PUC Minas

**Pontifícia Universidade Católica de Minas Gerais**

**I**nstituto de **C**iências **E**xatas e **I**nformática - **ICEI**

Engenharia de Software

Projeto de Software

Professor: Dr. João Paulo Aramuni

09 set. 2024

# "*Big Ball of Mud*": Desvendando a Arquitetura Caótica de Software

João Pedro Silva Braga

FOOTE, B.; YODER, J. **Big Ball of Mud.** Urbana-Champaign: University of Illinois, 2003.

Em um mundo ideal, todo software seria um exemplo de elegância arquitetônica, com código limpo, modularizado e fácil de entender. Contudo, a realidade do desenvolvimento de software frequentemente se desvia desse ideal, e muitos sistemas se assemelham mais a um emaranhado de código difícil de navegar e manter. É nesse contexto que se insere o artigo "*Big Ball of Mud*", de Brian Foote e Joseph Yoder, uma crítica perspicaz sobre a popularidade da arquitetura de software caótica, apelidada pelos autores de "Grande Bola de Lama".

O artigo argumenta que, apesar da proliferação de padrões de projeto e arquiteturas de alto nível, a arquitetura de software dominante na prática é frequentemente caracterizada pela falta de estrutura e planejamento, resultando em sistemas complexos e difíceis de manter. A principal pergunta de pesquisa que os autores buscam responder é: por que o "*Big Ball of Mud*" é tão prevalente, e quais lições podemos extrair dessa realidade?

A priori, as principais forças que levam a produção de “grandes bolas de lama”, isto é, os fatores que atuam como forças contrárias à construção de uma arquitetura de software sólida e bem definida, são apresentadas e discutidas, sendo elas:

- Tempo: A pressão por entregas rápidas e o cumprimento de prazos apertados frequentemente relega a arquitetura a um segundo plano. A urgência em lançar um produto no mercado competitivo muitas vezes impede a análise aprofundada das implicações a longo prazo das decisões arquiteturais;
- Experiência: A falta de experiência com o domínio do problema, especialmente no início do desenvolvimento, limita a capacidade de criar uma arquitetura eficiente e robusta. A curva de aprendizado da equipe, a

rotatividade de profissionais e a própria complexidade do problema influenciam diretamente na qualidade da arquitetura;

- Habilidade: As diferenças de habilidade, experiência e até mesmo de estilo entre os programadores impactam a arquitetura do software. Equipes desniveladas, sem um direcionamento claro e sem incentivo à comunicação, tendem a produzir código inconsistente e de difícil manutenção;
- Complexidade: A complexidade inerente ao problema em si, às vezes agravada pela estrutura da própria organização (Lei de *Conway*), pode se refletir em uma arquitetura confusa e fragmentada. A comunicação deficiente entre as partes do sistema e a dificuldade em adaptar os limites entre os módulos contribuem para o aumento da complexidade;
- Mudança: As mudanças frequentes nos requisitos, muitas vezes imprevisíveis, forçam a equipe a fazer alterações rápidas na estrutura do sistema, sacrificando a consistência da arquitetura em nome da funcionalidade imediata;
- Custo: Investir em uma arquitetura bem planejada requer tempo, recursos e profissionais experientes, o que pode ser visto como um custo inicial alto, especialmente em comparação com soluções rápidas e "sujas". A pressão por resultados imediatos e a dificuldade em mensurar o retorno do investimento em arquitetura contribuem para que ela seja negligenciada.

Além disso, os autores contextualizam o tema, contrastando a busca por arquiteturas elegantes com a realidade do desenvolvimento de software, onde prevalece a desordem. Fazendo analogia com uma favela, ilustrando a natureza orgânica e não planejada desses sistemas. Como também, apresentam seis “padrões” interligados, descrevendo as etapas que levam à formação e perpetuação da arquitetura caótica: *“Big Ball of Mud”, “Throwaway Code”, “Piecemeal Growth”, “Keep it Working”, “Sweeping it Under the Rug” e “Reconstruction”.* Esses padrões são definidos em contraste com os padrões de ciclo de vida.

O padrão “Grande Bola de Lama” (“*Big Ball of Mud*”) aborda uma analogia à construção de favelas: ambos surgem da necessidade de soluções rápidas e baratas, priorizando funcionalidade em detrimento de planejamento e estrutura. A "*Big Ball of Mud*" se caracteriza por código confuso, nomes pouco intuitivos e falta de documentação, tornando a manutenção e o entendimento do sistema difíceis.

Diversos fatores contribuem para esse problema, como a pressão por entregas rápidas, falta de investimento em arquitetura e clientes que priorizam o "barato" e o "rápido". Apesar de ser um anti-padrão, a "*Big Ball of Mud*" persiste por ser uma solução fácil e rápida em fases iniciais de desenvolvimento. Dado essa popularidade, inexoravelmente é chamado de padrão de projeto pelos autores. Além disso, o texto traz a importância da experiência no domínio do problema, de boas ferramentas e comunicação clara na equipe, além de seguir a filosofia "Faça funcionar. Faça direito. Faça rápido" para construir um software funcional, bem estruturado e eficiente.

Ademais, o artigo utiliza a analogia de construções temporárias que se tornam permanentes para ilustrar o conceito de "Código Descartável" ("*Throwaway Code")* em

desenvolvimento de software. Assim como um galpão improvisado que, por falta de tempo ou recursos, acaba integrado à estrutura principal, códigos escritos para testes rápidos ou provas de conceito podem ser incorporados ao produto final, mesmo sem a robustez necessária. A urgência do "funcionar" supera o ideal do "bem feito".

Dois exemplos conhecidos ilustram esse fenômeno: o sistema de inscrição online do PLoP e o próprio Wiki-Wiki Web. Ambos tiveram início como experimentos rápidos e funcionais, apesar do código "sujo" e improvisado. Com o tempo, manutenções e expansões mantiveram esses sistemas em funcionamento, demonstrando como a praticidade e a urgência podem moldar a arquitetura de um sistema, sacrificando, em alguns casos, as boas práticas de programação em nome da funcionalidade imediata.

O conceito de "Crescimento Fragmentado" ("*Piecemeal Growth*") é abordado no desenvolvimento de software, comparando-o ao crescimento orgânico de cidades com o Houston e à capacidade de reconfiguração da Estação Espacial Mir. Assim como essas estruturas evoluem gradualmente, softwares de sucesso também passam por adaptações e expansões para acomodar novas necessidades e um público maior.

No entanto, o texto alerta para os riscos do crescimento descontrolado. A pressão por soluções rápidas pode levar a gambiarras que comprometem a arquitetura original, culminando em um sistema confuso e difícil de manter. Para evitar esse destino, o texto defende um compromisso contínuo com a refatoração e consolidação do código, conciliando as demandas imediatas com a saúde estrutural do sistema a longo prazo. A metáfora do "aprendizado" é utilizada para ilustrar a importância da adaptação e da constante evolução, garantindo que o sistema se mantenha funcional e elegante ao longo de seu ciclo de vida.

Além disso, o artigo destaca a importância de "Manter em Funcionamento" ("*Keep it Working*") como estratégia central na manutenção de software, comparando sistemas críticos a serviços essenciais como redes elétricas e abastecimento de água. Assim como uma cidade não pode parar por causa de um problema na infraestrutura, sistemas vitais de empresas e organizações dependem da disponibilidade e funcionalidade contínua do software.

Priorizar o funcionamento contínuo significa focar em pequenas melhorias e correções constantes, evitando grandes reformulações que podem desestabilizar o sistema. Essa abordagem incremental, similar à técnica de "Crescimento Fragmentado", permite testar e implementar mudanças gradualmente, minimizando riscos e garantindo que o sistema permaneça sempre operacional. Demais, o texto defende a importância de preservar uma versão funcional como ponto de retorno e cita exemplos como a Microsoft e a Nortel, que adotam políticas de "*Daily Build*" ou semanais para garantir a estabilidade e a evolução segura de seus softwares.

A prática de "Esconder a Poeira Embaixo do Tapete" ("*Sweeping it Under the Rug*") é explorada como uma estratégia para lidar com código problemático, traçando um paralelo com a organização de uma casa bagunçada. Assim como isolar um problema não o resolve magicamente, esconder código confuso não o torna menos problemático.

No entanto, essa técnica pode ser útil para conter a desordem e facilitar etapas futuras de refatoração. Ao isolar o código problemático, define-se um perímetro para trabalhos futuros de limpeza e reestruturação. Criar interfaces claras e "fachadas" que abstraiam a complexidade interna permite que o código seja utilizado sem expor sua

estrutura confusa. A importância de não se acomodar com essa solução de curto prazo e a necessidade de, eventualmente, enfrentar o problema de frente para evitar que a "sujeira" se espalhe novamente é enfatizada pelos autores.

A demolição e reconstrução do Fulton County Stadium é utilizada como analogia para o processo de "Reconstrução" ("*Reconstruction*") em desenvolvimento de software. Assim como o estádio se tornou obsoleto e incapaz de acompanhar as novas demandas, sistemas de software também podem chegar a um ponto em que a manutenção se torna insustentável e a única saída é recomeçar do zero.

Diversos fatores podem levar à necessidade de reconstrução, como a obsolescência tecnológica, mudanças drásticas nos requisitos ou um código tão degradado que se torna impossível de recuperar. Embora reconstruir signifique abrir mão do código existente, a experiência adquirida com o sistema anterior é valiosa para a construção de uma nova arquitetura, mais sólida e adaptável. Os autores ressaltam a importância de analisar os acertos e erros do sistema anterior para evitar repetir os mesmos erros e para preservar os aspectos positivos na nova implementação. A reconstrução, portanto, não deve ser vista como uma derrota, mas como uma oportunidade para aplicar o conhecimento adquirido e criar um sistema ainda melhor.

A conclusão do artigo reconhece que a "*Big Ball of Mud*", apesar de ser um padrão a ser aprimorado, é muitas vezes um reflexo das realidades do desenvolvimento de software. A pressão por resultados rápidos e a volatilidade do mercado podem tornar a busca por uma arquitetura perfeita uma utopia.

No entanto, o texto deixa claro que o objetivo não é condenar a *"Big Ball of Mud"*, mas sim incentivar a busca por soluções mais robustas e duráveis. Ao compreender as forças que levam à criação de sistemas confusos, os desenvolvedores podem trabalhar para minimizá-las e construir sistemas mais elegantes e fáceis de manter. A chave para esse processo está no aprendizado contínuo, tanto da equipe quanto da própria estrutura do sistema, permitindo que a arquitetura evolua em conjunto com o software e com o ambiente em que ele está inserido.

Sendo assim, o artigo apresenta uma contribuição valiosa, tendo em vista que aborda um tema frequentemente negligenciado na área de arquitetura de *software*. A pesquisa se destaca pela originalidade e relevância, uma vez que trata de um problema real no desenvolvimento de *software*, frequentemente ignorado em favor de padrões de arquitetura idealizados.

Os autores argumentam de forma convincente que a "*Big Ball of Mud*" não é apenas um conjunto de erros, mas sim um padrão arquitetural que emerge da interação de diversas forças presentes no desenvolvimento de *software*, como prazos apertados, mudanças de requisitos e custos. O uso de analogias com cidades e construções, como a cidade de Houston, torna a leitura mais leve e facilita a compreensão de conceitos abstratos relacionados à arquitetura de *software*.

Além de descrever o problema, o artigo também apresenta soluções e estratégias para lidar com a "*Big Ball of Mud*", como a técnica "*Sweeping It Under the Rug*" e o padrão "*Keep It Working*", demonstrando a preocupação dos autores em fornecer ferramentas para enfrentar o desafio.

Apesar dos pontos fortes, o artigo apresenta algumas limitações. A argumentação seria enriquecida com a apresentação de dados quantitativos que demonstrassem a real extensão do problema. A inclusão de estudos de caso, por exemplo, poderia fortalecer a argumentação e ilustrar as consequências da "*Big Ball of Mud*" em projetos reais.

Outra possível falha reside na discussão superficial de alternativas à "*Big Ball of Mud*". O artigo poderia explorar de forma mais aprofundada desenvolvimento orientado a testes como soluções para a construção de sistemas mais robustos e outras metodologias que mitigam dividas técnicas. Vale salientar também que novas tecnologias e abordagens surgiram no desenvolvimento de software. Uma revisão do tema à luz dessas novas tecnologias e uma discussão sobre seu impacto na arquitetura de *software* seriam extremamente válidas e enriquecedoras.

Portanto, apesar de algumas limitações, "*Big Ball of Mud*" é um artigo fundamental para profissionais da área de Engenharia de *Software*. Ele traz à tona um problema real e frequentemente negligenciado, incentivando a comunidade a buscar soluções para a construção de sistemas mais robustos, compreensíveis e fáceis de manter. Para mais, o artigo apresenta uma contribuição valiosa para a área de arquitetura de *software* ao desmistificar a ideia de que sistemas confusos são necessariamente resultado de má prática profissional.

Dessa forma, a pesquisa abre caminhos para a compreensão das forças que levam à "Grande Bola de Lama", incentivando o desenvolvimento de estratégias para evitar ou mitigar esse problema. Os resultados da pesquisa têm implicações importantes para a prática, pois alertam para a necessidade de se considerar o ciclo de vida do *software* e a importância da manutenção e refatoração contínuas para evitar que sistemas bem estruturados se degradem ao longo do tempo.