Resenha Big Ball of Mud

“Still, this approach endures and thrives. Why is this architecture so popular? Is it as bad as it seems, or might it serve as a way-station on the road to more enduring, elegant artifacts? What forces drive good programmers to build ugly systems? Can we avoid this? Should we? How can we make such systems better?”

O artigo Big Ball of Mud, ou grande bola de lama, termo popularizado por Brian Foote e Joseph Yoder em 1997, analisa se esse padrão - ou antipadrão - de arquitetura deveria ou não existir. Como diz no próprio abstrato do artigo: por que tantos sistemas existentes são arquitetonicamente indistintos e o que podemos fazer para melhorá-los?

O artigo de Foote e Yoder é estruturado em torno de seis padrões que analisam a discrepância entre o que é pregado e o que é praticado. São eles: (1) Big Ball of Mud (grande bola de lama), (2) Throwaway Code (código descartável), (3) Piecemeal Growth (crescimento gradual), (4) Keep It Working (manter funcionando), (5) Sweeping It Under The Rug (esconder sob o tapete) e (6) Reconstruction (reconstrução).

O termo "Big Ball of Mud", também conhecido como “Shantytown”, descreve sistemas de software mal estruturados, desenvolvidos com pouca organização e planejamento. A analogia com uma favela (shantytown) é apropriada, pois assim como esses assentamentos crescem sem um plano definido, esses sistemas de software se expandem com componentes sobrepostos, interdependências complexas e baixa manutenibilidade. Isso leva a um código difícil de entender, modificar ou ampliar. Esse cenário muitas vezes surge devido à pressão por entregas rápidas, resultando em um código funcional, porém de baixa qualidade, que se torna difícil de manter e evoluir. Esse padrão é amplamente considerado um antipadrão, mas frequentemente é uma consequência inevitável em ambientes de desenvolvimento ágeis ou de alta pressão. O artigo também cita Kent Beck, renomado engenheiro de software, destacando que a construção de software segue três fases: "faça funcionar, faça certo e faça rápido". Ou seja, o foco inicial deve ser a funcionalidade; depois, a estrutura adequada do sistema deve ser priorizada, após a compreensão das peças necessárias para resolver o problema; e por último, a otimização de desempenho deve ser feita, com base no aprendizado sobre a solução. Dessa forma, é possível também considerar o aspecto de custo na construção do software.

O padrão Throwaway Code refere-se ao desenvolvimento rápido de código com o objetivo inicial de ser temporário, para testar ideias, criar protótipos ou resolver problemas específicos rapidamente, é caracterizado por sua falta de estrutura e documentação adequada. Idealmente, o código seria descartado assim que a funcionalidade temporária fosse atingida e uma solução mais robusta fosse desenvolvida posteriormente. No entanto, na prática, esse código muitas vezes persiste e se torna parte permanente do sistema. Essa permanência acontece por diversos motivos, como falta de tempo para refatorar ou escrever código adequado, restrições de recursos, ou até a percepção de que, uma vez que o código está "funcionando", ele não precisa ser reescrito. Com o tempo, esse código descartável tende a crescer e a acumular problemas,

como a falta de manutenção e a introdução de erros, especialmente quando novos requisitos começam a forçar adaptações em uma arquitetura já comprometida.

O conceito de Piecemeal Growth, ou crescimento gradual, refere-se ao desenvolvimento incremental de sistemas. Assim como uma cidade cresce sem planejamento centralizado, sistemas de software podem crescer de maneira desordenada, conforme novas funcionalidades são adicionadas sem considerar a arquitetura original. Embora essa abordagem permita flexibilidade, ela também pode levar a um sistema que se degrada ao longo do tempo. O artigo cita como exemplo a construção da cidade de Brasília, que é exatamente o contrário do que esse padrão significa, enquanto a capital brasileira foi construída com uma visão unificada e clara do que deveria ser, sistemas que seguem o padrão de crescimento gradual muitas vezes evoluem sem um plano geral estruturado, resultando em uma estrutura que pode se tornar fragmentada e difícil de manter ao longo do tempo. Esse contraste ilustra as vantagens e desvantagens de cada abordagem: o planejamento centralizado oferece coesão e previsibilidade, enquanto o crescimento incremental pode fornecer flexibilidade, mas frequentemente à custa de uma organização eficaz e da consistência. Com o tempo, sistemas que seguem o padrão de Piecemeal Growth podem sofrer de problemas de escalabilidade, dificuldades de manutenção e uma arquitetura cada vez mais complexa e fragmentada.

O padrão Keep It Working destaca a necessidade de preservar a funcionalidade do sistema a todo custo, mesmo que isso signifique adiar melhorias mais profundas na arquitetura. Este padrão é amplamente aplicado em sistemas que precisam operar continuamente, como sistemas críticos de empresas. A lógica aqui é que a estabilidade e a operação contínua são mais importantes que refatorações profundas, pois derrubar o sistema para reestruturá-lo pode ter consequências catastróficas. A estratégia sugere fazer pequenas correções incrementais sem interromper a funcionalidade geral. No entanto, essa abordagem pode levar a uma gradual deterioração estrutural se os problemas maiores não forem resolvidos a tempo.

O padrão Sweeping It Under the Rug refere-se à prática de esconder problemas, ao invés de resolvê-los adequadamente. É comum em projetos onde o tempo é escasso e as soluções rápidas são preferidas, mas isso cria o que é conhecido como "dívida técnica". Os desenvolvedores muitas vezes isolam ou encapsulam o código problemático, impedindo que ele interfira imediatamente no funcionamento do sistema, mas sem realmente resolver a causa raiz. Essa abordagem resolve o problema no curto prazo, mas com o tempo, o código mal projetado e mal estruturado cresce, tornando o sistema cada vez mais difícil de manter e expandir.

O conceito de Reconstruction propõe que, quando a arquitetura de um sistema está tão comprometida que não pode mais ser mantida, a única solução viável é derrubar a estrutura existente e começar do zero. Essa abordagem pode parecer radical, mas em certos casos é a única maneira de restaurar a integridade do sistema e evitar o colapso. O custo e o tempo envolvidos na reconstrução são altos, mas podem ser compensados pela

longevidade e eficiência do novo sistema. É importante que a reconstrução leve em consideração os erros do passado, permitindo que a nova arquitetura evite os problemas que levaram à degradação do sistema original.