Fatores que Afetam a Produtividade em Projetos de *Software*: Uma Visão Geral

José Adson O. Guedes da Cunha, Hermano Perrelli de Moura

Centro de Informática – Universidade Federal de Pernambuco
Recife, Brasil, 50732-970

{jaogc, hermano}@cin.ufpe.br

Abstract. *With the growth of the software development market and the high degree of competitiveness between the companies, the project managers are looking for ways to deliver quality products faster. In general, the development time and product outcome of any software project can be influenced by five groups of factors: the processes they use, the underlying technology, the people involved, the organizational culture and the product characteristics. Ahead of this, it is responsability of the project manager to balance and manager the necessities in accordance with the characteristics of the project. This work proposes to compile the factors that affect productivity based on the existing studies in the literature, classifying them in accordance with the established groups above.*

Resumo. *Com o crescimento do mercado de desenvolvimento de software e o alto grau de competitividade entre as empresas, os gerentes de projeto estão procurando maneiras de entregar produtos de qualidade no mínimo tempo possível. De modo geral, o tempo de desenvolvimento e o resultado de qualquer projeto de software podem ser influenciados por cinco grupos de fatores: os processos usados, a tecnologia adotada, a equipe de pessoas envolvida, a cultura organizacional e as características do produto. Diante disso, cabe ao gerente de projeto balancear e gerenciar as necessidades de acordo com as características do projeto. Este trabalho propõe realizar uma compilação dos fatores que afetam a produtividade baseada nos estudos existentes na literatura, classificando-os de acordo com os grupos estabelecidos acima.*

1. Introdução

No final da década de 40, quando os primeiros computadores eletrônicos começaram a aparecer, a Engenharia de *Software* estava nascendo. Desde aquele tempo, gerentes de projeto têm experimentado diferentes maneiras de se gerenciar o desenvolvimento de *software*. Quase 15 anos depois, um processo que começou com programas em linguagens de máquina de propósitos simples desenvolvidos por um único desenvolvedor cresceu na direção de projetos com duração de anos, compostos por equipes de engenheiros de *software*. Os primeiros sistemas executavam tipicamente em modo *batch* em uma única plataforma, do contrário dos sistemas atuais que devem executar de maneira interativa em múltiplas plataformas e configurações de sistemas operacionais. O desenvolvimento de *software* se expandiu, tornando-se uma indústria

bilionária, envolvida em todo projeto governamental ou do setor privado. Linguagens de baixo nível têm se tornado defasadas diante das numerosas linguagens de alto nível, suportadas por ferramentas poderosas. No entanto, os benefícios conquistados com a relativa facilidade de programação diante das linguagens de alto nível e ferramentas têm sido ofuscados pelo aumento da complexidade dos projetos de *software*.

Atualmente, gerentes de projetos estão constantemente procurando maneiras de entregar produtos de qualidade o mais rápido possível. De maneira geral, projetos de desenvolvimento de *software* são como quaisquer outros tipos de projetos, de modo que existem cronogramas a serem cumpridos, recursos a serem gerenciados e objetivos a serem conquistados. Entretanto, a grande diferença vem do fato de que cada projeto de *software* é único.

De modo que um projeto obtenha sucesso, é preciso que o mesmo crie um produto que atenda as expectativas do cliente, sendo entregue em um prazo adequado. Diante dos recursos limitados e da quantidade ilimitada de trabalho, a produtividade do time responsável pelo projeto se torna um fator crítico para seu sucesso, visto que a eficiência com que os recursos são empregados no desenvolvimento de *software* é um fator primordial para o sucesso do projeto.

Diversos estudos ao longo da história da Engenharia de *Software* procuraram identificar que fatores influenciam na produtividade e qualidade no desenvolvimento de *software*. [Scacchi 1995] realizou um *survey*, no qual discute as várias abordagens de medição da produtividade, bem como os fatores que afetam a produtividade, sugerindo a construção de um sistema de modelagem e simulação da produtividade em projetos de *software* baseado em conhecimento. Em seu *survey*, Scacchi identificou dezoito fatores, dividindo-os em três grupos: os relacionados com o ambiente de desenvolvimento, com o produto a ser desenvolvido e com a equipe envolvida. [Jones 1986] aponta mais de quarenta variáveis que podem influenciar na produtividade do desenvolvimento de *software*. O modelo de estimativa de custo COCOMO II [Boehm 1996] divide os fatores em quatro grupos: relacionados com o produto, com os computadores utilizados no projeto, com a equipe de pessoas envolvida e com o projeto (que inclui práticas de desenvolvimento). Em [White 1999], o autor divide os fatores em quatro grupos: relacionados com o produto, com a equipe de pessoas envolvidas, com a tecnologia utilizada e com o processo utilizado. De acordo com [Chiang 2004], a melhoria na produtividade no desenvolvimento de *software* se dá através do balanceamento de três pilares do gerenciamento de *software*: tecnologia, pessoas e processo. [Clincy 2003] realizou um *survey* com líderes de projeto de diferentes organizações e concluiu que as áreas que mais impactam na habilidade da organização de aumentar sua produtividade são: estrutura e clima organizacional, sistemas de recompensa, processos de desenvolvimento de *software* e uso de ferramentas. Em [Taylor 2005], o autor conclui que, além dos demais fatores, o foco na melhoria da cultura organizacional pode trazer grandes benefícios na produtividade da organização.

Outros trabalhos, além de realizar análises dos vários fatores que afetam a produtividade e qualidade, apontam padrões de comportamento e organização de projetos de *software* que possibilitam alta produtividade. [Bailey et al. 1981] concluiu que a alta produtividade está diretamente relacionada com o uso de metodologia de

desenvolvimento de *software* disciplinada. Já [Behrens 1983] concluiu em sua análise realizada em vinte e cinco projetos de desenvolvimento de *software* que o tamanho do projeto, o ambiente de desenvolvimento e linguagem de programação impactam na produtividade. Em particular, o autor concluiu que pequenos times produzem mais código do que grandes times. [Boehm 1981] reporta que a produtividade em um projeto de desenvolvimento de *software* é mais fortemente afetada por aqueles que desenvolvem o sistema e pela forma como estes estão organizados e gerenciados como time. [Scacchi 1984] complementa esta evidência revisando alguns relatórios sobre gerenciamento de grandes projetos. Nesta revisão, ele confirma que quando projetos são mal gerenciados ou organizados, a produtividade é substancialmente mais baixa do que o normal.

[DeMarco 1999] em seu trabalho que envolveu a análise de aproximadamente 600 desenvolvedores de 92 organizações, defende que os principais problemas de produtividade estão relacionados a fatores humanos e organizacionais e não a fatores técnicos como muitos defendem. O autor apresenta, inclusive, uma série de experimentos realizados com o objetivo de identificar tais fatores e em seus resultados fica explícita tal afirmação. Nestes experimentos, fatores como linguagem de programação adotada, faixa salarial e experiência na função não influenciam significativamente na produtividade. Já outros fatores, como espaço físico individual e nível de ruído no ambiente de trabalho afetam de um modo muito intenso.

Em um estudo realizado com 44 projetos, [Vosburgh 1984] dividiu os fatores que afetam a produtividade de *software* em dois grupos: os que são controlados pelo gerente (relacionados com o projeto) e os que não estão sob o controle do gerente (relacionados com o produto). De acordo com sua análise, o autor indica que a produtividade é determinada em 28% pelos fatores relacionados com o produto e em 37% pelos fatores relacionados com o projeto. Sendo assim, o estudo indica que a melhoria da produtividade está nas mãos dos gerentes de projeto, cabendo aos mesmos balancear e gerenciar as necessidades.

De fato, vários estudos ao longo da história analisaram projetos de desenvolvimento de *software* e apontaram os fatores que afetam a qualidade e produtividade em projetos de *software*. Cada um dos estudos produziu resultados de grande relevância, mas que podem ser combinados de modo a permitir uma análise da sistêmica e correlação entre os fatores. Desse modo, se torna importante ter uma definição clara dos fatores que afetam a produtividade, bem como uma categorização de modo a permitir uma melhor identificação dos mesmos.

Diante disso, este trabalho tem como objetivo obter uma visão geral dos fatores que afetam a produtividade em projetos de *software*, classificando-os de acordo com cinco grupos, os quais, por sua vez, foram identificados a partir dos estudos existentes na literatura, sendo: as características do produto, a tecnologia adotada, a equipe de pessoas envolvida, os processos usados e a cultura organizacional.

As demais seções deste trabalho estão organizadas da seguinte forma: a Seção 2 apresenta algumas definições relacionadas com a produtividade em projetos de *software*, a Seção 3 discute os fatores que afetam a produtividade, de acordo com os

grupos previamente estabelecidos e por fim, a Seção 4 apresenta as conclusões e trabalhos futuros.

2. Definições

A produtividade de *software* não é um conceito simples, sendo assim um assunto que merece discussão, possuindo um papel importante na Engenharia de *Software*.

Nos termos padrões da Economia, a produtividade de um modo geral é a relação entre a quantidade de bens ou serviços produzidos e a despesa ou trabalho necessário para produzi-los [Jones 1996]. Conseqüentemente, a produtividade de *software* é a relação entre a quantidade de *software* produzida e a despesa ou trabalho para produzi-la. Apesar de simples na teoria, na prática tal definição se torna um problema não muito trivial.

A fim de definir produtividade de *software*, precisa-se primeiramente estabelecer uma definição de *software*. No seu nível mais fundamental, *software* é um programa de computador composto por linhas de código. Entretanto, linhas de código não são os entregáveis prioritários de um projeto e seus consumidores geralmente não sabem quantas linhas de código estão contidas no *software* que os mesmos estão adquirindo [Jones 1996]. Diante disso, uma definição mais abrangente de *software* não compreende somente o programa de computador, mas também os procedimentos e documentos relacionados com o programa, que geralmente incluem a documentação dos requisitos, especificações, projeto do *software* e procedimentos direcionados ao usuário final [Mills 1983]. Desse modo, um conjunto completo de documentação provê algo mais tangível como resultado de um projeto de *software*, do que somente o programa em si.

Por outro lado, mesmo considerando o código do programa e a documentação associada como saídas de um projeto de *software*, os mesmos não são de interesse direto do consumidor, visto que o *software* é comprado baseado no que ele faz, e não como ele foi codificado ou documentado. Isso significa que o valor econômico dos bens e serviços consumidos não é medido nas mesmas unidades da produção do *software* [Sidler 2002]. Conseqüentemente, uma medição diferente do *software* precisa ser usada, de modo a obter uma definição significativa da produtividade. Essa medida necessita refletir o valor do serviço do *software*, isto é, a função que o mesmo se propõe a desempenhar [Jones 1996].

Baseando as medições no valor utilitário do *software*, a suposição original pode ser revista e a produtividade de *software* definida como a relação entre o valor funcional do *software* produzido e o trabalho ou despesa para produzi-lo. Tal definição nos permite medir a produtividade baseada no valor oferecido ao consumidor do *software*, sendo assim mais realístico do que se basear nas linhas de código produzidas [Mills 1983]. Dessa forma, poderiam ser considerados como valor funcional o conhecimento e experiência que o desenvolvimento do produto incorporou na organização; a qualidade do produto (interna ou externa); o valor para o cliente, considerando o retorno de investimento; a inovação embutida no produto; dentre outros. Como despesa poderiam ser considerados os custos diretos do projeto; a qualidade de vida das pessoas envolvidas no projeto; o desvio do foco da empresa; dentre outros.

Apesar disso, a maioria dos estudos de análise da produtividade se baseiam na perspectiva quantitativa da produção. Uma vez que o nível de produtividade está associado ao custo de desenvolvimento e que o custo é fortemente influenciado pelo tamanho do *software*, é necessário analisar as métricas utilizadas para a medição do tamanho do *software*.

O tamanho do *software* é fator mais importante que afeta o custo do *software*. Cinco das métricas usadas na prática são:

- Linhas de Código: É o número de linhas do código fonte, excluindo os comentários e linhas em branco, sendo comumente chamado de LOC (do inglês, *Lines of Code*) [Fenton 1997].
- *Software Science*: Proposto por [Halstead 1977], de modo a medir o tamanho do código, através do número de ocorrências de operadores e operandos, e o volume, que corresponde à quantidade requerida de espaço em disco.
- Pontos de Função: Baseado na funcionalidade do programa, sendo o número total de pontos de função influenciados por cinco classes: quantidade de entradas, quantidade de saídas, quantidade de entradas interativas, quantidade de arquivos internos ao sistema e quantidade de arquivos externos ao sistema [Albretch 1983].
- Extensões de Ponto de Função: *Feature Points* extende os pontos de função de modo a incluir algoritmos como uma nova classe [Jones 1996]. *Full Function Point* (FFP) é uma outra extensão de modo a medir aplicações de tempo real [St-Pierre 1997]. Outros métodos de medição de pontos de função são IFPUG, Mark II, 3D, Asset-R, Experience e Cosmic [Maya 1998], [Rehesaar 1998], [COSMIC-FFP 2001], [Laturi 1996], [Function Point 1994].
- *Object Points*: Enquanto que *feature points* e FFP extendem os pontos de função, *object points* medem o tamanho através de outra dimensão. Tal métrica é baseada na quantidade e complexidade dos seguintes objetos: telas, relatórios e componentes 3GL. Apesar de não ser muito popular, *object points* são fáceis de serem usados nas fases iniciais do ciclo de desenvolvimento, sendo usados nos maiores modelos de estimativa de custo, como o COCOMO II [Boehm 1996].

Além da saída de um projeto de *software*, descrito acima através das diversas métricas de tamanho de *software*, é necessário definir o significado de esforço, correspondente à entrada do projeto. O mesmo será medido em horas ou meses? Será considerado o tempo alocado para atividades gerenciais, de suporte à equipe ou apenas o tempo de desenvolvimento propriamente dito? Serão consideradas as horas extras não pagas? Caso o cliente atue ativamente no projeto, isso contará como uma ajuda no esforço total? Quais fases do ciclo de desenvolvimento estão sendo consideradas?

Como se pode notar, o esforço é notavelmente difícil de ser medido com acurácia. [Shepperd et al. 2001] descreveram a experiência de acompanhar uma organização nas suas práticas de estimativa de esforço. Os dados referentes ao esforço de um mesmo projeto, de três diferentes fontes, possuíam discrepância de até 30%.

As medições da produtividade e qualidade do *software* em termos de linhas de código e no custo de detectar e remover defeitos são paradoxais, de modo que linhas de código por unidade de esforço tendem a enfatizar grandes programas, em vez de programas eficientes e de qualidade [Jones 1978]. De outro modo, o custo de detectar e remover defeitos indica que seria menos custoso reparar programas de baixa qualidade do que programas de alta qualidade. Sendo assim, como alternativa, Jones recomenda separar as medições de produtividade em unidades de trabalho e unidades de custo, enquanto que as medições de qualidade na eficiência para remover defeitos e prevenção dos mesmos.

Os fatores que regem os custos do *software* são efetivamente opostos aos fatores que regem a produtividade [Boehm 1981]. Por exemplo, a capacidade do time e a complexidade do produto possuem um grande efeito nos custos e produtividade no desenvolvimento de *software*. A alta capacidade da equipe e a baixa complexidade do produto levam à alta produtividade e baixo custo de produção. Em contrapartida, a baixa capacidade da equipe e a alta complexidade do produto levam à baixa produtividade e alto custo de produção. Diante da relação existente entre os fatores que regem o custo de desenvolvimento e os fatores que regem a produtividade, é importante destacar os modelos de estimativa de custo existentes.

Os modelos de estimativa de custo são divididos em não-algorítmicos e algorítmicos, sendo estes últimos subdivididos em empíricos e analíticos. Os modelos não-algorítmicos são baseados em deduções, como *Analogy Costing*, *Expert Judgment*, *Parkinson*, *Price-to-Win*, *Bottom-up* e *Top-down* [Shepperd 1997], [Parkinson 1957], enquanto que os modelos algorítmicos utilizam fórmulas matemáticas sofisticadas. Um modelo empírico usa dados de projetos passados para avaliar o projeto atual, derivando uma fórmula básica a partir dos dados disponíveis dos projetos passados. Exemplos de tais modelos são: *Nelson*, *Walston-Felix*, COCOMOs, *Aron*, *Boeing*, *Wolverton* e *Price-S* [Nelson 1966], [Tausworthe 1981]. Por outro lado, um modelo analítico usa fórmulas baseadas em suposições gerais. Como exemplo, tem-se os modelos *Putnam* e *SoftCost* [Park 1988], [Putnam 1978].

Apesar da existência de vários modelos de estimativa de custo, [Bailey et al. 1981], [Mohanty 1981] e [Kemerer 1987] concluíram que as estimativas de utilização de recursos devem ser específicas para cada organização e ambiente de desenvolvimento de modo a apresentar medições com maior acurácia. Dessa forma, os resultados de nenhum modelo de estimativa de custo é seguramente confiável.

Em parte, isso se deve a não consideração de fatores influenciadores no custo e conseqüentemente na produtividade do desenvolvimento de *software*. O modelo que mais se leva em consideração tais fatores é o COCOMO II [Boehm 1996], o qual aponta 15 fatores.

Organizações especializadas em *benchmarking* tem disponibilizado dados, gratuitamente ou não, sobre inúmeros projetos de forma a possibilitar a comparação de projetos com outros similares. Cada uma provê de uma série de variáveis de forma que projetos a serem desenvolvidos possam comparar suas características com a de outros projetos e assim prever a possível produtividade a ser conquistada. Em [Maxwell 2000],

os autores propõem um modelo de produtividade de acordo com o negócio do projeto, baseado nos dados disponibilizados no banco de dados *Experience*[1]. Outras fontes de dados para *benchmarking* de projetos de software são: *European Space Agency/INSEAD*[2], *International Software Benchmarking Standards Group* (*ISBSG*)[3], *Rubin Systems*[4] e *Software Productivity Research*[5].

Como se pode notar, existem vários conceitos relacionados com a produtividade de *software*, de forma que entendê-los tornará possível uma visão mais clara sobre os problemas enfrentados na definição de métricas de produtividade, bem como na definição dos fatores que a influencia. A próxima seção apresenta a compilação dos estudos existentes na literatura sobre os fatores que afetam a produtividade de *software*.

3. Fatores que afetam a Produtividade de *Software*

A produtividade é determinada pela interação de muitos fatores, de modo que nenhum fator em especial é capaz de garantir a alta produtividade em um projeto de *software*. Altos níveis de um determinado fator podem beneficiar o aumento da produtividade, mas um fator isoladamente não é suficiente para a obtenção de um alto nível de produtividade. Diante disso, é necessário que os fatores sejam balanceados e gerenciados de forma a obter a melhor produtividade possível de acordo com as características específicas de um projeto.

As seções abaixo descrevem os fatores que afetam a produtividade, identificando-os de acordo com cinco grupos: processo, tecnologia, pessoas, cultura organizacional e produto.

3.1. Fatores relacionados com o processo

Na comunidade do desenvolvimento de *software*, algumas pessoas pensam que a melhor maneira de se desenvolver um projeto é contratando as melhores pessoas possíveis, dando-lhes os recursos necessários e deixando-as livres para fazerem o que sabem de melhor. Seguindo tal ponto de vista, projetos sem processo definido podem ser desenvolvidos eficientemente, de modo que o uso de processo exigiria esforços adicionais, diminuindo assim a produtividade. Em [McConnell 1998], o autor descreve a importância do uso do processo, bem como as conseqüências de um projeto sem processo. As Figuras 1a e 1b ilustram os dois pontos de vista, de modo que o foco no processo leva a uma perda na produtividade, principalmente no início do projeto (ilustrado através da área mais escura), de acordo com a Figura 1b. A área no canto superior corresponde ao trabalho improdutivo, presente em ambos os casos.

Baseado na experiência da indústria de *software*, entretanto, a tendência ilustrada na Figura 1b não se prolonga por todo o projeto. Organizações que não se preocupam em estabelecer um efetivo processo no início de seus projetos são forçadas a estabelecê-

[1] http://www.datamax-france.com
[2] http://xrise.insead.fr/risenew/rise_esa.html
[3] http://www.isbsg.org.au
[4] http://www.hrubin.com
[5] http://www.spr.com

los mais tarde, tendo um maior custo e menos benefícios. Mudanças necessárias ao longo do projeto, bem como a integração entre os módulos tornar-se-iam problemas ao longo do projeto, de modo que os desenvolvedores gastariam seus tempos reprojetando e reescrevendo seus códigos, em vez de progredirem nas funcionalidades do sistema, o que fatalmente tornaria o projeto um fracasso.

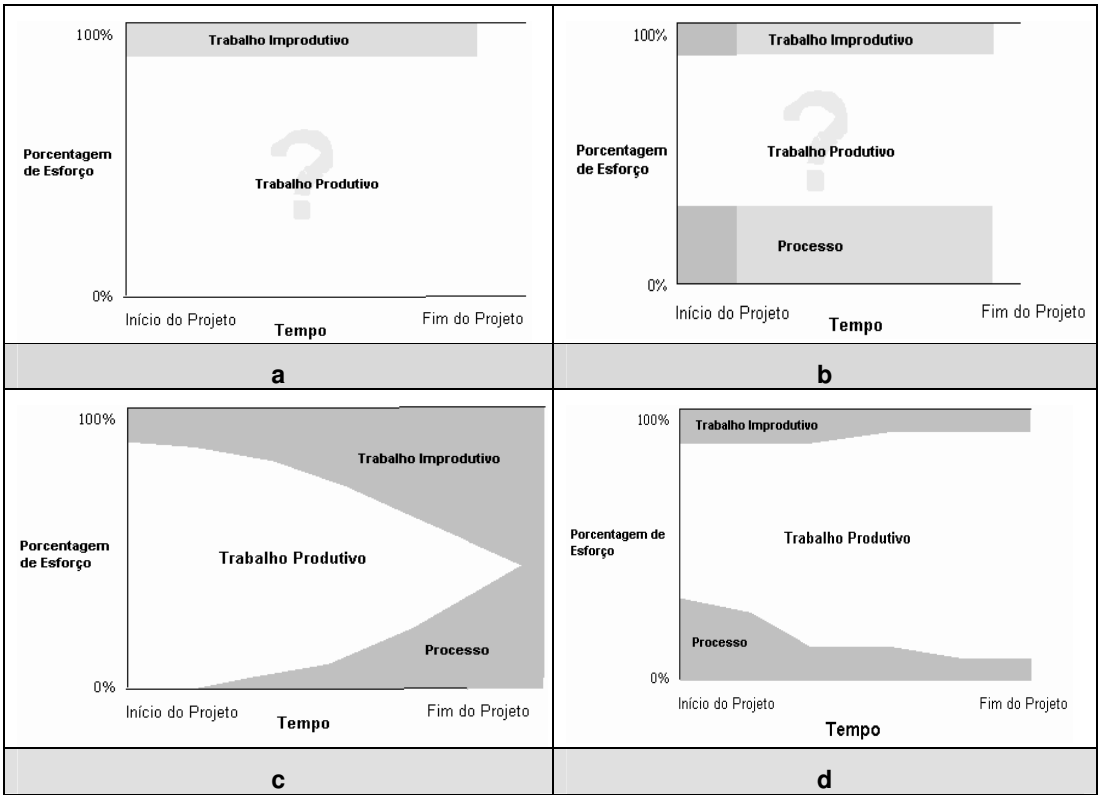

Figura 1. Produtividade adquirida com o uso de processos. Adaptado de [McConnell 1998]

De acordo com [McConnell 1998], longe do nível constante de produtividade ilustrado na Figura 1a, o padrão experimentado pelos projetos de médio e grande porte conduzidos sem um processo estabelecido se dá conforme ilustrado na Figura 1c, de modo que tentar estabelecer um processo no decorrer do projeto apenas aumentaria a improdutividade dos desenvolvedores. Diante disso, o investimento realizado no início do projeto, adotando um processo efetivo, terá como conseqüência um maior retorno de investimento ao longo do projeto, diminuindo o esforço com atividades improdutivas, como ilustrado na Figura 1d.

Organizações que têm focado explicitamente na melhoria dos seus processos de *software*, ao longo do tempo, têm diminuído o *time-to-market* pela metade, bem como reduzido os custos e defeitos em fatores de 3 a 10. De acordo com [Pietrasanta 1991],

em um período de cinco anos, a *Lockheed Martin Corporation*[6] cortou seus custos de desenvolvimento em 75%, reduziu o *time-to-market* em 40% e os defeitos em 90%. A *Raytheon Electronic Systems*[7], em um período de seis anos e meio, triplicou sua produtividade e obteve um retorno de investimento (ROI) na melhoria do processo de oito para um [Saiedian 1995]. Segundo [Herbsleb 1994], a *Bull HN Information Systems*[8] obteve um retorno de investimento de quatro para um depois de quatro anos de melhoria no processo de *software* e a *Schlumberger*[9] obteve um ROI de quase nove para um depois de três anos e meio de melhoria no processo. O laboratório de Engenharia de *Software* da NASA cortou seu custo por missão em 50% e sua taxa de defeitos em 75% ao longo de oito anos, mesmo aumentando dramaticamente a complexidade dos *softwares* usados em cada missão [Basili 1995]. Ao contrário do que se possa pensar, os custos relacionados com a melhoria do processo, de acordo com [Herbsleb 1994] é de 2% do total de custos com o desenvolvimento.

No entanto, apesar dos números, uma das objeções à implantação de um processo de *software* é o pensamento de que o mesmo irá limitar a criatividade dos programadores. De fato, é possível criar um ambiente opressivo, no qual a necessidade de criatividade do programador é menosprezada diante dos objetivos da gerência, como algumas organizações têm feito. Entretanto, é possível criar um ambiente onde tais objetivos são conquistados simultaneamente. Organizações que têm focado em processos têm concluído que processos efetivos prezam pela criatividade e moral dos desenvolvedores. Em um *survey* de aproximadamente 50 organizações, apenas 20% das pessoas taxaram o moral da equipe como "bom" ou "excelente" nas organizações com pouco foco no processo [Herbsleb 1997]. Naqueles com maior foco no processo, a proporção das pessoas que taxaram o moral como "bom" ou "excelente" subiu para 50%. Sendo assim, bons líderes de projeto põem o foco nos processos que permitam que os programadores se sintam produtivos, beneficiando assim tanto os desenvolvedores e seus projetos, como a organização como um todo. Segundo [Jones 1999], o uso efetivo de métodos e processo em projetos de *software* provê um aumento na produtividade de 35%, enquanto que o uso inadequado ou ineficiente provê uma diminuição na produtividade de 41%.

Um dos conceitos de qualidade, segundo [McConnell 1996], refere-se à baixa taxa de defeitos. Sendo assim, o controle de defeitos pode diminuir o cronograma dos projetos, aumentando assim a produtividade. Geralmente, gerentes de projeto tentam reduzir seus cronogramas reduzindo o tempo gasto com práticas de garantia da qualidade como revisão de código e *design*, reduzindo as atividades de análise de requisitos, bem como reduzindo o tempo alocado para testes. De acordo com [McConnell 1996], tais decisões são as piores a serem tomadas visando maximizar a produtividade de desenvolvimento. Segundo [Jones 1991], os projetos que atingem baixas taxas de defeitos terão os menores cronogramas, de modo que quanto maior for a taxa de remoção de defeitos antes do *release*, menores serão os cronogramas e esforços

[6] http://www.lockheedmartin.com/
[7] http://www.raytheon.com/
[8] http://www.bull.com/
[9] http://www.slb.com/

e maior será a satisfação do usuário final. Após um *survey* de aproximadamente 4.000 projetos de *software*, [Jones 1994] concluiu que a baixa qualidade era uma das razões principais para a extrapolação dos cronogramas. O autor também concluiu que a baixa qualidade é responsável pelo cancelamento da metade dos projetos. Segundo [Kitson et al. 1993], através de um *survey* realizado pela SEI[10], foi concluído que mais de 60% das organizações pesquisadas possuíam práticas inadequadas de garantia da qualidade.

Ainda segundo McConnell, projetos com cronogramas atrasados são vulneráveis à diminuição das práticas de garantia de qualidade a nível de desenvolvedor, de modo que os módulos são mal projetados, prejudicando a manutenibilidade, apenas pensando na rápida entrega do produto. Conseqüentemente, tais módulos estarão propensos a erros, aumentando o esforço na manutenção dos mesmos, bem como sua integração com o restante do sistema.

Um dos aspectos da garantia de qualidade particularmente importante no desenvolvimento rápido de projetos é a existência de módulos propensos a erros, ou seja, módulos responsáveis por um número desproporcional de erros. [Boehm 1987] reportou que 20% dos módulos de um programa são tipicamente responsáveis por 80% dos erros. Tais módulos são mais caros, levando um tempo maior para serem desenvolvidos, do que os módulos menos propensos a erros, uma vez que tendem a ser mais complexos, menos estruturados e geralmente muito grandes, sendo geralmente desenvolvidos em cronogramas sob pressão e pouco testados.

Com isso, se o desenvolvimento rápido é essencial em um projeto, a identificação e reprojeto dos módulos propensos a erros se tornam uma prioridade. Uma vez que a taxa de erros de um módulo é de aproximadamente dez erros para mil linhas de código, a revisão do código é importante para determinar se o módulo precisa ser reprojetado ou reimplementado [McConnell 1996]. Com isso, o cronograma diminuirá ao mesmo tempo em que haverá uma melhoria na qualidade do produto.

Estudos têm concluído que o retrabalho proveniente de requisitos, projeto e código defeituosos tipicamente consome 40 a 50% do custo total de desenvolvimento de *software* [Jones 1986], de modo que cada hora gasta com a prevenção de defeitos reduzirá o tempo de reparo em 3 a 10 horas [McConnell 1996]. No pior caso, o retrabalho proveniente de problemas quanto aos requisitos quando o *software* está em operação custa de 50 a 200 vezes mais caso tais problemas fossem resolvidos na fase de análise dos requisitos [Boehm 1988]. Diante disso, práticas de garantia de qualidade como revisões técnicas, que incluem *walkthrough*, leitura de código e inspeções são componentes críticos de todo esforço de desenvolvimento que pretende obter o menor cronograma possível. Uma vez que as inspeções podem ser feitas desde cedo durante o ciclo de desenvolvimento, [Gilb 1993] constatou que o uso de tal prática reduzirá o cronograma de 10 a 30%. De acordo com [Russell 1991], cada hora gasta com inspeção evita aproximadamente 33 horas de manutenção, sendo as inspeções 20 vezes mais eficientes que os testes. De acordo com [Freedman et al. 1990], em uma organização de *software*, 55% de mudanças em uma linha de código relacionadas com manutenção se

[10] http://www.sei.cmu.edu/

deram devido a erros antes da introdução de revisões de código. Depois da introdução de revisões, apenas 2% se referiam a erros. Segundo [Jones 1999], o uso formal de inspeções provê uma melhoria na produtividade de até 15%, enquanto que o não uso de tal prática pode diminuir a produtividade em até 48%.

Pequenos projetos tendem a focar nas atividades relacionadas com a construção. Isso não significa que as demais atividades, como arquitetura, design e planejamento devem ser ignoradas. Tais atividades desempenham um papel importante para o sucesso do projeto, de modo que negligenciá-los pode levar ao aumento de erros ao longo do desenvolvimento e conseqüente insucesso no projeto. Segundo [Jones 1999], a atividade de medição da qualidade pode proporcionar um aumento na produtividade em 14%, visto que a qualidade do produto está ligada à auto-estima do desenvolvedor [DeMarco 1999]. A não prática de medição da qualidade pode diminuir a produtividade em 10% [Jones 1999].

Alguns estudos procuram prover mecanismos para dinamizar o processo e suas atividades de modo a conquistar um alto nível de produtividade. Em [DeGrace et al. 1990], os autores catalogam abordagens para o desenvolvimento de *software*, enfatizando que qualquer abordagem de desenvolvimento deve variar de acordo com o tamanho do projeto. Os autores discutem a customização de processos de desenvolvimento de *software* baseado no tamanho do projeto e formalismo necessário, descrevendo alguns modelos adotados na NASA e Departamento de Defesa dos Estados Unidos. Em [Boehm 1981], o autor trata das ramificações de custo, produtividade e qualidade a partir do tamanho dos projetos e variações dos processos de desenvolvimento de *software*, discutindo o efeito do tamanho nas diversas fases do ciclo de desenvolvimento de *software*.

De modo a prover dados sobre o impacto de estimativas de esforço, em 1985, dois pesquisadores da *University of New South Wales*[11], Michael Lawrence e Ross Jeffery fizeram um *survey* em 103 projetos, formando uma métrica pra avaliar a produtividade, cujo resultado está ilustrado na Tabela 1.

Tabela 1. Responsável pela estimativa de esforço e produtividade associada. Adaptado de [DeMarco 1999]

Estimativa de esforço preparada por	Produtividade Média	Número de Projetos
Programador	8,0	19
Supervisor	6,6	23
Programador e Supervisor	7,8	16
Analista de Sistemas	9,5	21
(Nenhuma Estimativa)	12,0	24

[11] http://www.unsw.edu.au/

De acordo com o resultado, programadores parecem ser mais produtivos quando eles próprios fazem as estimativas, em relação a quando os supervisores fazem as estimativas. Em contra-partida, são mais produtivos quando os analistas fazem as estimativas. Tal fato se deve à tendência de os analistas serem melhor estimadores em relação aos programadores e supervisores. Já nos 24 projetos nos quais nenhuma estimativa foi realizada, a produtividade foi maior. Isso se deve ao fato de que pessoas não trabalham eficientemente quando o cronograma é impossível de se cumprir, que ocorre quando estimativas são feitas inadequadamente.

Segundo [Jones 1999], o uso formal de estimativas de custo e cronograma pode aumentar a produtividade em 17%, enquanto que o uso informal pode diminuir a produtividade em até 22%.

De modo a propiciar uma boa visão do produto a ser desenvolvido, e assim ter conhecimento o quanto antes das funções necessárias, bem como a complexidade de tais funções, faz-se necessária a participação ativa do cliente em todas as fases do projeto. Dessa forma, o gerente de projeto pode influenciar positivamente no cronograma.

O constante *feedback* por parte do cliente pode evitar uma grande quantidade de mudanças futuras no produto, de modo que quanto mais próximos estiverem os desenvolvedores e os clientes durante todas as fases de desenvolvimento, maiores serão as chances de o produto atender as expectativas dos clientes, economizando assim, a alocação de tempo com mudanças que poderiam ser evitadas. Segundo [Jones 1999], a participação efetiva do cliente pode provê um aumento na produtividade de 18%, enquanto que nenhum tipo de participação pode diminuir a produtividade em 13%.

Entre todos os fatores, a prática do reuso é o fator que mais contribui para a alta produtividade. Métricas coletadas em dois programas da *Hewlett-Packard*[12] demonstraram melhoria na qualidade, aumento da produtividade e redução do *time-to-market* devido ao reuso. Os resultados do custo-benefício indicaram que o reuso provê um substancial retorno de investimento [Lim 1994]. De acordo com [Boehm 1987], enquanto que o trabalho mais rápido, através de ferramentas, e mais inteligente, através de planejamento, possui um impacto de 8% e 17%, respectivamente, evitar o trabalho, através da prática de reuso apresenta um impacto de 47% na produtividade. Em [Jones, 1999], o autor afirma que a prática eficiente do reuso pode elevar a produtividade em até 350%.

Em resumo, os fatores relacionados com o processo que afetam a produtividade em projetos de *software* são: uso efetivo de processo de software; uso de práticas de inspeção; medição da qualidade; estimativa de custo e esforço; participação do cliente e reuso.

3.2. Fatores relacionados com a tecnologia

A tecnologia associada ao projeto também possui uma grande influência na produtividade geral do projeto. [Behrens 1983] utilizou a medição através de pontos de

[12] http://www.hp.com

função para comparar a produtividade entre diferentes projetos e concluiu que, além tamanho do projeto, o ambiente de desenvolvimento e a linguagem de programação impactam na produtividade.

[Thadhani 1984] e [Lambert 1984] examinaram os efeitos do uso de bons computadores na produtividade do programador e do projeto, com ênfase na importância do curto tempo de resposta. De acordo com [Jones 1999], o curto tempo de resposta pode ocasionar um aumento na produtividade de 12%, enquanto que um alto tempo de resposta pode diminuir a produtividade em 30%.

O uso de ferramentas pode melhorar a produtividade em projetos de *software*, facilitando a execução de atividades não praticadas até então. No entanto, a introdução de ferramentas pode também diminuir a produtividade, que pode ocorrer quando o uso da ferramenta aumenta o esforço necessário para desempenhar determinadas atividades ou até mesmo introduz novas atividades ao processo utilizado. Com isso, dependendo das características do projeto, a mesma ferramenta pode ocasionar diferentes efeitos na produtividade.

De acordo [Bruckhaus et al. 1996], de modo a aumentar a produtividade, as ferramentas devem ser selecionadas tendo em mente o tamanho do projeto e o processo de desenvolvimento adotado. Tais características são importantes uma vez que influenciam sobre que funções das ferramentas serão utilizadas e com que freqüência serão utilizadas.

De modo geral, a introdução de ferramentas parece ser justificada quando se planeja adotar um processo sofisticado, incorporando também um esforço maior de acordo com a necessidade do processo. A quantidade adicional de esforço pode ser determinada por métodos de análise quantitativa, como o *Software Productivity Analysis Method* (SPAM) [Bruckhaus 1997], [Yin 1994].

Segundo [Jones 1999], o uso efetivo de ferramentas CASE (*Computer Aided Software Engineering*) e de gerenciamento pode ocasionar a melhoria na produtividade em 30%. Em contra-partida, o uso inadequado de ferramentas de gerenciamento e CASE pode diminuir a produtividade em 45% e 75%, respectivamente. O ganho na produtividade proveniente do uso de ferramentas de estimativa de qualidade é menor, 19%. Entretanto, o não uso de tais ferramentas pode afetar negativamente a produtividade em até 40%.

As linguagens de programação utilizadas nos projetos também contribuem para a melhoria da produtividade. Os tipos de linguagens de programação podem ser classificados em 2GL, 3GL e 4GL. As linguagens 2GL (*Second-Generation Language*) são linguagens simbólicas, consistindo de acrônimos semelhantes à linguagem natural. As linguagens 3GL, como FORTRAN, COBOL e BASIC, são mais sofisticadas, fáceis de usar e direcionadas a tipos especializados de problemas, sendo conhecidas como linguagens de alto nível. As linguagens 4GL possuem larga utilização no mercado, tornando possível o desenvolvimento de tarefas por indivíduos não-técnicos.

Apesar de que [DeMarco, 1999] afirma que não há correlação entre a linguagem de programação e a produtividade, em [Jones, 1999], o autor afirma que o uso de linguagens de alto nível pode prover uma melhoria na produtividade de até 24%,

enquanto que o uso de linguagens de baixo nível pode diminuir a produtividade em 25%. Em [Jiang 2007], através de um *survey* realizado nos projetos da base de dados do ISBSG, o autor conclui que o tipo da linguagem interfere largamente na produtividade final de um projeto.

De modo geral, os fatores relacionados com a tecnologia que afetam a produtividade em projetos de *software* são: parque computacional utilizado; ferramentas CASE, de gerenciamento e de estimativa de qualidade; e linguagem de programação adotada.

3.3. Fatores relacionados com as pessoas

A comunicação flui mais facilmente em pequenos times do que em grandes times. Caso um projeto disponha de apenas uma pessoa, a comunicação será simples, de modo que o único caminho de comunicação será entre tal pessoa e o cliente. À medida que o número de pessoas em um projeto aumenta, o número de caminhos de comunicação também aumentará, de modo exponencial. A Figura 2 ilustra como um projeto composto por duas pessoas apresenta apenas um caminho de comunicação. Um projeto composto por cinco pessoas apresenta 10 caminhos. Um projeto composto por 10 pessoas apresenta 45 caminhos, assumindo que toda pessoa conversa com todas as outras pessoas. Os projetos que possuem 50 ou mais desenvolvedores possuem ao menos 1.200 caminhos potenciais de comunicação. Desse modo, quanto mais caminhos de comunicação um projeto apresentar, maior será o tempo gasto com comunicação, ao mesmo tempo em que maiores oportunidades de erros de comunicação serão proporcionadas. Portanto, grandes projetos demandam de técnicas organizacionais que procuram limitar o quanto possível a comunicação entre as pessoas.

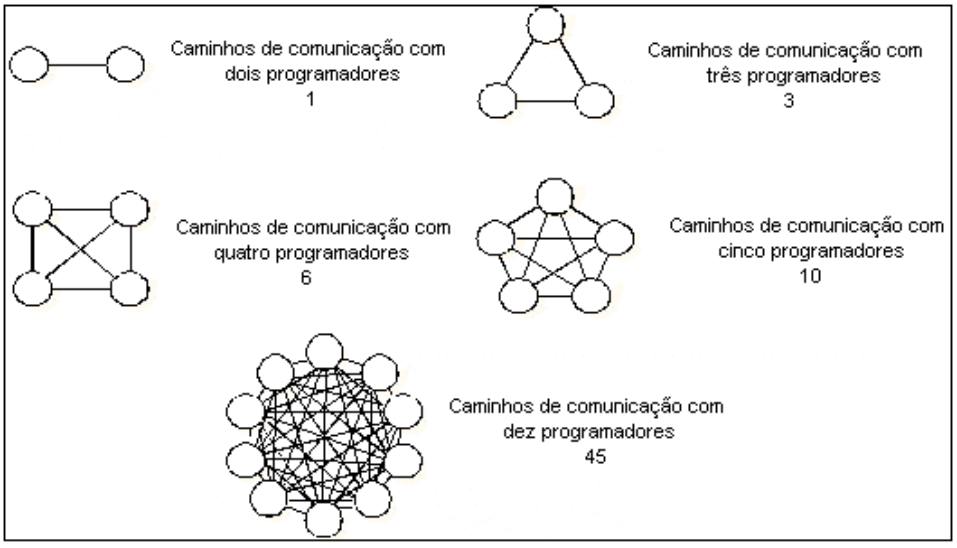

Figura 2. Aumento dos caminhos de comunicação à medida que pessoas são adicionadas ao projeto. Adaptado de [McConnell 1997]

A abordagem típica adotada para limitar a necessidade de comunicação em grandes projetos é formalizar a comunicação através de documentos escritos, de modo

que em vez de 50 pessoas conversarem entre si através de cada combinação possível, as mesmas lerão e escreverão documentos. Em pequenos projetos, no entanto, tais documentos podem ser evitados.

Em [Boehm et al. 1984], os autores reportaram que pequenos times completaram seus projetos com 39% maior de produtividade em relação a grandes times. [Behrens 1983] concluiu que pequenos times produzem mais pontos de função do que grandes times em prazos proporcionais.

De forma a aumentar a produtividade em projetos atrasados em seus cronogramas, muitos gerentes adicionam pessoas aos mesmos. No entanto, de acordo com [Brooks 1995], adicionar pessoas a um projeto com cronograma atrasado, apenas o tornará ainda mais atrasado, uma vez que aumentará o esforço total de três maneiras: o reparticionamento do trabalho como um todo, o treinamento das novas pessoas e o aumento da necessidade de comunicação. Na Figura 3, [Simmons 2001] confirma que a adição de pessoas ao projeto não aumenta a produtividade global na mesma proporção.

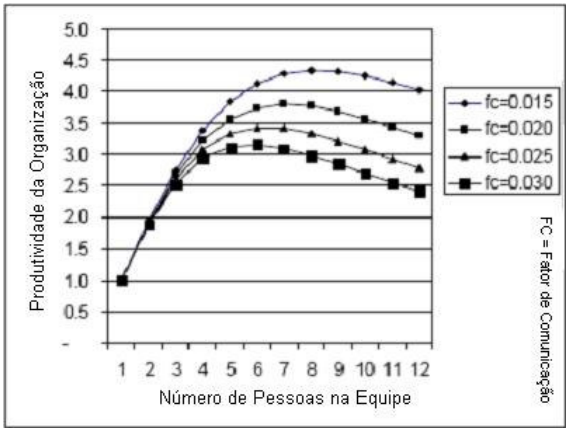

Figura 3. Produtividade da organização quanto ao número de pessoas na equipe. Adaptado de [Simmons 2001]

Segundo [Jones, 1999], a definição de ocupações específicas para cada funcionário, de modo que o mesmo possa se especializar em determinada área pode prover um aumento na produtividade em 18%, enquanto que a não definição de ocupações pode diminuir a produtividade em 15% . Em [DeMarco, 1999], o autor enfatiza a necessidade da existência de gurus nas empresas, as quais contribuem desproporcionalmente para a organização, sendo assim motivados a compensarem a posição que a organização criou para os mesmos.

Segundo [DeMarco 1999], anos de experiência por parte dos desenvolvedores não possui grande influência na produtividade, visto que em experiências realizadas, programadores com até seis meses de experiência na linguagem de programação utilizada saíram-se pior que programadores mais experientes. No entanto, programadores com dez anos de experiência não se saíram melhor do que programadores com apenas dois anos de experiência. Apesar disso, [Jones 1999] indica que a grande experiência da equipe pode proporcionar um aumento na produtividade de até 55%, enquanto que a falta de experiência pode prejudicar a produtividade em 87%.

Da mesma forma, segundo Jones, o gerente precisa ter uma boa base de conhecimento e experiência no cargo, o que contribuirá para o aumento na produtividade de até 65%. A inexperiência do gerente pode levar a uma queda na produtividade de até 90%.

Além dos fatores acima, o COCOMO II [Boehm 1996] leva em consideração a continuidade do pessoal, ou seja, o *turnover*. De acordo com [DeMarco 1999], as organizações não têm procurado ter um controle de tal fator, de modo que o alto *turnover* tem ocasionado uma diminuição significativa em projetos de *software*.

Outros fatores relacionados com pessoas foram identificados. Entretanto, os mesmos foram agrupados nos fatores relacionados com a cultura organizacional, uma vez que é a organização a responsável por garantir tais fatores.

Em resumo, os fatores relacionados com pessoas que afetam a produtividade em projetos de *software* são: tamanho da equipe; experiência da equipe de desenvolvimento; existência de especialistas; experiência da gerência e *turnover*.

3.4. Fatores relacionados com a cultura organizacional

As causas de perdas de dias e horas de trabalho são influenciadas pelo ambiente provido pela organização. Para decidir o tipo de ambiente para cada pessoa, assim como a quantidade de espaço a ser alocada, deve-se estudar as maneiras que as pessoas usam seus espaços, bem como a quantidade de horas gastas trabalhando sozinhas ou em grupo. Deve-se também investigar o grau de impacto do barulho na eficiência do trabalho. Desse modo, faz-se necessário obter um *trade-off* entre custo e privacidade.

No entanto, geralmente não existe preocupação nesse sentido por parte das organizações. Através de um *survey* em 12 diferentes organizações, [DeMarco 1999] percebeu que em todos os casos os ambientes de trabalho eram barulhentos. Alguns eram mais bonitos que outros, mas não funcionais. Em uma empresa da Califórnia, nos Estados Unidos, foi feito um *survey* entre os desenvolvedores (em torno de 1000) sobre os problemas da organização. De acordo com os mesmos, o segundo maior problema era a pobre comunicação com os gerentes superiores e o maior problema, o ambiente barulhento.

Em 1984, Tom DeMarco e Timothy Lister realizaram um *survey* no qual times de desenvolvedores de diferentes organizações participavam de uma competição para codificar e testar no mínimo tempo e com o mínimo de defeitos, chamado de *Coding War Games*, usando seus próprios ambientes de trabalho. Ao final da competição, concluiu-se que o espaço dos *top-performers* era mais silencioso, mais privado e melhor protegido da interrupção. Tal fato não quer dizer que ambientes melhores irão melhorar largamente a produtividade dos funcionários, mas que os funcionários que têm maior produtividade tendem a trabalhar em ambientes melhores. Com isso, o que pode ser provado com o *survey* é que o uso de um ambiente uniforme é um erro.

A razão óbvia para o fato de as organizações não prezarem pela privacidade, tendo assim menos espaço dedicado e mais barulho é o custo, esquecendo assim do benefício. Dessa forma, a redução do custo deve ser comparada com o risco da perda da eficiência. Em um exercício com pessoas que consideravam os ambientes de trabalho

silenciosos e outros que consideravam barulhento foi constatado que os funcionários que reportaram antes do exercício que o ambiente de trabalho era silencioso foram 1/3 mais propícios a entregarem trabalhos não defeituosos [DeMarco 1999].

Baseado em um estudo dos hábitos de gerentes, desenvolvedores e engenheiros de qualidade de uma filial da IBM, o arquiteto Gerald McCue concluiu que um espaço de trabalho deve ter 9 m^2 por pessoa. No entanto, baseado num estudo de 32.346 empresas, foi concluído que a densidade de pessoas apresenta uma relação inversamente proporcional ao espaço dedicado por pessoa [DeMarco 1999].

Um estudo realizado por McCue mostra como os desenvolvedores gastam seus tempos, como mostra a Tabela 2. Em 30% do tempo, as pessoas são sensíveis ao barulho e no resto do tempo, sensíveis ao mesmo. Durante o tempo, no qual a pessoa trabalha sozinha, a mesma está no estado que psicologicamente se chama de *flow*, ou seja, uma condição de envolvimento profundo no trabalho.

Tabela 2. Porcentagem de tempo gasto por modo de trabalho. Adaptado de [DeMarco 1999]

Modo de Trabalho	Porcentagem do Tempo
Trabalhando sozinho	30%
Trabalhando com outra única pessoa	50%
Trabalhando com duas ou mais pessoas	20%

Quando se tem uma pessoa com tendência a trabalhar com uma pessoa em particular, é aconselhável dividirem um mesmo ambiente de trabalho, uma vez que eles estarão em sintonia quanto ao estado de *flow*.

Em um ambiente barulhento, uma das principais causas de interrupção é o telefone. Desse modo, um importante passo para minimizar o efeito negativo do telefone é conscientizar do tempo de uso do telefone durante o tempo de trabalho. Uma alternativa ao uso do telefone seria o uso do e-mail, provendo assim resposta de acordo com a conveniência.

De acordo com [Jones 1999], a forma como o ambiente de trabalho está organizado pode ocasionar um aumento na produtividade em 15%, quando estruturado adequadamente e uma diminuição em 27%, quando mal estruturado. Já a separação física da equipe envolvida no projeto pode ocasionar uma diminuição na produtividade de até 24%. Quando não há separação, a produtividade pode ter um aumento de 8%, devido à facilidade de comunicação entre os integrantes do projeto.

Medir a produtividade é importante para que se busque sua constante melhoria. Geralmente, uma vez que o funcionário não tem idéia de sua produtividade, não importa a notícia sobre sua produtividade, seja ela boa ou ruim, ele sempre ficará surpreso. Apesar de não haver um consenso quanto à medição da produtividade, [Gilb 1977] afirma que qualquer coisa que se queira quantificar pode ser medida de tal forma que seja melhor do que não medir nada. A medição da produtividade pode ser uma ferramenta útil para melhoria do processo, motivação e conseqüente satisfação do trabalho, mas quase nunca é usado para tais propósitos. Segundo [Jones 1999], a

medição da produtividade pode proporcionar um ganho de 10% na produtividade, enquanto que nenhum tipo de medição pode proporcionar uma diminuição de 7%.

De modo a melhorar a produtividade, muitas organizações utilizam a tática de horas extras não pagas. De acordo com [DeMarco 1999], horas extras são como uma ficção na imaginação de gerentes ingênuos, de modo que ninguém pode trabalhar mais do que 40 horas por semana, ao menos que não seja continuamente e com nível de intensidade para o trabalho requerido. Com isso, os esforços serão mais ou menos de 1 hora de "relaxamento no trabalho" para cada hora extra. De acordo com o autor, horas extras são como *sprinting*, ou seja, fazem sentido no final da maratona e não no início, de modo que tentar fazer com que os funcionários estejam em constante *sprint* fará com que o gerente perca o respeito. *Workaholics* trabalham muitas horas, mesmo com pouca eficiência, colocando-se sob pressão e estragando suas vidas pessoais. Desse modo, deve-se controlar tais pessoas.

Segundo [DeMarco 1999], muitos gerentes têm usado da Teoria de Valor Espanhola, que prega o uso da pressão em projetos de *software*, com *deadlines* impossíveis de serem cumpridos. De acordo com o autor, tais gerentes procuram atingir altos níveis de produtividade com o mecanismo de horas extras não pagas, dividindo qualquer trabalho feito por 40 horas, e não por 80, 90 horas que o funcionário tenha realmente trabalhado. De acordo com [Jones 1999], o uso de horas extras não pagas pode oferecer um ganho na produtividade de até 15%. Ainda segundo [Jones 1999], o não uso de horas extras não pagas não acarreta em efeito negativo à produtividade total do projeto. Apesar de proporcionar uma certa melhoria na produtividade, tal prática não é muito recomendada, uma vez que prejudicará a qualidade de vida e satisfação no trabalho dos funcionários [DeMarco 1999].

Gerentes de projeto têm comprometido a qualidade do projeto estipulando *deadlines* impossíveis, embora saibam que existe uma tendência por parte dos desenvolvedores de se vincular a auto-estima à qualidade do trabalho, e não à quantidade. No Japão, a qualidade está intrinsecamente relacionada com a produtividade. Segundo [Crosby 1980], deixar que o desenvolvedor estipule um padrão de qualidade próprio resultará num ganho suficiente da produtividade para eliminar o custo da melhoria da qualidade. A Hewlett-Packard é um exemplo de uma organização que colhe os benefícios de aumentar a produtividade devido a padrões de qualidade estipulados pelos desenvolvedores. Existe uma cultura de qualidade, na qual o argumento de que mais tempo e dinheiro é necessário para se ter mais qualidade não é aceito.

Na maioria dos casos, a decisão de pressionar as pessoas para entregar o projeto que não satisfaça com sua qualidade é quase sempre um erro [DeMarco 1999]. Apesar disso, [Jones 1999] afirma que a existência de uma pressão moderada por parte da gerência pode ocasionar um aumento na produtividade em torno de 11%. Caso tal pressão seja excessiva, a produtividade pode ter uma diminuição de até 30%.

[DeMarco 1999] afirma que as organizações devem incentivar o surgimento de grupos de pessoas, nos quais a força do grupo seja mais forte do que a soma das forças dos integrantes do grupo. O autor define tais grupos como *jelled teams*, os quais

possuem como características: forte senso de identidade de grupo; menor interesse pelo dinheiro, status ou elevação de posição; propriedade conjunta do produto que está sendo desenvolvido e acima de tudo, trabalho com diversão.

De forma a tornar possível o surgimento de tais grupos, faz-se necessárias algumas posturas por parte da gerência, como evitar o gerenciamento defensivo, no qual o gerente tem o pensamento de que nos pontos em que os mesmo não interferir, a equipe de projeto estará mais propensa a cometer erros; evitar a burocracia, como o *paperwork*, ou seja, trabalho com documentos, que segundo Jones, corresponde à cerca de 30% do custo de produção de um produto, sendo um fator desmotivante para a formação de *jelled teams*; evitar a separação física do grupo, uma vez que quando as pessoas que trabalham em uma mesma equipe, estarão sempre em sintonia quanto ao estado de *flow*; evitar a produção de produtos com baixa qualidade, visto que a auto-estima dos desenvolvedores está relacionada com a qualidade do produto e não com a quantidade; e não estimar *deadlines* impossíveis de serem cumpridos. Com isso, a moral do grupo estará elevada, conseqüentemente melhorando a produtividade do desenvolvimento. Segundo [Jones 1999], a alta motivação e moral das pessoas pode aumentar a produtividade em 7%, enquanto que a baixa motivação e moral pode reduzir a produtividade em 6%. De modo a aumentar a motivação dos desenvolvedores, [Clincy 2003] sugere o uso de recompensas, financeiramente ou não, embora tal prática não seja recomendada por [DeMarco, 1999], uma vez que inibirá a formação de *jelled teams*.

Apesar da necessidade da formação de equipes unidas, [Jones, 1999] indica da necessidade de se ter uma certa hierarquia na organização, o que proverá um aumento de até 5% na produtividade.

As organizações têm procurado melhorar seus métodos, desenvolvendo seus negócios de maneira mais ordenada [DeMarco 1999]. Segundo o autor, faz-se necessária a introdução de um certo "caos", visto que, segundo o *Hawthorn Effect*, as pessoas executam melhor quando estão fazendo algo novo, havendo assim uma melhoria na produtividade. Com isso, DeMarco sugere a introdução de projetos piloto, nos quais são setados um conjunto de padrões não comprovados tecnicamente para atingir um determinado objetivo. Além de projetos piloto, o autor sugere treinamentos anuais aos funcionários. Em [Jones 1999], treinamentos anuais de duração maior do que dez dias podem influenciar positivamente na produtividade, obtendo um aumento de 8%. Já caso não ocorra nenhum tipo de treinamento, a produtividade pode diminuir em até 12%.

Em resumo, os fatores relacionados com a cultura organizacional que afetam a produtividade em projetos de *software* são: estrutura do ambiente de trabalho; separação das equipes; moral das pessoas envolvidas; estrutura hierárquica das pessoas; uso de horas extras não pagas; pressão no cronograma; medição da produtividade; treinamentos anuais; e uso de recompensas.

3.5. Fatores relacionados com o produto

De acordo com [Vosburgh 1984], os fatores relacionados com o produto não podem ser controlados pelo gerente, não sendo possível obter uma melhoria na produtividade através do balanceamento de tais fatores. Apesar disso, por afetar a produtividade de qualquer projeto, tais fatores são tratados neste trabalho.

À medida que o tamanho do produto e, conseqüentemente, do projeto aumenta, as atividades desempenhadas durante o ciclo de desenvolvimento de um projeto de *software* mudam drasticamente, devido ao aumento da necessidade de comunicação formal. A Figura 4 apresenta a proporção de esforço por atividade em projetos de diferentes tamanhos, de modo que à medida que o tamanho do projeto aumenta, as atividades relacionadas com a construção (mostradas na área cinza do gráfico) consomem uma porcentagem menor de esforço em relação ao esforço total de desenvolvimento.

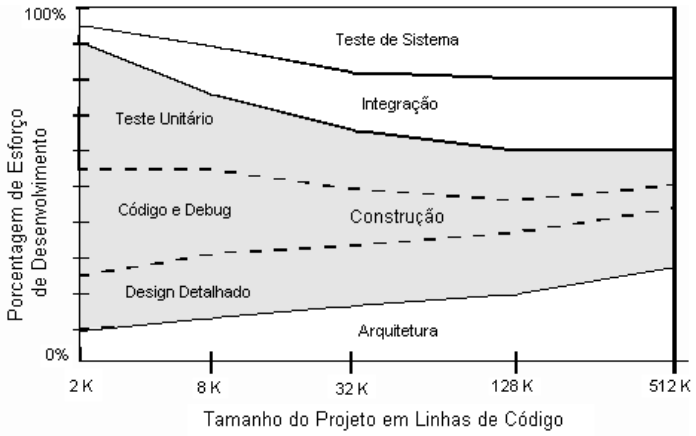

Figura 4. Com o aumento do projeto, a fase de construção consome uma pequena porcentagem do esforço total de desenvolvimento. Adaptado de [McConnell 1997]

De acordo com [McConnell 1997], em grandes projetos, as atividades relacionadas com arquitetura, construção, integração e teste de sistema possuem a mesma quantidade de esforço. Em projetos de porte médio, a construção começa a se tornar uma atividade predominante. Em pequenos projetos, a construção é a atividade predominante, consumindo cerca de 80% de todo o esforço. De modo geral, à medida que o tamanho do projeto diminui, as atividades relacionadas com a construção se tornam a parte predominante do esforço total de desenvolvimento.

A Figura 5 mostra a relação da quantidade de erros por fase do ciclo de desenvolvimento conforme o tamanho do projeto, em linhas de código. De acordo com [Jones 1977], em pequenos projetos, os erros ocorridos na fase de construção correspondem a aproximadamente 75% do total de erros encontrados no projeto. Uma explicação para tal fato é a baixa predominância de uma metodologia e a falta de um programa de qualidade, sendo o mesmo baseado apenas nas habilidades individuais de quem escreve os programas, dependendo assim do quão inteligíveis e manuteníveis são os códigos implementados pelos desenvolvedores.

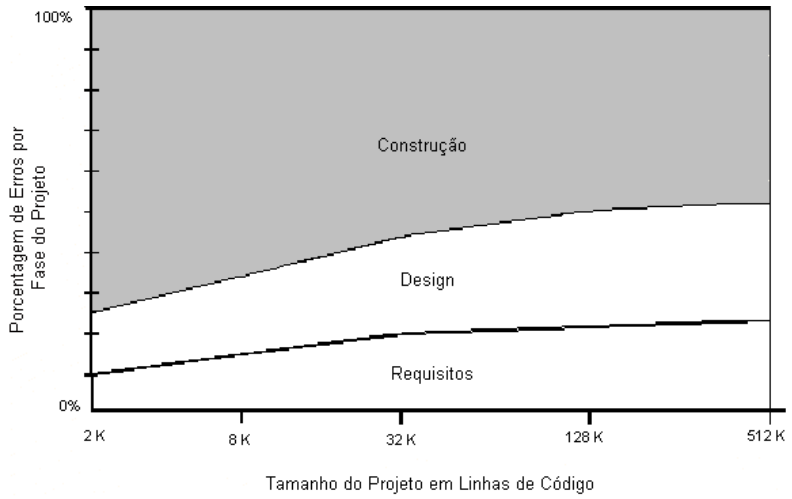

Figura 5. Porcentagem de erros por fase de acordo com o tamanho do projeto. Adaptado de [McConnell 1997]

Já em grandes projetos, os erros relacionados com a fase de construção correspondem à cerca de 50% do total de erros, fazendo com que os erros relacionados com requisitos e arquitetura façam a diferença. Isso se deve ao fato de que proporcionalmente mais tempo é gasto na fase de análise de requisitos e arquitetura em grandes projetos, fazendo com que a oportunidade para o surgimento de erros em tais atividades seja proporcionalmente maior.

A densidade de defeitos (número de defeitos por linha de código ou pontos de função) também muda com o tamanho do projeto. De acordo com a Tabela 3, o número de erros aumenta drasticamente à medida que o tamanho do projeto aumenta, de modo que grandes projetos tenham muito mais erros por ponto de função do que pequenos projetos.

Tabela 3. Aumento da densidade de erros de acordo com o tamanho do projeto. Adaptado de [McConnell 1997]

Tamanho do Projeto (Em pontos de função)	Densidade de Defeitos (Defeitos por ponto de função)
10	0,07
100	0,40
1.000	0,60
10.000	0,80
100.000	1,30

A produtividade e a qualidade de *software* têm muito em comum em relação ao tamanho do projeto, uma vez que ambos são melhores em pequenos projetos. A Tabela 4 mostra a relação geral entre o tamanho do projeto e a produtividade.

Tabela 4. Diminuição da produtividade com o aumento do tamanho do projeto. Adaptado de [McConnell 1997]

Tamanho do Projeto (Em pontos de função)	Pontos de Função por Mês
10	13,0
100	10,0
1.000	4,0
10.000	2,6
100.000	1,7

A documentação necessária em um projeto depende do tamanho do produto a ser desenvolvido, de modo que quanto menor for o produto, menor será a quantidade de documentação necessária e assim mais esforço será alocado ao desenvolvimento do produto. De modo geral, em pequenos times, a documentação serve como auxílio para a memória falha, característica dos desenvolvedores. De acordo com [McConnell 1997], a memória das pessoas não são tão eficientes em pequenos projetos como o são em grandes projetos sendo, portanto, necessária a documentação das informações de projetos que duram pouco mais de algumas semanas. O tamanho de pequenos projetos significa que a documentação pode ser menos formal. O design de pequenos projetos consiste de um conjunto de diagramas capturados através de *flip charts* em vez de um documento formal da arquitetura. Portanto, pequenos projetos podem obter algumas economias devido ao pequeno tamanho do time, mas sempre apresentarão alguma documentação de modo a servir como suplemento à memória das pessoas.

Como se pode notar, o tamanho do produto possui uma grande influência na produtividade em projetos de *software*. No entanto, outros fatores a nível de produto também são influenciadores. O COCOMO II [Boehm 1996] considera três fatores, além do tamanho: confiabilidade requerida, tamanho do banco de dados e complexidade do produto. De acordo com [McConnell 1996], existem dois conceitos de qualidade que podem influenciar os cronogramas do projeto de diferentes maneiras. Além do conceito apresentado na Seção 3.1, o outro conceito de qualidade refere-se às características de um produto, como funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade, descritos em [ISO/IEC 9126 2001], as quais, dependendo do grau de necessidade no produto, podem alongar o cronograma do projeto. A medição através de pontos de função leva em consideração alguns fatores, chamados de Fatores de Ajuste de Valor (FAV), os quais podem se encaixar como sub-características do modelo de qualidade ISO 9126. Sendo assim, além da confiabilidade requerida apresentada no COCOMO II, outras características devem ser levadas em consideração.

Além das características intrínsecas do produto, a complexidade do produto contribui satisfatoriamente para a produtividade com que o mesmo é desenvolvido. Segundo [Jones 1999], a complexidade pode prover um aumento na produtividade de 13%, nos casos de produtos pouco complexos, e uma diminuição de 35% na produtividade, nos casos de produtos muito complexos.

No *survey* realizado em 1994 pela *Standish Group* de modo a averiguar sobre os fatores que fazem com que os projetos extrapolem o orçamento e cronograma, foi constatado que dois dos três primeiros fatores correspondiam a requisitos e especificações incompletas e mudanças nos requisitos ao longo do projeto, justificadas tanto por necessidades do mercado, quanto por necessidades do próprio cliente. Segundo [Jones 1999], requisitos pouco voláteis podem aumentar a produtividade em torno de 9%, enquanto que a alta volatilidade dos requisitos pode levar a uma baixa na produtividade de até 77%.

De modo geral, os fatores relacionados com o produto que afetam a produtividade em projetos de *software* podem ser resumidos em: tamanho do produto, características do produto, complexidade do produto e volatilidade dos requisitos do produto.

4. Conclusões e Trabalhos Futuros

O sucesso de um projeto está relacionado com a produtividade com que o mesmo é desenvolvido. A maioria dos estudos de análise da produtividade se baseia na perspectiva quantitativa da produção, esquecendo assim dos fatores que afetam a produtividade. Dos modelos de estimativa de custo, apenas o COCOMO [Boehm 1996], leva em consideração alguns fatores.

Este trabalho propôs realizar uma compilação dos fatores que afetam a produtividade baseada nos estudos existentes na literatura, de modo a prover uma visão geral, agrupando-os em cinco grupos: fatores relacionados com processo, tecnologia, pessoas, cultura organizacional e produto. A Figura 6 apresenta um resumo da classificação apresentada ao longo do trabalho.

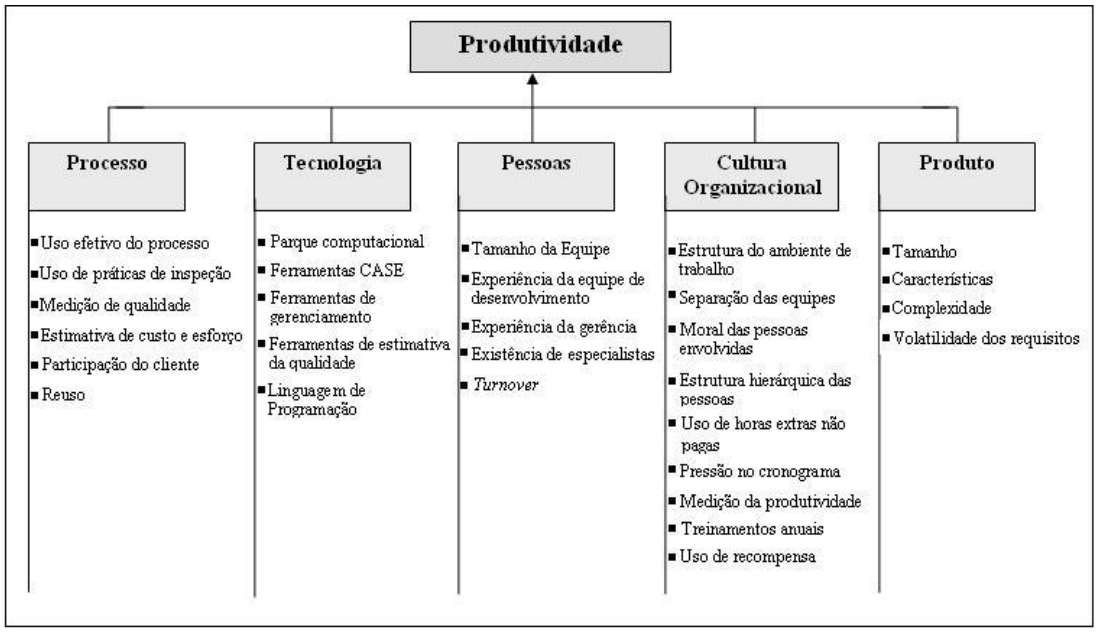

Figura 6. Classificação dos fatores que afetam a produtividade

[Bailey et al. 1981], [Mohanty 1981] e [Kemerer 1987] concluíram que as medições da produtividade, bem como as estimativas de utilização de recursos devem ser específicas para cada organização e ambiente de desenvolvimento de modo a apresentar medições com maior acurácia. Desse modo, é inviável propor um modelo de produtividade capaz de atender a todos os tipos de projetos, sendo necessário assim um modelo adaptável de acordo com as necessidades do projeto.

De acordo com [Drucker 1993], "gerir um negócio é balancear uma variedade de necessidades e objetivos". Baseado nisso, a Engenharia de Apoio a Decisão [Thomaz 2005] provê abordagens e metodologias para possibilitar melhores decisões diante dos mais variados problemas enfrentados nas mais variadas áreas. Uma delas é metodologia multicritério [Bouyssou, 1990] que se caracteriza pela construção de vários critérios através da utilização de vários pontos de vista que representam os eixos através dos quais os atores de um processo de decisão justificam, transformam e questionam as suas preferências, realizando comparações com base na avaliação das alternativas de acordo com estes pontos de vista e estabelecendo, então, as preferências parciais. Assim, uma metodologia multicritério de apoio à decisão, baseada na precisa identificação das situações de decisão e dos atores e seus problemas, procura, em primeiro lugar, construir uma estrutura partilhada, onde são consideradas as dimensões desejadas pelos atores, em vez de partir de uma situação pré-existente, sendo de seguida, se necessário, elaborado um modelo de avaliação onde, através de um procedimento técnico, as preferências dos decisores sejam agregadas a cada ação potencial que se queira avaliar. Sendo assim, um dos principais objetivos dos modelos multicritério é a agregação de diferentes (conflitantes ou irredutíveis) critérios num indicador de desempenho [Santana 2002].

Diante disso, como trabalho futuro, pretende-se propor um modelo multicritério de apoio à tomada de decisão no gerenciamento de projetos baseado nos fatores que afetam a produtividade, de modo a permitir que o decisor, no caso o gerente de projeto, tome as melhores decisões quanto ao gerenciamento baseado nos fatores que afetam a produtividade de *software* de acordo com as características específicas de um projeto.

Referências

Albrecht, A. J. and Gaffney, J. E. (1983) Software Function, Source Lines of codes, and Development Effort Prediction: A Software Science Validation. IEEE Trans Software Eng. SE-9, pp.639-648.

Bailey, J. and Basili, V. (1981) A Meta-Model for Software Development Resource Expenditures. In: Proc. 5th. Intern. Conf. Soft. Engr., IEEE Computer Society, pp 107-116.

Basili, V. et al. (1995) SEL's Software Process Improvement Program. IEEE Software, November 1995, pp. 887

Behrens, C. A. (1983) Measuring Software Productivity of Computer System Development Activities with Point Functions. IEEE Trans. Soft. Engr. SE-9(6), pp. 648-652.

Boehm, B. (1981) Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ.

Boehm, B. (1987) Improving Software Productivity. IEEE Computer, September, pp. 43-57.

Boehm, B. and Papaccio, P. N. (1988) Understanding and Controlling Software Costs. IEEE Transactions on Software Engineering, v. 14, no. 10, October, pp. 1462-1477.

Boehm, B. W. et al. (1996) The COCOMO 2.0 Software Cost Estimation Model. American Programmer, July, pp.2-17.

Boehm, B. W., Gray, T. E. and Seewaldt, T. (1984) Prototyping Versus Specifying: A Multiproject Experiment. IEEE Transactions on Software Engineering, May.

Bouyssou, D. (1990) Building Criteria: A Prerequisite for MCDA". In: C. A. Bana e Costa (Ed.), Readings in Multiple Criteria Decision Aid, Springer-Verlag, Berlin (pp. 58–80).

Brooks, F. (1995) Mythical Man-Month, Anniversary Edition. Addison-Wesley.

Bruckhaus, T. (1997) A Quantitative Approach for Analyzing the Impact of Tools on Software Productiviy. Doctoral Dissertation, McGill Univ , Montreal.

Bruckhaus, T., Madhavji, N. H., Janssen, I. and Henshaw, J. (1996) The Impact of Tools on Software Productivity. IEEE Softw. 13, 5, pp. 29-38.

Chiang, R. and Mookerjee, S. (2004) Improving Software Team Productivity. Communications of the ACM. Volume 47 , Issue 5. Pages: 89 – 93.

Clincy, V. A. (2003) Software Development Productivity and Cycle Time Reduction. Journal of Computing Sciences in Colleges. Volume 19, Issue 2. Pages: 278-287.

COSMIC-FFP Measurement Manual Version 2.1 (2001) Software Eng. Management Research Laboratory, Univ. of Quebec, Montreal, www.lrgl.uqam.ca/cosmic-ffp .

Crosby, P. B. (1980) Quality Is Free.

DeGrace, P. and Stahl, L. (1990) Wicked Problems, Righteous Solutions: A Catalog of Modern Software Engineering Paradigms, Yourdon Press.

DeMarco, T. (1999) Peopleware: Productive Projects and Teams, 2nd Ed. Dorset House Publishing.

Drucker, P. F. (1993) Management: Tasks, Responsibilities, Practices, Harper & Row Publishers, New York, NY.

Fenton, N. E. and Pfleeger, S. L. (1997) Software Metrics: A Rigorous and Practical Approach, PWS Publishing Company.

Freedman, D. P. and Weinberg, G. M. (1990) Handbook of Walkthroughs, Inspections and Technical Reviews, Third Edition, Dorset House.

Function Point Counting Practices Manual Release 4.0 (1994) Int'l Function Point Users Group, Westerville, Ohio.

Gilb, T. (1997) Software Metrics.

Gilb, T. and Graham, D. (1993) Software Inspection. Wokingham, England, Addison-Wesley.

Halstead, M. H. (1977) Elements of Software Science, Elsevier, New York.

Herbsleb, J. et al. (1994) Benefits of CMM Based Software Process Improvement: Initial Results. Pittsburgh: Software Engineering Institute, Document CMU/SEI-94-TR-13, August.

ISO/IEC 9126 (2001) Information Technology – Product Quality – Part1: Quality Model. In: International Standard ISO/IEC 9126, International Standard Organization, Junho.

Jiang, Z. and Comstock, C. (2007) The Factors Significant to Software Development Productivity. In: Transactions On Engineering, Computing And Technology. Volume 19.

Jones, C. (1977) Program Quality and Programmer Productivity. IBM Technical Report TR 02.764, January.

Jones, C. (1986) Programming Productivity, McGraw-Hill, New York.

Jones, C. (1991) Applied Software Measurement: Assuring Productivity and Quality. New York: McGraw-Hill.

Jones, C. (1994) Assessment and Control of Software Risks. Englewood Cliffs, N.J. Yourdon Press.

Jones, C. (1996) Applied Software Measurement: Assuring Productivity and Quality. 2 ed. McGraw-Hill.

Jones, C. (1999) Software Assesments, Benchmarks and Best Practices. Addison Wesley.

Jones, T. C. (1978) Measuring Programming Quality and Productivity. IBM System J. 17, pp 39-63.

Kemerer, C. F. (1987) An Empirical Validation of Software Cost Estimation Models. Communications ACM, 30(5), pp 416-429.

Kitson, D. H. and Masters, S. (1993) An Analysis of SEI Software Process Assessment Results, 1987-1991. In: Proceedings of the Fifteenth International Conference on Software Engineering (Washington, DC: IEEE Computer Society Press), pp. 68-77.

Lambert, G.N. (1984) A Comparative Study of System Response Time on Programmer Development Productivity, IBM Systems J. 23(1), pp. 36-43.

Laturi-System Product Manual Version 2.0 (1996) Information Technology Development Center, Helsinki, Finland.

Lim, W.C. (1994) Effects of reuse on quality, productivity, and economics. Hewlett-Packard Co. IEEE Software.

Maxwell, K. D. and Forselius, P. (2000) Benchmarking Software Development Productivity. IEEE Software.

Maya, M. et al. (2001) Measuring the Functional Size of Real-Time Software. Proc. 1998 European Software Control and Metrics Conf., Shaker Publishing BV, Maastricht, The Netherlands, pp. 191–199.

McConnell, S. (1996) Software Quality at Top Speed. Software Development, August.

McConnell, S. (1998) The Power of Process. IEEE Computer, May.

Mills, H. (1983) Software Productivity. Little, Brown & Co.

Mohanty, S. N. (1981) Software Cost Estimation: Present and Future. Software-Practice and Experience 11, pp 103-121.

Nelson, R. (1966) Management HandBook for the Estimation of Computer Programming Costs, AD-14 A648750, Systems Development Corp.

Park, R. E. (1988) PRICE S: The calculation within and why. In: Proceedings of ISPA Tenth Annual Conference, Brighton, England.

Parkinson, G. N. (1957) Parkinson's Law and Other Studies in Administration. Houghton-Miffin, Boston.

Pietrasanta, A. M. (1991) A Strategy for Software Process Improvement. In: Ninth Annual Pacific Northwest Software Quality Conference, October 7-8, Oregon Convention Center, Portland.

Putnam, L. H. (1978) A general empirical solution to the macro software sizing and estimating problem. IEEE Trans. Soft. Eng., pp. 345-361.

Rehesaar, H. (2001) Software Size: The Past and the Future. Proc. 1998 European Software Control and Metrics Conf., Shaker Publishing BV, Maastricht, The Netherlands, pp. 200–208.

Saiedian, H. and Hamilton, S. (1995) Case Studies of Hughes and Raytheon's CMM Efforts. IEEE Computer, January, pp. 20-21.

Santana, E. A. (2002) Contrato satisfatório multidimensional e a teoria do incentivo. Revista Brasileira de Economia, 56, 4 (pp. 661–694).

Scacchi, W. (1984) Managing Software Engineering Projects: A social Analysis. IEEE Trans. Soft. Engr.,SE-10(1), pp. 49-59.

Scacchi, W. (1995) Understanding Software Productivity. Appears in Advances in Software Engineering and Knowledge Engineering, D. Hurley (ed.), Volume 4, pp. 37-70.

Shepperd, M. and Cartwright, M. (2001) Predicting with Sparse Data. In: Proc. 7th Int'l Software Metrics Symposium, IEEE CS Press, Los Alamitos, Calif., pp. 28–39.

Shepperd, M. and Schofield, C. (1997) Estimating Software Project Effort Using Analogy. IEEE Trans. Soft. Eng. SE-23:12, pp. 736-743.

Shepperd, M. and Schofield, C. (1997) Estimating software project effort using analogy. IEEE Trans. Soft. Eng. SE-23:12, pp. 736-743.

Sidler, R. (2002) Software Productivity.

Simmons, D. B. (2001) Software Organization Productivity.

St-Pierre, D., Maya, M., Abran, A., Desharnais, J. and Bourque, P. (1997) Full Function Points: Counting Practice Manual, Technical Report 1997-04, University of Quebec at Montreal.

Tausworthe, R.(1981) Deep Space Network Software Cost Estimation Model. Jet Propulsion Laboratory Publication 81-7.

Taylor, B. (2005) Organizational Culture is Important in Software Productivity. http://www.workinginunison.com/papers/cultureandproductivity.pdf.

Thadhani, A. J. (1984) Factors Affecting Programmer Productivity During Application Development. IBM Systems J. 23(1), pp. 19-35.

Thomaz, J. P. (2005) O Apoio À Tomada De Decisão Na Avaliação Do Desempenho De Pessoas: Contributos Para O Processo De Decisão Militar Em Tempo De Paz. Tese de Doutorado. Universidade Técnica de Lisboa. Instituto Superior Técnico.

Vosburgh, J., Curtis, B., Wolverton, R., Albert, B., Malec, H., Hoben, S., and Liu, Y. (1984) Productivity Factors and Programming Environments. International Conference on Software Engineering. In: Proceedings of the 7th International Conference on Software Engineering. Orlando, Florida, United States. Pages: 143 – 152.

White, K. S. (1999) Software Engineering Management For Productivity And Quality. In: International Conference on Accelerator and Large Experimental Physics Control Systems, Trieste, Italy.

Yin, R. K. (1994) Applied Social Research Methods Series: Case Study. Sage Publications, London.