# No Silver Bullet

Frederick P. Brooks, compara o desenvolvimento de software a um lobisomem, afirmando que ambos, não sendo tratados, transformam-se em algo assustador ao longo do tempo. No entanto, os horrores do software, como prazos não cumpridos, falta orçamentos e produtos com falhas, não podem ser abatidos com uma bala de prata.

Os problemas podem ser divididos em dois grupos: **acidentais,** ligados com a produção do software, os menos desafiantes,e os **essenciais**, ligados a arquitetura de um software ou sistema. Não existe um caminho fácil para os resolver, mas existe um caminho.

Os problemas acidentais têm sido resolvidos ao longo do tempo, através de melhorias de memória e velocidade de processamento, aparecimento de linguagens de alto nível, que simplificam o desenvolvimento.

Relativamente aos problemas essenciais, são denotados quatro aspetos únicos específicos do desenvolvimento de software:

Complexidade:

Refere-se ao facto de o escalamento do software para desafios maiores ser algo bastante complexo, uma vez que não basta o uso dos mesmos componentes em tamanho maior, como muitas vezes acontece com o Hardware.

Pode-se afirmar ainda que os sistemas de software atuais são mais complexos do que a maior parte dos objetos que as pessoas constroem. No entanto a complexidade pode causar a má comunicação entre os membros da equipa que pode levar a falhas no produtos e custo adicionais.

Portanto para minimizar a complexidade, é abstraída a sua essência. A não ser que esta faça parte da essência do programa.

Conformidade:

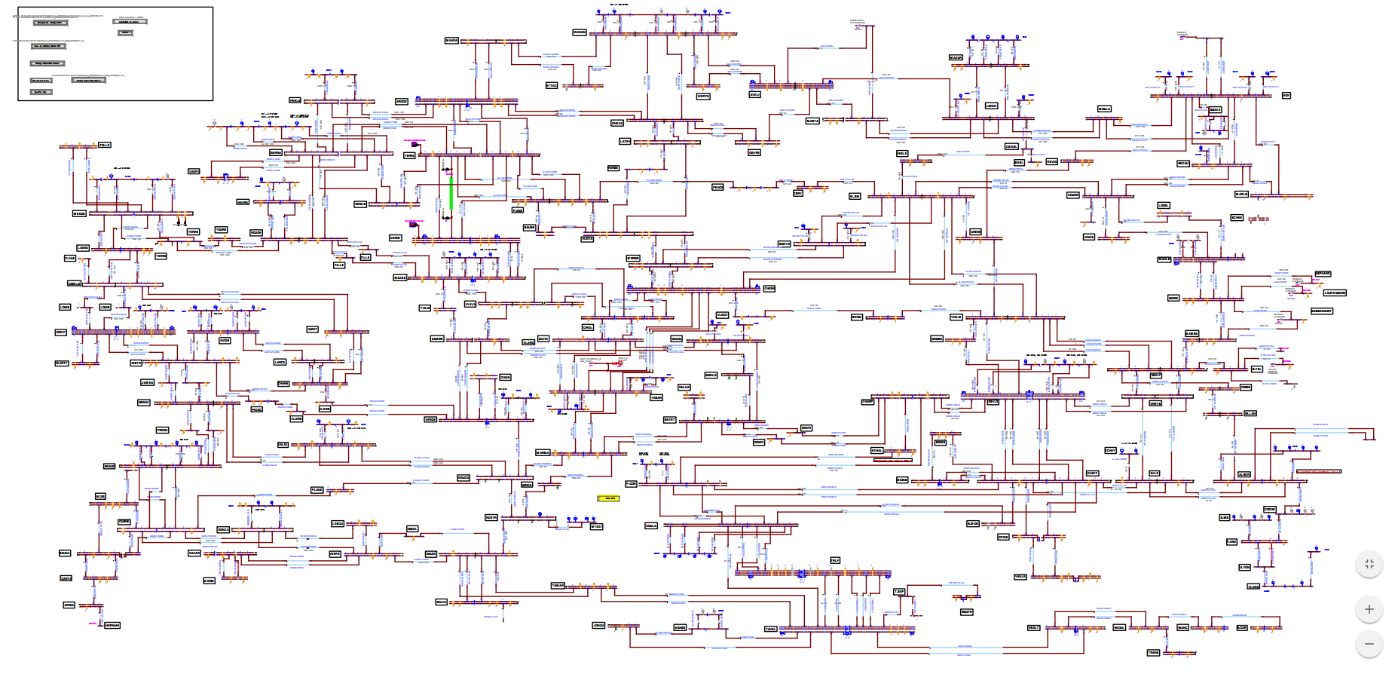
Prende-se com o facto de o software ter de se adaptar a vários tipos de interface, sistema ou instituição existente, para além de não ter padrões fixos.

Embora, em muitos casos da complexidade vem a conformidade com as outras interfaces. E esta não pode ser simplificada com o simples redesenho do software.

Alterabilidade:

Uma das características (positivas) do software é poder ser fácil e rapidamente alterado, no entanto, isto faz com que esteja sistematicamente a ser pressionado a ser atualizado e mudado. Naturalmente todo o software que é popular sofre alterações e para tal é ajustado para as tecnologias mais recente.

Invisibilidade:

 Sem duvida que software não é visível e tal resulta da sua abstração tornando a representação para membros exteriores de um projeto bastante complicada.

*https://www.google.pt/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwj9r7ad7\_ndAhUQHxoKHVPNDiwQjRx6BAgBEAU&url=http%3A%2F%2Fcase-hub.co%2Fcomplex-software-system-diagrams.html&psig=AOvVaw2CmS6J-lL2r\_Lylw4gOlwX&ust=1539191980118721*

É possível observar a complexidade do gráfico, sendo que pode ainda haver vários tipos, representando varias relação dentro do software.

*“Past Breakthroughs Solved Accidental Difficulties”*

Linguagens de alto nível:

Uma das soluções encontradas foi o uso de linguagens de alto nível. Estas conseguiram aumentar a produção de software, a sua leitura, simplicidade e compressividade.

Pode-se afirmar que este tipo de linguagens retira muita da complexidade acidental que um programa possa vir a ter.

Ambiente de programação unificados:

Esta solução resolve dificuldades acidentais ao fornecer librarias integradas com formatos de ficheiros unificados. Fazendo assim com o uso de funções em outro tipo de programas seja possível de forma mais rápida e eficaz.

Hopes for the Silver

Inteligência artificial:

*Two quite different definitions of AI are in common use today. AI-1: The use of computers to solve problems that previously could only be solved by applying human intelligence. AI-2: The use of a specific set of programming techniques knows as heuristic or rule-based programming. In this approach human experts are studies to determine what heuristics or rules of thumb they use in solving problems. . . . The program is designed to solve a problem the way that humans seem to solve it.*

Parnas, D.L., “Software aspects of strategic defense systems,” *Communications of the ACM*, **28**, 12 (Dec., 1985), pp. 1326-1335. Also in *American Scientist*, **73**, 5 (Sept.-Oct., 1985), pp. 432-440.

Verificação do programa:

Parte dos esforços da programação moderna é dedicada a testar a reparar erros. No entanto isto não significa que os programas não vão falhar. Por outro lado, as verificações reduzem o tempo de procura de erros, mas não os eliminam por completo.

Ambientes de trabalho e ferramentas:

Estes podem ajudar o programar a corrigir erros de sintaxe e erros de semântica, bem como ter ferramentas e interfaces que ajudam com trabalho em equipa ou com a organização de ficheiros com apresentações hierárquicas.

*Promising Attacks on the Conceptual Essence*

Refinamento dos requerimentos e protótipos rápidos:

Uma das partes mais difíceis em construir um sistema de software é decidir precisamente o que construir. Sendo então esta uma das partes mais que se for feita de modo errados pode causar problemas no produto final apresentado ao cliente.

De certa forma, uma faz funções mais importante da construção de software é a interação com o cliente. No entanto os cliente nunca tem a certeza do que pretendem do produto final, nem conseguem especificar com detalhes o programa que tem em mente.

Para tal são feitos protótipos rápidos para poder ter uma opinião do cliente em relação ao programa. A partir dos testes feitos pelo lado do cliente, são possíveis ter mais ideias sobre a consistência e usabilidade que são pretendidas dar ao software.

Concluindo, não existe uma Silver Bullet que consigam resolver os problemas de desenvolvimento de software mas tem havido desenvolvimentos que ajudam os programadores e as empresas a reduzir tempo de produção e custo, tornando este trabalho mais fácil e eficiente.