# 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 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).

Summerville (2011, p. 4) diz que “nós não podemos andar no mundo moderno sem software”. O autor informa 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. Como informa Boehm (2006, p. 13), 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 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 apresentados por Charrete (2005, p. 3) são: 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.

# Engenharia de Software

A engenharia de software é importante, pois ela define, processos, métodos e ferramentas necessários para o correto desenvolvimento de software (PRESSMAN, 2009, p. 14). Esse capítulo irá conceituar a engenharia de software que é a responsável pela criação, aperfeiçoamento, implementação e manutenção de sistemas de informações profissionais.

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).

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.

Já 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”.

A engenharia de software nos ajudar a construir sistemas maiores, mais complexos, mais adaptáveis à mudança de demandas (SOMMERVILLE, 2011, p. 4).

A engenharia de software se destina 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).

Já para Boehm (1984, p.1), 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.

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 complicado. 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.

Para Pressman (2009, p. 12), o 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.

Para esse trabalho será utilizada para definir o desenvolvimento de software as definições de Pressman que diz: “A engenharia de software é a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção do software” (PRESSMAN, 2009, p.13) e de Sommerville: “A Engenharia de software é uma disciplina da engenharia que se preocupa com todos os aspectos da produção de software, desde as fases iniciais de especificação do sistema até a manutenção do mesmo depois de ter entrado em uso”.

# Facilidades e Dificuldades no Desenvolvimento de Software

Conforme afirmado por Sommerville (2011, p. 4): “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”. Isso faz com que 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).

Esse capítulo abordará algumas facilidades e dificuldades no desenvolvimento de software.

Dentre as principais facilidades podem ser descritas: A quantidade de ferramentas (linguagens de programação) disponíveis, o reuso de partes de outros sistemas, o custo decrescente de hardware e as características do profissional de software. Essas vantagens serão agora melhor detalhadas.

## Facilidades

A quantidade de linguagens de programação disponíveis. As linguagens de programação são as principais ferramentas de desenvolvimento de software. O site Scriptol (2012) contabiliza mais de 400 linguagens de programação disponíveis para serem utilizadas nas mais variados tipos de problemas. Desde as linguagens mais básicas até aquelas utilizadas para manipular Big Data e sistemas críticos.

O reuso de partes de outros sistemas. Os especialistas em desenvolvimento de software sabem que não criam um sistema completo, do zero. Ao contrário eles reutilizam suas soluções que funcionaram no passado e que pode ser aproveitada no novo software (GAMMA et al., 1994, p. 11).

O custo decrescente de hardware o que possibilita que mais e mais pessoas tenham equipamentos eletrônicos e aumenta a demanda de software. Uma tabela comparativa desenhada pela Software Metrics (2012) apresenta o comparativo entre o índice de custo de hardware e a estimativa de softwares desenvolvidos por ano.

| **Ano** | **Índice do Custo de Hardware** | **Quantidade de Softwares** |
| --- | --- | --- |
| 1960 | 678 |  |
| 1970 | 75 |  |
| 1980 | 10,5 | 150 |
| 1990 | 0,5 | 19.000 |
| 2000 | 0,3 | 42.000 |

Tabela 1Estimativa de custo de hardware e quantidade de softwares

Os desenvolvedores de software são classificados como profissionais de conhecimento (do inglês (knowledge worker) e como tal são motivados não só pelo salário, mas principalmente conquistas e capacidade de resolver problemas (AMABILE et al., 2010). Visão corroborada por Asproni (2004, p. 3) que classifica como o item motivacional mais importante para esses profissionais a conquista, a possibilidade de crescimento na carreira e o trabalho em si. Nas pesquisas realizadas o salário aparece apenas na décima posição.

## Dificuldades

Porém, nem tudo são flores no processo de desenvolvimento de softwares, há grandes dificuldades que devem ser superadas. Dentre elas, algumas das mais importantes são: Restrições inerentes a projetos, a dificuldade de comunicação, complexidade dos softwares atuais, redução de custos de TI, necessidade de conformidade de padrões de mercado.

As restrições inerentes ao projeto são definidas pelo GP3 (2012) como um conjunto de fatores que tomam grande parte do tempo do gerente e da equipe do projeto. Eles chamam o conjunto dessas restrições de “restrição tripla” — escopo, tempo e custo do projeto — de necessidades conflitantes do projeto. Considera-se ainda a qualidade do projeto como uma quarta variável, a qual é afetada pelo balanceamento das três demais. O site afirma que “Existe um equilíbrio natural entre essas quatro ‘forças’ dentro do projeto, que é estabelecido no momento em que as linhas de base de escopo, tempo e custo são acordadas entre as partes envolvidas na realização do projeto. A partir desse ponto, mudanças em uma dimensão (por exemplo, escopo) serão refletidas em uma ou mais entre as três outras” (GP3, 2012).

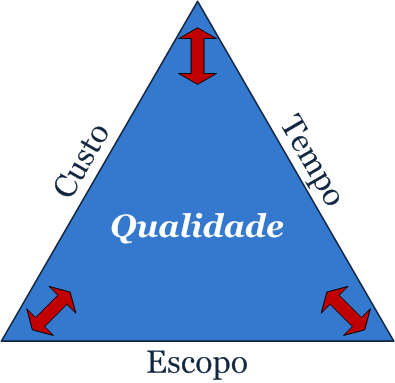


Figura 1Triângulo das restrições de projetos

O PMI (2008, p.6) em seu famoso livro de conhecimento PMBOK vai além e diz que os projetos sofrem pelo menos as seguintes restrições: Escopo, custo, tempo, qualidade, risco e recursos. A renomada instituição afirma ainda que esses não são as únicas restrições possíveis podendo existir ainda mais.

Boehm corrobora com a informação e diz: “No processo de criação de software (e em todo mundo corporativo) nós lidamos com restrição de recursos. Nunca há tempo ou dinheiro suficiente para cobrir todas as funcionalidades que são desejadas no produto de software” (BOEHM, 1984, p.1).

Klawe (KLAWE et al., 2005, p.27) ressalta a restrição de recursos humanos. Cada vez há menos profissionais disponíveis. O mesmo diz prof. de Stanford Mehran (2012), apenas 6% dos alunos da graduação da ilustre universidade que iniciam o curso de Computação Científica irão tornar-se “engenheiros” de software.

Além das restrições de projetos, outra grande dificuldade é a comunicação. Ela é necessária para estabelecer confiança entre a equipe de desenvolvimento e seus clientes, para obtenção de requisito, feedback, negociação, planejamento, auxiliar na execução entre outros, porém embora seja algo primordial, não é simples de ser estabelecida. Passadori (2003) evidencia alguns fatores que atrapalham a comunicação: Desorganização de ideias, vícios de linguagem, dificuldade com o vocabulário, inadequação de recursos audiovisuais, prolixidade, excessiva objetividade, arrogância e prepotência e excesso de humildade. As consequências geradas pela baixa comunicação podem ser falha no levantamento de requisitos (BLASCHEK, 2012, p.1), informação desnecessária, criação de “silos de informação”, interrupções, informações que chegam desatualizadas, reuniões sem foco (GRIMES, 2009). Para (DOWN, 2008, p. xxix) 90% das razões de falha de um projeto podem ser representadas por problemas de comunicação. Cockburn (2001, p. 84) apresenta um gráfico em que contraria a ideia básica de uma boa comunicação. Para ele a comunicação escrita possui uma eficiência muito baixa e a comunicação frente a frente, é a mais eficiente.

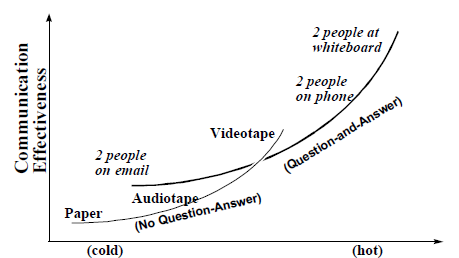


Figura 2Gráfico de Eficiência da Comunicação de Cockburn

Outra grande dificuldade é a complexidade dos softwares.

# Necessidade do uso de metodologia

As dificuldades mencionadas no tópico anterior leva à necessidade de utilização de metodologia para desenvolvimento de sistemas que é “o estudo dos caminhos, dos instrumentos usados para se fazer pesquisa científica, os quais respondem o como fazê-la de forma eficiente” (.

# Tipos de Metodologias

# O porquê do Agile

# PMBOK

# PMBOK e Agile

# Bibliografia

AMABILE, Teresa M. KRAMER, Steven J. The HBR List: Breakthrough Ideas for 2010. Harvard Business Review. In: http://hbr.org/2010/01/the-hbr-list-breakthrough-ideas-for-2010/ar/1 . Acessado em dezembro de 2012: Publicado em 2010.

BIZYMONS. History of software development. In:<http://www.bizymoms.com/computers-and-technology/software-development.html>. Acessado em dezembro de 2012.

BLASCHEK, José R. Gerência de Requisitos: O principal problema dos projetos de software. Developers’ Magazine. Rio de Janeiro - RJ. In:<http://www.bfpug.com.br/islig-rio/Downloads/Ger%C3%AAncia%20de%20Requisitos-o%20Principal%20Problema%20dos%20Projetos%20de%20SW.pdf>: Acessado em setembro de 2012.

BOEHM, Barry W. A view of 20th and 21st century software engineering. University of Southern California – USA: 2006.

BOEHM, Barry W. Software Engineering Economics.IEEE Transactions. In: http://userfs.cec.wustl.edu/~cse528/Boehm-SE-Economics.pdf : 1984

BROOKS, Jr, Frederick P. The Mythical Man-Mounth. Ed. Addison Wesley Longman. 2nd Edition:  1995.

CHARRETE, Robert N. Why Software Fails. in [http://spectrum.ieee.org/computing/software/why-software-fails/](http://spectrum.ieee.org/computing/software/why-software-fails/3); Acessado em setembro de 2012: Publicado em 2005.

COCKBURN. Alistar. Agile Software Development. Addison-Wesley Professional. Boston – USA: 2001.

COMPUTERHOPE.  Software. In:<http://www.computerhope.com/jargon/s/software.htm>; Acessado em novembro de 2012.

DOWN, Willian. TAYLOR, Bruce. Project Management Communications Bible. Wiley Publishing, inc. 1st edition. Indianapolis – USA: 2008.

GAMMA, Erich. HELM, Richard. JOHNSON, Ralph. VISSIDES, John. Design Patterns: Elements of Reusable Object-Oriented Software. Ed. Addison-Wesley: 1994.

GLASS, Robert L. IT failure rates - 70% or 10-15%. IEEE Software. Maio / Junho: 2005.

GP3. O Triângulo das Restrições de Gerenciamento de Projetos. In: <http://www.gp3.com.br/index.php?/o-triangulo-das-restricoes-de-gerenciamento-de-projetos.html>. Acessado em dezembro de 2012.

GRIMES, Andrew. Five dangers of poor project communication. In: <http://www.semantico.com/2009/11/five-dangers-of-poor-project-communication/>. Acessado em dezembro de 2012. Publicado em 2009.

PASSADORI, Reinaldo. Problemas mais comuns de comunicação. In: <http://www.consultores.com.br/artigos.asp?cod_artigo=244>. Acessado em dezembro de 2012. Publicado em outubro de 2003.

PMI, Project Management Institute. PMBOK – Project Management Book of Knowledge. 4th edition. PMI. Pennsylvania – USA: 2008.

PRESSMAN, R.S. Software Engineering – A Practitioner’s Approach. 7 edition. McGraw-Hill Science: 2009.

KLAWE, Maria. SHNEIDERMAN, Ben. Crisis and Opportunity in Computer Science. Communications of ACM. Novembro: 2005

KRISHNA. Programming is Easy, Software Development is Hard. In: <http://www.thoughtclusters.com/2011/01/programming-is-easy-software-development-is-hard/#comment-22034>. Acessado em dezembro de 2012: Publicado em 2011

RICE, Randall W. The 20 Most Common Software Problems. in:<http://www.riceconsulting.com/articles/20-most-common-software-problems.htm>. Acessado em setembro de 2012: Publicado em 2006.

ROUSE, Margaret. Software. In:<http://searchsoa.techtarget.com/definition/software>. Acessado em novembro de 2012: Publicado em 2006.

SAHAMI, Mehran. Lecture 1 | Programming Methodology (Stanford). In:<http://www.youtube.com/watch?v=fiSoT3Wi0N8&feature=share&list=PLF1F29EF4D95C7957>. Acessado em dezembro de 2012.

SCRIPTOL. List of Programming Languages in Alphabetical Order. In: <http://www.scriptol.com/programming/list-programming-languages.php>. Acessado em dezembro de 2012.

SEBESTA, Robert W. Conceitos de linguagem de programação. 5ª Edição. Bookman Ed: 2003.

SOFTWARE METRICS. Hardware Costs and Software Development. In: <http://www.softwaremetrics.com/Articles/HardwareandSoftware.htm>. Acessado em dezembro de 2012.

SOMMERVILLE, Ian. Software Engineering. 9th ed. Pearson: 2011.

SONMEZ, John. Why Software Development Will Never be Engineering. In:<http://elegantcode.com/2011/06/22/why-software-development-will-never-be-engineering/> : 2011.

VALLE, André Bittencourt. SOARES, Carlos Alberto Pereira. FINOCCHIO JR., José. SILVA, Lincoln de Souza Firminio. Fundamentos do gerenciamento de projetos. 2a. edição. FGV Editora. Rio de Janeiro: 2010.