O artigo faz uma analogia, comparando um sistema á um lobisomem, algo que parece comum, familiar mas rapidamente se transforma em algo terrível. No software, essa “transformação” pode ter diversos nomes, desde cronogramas perdidos, à orçamentos estourados e produtos falhos. No entanto, ao contrário da lenda, o artigo reforça que não há uma “bala de prata”, que impeça um projeto de se tornar um caos completo.  
  
E não há nenhum indício de que exista para software, as mesmas soluções absolutas que muitas vezes revolucionaram os problemas em hardwares. O autor faz uma relação entre o progresso que o hardware teve, desde sua criação, e o quão rápida essa evolução parece (e de fato é, quando temos em conta que nenhuma outra tecnologia ao longo da história apresentou a mesma evolução e o custo-desempenho dos hardwares), se comparada com o software.

Ainda no início do artigo, Frederick Brooks então decide analisar e separar as dificuldades enfrentadas nessa tecnologia por dificuldades essenciais e ambientais. As dificuldades *essenciais* estão ligadas à natureza do software, e entre elas estão a complexidade, conformidade, mutabilidade e invisibilidade do software.

O fator complexidade, está diretamente ligado ao fato de que sistemas de software são muito mais complexos do que grande parte das coisas produzidas pelo ser humano, a julgar principalmente pelo nível de abstração. Muitos dos problemas clássicos de desenvolvimento de software são derivados da **complexidade** e seu crescimento não-linear com o aumento de tamanho, como a dificuldade de comunicação entre os membros da equipe, o que leva a falhas de produtos.   
  
 A área de software e projetos, não é a única a lidar com a complexidade dos objetos. É difícil fazer o software atender a interfaces de uso muito diferentes. Cada pessoa, pensa de um jeito diferente sobre como usar aquele software, que deve ser adaptável **conforme** o sistema ou instituição onde será utilizado. **Mutabilidad**e consiste na gestão das mudanças, já que diferente de outras construções, o software está sujeito à alterações o tempo todo. O feedback em tempo real é um grande fator conjunto com a mutabilidade, pois a necessidade de melhoria é percebida de acordo com o uso. Essa gestão de mudanças, mantendo a qualidade, é um grande desafio no dia a dia.

A falta de uma representação visual do produto final do software, também é uma dificuldade inevitável. Então desde a projeção até a entrega dos artefatos, é difícil acompanhar os gaps gerados. As técnicas atuais de visualização ainda não resolvem plenamente essa dificuldade.  
  
 Os erros acidentais podem surgir tanto de erros essenciais quanto de outros erros acidentais, que, em cadeia, acabam gerando novos erros. À primeira vista, pode parecer complexo, afinal, o que é acidental ocorre de forma imprevisível. No entanto, erros e avanços passados podem contribuir tanto para a prevenção quanto para a correção desse tipo de falha.

O artigo mostra que, por exemplo, ao usar linguagens de alto-nível o programa tem uma grande parte de complexidade acidental, e que ainda não é o suficiente para resolver a existência de problemas acidentais.A orientação a objetos, permite uma organização mais clara e visual do software, reduzindo parte das dificuldades acidentais. Porém, segundo Brooks, essa abordagem melhora apenas a expressão do design, sem simplificar a complexidade profunda do software.   
  
 Outra ferramenta citada, foi a inteligência artificial, que é tida para algumas pessoas como a atual “bala de prata” do software, no entanto, o autor argumenta que, embora a IA seja útil para tarefas específicas, ela não oferece uma solução universal e que pode facilitar a expressão do conhecimento, mas não resolve a complexidade do software.  
  
 Brooks enfatiza que um dos maiores desafios do desenvolvimento de software é definir precisamente o que deve ser construído. Ele propõe a prototipagem rápida como uma solução para refinar os requisitos junto aos clientes, permitindo que experimentem uma versão inicial do sistema, ajustando suas expectativas e contribuindo para uma definição mais clara e realista do projeto final, como funcionam por exemplo, as metodologias ágeis.   
  
 Concluindo, o artigo dá a entender, que apesar dos avanços, nenhum desenvolvimento isolado garante ao software os saltos mágicos de desempenho observados no hardware. E que o progresso vem de ataques conceituais promissores, não de soluções milagrosas.Brooks defende uma abordagem disciplinada e paciente, que se relaciona com evolução da medicina. E enfatiza que, apesar da ausência de soluções mágicas, é possível alcançar melhorias significativas em produtividade e qualidade na engenharia de software.