Universidade Federal de Pernambuco – UFPE
Centro de Informática – CIn

Emanuella Aleixo de Barros, Maria Joelza Lopes Guimarães Vasconcelos
emanuellaaleixo17@gmail.com, joelza.gv@gmail.com

Produtividade em projetos de software: Métricas e fatores que influenciam

Resumo

A busca pela melhoria contínua da produtividade tem se tornado primordial para a sobrevivência das organizações no mundo globalizado e competitivo que vivemos. No entanto, a grande questão é definir a forma de medir a produtividade quando se trata de desenvolvimento de software e o que afeta essa produtividade. Diversos estudos têm sido desenvolvidos no intuito de identificar quais são os fatores que influenciam na produtividade no desenvolvimento de projetos de software. Assim, os objetivos desta pesquisa são:

1. Prover informações significantes e úteis sobre os principais fatores que influenciam na produtividade através de um levantamento do estado da arte na área de produtividade de software
2. Mostrar algumas métricas utilizadas, as dificuldades para medir a produtividade de equipes de desenvolvimento de software e indicar boas práticas para medição.

1. Introdução

A competitividade empresarial, motivada pela globalização econômica, passou a ser a principal preocupação do mercado. O contexto mundial exige que as empresas melhorem cada vez mais seus níveis de qualidade e diminuam os seus custos de produção. A principal forma de diminuir custos de produção, especificamente no desenvolvimento de software, é através do aumento da produtividade, ou seja, desenvolver mais com um menor custo. Um diferencial competitivo para empresas ou grupos que atuam no desenvolvimento de sistemas é a capacidade de liberar novos produtos ou novas versões do produto com maior agilidade e menor custo, trazendo à tona a questão da produtividade no desenvolvimento de software. Além disto, a diminuição gradual do custo do hardware, o aumento da capacidade de processamento e a evolução da sociedade atual, fizeram com que a demanda pelo desenvolvimento de software venha crescendo de forma acelerada nos últimos anos.

Os objetivos deste artigo são dois: prover informações significantes e úteis sobre os principais fatores que influenciam na produtividade através de um levantamento do estado da arte na área de produtividade de software e mostrar algumas métricas utilizadas, as dificuldades para medir a produtividade de equipes de desenvolvimento de software, além de indicar boas práticas para medição.

O que é produtividade?

A definição básica de produtividade é dada por

$$\frac{\text{OUTPUT}}{\text{INPUT}}$$

Ou seja, é a razão entre o que é produzido (OUTPUT) e o que é consumido (INPUT). Ou ainda, podemos dizer que Produtividade = valor produzido/valor consumido. No contexto de desenvolvimento de software, para muitos autores, o valor produzido pode ser entendido como sendo o tamanho do software gerado e o valor consumido pode ser traduzido através do esforço necessário para desenvolver o software. Assim, a equação de produtividade fica sendo:

$$\frac{\text{TAMANHO DO SOFTWARE}}{\text{ESFORÇO PARA PRODUZI-LO}}$$

Qual a importância nas organizações?

A crescente competitividade entre as organizações devido à globalização torna a sobrevivência das organizações cada vez mais difícil. Para se manter no mercado, é preciso ser competitivo. Para ser competitivo, é preciso produzir cada vez mais produtos de qualidade com cada vez menos. É nesse contexto que a produtividade torna-se um aspecto de alta relevância para as empresas.

Por que medir a produtividade?

Somente medindo a produtividade, é possível avaliar possibilidades de melhoria, além de identificar pontos eficientes e ineficientes. Como a produtividade é essencial para que as organizações se mantenham competitivas, é preciso medi-la sempre.

2. Fatores que influenciam na produtividade em projetos de software

18 fatores que influenciam na produtividade foram identificados entre as fontes de pesquisa utilizadas para este levantamento. Os fatores foram classificados em 4 categorias: relacionados ao produto, às pessoas, ao projeto e à organização.

2.1. **Fatores relacionados ao produto:**

2.1.1. Restrições de recursos [1][4][5][6]

Tempo de execução, tempo de resposta, utilização de memória, armazenamento e ocupação da CPU são fatores que estão altamente correlacionados uns com os outros. Em [5], os resultados mostraram que a combinação de dois ou mais dessas restrições está particularmente associada com diminuição da produtividade.

2.1.2. Complexidade do software [1][5][6]

A complexidade do software pode ser medida pela porcentagem da dificuldade de escrever o código de um programa. Os tipos mais complexos de códigos são sistemas operacionais, controle em tempo real e rotinas de recuperação. A alta porcentagem de complexidade do código tem um impacto negativo na produtividade.

2.1.3. Confiança requerida pelo software [1][4]

No COCOMO [1], quantitativamente, a confiança requerida pelo software é definida como a probabilidade do software executar suas funções planejadas satisfatoriamente durante e nas próximas execuções. Em softwares com baixa confiança requerida o efeito de sua falha é de baixo nível e a perda é facilmente recuperável para os usuários, como por exemplo, um sistema de cadastro de funcionários. Em sistemas que requerem alta confiança a conseqüência da sua falha pode levar a uma perda financeira ou a uma grande inconveniência para as pessoas. Exemplos típicos são sistemas bancários e de distribuição de energia. Já em sistemas onde o nível de confiança precisa ser altíssimo, o resultado de uma falha pode levar à perda de vidas humanas como em sistemas de comando e controle militar ou de usinas nucleares.

Assim, quanto maior a confiança requerida pelo software maior será o impacto negativo na produtividade. A tabela 1 demonstra o esforço necessário de acordo com o nível de confiança requerido. Aplicações que necessitam de um alto nível de confiança despendem mais esforços na fase de integração e teste do que nas demais fases.

Tabela 1
Required Software Reliability (RELY) Effort Multipliers

Phase / RELY Rating	**Requirements and Product Design**	**Detailed Design**	**Code and Unit Test**	**Integration and Test**	**Overall**
Very low	0.80	0.80	0.80	0.60	0.75
Low	0.90	0.90	0.90	0.80	0.88
Nominal	1.00	1.00	1.00	1.00	1.00
High	1.10	1.10	1.10	1.30	1.15
Very High	1.30	1.30	1.30	1.70	1.40

Fonte: [1]

2.2. Fatores relacionados às pessoas

2.2.1. Experiência na linguagem de programação [1][4][6][11][14][15]

No estudo feito por Maxwell [4], a experiência na linguagem de programação não apontou um significante efeito na produtividade. Kitchenham [14] também encontrou pouca evidência empírica que mais experiência aumenta a produtividade de projetos. Embora a experiência na linguagem tenha a segunda maior taxa dentre os 29 fatores estudados por Walston and Felix [6], Brooks [11] analisou a mesma base de dados e determinou que experiência na linguagem de programação não é significante para projetos grandes. Boehm [1] incluiu a experiência na linguagem como um dos fatores do modelo COCOMO, embora tenha apresentado pouca taxa de variação em todas as fases do desenvolvimento, como pode ser visto na tabela 2. A análise feita por Kitchenham na mesma base de dados de Boehm confirmou que este resultado é significante, embora ela lembre que projetos com equipes com experiência normal têm nível de produtividade semelhante à projetos com equipes com alta experiência. Esta tendência também foi confirmada por Jeffrey e Lawrence [15] que encontraram que programadores COBOL não aumentaram sua produtividade depois de um ano de experiência com a linguagem.

Tabela 2
Programming Language Experience (LEXP) Effort Multipliers

Phase / LEXP Rating	Requirements and Product Design	Detailed Design	Code and Unit Test	Integration and Test	Overall
Very low (≤1 month)	1.02	1.10	1.20	1.20	1.14
Low (4 months)	1.00	1.05	1.10	1.10	1.07
Nominal (1 year)	1.00	1.00	1.00	1.00	1.00
High (≥ 3 years)	1.00	0.98	0.92	0.92	0.95

Fonte: [1]

2.2.2. Experiência na aplicação[1][6]

O esforço requerido para se desenvolver um sistema em relação à experiência na aplicação é mostrado na tabela 3. A experiência afeta principalmente as fases de requisitos e design do produto. Comparando as tabelas 2 e 3, na fase de requisitos e design do produto, observa-se que a diferença de esforço entre ter pouca (≤ 1 mês) e muita (≥ 3 anos) experiência na linguagem é 0,00. Já em relação à experiência na aplicação, a diferença de esforço é bem maior entre quem tem pouca (1 ano) e muita (6 anos) chegando a 0,33. Isto quer dizer que ter experiência na aplicação influencia bem mais na produtividade do que ter experiência na linguagem de programação.

Tabela 3
Applications Experience (AEXP) Effort Multipliers

Phase / AEXP Rating	Requirements and Product Design	Detailed Design	Code and Unit Test	Integration and Test	Overall
Very low (≤4 month)	1.40	1.30	1.25	1.25	1.29
Low (1 year)	1.20	1.15	1.10	1.10	1.13
Nominal (3 years)	1.00	1.00	1.00	1.00	1.00
High (6 years)	0.87	0.90	0.92	0.92	0.91
Very High (≥ 12 years)	0.75	0.80	0.85	0.85	0.82

Fonte: [1]

2.2.3. Motivação [1]

Motivação é o processo responsável pela intensidade, direção e persistência dos esforços de uma pessoa para o alcance de uma determinada meta. Muitos estudos encontraram que motivação tem uma influência mais forte na produtividade do que qualquer outro fator.

A experiência clássica [16] relativa à motivação e produtividade foi conduzida por Elton Mayo de 1927 à 1932 na fábrica da Western Electric Company, situada em Chicago, no bairro de Hawthorne e ficou conhecida como Efeito Hawthorne. Na época, a empresa valorizava o bem-estar dos operários, mantendo salários satisfatórios e boas condições de trabalho e não estava interessada em aumentar a produção, mas em conhecer melhor seus empregados.

A finalidade do experimento era determinar a relação entre a intensidade da iluminação e a eficiência dos operários medida através da produção. Os resultados dos grupos examinados foram os seguintes:

- Quando a iluminação foi aumentada, a produtividade aumentou.
- Quando a iluminação foi diminuída, a produtividade aumentou.
- Quando a iluminação foi mantida constante, a produtividade aumentou.

Depois de uma longa série de novas experiências que tentaram explicar esse efeito, chegou-se a conclusão que a produtividade aumenta quando há a percepção dos trabalhadores que a direção da empresa dá atenção a eles.

2.3. Fatores relacionados ao projeto

2.3.1. Desenvolvimento de software concorrente com hardware [5][6]

Produtos que requerem desenvolvimento de software concorrente com hardware têm uma menor produtividade. Uma explicação para este impacto negativo na produtividade é que mudanças na configuração final do hardware afeta a interface de programação, o que normalmente requer uma nova implementação do software.

2.3.2. Interface com o cliente [5][6]

Maior participação do cliente na preparação dos requisitos e maior experiência dele com a aplicação, ambas estão associadas com a alta produtividade. O cliente deve estar envolvido no projeto desde o início, pois é quando os problemas podem ser resolvidos com menos dificuldades e custos.

2.3.3. Estabilidade dos requisitos [1][6]

A quantidade de mudanças nos requisitos do software entre o início e o fim do projeto de desenvolvimento é claramente um fator que afeta a produtividade de software. Segundo Vosburgh [5], a estabilidade e a precisão da especificação dos requisitos estão relacionadas à questão de quando estes são especificados pelo cliente ou pela empresa que vai desenvolver o software. Neste estudo, tanto a estabilidade quanto a precisão foram baixas quando as especificações foram escritas pelo cliente havendo assim, a necessidade de reescrita destas.

2.3.4. Tamanho do projeto [1][4]

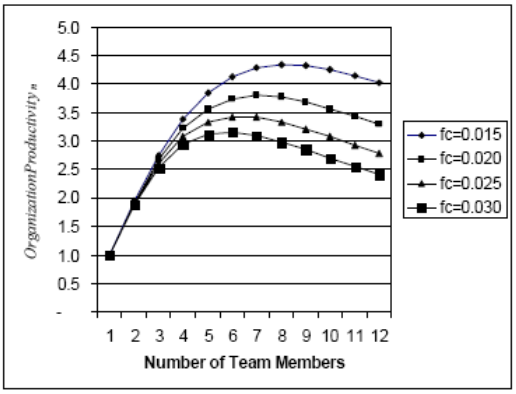
Em [4] foi encontrado um pequeno aumento da produtividade em projetos maiores. Mas, este resultado é o oposto ao de outras pesquisas, citadas neste mesmo estudo, onde o aumento do tamanho do projeto pode levar à diminuição da produtividade, principalmente devido à complexidade para gerenciá-los.

2.3.5. Tamanho da equipe [4][5][12][13]

Quando o tamanho da equipe aumenta, a produtividade diminui. Isto provavelmente ocorre devido a problemas de comunicação e coordenação que acontecem quando muitas pessoas trabalham em um projeto. A lei de Brooks [12] diz que: “adicionar mais gente a um projeto que já está atrasado aumenta o atraso ainda mais”, ou seja, adicionar pessoas ao projeto não aumenta a produtividade global na mesma proporção, o pior é que pode diminuir. Uma das explicações para isso é que novos recursos humanos levam mais tempo para se integrarem às tarefas já realizadas e para aprenderem métodos e modelos já em uso pela equipe original.

Simmons [13] propôs uma fórmula que determina o número máximo de membros que podem ser adicionados à uma equipe para que a produtividade possa aumentar. A figura 1 mostra o que acontece com a produtividade quando membros são adicionados à uma equipe.

Figura 1
Organization Productivity$_n$ of teams that have different communication factors where *fs=0*.

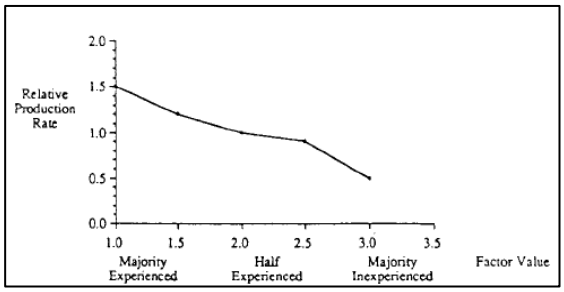


Fonte: [13]

2.3.6. Experiência da equipe [5][7][8]

Em [7] a experiência da equipe foi classificada como tendo um alto impacto na produtividade. Conforme figura abaixo, a diferença de produtividade entre uma equipe que tem pessoas mais experientes e outra que tem pessoas mais inexperientes é cerca de 100%.

Figura 2
Staff Experience Factor



Fonte: [7]

2.3.7. Turnover [1][9]

Quanto maior a rotatividade de uma equipe, menor é a sua produtividade, pois uma pessoa, que integra uma equipe de um projeto em andamento, precisa de tempo para conhecer ambiente de desenvolvimento, modelos de objetos, especificações e outros componentes já produzidos.

2.3.8. Linguagem de programação [1][4][10]

Diversas pesquisas encontraram que a produtividade, medida usando linha de código ou pontos de função, varia de acordo com o nível da linguagem de programação. Já alguns estudos de produtividade removeram este fator por considerarem apenas

programas escritos na mesma linguagem ou por converterem todos os dados em uma linguagem usando fatores de conversão. No estudo feito por Maxwell, et. al [4] linguagens de alto nível não parecem menos produtivas que linguagens de baixo nível quando a produtividade é medida em linhas de código. Este resultado contradiz o paradoxo da linha de código proposto por Jones [10] no qual "quando programas são escritos em linguagens de alto nível sua produtividade aparente, expressa em código fonte por unidade de tempo, é menor do que em aplicações similares escritas em linguagens de baixo nível".

2.3.9. Uso de práticas modernas de programação [1][4][5][6]

Diversos estudos apontaram que o uso de práticas modernas de programação como, por exemplo, arquitetura top-down e modular, revisão e inspeção de código, e garantia da qualidade têm uma forte correlação com o aumento da produtividade, uma vez que, com estas práticas, erros e/ou defeitos podem ser evitados e/ou identificados o mais breve possível.

2.3.10. Uso de ferramentas [1][2][4]

Ferramentas podem ajudar a melhorar o processo de desenvolvimento facilitando atividades que antes não eram realizadas como, por exemplo, ferramentas de teste que ajudam a introduzir novas atividades de teste. Ferramentas também podem aumentar a produtividade dando suporte à atividades de desenvolvimento que são normalmente realizadas com pouco ou nenhum suporte de ferramentas. No entanto, em certas situações, a introdução de uma ferramenta pode também diminuir a produtividade. Isto acontece quando a ferramenta aumenta o esforço na especificação das atividades ou introduz novas atividades no processo tal como geração e manutenção de novos dados. Logo, dependendo das características do projeto, a mesma ferramenta pode ter diferentes efeitos na produtividade.

2.3.11. Reuso [1][3]

Em geral, o reuso aumenta a produtividade devido à redução da quantidade de tempo e esforço necessários para desenvolver e manter um produto de software uma vez que o ciclo de vida requer menos input para obter o mesmo output.

2.4. Fatores relacionados à organização

2.4.1. Companhia [4]

A produtividade pode variar de acordo com as características da companhia que desenvolve o sistema. Esta variação pode ocorrer devido à linguagem de programação usada pela companhia, à qualidade do produto final, aos computadores usados para o desenvolvimento, às pessoas da equipe, aos atributos do projeto, ao estilo de gerenciamento, ao critério de seleção, dentre outros.

3. Visão geral

A visão geral dos fatores que influenciam na produtividade, citados neste estudo, está sumarizada na tabela 4.

Tabela 4
Visão geral dos fatores que influenciam na produtividade citados neste estudo

Fator	[1]*	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	PD	PE	PJ	OR	C	N C
RR	X			X	X	X										X					X
CSW	X				X	X										X					X
CRS	X			X												X					X
EXPLP	X			X		X					X			X	X		X			X	
EXPAP	X					X											X			X	
MT	X																X			X	
DSCH					X	X												X			X
IC					X	X												X			X
ER	X					X												X		X	
TP	X			X														X			X
TE				X	X							X	X					X		X	
EXPE					X		X	X										X		X	
TO	X								X									X			X
LP	X			X						X								X		X	
UMPP	X			X	X	X												X		X	
UF	X	X		X														X		X	
RS	X		X															X		X	
CIA				X															X		X

*Legenda:

[1] à [15]: Fatores citados nos estudos referenciados na bibliografia.

RR	Restrições de Recursos	**TP**	Tamanho do Projeto
CSW	Complexidade do Software	**TE**	Tamanho da Equipe
CRS	Confiança Requerida pelo Software	**EXPE**	Experiência da Equipe
EXPLP	Experiência na Linguagem de Programação	**TO**	Turnover
EXPAP	Experiência na Aplicação	**LP**	Linguagem de Programação
MT	Motivação	**UMPP**	Uso de Modernas Práticas de Programação
DSCH	Desenv. de Software Concorrente com Hardware	**UF**	Uso de Ferramentas
IC	Interface com o Cliente	**RS**	Reuso
ER	Estabilidade dos Requisitos	**CIA**	Companhia

PD	Fatores relacionados ao Produto	**PJ**	Fatores relacionados ao Projeto
PE	Fatores relacionados às Pessoas	**OR**	Fatores relacionados à Organização

C	Fator que está sob o controle do gerente de projeto	**NC**	Fator que não está sob o controle do gerente de projeto

4. Métricas utilizadas para medir a produtividade

Nos artigos pesquisados, algumas métricas e modelos para medir a produtividade são mostrados:

- Pontos de função
- Métricas para softwares orientados a objetos
- Abordagem vetorial para medir o tamanho e o esforço
- Pontos de casos de uso
- Quantidade de linhas de código
- Quantidade de defeitos encontrados (métrica de qualidade)
- Unidades de trabalho e de custos

Uso de pontos de função para medir produtividade

O artigo [17] mostra o resultado de um estudo realizado em 1980 e 1981 em uma grande instituição financeira que utilizou pontos de função como metodologia para medição de produtividade.

Foram coletados dados de 11 projetos em 1980 e 14 projetos em 1981.

Ano	1980	1981
Qtd de pontos de função dos projetos	De 27 a 599	De 22 a 435
Qtd de horas utilizadas nos projetos	De 600 a 28.700	De 85 a 10.600
Qtd de horas por ponto de função	De 9,7 a 47,9	De 2,1 a 23,4
Média de horas por ponto de função	18,3	9,14

Houve uma redução de 8,9 horas por pontos de função, ou 49% nos projetos de 1981. Por que isso aconteceu? Foram feitas análises de vários fatores que pudessem ter contribuído para esta redução. A princípio, a conclusão foi que tamanho do projeto, ambiente de desenvolvimento e a linguagem de programação haviam sido determinantes para a produtividade. Outros fatores como experiência no sistema, características do usuário (cliente) não foram relevantes para a produtividade. A seguir foi verificado que a linguagem de programação não foi considerada estatisticamente significante. Sendo assim, nesse estudo, o tamanho do projeto e o ambiente de desenvolvimento, claramente foram fatores com grande importância e influência na produtividade.

Uso gerencial de métricas para softwares orientados a objetos

No artigo [18] é utilizado um conjunto com seis métricas proposto por Chidamber e Kemerer (simplesmente CK daqui em diante) em 3 grandes projetos de uma instituição financeira.

- NOC: quantidade de subclasses imediata da classe analisada.
- WMC: quantidade de métodos da classe.
- DIT: distância (quantidade de níveis) até a classe raiz da hierarquia.
- CBO: acoplamento com as outras classes, medido através da quantidade de métodos e variáveis de instância usados de outras classes.
- RFC: quantidade de métodos que podem ser executados através de outras classes.

- LCOM: quantidade de métodos que não tem variáveis de instância menos a quantidade de métodos que tem.

Essas métricas foram desenvolvidas para medir a complexidade de projeto, a variação na produtividade, o esforço de retrabalho e o esforço de projeto, de forma que possam ser estimados, em parte, em função dos valores das métricas. Os dados coletados visam examinar a relação entre essas variáveis e as métricas definidas.

Para realizar essa análise, a organização verificada precisa está familiarizada com o desenvolvimento orientado a objetos. É preciso ressaltar que, devido ao estado ainda incipiente das pesquisas em métricas OO, essa análise é explanatória por natureza, não podendo ser generalizada.

Todos os 3 projetos foram concluídos na década de 1990. Dos 3, um deles, havia concluído a etapa de projeto, não tendo, portanto, código implementado.

Além das 6 métricas citadas acima, as variáveis abaixo foram usadas na análise desse estudo:

- SIZE: quantidade de linhas de código da classe.
- EFFORT: quantidade em horas do tempo do desenvolvedor.
- PRODUCTIVITY: SIZE/EFFORT.
- STAFF_ON: quantidade de variáveis consideradas de controle nos modelos, pelos desenvolvedores.

Produtividade, esforço de retrabalho e esforço de projeto são as variáveis dependentes analisadas. Foram usados modelos de regressão neste estudo.

Conclusões:

- Dados para métricas podem ser coletados de sistemas desenvolvidos em diferentes linguagens e de sistemas que ainda foram codificados;
- Nenhuma das 3 aplicações mostrou uso relevante de herança (DIT e NOC tenderam a valores mínimos);
- WMC, CBO e RFC tendem a estar altamente correlacionadas;
- Altos níveis de acoplamento (CBO) e baixa coesão (LCOM) estão associados a:
    - menor produtividade
    - maior esforço de retrabalho
    - maior esforço de projeto

Os baixos valores de DIT e NOC indicam que as oportunidades de reuso oferecidas pela herança estão sendo talvez comprometidas em favor de uma melhor compreensão global da arquitetura do sistema.

O alto grau de correlação entre WMC, CBO e RFC pode explicar porque CBO foi considerado um preditor estatisticamente significante nos 3 modelos. No entanto, essa correlação precisa ser replicada antes que seu grau de independência possa ser determinado com confiança.

Nos 3 sistemas, a análise de regressão mostrou que CBO e LCOM são variáveis explanatórias significantes. Este resultado pode ser considerado mais importante, uma vez que, cada um dos modelos usou uma variável diferente (produtividade, esforço de retrabalho e esforço de projeto).

O pequeno conjunto de dados reduz o poder estatístico e a confiabilidade das estimativas, porém o conjunto de métricas CK tem sido mostrado como validação empírica que impacta as decisões de projeto OO e as variáveis gerenciais de produtividade e custo.

Uma abordagem vetorial para medir o tamanho do software e estimar esforço

O artigo [19] define e valida os modelos VSM (Medição Vetorial de Tamanho) e VPM (Modelo Vetorial de Predição). A proposta do VSM é medir o tamanho e classificar os softwares. Já a proposta do VPM é estimar o esforço de desenvolvimento no início do ciclo de vida do software.

O tamanho do software pode ser considerado como sendo uma função do comprimento (número total de entidades do sistema), funcionalidades do sistema e complexidade do mesmo. É difícil medir o software no início do seu ciclo de vida, mas sabemos que é possível fazer isso a partir das especificações. Os modelos VSM e VPM irão utilizar a linguagem ASL (Algebraic Specification Language). ASL não possui palavras reservadas e possui poucos símbolos.

- ASL: será usada como mecanismo para derivar o VSM.
- VPM: pode ser aplicado no início do ciclo de vida do software para estimar o esforço medindo as especificações escritas em ASL.
- VPM: utiliza um modelo de regressão multivariável para determinar o relacionamento entre esforço, magnitude e gradiente.
- Magnitude: é uma entrada primária para o VPM e representa a medida do tamanho do software levando em consideração as funcionalidades e a complexidade do problema.
- Gradiente: é utilizado como guia para custos e é uma medida relacionada às dimensões do sistema usadas para classificá-los. Por exemplo, sistemas em tempo real são considerados computacionalmente complexos.

No estudo realizado, todos os modelos de predição mostram uma correlação entre os dados reais e estimados estatisticamente significantes ($p < 0.05$).

O VSM pode ser usado para classificar os sistemas que é bastante importante para o VPM, que utilizada a classificação do sistema como gradiente (entrada). Foi verificado que a correlação entre magnitude e gradiente é muito baixa, o que significa que são variáveis independentes. Já a relação entre funcionalidade e complexidade do problema é forte, uma vez que a funcionalidade impacta diretamente na complexidade e vice-versa. E a correlação entre magnitude e esforço é alta e significativamente melhor que a de pontos de função e esforço.

As limitações dessa abordagem são:

a) Especificações precisam ser escritas em ASL;

b) À medida que o projeto anda, o tamanho do software pode mudar;

c) Diferentes pessoas modelam e especificam sistemas de diferentes formas, o que pode resultar em diferentes medições de tamanho de software e esforço;

d) Há projetos e produtos que não se adéquam a esse método. È necessário existir um processo de desenvolvimento de software bem definido;

e) Alguns tipos de problemas não podem ser especificados adequadamente em ASL. Isso pode resultar em medições de tamanho desproporcionais e estimativa de esforço incorreta.

A proposta da representação vetorial provê uma visão balanceada onde o tamanho do software é medido em função da magnitude e do gradiente. Essa forma dá uma visão mais completa quando comparada a abordagens que levam em consideração

apenas as funcionalidades ou comprimento isoladamente. É necessário, no entanto, métodos e técnicas para colocar em prática essa abordagem. A UML (Unified Modeling Language) pode ser usada para colocar em prática VSM e VPM. Por exemplo, requisitos (especificados em termos de modelo de casos de uso, modelo de domínio e modelo de transição de estado) usados no estudo foram modelos usando UML. O principal problema é ASL que foi especificamente projetada para ser usada no VSM, é necessário, portanto substituí-la. A UML Object Constraint Language (OCL) tem potencial para substituir ASL e está diretamente integrada com uma linguagem de modelagem com ferramentas CASE que a suporta.

Uso de pontos de casos de uso para medir tamanho de software

O artigo [20] utiliza o método de pontos de casos de uso para medir tamanho de softwares baseados nas especificações dos requisitos. Com o tamanho e esforço medidos, a produtividade real pode ser calculada em função das funcionalidades entregues.

No final do ano de 1993, foram coletados dados em uma instituição financeira da Suíça de 23 projetos (análise quantitativa) e foram avaliados 64 questionários respondidos por membros dos projetos e 11 entrevistas com gerentes de projetos selecionados (análise qualitativa).

A idéia de pontos de caso de uso foi inspirada pelo método de pontos de função e começou com a medida das funcionalidades especificadas em casos de uso e cenários. Fatores técnicos que são relevantes à funcionalidade requerida do software são integrados nessa medição. O resultado de pontos de casos de uso pode ser diretamente comparado aos pontos de função, inclusive, este último foi utilizado para calibrar o método de pontos de casos de uso, de forma que, os pontos de caso de uso ficaram entre 90% e 110% dos pontos de função comparados. Esse nível de incerteza é suficiente para a proposta, uma vez que, o método de pontos de função pode ter uma incerteza de 30%, quando a estimativa de tamanho é feita por diferentes líderes de projeto.

Os 23 projetos que fizeram parte da análise quantitativa podem ser caracterizados da seguinte forma:
- Todos são projetos para desenvolvimento de aplicações bancárias;
- Todos são aplicações interativas;
- O tamanho das aplicações varia entre 200 e 4.440 pontos de caso de uso;
- A duração do desenvolvimento varia entre 1 e 5 anos.

A utilização de pontos de casos de uso foi bem aceita pelos líderes de projetos. A medição do tamanho do software é normalmente feita em uma ou duas horas. A grande vantagem dos pontos de caso de uso é a aceitação que pode ser explicada pela simplicidade do método e a facilidade da notação.

Problemas encontrados:
- Os casos de uso nem sempre são atualizados (é escrito uma vez e não é nunca revisado);
- Os projetos diferem no grau detalhamento das descrições dos casos de uso o que impacta diretamente na medição do tamanho do software;
- A modelagem de cenário não é muito bem entendida;
- É difícil documentar grandes projetos com os conceitos dados pelo método de pontos de casos de uso;

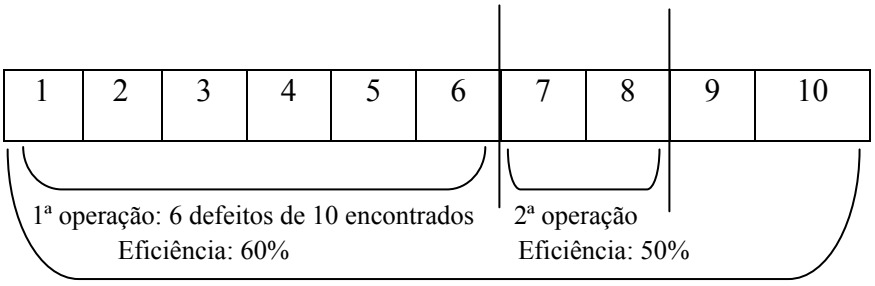
- Apesar da alta aceitação na prática, a sintaxe e semântica da notação gráfica descreve o modelo de casos de uso e de cenários são incompletas, imprecisas e parcialmente inconsistentes;
- O uso de descrições textuais livres torna, algumas vezes, difícil de se obter especificação sem ambigüidades;
- É difícil detectar ambigüidades e inconsistências nas especificações devido à diversidade de notação e fragmentação das informações dos requisitos em vários modelos, o que gera ainda problemas de integração;
- O processo de elicitação de requisitos, descreve e revisa diferentes modelos baseado principalmente em regras fracas e heurísticas;
- Alguns gerentes de projetos não aceitam a importância da especificação de requisitos;
- Não há regras para definir o grau correto de detalhamento nas descrições.

Lições aprendidas e conclusões:
- Especificações de requisitos com casos de uso e cenários podem ser usadas para medir o tamanho do software;
- O método de pontos de caso pode ser suportado por ferramentas simples e de baixo custo;
- As descrições textuais livres não trazem dados suficientes para medir o tamanho do software;
- 6 dos gerentes de projeto entrevistados desconfiaram da medição de tamanho de software realizada;
- Medições automáticas não são possíveis de serem feitas no momento devido à distância entre a formalização e as ferramentas de suporte;
- Desenvolvedores devem se aproximar mais dos usuários para que isso gere documentos de requisitos melhores de melhor qualidade. A completude deve ser buscada por desenvolvedores e usuários a fim de se obter medições de tamanho mais exatas;
- Devem ser feitos esforços no intuito de padronizar as documentações.

Quantidade de defeitos encontrados (métrica de qualidade)

O artigo [21] define qualidade como ausência de defeitos que possam causar um comportamento inesperado ou parar a execução de um programa. Há duas formas de minimizar os defeitos, uma delas é evitar que eles ocorram e a outra é remover os defeitos que venham a ocorrer. Portanto, uma métrica de qualidade está diretamente relacionada com a quantidade de defeitos encontrada e a eficiência em removê-los.

No que diz respeito à remoção de defeitos, devemos considerar a eficiência de remoção de defeitos e a eficiência cumulativa de remoção de defeitos. Esses conceitos de eficiência e eficiência cumulativa de remoção de defeitos são melhores explicados no desenho a seguir:



Eficiência cumulativa: 8 defeitos encontrados de 10: 80%

Na segunda operação, a eficiência foi de 50% uma vez que 2 de 4 defeitos foram encontrados. Ao final das duas operações, a eficiência cumulativa foi de 80% (8 erros encontrados de 10).

Problemas na correção é um aspecto relevante, pois é necessário quantificar a eficiência de remoção de defeitos corretamente. Mudanças de requisitos também afetam essa medição, uma vez que um programa que entra na segunda operação pode ser completamente diferente do que saiu da primeira operação. É preciso atentar para a possibilidade de inserção de novos erros ao se tentar remover antigos. Assim é possível obter a eficiência cumulativa através de uma equação simples, mostrada a seguir:

$$\frac{\text{Nº de defeitos encontrados antes do release}}{\text{Nº de defeitos encontrados antes e depois do release}}$$

Embora a qualidade esteja relacionada com vários outros fatores, sabemos que se pudermos tornar mais eficiente a forma de remoção de defeitos e se reduzirmos a quantidade de defeitos originais encontrados, teremos um produto com qualidade melhor.

Unidades de trabalho e de custos

Nos exemplos mostrados no artigo [21], foi observado que a média de produtividade cai a medida que o tamanho do software aumenta. No entanto, quando se mede o custo de manutenção e de mudanças, ocorre o inverso. Sendo assim, unidades de custo são mais adequadas que as de trabalho. Porém, são mostrados dois problemas com essas unidades: custo por defeito (manutenção corretiva) e custo por página (documentação). O custo por defeito não é uma unidade confiável, desde que penaliza os programas de alta qualidade (tendem a não apresentar problemas simples). Um método melhor é avaliar o custo de correção pó 1.000 linhas de código ou por ponto de função. Quanto ao custo por página, o problema é deixar o valor da página muito baixo quando há grande quantidade de espaços em branco.

4.1. Dificuldades em medir a produtividade de equipes de desenvolvimento de software

Mesmo quando se define uma métrica, existe um risco alto de que ocorram disfunções. Uma disfunção ocorre quando o agente fornecedor do valor da métrica realiza alguma manipulação na coleta. Vamos tomar como exemplo uma métrica de uma equipe de suporte representada pela quantidade de chamados atendidos. Se o agente (qualquer pessoa da equipe de suporte) souber que a quantidade de chamados atendidos no dia por ele está sendo analisada por seu superior, ele pode manipular de alguma forma esse valor, como por exemplo, solicitando que um chamado que poderia ser atendido facilmente seja quebrado em dois ou mais, o que tornará o processo de atendimento de chamado muito mais burocrático e lento para a empresa. Se um segundo agente não agir dessa forma estará agregando valor e contribuindo para a rapidez de atendimento dos chamados, por outro lado se a métrica que está sendo avaliada por apenas a citada acima (quantidade de chamados atendidos), o primeiro agente será considerando mais produtivo que o segundo. Isso é o que caracteriza uma disfunção.

Um programa de medição pode ser implantado de 3 formas:

- Sem supervisão;
- Totalmente supervisionado;
- Parcialmente supervisionado.

Em um ambiente totalmente supervisionado, todas as variáveis ou dimensões envolvidas na produtividade são medidas. Já um ambiente parcialmente supervisionado, apenas algumas variáveis ou dimensões selecionadas são medidas. Em um ambiente sem supervisão, não há variáveis medidas, logo não há métricas geradas.

O cenário ideal é medir todas as dimensões envolvidas, assim torna-se praticamente nula a probabilidade de ocorrerem disfunções. Exemplo: quantidade de chamados, tempo de atendimento, grau de satisfação do cliente, etc. Porém, é praticamente, se não totalmente, inviável a implantação de um programa de medição totalmente supervisionado, porque além de extremamente custoso, não podemos garantir que todas as dimensões estão sendo medidas. Com isso, os cenários que temos presentes nas organizações são ambientes não supervisionados ou parcialmente supervisionados. No caso de um ambiente não supervisionado não temos a presença da disfunção, uma vez que não há métricas. Por outro lado, não existem métricas para que a produtividade possa ser avaliada, comparada e evoluída. Logo, um ambiente parcialmente supervisionado é o mais comum de encontrarmos nas organizações que querem medir a produtividade de alguma forma. São selecionadas algumas variáveis consideradas importantes, de acordo com a necessidade da empresa e elas serão medidas para posterior avaliação. Neste cenário, podemos imaginar que mesmo não tendo todas as variáveis sendo medidas, podemos aferir a produtividade das organizações de alguma forma. O grande problema é justamente a possibilidade de ocorrerem disfunções.

4.2. Boas práticas pra medição de produtividade em equipes

O que pode propiciar a disfunção?

Não devemos usar as métricas com o intuito de premiar, recompensar ou punir as pessoas. Se isso for feito, certamente em algum momento os agentes perceberão e irão manipular os valores das métricas trazendo, inclusive, um efeito negativo para a produtividade.

O que pode ser feito?

Medições agregadas podem trazer benefícios [22]. Medir coletivamente ao invés de individualmente, fazendo com que o superior veja a métrica coletiva de sua equipe. Nessa forma de medição, é possível ver as métricas de seus pares, mas nunca de seus subordinados. No entanto, isso só irá funcionar de fato se o coletor das métricas for externo e isento à empresa e se houver garantia total de que os dados nunca poderão ser quebrados. Se houver qualquer dúvida sobre isso, o sistema falhará. Para equipes muito pequenas, não irá funcionar, pois será possível facilmente deduzir as métricas de seus pares.

5. Conclusão

Os objetivos deste artigo foram prover uma visão geral sobre os principais fatores, citados na literatura, que influenciam na produtividade em projetos de software e mostrar algumas métricas utilizadas, as dificuldades para medir produtividade, além de indicar boas práticas pra medição. Como pôde ser visto, alguns autores não têm a mesma visão do impacto (positivo ou negativo) do fator na produtividade como, por exemplo, a linguagem de programação e a experiência nela e o tamanho do projeto, mas a grande maioria dos fatores são consenso nos estudos. A categoria que mais concentra os fatores é a relacionada ao projeto, 11 ao todo. Não estão sob o controle do gerente de projeto os fatores relacionados ao produto e outros relacionados ao projeto e à organização.

"Não se pode controlar o que não se pode medir". Essa frase de Tom DeMarco [26] explica bem a necessidade de usar um programa de medição. Porém, é preciso muito cuidado na escolha das métricas e conhecer os fatores que mais afetam a produtividade no projeto e só depois selecionar bem as métricas de forma a medir o maior número possível de dimensões que agreguem valor para o usuário (cliente).

Bibliografia

[1] Boehm, B.W., Software Engineering Economics, Prentice–Hall, Englewood Cliffs, New Jersey, 1981.

[2] Bruckhaus T., Madhavji N. H., Janssen I., Henshaw J., The Impact of Tools on Software Productivity, IEEE Software, v.13 n.5, p.29-38, September 1996.

[3] Lim C. W., Effects of Reuse on Quality, Productivity, and Economics, IEEE Software, v.11 n.5, p.23-30, September 1994.

[4] Maxwell K. D., Wassenhove L. V., and Dutta S., Software development productivity of European space, military, and industrial applications, Software Engineering, IEEE Transactions on, Vol. 22, No. 10. (1996), pp. 706-718.

[5] Vosburgh, J., Curtis, B., Wolverton, R., Albert, B., Malec, H., Hoben, S., and Liu, Y., Productivity Factors and Programming Environments, Proc. Seventh Int'l Conf. Software Eng., pp. 143-152, 1984.

[6] Walston, C. E., and Felix, C. P., A method of programming measurement and estimation, IBM Systems Journal, vol. 16, Number 1, pp. 54-73 1977.

[7] Yu, W. Smith, D. Huang, S. Software productivity measurements. Proceedings of Computer Software and Applications Conference, 1991.

[8] http://www.bfpug.com.br/Artigos/sistematica_metricas_simoes.htm
Acesso em: 16/10/2008

[9] http://www.inf.ufsc.br/~sbes99/anais/SBES-Completo/12.pdf
Acesso em: 20/10/2008

[10] Jones, C. Assessment and Control of Software Risks. Englewood Cliffs, N.J.: PTR Prentice Hall, 1994.

[11] Brooks, W.D., Software Technology Payoff Some Statistical Evidence, J. Systems and Software, vol. *2,* pp. 3-9, 1981.

[12] Brooks, F.P.: The Mythical Man Month, Essays on Software Engineering, Anniversary Edition, Addison-Wesley, Boston etc., 1995

[13] Simmons, D., Ellis, N., Fujihara, H., and Kuo W., Software Measurement: A Visualization Toolkit for Project Control and Process Improvement, Prentice Hall, Upper Saddle River, NJ, 1998.

[14] Kitchenham, B. A, Empirical Studies of Assumptions that Underlie Software Cost-Estimation Models, Information and Software Technology, vol. 34, no. 4, Apr. 1992.

[15] Jeffrey, D.R. and Lawrence, M.J., Managing Programming Productivity, J. Systems and Software, vol. 5, pp. 49-58, 1985.

[16] Mayo, E., The Social Problems of an Industrial Civilization. Boston, Graduate School of Business Administration 1945, Harvard University.

[17] Behrens C.A, Measuring the Productivity of Computer Systems Development Activities with Function Points.

[18] Chidamber, S. R, Darcy, D. P., and Kemerer, C. F., Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis.

[19] Hastings, T.E., Member, IEEE, and Sajeev, A.S.M., Member, IEEE, A Vector-Based Approach to Software Size Measurement and Effort Estimation.

[20] Buglione, L., and Abran, A., Multidimensional software performance measurement models: A tetrahedron based design.

[21] Arnold, M., and Pedross, P., Software Size Measurement and Productivity Rating in a Large-scale Software Development Department.

[22] Jones, T. C., Measuring programming quality and productivity.

[23] Austin, R. D., Measuring an Managing Performance in Organizations.

[24] Chatman, V. V., Change Points: A proposal for software productivity measurement.

[25] Campos, V. F., TQC controle da qualidade total: (no estilo japonês). 8 ed. Belo Horizonte: Desenvolvimento Gerencial, 1999.

[26] DeMarco, T. and Lister T., Peopleware: Productive Projects and Teams, 2nd Ed. Dorset House Publishing Co., 1999.