *software*, e deve diversificar o modelo de ciclo de vida dependendo dos recursos do produto característico em desenvolvimento no momento. Um modelo desses incorpora aspectos aprimorados dos vários modelos de ciclo de vida utilizando seus pontos fortes e minimizando seus pontos fracos.

Já na visão Martins (2010, p.06), não existe um ciclo de vida ideal, algumas empresas de desenvolvimento de *software* detêm seus próprios modelos de ciclo de

vida, enquanto outras deixam a critério da equipe de projeto a definição do ciclo de vida mais adequado. Em alguns casos o setor de negócios pode forçar uma adoção de um ciclo de vida padronizado. Segundo o PMBOK (*Project Management Body Of Knowledge*), os ciclos de vida definem o trabalho técnico de cada fase do projeto, os prazos de entregas de um sistema total ou parcial, as pessoas que estão envolvidas em cada fase e os mecanismos de controle e aprovação para cada fase.

3. EXEMPLOS DE CICLOS DE VIDA

Como foi dito em seções anteriores, o ciclo de vida é um processo de software que se inicia pela percepção da necessidade de implantação de um sistema, e se estende em outras tarefas de desenvolvimento e manutenção, até a etapa final quando esse sistema é retirado de operação. Para que tudo isso seja aplicado de maneira adequada, torna-se necessário definir um modelo de ciclo de vida logo no início do projeto, chegando-se a um produto final que satisfaça o cliente e usuário do sistema.

Nessa seção serão apresentados alguns modelos de ciclos de vida mais utilizados em projetos de *software*, destacando suas principais características, aspectos que favorecem sua aplicação em determinados contextos, além de possíveis vantagens e desvantagens de cada abordagem. O levantamento teórico focou seus estudos em três modelos: cascata, prototipagem e espiral.

3.1. Modelo Cascata

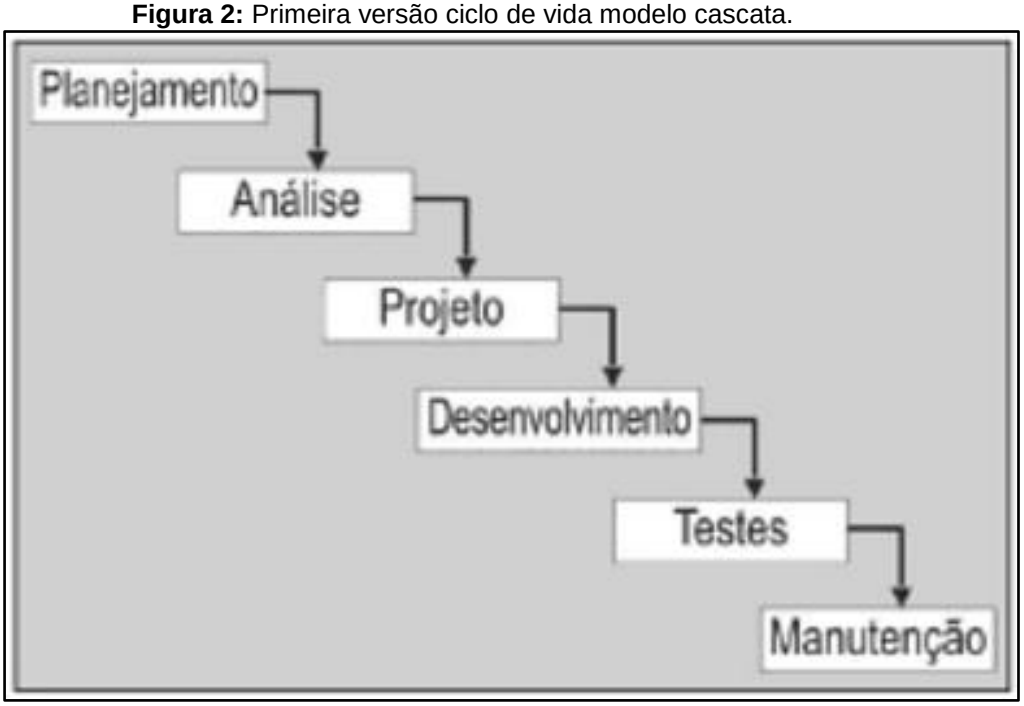
No processo de ciclo de vida em cascata, existe uma tentativa de demarcar em detalhe todos ou a maior parte dos requisitos antes da programação. Freqüentemente, se elabora também um projeto abrangente (conjunto de modelos) antes da programação. Igualmente, há uma tentativa de definir um plano ou cronograma “confiável” logo no começo (LARMAN, 2007, p.51).

Na visão de Ramos (2006, p.39), os processos tradicionais de desenvolvimento de *software* são caracterizados por deterem um modelo em cascata, em que as atividades que serão executadas são agrupadas em tarefas,

aplicadas seqüencialmente, de maneira que uma tarefa só tenha início quando a tarefa anterior tiver terminado.

Para entender este processo melhor, na Figura 2 tem-se uma representação de um ciclo de vida em modelo cascata, onde uma fase só pode se iniciar ao término de outra. O ciclo se inicia na etapa planejamento do *software*, por seqüência vem a análise, onde se é levantado os requisitos para planejar-se e desenvolver-se o produto. Ao termino da análise se inicia projeto, e conseqüentemente o desenvolvimento, onde se codifica a *interface* e as funcionalidades do *software*. No fim do desenvolvimento se inicia os testes, onde se tem a interação do usuário diretamente com produto, para melhor identificação de possíveis erros. E, por último, tem-se a manutenção do *software* já implantado, onde são corrigidos erros e adequados para melhor utilização.

Figura 2: Primeira versão ciclo de vida modelo cascata.



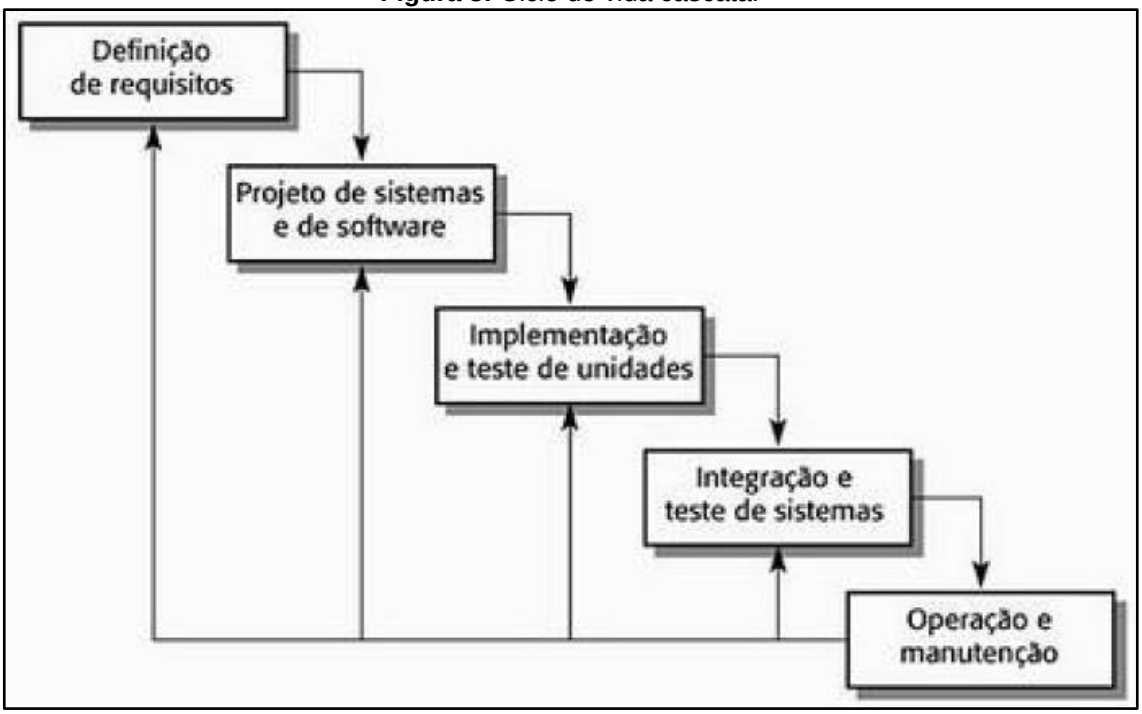
Fonte: Retirado de Ramos (2006, p.40).

Na visão de Sommerville (2003, p.37), esse ciclo de vida tem este nome por cauda de seu processo que tem uma sequência em forma de cascata de uma fase para outra. Os principais estágios desse modelo retratam as atividades de desenvolvimento de *software* fundamentais, como pode se observar na Figura 3.

O processo se inicia através da analise e definição de requisitos (que são os objetivos estabelecidos por meio da consulta aos usuários do sistema). Na próxima etapa, ele segue para o projeto (onde se agrupa requisitos do sistema para

estabelecer sua arquitetura), ao término do projeto passa-se para a implementação e os testes de unidade (isso envolve a verificação de cada unidade se atende a sua real especificação). Após essas etapas é feita a integração e o teste do sistema (onde se integram todas as unidades que vão compor o sistema, assim se faz o teste e, após isso, está pronto para entrega ao cliente). Por último, vem a operação e a manutenção (onde normalmente é a fase mais longa do ciclo de vida, é quando o sistema é colocado em operação, e dado sua manutenção, e podendo este ser estendido em suas funções a medida que novos requisitos são descobertos).

Figura 3: Ciclo de vida cascata.



Fonte: Retirado de Sommerville (2003, p.38).

3.1.1 Vantagens e Desvantagens do Modelo Cascata

O modelo em cascata tem a vantagem de só avançar para a tarefa seguinte quando o cliente aceita os produtos finais (documentos, modelos e programas) da tarefa atual. Esse modelo baseia-se no pressuposto de que o cliente participa ativamente do projeto e que sabe muito bem o que quer (RAMOS, 2006, p.40).

Outra vantagem do modelo cascata é que os principais subprocessos são executados em estrita seqüência, o que permite demarcá-las com pontos de controle

bem definidos. Os pontos de controle facilitam a gerencia do projeto, o que faz com que este processo seja, confiável e utilizável em projetos de qualquer escala.

Por outra visão, este é um processo inflexível e burocrático, onde as atividades de requisitos, análise e desenho têm de ser bem dominadas, pois não são admitidos erros. O modelo de cascata puro é de baixa nitidez ao cliente, que só recebe o resultado ao termino do projeto (PAULA FILHO, 2000, p.24).

O modelo em cascata tem ainda outras características, por exemplo, promove a divisão dos esforços ao longo das distintas tarefas e, conseqüentemente, desencoraja a comunicação e o compartilhamento de visões entre todos os participantes, como os analistas, os responsáveis pela etapa de projeto, os programadores e usuários. Além disso, minimiza o impacto da compreensão adquirida no decurso de um projeto, uma vez que se um processo não pode voltar de maneira a alterar os modelos e as conclusões das tarefas anteriores, é normal que as novas idéias sobre o sistema não sejam aproveitadas (RAMOS, 2006, p.40).

Segundo Sommerville (2003, p.39), um dos problemas com o modelo em cascata é sua inflexibilidade de divisão do projeto nos estágios distinto. Os acordos devem ser feitos em um estágio inicial do processo, isso significa que é difícil responder aos requisitos do cliente, que sempre se modificam.

3.2. Modelo Prototipagem

O objetivo do modelo de prototipagem é detalhar os requisitos da *interface* do usuário e os integrar aos outros requisitos como caso de uso, cenários, regra de dados e de negócio. As partes interessadas constantemente consideram a prototipagem como um meio concreto de identificar, descrever e autenticar, suas necessidades de *interface* (LAGUNA; KERBER, 2006, p.202-203).

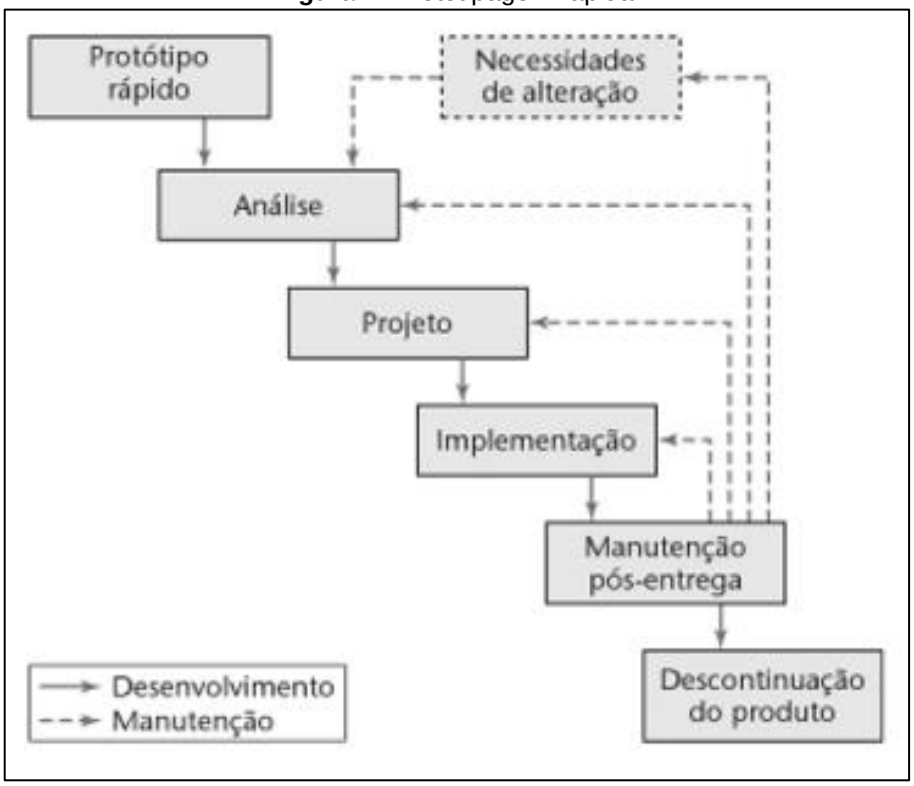
O processo de prototipação, é um processo que capacita o desenvolvedor a criar um modelo de *software* que será implementado. Trata-se de uma modelagem eficiente da engenharia de *software,* tendo como primordial fator a concordância entre cliente e desenvolvedor para que o protótipo seja desenvolvido para servir como um mecanismo a fim de definir os requisitos (PRESSMAN, 1995 *apud* ALVES; VANALLE, 2001, p.5).

Um protótipo rápido é um modelo operacional que é funcionalmente equivalente a um subconjunto do produto. Por exemplo, se o produto desejado deve

lidar com as contas a pagar, contas a receber e estoques, então o protótipo rápido é construído de um produto que executa na tela a captura dos dados e imprime relatórios, mas não realiza nenhuma atualização de arquivos e tratamento de erros.

O protótipo rápido é criado para deixar os usuários futuros interessados, fazendo assim uma assim uma melhor interação com o processo. Uma vez que o cliente fique satisfeito pelo fato do protótipo fazer a maior parte do que é necessário, os desenvolvedores podem redigir o documento de especificação com a certeza de que o produto atende as verdadeiras necessidades do cliente. Um ponto forte do modelo de prototipagem rápida pode ser observado na Figura 4 que é o desenvolvimento do produto essencialmente linear, progredindo do protótipo rápido para o produto entregue (SCHACH, 2010, p.54).

Figura 4: Prototipagem rápida.



Fonte: Retirado de Schach (2010, p.53).

3.2.1 Vantagens e Desvantagens do Modelo Prototipagem

Um dos benefícios em usar a prototipagem é apoiar os usuários a se sentirem mais confortáveis e efetivos na articulação das suas necessidades, utilizando imagens, uma vez que prototipagem os deixa “ver” a *interface* futura do sistema. Um protótipo descartável pode ser um meio barato pra se descobrir e

confirmar rapidamente uma diversidade de requisitos que vai além da *interface*, tais como processos e regras de dados e negócio.

Uma das desvantagens deste modelo é que dependendo da complexidade do sistema alvo, o uso de prototipagem pra requerer requisitos pode tomar um tempo considerável se o processo se prender “como é” ao invés “o que é”. Um protótipo pode levar os usuários a desenvolver expectativas não realistas quando ao desempenho do sistema entregue, data de entrega, características de confiabilidade e usabilidade. Isso acontece porque um protótipo desenvolvido e detalhado pode se parecer muito com um sistema funcional.

Os usuários podem focar nas especificações de *design* da solução, ao invés dos requisitos que a solução deve atender. Isso, por sua vez, os limita na concepção da solução. Os desenvolvedores podem crer que eles devem entregar uma interface do usuário que atenda principalmente ao protótipo, mesmo se a tecnologia de abordagens de interfaces melhores existirem (LAGUNA, KERBER, 2006, p.203).

A prototipação também sofre com algumas dificuldades, O cliente vê o protótipo e tem pressa para colocá-lo em funcionamento, não levando em consideração a qualidade global do sistema. Muitas vezes o desenvolvedor quer colocar o protótipo em funcionamento rapidamente, com isso levando o desenvolvedor a usar uma linguagem imprópria simplesmente porque esta a disposição (PRESSMAN, 1995 *apud* ALVES; VANALLE, 2001, p.5).

O modelo de prototipagem se torna mais eficiente que a abordagem cascata, porém apresenta alguns problemas, por exemplo, o processo não é visível, os sistemas são pobremente estruturados, quando um protótipo a ser descartado é construído o usuário exige algumas mudanças tentando igualá-lo ao produto final e, finalmente, o desenvolvedor faz algumas exceções e assume compromissos de implementações para garantir o funcionamento do produto com rapidez. (CARVALHO; CHIOSSI, 2001 *apud* LOURENÇO; BENINE, 2011, p.187).

Outra vantagem da abordagem prototipagem pode-se avaliar a qualidade de uma *interface* de usuário antes do protótipo ser construído fazendo que os problemas sejam identificados e corrigidos precocemente (PRESSMAN, 2011, p.312).

3.3. Modelo Espiral

Na visão de Audy e Prikladnicki (2007, p.17), o modelo espiral é relativamente novo, se comparado aos outros dois, e não costuma ser amplamente utilizado como os dois ciclos anteriormente explicados. Existem diversos casos práticos de uso do modelo espiral no desenvolvimento de *software*. Em muitos deles, identificam-se dificuldades na relação com o cliente, que muitas vezes não entende a necessidade de seguir essa abordagem.

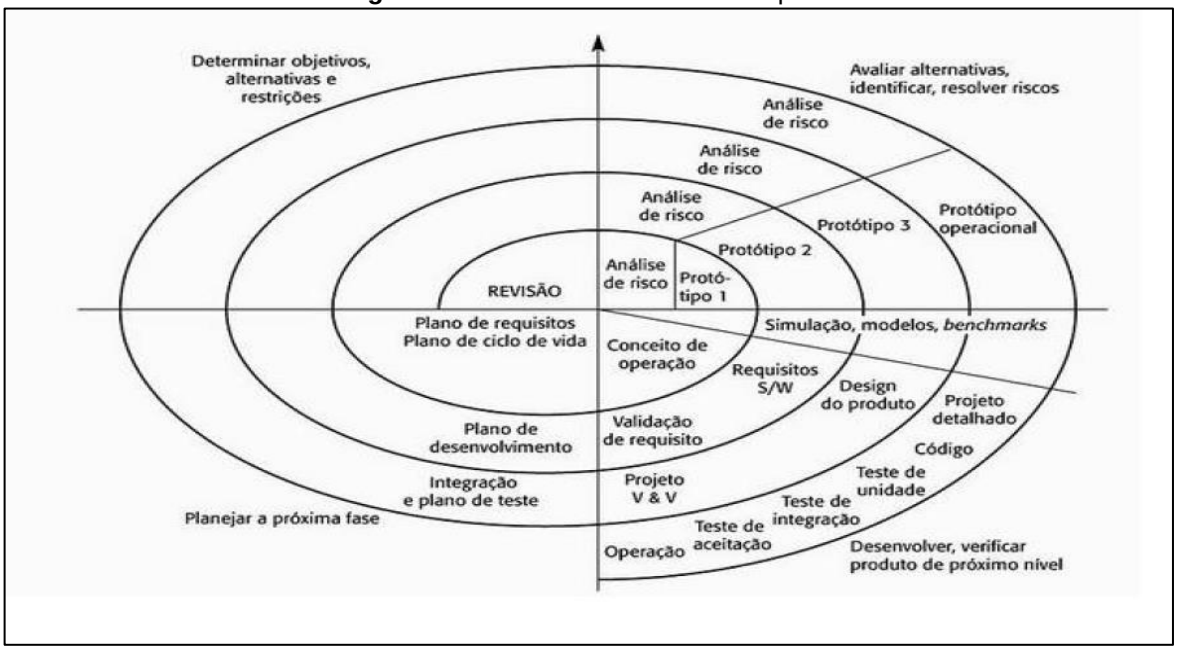
Mas só o tempo, que gera a experiência poderá comprovar a eficácia do modelo espiral. Baseado principalmente em decisões de prosseguir e não prosseguir, de acordo com a avaliação, seja do cliente ou da equipe responsável pelo projeto, o modelo espiral tende a uma trajetória que ruma para o modelo mais completo do sistema.

> O ciclo de vida de modelo espiral é atualmente a abordagem mais realista pra o desenvolvimento de softwares e sistemas em grande escala. Ele usa uma abordagem “evolucionária”, capacitando a equipe do projeto e o cliente a entenderem e reagirem aos riscos, em cada etapa evolutiva da espiral (AUDY, PRIKLADNICKI, 2007, p.17).

Segundo Sommerville (2003, p.44-45), o modelo de processo de desenvolvimento em espiral, surgiu sendo proposto por Boehm em 1988, e é agora muito conhecido. Em vez de representar um processo de *software* como seqüência de atividade com algum retorno de atividade para outra, o processo é representado como uma espiral. Cada volta no espiral representa uma etapa no processo. Cada *loop* na espiral representa uma fase do processo de *software.*

Sendo assim, as voltas são divididas em quatro setores como pode se ver na Figura 5, iniciando-se com a definição de objeto (nesta etapa são definidos os objetivos específicos e identificadas as restrições e riscos do processo), passando para a etapa de avaliação e redução de riscos (onde é realizada uma análise detalhada para cada um dos riscos e são tomadas providências). Após esta etapa, tem-se o desenvolvimento e a validação (etapa onde é escolhido um modelo de desenvolvimento do sistema de acordo com sua avaliação de risco) e, por último, o planejamento (onde o projeto é revisado e continua na próxima etapa do espiral).

Figura 5: Modelo de ciclo de vida espiral.



Fonte: Retirado de Sommerville (2003, p.45).

3.3.1 Vantagens e Desvantagens do Modelo Espiral

Segundo Paula Filho (2000, p.25-26), o modelo de ciclo de vida em espiral é totalmente diferente de outros modelos. O produto é desenvolvido em uma seqüência de iterações. Cada nova iteração corresponde a uma volta na espiral. Isto permite produzir produtos em prazos curtos, com novos aspectos e recursos que são acrescentados na medida em que a experiência descobre sua necessidade. As atividades de manutenção são utilizadas para identificar problemas; seus registros oferecem dados para definir os requisitos das próximas liberações. A principal desvantagem do ciclo de vida em espiral é que ele requer um gerenciamento bem sofisticado para ser previsível e confiável.

> Uma variante do modelo em espiral é o modelo de prototipagem evolutiva. Neste modelo, a espiral é usada não para desenvolver o produto completo, mas para construir uma série de versões provisórias que são chamadas de protótipos. Os protótipos cobrem cada vez mais requisitos, até que se atinja o produto desejado. A prototipagem evolutiva permite que os requisitos sejam definidos progressivamente, e apresenta alta flexibilidade e visibilidade para os clientes. Entretanto, também requer gestão sofisticada, e o desenho deve ser de excelente qualidade, para que a estrutura do produto não se degenere ao longo dos protótipos (PAULA FILHO, 2000, p.25-26).

Já na visão de Audy e Prikladnicki (2007, p17-18), uma das principais virtudes desse modelo é permitir o desenvolvimento evolutivo de software e permitir

a interação com o usuário, os requisitos não precisão ser todos definidos no começo. É iterativo, com um marco para avaliação ao final de cada iteração. Além disso, esse modelo busca integração do desenvolvimento tradicional com a prototipação e introduz a análise de risco.

As principais desvantagens do modelo espiral é a difícil compreensão dos grandes clientes de que essa abordagem evolutiva é controlável, pois se um grande risco não for descoberto com certeza ocorrerão problemas. Essa abordagem é mais complexa e tem um custo mais alto que os outros ciclos de vida, podendo não convergir pra uma solução e, dependendo o projeto, a relação custo benefício pode ser duvidosa. Em muitos deles, identificam-se dificuldades na relação com o cliente, que muitas vezes não entende a necessidade de seguir essa abordagem.

CONSIDERAÇÕES FINAIS

Com este estudo realizado, conclui-se que, cada vez mais empresas de desenvolvimento de sistemas necessitam adotar processos de *software* adequados. Isto é importante, pois construir um *software* com qualidade, demanda a utilização e implantação de processos e técnicas de engenharia de *software*. Isto é indispensável para que o produto seja entregue ao cliente dentro do prazo e orçamento planejado, alcançando a qualidade esperada.

A adoção do ciclo de vida em um projeto de *software* é importante para se ter a visão completa do desenvolvimento do *software*. Com isto, é possível definir etapas que abrangem desde a análise dos requisitos até a entrega do *software* para o cliente, além de reduzir o tempo gasto no desenvolvimento e também minimizar custos projeto.

Espera-se que essa pesquisa bibliográfica possa contribuir em futuros trabalhos a serem desenvolvido nesta área de planejamento e construção de software, assim auxiliando desenvolvedores acadêmicos ou comerciais, em suas escolhas de ciclos de vida. Mostrando supostos benefícios e desvantagens de cada modelo, é possível facilitar o entendimento dos processos, visando melhorias na aplicação dos ciclos de vida nos projetos de software, pois para cada tipo de requisito do sistema exige-se um estudo, tendo-se que analisar as melhores

condições do ciclo a ser adotado, assim pode-se obter um produto final com mais agilidade e perfeição.

Uma das principais limitações que se enfrenta ao fazer estudos sobre ciclo de vida de *software*, é a amplitude e a importância das informações encontradas e analisadas, pois ao resumi-las tem que se ter muita cautela para não retirar suas partes cruciais e principais. Apesar das limitações que todo trabalho desta natureza envolve, acredita-se que alguns caminhos foram percorridos para melhor compreensão dos modelos de ciclos de vida. Tem-se clareza de que muito precisa ser pesquisado ainda, pois são inúmeros os princípios e as vantagens destes processos.

Portanto, uma das atividades mais importantes para melhorar os projetos futuros é o levantamento das lições aprendidas com os projetos anteriores, quais superaram as expectativas, quais não atenderam e poderiam ter sido feitas de outra forma, e o que poderia ter feito de maneira diferente para melhorar.

Com o conhecimento adquirido por intermédio dos levantamentos bibliográficos que foram realizados neste artigo, visa-se, em projetos futuros, colocar em prática cada ciclo de vida citado para se tirar conclusões concretas de qual ciclo é mais viável e qual e mais propenso a falhas, o ciclo mais seguro e como identificar cada uma dessas informações.

REFERÊNCIAS

ALVES, Rafael; VANALLE, Rosângela. **Ciclo de Vida de Desenvolvimento de Sistemas: Visão Conceitual dos Modelos Clássico, Espiral e Prototipação**. In: XXI Encontro Nacional de Engenharia de Produção (ENEGEP), Salvador, Brasil, 2001.

AUDY, Jorge; PRIKLADNICKI, Rafael. **Desenvolvimento Distribuído de *Software***. Rio de Janeiro: Elsevier, 2007.

CARVALHO, Ariadne Rizzoni; CHIOSSI, Thelma dos Santos. **Introdução à Engenharia de Software**. São Paulo: Unicamp, 2001.

LAGUNA, Fabrício; KERBER, Claudio. **Um Guia para o Corpo de Conhecimento de Análise de Negócios.** Toronto: *International Institute of Business Analysis*, 2011.

LARMAN, Craing. **Utilizando UML: Padrões.** 3ª Ed., São Paulo: Bookman, 2007.

LOURENÇO, Fabrício; BENINE, Márcio. **Estudo do Ciclo de Vida do *Software* em uma Empresa de Desenvolvimento de Sistemas**. Linguagem Acadêmica, vol.1, n.2, p.183-201, 2011.

MARTINS, José Carlos Cordeiro. **Gerenciando Projetos de Desenvolvimento de *Software* com PMI, RUP e UML**. 5ª Ed., Rio de Janeiro: Brasport, 2010.

PAULA FILHO, Wilson de Pádua. **Alguns Fundamentos da Engenharia de *Software***. Rio de Janeiro: LTC, 2000.

PAULA FILHO, Wilson de Pádua. **Engenharia de *Software*: Fundamentos, Métodos e Padrões**. Revista Engenharia de Software, vol.1, p.4-8, 2008.

PRESSMAN, Roger. **Engenharia de Software**. São Paulo: Makron Books, 1995.

PRESSMAN, Roger. **Engenharia de *Software***. 7ª Ed., São Paulo: MacGraw-Hill, 2011.

RAMOS, Ricardo Argenton. **Treinamento Prático em UML**. São Paulo: Digerati Books, 2006.

REZENDE, Denis Alcides. **Engenharia de *Software* e Sistemas de Informação**. 3ª Ed., Rio de Janeiro: Brasport, 2005.

SCHACH, Stephen. **Engenharia de *Software*: Os Paradigmas Clássicos e Orientado a Objetos**. 7ª Ed., Porto Alegre: AMGH, 2010.

SOMMERVILLE, Ian. **Engenharia de *Software***. São Paulo: Addison Wesley, 2003.

YOURDON, Edward. ***Modern Structured Analysis***. London: Prentice-Hall International, 1989.