Resumo do artigo - No Silver Bullet: Essence and Accidents of Software Engineering

Sungwon Yoon
Nro USP 9822261

O lobisomem tem a característica de se transformar de uma aparência familiar de um ser humano em uma criatura assustadora, enquanto a bala de prata é conhecida como a única arma capaz de matar o lobisomem. Da mesma maneira, os projetos de software também podem ter uma aparência simples e correto, mas devido a diversos problemas, pode acabar se tornando em uma criatura defeituosa. Diante disso, são esperadas soluções que possam reduzir os custos do desenvolvimento rápido quanto fazemos com o hardware. Mas a experiência dos últimos 10 anos fez perceber que essa bala de prata não existe.

O software abriga complexidade e envolve várias questões — dificuldades essenciais e dificuldades acidentais. Embora não haja um único método de desenvolvimento, tecnologia ou gerenciamento que possa resolver esses problemas de uma só vez, a complexidade essencial pode ser melhorada através de inovação contínua.

Tem que ser difícil? - Dificuldades Essenciais

O autor descreve as dificuldades essenciais devido a propriedades inerentes dos sistemas de software modernos.

Complexidade: As entidades de software são muito mais complexos do que qualquer outra construção humana. Não há partes iguais, e, se houver, as partes similares são todas criadas como sub-rotinas, resultando na existência de apenas elementos diferentes. Em outras palavras, o aumento do tamanho do software significa que a quantidade desses elementos diferentes aumenta, e esses elementos diferentes interagem de maneira não-linear com outros elementos. Essa complexidade, sendo a essência de software, faz com que não seja possível simplificar problemas da mesma forma que as leis naturais, como as leis da física. Isso não só torna as questões técnicas desafiadoras, mas também complicações em termos de gerenciamento.

Conformidade: Projetos de software envolvem a colaboração de várias pessoas, e isso traz desafios devido às diferenças de estilos de desenvolvimento individuais e nas interfaces resultantes dessas diferenças. E simplesmente redesenhar o software não é a solução.

Mutabilidade: O software sempre está sujeito às mudanças. Ao contrário de infraestruturas físicas, ele não está restrito por limitações físicas, o que permite uma flexibilidade maior. Todo software bem-sucedido acaba oferecendo mais do que as funcionalidades originais, de acordo com as demandas das pessoas. Além disso, se o hardware é alterado, o software também deve ser modificado para se adequar a essas mudanças.

Invisibilidade: O software não é visível e também não pode ser visualizado, pois a essência do software não existe no espaço físico. É possível de usar diagramas, mas são necessários vários para descrever o software e ainda assim não é capaz de fazer de forma completa.

Avanços Passados Solucionados Dificuldades Acidentais

Idiomas de alto nível: Contribui significativamente para a produtividade, confiabilidade e simplicidade do software, fornecendo as estruturas abstratas, permitindo a implementação das configurações desejadas e eliminando a complexidade que não fazem inerentemente parte do programa. No entanto, por consequência, a taxa de avanço de construtos está diminuindo progressivamente e a exigência de domínio de ferramentas está aumentando.

Compartilhamento de tempo: O compartilhamento de tempo melhora significativamente a produtividade e a qualidade, embora não seja tanto quanto as linguagens de alto nível. A redução do tempo

de resposta do sistema ajuda a não interromper o fluxo de pensamento. No entanto, não há benefícios perceptíveis além do limite humano notável (100ms).

Ambientes de programação unificada: As bibliotecas integradas, formatos de arquivos unificados e pipes e filtros garantem que a estrutura conceitual permaneça consistente e reutilizável, mesmo que o ambiente de operação fosse diferente.

Esperanças para a Prata

Ada e outros avanços de linguagem de alto nível: A linguagem Ada não apenas reflete melhorias evolutivas nos conceitos linguísticos, mas também implementa recursos de design moderno e a modularização. No entanto, não é uma bala de prata, pois é apenas uma outra linguagem de alto nível, cujos problemas foram citadas acima.

Programação orientada a objetos: Os tipos abstratos de dados e os tipos hierárquicos representam um verdadeiro avanço na tecnologia de construção de software, eliminando as dificuldades acidentais de alto nível e permitindo expressões de design de ordem superior. No entanto, eles não podem fazer mais do que eliminar as dificuldades acidentais na expressão do design. A complexidade inerente ao próprio design é inevitável.

Inteligência artificial: A maioria das tarefas é específica para um problema e envolve criatividade na abstração.

Sistemas especializados: Um sistema especialista é um programa que contém um mecanismo de inferência generalizada e uma base de regras, pega dados de entrada e suposições, explora as inferências deriváveis da base de regras, produz conclusões e conselhos, e se oferece para explicar seus resultados refazendo seu raciocínio para o usuário. No entanto, há muitas dificuldades na implementação desses sistemas: encontrar boas especialistas que possuam um entendimento claro do motivo do trabalho e que sejam autoanalíticos, e desenvolver uma tecnologia eficiente para extrair o conhecimento em uma abordagem baseada em regras.

Programação "automática": Geração de programas a partir de uma declaração das espeicifações do problema. É difícil de ser generalizado uma vez que frequentemente não possuem propriedades bem-definidas e cada problema tem exceções específicas.

Programação gráfica: Representção de software em design gráfico. No entanto, os softwares são difíceis de ser visualizaods, as formas de representação gráficas existentes hojem em dia não são capazes de fornecer uma visualização detalhada ("píxeis").

Verificação do programa: Na programação moderna, é dedicado muito esforço a testes e correção de bugs, levando a buscar um método de verificação eficaz que garanta o funcionamento adequado do programa. No entanto, contar apenas com provas matemáticas não é suficiente, e surge uma contradição sobre como encontrar um método para verificar os próprios métodos de verificação.

Ambientes e ferramentas: Usar banco de dados unificado ou editores inteligentes. Aumenta a confiabilidade e produtividade, porém devido a sua natureza, o ganho é pouco.

Estações de trabalho: Aumentar a velocidade de compilação e de memória, mas o ganho não é grande.

Ataques Promissores à Essência Conceitual

A maior parte do tempo é gasto na organização conceitual das tarefas. Então devemos atacar à essência do problema de software, a formulação dessas estruturas conceituais complexas.

Compra versus compilação: É mais barato comprar do que construir de novo

Requinte de requisitos e prototipagem rápida: A parte mais difícil da construção de um sistema de software é decidir precisamente o que construir. Portanto, a função mais importante que o desenvolvedor de software executa para o cliente é a extração e o refinamento iterativo dos requisitos do produto, porque nem o cliente sabe direito o que ele quer.

Desenvolvimento incremental: Cresça, não construa software.

Grandes Designers: Um bom design é feito por um bom designer. É importante desenvolver uma maneira de desenvolver bons designers.