Revisitando as Proposições de Brooks em The Mythical Man-Month – Capítulo 8 – Calling the Shot (http://luizricardo.org/2016/08/revisitando-as-proposicoes-de-brooks-em-the-mythical-man-month-capitulo-8-calling-the-shot/)

23/08/2016 (http://luizricardo.org/2016/08/revisitando-as-proposicoes-de-brooks-em-the-mythical-man-month-capitulo-8-calling-the-shot/) Assuntos Randômicos (http://luizricardo.org/categoria/assuntos-randomicos/)

Em 1975, Frederick P. Brooks escreveu o revolucionário *The Mythical Man-Month*. **Quarenta** anos depois, esta série de artigos pretende revisar as afirmações e os argumentos do autor. Quanto o ofício da Engenharia de Software evoluiu desde então?

Capítulo 8: *Calling the Shot*

Neste capítulo, Brooks discorre sobre as dificuldades e algumas armadilhas relacionadas à estimação e à produtividade em projetos de software com base em dados de algumas pesquisas realizadas na época. Vamos conferir o que o autor encontrou a seguir.

> *Não se estima o esforço total ou cronograma de um projeto de programação simplesmente estimando o tempo de codificação e multiplicando por fatores para incluir as outras tarefas.*

O autor propôs em capítulos anteriores a proporção de tempo para se investir em codificação e testes. Porém, trabalhar com código é uma parte menor do total investido na maioria dos projetos de software.

> *Dados para o desenvolvimento de pequenos sistemas isolados não se aplicam para projetos de sistemas maiores.*

Fatores como quantidade de pessoas envolvidas, tamanho do projeto e complexidade do domínio fazem com que não se possa estimar o tamanho do projeto apenas com base na quantidade de código.

> *A codificação aumenta como uma potência do tamanho do programa. Algumas publicações mostram um expoente de aproximadamente 1,5.*

Brooks cita um estudo comparando a produtividade, medida em número de comandos codificados, entre dois projetos de tamanhos diferentes. Num projeto maior, a produtividade cai consideravelmente e, basicamente, o esforço aumenta exponencialmente com uma taxa de 1,5 em relação ao número de instruções.

Outro estudo mostrou que programadores gastam apenas 50% do seu tempo codificando e depurando código, o restante sendo preenchido com pequenas tarefas de alta-prioridade não relacionados com o trabalho principal, reuniões, documentação, coisas da empresa, doenças, tempo pessoal, entre outros.

Ainda outro estudo mostrou que o número de interações necessárias entre os programadores reduz drasticamente a produtividade, o que significa que, quanto maior o projeto e maior o número de pessoas, menos cada um produz.

Finalmente, outro estudo realizado durante o desenvolvimento de um compilador, onde diferente equipes com bastante experiência implementaram partes diferentes do produto, mostrou que a produtividade varia grandemente dependendo da complexidade do módulo em que tais equipes trabalharam.

> *Produtividade na programação pode ser aumentada até cinco vezes usando uma linguagem de auto nível.*

Aqui, Brooks cita mais um estudo que mediu o número de linhas de código produzidos numa linguagem de alto nível, em contraste com o número de instruções produzidas em linguagens de baixo nível. Houve um aparente aumento da produtividade próximo de cinco vezes.

Considerações

Se você já participou de algum processo de estimação, já deve ter visto alguém pegar o resultado e arbitrariamente multiplicar por algum fator. É a tal da *gordurinha*. É muito interessante que, isto que muitos fazem instintivamente, "no chute", tem uma base científica.

Os dados fornecidos pelo autor neste capítulo são importantíssimos para entendermos como as tarefas adicionais em um projeto de software realmente influenciam sobre o cronograma e não são exatamente proporcionais ao tamanho do código.

- O que é um sistema legado? (http://luizricardo.org/2013/09/o-que-e-um-sistema-legado/)
- Instalando e Configurando o Java Development Kit 7 para Desenvolvimento (http://luizricardo.org/2013/11/instalando-e-configurando-o-java-development-kit-7-para-desenvolvimento/)
- Morando no interior, trabalhando na capital: experiências das viagens de fretado (http://luizricardo.org/2013/12/morando-no-interior-trabalhando-na-capital-experiencias-das-viagens-de-fretado/)
- Entendendo o Triângulo de Ferro, porque não podemos ter tudo (http://luizricardo.org/2013/09/entendendo-o-triangulo-de-ferro-porque-nao-podemos-ter-tudo/)

DESCUBRA

Selecionar categoria

Selecionar o mês

agile (http://luizricardo.org/tag/agile/)

ambiente de trabalho (http://luizricardo.org/tag/ambiente-de-trabalho/)

apostila (http://luizricardo.org/tag/apostila/)

apresentação (http://luizricardo.org/tag/apresentacao/)

automação (http://luizricardo.org/tag/automacao/)

boas práticas (http://luizricardo.org/tag/boas-praticas/)

bootstrap (http://luizricardo.org/tag/bootstrap/)

css (http://luizricardo.org/tag/css/)

curso (http://luizricardo.org/tag/curso/)

desempenho (http://luizricardo.org/tag/desempenho/)

desenvolvimento profissional (http://luizricardo.org/tag/desenvolvimento-profissional/)

design patterns (http://luizricardo.org/tag/design-patterns/)

dica (http://luizricardo.org/tag/dica/)

eclipse (http://luizricardo.org/tag/eclipse/)

emprego (http://luizricardo.org/tag/emprego/)

engenharia de software (http://luizricardo.org/tag/engenharia-de-software/)

estimação (http://luizricardo.org/tag/estimacao/)

estudo (http://luizricardo.org/tag/estudo/)

estudo de caso (http://luizricardo.org/tag/estudo-de-caso/)

framework web (http://luizricardo.org/tag/framework-web/)

gerenciamento de projetos (http://luizricardo.org/tag/gerenciamento-de-projetos/)

html5 (http://luizricardo.org/tag/html5/)

A regra é simples: quanto maior ou mais complexo o programa, mais sobrecarga, menos produtividade. O "favor de produtividade" ou "gordurinha" não pode ser o mesmo para diferentes tipos de projetos.

Ao mesmo tempo em que não existe uma fórmula mágica para estimar um projeto, conhecer as variáveis que influenciam no mesmo ajuda a determinar uma aproximação mais favorável ao sucesso do projeto.

Um exemplo prático

Para você ter uma ideia, em minha experiência, algumas vezes vi projetos que estimaram o tempo de codificação e multiplicavam isso por 1,5 ou 2 e sempre atrasavam.

Bem, geralmente são os próprios programadores que estimam o tempo de codificação e, como você deve ter lido nos primeiros capítulos, programadores são otimistas. A primeira coisa que o responsável pela estimação deve fazer é confrontar (no bom sentido) os programadores sobre possíveis riscos e problemas, talvez perguntar o tempo que levaria no **pior caso**. Acredite, por padrão, frequentemente programadores dão a estimativa do melhor caso.

Além disso, como você notou neste capítulo, programadores codificam apenas 50% do seu tempo, em média. Então o fator mínimo que você deve adotar para multiplicar a estimativa do tempo de codificação é 2.

Finalmente, deve-se levar em conta a complexidade e riscos do projetos que podem comprometer sua entrega.

Paradigma de programação e nível de abstração

Linguagens de alto nível aumentam a produtividade porque permitem ao programador se concentrar no problema a ser resolvido ao invés de ter que lidar com complexidades desnecessários do ambiente operacional.

O autor demonstra ao longo do livro, como vimos na última citação, uma grande esperança que linguagens modernas venham a aumentar muito a produtividade. Vemos tal sentimento reproduzido hoje quando se fala sobre paradigmas de programação mais avançados, tal como as diferentes aplicações do paradigma funcional.

Assim como linguagens procedurais de alto nível foram um grande avanço com relação, por exemplo, à programação em instruções assembler, onde um comando poderia equivaler a dezenas de instruções de máquina, a programação funcional permite expressar conceitos complexos, equivalentes a muitas linhas procedurais, em poucos caracteres.

Entretanto, há pelo menos dois problemas com a tese do autor, isto é, de que usar uma linguagem de mais alto nível aumenta a produtividade:

1. Medir produtividade baseando-se em linhas de código ou instruções não faz sentido. Existem muitos tipos de código que são fáceis de serem produzidos e não acrescentam funcionalidade real.
2. Sou muito cauteloso ao afirmar que elevar a abstração efetivamente aumenta a produtividade. Uma linguagem que faz muito com poucos comandos, necessariamente possui maior densidade de complexidade, isto é, mais complexidade por cada caractere digitado. Isto necessariamente leva a uma maior curva de aprendizado e codificação mais lenta. Se isto se paga com o tempo ou se a complexidade chega a tal ponto que não se consegue mais evoluir o sistema, é outra discussão. Meu ponto aqui é que não podemos pular rapidamente a conclusões generalizadas.

javascript (http://luizricardo.org/tag/javascript/)
jquery (http://luizricardo.org/tag/jquery/)
maturidade (http://luizricardo.org/tag/maturidade/)
maven (http://luizricardo.org/tag/maven/)
meme (http://luizricardo.org/tag/meme/)
monografia (http://luizricardo.org/tag/monografia/)
mythical man-month (http://luizricardo.org/tag/mythical-man-month/)
oracle (http://luizricardo.org/tag/oracle/)
play! framework (http://luizricardo.org/tag/play-framework/)
programação (http://luizricardo.org/tag/programacao/)
python (http://luizricardo.org/tag/python/)
reflexões (http://luizricardo.org/tag/reflexoes/)
requisitos (http://luizricardo.org/tag/requisitos/)
segurança (http://luizricardo.org/tag/seguranca/)
sql server (http://luizricardo.org/tag/sql-server/)
stackoverflow (http://luizricardo.org/tag/stackoverflow/)
sysdeo (http://luizricardo.org/tag/sysdeo/)
tdd (http://luizricardo.org/tag/tdd/)
testes (http://luizricardo.org/tag/testes/)
tomcat (http://luizricardo.org/tag/tomcat/)
transação (http://luizricardo.org/tag/transacao/)
tutorial (http://luizricardo.org/tag/tutorial/)
ubuntu linux (http://luizricardo.org/tag/ubuntu-linux/)

SOCIAL


(http://github.com/utluiz/) (http://linkedin.com/in/u