

# Dez Anti-Mitos em Estimativa de Projetos

Phillip G. Armour

(Publicado originalmente na Communications of the ACM, vol. 45, n. 11, Nov. 2002, p. 15-18)  
(Traduzido e adaptado em 30 de Dezembro de 2002 por Ricardo Caetano de Moraes)

De acordo com Joseph Campbell um mito não é falso, como poderíamos pensar. Um mito é alguma coisa que é extremamente verdadeira. Ele é a essência da verdade vestida com uma roupagem simbólica que nos ajuda a lembrar sua lição. Se um mito é uma verdade disfarçada de mentira, então alguma coisa que parece perfeitamente razoável mas é essencialmente errada seria um “anti-mito”. Existem alguns “anti-mitos” em gerência e estimativa de projetos. Existem verdades que, quando as olhamos mais de perto, elas não são, bem, *verdadeiras*.

Quando fazemos uma estimativa de projeto, estamos tentando prever o futuro. Historicamente, esta tem sido uma tarefa perigosa. Oráculos e adivinhos têm sido aclamados e honrados no passado. Mas eles também têm sido apedrejados até a morte – se eles se tiverem enganado, e algumas vezes, se estiverem certos. E o mesmo se aplica na predição do tempo e do esforço necessários para projetos de software. Talvez a estimativa de projetos devesse vir acompanhada de um aviso: “O Ministério da Saúde adverte: Estimativa de Projetos pode ser perigosa para sua carreira.” De qualquer forma, como Woody Allen observou certa vez: a única coisa que eu não posso prever com precisão é o futuro.

Existem vários componentes da estimativa que não são muito “corretos”. Eles podem possuir elementos de certeza, um verniz da verdade, mas também merecem um olhar crítico. Isso poderia ajudar a compreender como e porquê podemos às vezes estar tão errados nas nossas estimativas, e talvez apontar o que podemos fazer a respeito.

***O Anti-Mito da Precisão: Nós podemos ter uma “estimativa precisa”.*** Deixando de lado o fato de ser um paradoxo, existe uma razão muito simples pela qual as estimativas não podem ser “precisas” – nós simplesmente *não possuímos os dados* necessários para ser precisos. É um triste fato que, quanto mais cedo uma estimativa for feita, menos dados temos para isso, e, portanto, seremos menos “precisos”. O único momento no qual possuímos dados suficientes para garantir legitimamente a precisão é no final do projeto, quando todas as variáveis estão resolvidas. Infelizmente, ninguém jamais pedirá uma estimativa neste estágio. Podemos, no entanto, ter uma estimativa “feliz”. Isso acontece quando todas as coisas em que não pensamos que aceleram o projeto e todas as coisas em que não pensamos que atrasam o projeto se igualam. Isso é sorte, não precisão, e, uma vez que existem muito mais coisas que atrasam um projeto do que coisas que o aceleram (uma aplicação da Segunda Lei da Termodinâmica aos projetos), quase invariavelmente estaremos subestimando os projetos.

***O Anti-Mito da Data de Término: A tarefa da estimativa é determinar uma data para conclusão.*** Isto pode ser facilmente demonstrado usando o programa SDI (também conhecido como “Guerra nas Estrelas”), recentemente ressuscitado. Qual é a probabilidade de completar este programa, digamos, na próxima quarta-feira? Sem pensar muito, eu diria quase zero. No entanto, poderíamos concluir com razoável segurança que a probabilidade de concluí-lo daqui a 100 anos, contados a partir de quarta-feira, é perto de 100%. Se a probabilidade começa em 0% e atinge 100% um século mais tarde, seu valor deve variar com o tempo. Como ele varia depende do tipo

de projeto. Mais especificamente, ele depende da incerteza (ou da quantidade de conhecimento desconhecido no momento da estimativa, o que é a mesma coisa). Podemos pegar qualquer data que desejarmos entre a próxima quarta-feira até 2102. Cada data possui alguma probabilidade de conclusão, variando de algo infinitesimalmente pequeno até uma grande chance de sucesso. Uma vez que podemos usar qualquer data, a estimativa não nos dá uma data para conclusão, mas sim uma *probabilidade* de conclusão.

***O Anti-Mito do Compromisso: A estimativa e o compromisso são a mesma coisa.*** O Anti-Mito da Data de Término leva a outra premissa comum que é muito perigosa na prática. Trata-se da nossa falha em diferenciar uma estimativa do compromisso efetivo em cumpri-la. A estimativa gera uma probabilidade, não uma data; mas não podemos nos comprometer com uma probabilidade, somente podemos nos comprometer com uma data. O *processo* de comprometimento é o que transforma uma probabilidade em uma data. Utilizando o conhecimento dos recursos disponíveis, objetivos de negócio, retorno do investimento e assim por diante, o processo de comprometimento seleciona uma data que otimiza a probabilidade de uma bom retorno do investimento e permite o gerenciamento do risco. A falta de compreensão disto é que leva grandes empresas a reservar enormes quantidades de dinheiro baseadas em palpites de programadores iniciantes.

***O Anti-Mito do Tamanho: Uma estimativa de projeto é dependente do tamanho do sistema final.*** De certa maneira, isto é uma verdade – mas apenas como uma média bastante grosseira. Uma vez que o desenvolvimento de software é na verdade uma atividade de aquisição de conhecimento, ao invés de uma atividade de produção de código, o tamanho do produto pode ser irrelevante. Se eu puder reutilizar um novo sistema inteiro, o tamanho pode ser grande, mas o esforço será muito pequeno. Da mesma forma, eu posso ter um minúsculo sistema embutido de tempo real que é extremamente complexo e que requer um grande esforço. Isto é tacitamente reconhecido na calibragem de alguns dos mais populares métodos e ferramentas de estimativa – sistemas embutidos de tempo real (que são tipicamente “menores” mas mais complexos) têm “fatores de produtividade” muito menores que sistemas convencionais. É claro, o esforço é dependente da quantidade de conhecimento que têm que ser obtida. O tamanho final do sistema é dependente da quantidade de conhecimento que pode ser reutilizado de algum outro lugar *mais* a quantidade de conhecimento que tem que ser adquirida. É o “conhecimento-a-ser-adquirido” que determina o esforço no projeto.

***O Anti-Mito da História: Os dados históricos são um indicador preciso de produtividade.*** Isto é também um certo tipo de verdade, mas usamos isto principalmente porque não temos nada melhor. O problema com o histórico é que os dados foram coletados em projetos anteriores e diferentes. O projeto que estamos estimando certamente é diferente dos projetos anteriores. Se tivermos conservado o conhecimento dos projetos anteriores, o “conhecimento-a-ser-adquirido” neste projeto é a diferença. Mas é a aquisição deste conhecimento que constitui o verdadeiro esforço. A calibragem da produtividade foi feita a partir de um conjunto de conhecimentos diferente, logo provavelmente não será aplicável corretamente quase por definição. Com sorte, especialmente em projetos grandes, nossa capacidade de adquirir conhecimento que não possuímos tende a ser semelhante, não importando qual conhecimento venha a estar envolvido. Mas isso é apenas uma média, e, é claro, não é verdade se o conhecimento é realmente diferente, ou os desenvolvedores são diferentes, ou o cliente é diferente, ou...

***O Anti-Mito da Produtividade: A produtividade é um indicador preciso da duração do projeto.*** A maioria das medidas de produtividade implica em uma taxa de produção, o

que implica que estamos construindo um produto, o que não é exatamente o que estamos fazendo. Uma vez que muito do desenvolvimento de software envolve obter conhecimento que nós ainda não possuímos, não podemos nem estimar precisamente quanto tempo isso irá levar (veja o Anti-Mito da Data de Término), ou mesmo se iremos consegui-lo (devido à Ignorância de Segunda Ordem (2OI)). Um exemplo simples: nós poderíamos ser maravilhosamente produtivos na criação de módulos de software; nós construímos, testamos e entregamos o sistema, apenas para descobrir que está faltando algum conhecimento do ambiente. Há algum fato chave na organização do cliente, no modelo de negócios ou em alguma outra área que não capturamos. Epa! Lá vem o sistema de volta para ser consertado (ter o novo conhecimento incorporado). Grande atraso. Certamente fomos produtivos sob a perspectiva de linhas de código, mas o projeto ainda levou muito mais tempo do que deveria.

***O Anti-Mito das LOC: Uma contagem das Linhas de Código (LOC) é uma boa maneira de determinar o tamanho de um sistema.*** Nossa tarefa não é produzir LOC, é obter conhecimento; se as linhas de código não contêm o conhecimento “correto”, não importa quantas elas são. Aqui jaz a dificuldade – o tamanho do sistema para fins de estimativa é relacionado à quantidade de conhecimento que tem que ser obtido. Ele também é relacionado à dificuldade de conseguir este conhecimento, o que é assunto para outro artigo. O problema com a medição do tamanho é que estamos tentando estimar uma quantidade de conhecimento e a raça humana simplesmente não encontrou uma maneira de quantificar conhecimento. Depois de pensar a respeito pelos últimos dois mil anos, ainda não conseguimos definir uma “unidade de conhecimento”. As LOC podem ser um bom valor médio, como as médias costumam ser, mas usá-las como uma medida da quantidade de conhecimento é mais ou menos como pesar um livro para saber quanto conhecimento ele contém.

***O Anti-Mito dos Pontos por Função: Pontos por Função são uma boa maneira de determinar o tamanho de um sistema.*** Se as LOC não são uma boa medida de tamanho, o que dizer dos Pontos por Função, em qualquer uma das suas numerosas formas? Muitas das mesmas questões se aplicam, é claro. Os Pontos por Função têm algumas vantagens e algumas desvantagens em relação às LOC. A maioria dos métodos de contagem dos Pontos por Função não são tão diretamente mensuráveis no sistema quanto as contagens de LOC, embora elas possam ser normalmente medidas a partir de alguma representação do sistema. No lado positivo, a abordagem de Pontos por Função conta elementos, tais como entradas e saídas de um sistema, que podemos relacionar intuitivamente ao seu tamanho, e presumivelmente ao conhecimento representado. Isto significa que, se aumentarmos o número e a complexidade das entradas, iremos aumentar proporcionalmente o tamanho do sistema e a quantidade de conhecimento que ele contém.

Infelizmente, existem muitos aspectos da complexidade do sistema (quantidade de conhecimento) que não são relacionados diretamente (ou sequer se relacionam) aos itens contabilizados pelos Pontos por Função. Por exemplo: a complexidade de um sistema de telecomunicações pode ser quase inteiramente dissociada das suas entradas e saídas. Uma rica variedade das fórmulas de Pontos por Função têm surgido para tentar resolver essas questões, mas eles sofrem do mesmo problema que importunam as LOC – eles também não são “unidades de conhecimento”.

***O Anti-Mito de Mais Pessoas: Podemos fazer o sistema mais rapidamente alocando mais recursos a ele.*** Ao que parece, esta é uma lição que temos que aprender continuamente. Embora algumas fórmulas e ferramentas de estimativas mostrem uma grande dependência do tempo e dos recursos na data de entrega, aumentar a equipe para

executar os projetos mais rapidamente ainda é o recurso de gerenciamento mais utilizado. Uma vez que o desenvolvimento de software é uma atividade de aquisição de conhecimento, há muitos problemas envolvidos. O primeiro é que existe um limite para a velocidade da aquisição de conhecimentos pelas pessoas – ela não pode ser aumentada além desse limite. O segundo é que, como o conhecimento é intrinsecamente conectado, desmembrá-lo e repartir os pedaços entre pessoas diferentes não permite a visualização de todo o quadro.

Costumam ocorrer revelações infelizes nas fases de integração e testes de tais projetos. Além disso, há outros problemas: uma equipe maior produzirá muito mais canais de comunicação; podemos compor a equipe de um projeto com pessoas que têm muito mais a aprender (isto é, são menos experientes); o ritmo de um projeto pode fazer com que pensemos conhecer alguma coisa sem realmente validar esta suposição – o que a realidade irá fazer por nós, é claro, assim que liberarmos o sistema.

***O Anti-Mito do Defeito Zero: Se tivermos tempo bastante, podemos criar um sistema sem nenhum defeito.*** Ou bastante experiência, bastante poder de processamento, o processo correto, e assim por diante. Defeito Zero é quase uma meta Zen. Mesmo compreendendo que não podemos consegui-lo, é o único objetivo de qualidade pura. Não podemos consegui-lo pela mesma razão que não podemos conseguir a iluminação plena – não sabemos o que é isso. Um defeito é uma falta de conhecimento, é um mal-entendido, é alguma coisa que não sabemos. A falta de conhecimento se transforma em um defeito quando ela se manifesta através do mundo real (ou alguma derivação do mundo real, tal como os testes). Eu chamo este momento quando o erro é detectado de *epifania do conhecimento*. É o momento quando o mundo real nos avisa que existe alguma coisa aqui que não conhecemos. A única maneira pela qual poderíamos ter verdadeiramente um sistema sem nenhum defeito seria rodar o sistema contra todos os conjuntos imagináveis de variáveis. Não apenas o conjunto de todas as possíveis entradas é muito grande, mas também temos um paradoxo relacionado à palavra *imagináveis*. Somente podemos inspecionar e testar coisas que sabemos, ou que pelo menos sabemos que não sabemos. Não podemos testar adequadamente coisas que não sabemos que não sabemos (2OI). Infelizmente, é a 2OI que é o conhecimento verdadeiramente valioso, e por conseguinte o propósito real do projeto.

## **Desmitificando os Anti-Mitos**

***Precisão.*** As estimativas não têm que ser precisas – elas têm que ser precisas o suficiente para se encaixar nos graus de liberdade do nosso negócio. Isto é, o processo de estimativa tem apenas que produzir estimativas que nos permitam evitar decisões realmente ruins, das quais não possamos nos recuperar. Bem, pelo menos a maior parte do tempo.

***Data de Término.*** Podemos definir as saídas das estimativas em termos de uma *faixa de probabilidades* associada às datas (e outros fatores de recursos) para ajudar nisto..

***Compromisso.*** Estabeleça processos com estágios discretos de estimativa e compromisso, com entradas e saídas definidas.

***Tamanho.*** Não há um remédio real para isto, mas manter um histórico de faixas de tamanho poderia ajudar a compreender a variância provável. Precisamos realmente executar uma validação do conhecimento (ou melhor, da falta de conhecimento) ao invés de um cálculo de tamanho.

**História.** Isto também é um tanto intratável, mas ter um processo de estimativa que aproxime o histórico da situação corrente ajudaria. Muitas ferramentas de estimativa fazem um bom trabalho neste sentido.

**Produtividade.** Também é um problema constante. Ajuda usar a produtividade histórica como um indicador ao invés de um previsor explícito.

**LOC.** Invente uma unidade real de conhecimento que pode ser medida empiricamente, que possa ser calibrada em relação ao ambiente e ao entendimento individual e coletivo das pessoas, e que também possa ser usada para avaliar a falta de conhecimento evidente, projetada ou assumida. (Boa sorte, e por favor me avise quando você terminar.)

**Pontos por Função.** Idem, embora ajude usar esta técnica apenas onde existe um forte relacionamento entre os tipos de conhecimento e as unidades medidas pelos Pontos por Função.

**Mais Pessoas.** Isso somente pode funcionar onde existe uma separação bem definida do conhecimento em funções, apoiada pela arquitetura do produto, a metodologia de desenvolvimento, a base de conhecimento coletiva das pessoas, os métodos de aquisição de conhecimento, e a gerência de projeto e atribuição de tarefas. De outra forma simplesmente não funciona.

**Defeito Zero.** Qualidade boa o bastante? Isto é o que sempre é buscado, mesmo se nós não desejamos admitir isso (e provavelmente não deveríamos trabalhar intencionalmente para isso). O conhecimento no software somente tem uma função – entregar valor ao cliente. Para permitir ao cliente fazer coisas que ele não poderia fazer sem o software; permitir ao usuário acessar o conhecimento que ele(a) não poderia ter de outra maneira. Quando o valor entregue supera o custo do uso (incluindo uma compensação pelos defeitos) acima de um certo mínimo, o software é valioso para o usuário. Esta forma de valor é o único critério que realmente interessa.

No fim, uma estimativa é somente uma estimativa, não é exata. Afinal, o processo é chamado estimativa, não exatimativa.

---

**PHILLIP G. ARMOUR** ([armour@corvusintl.com](mailto:armour@corvusintl.com)) é vice-presidente e consultor sênior na Corvus International Inc., Deer Park, IL.

---

\*  
\*\*