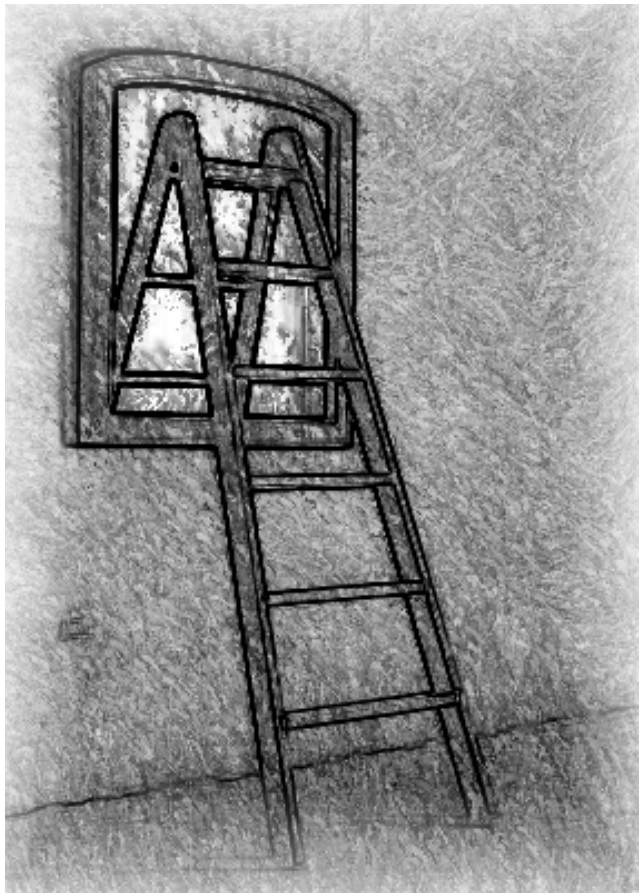
De Brooks a Berkun



Um estudo informal dos livros
"**The Mythical Man-Month**" e
"**The Art of Project Management**",
e uma homenagem aos seus autores,
Fred Brooks e Scott Berkun.

Paulo Vasconcellos
Abril/2006

Prólogo

Em 2005 comemorou-se 30 anos do lançamento da primeira edição de "The Mythical Man-Month", de Frederick P. Brooks Jr. O livro pode ser considerado o primeiro clássico das áreas de Engenharia de Software e Gerenciamento de Projetos. Fred Brooks trabalhou por 8 anos na IBM, entre 1956 e 1964. Seu último ano foi dedicado ao desenvolvimento do OS/360, um empreendimento que envolveu mais de 5 mil homens/ano. Surge então a provocação que o levaria ao livro, uma 'pergunta básica' de Thomas Watson (CEO da IBM):

"Por que programação é tão difícil de ser gerenciada?"

Três décadas de avanços tecnológicos e de consolidação das ciências Engenharia de Software e Gerenciamento de Projetos não foram suficientes para tornar o texto obsoleto. Muito pelo contrário: até hoje o livro é considerado uma referência obrigatória. Em 2004, quando questionado sobre a razão da longevidade de sua obra-prima Fred Brooks respondeu: **"Eles falam que o livro é a Bíblia da Engenharia de Software... é por isso que todo mundo o lê mas ninguém o usa!"**.

No ano em que comemorou seu trigésimo aniversário, "The Mythical Man-Month" ganhou um tipo de *upgrade*, de complemento. Trata-se de "The Art of Project Management", de Scott Berkun. Não é só uma coincidência de temas, afinal existem milhares de títulos sobre Gestão de Projetos e Engenharia de Software. Mas vários outros aspectos aproximam as duas obras. Scott Berkun trabalhou por dez anos na Microsoft, em projetos como *Windows*, *MSN* e *Internet Explorer*. Ou seja, Berkun reviveu experiências parecidas com aquelas de Brooks em uma corporação de porte e relevância quase idênticas.

Não é por acaso que os dois livros possuem linguagem e estrutura muito parecidos. Podem ser classificados como ‘histórias de guerras’. Ambos são muito fáceis de ler e seguem uma ordem própria, em detrimento de padrões, nomenclaturas e sequências que caracterizam 9 em 10 títulos sobre gerenciamento de projetos lançados nos últimos tempos. Este artigo foi concebido originalmente para comemorar os 30 anos de "The Mythical Man-Month". Compará-lo ao recém lançado "The Art of Project Management" é só uma maneira de tornar a homenagem e, por que não dizer, as provocações, um pouco mais ricas.

Por incrível que pareça, nem "The Mythical Man-Month" nem "The Art of Project Management" mereceram uma edição em português do Brasil. Um lapso que ainda pode ser corrigido. E deveria. Como são *best sellers*, mesmo por aqui, razão comercial não há. Como Gestão de Projetos está na moda, resta-nos torcer para que, no meio daquele tanto de livro lançado para ‘ajudar a gente a passar na prova’, apareçam mais títulos como os de Brooks e Berkun.

Este artigo não deve ser visto, de forma alguma, como uma alternativa aos textos originais. Ele deve servir como um incentivo para a leitura de ambos. Ou releitura, por que não? Em 1995, foi publicada uma edição de "The Mysthical Man-Month", comemorando os 20 anos da obra. Ela traz uma série de artigos e capítulos complementares. Mas o texto original, como era de se esperar, foi mantido.

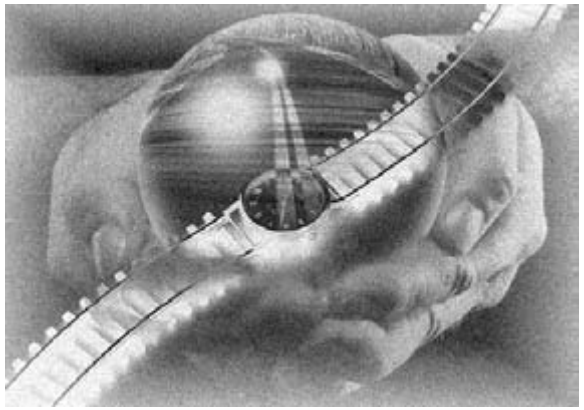
Que a experiência seja agradável e útil.

Índice

"Seres humanos, que são quase únicos em sua capacidade de aprender com as experiências dos outros, também se caracterizam por sua resistência em fazê-lo."

Douglas Adams

Entre o Relógio e a Bola de Cristal



No segundo capítulo de "The Mythical Man-Month", homônimo, Fred Brooks mostra um fac-símile do cardápio do *Restaurant Antoine*, de New Orleans. Destaca um *Avis au Public*, que aparece logo no cabeçalho do *menu*:

"Good cooking takes time. If you era made to wait, it is to serve you better, and to please you."

Brooks apresenta cinco razões para nossos constantes e intermináveis atrasos. O primeiro é o incurável otimismo de programadores e afins. Ele diz que nossas técnicas de estimativas são pobres e refletem uma situação irreal: de que tudo vai dar certo. Esse mesmo otimismo nos faria negligenciar, particularmente, a fase de testes de um projeto.

O segundo motivo seria a confusão entre "Esforço" e "Progresso". Tal confusão nasceria da falsa crença de que Homens e Meses (Horas) são intercambiáveis. Brooks reforça que só conseguimos trocar alguns 'meses' por um monte de 'gente' em um conjunto de atividades que não exijam nenhum tipo de interação entre as pessoas. Acho que nem em panha de café existe tal isolamento...

A terceira razão relacionada por Brooks para os atrasos em nossos projetos é chata e provocativa: nós não somos sinceros como o *chef* do Antoine. A facilidade com que damos 'chutes' e apresentamos cronogramas sem um mínimo de embasamento é realmente assustadora. Pior: apresentando-os como 'sérios'. E a responsabilidade é tanto de quem pede quanto de quem dá. Já vi empresa muito grande solicitar uma proposta para um projeto de milhões apresentando um documento de 7 (sete!) páginas e fazendo uma reuniãozinha com os possíveis contratados. Tamanha negligência em qualquer outra área deve ser caso de polícia. Na nossa parece 'normal'.

A forma como acompanhamos e monitoramos a evolução de nossos cronogramas seria a quarta razão. A impressão que grande parte dos projetos nos transmite é que, assim que eles ganham ritmo, perde-se o controle.

> **"Adicionar pessoas a um projeto de software atrasado só adiará a sua entrega."**
> **- A Lei de Brooks**

Por último Brooks cita sua lei como outra grande razão para nossos atrasos. Segundo ele, adicionar pessoas a um projeto atrasado é como "tentar apagar um incêndio com gasolina". Cada novo membro pode significar uma nova cadeia de interações e restrições. A quantidade de meses de um projeto dependeria de suas restrições sequenciais. O número máximo de pessoas dependeria do número de sub-tarefas independentes. Podemos elaborar um cronograma mais 'demorado' utilizando menos pessoas. Mas não podemos encurtar seu prazo natural adicionando mãos!

Recentemente Scott Berkun publicou uma série de 'exceções' à Lei de Brooks. Alertando que não queria, de maneira alguma, sugerir o abandono da lei. O próprio Brooks já reconheceu que sua lei é "ultrajantemente super-simplificada". Abaixo as exceções apontadas por Berkun:

- **Depende da pessoa**: lógico! O acréscimo de uma ou mais pessoas mais experientes que aquelas da equipe atual pode, eu disse Pode, gerar um efeito positivo no projeto.
- **Alguns times podem assumir mudanças mais facilmente**: acho que depende inclusive de um entrosamento prévio dos novos integrantes com os antigos. Um entrosamento proveniente de projetos anteriores. Mas depende principalmente das outras ações do coordenador (último tópico abaixo).
- **Há coisas piores que estar atrasado**: com certeza, como entregar algo totalmente diferente daquilo que o cliente espera. Berkun lembra bem: "a única coisa que a lei fala é que o projeto atrasará 'mais' com a adição de mais pessoas". Mas tal acréscimo pode ser benéfico ou crucial para o resultado final do projeto.
- **Há diferentes maneiras de adicionar pessoas a um projeto**: mas Berkun lembra bem que nossa tradição é “jogar gente lá para ver no que é que dá”. Se bem pensada, a adição (ou troca) de pessoas em um time pode ser benéfica.
- **Depende da razão do atraso do projeto, para começo de conversa**: claro! Se o(s) motivo(s) não for(em) bem identificado(s), de que adianta jogar mais pessoas na frigideira?
- **A adição de pessoas pode ser combinada com outras ações do coordenador**: pode não, deve. No mínimo ações que otimizem (minimizem) o número de interações e facilitem a rápida absorção do projeto pelo(s) novo(s) integrante(s).

Cronogramas são só um Tipo de Previsão

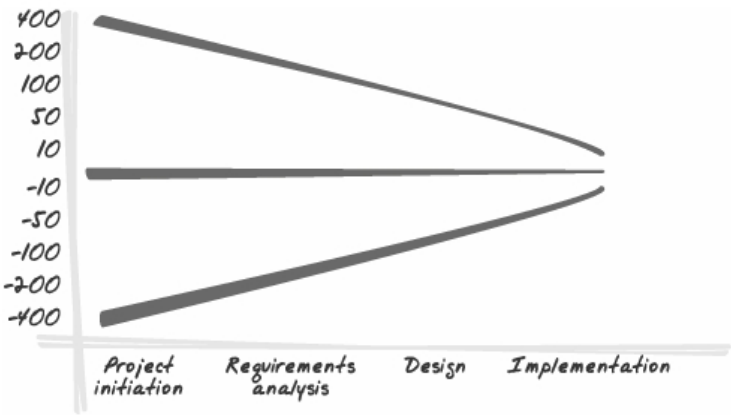
Mas há quem os trate como se fossem documentos sagrados. Para Scott Berkun os cronogramas "não são remédio para um projeto (*design*) ou práticas de engenharia ruins e também não podem proteger um projeto de uma liderança fraca, objetivos obscuros e comunicação pobre". Independentemente do tempo gasto e do capricho com que um cronograma foi elaborado, "no final das contas eles são apenas uma lista de palavras e números". Uma lista que, segundo Berkun, deve ter três objetivos bem claros:

- Descrever quais tarefas devem ser executadas, quando e quais produtos serão gerados;
- Funcionar como um integrador da equipe, indicando dependências e amplificando os compromissos mútuos; e
- Fornecer para o time uma ferramenta que possibilite o acompanhamento da evolução do projeto e que permita estruturá-lo em etapas mais gerenciáveis.

E não é que tem cliente que, inconscientemente (quero acreditar), prefere ver um cronograma atualizado do que software rodando? Querendo ou não, os cronogramas viraram o 'grande' documento de um projeto de software. Responsabilidade demais para algo que deveria ser só mais uma ferramenta. Pior: para algo que deveria evoluir junto com o projeto.

> **"A elaboração do melhor cronograma, usando as mais capacitadas pessoas e as melhores ferramentas, também será uma tentativa de prever o futuro. Algo que nossa espécie raramente faz bem."**
> **- Scott Berkun**

É impossível gerar um cronograma com um mínimo de acuracidade no momento inicial de um projeto. Berkun afirma e milhares de projetos confirmam esta 'lei'. Mas continuamos insistindo. Berkun gerou o gráfico ao lado a partir de "Software Engineering Economics" [1], de Barry Boehm. Nos momentos iniciais de um projeto nosso 'chute' pode ter uma variação de 400%. Há quem arrisque 4 dígitos...



Berkun afirma que a volatilidade dos requisitos, dentre outros fatores, pode gerar estimativas de baixa qualidade. E que não há nada de errado com elas desde que sejam apresentadas como tal: Estimativas de Baixa Qualidade. E sugere a adoção de Níveis de Confiabilidade para nossas estimativas:

- 40% - é só um chute;
- 70% - uma boa estimativa;
- 90% - estudamos e detalhamos (quase) tudo.

Uma boa estimativa dependeria principalmente de dois fatores: bons requisitos e um bom *design* (temas que serão tratados posteriormente).E Berkun ainda pede: confie em seus programadores como você confiaria em um neuro-cirurgião. Se este falar que aquela operação no cérebro demorará 12 horas, você tem coragem de pedir para que ele reduza para, digamos, 4 horas? Você seria o louco o suficiente?

Questão de Confiança

Berkun diz que o problema com nossos projetos não está nos cronogramas mas sim na forma como eles são elaborados e usados. O coordenador deve confiar nas estimativas apresentadas pelos programadores e demais membros do time. Se cada estimativa apresentada for bem justificada, não há porque desconfiar delas. Uma questão de validação: quão prováveis são os prazos fixados? "Se nenhuma probabilidade é oferecida e nenhuma pré-condição é colocada, então a realização do cronograma pode até ser possível, mas não é provável".

Que base de comparação, quais referências nós possuímos para avaliar corretamente as estimativas apresentadas? Tanto Brooks (em 75!) quanto Berkun insistem: só o domínio do nosso histórico, tanto de equipes quanto dos indivíduos, permitirá um julgamento justo. Qual a nossa produtividade, nossa 'performance histórica'? Qual o índice de incidência de *bugs*? Existem estimativas para módulos/projetos semelhantes? Como elas foram elaboradas e quais fatores elas consideraram? Não há mágica!

Por outro lado, os programadores devem confiar no plano, no cronograma apresentado. Como? Entendendo a lógica que o criou.

O tema me faz lembrar duas provocações daquelas, feitas por Watts Humphrey:

"Por que **profissionais competentes** concordam com cronogramas quando não têm a **menor idéia** sobre como irão cumpri-los?"

"Por que **executivos racionais** aceitam tais cronogramas quando os engenheiros não oferecem a **menor evidência** de que poderão respeitá-los?"

Berkun fecha o tema apresentando uma série de pequenas dicas muito úteis:

- A duração de uma iteração deve ser coerente com a volatilidade do projeto. Quanto mais volátil, menor deve ser a duração da iteração;
- Devemos ser otimistas na elaboração do Documento de Visão (que será apresentado posteriormente) e pessimistas no cronograma;
- Devemos apostar no *Design*;
- E planejar pontos em que as alterações de escopo serão permitidas;
- Tornar pública a 'filosofia' do Plano – Cronograma;
- Considerar a experiência da equipe no tipo de projeto que estamos tratando;
- Assim como seu entrosamento;
- E antecipar os riscos. Sempre! (o 'Sempre' foi meu).

Só então, estabelecido um compromisso entre todos aqueles que se envolverão diretamente no desenvolvimento do projeto, é que o cronograma deveria ser negociado com o cliente. Choque de realidade: muitas equipes são estruturadas após o fechamento do negócio. É triste, mas temo que não seja uma exceção.

Não é difícil entender o 'outro lado'. Normalmente, quando um projeto sai da área de negócios para aquisição, via departamento de TI, ele já está atrasado. Já é 'para ontem'. Normal...

... O problema começa quando a aquisição é fechada, o cronograma é apresentado desprovido "da menor evidência de que poderá ser cumprido".

Aos poucos estamos aprendendo que a Aquisição Progressiva é uma alternativa muito superior para contratações de projetos de software. Em linhas gerais: um projeto é fatiado em fases (normalmente todos já são); e as partes negociam apenas a fase imediatamente seguinte, aquela que apresenta o menor grau de incerteza. As partes começam com um grande número e um cronograma 'genérico', e concordam em refiná-lo no decorrer do projeto. O contratante pode optar por abrir uma nova concorrência a cada iteração/fase, mostrando independência e, principalmente, muita maturidade (em seus processos de desenvolvimento e aquisição de sistemas).

Cronogramas: Um Meta-Modelo

No texto original de "The Mythical Man-Month" Fred Brooks apresenta um ‘meta-modelo’ que deveria guiar a construção de todos os cronogramas. As regrinhas:

- 1/3 – Planejamento
- 1/6 – Codificação
- 1/4 – Testes individuais
- 1/4 – Testes de integração

No capítulo 19 da edição especial do livro, "... *after 20 years*", Brooks admite que seu modelo é muito *waterfall* (cascata). É, mas também pode ser adaptado para modelos de ciclo de vida mais modernos, iterativos (cíclicos) e incrementais. Mas... que luxo! Gastar 33% do tempo do projeto só em em planejamento!! E 50% do tempo do projeto em testes!?!

Scott Berkun não deixa por menos e nos apresenta a "regra dos terços":

- 30% - Design
- 30% - Programação
- 30% - Testes

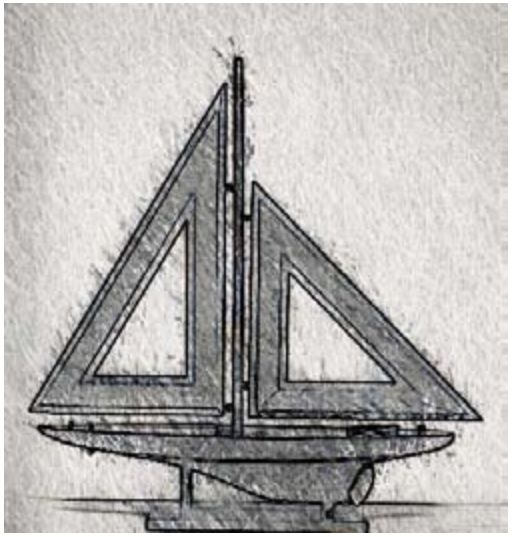
Provavelmente gastamos os 10% que restam tentando justificar o cronograma... Brincadeirinha...
A simplicidade e objetividade das duas sugestões acima assustam. Mas, pense um pouco: Elas são válidas!

Ambos começam concordando que devemos consumir cerca de 30% de todo o tempo do projeto apenas em seu planejamento e arquitetura. Isso não significa que, em um projeto de 9 meses, os três primeiros serão consumidos exclusivamente em atividades de planejamento e *design*. Em um processo de desenvolvimento mais moderno você planeja cada iteração. Mas se trata de um número justo. Eu diria generoso, se considerarmos diversos de nossos projetos.

Berkun é também um desenvolvedor. Brooks nunca foi. Talvez isso explique o fato de Berkun dar o dobro de tempo para as atividades de codificação. Um pouco mais de 15%, como sugerido por Brooks, realmente é muito pouco. Mesmo com todos os *frameworks* e geradores de código 'da vida'.

Por fim temos as atividades tão ignoradas em tantos cronogramas: os famigerados Testes. Brooks pesa a mão e destina 50% de um cronograma exclusivamente para eles. Berkun se contenta com 30%. (Por favor, tentem esquecer por alguns instantes que ele trabalhou na MS, no projeto Windows, ok?).

Régua, Esquadro e Bom Senso

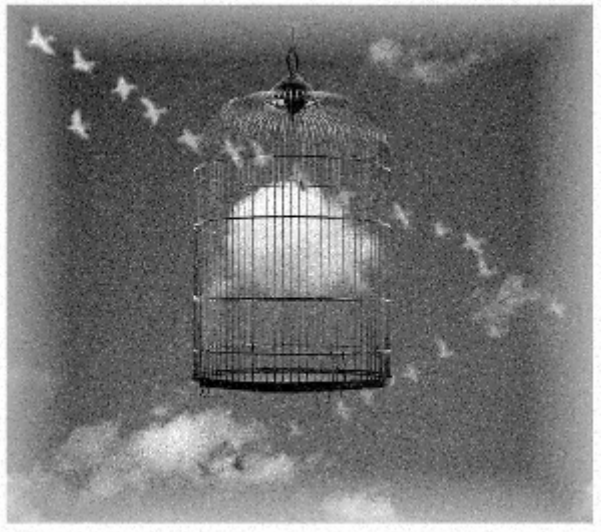


São os três elementos que devem existir entre o relógio-cronograma e a bola de cristal que apóia nossas estimativas. A Régua é nosso histórico de métricas, nossos índices de produtividade e coisas do tipo. Concordo que a máxima "não se gerencia o que não se mede" não se aplica totalmente em nossa área. Mas ignorar nossos números, definitivamente, não ajudará em nada.

O Esquadro deve representar nossas ferramentas de apoio e ajuste. Se estamos em uma fase inicial do projeto, talvez os *Use-Case Points* sejam úteis. Já temos informações suficientes para municiar estimativas baseadas em Análise por Pontos de Função? Lancemos mão dela! Desde que não ignoremos o que a Régua nos ensinou.

Por fim o mais importante: o tal Bom Senso. Na boca de muitos e em tão poucos projetos! O cliente não forneceu informações suficientes para uma boa estimativa? Então seja sincero e diga para ele que a estimativa apresentada é de 'baixa qualidade'. Você suspeita que os requisitos são muito voláteis? Por que não sugerir um contrato de Aquisição Progressiva? Você não tem uma mínima equipe apoiando-o na elaboração das estimativas? Cobre o chefe. Ou contrate a mãe Dinah, sei lá...

Como montar Times e influenciar Projetos



A experiência prática de Fred Brooks, como citado anteriormente, foi com projetos mastodônticos: 1000 pessoas envolvidas ou mais. Mas ele lembra que desde aquela época os gerentes de programação preferiam "pequenos e 'agudos' times formados por pessoas de 'primeira classe'". Brooks cita um estudo (de Sackman, Erikson e Grant [2]) que mostra que um programador de 'primeira classe', que recebia US$20.000/ano, podia ser até 10 vezes mais produtivo que um programador de US$10.000/ano*. Mas um pequeno grupo de 'estrelas' seria capaz de desenvolver um OS/360? Talvez em 10 ou 25 anos, segundo cálculos do autor.

Por outro lado Brooks lembra que times grandes, orientados para a execução de um trabalho na base da 'força bruta', são "lentos, caros, ineficientes, e produzem sistemas que não possuem 'integridade arquitetônica'. OS/360, Exec 8, Scope 6600, Multics, TSS, SAGE, etc. A lista é longa". E "o dilema é cruel", escreve Brooks. Haveria solução?

* Curiosidade: em 30 anos o salário anual de um bom engenheiro saltou de US$20 mil para US$ 80 mil nos EUA. O cafezinho do *Antoine* custava US$0,20. Hoje é vendido por US$2,75!

A sugestão de Brooks, baseada em uma proposta de Harlan Mills [3], dá título para o terceiro capítulo do seu livro, "O Time Cirúrgico". Segundo ele, o time ideal é formado por:

- Programador Chefe (Cirurgião)
- Co-Piloto
- Administrador
- Editor
- Secretárias (duas)
- Bibliotecário
- Almoxarife
- Testador
- Advogado (da Linguagem)

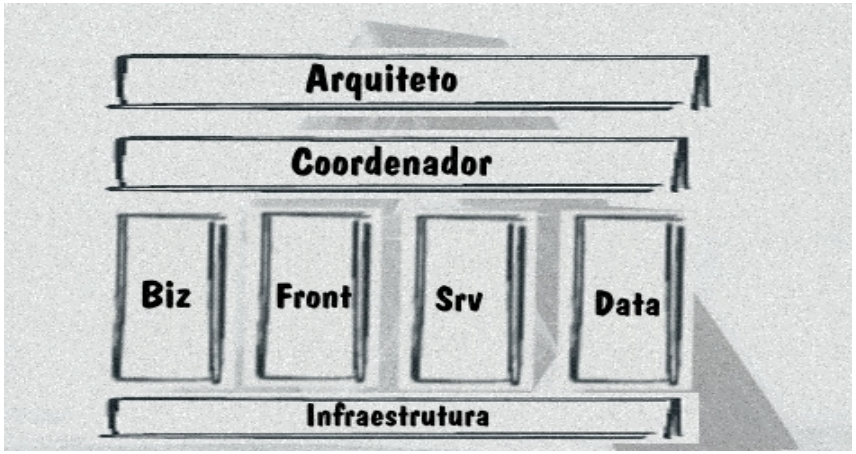
Reparem bem. Nós temos praticamente 9 pessoas trabalhando para o programador! Dando-lhe "todo suporte que fará aumentar sua eficácia e produtividade". Lembram-se que o Brooks também sugeria que alocássemos apenas 1/6 de nosso cronograma para tarefas de codificação? Outro mundo, não é mesmo? Pode e deve ser factível em um time cirúrgico de verdade mas aplica-se a equipes montadas para o desenvolvimento de sistemas?

O 'cirurgião' seria responsável pela execução de todas as tarefas principais daquela parte do projeto: sua especificação, *design*, codificação, testes e documentação. Não difere muito de alguns *job descriptions* e anúncios de vagas que ainda vemos por aí. Seria um "analista-programador" que, segundo Brooks, deveria ter 10 anos de experiência.

O mundo da programação mudou muito desde os tempos do COBOL e da PL/I. A complexidade e abrangência de nossas linguagens e arquiteturas de aplicações aumentaram 'para os lados e para baixo'. E é cada vez mais comum que cada uma das tarefas listadas acima seja atribuída a um especialista. Uma divisão que é nítida em fábricas de software, por exemplo. Em caminho oposto, alguns advogados de métodos ágeis enaltecem a importância de um time formado por 'generalistas'.

A analogia com equipes médicas reaparece em um artigo de Peter Drucker publicado pela Harvard Business Review em 1988, "O Advento da Nova Organização" [4]. Segundo Drucker, "informação é dado investido de relevância e propósito. Por conseguinte, a conversão de dados em informação requer conhecimento. E conhecimento, por definição, é especializado. (Com efeito, as pessoas realmente detentoras de conhecimentos tendem ao excesso de especialização, qualquer que seja seu campo de atuação, exatamente porque sempre se deparam com muito mais a aprender)." Apesar de ser simpático aos chamados métodos ágeis, não acredito na possibilidade ou eficácia de um time composto por generalistas. Prefiro apostar em uma formação que valorize o perfil e a experiência de cada um de seus membros. No diagrama abaixo apresento um exemplo de um time estruturado por especializações:

O **arquiteto** é o novo 'cirurgião'. É o dono e principal responsável por aquele projeto ou módulo. Mas só coloca 'a mão na massa', codificando, para transferir conhecimentos. Ocupa-se com a concepção e manutenção da integridade arquitetônica da solução.



Ele é diretamente apoiado por cinco especialistas:

- **Analista de Negócios** (biz): cuida da coleta e organização dos requisitos, e apóia seu desenvolvimento, fazendo a ponte entre os usuários e a equipe de desenvolvimento;
- **Desenvolvedor de Interfaces** (front): é especialista em usabilidade e 'manda bem' em conceitos de arquitetura de informações. Hoje deve 'brincar' com AJAX (*Javascript*), *Flash*, HTML, CSS, ASP, JSP, JSF e afins.
- **Desenvolvedor de Serviços** (srv): domina orientação a objetos, componentização e, mais recentemente, serviços. É um programador clássico que, como tal, não leva o menor jeito com as 'frescuras da camada de apresentação'.
- **Arquiteto de Informações** (data): uma versão remoçada dos DBA's (administradores de bancos de dados). Domina o desenho e desenvolvimento de bases tradicionais, transacionais, mas também sabe quando lançar mão de sistemas gerenciadores de conteúdo e bases analíticas.
- **Nosso antigo inimigo** (infraestrutura): são os responsáveis por nossos servidores, redes, *firewalls* e outros brinquedinhos. Tê-los como parte da equipe de projeto desde os primeiros dias é uma prática que, com certeza, minimiza aquele 'jogo de empurra' que costuma acontecer um pouco antes ou um pouco depois do delivery da solução.

Mas... e o coordenador? No próximo sub-capítulo falo especificamente sobre ele e sua convivência com o arquiteto.

Agora, seguindo com o time cirúrgico proposto por Fred Brooks. O co-piloto que ele sugere é o 'alter ego' do cirurgião, capaz de executar qualquer tarefa atribuída a este. A única diferença é que o co-piloto seria menos experiente. Mas, ainda assim, funcionaria como um *backup* do cirurgião, inclusive representando-o em reuniões com outras equipes. Porém sua principal função é avaliar e discutir as idéias do programador-chefe. Lembra (vagamente) uma das práticas sugeridas pelo método conhecido como *eXtreme Programming* (XisPê - para os íntimos), a *Pair Programming*.

O 'Administrador' sugerido por Brooks é um tipo de Coordenador do Projeto. Apesar do cirurgião ter a palavra final em todas as questões, é o administrador que cuida do dia-a-dia da gestão financeira, de pessoal, suprimentos etc. Mais sobre o assunto no sub-capítulo seguinte.

O 'Editor' é o responsável pela documentação. O cirurgião, segundo a proposta de Brooks, gera os documentos principais, tanto os técnicos quanto aqueles para os usuários finais. Mas seria o editor o responsável pela compilação final, anexação de referências, bibliografia etc. É um luxo. Porém, ainda hoje brigamos para obter bons documentos de bons programadores, inutilmente.

O 'Bibliotecário' - 'escriturário', ou *program clerk*, como sugerido por Brooks, parece não fazer muito sentido nos dias de hoje. Mas parecia ser crucial nos tempos dos cartões perfurados e das intermináveis listas impressas em formulários contínuos. No entanto eu acredito que toda organização que esteja

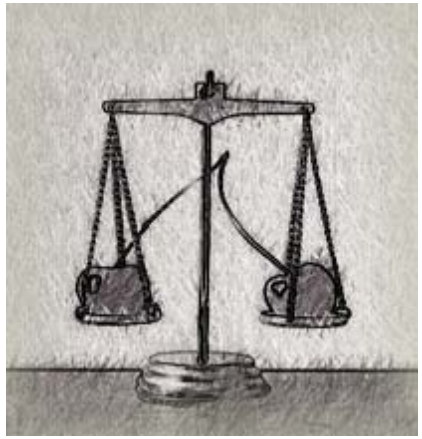
buscando o reuso de seus ativos de software, ou implantando uma SOA (Arquitetura Orientada a Serviços), demandará a presença de um bibliotecário, um especialista que em outro artigo eu chamei de GBA (Gestor da Biblioteca de Ativos).

Se prestarmos atenção na complexidade dos atuais ambientes integrados de desenvolvimento (IDE's – *Integrated Development Environment*), com seus *frameworks*, testes automáticos, integração com ferramentas de modelagem etc, veremos que pode fazer sentido o papel do 'Almoxarife', ou *toolsmith*, na nomenclatura original utilizada por Brooks. Um profissional especializado nas ferramentas e na sua adequação para cada tipo de projeto e função. Em equipes grandes ou em 'fábricas', parece ser uma função de grande utilidade.

O 'Testador' é o nosso "Engenheiro de Testes", uma função que aos poucos vai se tornando mais comum. Pena que muitos ainda o confundam com um 'programador que não deu certo' ou com programadores iniciantes. Teste é coisa séria, tanto que Brooks, como mostramos na 1ª parte do artigo, dedicaria 50% do esforço de um projeto exclusivamente para ele.

Por fim Brooks sugere a alocação de um 'Advogado da Linguagem'. Creio que nossos espertíssimos e modernos IDE's, nossos 'testadores' e, lógico, o arquiteto, eliminam a necessidade de um *'language lawyer'*. Advogado não, né?

Os Dois Donos do Projeto



Fred Brooks encerra "O Time Cirúrgico", terceiro capítulo de "The Mythical Man-Month", falando que um sistema deve ter total Integridade Conceitual, e que isso só seria possível através da dedicação integral de um Arquiteto (*System Architect*, no texto original). Logo depois, em "Por que a Torre de Babel falhou?", ele fala de dois líderes em um projeto: o Arquiteto (ou Diretor Técnico) e o Produtor. Mas o dito popular não ensina que 'Totó com dois donos morre de fome'? Como um projeto pode ter dois líderes e não parecer uma organização confusa?

Segundo Brooks, o Produtor é o cara que monta o time, divide as tarefas e elabora o cronograma. Também é ele quem cuida da aquisição de recursos durante todo o projeto. Portanto, na maior parte do tempo, o Produtor estaria interagindo com entidades externas ao projeto. Ainda assim, é ele quem estabelece padrões de comunicação e relatórios com o time.

Já o Arquiteto (ou Diretor Técnico) é responsável pelo desenho do sistema a ser construído, seus módulos e também seu aspecto exterior. Interage principalmente com o time, resolvendo questões técnicas.

Considerando que os dois perfis são necessários em projetos de qualquer porte, Brooks aponta três relacionamentos possíveis entre eles:

- **Arquiteto e Produtor são a mesma pessoa**: possível em pequenos projetos (para Brooks, de 3 a 6 programadores). Mas uma combinação arriscada em projetos maiores. Brooks justifica: "Pensadores são raros; executores são raros; pensadores-executores são raríssimos".
- **O Produtor é o chefe e o Arquiteto o seu braço direito**: a dificuldade aqui, segundo Brooks, é o estabelecimento da autoridade do Produtor em questões técnicas. Ele diz que o entrosamento entre o Produtor e o Arquiteto é fundamental. E que o primeiro deve respeitar muito o valor do arquiteto.
- **O Arquiteto é o chefe e o Produtor o seu braço direito**: segundo Brooks, este seria o arranjo ideal para equipes pequenas, enquanto o anterior (Produtor é o chefe) funcionaria melhor em projetos maiores.

É impossível evitar o paralelo das sugestões de Fred Brooks com aquelas apresentadas por Jeff Sutherland (depois de Takeuchi e Nonaka [5]) em seu método chamado Scrum.

Jeff Sutherland, ao contrário de Brooks, não deixa dúvida sobre quem 'fala mais alto' em um projeto. O Arquiteto, que no *Scrum* é chamado de *Product Owner*, tem a palavra final. Enquanto o Coordenador (Produtor), ou *Scrum Master*, trata de eliminar riscos e obstáculos que estejam impedindo o time de obter sua melhor performance. São, respectivamente, o 'Navegador' e o 'Piloto' em uma equipe de *rally*, uma analogia criada pelo próprio Sutherland. Este desenho faz lembrar também uma colocação de Tom DeMarco em um de seus principais trabalhos, "Peopleware"[6]:

"A função do gerente não é fazer as pessoas trabalharem e sim tornar possível que elas trabalhem."

Podemos perceber o mesmo tipo de separação de funções e responsabilidades na indústria cinematográfica. O diretor do filme (arquiteto) tem a palavra final sobre todos os aspectos da criação. E os produtores (coordenadores) tratam do dia-a-dia do projeto.

Antes de encerrar o tema uma observação: Fred Brooks, assim como Scott Berkun, trabalhou na indústria, desenvolvendo produtos. Falta em seus '*job descriptions*' do Arquiteto e do Produtor (Coordenador) a preocupação com um Cliente. Parece óbvio que o chefe, seja ele o Arquiteto ou o Coodenador, seja a principal interface da equipe do projeto com o cliente. Eu prefiro deixar que o Coordenador seja sempre o canal de contato principal com o cliente, independente de ter ou não a palavra final no projeto. As questões técnicas e funcionais, na maior parte das vezes, deveriam ser solucionadas (ou melhor, direcionadas) pelo Analista de Negócio, apresentado no sub-capítulo anterior.

A Outra função da Organização

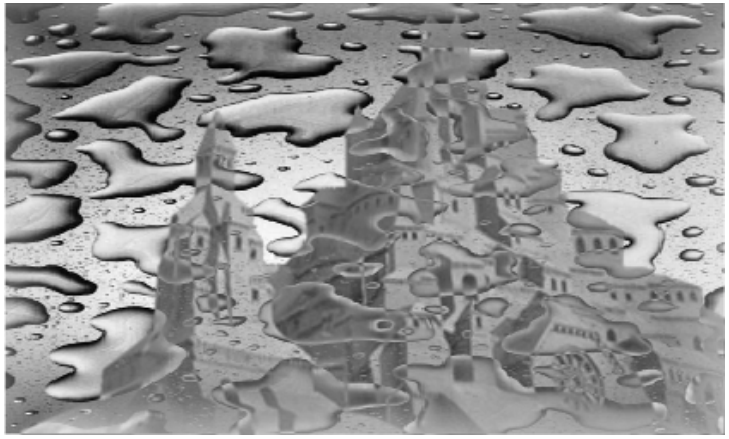
Quando falamos em Organização, Estrutura, a primeira imagem que aparece é a de um organograma. Preocupamo-nos, quase que exclusivamente, em saber quem é que manda no pedaço e quem deve (ter o juízo de) obedecer. Fred Brooks diz que precisamos aprender a desenhar estruturas e processos que incentivem, e não que inibam a criatividade e a iniciativa. E lembra: "a Criatividade vem dos indivíduos e não das estruturas e processos".

Scott Berkun segue linha parecida dizendo que os "projetos dependem muito da habilidade do time em fazer uso do conhecimento de seus membros, de compartilhar idéias e trabalhar em sincronicidade, ao invés de seguir restritivas linhas de autoridade, uma rigorosa disciplina e uma compulsão a seguir ordens sem nenhum tipo de questionamento".

Esse embate é muitíssimo bem documentado por Domenico de Masi no livro "Criatividade e Grupos Criativos"[7]. Domenico afirma que atravessamos uma fase caracterizada por um *Cultural Gap*, um fenômeno que "constrangeu os atuais *knowledge workers*, os trabalhadores do conhecimento, a se organizarem segundo os velhos princípios industriais". Que seja só uma fase. Mas ainda não há muitos indícios de que esteja para terminar. Por exemplo, o termo ‘fábrica de software’ é de uma infelicidade incrível. Um oxímoro, no meu ponto de vista. Por outro lado temos a consolidação de produtos como Linux, Apache, Eclipse e Firefox, que são criados em ambientes onde parece imperar o caos. Deve haver um meio termo, não?

Castelos de Areia...

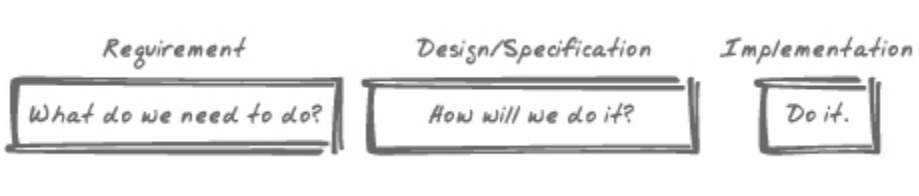
Em 1986 Fred Brooks publicou o artigo "*No Silver Bullet*", que aparece como o capítulo 16 da edição que comemorou os 20 anos de "The Mythical Man-Month". Para Brooks, "ainda cometemos erros de sintaxe, com certeza, mas eles não são nada quando comparados aos erros conceituais da maioria dos sistemas". Scott Berkun cita Brooks na abertura do capítulo *"How to figure out what to do"*, o terceiro de "The Art of Project Management":



"A parte mais difícil da construção de software é decidir o que construir. Nenhuma outra etapa do trabalho conceitual é tão difícil quanto a fixação dos requisitos técnicos detalhados, incluindo todas as interfaces com usuários finais, com máquinas e outros sistemas. Nenhuma outra etapa compromete tanto o projeto se executada erroneamente. Portanto, a função mais importante que o construtor de software executa para seu cliente é a extração iterativa e o refinamento dos requisitos do produto".

Para Brooks, trata-se de uma função que deveria ser executada por uma pessoa ou um grupo pequeno de pessoas. Seria, para ele, uma forma de garantir a Integridade Conceitual de uma solução. A separação desta etapa, quando decidimos 'o que deve ser construído', da etapa de implementação, quando decidimos 'como construir', seria outra forma poderosa de buscar tal integridade.

Berkun chama de 'insanamente simples' a forma como ele vê a etapa de planejamento de um projeto. Ela é representada no gráfico abaixo:



E, seguindo em sua 'insanidade', Berkun justifica a necessidade de Planos. Segundo ele, os planos "funcionam como um remédio contra todo tipo de estupidez porque forçam que questões importantes sejam resolvidas enquanto há tempo para considerar outras opções".

Apesar de uma (sutil) diferença, ambos os autores concordam com a separação das fases de Arquitetura (ou ‘o que’, coleta de Requisitos, Definição etc) e de Especificação (ou ‘como’, design/especificação etc). Uma distinção óbvia, simples, mas deveras negligenciada. Quantas e quantas vezes testemunhamos 'arquitetos' (assim, entre aspas mesmo) discutindo 'comos' em dias iniciais de um projeto?

Brooks radicaliza ao propor que o principal artefato gerado pelo Arquiteto de uma solução é o Manual, uma especificação de toda a parte externa de um sistema. É o manual do usuário mesmo, nas fases iniciais de um projeto! Um documento que não deveria se preocupar em explicar os ‘comos’.

As Visões e o Documento de Visão

Berkun é um pouco mais metódico e indica a necessidade de 4 documentos principais. Antes, porém, ele alerta para a criticidade do balanceamento de três perpectivas:

- **Negócio**: "... quando equipes de engenharia ignoram como seu negócio funciona, muitas decisões da gerência parecerão ilógicas ou estúpidas";
- **Tecnologia**: "...é o *mindset* de construção e materiais. Tem o foco em 'como' as coisas devem ser feitas";
- **Cliente**: "A mais importante das três perspectivas... Mas, infelizmente, a mais fraca em muitas organizações."

Segundo Berkun, as três perpectivas sempre se sobrepõem. "Cada consideração 'de negócio' tem implicações Técnicas e para o Cliente (e assim por diante)". E cada decisão pode favorecer determinado ponto de vista, em detrimento de outro. "Ao investir tempo explorando cada uma das perspectivas", diz Berkun, "é possível ver oportunidades para melhores decisões estratégicas".

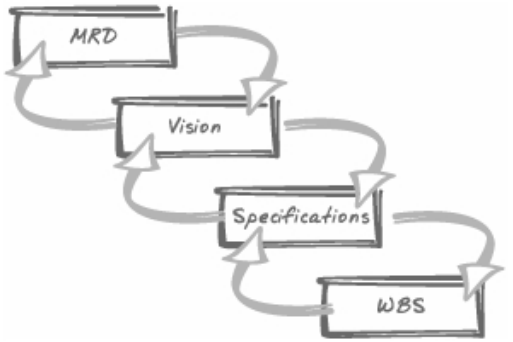
Para Berkun, a fase de planejamento de um projeto só se encerra quando os 4 documentos propostos por ele estão prontos. Ou, em suas palavras: quando "as decisões que eles contêm estão tomadas". Os documentos são:

- **Marketing Requirements Document (MRD)**
  Trata-se da 'Visão do Mundo' apresentada pelo negócio ou sua equipe de marketing. Apresenta os objetivos do negócio e ajuda a definir "o que" o projeto deve contruir visando sua satisfação;

- **Documento de Visão (Escopo)**
  Define os objetivos do projeto, explica sua lógica e apresenta características (em alto nível), requisitos e datas. Definem diretamente "o que" o projeto deve realizar;

- **Especificações**
  Define o "como" de um projeto, com uma perspectiva de design e engenharia;

- **Work Breakdown Structure (WBS)**
  Mostra como o time trabalhará para realizar o projeto.



Berkun também parece cair na 'armadilha' waterfall quando propõe que a fase de planejamento de um projeto só se encerra com a entrega (definitiva) destes 4 documentos. Apesar de indicar os aspectos iterativos e incrementais de sua elaboração. Não deveriam ser apenas os 'acidentes' (também conhecidos como 'mudanças') que nos forçam a voltar a tais documentos (tal fase).

Se o MRD (que pode ser um RFP – *Request for Proposal*, por que não?) é (ou pelo menos deveria ser) elaborado pelo Cliente, temos que o primeiro documento confeccionado pelo time do projeto é o Documento de Visão. Segundo Berkun, além de apresentar e explicar todas as características (*features*) do produto a ser gerado, como no Manual sugerido por Brooks, o documento deve:

- Explicar o projeto em apenas uma sentença (uma 'declaração de visão');
- Mostrar como o projeto contribuirá para a satisfação dos objetivos do negócio;
- Apresentar as características/cenários essenciais para os Clientes (prioridade 1);
- Mostrar as características/cenários considerados 'desejáveis' mas não essenciais (prioridade 2);
- Apresentar os clientes e os problemas que o projeto deve solucionar para eles;
- Bem como apresentar os atores (*stakeholders*);
- Explicar por que os clientes comprariam (ou alugariam) o produto do projeto;
- Apresentar os concorrentes e uma comparação de seus produtos com aquele que o projeto deve gerar;
- Explicitar o que está "fora do escopo" do projeto;
- Mostrar quais os riscos do projeto;
- Suas dependências externas (sub-contratados e afins);
- Em alto nível, como o trabalho será organizado; e
- Apresentar todas as suposições e dependências.

Ou seja, o Documento de Visão, como proposto por Berkun, compila todas as informações e decisões elaboradas no planejamento inicial de um projeto. Esta compilação, escrita de forma a ser acessível/legível para todos os atores de um projeto, se torna um dos principais meios de comunicação/negociação com os clientes. Não entendo porque Berkun não sugere a inserção de um cronograma. Outra alteração que eu faria é a criação de uma 'prioridade 3', com a lista de todas as características/cenários classificados como 'perfumaria' ou supérfluos. Eles sempre estão lá. É só uma questão de classificá-los.

Berkun sugere que 5 iterações seriam suficientes até a geração de uma versão final do Documento de Visão. Acho arriscado fixar assim o número de 'versões'. Cada projeto pode determinar um ritmo/ciclo bastante particular. Mas creio que um mínimo de 3 iterações sejam necessárias. O trabalho nas Especificações e na WBS gerará, sem dúvidas, alterações na Visão.

Por fim, Berkun destaca as 5 qualidades de uma "Boa Visão":

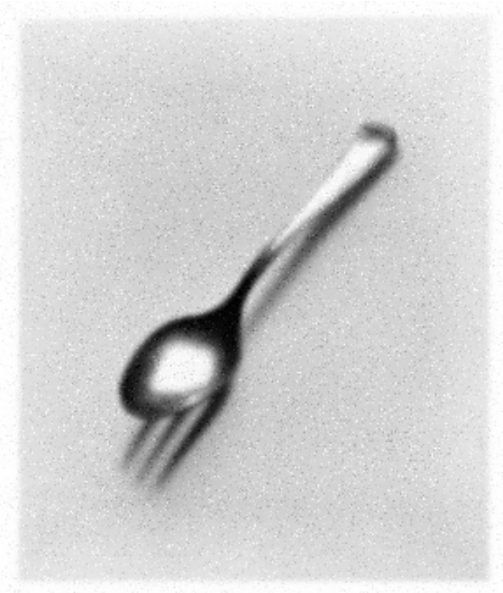
- Efeito “Simplificador”;
- Apresenta de 3 a 5 objetivos de alto nível;
- Consolidada;
- Inspiradora; e
- Memorável.

O fato do autor, no caso Scott Berkun, manter um blog faz com que seu livro seja constantemente atualizado. Recentemente Berkun publicou um pequeno artigo chamado "Why vision documents stink". Isso mesmo, 'cheiram mal'. Ele comenta que em levantamentos informais, em suas palestras por exemplo, constatou que a grande maioria dos Documentos de Visão são muito ruins. Ele tem três teorias para o problema:

- A falta de um 'autor-líder' que, no meu ponto de vista, deveria ser o próprio Arquiteto;
- Os documentos não seriam escritos para servir ao leitor, denotando falta de compreensão e de interação com o cliente; e
- A confusão entre *'hype'* e realidade, que geraria documentos meio 'marketeiros' e pouco objetivos.

Vai Entender o que o Cliente Quer

Scott Berkun dedica várias páginas de seu livro para tratar da tal 'Engenharia de Requisitos' (termo que ele não utiliza) e do *'Design'* (termo que ele usa insistentemente) de uma solução. Capítulos como "*How to figure out what to do*", "*Where Ideas come from*" e "*What to do with ideas once you have them*" dão o merecido tratamento aos temas.



Para Berkun, a Perspectiva do Cliente pode ser obtida de duas formas: Requisitos e Pesquisas. E sugere que para cuidar das funções relacionadas à esses tipos de levantamentos deveríamos alocar dois tipos de *experts*: Engenheiros de Usabilidade e *Designers* de Produtos. Mas Berkun observa: "Mesmo que na última década muito progresso tenha sido feito no refinamento de métodos de pesquisa e compreensão de clientes, grande parte dele não foi assimilado pelo corpo gerencial – ou em organizações centradas na engenharia. Ainda é incomum que times de projetos tenham um *expert* em pesquisa de cliente, *design* de interfaces e usabilidade disponibilizado para os tomadores de decisão". Na sugestão que apresentei na 2ª parte desta série, o Desenvolvedor de Interfaces realiza tais funções. Trabalhando bem próximo do Analista de Negócios.

Muito se fala, e eu não tenho dúvidas, de que o tema requisitos responde por mais de 50% dos problemas em nossos projetos. Berkun nos apresenta 5 dicas para a obtenção do que ele chama de "Requisitos de Alta Qualidade":

- **Planeje as negociações de requisitos e as iterações**
  Identificados nossos clientes e demais atores do projeto, torna-se factível a elaboração de uma Agenda para a Coleta de Requisitos. No pior cenário, uma RFP deve dar pistas suficientes para o rascunho de um plano inicial.

- **Elimine as Suposições Erradas**
  Como identificá-las? Só perguntando. Como diz o Berkun, "se você é o coordenador do projeto ... religiosamente faça perguntas esclarecedoras, como 'por que precisamos disso?', 'que problema isso resolve?'", e assim por diante. Só não permita que seu time assuma como verdadeiras solicitações que podem nem mesmo ter partido do cliente, ou dos verdadeiros usuários.

- **Busque as Informações que Faltam**
  "Os mais notáveis erros em requisitos são erros de omissão". A coleta e análise bem realizadas devem tornar bem visíveis todas as peças que faltam no tabuleiro do projeto.

- **Defina Prioridades Relativas para todos os Requisitos**
  Costumo solicitar, ainda na coleta, que o cliente/usuário defina se aquela solicitação é Fundamental, Importante ou Acessória. Uma classificação clara e simples assim ajuda muito em diversas tomadas de decisão.

- **Refina ou Elimine a Linguagem Ambígua não-Intencional**
  "Palavras como *rápido, grande, pequeno, bonito* e *usável* requerem métricas relativas para serem entendidas. Em alguns casos, pode ser do interesse de todos que elas permaneçam 'em aberto' até momentos posteriores do projeto". Mas, a princípio, devemos eliminar toda redação que não permita um entendimento único de um requisito.

Cabe aqui outra intromissão minha. Ainda há muito trabalho a ser realizado antes do Analista de Negócios/Sistemas repassar os requisitos para o time de Arquitetura da Solução. As dicas do Berkun listadas acima são úteis mas não suficientes. No meu ponto de vista, todos os requisitos coletados devem ser estruturados, formando uma Base de Conhecimentos do projeto. Existem algumas ferramentas desenhadas especificamente para isso. Mas o mais importante é o modelo da base. Nesta palestra, já meio velhinha (apresentada em 2003 em evento da SUCESU/PMI), apresento uma sugestão de um modelo 'mínimo'. Em "Requirements-Led Project Management" [8], de Suzanne Robertson e James Robertson, há um modelo um pouco diferente, que eles chamam de '*Requirements Knowledge Model*'. Uma base persistida em um gerenciador de bancos de dados, ao invés de documentos textuais ou planilhas eletrônicas, é mais gerenciável. Fica mais fácil manter a tal rastreabilidade bi-direcional dos requisitos e também seu relacionamento com todos os demais artefatos produzidos no projeto.

Uma vez gravado, um requisito nunca mais pode ser deletado. Parece boba, mas se trata de uma regra fundamental. Cada mínima alteração na redação do requisito ou em algum de seus atributos significa a inserção de um novo requisito, que substitui o anterior mas mantém um vínculo. Assim conseguimos rastrear todas as mudanças que ocorreram desde os instantes iniciais de um projeto. Portanto a base de conhecimentos acaba se tornando um dos, senão o principal subsídio para o Gerenciamento de Mudanças (tema do próximo capítulo).

Perdidos no Espaço (entre os Requisitos e a Solução)

Constatação inquietante de Berkun: "Por razões que não consigo explicar totalmente, muitas pessoas têm dificuldade com o planejamento do trabalho criativo. Na maioria dos livros que li sobre gestão de projetos e desenvolvimento de software, há pouca cobertura sobre como sair de uma lista de requisitos do que deve ser implementado para um bom *design*. Cronogramas apresentam uma data para quando os requisitos supostamente estarão finalizados, e outra data para quando as especificações supostamente estarão prontas, mas poucas instruções são fornecidades para a execução daquilo que está entre essas duas datas".

Para Berkun, tão logo estejamos com os requisitos 'no lugar', "os *designers* podem explorar o território desenhado pelos requisitos. Há um largo espaço, chamado 'Espaço do Problema', de maneiras potenciais para resolver o problema colocado. Dependendo dos requisitos, este espaço pode ser bem grande; por exemplo, há um infinito número de possibilidades para se desenhar uma casa, um sistema de contabilidade, um *web site* ou seja lá o que for que esteja lhe pagando. Portanto, até que você tenha alguma noção das possibilidades que existem, não é muito esperto se comprometer com qualquer coisa descoberta nos momentos iniciais".

É a fase em que temos só uma folha ou um quadro branco. Brancos e vazios. É a fase das Idéias - apaixonante para alguns e apavorante para outros. As idéias vêm das pessoas, lembra Berkun: "nenhuma idéia na história da humanidade brotou de uma pilha de pedras, ... , de livros de auto-ajuda, de seminários de criatividade ou de sessões de *brainstorming*".

Por que então a fase de *design* (Arquitetura da Solução) seria tão negligenciada? Para Berkun, "talvez seja porque as pessoas temem a exploração [de idéias], especialmente quando estão sendo observadas".

Berkun diz ter "evidências irrefutáveis de que há um número infinito de idéias prejudiciais, horríveis, inúteis, comicamente estúpidas e embaraçosamente ruins". Ele solta essa pérola ao questionar a máxima (segundo ele oriunda de comerciais de TV e de sessões de *brainstorming* – ou de comerciais de TV vendendo sessões de *brainstorming*) que diz que "não existem más idéias". Lógico que elas existem. Aos borbotões. Dentre outras coisas, ele sugere algo muito simples: "Boas perguntas atraem boas idéias!"

Para Berkun existem dois tipos de perguntas 'Boas' e um tipo de pergunta 'Ruim' quando estamos envolvidos em um trabalho criativo:

- **Perguntas Direcionadoras (Boas)**
  "Chamam a atenção do time para algo importante, útil ou mesmo central para o trabalho em execução". Tipo: Como o usuário saberá que deve clicar neste botão para ir para a próxima tela?

- **Perguntas Criativas (Boas)**
  Ao contrário das questões direcionadoras, estas devem "levar o time para territórios ainda não explorados do problema em questão". Algo como: Haverão outros meios para o cliente chegar na próxima tela?

- **Perguntas Retóricas (Ruins)**
  "Gêmeas das questões criativas, diferenciam-se pelo fato de serem insinceras, perguntadas sem a intenção de se obter uma resposta". Por exemplo: Aí 'cai a ficha' e o cliente de repente descobre que tem que ir para outra tela, certo?

Mas, independente da qualidade (e da inteligência) de nossas perguntas, devemos esperar diversas idéias 'ruins, comicamente estúpidas, hilariantes' etc. Triste é o ambiente pouco tolerante a elas, porque, além de sisudo e monótono, inibe a participação de todos os membros da equipe, mina o espírito de colaboração que deveria existir durante todo o projeto. Além de desperdiçar boas piadas, né?

Seguindo com Berkun: "É melhor que a gente suje as mãos e cometa vários erros nesta fase. Quanto antes isso acontecer, mais rápido a gente chega às boas idéias."

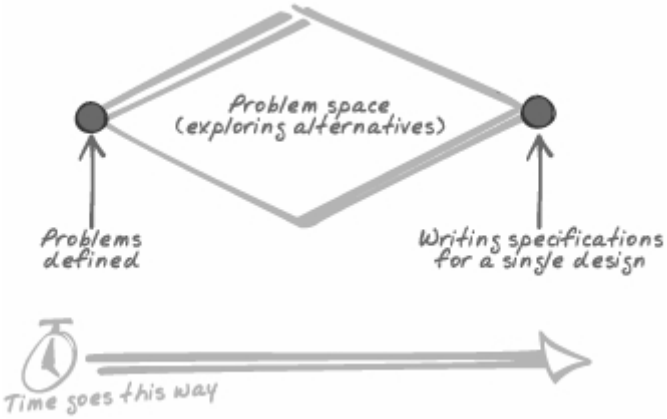
"As duas mais importantes ferramentas de um arquiteto são a borracha na sala de desenhos e a marreta na construção"
- Frank Lloyd Wright

"A mais importante ferramenta do físico é sua cesta de lixo."
- Albert Einstein

Berkun também detona outro 'mantra', comum em certos círculos por aí, o tal *"think outside the box"*: "Não é a eliminação das caixas a parte mais difícil – é exatamente saber quais caixas utilizar e quando. Restrições estarão sempre presentes". E completa: "faça o que você quiser com a caixa. Pense dentro da caixa, fora da caixa, debaixo da caixa, corte-a e faça fogo com ela, qualquer coisa, desde que você gerencie para resolver os problemas identificados como críticos para o projeto".

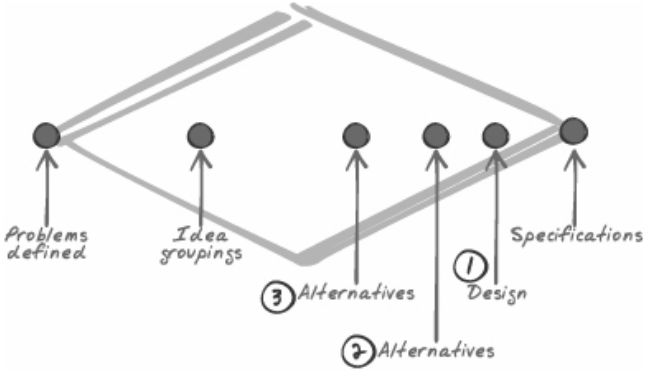
O Problema do Tamanho do 'Espaço do Problema'

"Idéias fogem do controle", diz o título de um sub-capítulo do livro de Berkun. Por isso, "se é difícil encontrar boas idéias, é muito mais complicado gerenciá-las". Berkun diz que um erro muito comum é tratar o processo de *design* como se fosse um grande 'interruptor'. Para ilustrar melhor a questão Berkun apresenta o gráfico abaixo:



A partir de uma sólida base de (bons) requisitos, começamos a explorar alternativas de solução. Com o passar do tempo a tendência é que o número de alternativas (cenários) aumente. Aumentamos assim o tamanho do 'Espaço do Problema'. Um dos riscos, segundo Berkun, é não saber o momento de parar com a geração de idéias e começar a filtrá-las. Para tal Berkun sugere a fixação de alguns pontos de checagem.

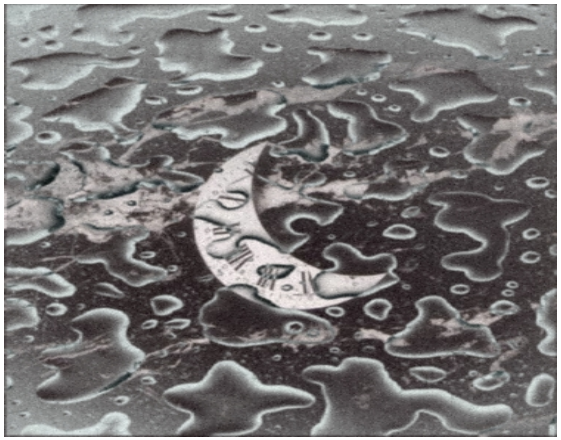
Como ilustra o gráfico abaixo, são 6 os grandes *check-points* que deveriam nos guiar no processo de arquitetura da solução:



a. **Visão e Prova de Conceito**
   Nosso ponto de partida deveria ser um documento de visão e uma prova de conceito. Traduzindo para Pindorama: será a tal 'proposta técnica' para muitos corajosos colegas.
b. **Agrupamentos de Idéias**
   Depois de um certo tempo jogando conversa fora, digo, idéias 'para fora', é conveniente que elas sejam agrupadas e analisadas.
c. **Três Alternativas**
   Naquele que deve ser identificado como 'meio do caminho' desta etapa do projeto, é indicado que a lista de alternativas seja filtrada, gerando de 3 a 5 alternativas mais viáveis.
d. **Duas Alternativas**
   Pouco tempo depois deveríamos ser capazes de escolher apenas 2 alternativas de implementação.
e. **Um Design**
   E então apenas um *design* (ou Arquitetura da Solução).
f. **Especificações**
   Então, com a Arquitetura definida, geramos a especificação técnica da solução.

... E a inevitabilidade das Marés

"**O primeiro passo é aceitar as mudanças como um estilo de vida, e não como um desvio, uma exceção**". Assim, de forma simples e direta, Fred Brooks começa a tratar o tema "Mudanças".



Mudanças ocorrerão em um projeto não só porque o trabalho inicial (coleta e análise de requisitos e arquitetura da solução) não foi bem feito. Segundo Brooks, "a entidade Software está sempre sujeita a pressões por mudanças. Claro, como prédios, carros e computadores. Mas coisas manufaturadas raramente mudam após sua produção". Já o software sim, dada sua "infinita maleabilidade". Não carecemos nem mesmo das borrachas e marretas de Frank Lloyd Wright.

"Longe de mim", diz Brooks, "sugerir que todas as mudanças de objetivos do cliente podem ou devem ser incorporadas ao desenho da solução. Um delimitador deve ser estabelecido, e ele deve ficar mais restritivo na medida em que o desenvolvimento avança, ou o projeto nunca terminará".

O delimitador parece óbvio na teoria, mas é peça rara na prática. Se a mudança solicitada for crucial para o pleno atendimento das necessidades de negócio, o que fazer? Ignorá-la? Dizer que ela será implementada na '2ª versão'? Toda solicitação de mudança deve ser analisada com carinho, independente do que estejam indicando o termômetro e o taxímetro do projeto. Independente da fase do projeto e da fase da lua.

Para Scott Berkun toda solicitação de mudança deveria seguir o mesmo processo de negociação que guiou a fase inicial de coleta de requisitos. Creio que a assimilação do processo se torna mais simples se entendermos que toda solicitação de mudança nada mais é que um novo requisito. Ou, em muitos casos, uma 'nova versão' de um requisito. Quando executamos corretamente a Engenharia de Requisitos, avaliamos os impactos que cada nova solicitação pode causar naquelas previamente coletadas. Agora, recebendo um *change request*, executaríamos o mesmo tipo de análise. Dependendo do porte do projeto e do número de dependências (grau de 'acoplamento') dos requisitos, tal avaliação pode ser penosa e demorada. É inevitável? Berkun sugere um breve *check-list* para uma avaliação prévia dos requisitos que apareceram 'fora de hora':

- Qual problema estamos tentando resolver? Precisamos resolvê-lo para obtermos sucesso? Precisamos resolvê-lo na iteração atual? Podemos viver com o problema?
- Trata-se de um sintoma ou uma causa? É aceitável que tratemos apenas o sintoma?
- Temos noção do impacto desta mudança?
- O custo da mudança será compensado pelo benefício gerado?
- E os riscos de novos problemas são compensados pelo benefício da mudança?

A menos que a solicitação de mudança seja absurdamente ridícula, a execução do *check-list* acima não será rápida e muito menos trivial. Por isso cabem aqui dois alertas: i) O cliente, ou usuário ou o *stakeholder*-Zezinho que solicitou a mudança deve participar do processo acima. Ele precisa ter noção do 'estrago que está prestes a causar'. E ser co-responsável por ele; e ii) O processo de desenvolvimento em uso (a metodologia) deve tentar programar o momento certo para a avaliação das mudanças solicitadas.

Como foi apresentado na 1ª parte desta série, quanto maior a incerteza (a volatilidade dos requisitos), menor deve ser a duração de uma iteração. No mundo ideal, todas as solicitações de mudanças são analisadas no momento em que a equipe planeja a próxima iteração. Se uma triagem foi executada anteriormente pelo coordenador ou analista de negócios, em conjunto com o Zezinho, então não é o mundo ideal. É o paraíso mesmo.

Scott Berkun apresenta então uma forma muito simples de gestão de mudanças, que ele chama de "versão *super-lean* do processo de especificação". Consiste do seguinte:

1. O Coordenador do Projeto (ou o Analista de Negócios - interferência minha), escreve um sumário da mudança solicitada, incluindo sua relação com os objetivos do projeto e com os requisitos previamente apresentados; justifica a necessidade da mudança; e apresenta o desenho da mudança a ser implementada. Berkun sugere que este documento tenha no máximo duas páginas;
2. O programador, o testador e todos significativamente impactados pela solicitação devem analisar o documento gerado e dar suas contribuições. As mais notáveis (e ansiosamente aguardadas por todos) são as estimativas de desenvolvimento e testes.
3. O documento finalmente é apresentado aos líderes do projeto (e, como sugeri anteriormente, ao cliente, usuários, zezinhos etc). Nessa breve reunião a mudança é aceita ou não. Se recusado, diz Berkun, "o documento deve rastejar até o canto mais próximo e, soluçando incontrolavelmente, desaparecer do universo do projeto".

Insisto que a reunião citada no item 3 acima deveria ser programada e tratar de um conjunto de solicitações de mudanças. Se executada *ad hoc* e a granel, se transformará rapidamente no 'inferninho' do projeto.

Fred Brooks cita um estudo de Lehman e Belady [9] que mostra que em cada nova versão o número de novos módulos cresce linearmente, mas o número de módulos afetados pelas mudanças aumenta exponencialmente. Todas as correções e alterações de rumo (em relação à arquitetura original) "tendem a destruir a estrutura, a aumentar a entropia e desordenar o sistema".

O divisor de águas, a separação entre marés (mudanças) benéficas e *tsunamis* detonantes, deveria ser mais nítido. Mas a prática prova que não é. Está no discurso de todos os processos modernos que devemos aceitar as mudanças. Afinal, elas são inevitáveis. Mas sabemos que alguns *change requests* podem simplesmente inundar o projeto com efeitos colaterais mortais. O que permite sua distinção? Como avaliar corretamente o impacto e os riscos de uma mudança? Creio que é impossível sem uma clara visão da arquitetura do sistema. Um modelo detalhado, que exponha todas as interfaces entre todos os módulos, parece ser a melhor vacina contra mudanças maléficas. Mas um modelo só mede o impacto das mudanças na arquitetura do sistema. E os planos, cronogramas, agendas e finais de semana prolongados? Como ficam?

Planejar ou não Planejar? É uma questão?

Apesar de demonstrar uma certa simpatia por XP (*eXtreme Programming*) e suas breves iterações, Berkun reforça a utilidade dos planos de longo prazo: "mesmo quando eles são grosseiros, eles tornam as mudanças de curto ou médio prazos mais fáceis". E justifica: "quando uma mudança nos objetivos, requisitos ou no *design* ocorre, raramente o plano original vai parar na lixeira". O plano original talvez seja nossa melhor (senão única) base de comparação para uma correta avaliação das mudanças propostas. Berkun cita Dwight D. Eisenhower:

> **"Nenhuma batalha é vencida de acordo com um plano, mas nenhuma batalha é vencida sem um."**

Berkun (e mais um monte de gente) gosta de comparar Projeto com partidas de xadrez. Tanto que o capítulo de seu livro que trata de forma mais específica o tema mudanças chama-se *"Middle-Game Strategy"*. Cada decisão do gerente do projeto, assim como cada movimento de um enxadrista, só pode assumir uma de duas características possíveis:

- **Conservadora**: deixa-o com o maior número possível de 'movimentos futuros', de opções. Em um projeto pode significar uma desaceleração do ritmo. Mas, escreve Berkun, "este é o preço do seguro que você está comprando".
- **Agressiva**: mostra total domínio e comprometimento com uma estratégia. O gerente confia em seu plano e em sua 'força'.

A ausência de um plano não permite nem mesmo avaliar o perfil das decisões do gerente do projeto. E a maneira como elas são apresentadas pode ser um péssimo indicador. Lembra aquela piada do marido que diz sempre ter a última palavra em casa: 'Sim senhora!'.

Para Scott Berkun o Gerente que tem total controle do projeto está sempre 'um passo à frente'. Para tanto ele sugere a realização de dois *check-lists* para a verificação de nossa sanidade, digo, da sanidade do projeto. O primeiro é tático (diário), e apresenta as seguintes questões:

- **Quais são nossos objetivos e compromissos? Eles ainda são válidos?**
  No meio de tanto trabalho, diz Berkun, "é muito fácil perder os objetivos de vista". Olhar para eles diariamente é uma forma de manter o foco e avaliar corretamente as prioridades.
- **O que estamos realizando hoje contribui para a realização dos objetivos?**
  É claro como os trabalhos em execução contribuirão para a satisfação dos objetivos e dos requisitos? Se não, diz Berkun, "o barco tá começando a ficar à deriva".
- **Os trabalhos estão sendo concluídos de forma a satisfazer os requisitos e cenários?**
  "Há mil maneiras de completar uma unidade de trabalho que não satisfaz o espírito e a intenção do *design*", lembra Berkun. Sabemos que a distância entre o 'tá pronto' do programador e o 'tá pronto' do usuário pode ser quilométrica.

O outro *check-list* é estratégico e, segundo Berkun, deveria ser executado semanalmente ou mensalmente, seja em reuniões de discussão do *status* do projeto ou mesmo individualmente. As questões são:

- **Qual a probabilidade de atingirmos o próximo *milestone* com o apropriado nível de qualidade?**
  Com certeza aconteceram mudanças desde o trabalho de estimativas. E mesmo que não, não custa nada perguntar ao time se o cronograma segue verdadeiro e exequível.
- **Quais ajustes são necessários para aumentar tal probabilidade?**
  Berkun diz que "é raro obter 100% de confiança na próxima data de qualquer um que seja honesto e são". Portanto esta pergunta (quase) sempre será colocada.
- **Como executar tais ajustes com cuidado e de forma isolada?**
  "Um telefonema? Um email? Despedindo alguém?". Berkun alerta que devemos pensar de forma 'cirúrgica', mas não devemos temer as ações e decisões que precisam ser executadas/tomadas.
- **Quais são os maiores ou mais prováveis riscos que enfrentamos hoje, na próxima semana ou no próximo mês? E quais são as contingências?**
  A simples identificação dos maiores ou mais prováveis riscos já é, de acordo com Berkun, um grande passo no sentido de prevení-los.
- **O mundo mudou e eu não estou sabendo?**
  Os patrocinadores ainda são os mesmos? E seus objetivos, mudaram? O time se preocupa com algo que eu não conheço? Nossas antenas e sentimentos devem estar atentos para mudanças que ocorram tanto no micro-universo quanto no macro-universo do projeto.

Na sequência do mesmo capítulo de "The Art of Project Management" Scott Berkun apresenta várias outras ferramentas para o (micro) gerenciamento (diário) do projeto. Brinquei com os parênteses para destacar a mensagem: gerenciar um projeto significa (tentar) cuidar de um número imenso de varíaveis, a maioria delas muito pequena e volúvel, durante todo o dia. Todos os dias.

A Receita e o Bolo de Fubá

Quando pensei no título deste último capítulo busquei algo que fosse extremamente simples, uma receita culinária de um prato bem *'default'*. Mas que ao mesmo tempo parecesse único em cada fornada. O bolo de fubá foi uma lembrança imediata. Nunca vi dois iguais. Mais seco, mais molhado, dois ou quatro ovos, com queijo ou não... com vinagre!?! Pois é, achei mais de 30 mil ocorrências para 'bolo de fubá' no Google. Minha família (mineira, obviamente) compartilha uma mesma receita. Mas o resultado é sempre diferente. Para algo que deveria ser incrivelmente simples: são só nove ingredientes. Um mesmo processo. Um mesmo cronograma.

Mas, surpreendente mesmo é a ilusão que todos nós que lidamos com desenvolvimento de sistemas já alimentamos pelo menos uma vez na vida: a ilusão de que existiria uma receita mágica, uma "bala de prata", que nos ajudaria a entregar nossos projetos no prazo, dentro do orçamento e excedendo todas as expectativas de nossos clientes e usuários. Se é quase impossível reproduzir com exatidão a simplicidade de um bolo de fubá, o que dizer de software?

Em 1986 Fred Brooks publicou o artigo "*No Silver Bullet*", que aparece como o capítulo 16 na edição de 20º aniversário de "The Mythical Man-Month". No texto ele previa que em um horizonte de 10 anos não apareceria nenhuma evolução, nem tecnológica nem gerencial, que promoveria ganhos consideráveis de

produtividade e confiabilidade. "Ceticismo não é pessimismo", Brooks frisava. Nove anos depois, para a edição comemorativa, ele escreveu "*'No Silver Bullet' Refired*", seu 17º capítulo. Responde algumas críticas e conclui que estava certo em sua avaliação. Uma avaliação que pode ser resumida em uma frase apenas: "**Construir software será sempre difícil**". Brooks fundamenta sua tese apresentando quatro propriedades ("irredutíveis") da 'entidade' software:

- **Complexidade**: uma propriedade essencial, não acidental. Ou seja, software é uma entidade complexa por natureza, "talvez a mais complexa de todas as construções humanas". De tal complexidade vem a dificuldade de comunicação entre os membros do time; dela deriva a impossibilidade de enumerar ou mesmo compreender todos os estados de um programa, e daí vem a falta de confiança. É da complexidade da estrutura que vem a dificuldade de se criar novas funcionalidades que não resultem em efeitos colaterais indesejados. Em suma e para usar um termo bem nosso, tupiniquim: Software é um 'bicho de sete cabeças'. Ponto.
- **Conformidade**: "grande parte da complexidade que deve ser dominada pelo engenheiro de software é arbitrária, forçada sem rima ou razão pelas diversas instituições humanas e outros sistemas os quais suas interfaces devem conformar. Isso muda de interface para interface, de tempos em tempos, não apenas porque há uma necessidade, mas porque as interfaces são desenhadas por pessoas diferentes".
- **Instabilidade** (de *changeability*): "A entidade software está constantemente sujeita a pressões por mudanças". "Software pode ser alterado facilmente - ele é infinitamente maleável".
- **Invisibilidade**: abstrações geométricas são muito úteis, mas não conseguem representar toda a complexidade de um software.

Brooks lista então uma série de avanços que podem ajudar a melhorar a qualidade de nossos projetos. Mas frisa que nenhum deles é uma 'bala de prata': Linguagens de alto nível (ele cita Ada - lembrem-se, o artigo

é de 1986); Orientação a Objetos; Programação Automática; Programação Gráfica; etc. Na sequência ele lista alguns princípios que podem atacar diretamente a essência dos problemas com software:

- **Comprar ao invés de Construir**: "a solução mais radical para os problemas com a construção de software é não construí-los". De certa forma as ondas ERP e CRM livraram várias empresas de grande parte do peso da construção e manutenção de sistemas. Mas todas as organizações ainda demandam soluções específicas. Se não as constroem internamente, contratam serviços de desenvolvimento. Não acredito que um dia será possível comprar pacotes (ou componentes ou serviços) para todo e qualquer tipo de problema de negócio.
- **Refinamento dos Requisitos e Prototipação Rápida**: "a parte mais difícil da construção de software é definir precisamente o que construir". "Creio que é impossível para os clientes, mesmo aqueles que trabalham ao lado dos engenheiros, especificar completamente, precisamente e corretamente todos os requisitos do software antes de experimentar algumas versões". Portanto, segue Brooks, "um dos mais promissores avanços é o desenvolvimento de métodos e ferramentas para a prototipação rápida de sistemas como parte do processo iterativo de especificação dos requisitos".
- **Desenvolvimento Incremental**: aumente (cresça) um software, não construa (no texto original, "*grow, not build, software*"). Para Brooks a metáfora da construção já deu o que tinha que dar. "Vamor olhar para a natureza e estudar a complexidade dos seres vivos ao invés dos trabalhos 'mortos' do homem". Este princípio pode ser aplicado tanto no ciclo de vida do projeto quanto no ciclo de vida do próprio produto. Processos iterativos e incrementais já são comuns. Quase 'carne de vaca'. O que é novo é a forma como o Google, por exemplo, 'cresce' e evolui seus serviços. Não se trata meramente de uma manutenção, e eu acredito que muitos de nossos produtos podem ganhar com esse novo enfoque. Independente do público e da tecnologia, diga-se de passagem.
- **Grandes** *Designers*: "Grandes projetos (*designs*) vêm de grandes arquitetos (*designers*). A construção de software é um processo criativo. Uma boa metodologia pode fortalecer e liberar uma mente criativa; mas não pode inflamá-la ou inspirá-la". Brooks conclui: "a principal questão para a evolução da arte do software está centrada, como sempre esteve, nas Pessoas". Não é por acaso que Brooks encerra seu livro recomendando a leitura de "Peopleware" , de Tom DeMarco.

Receitas, Metodologias, Processos...

E não parece ser uma mera coincidência que Scott Berkun inicie seu livro citando... "Peopleware", de Tom DeMarco:

> **"A obsessão com metodologias é outra instância da ilusão high-tech. Deriva da crença de que o que realmente importa é a tecnologia... Independente de qual seja o avanço tecnológico, ele cobrará seu preço com a deterioração da sociologia do time."**

Para Berkun, "a pior coisa é seguir cegamente um conjunto de regras e procedimentos só porque eles apareceram em um livro famoso ou porque são promovidos por um respeitado guru". Berkun coloca que processos e metodologias são muito importantes, mas nunca serão 'balas de prata', entregadores de projetos bem sucedidos. E alerta para o perigo dos gerentes, em determinado momento de um projeto, "começarem a acreditar que o Processo é o Projeto". Pode parecer absurdo, mas este desvio é mais comum do que se imagina.

Um bom processo, segundo Berkun, apóia as pessoas e amplifica o seu valor. E seria o resultado da combinação de duas coisas: i) o que torna projetos e times bem sucedidos de uma maneira geral; e, ii) o que torna o projeto e o time atuais diferentes dos outros.

Eu gosto de acrescentar uma terceira variável, tão importante quanto o projeto em si e o time, no momento da seleção/customização de um processo: o Cliente. Quando se trabalha na indústria, como é o caso de Berkun, raramente se dispara um projeto para um cliente específico. Daí sua omissão. Mas no mercado de prestação de serviços de desenvolvimento é altamente recomendado que o perfil (valores e a cultura) do cliente seja levado em consideração no momento da customização do processo. Em alguns casos o próprio cliente solicita a incorporação de alguns métodos e artefatos, quando não a adoção de um ciclo de vida específico.

Mas o importante aqui é entender que não existe e nem nunca existirá uma 'metodologia mágica', aplicável em vários projetos. Cada projeto exigirá um processo específico. Mas isso não significa que a organização sempre partirá 'do zero'. Muito pelo contrário. A primeira variável colocada por Berkun acima é "o que torna nossos projetos e times bem sucedidos de uma maneira geral". Trata-se de um corpo de conhecimentos único, exclusivo. E orgânico, já que o natural é que ele cresça e fique mais rico a cada projeto.

Infelizmente, quando olhamos algumas particularidades do mercado brasileiro de prestação de serviços de desenvolvimento, percebemos que muitas empresas dão pouquíssimo valor para esse aprendizado, lançando mão de vínculos empregatícios muitos frágeis (PJs e afins), o que resulta em uma alta rotatividade de seus colaboradores. Há quem acredite que este tipo de conhecimento pode ser capturado e aprisionado em documentos e coisas do tipo. Não é o meu caso. A explicitação de conhecimentos, sua compilação em artefatos que podem ser recuperados a qualquer momento, é benéfica para todas as organizações. Mas não consegue representar nem um pequeno pedaço de toda a experiência adquirida por um time durante a execução de um projeto. Trata-se de outra 'ilusão *high-tech*', para usar um termo do DeMarco.

Voltando ao Berkun. Logo no início de "The Art of Project Management" ele ensina três 'lições-chave' que guiam boa parte de seus métodos, guias e sugestões. São elas:

- **Gerenciamento de Projetos e Desenvolvimento de Software não são artes sagradas**: apesar do ar de 'novidade' que impera em nossa área, é crucial lembrar que existem ensinamentos que podem vir de outros lugares.
- **Quanto mais simples for a sua visão do que você faz, mais poder e foco você terá em sua execução**: estar sempre curioso e aberto à novas idéias é o que torna o crescimento possível. Uma visão simples de nosso trabalho pode facilitar sua comparação com outras coisas que existem ao nosso redor. Aumentamos assim a possibilidade de aprendizagem.
- **Simples não é sinônimo de fácil**: correr uma maratona é simples, certo? Basta começar a correr e não parar até alcançar o quadragésimo kilômetro. Você diria que é fácil? Liderança e gerenciamento também são difíceis, mas sua natureza - realizar coisas de determinada maneira atrás de um objetivo específico - é simples. Bolos de fubá e pães de queijo também são extremamente simples. Não consigo entender porque só aqui em Minas eles são realmente gostosos.

Epílogo

Encerro assim o artigo que elaborei com a intenção de homenagear Fred Brooks e sua obra prima, "The Mythical Man-Month", e também para apresentar o caçula dos gurus (ele detestaria o rótulo), Scott Berkun. Como eu disse lá no início da viagem, estes artigos não substituem de forma alguma o prazer de ler os dois livros. E, posso garantir, não cobrem nem 10% de seu conteúdo.

O retorno que eu recebi (infelizmente a maioria foi *off-blog*) compensou com sobras o trabalho. O próprio Scott Berkun deu uma olhada no trabalho. E comentou:

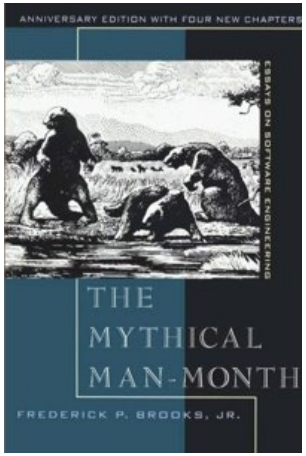
"I did read the tribute you wrote and was flattered by it. I wouldn't compare myself to Brooks - maybe if in 25 years 'the art of project management' is even still in print can a few modest comparisons begin."

Apesar de não concordar com minha comparação, Scott me presenteou com uma cópia autografada de seu livro. Ah, se ele soubesse como torço para que seu livro não permaneça atual daqui 25 anos. A longevidade do livro de Brooks indica, de certa forma, que não aprendemos muito. Ou que não aprendemos direito. Eu sinceramente gostaria que ambos se transformassem em relíquias, documentos de um tempo em que a gente estava brincando de aprender a desenvolver software e a gerenciar projetos.

O próximo passo, ensinou Brooks, é aceitar que

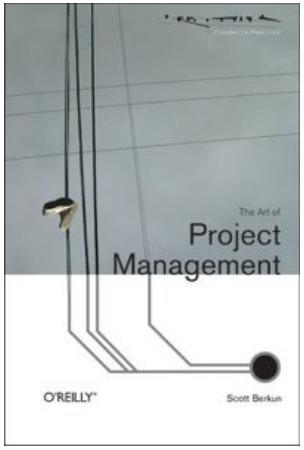
"software é um dos mais complexos trabalhos manuais do homem. Tal complexidade demanda nosso contínuo aprendizado, a melhor utilização das novas ferramentas, uma melhor adaptação a métodos gerenciais, a aplicação do bom senso e, acima de tudo, humildade para reconhecer nossas falhas e limitações".

Serviço



The Mythical Man-Month

Frederick P. Brooks Jr
Addison Wesley
R$ 89,20 (Na Cultura, em 09/mar/06)



The Art of Project Management

Scott Berkun
O'Reilly
R$ 124,24 (Na Cultura, em 09/mar/06)

Outras Obras Citadas

1. **Software Engineering Economics**
   Boehm, Barry
   Prentice-Hall, 1981.
2. **Exploratory experimental studies comparing online and offline programming performance**
   Sackman, H., W.J.Erikson, e E.E.Grant
   CACM, 1968.
3. **Chief programmer teams, principles, and procedures**
   Mills, H.
   IBM Federal Systems Division Report FSC 71-5108, 1971.
4. **O Advento da Nova Organização**
   Drucker, Peter
   Harvard Business Review, 1991.
   Republicado em "HBR – Gestão do Conhecimento" – Editora Campus, 2001.
5. **The New New Product Development Game**
   Takeuchi, H. e I. Nonaka
   Harvard Business Review, 1986.
6. **Peopleware - Productive Projects and Teams**
   DeMarco, Tom e Timothy Lister
   Dorset House, 1999 e 1987.
7. **Criatividade e Grupos Criativos**
   De Masi, Domenico
   Sextante, 2003.
8. **Requirements-Led Project Management**
   Robertson, Suzanne e James Robertson
   Addison-Wesley, 2003.
9. **Programming System Dynamics**
   Lehman, M. e L. Belady
   ACM SIGOPS, 1971.

Créditos e Considerações Finais

As imagens utilizadas neste artigo são de Chema Madoz, fotógrafo espanhol que não faz nenhuma questão de esconder sua maior influência, o louco mestre Salvador Dali. Os puristas podem reclamar dos indevidos *'mashups'* que fiz nas imagens. Entendam como sendo uma forma de forçá-los a visitar o belo site de Chema, onde as imagens podem ser vistas em seu formato original.

As demais imagens, diagramas rabiscados, foram surrupiados digitalmente de "The Art of Project Management". Aliás, boa parte das fotos presentes no livro são do próprio Scott Berkun.
Que faz uma coisa que nunca vi em livros técnicos: lista os sons que 'mantiveram sua sanidade' durante as longas horas em frente ao micro. De Charles Mingus a Aimee Mann, passando por Beatles, Clash, Radiohead e Audioslave. O cara escuta de tudo. E tem bom gosto.

Jonas Fagundes, J. Werther, Roberto, Régis, José Papo, Ivo Michalick e outros amigos 'ocultos' deixaram sua contribuição, na forma de comentários neste blog ou no grupo de discussão CMM-Brasil. Muito obrigado a todos.

Guz Vasconcellos colaborou na revisão, com palpites de diagramação e nos momentos de descontração, levando consideráveis goleadas no *Winning Eleven 8*.

That's all, Folks?

Claro que não. O finito seguirá com um artigo por semana, sempre girando em torno dos temas Engenharia de Software, Gerenciamento de Projetos, Arquitetura de Soluções, e tudo o mais que caiba neste 'torto' triângulo. Sugestões de temas? Quer trocar idéias? Levar palestras e *workshops* para sua escola ou empresa? Demorou!

Ops... err... Vc fez uma busca por 'bolo de fubá' e caiu aqui por engano? Ou então ficou morrendo de curiosidade? É o seguinte:

Ingredientes:
4 ovos
4 copos de leite
1 xícara e meia de açúcar
1 xícara e meia de fubá
2 colheres de sopa de manteiga
2 colheres de sopa de farinha de trigo
1 colher de sopa de fermento em pó
1 xícara de queijo (canastra ou parmesão) ralado
1 pitadinha de sal

Preparo:
Em uma vasilha misture os ovos, acrescente o leite e aos poucos coloque o fubá misturando bem. Coloque o açúcar, o sal, a manteiga e misture novamente. Junte a farinha de trigo e por último o fermento em pó. Leve ao liquidificador, batendo por alguns minutos. Volte com a massa para a vasilha e acrescente o queijo ralado.

Despeje numa fôrma untada e leve ao forno quente por mais ou menos trinta minutos.

Se vai ficar bom como o da minha Vó eu não posso garantir.
Não existe receita mágica, certo?

O que é o finito?

O **finito** é um conjunto de serviços desenhado para atender organizações que executam projetos, particularmente aqueles voltados para o desenvolvimento de software. Os serviços foram estruturados na seguinte sequência:

- **Inicio bem o projeto**, com o pé direito e o correto entendimento da dor que devo sanar;
- **Me comunico de maneira adequada**, com todos e o tempo todo;
- **Sou flexível e me adapto**, entendo que cada projeto demanda um processo específico; e
- **Vivo aprendendo**, faço de cada projeto uma escola.

Os serviços podem ser contratados individualmente ou em grupos configurados pelo próprio cliente. Dentre os formatos de contratação disponíveis, além de consultoria, estão palestras, *workshops* e treinamentos. Para maiores informações consulte este **prospecto (pdf)** ou entre em contato: **finito@pfvasconcellos.eti.br**.

"**What needs to be done?**" é uma questão formulada por Peter Drucker em uma entrevista para a revista Business 2.0. É a 'Grande Pergunta' que, segundo Drucker, todo executivo ou gerente deveria fazer todo dia.