**Big Ball of Mud – Um Olhar Realista sobre Arquitetura de Software**
Aluno: Gabriel Chagas
Engenharia de Software – 4º Período

O artigo *Big Ball of Mud*, escrito por Brian Foote e Joseph Yoder, oferece uma análise diferenciada da arquitetura de software ao discutir um tema que, embora frequente na prática, costuma ser pouco valorizado no meio acadêmico: a existência de sistemas desorganizados e mal estruturados. Em vez de tratar essa realidade apenas como um erro ou falha, os autores a reconhecem como consequência natural de pressões e restrições presentes no desenvolvimento de software.

A chamada “Grande Bola de Lama” é apresentada como o padrão arquitetural mais comum na indústria. Trata-se de sistemas que crescem de forma improvisada, acumulando código de baixa qualidade, acoplamentos desnecessários e soluções pontuais. O que torna o artigo interessante é que, ao invés de condenar esse fenômeno, ele o analisa como um resultado inevitável de fatores como prazos curtos, falta de experiência da equipe, mudanças constantes de requisitos e limitações orçamentárias. Assim, o texto traz uma visão pragmática, mostrando que nem sempre é possível aplicar metodologias e padrões ideais no mundo real.

A taxonomia descrita pelos autores, composta por padrões como *Throwaway Code*, *Piecemeal Growth* e *Keep it Working*, demonstra como essas situações surgem naturalmente no desenvolvimento. O conceito de *Throwaway Code* é particularmente familiar para quem já trabalhou em projetos acadêmicos ou profissionais. No meu caso, lembro de um script criado rapidamente para atender a uma demanda temporária em um projeto de pesquisa. Mesmo sem a intenção de ser permanente, ele continuou sendo utilizado por mais de um ano, tornando-se parte da rotina da equipe. Esse exemplo mostra que, muitas vezes, o código improvisado acaba assumindo um papel essencial, mesmo sem ter sido planejado para isso.

Outro ponto importante é o *Piecemeal Growth*, que descreve sistemas que evoluem de forma incremental, com adições feitas pouco a pouco conforme novas demandas surgem. Essa metáfora se assemelha ao crescimento urbano desorganizado: cada construção pode fazer sentido isoladamente, mas o conjunto final perde coesão. Vivi essa situação em um projeto de pesquisa acadêmica no qual o sistema, criado inicialmente para coleta de dados, passou a incorporar funções de análise e relatórios. Cada etapa parecia lógica, mas ao final o software se tornou complexo, acoplado e difícil de manter.

O artigo também apresenta estratégias de mitigação. Entre elas, destaco *Sweeping it Under the Rug*, que consiste em isolar partes problemáticas do sistema por meio de interfaces mais limpas, permitindo melhorias graduais. Em meu estágio, tive contato direto com essa abordagem. A empresa utilizava um sistema legado que não podia ser interrompido, mas que apresentava diversos problemas arquiteturais. A solução foi criar camadas de abstração, possibilitando refatorações sem afetar o funcionamento do serviço. Essa experiência mostrou como as soluções propostas pelos autores são aplicáveis no dia a dia.

Um aspecto relevante do texto é a crítica à formação acadêmica tradicional. Na universidade, o foco está geralmente em padrões ideais, boas práticas de código e

metodologias formais, como RUP e Clean Code. Embora importantes, esses ensinamentos muitas vezes não preparam o aluno para lidar com sistemas que já nascem ou se tornam uma “bola de lama”. Os autores citam a famosa frase de Kent Beck: “Make it work, make it right, make it fast”. Essa sequência mostra uma lógica prática: primeiro garantir que o sistema funcione, depois torná-lo correto e, por último, otimizá-lo.

O conceito de *Reconstruction* também merece destaque. Existem situações em que o sistema se torna tão problemático que a única saída é reconstruí-lo do zero. Entretanto, os autores lembram que mesmo nesse processo é fundamental preservar os aprendizados adquiridos com o sistema antigo. Essa perspectiva reforça o caráter iterativo do desenvolvimento de software e a importância de aprender com a experiência, mesmo quando esta envolve erros.

Além das questões técnicas, o artigo aborda também o lado organizacional. É interessante a observação de que, em algumas empresas, programadores que conseguem navegar e manter sistemas confusos acabam sendo mais valorizados do que arquitetos que se preocupam com a estrutura do sistema. Isso revela uma dinâmica de mercado que valoriza a capacidade prática de resolver problemas imediatos, ainda que o preço a longo prazo seja maior.

Do ponto de vista profissional, a leitura de *Big Ball of Mud* amplia a percepção sobre o papel do engenheiro de software. Fica claro que, ao longo da carreira, não trabalharemos apenas com sistemas exemplares em termos de arquitetura. Muito pelo contrário, será necessário lidar com legados complexos e imperfeitos. Nesses casos, a habilidade de compreender, manter e gradualmente melhorar sistemas desorganizados pode ser tão valiosa quanto o conhecimento de novas tecnologias.

Refletindo sobre minha trajetória, percebo que no início dos estudos eu estava bastante focado em padrões e boas práticas. Com o tempo e com a experiência em estágios e projetos, passei a reconhecer que soluções imperfeitas, mas funcionais, também têm seu valor. O artigo reforçou essa visão ao mostrar que a busca por uma arquitetura “ideal” nem sempre é viável, e que a adaptação às circunstâncias é parte essencial da prática profissional.

**Conclusão**

A leitura de *Big Ball of Mud* é essencial para estudantes de Engenharia de Software porque oferece uma visão honesta e pragmática da profissão. O texto mostra que sistemas imperfeitos não representam necessariamente falhas, mas respostas a restrições concretas. Mais do que criticar más práticas, os autores fornecem estratégias para lidar com essas situações e gradualmente transformá-las.

Assim, o artigo contribui não apenas para a formação técnica, mas também para a construção de uma postura profissional madura. Reconhecer que a “bola de lama” faz parte da realidade, e aprender a lidar com ela de maneira construtiva, é um passo fundamental para qualquer futuro engenheiro de software.