Resenha “No Silver Bullet” - Pedro Henrique Maia Alves

Como citado por *Frederick P. Brooks, Jr. “De todos os monstros que povoam os pesadelos do nosso folclore, nenhum aterroriza mais do que os lobisomens, pois eles se transformam inesperadamente do familiar em horrores.”* Está é a comparação que o autor faz aos softwares que ao longo do tempo se tornam monstros de prazos perdidos, orçamentos estourados e produtos defeituosos. Não existe uma receita de bolo para desenvolver software. O avanço se dará de maneira gradual, disciplinada a partir de boas práticas e pequenas inovações e não de um milagre. O artigo mostra que não apenas não há bala de prata, como também a natureza do software torna improvável que venha um dia existir alguma pois nunca existirá invenções que tragam o mesmo ganho para softwares em produtividade, confiabilidade e simplicidade, igual ao que transistores, e circuitos integrados trouxeram para hardware. Pois o problema desta comparação é que, o desenvolvimento de software não está lento, e sim que o desenvolvimento de hardware é mais rápido. Em nenhuma outra área pode se ganhar performance e desempenho em redução de custos como a de hardware.
Para entender as dificuldades no desenvolvimento de software o autor as divide em duas categorias:

- **Essência -** Dificuldades que estão atreladas a existência de um software.
- **Acidentes -** Aquelas dificuldades que acompanham o desenvolvimento hoje porém não são inerentes ao software.

Conceitos como conjunto de dados, relacionamento entre eles, algoritmos e chamadas de funções, são conceitos abstratos porém precisa e detalhada de um software e é nesta parte que se encontra a dificuldade do software, onde o difícil não é escrever mas sim especificar corretamente, projetar bem e testar os modelos conceituais. Embora ainda erramos a sintaxe ao desenvolver, isso é fácil de perceber e corrigir, o problema mesmo está em projetar mal, não entender o problema do cliente ou escolher algoritmos errados e funções mal escritas. Estes problemas são difíceis e caros para consertar pois afetam o sistema como um todo. Ou seja, o autor conclui que não existe uma bala de prata pois a dificuldade está atrelada diretamente à natureza do trabalho em mexer com abstrações e conceitos complexos. A partir disto ele disserta algumas propriedades inerentes aos softwares:

- **Complexidade -** Softwares são naturalmente complexos porque cada sistema é único. Isso gera dificuldades de comunicação entre membros da equipe, de entendimento do funcionamento, de manter uma visão geral e de aprendizado. Como consequência, surgem falhas, altos custos, perda de integridade e uma curva de aprendizado muito elevada.

- **Conformidade -** Softwares precisam se adaptar a restrições impostas por fatores externos como leis, sistemas já existentes, novos requisitos. Onde isso não pode ser simplificado redesenhando o software.
- **Mutabilidade** - Diferente das outras engenharias, softwares estão sempre sujeitos a mudanças conforme as novas necessidades e isto é facilitado pois softwares não exigem um alto custo físico para serem mutados.
- **Invisibilidade** - Software não é algo físico, é um conceito abstrato, onde só pode ser visto por representações em diagramas e códigos. Tornando-os difíceis de entender.

Os mitos da bala de prata sempre surgem, como promessas de soluções milagrosas que vão melhorar os custos e a produtividade, entretanto o autor sugere que essas melhorias ajudam nos acidentes relacionados ao desenvolvimento de software e não na essência, portanto essas tecnologias podem ajudar mas não solucionam mágicamente os problemas. Portanto entre elas algumas foram úteis como:

- **Linguagens de alto nível -** Avanços evolutivos em modularização, tipos abstratos e hierarquia, ajudou programadores a adotar boas práticas, mas não resolveu os problemas essenciais do design.
- **Paradigma da programação orientada a objeto -** Conceitos como tipos de dados abstratos e hierarquia de classes facilitam lidar com detalhes sintáticos e estruturais, reduzindo dificuldades acidentais, mas não diminui a complexidade conceitual do software.
- **Ambientes e ferramentas integradas -** Melhoria de editores, bibliotecas, bancos de dados e sistemas de arquivos ajudava na organização do trabalho e redução de erros triviais, mas os ganhos eram incrementais.
- **Workstations -** Aumentar velocidade de máquinas melhorava compilação e edição, mas o gargalo no desenvolvimento permanecia, o raciocínio humano.

Brooks acredita que devemos focar em soluções que atacam os problemas essenciais no desenvolvimento de software. Entre essas soluções, ele defende a compra de softwares prontos em vez de construí-los do zero, pois o custo de desenvolvimento está atrelado principalmente à fase inicial. Ao dividir esse custo entre todos os usuários, o preço por pessoa diminui drasticamente. Embora essa seja uma solução mais radical, o mercado de desenvolvimento de software em massa tornou-se uma tendência nos dias atuais.

Brooks também enfatiza a importância do refinamento de requisitos e da prototipagem rápida, já que a parte mais difícil do desenvolvimento é decidir o que construir. Caso essa decisão seja tomada de forma inadequada, todo o sistema é afetado. A prototipagem permite que o cliente teste funcionalidades e avalie a usabilidade, tornando o desenvolvimento mais seguro e eficiente. Ele considera, portanto, falho o desenvolvimento sem protótipos.

Além disso, Brooks defende o desenvolvimento incremental, pois é mais fácil cultivar um software ao longo do tempo do que construí-lo integralmente de uma só vez. Esse método facilita a refatoração e a implementação de mudanças, mantém a equipe motivada, resulta em projetos mais sólidos e entrega ao cliente um sistema minimamente funcional desde as primeiras versões.

Por fim, Brooks ressalta que a chave para melhorar os problemas essenciais do software está nas pessoas. Boas práticas podem transformar projetos ruins em bons, mas projetistas talentosos e criativos são aqueles que produzem sistemas rápidos, simples, elegantes e eficientes, usando menos esforço. Identificar, valorizar e investir na formação desses profissionais é fundamental. Brooks sugere oferecer mentoria, planos de carreira e oportunidades de aprendizado e troca de conhecimento, cultivando esses projetistas, pois é por meio deles que a excelência técnica no desenvolvimento de software será alcançada.

Ao longo da leitura do artigo que foi publicado no ano de 1986, pude perceber que muitos conceitos propostos se refletem nas práticas hoje em dia como o desenvolvimento incremental e iterativo, a prototipagem e refinamento de requisitos rápida e a entrega gradual de sistemas funcionais, inspiram o que hoje conhecemos como metodologias ágeis e a prática MVP ao cliente. Além disso, a defesa de Brooks em comprar software em vez de construir tudo do zero previa modelos modernos de SaaS amplamente usados nos dias de hoje. Assim, sua reflexão continua atual, pois nos lembra que não existem atalhos milagrosos: a qualidade do software nasce de boas práticas, de uma visão clara de requisitos e, sobretudo, da competência das pessoas envolvidas.