Resenha Descritiva: Explorando ‘Big Ball of Mud’ e Seus Padrões de Software

O artigo “Big Ball of Mud”, escrito por Brian Foote e Joseph Yoder, explora e justifica a popularidade do que eles chamam de “anti-padrão” nos sistemas de software. O termo “Big Ball of Mud” (grande bola de lama) é utilizado para descrever sistemas de software com arquitetura desorganizada e difícil de entender “*A BIG BALL OF MUD is haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle.”. (FOOTE; YODER, 1997, p. 2).*
Os autores identificam vários padrões que contribuem para a formação desses sistemas caóticos, oferecendo uma análise detalhada das razões por trás de sua prevalência e das consequências para o desenvolvimento e manutenção de software. Estes padrões são BIG BALL OF MUD, THROWAWAY CODE, PIECEMEAL GROWTH, KEEP IT WORKING, SWEEPING IT UNDER THE RUG, RECONSTRUCTION.

Uma analogia interessante feita pelos autores é a comparação dos sistemas “Big Ball of Mud” com favelas. Eles descrevem as favelas como assentamentos construídos com materiais comuns e baratos, utilizando mão de obra relativamente não qualificada. A construção e manutenção dessas moradias são intensivas em mão de obra e requerem uma ampla gama de habilidades, mas há pouca especialização. Da mesma forma, muitos sistemas de software são construídos com ferramentas primitivas e infraestrutura inadequada, resultando em um crescimento desorganizado e problemas que se espalham por todo o sistema.

Os autores destacam que, assim como as favelas, esses sistemas de software surgem para atender a uma necessidade imediata, utilizando os recursos disponíveis. No entanto, a falta de planejamento e investimento em infraestrutura leva a uma manutenção difícil e a uma arquitetura que parece inatingível.

Um dos principais fatores que contribuíram para a popularização deste antipadrão é o custo. Na era atual, onde a redução de gastos e a entrega rápida são altamente valorizadas, a implementação de uma arquitetura bem planejada é frequentemente vista como um luxo dispensável. Desenvolver uma arquitetura robusta requer um investimento significativo de tempo e dinheiro, algo que muitos clientes e desenvolvedores não estão dispostos a comprometer. Este investimento é um “contrato” a longo prazo, que muitas vezes não se alinha com a urgência de obter resultados imediatos. Como os autores destacam: “*You need to deliver quality software on time, and under budget.*” (FOOTE; YODER, 1997, p. 6).

Além disso, a pressão para lançar produtos rapidamente favorece soluções rápidas e temporárias, que podem ser implementadas com menos recursos e em menos tempo. Para os clientes, um sistema que atende às suas necessidades imediatas é mais atraente, mesmo que isso signifique comprometer a qualidade a longo prazo.

Para os desenvolvedores, entregar projetos rapidamente permite que eles assumam novos trabalhos e aumentem seus ganhos.

Foote destaca também que a criação de uma arquitetura de software bem feita exige habilidades específicas e um investimento significativo de tempo. No entanto, muitos desenvolvedores não estão dispostos a dedicar tempo para aprender essas habilidades, preferindo soluções rápidas que atendam às demandas imediatas. Aqueles que possuem as habilidades necessárias muitas vezes não as aplicam, pois a prática de desenvolver uma arquitetura robusta não é comum na indústria. A arquitetura de software é frequentemente deixada para depois, sem prioridade, devido à pressão para entregar resultados rapidamente e dentro do orçamento.

Kent Beck propõe uma abordagem pragmática que poderia resolver o problema do antipadrão para o desenvolvimento de software em três etapas: "Make it work, make it right, make it fast". Primeiro, o foco é garantir que o software funcione e atenda aos requisitos. Em seguida, ajusta-se a arquitetura para suportar a funcionalidade e, por fim, otimiza-se o desempenho. Essa abordagem destaca um desenvolvimento iterativo e incremental, priorizando funcionalidade e estrutura antes de otimização. No entanto,na prática muitos desenvolvedores ignoram essas etapas devido à pressão por prazos e orçamento.

Os desenvolvedores estão tão habituados ao "dirty code" que até mesmo uma das linguagens mais utilizadas no mundo apresenta uma arquitetura no estilo "Big Ball of Mud". A prática de criar código temporário é amplamente adotada, muitas vezes em resposta à pressão por prazos. No entanto, a ausência de uma reescrita adequada acaba resultando em sistemas “Big Ball of Mud”. O artigo cita dois exemplos: o sistema de registro das conferências EuroPLoP/PLoP/UP e o Wiki-Web. Ambos começaram como soluções rápidas e improvisadas, mas devido ao sucesso inesperado, continuaram em uso muito além do previsto. O sistema de registro, por exemplo, foi criado com código ineficiente e nunca reescrito, ilustrando como soluções temporárias podem se tornar permanentes e contribuir para a desorganização arquitetônica.

O artigo também discute boas práticas que podem ajudar a evitar a formação de sistemas desorganizados. Eles enfatizam a importância de investir em uma arquitetura bem planejada desde o início do projeto. Práticas como a refatoração contínua, a adoção de padrões de design e a implementação de testes automatizados são cruciais para manter a qualidade do software. Além disso, os autores sugerem a importância de uma documentação clara e de uma comunicação eficaz entre os membros da equipe. Essas práticas não apenas ajudam a prevenir a degradação do sistema, mas também facilitam a manutenção e a evolução do software ao longo do tempo. Ao seguir essas boas práticas, os desenvolvedores podem criar sistemas mais robustos, escaláveis e fáceis de entender, evitando assim os problemas associados ao antipadrão.

Em resumo, “Big Ball of Mud” é uma leitura essencial para desenvolvedores e arquitetos de software, oferecendo insights valiosos sobre os desafios da manutenção de software e a importância de uma arquitetura bem planejada. Ele serve como um alerta sobre os perigos de negligenciar a arquitetura em favor de soluções rápidas e temporárias, e como essa negligência pode comprometer a qualidade e a sustentabilidade dos sistemas de software a longo prazo.