# Introdução

“Software é um lugar onde sonhos são plantados e pesadelos são colhidos...” (PRESSMAN apud Cox, Brad J., 2009, p. 3).

O software é uma coleção de instruções e dados que permitem que usuários interajam com o computador ou que o computador execute trabalhos específicos para eles. Sem o software o computador seria inútil (COMPUTERHOPE, 2012).

Para Rouse(2006),  software é o termo comum utilizado para diversos tipos de programas utilizado para operar computadores e dispositivos eletrônicos. Eles diferem do hardware que por sua vez são as partes físicas desses equipamentos como monitores, teclados, telas, mouse, etc.

Pressman (2009, p. 3) diz que o software entrega o produto mais valioso do nosso tempo: informação. Essa informação é proveniente de diversas fontes e serve para diversos fins, desde a pessoal conta bancária, sistemas corporativos, até softwares de atividades específicas. O autor define software como sendo: “Instruções (programas de computador) que quando executadas provêm características, funções e performance desejadas, bem como suas estruturas de dados que habilitam o programa a manipular informações adequadamente, e informação descritiva nas formas impressa ou virtual que descrevem a operação e uso do programa” (PRESSMAN, 2009, p. 4).

“O software esteja presente em quase toda a vida moderna, desde o controle de usinas hidrelétricas à armazenagem de registro de talões de cheque” (SEBESTA, 2003, p. 19). Corrobora com essa visão o ilustre autor Summerville (2011, p. 4): “nós não podemos andar no mundo moderno sem software”. O autor afirma que softwares não estão limitados pela lei da física, não sofrem restrições das propriedades de materiais, logo não há limites naturais para a potencialidade de um software. Porém, por causa dessa falta de restrições físicas, os softwares podem tornar-se complexos, difíceis de entender, caros para serem adaptados e muitas vezes insatisfatórios.

O bom software na definição de Summerville (2011, p. 6) deve entregar a funcionalidade e performance desejada para o usuário e deve ser sustentável, confiável e usável. Algo muito simples de ser dito, porém difícil de ser cumprido. Boehm (2006, p. 13) diz que nos anos 60 popularizou-se a técnica de desenvolvimento “code and fix” onde o software é construído, entregue, o usuário descobre uma série de problemas, o software é então reconstruído e o ciclo continua até o encerramento do projeto e, cinquenta anos depois, segundo o próprio autor essa abordagem ainda é bastante popular o que atrapalha que projetos de software alcancem a definição de Summerville.

Outras fontes de problemas que ocorrem na criação de software são apresentadas por Charrete (2005, p. 3), dentre elas: Metas de projeto não realistas ou não articuladas, estimativas inapropriadas de recursos, fraca comunicação sobre o andamento do projeto, riscos não gerenciados, fraca comunicação entre cliente, desenvolvedores e usuários, uso de tecnologia imatura, inabilidade de lidar com a complexidade do projeto, práticas de desenvolvimento inadequadas, fraco gerenciamento de projeto, política dos envolvidos no processo, pressões comerciais e má definição dos requisitos.

“Aos requisitos estão associados os principais problemas do desenvolvimento de Software. Requisitos que não refletem as reais necessidades dos usuários, incompletos e / ou inconsistentes, mudanças em requisitos já acordados e a dificuldade para conseguir um entendimento comum entre usuários e desenvolvedores são as principais dificuldades provocando retrabalho (...) e a insatisfação de clientes e usuários de Software” (BLASCHEK, 2012, p.1).

Já para Brooks (1995, p. 14): “Mais projetos de software se deram mal por falta de controle de tempo do que por todas as outras causas combinadas”.

Sejam quais forem as origens do problema, a correta gestão do processo construtivo de sistemas poderá evitar: “Incorreta implementação das regras de negócio; Performance inadequada; apresentação confusa da informação; pouca preocupação com a usabilidade; dificuldade de manutenção e entendimento pela equipe técnica; não confiabilidade dos resultados; suporte inadequado às necessidades de negócio” (RICE, 2006).

Segundo o Chaos Report publicado pela Standish Group (2009) apenas 32% dos softwares produzidos são um sucesso: Atendem às expectativas do cliente no prazo e orçamento correto. Embora esse número seja contestado por autores como Glass (2005, p. 112), que conclui que há muitos mais sucessos que aqueles estipulados pelo Standish Group a falta da gestão adequada ainda faz com que muitos erros sejam cometidos durante o processo.

O que ocorre é que o desenvolvimento de softwares é uma atividade laboriosa, complexa, cara, com alto grau de criatividade em que ao longo dos anos é tratada como uma profissão técnica. Esse tratamento produz um efeito interessante. Tecnicamente a engenharia de software evoluiu bastante, dos antigos sistemas de cartões perfurados para o processamento e aplicações em *clouding computing*, porém alguns problemas perpetuam: A dificuldade de extrair as necessidades dos usuários (BLASCHEK, 2012, p. 1), os problemas na comunicação (VALLE et al., 2010, p. 66) e a implementação insatisfatória dos produtos de software são algumas das grandes causas da insatisfação de usuários, clientes e desenvolvedores de software.

Para dirimir esses problemas, há disponível no mercado uma ampla gama de metodologias, frameworks e guias de melhores práticas. Esse trabalho apresenta a definição de software, como é o desenvolvimento de software atual e como as metodologias ágeis de desenvolvimento de software podem auxiliar as empresas a entregar produtos de alto valor agregado à organização de forma rápida e eficaz. O grande foco dele é apresentar como o PMBOK pode auxiliar essas metodologias ágeis a tratar aspectos gerenciais que não são por elas abordadas.

# Desenvolvimento de Software

O desenvolvimento de software remonta desde os anos 40 (BIZYMONS, 2012), já o termo engenharia de software nasceu durante os anos 50 (BIZYMONS, 2012). O problema básico com a engenharia de software é que ao contrário das outras engenharias, ela cria algo que não pode ser vista no mundo físico. Tudo é feito virtualmente ou no papel (planos). Por isso há dificuldade em implementá-la sem um modelo adequado (BIZYMONS, 2012).

Para Pressman (2009, p. 13), a realidade do mundo em que o software está inserido leva a uma conclusão: “software em todas as suas formas e por todos os seus domínios de aplicação devem ser tratados como engenharia”. O autor diz que para criar sistemas com sucesso, a equipe deve pensar em quatro camadas fundamentais. A primeira camada é aquela responsável por prover foco na qualidade, toda organização deve ser comprometida com a qualidade, pois é ela que proverá a satisfação de clientes e usuários. A segunda camada é a de processos, que funciona como uma espécie de cola que junta a camada de tecnologia e possibilita o desenvolvimento racional e oportuno de softwares. A terceira camada é a de métodos que dita o como fazer. A última são as ferramentas que, como o próprio nome diz são aquelas que suportam os métodos e processos na construção do software.

Sommerville (2011, p.4) diz que a engenharia de software nos ajudar a construir sistemas maiores, mais complexos, mais adaptáveis à mudança de demandas. Ela destina-se a apoiar o desenvolvimento profissional de software, em vez de programação individual. Inclui técnicas que suportam a especificação de programas, projeto e evolução (SOMMERVILLE, 2011, p. 5).

Boehm (1984, p.1) complementa a necessidade do rigor da engenharia dizendo: “a engenharia de software deve existir, pois, claramente o mundo possui recursos limitados. Nunca há tempo ou dinheiro para que toda funcionalidade desejada seja inserida em um produto de software”.

Já Sonmez (2011) acredita que o desenvolvimento de software não poderia ser comparado à engenharia. Para ele a engenharia existe há centenas de anos e amadureceu ao ponto de tornar-se uma ciência mensurável, porém software apareceu apenas há algumas décadas, portanto ainda é relativamente imaturo para ser considerado uma disciplina de engenharia.

O desenvolvimento tradicional baseado na engenharia de software é descrito pelo modelo de cascata, Sommerville (2011, p.30) apresenta o fluxograma básico desse modelo:

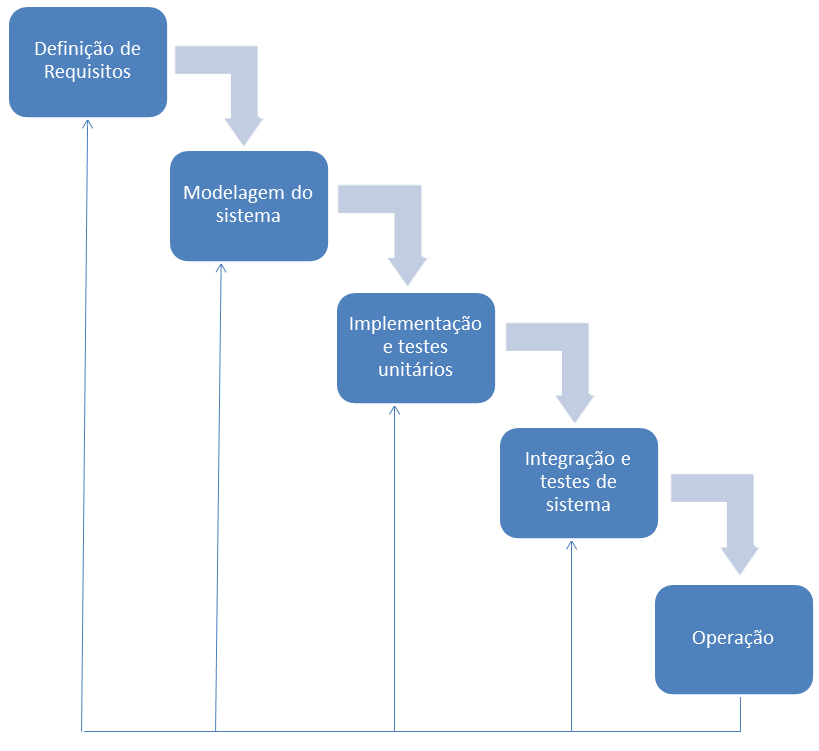


Figura 1Modelo em Cascata de Sommerville (2011, p.30).

Esse modelo se chama cascata devido ao seu comportamento. Ele funciona como se fosse uma fábrica onde cada atividade produz insumo para a próxima atividade. Esse insumo é trabalhado, processado e um resultado é gerado. Esse resultado serve de insumo para a próxima atividade até colocar o projeto inteiro em operação. “A princípio, o resultado de cada fase é um ou mais documentos que são aprovados. A próxima fase não deve começar até que a fase anterior se encerre” (SOMMERVILLE, 2011, p.31).

O modelo de Sommerville permite que, se detectado algum problema, o software retorne para uma etapa anterior, para ser “consertado”. Após o conserto ele retoma o fluxo principal.

As fases definidas por Sommerville são:

Definição dos requisitos. “Quando os serviços de sistemas, restrições e objetivos são definidos pelos analistas e usuários. Todos os requisitos são definidos em detalhes e servem como especificação de sistemas” (SOMMERVILLE, 2011, p.31).

Modelagem do sistema. “É o processo de modelar o sistema para que ele atenda os requisitos de hardware ou software para estabelecer toda a arquitetura de sistemas. Envolve a identificação, abstração e relacionamento dos requisitos de sistema” (SOMMERVILLE, 2011, p.31).

Implementação e testes unitários. Para Sommerville (2011, p. 31) é a criação de software em si, quando a modelagem é criada e testada.

Integração e testes de sistemas. Durante a implementação o sistema é dividido em pequenas partes e essas partes devem agora ser integradas para formar a versão final do software. Após a integração o sistema deve ser testado para verificar se ele está de acordo com os requisitos definidos (SOMMERVILLE, 2011, p.31).

Operação. “Normalmente, mas não necessariamente, esse é a atividade mais longa do ciclo de vida do software. O sistema é instalado e colocado para uso. A manutenção ocorrerá para corrigir erros (...) ou melhorar a implementação de unidades do sistema ou caso algum novo requisito seja descoberto” (SOMMERVILLE, 2011, p.31).

Pressman (2009, p. 15) define um modelo em cascata com algumas diferenças. Mais voltado para aspectos de gestão.

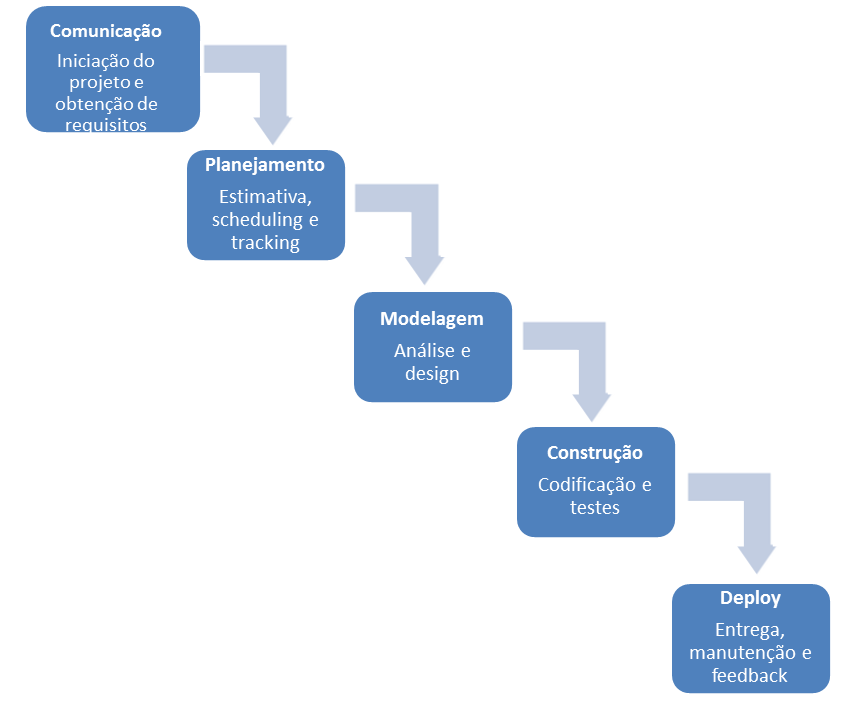


Figura 2Modelo em cascata de Pressman (2009, p. 39).

O modelo de Pressman tem um funcionamento similar ao de Sommerville. A primeira atividade é a de comunicação. “Antes do início de qualquer trabalho é criticamente importante se comunicar e colaborar com o cliente (e outros *stakeholders*). A intenção dessa etapa é entender os objetivos dos stakeholders o obter os requisitos que definam as funcionalidades e funções do sistema” (PRESSMAN, 2009, p.15).

Após a etapa de comunicação inicia-se a etapa de planejamento. “Qualquer jornada complicada pode ser simplificada se um mapa existir. (...) um plano de projeto de software define o trabalho da engenharia de software descrevendo as tarefas técnicas que devem ser realizadas, os riscos, os recursos que serão necessários, o produto que será criado e o cronograma do trabalho” (PRESSMAN, 2009, p.15).

A modelagem é para Pressman (2009, p.15) a etapa em que o planejamento é colocado em modelos para melhor entendimento do problema pela equipe técnica.

Modelagem feita, agora é hora de começar a construção. “A atividade que combina geração de código (manual ou automatizada) e testes necessários para descobrir erros no código” (PRESSMAN, 2009, p. 15).

Deploy. É a ultima fase do modelo. “O software (...) é entregue ao cliente que avalia o produto entregue e provê feedback...” (PRESSMAN, 2009, p. 15).

# Facilidades e Dificuldades no Desenvolvimento de Software

Algumas facilidades podem ser apontadas utilizando o modelo de desenvolvimento de software tradicional em cascata.

Como mencionado por Alam (2012), o modelo em cascata é fácil de ser implementado e exige poucos recursos.

Outra vantagem levantada pela Tech Republic (2006), é que nesse modelo de desenvolvimento há muita disciplina, pois cada fase define claramente um início e fim que deve ser definido de forma clara e objetiva. O retorno para uma etapa anterior é demasiadamente custoso o que faz com que as equipes que se disciplinem para evita-lo.

Rouse (2007) diz que o software gerado por esse modelo é altamente previsível, é possível gerar um cronograma adequado com *deadlines* com cada etapa do desenvolvimento e do produto bem descrita, como se fosse um carro andando por um lava-jato.

Este método é melhor do que o antigo *cowboy coding*, chamado por Boehm (2006, p. 13) de code and fix, pois ele define uma estrutura para o desenvolvimento de programas.

Entretanto, alguns empecilhos são trazidos por esse modelo.

Primeiramente, como afirmado por Alam (2012), “Ironicamente, a maior desvantagem é uma de suas maiores vantagens. Você não pode ir um passo para trás. Se a fase de modelagem der errado, as coisas podem tornar-se muito complicadas na fase de implementação.” Na prática é muito difícil perceber um erro até que o produto seja utilizado pelo cliente.

“Projetos reais raramente seguem um fluxo sequencial e interações nesse modelo são tratadas de forma indireta. Mudanças durante o projeto podem causar confusão” (SPARROW, 2012).

“Mesmo uma pequena mudança em qualquer etapa anterior pode causar grande problema para as fases subsequentes já que todas  são dependentes umas das outras” (SPARROW, 2012).

Como as fases são bem delimitadas, erros não capturados logo no início tornam-se extremamente custosos quando são encontrados em etapas mais tardias. King e Marasco (2009) dizem que um erro encontrado na fase de operação custa até 80 vezes o valor de um erro encontrado na fase de análise de requisitos. Cordeiro (2002, p. 22) afirma que encontrar um erro na operação pode ser 200 vezes maior do que encontra-lo logo no início.

Outra grande dificuldade enfrentada ao utilizar o processo tradicional é que “ele não pode prover *feedback* sobre pressupostos (assumidos durante o levantamento de requisitos) ou funcionalidades não detectadas e necessidades dos usuários” (LAPLANTE et al. 2004, p. 10).

Todas as desvantagens acima são importantes, porém a mais importante é que “esse modelo de desenvolvimento assume que os requisitos são definidos, estáveis, e totalmente evoluído antes da análise começar, porque o progresso do desenvolvimento é linear através das fases, desde os requisitos até o *deploy* e operação. Uma fase é revisada somente se os artefatos criados nessa fase falharem na inspeção, revisão ou teste” (LAPLANTE et al. 2004, p. 10). Os autores brincam ainda dizendo: “se você entrar em uma disputa com alguém sobre esse fato, lembre-os que a água não flui para cima da cascata (sempre para baixo)” (LAPLANTE et al. 2004, p. 10). Os autores concluem dizendo que “a realidade moderna do desenvolvimento de software é que mudança é inevitável e, portanto, deve ser explicitamente acomodada no ciclo de vida. Não é um erro que deve ser consertado, é um aspecto natural da construção de um sistema (...). O quanto mais nós entendemos algo mais nós percebemos falhas nas nossas suposições e concepções” (LAPLANTE et al. 2004, p. 10 e 11)

# Necessidade do uso de metodologia

Metodologia é: “o estudo científico através do método; a arte de guiar o espírito na investigação da verdade” (MICHAELIS, 2012). “O corpo de métodos, regras, e postulados empregados por uma disciplina, um procedimento ou conjunto de procedimentos” (WEBSTER, 2012).

“A construção de sistemas é complexa, pois deve lidar com requisitos intangíveis, restrições de integridade e a necessidade de um vasto conhecimento sobre a aplicação para que as interações esperadas entre o software e o ambiente possam ser adequadamente descritos. Quando os requisitos não são totalmente compreendidos, registrados e comunicados para a equipe de desenvolvimento, muito provavelmente, haverá discrepância entre o que o sistema construído faz e o que ele deveria fazer” (LEITE, 2012). Caetano (2009) completa a informação dizendo: “As metodologias de desenvolvimento de software servem para não tornar a tarefa, complexa por natureza, um verdadeiro caos”.

Khrishna (2011) fala que, programar é uma habilidade que pode ser aprendida de forma semelhante a uma matéria escolar como matemática, a prática faz com que o programador fique cada vez melhor. Assim como na matemática 1+2 sempre será igual a 3, na programação a instrução ou está certa, ou está errada. Não há outra solução possível. A autora alerta que já o desenvolvimento de sistemas é algo um tanto mais complexo. Desenvolver software é entender seus clientes e usuários, conversar com eles, observá-los, trabalhar em equipe e gerar um produto de qualidade. Não há necessariamente a resposta certa. Prever comportamento não é possível. Completa essa informação Phillips (2010): “As metodologias de software se preocupam com o processo de criação do software – não se preocupa tanto com a parte técnica, mas sim com aspectos organizacionais”.

Para Pavim (2006), uma metodologia deve ser utilizada, pois, as organizações tem a necessidade de adotar um padrão para o desenvolvimento de software em equipe. É necessário que ela estabeleça padrões de processos, comunicação, artefatos, marcos, etc. Além disso, o autor afirma que a metodologia é importante para a organização, compreensão, portabilidade, documentação e qualidade do código-fonte gerado; para auxiliar a administração de projetos de software; e adequar-se aos padrões mínimos de qualidade definidos na Engenharia de Software.

# Tipos de Metodologias

Existe uma infinidade de metodologias de desenvolvimento de software. Nesse tópico serão apresentadas as mais relevantes. A metodologia ágil será apresentada em um tópico subsequente.

A primeira metodologia que será descrita é a *Ad-hoc*, que não chega a ser de fato uma metodologia, pois como próprio nome diz ela é criada para resolver um problema específico e depois é descartada. Ela aparece nesse tópico, pois ainda é utilizado por mais de 30% das empresas segundo Forester (2009). É a “metodologia” que possui maior utilização dentre as empresas pesquisadas, a segunda maior, modelo iterativo, detêm apenas 16% do *Market share*. Nessa metodologia quando uma nova demanda chega, a equipe define na hora o que será feito, quais artefatos e quais os processos serão necessários. Após o final do projeto todo o método é descartado. Alguns problemas ficam claros com a adoção desse tipo de programação. Como citado por Phillips (2010) a imprevisibilidade dos resultados, dependência total da equipe do projeto e falta de visão de valor para o negócio. Spolsky (2009) vai além e afirma que equipes que trabalham dessa forma são profissionais de fita adesiva e WD-40, pois estão sempre remendando seus códigos.

# O porquê do Agile

# PMBOK

# PMBOK e Agile