Resenha do Artigo Big Ball of Mud

Júlia Moreira Nascimento
Projeto de Software

September 9, 2024

Em um mundo ideal, todo sistema seria um exemplo ideal de padrões de arquitetura de alto nível.

Uma *big ball of mud* é uma estrutura perigosa e cheias de gambiarras, com o seu código mostrando sinais de crescimento irregular, repetições, trabalho de reparação extensivo, informações importantes duplicadas, uma estrutura de sistema que nunca foi definida ou que foi definida e foi degradada ao ponto de ser irreconhecível.

Como uma maneira de tentar mapear e examinar os motivos relacionados as diferenças teoricas e práticas relativas as evoluções e implementações de arquitetura, foram definidos seis padrões, *big ball of mud, throwaway code, piecemeal growth, keep it working, sweeping it under the rug e reconstrução.*

Um dos motivos indicados para o porque de um sistema virar uma *big ball of mud* é o software surgir de com o uso de códigos temporários (*throwaway code*). Com o passar do tempo, o código não é modificado, uma vez que funciona, e, ao invéz de ser refatorado, continua a ser usado e tendo seus problemas endereçados.

Um sistema com uma arquitetura inicial bem definida, pode sofrer uma erosão arquitetural devido à mudanças constantes de requisitos, fazendo com que o sistema vire um "pedaço crescido" (*piecemeal growth*) que gradativamente permite que parte do sistema que estã mal estruturada cresçam descontroladamente. Isso faz com que o sistema passe a ter uma manutenção cada vez mais difícil, e, por consequência, mais cara. Isso está relacionado com a estratégia de manter funcionando (*keep it working*), que mantém códigos que funcionam em operação sem ser refatorados, pois eles funcionam, então não precisam ser mitigados.

Uma maneira usada para lidar com softwares envelhecem com arquitetura indadequada é varrer para debaixo do tapete (*sweeping it under the rug*). Em que os problemas são mitigados se cercando as partes decaídas por partes novas e bem planejadas. Em casos avançasos, pode ser necessário a reconstrução total, uma vez que não é mais possível salvar os padrões iniciais.

Alguns fatores contribuem para que até as empresas mais arquiteturalmente consientes gerem uma *big ball of mud.* São eles:

- **Tempo:** pode não haver tempo o suficiente para elaboração de uma arquitetura ideal para longo prazo;
- **Arquitetura ser considerada um risco:** visão de que recursos são melhor gastos se dirigifos a outras questões;

- **Experiência:** falta da experiencia necessária para criara uma arquitetura sofisticada adequada para o sistema;
- **Turnover:**Alto fluxo de entrada e saída de colaboradores nas empresas;
- **Habilidade:** Diferentes níveis de habilidade e especialidade dentro dos desenvolvedores;
- **Complexidade:** a solução em software é muito complexa e/ou mal compreendida;
- **Mudanças:** mesmo que uma arquitetura seja muito bem planejada, podem surgir questões não incluidas nos planos iniciais;
- **Custo:** um investimento em arquitetura não tem um retorno imediato, o que faz com que ele perca prioridade em relação a outros requistos de projeto;

Um ponto interessante levantado pelo artigo, é que a medida que um software entra em etapa de teste com usuários, pode ser necessário a mudança de estrutura de dados e de interface, que prejudicam decisões arquiteturais. Pelo software ser flexível, é geralmente necessário que certas decisões de arquitetura percam seu lugar devido a natureza flexível de um software.

Uma *big ball of mud* pode ser considerado como um anti-padrão, uma vez que possui grande popularidade, sendo uma maneira perversiva e recorrente d3e solucionar problemas de produção de um sistema no contexto do desenvolvimento de software.

Kent Beck observou que a maneira comum de contrução de software pode ser "*Make it work. Make it right. Make it fast.*", o que significa focar na funcionalidade, estruturar o sistema somente depois de entende-lo e optimizar o sistema depois que o problema foi solucionado. Somente depois de todas essas etapas, passa a ser possível fazer com que o sistema seja barato, uma vez que, algumas questões arquiteturais dos sistemas se tornam explícitas somente depois que a fase de desenvolvimento é finalizada.

Uma parte importante de uma *big ball of mud*, são os códigos ruins. [Foote and Yoder 1997] observaram que códigos ruins possuem uma chance maior de sobrevivencia em comparação à bons códigos, já que eles são difíceis de serem alterados e necessitam de desenvolvedores com um nível de conhecimento muito avançado para serem compreendidos.

O software é um produto evolutivo. Com o passar do tempo e com a mudança ou ampliação do público alvo, pode ser que uma nova gama de requisitos surjam. Esses novos requisitos não foram considerados na concepção inicial do sistema. A tendencia nessa fase é que haja uma diminuição da prioridade de elegância arquitetural ou a reparação rápida do problema. Essa situação se torna perigosa quando o trabalho de refatoramento nunca é feito. Então, a melhor maneira de se tratar melhorias e novas adições de funcionalidades é de forma incremental, continuando a possibilitar o crescimento e o refinamento.

A questão de manutenção de software passa a ser muito delicada já que, geralmente, tem que se manter o sistema rodando a qualquer custo. Sistemas que são colocados offline para manutenção e sobre várias mudanças, aumentam o risco sobre ele, visto que, caso haja algum problema, é muito difícil entender qual nova funcionalidade é a responsável. A melhor maneira de se lidar com situação desse tipo é por eliminação de qual parte do código causou o problema.

Uma questão muito importante de bolas de lama é o *keep it under the rug.* Existem razões, além de preocupações estéticas, orgulho profissional e culpa, para tentar limpar a bagunça de um código. Na medida que partes ruins do código começam a se ploriferar, o código se torna um spaghetti, sendo difícil de entender, reparar e expandir e tende a ficar ainda pior se essa erva daninha de problemas maquiados não seja controlada.

Uma *big ball of mud* pode pararecer muito intimidante a primeira vista, mas quando as áreas de problemas são identificadas e começam a ser mitigadas com uma tática de " dividir para consquistar", o problema pode ser mitigado. Então não existe maneira fácil de sumir com uma bagunça. o melhor caminho é restringir o distúrbio e prepara-lo para refatoração.

A decisão de abandonar um sistema de software quando ele se torna irreparável ou obsoleto, seja tecnicamente ou economicamente. Embora o software seja flexível, novas demandas às vezes exigem mudanças tão profundas que tornam impossível continuar com a arquitetura original, o que pode levar à necessidade de uma reescrita completa. Esse processo pode ser traumático, mas também oferece a oportunidade de renovar o projeto e aprender com os erros do passado.

Descartar um sistema antigo permite reter o design conceitual e aplicar os insights arquitetônicos ao novo sistema, isolando o código ruim e promovendo reutilização de componentes valiosos. Embora a substituição de um sistema possa ser vista como uma derrota, ela pode também ser uma oportunidade para revitalizar o projeto, aplicando lições aprendidas e melhorando a arquitetura subjacente.

No final, o texto reflete sobre a essência da arquitetura de software como um processo de destilar experiência em sabedoria e compartilhá-la. Embora sistemas como o "Big Ball of Mud" (grandes massas de código desorganizado) sejam frequentemente vistos como algo negativo, o texto defende que há razões válidas para que bons programadores os criem, principalmente devido à rapidez com que o mercado de software se move. Estratégias pragmáticas e de curto prazo muitas vezes são eficazes e consideradas de ponta. No entanto, o autor expressa a esperança de que, ao reconhecer as forças que levam ao caos arquitetônico, seja possível aspirar a sistemas mais duráveis e robustos, que permitam aos arquitetos de software dominar seus domínios por anos. O aprendizado contínuo da equipe e a evolução do sistema são fundamentais nesse processo.

A frase "Architecture is a long-term investment" destaca uma verdade fundamental sobre o desenvolvimento de software. Embora o retorno financeiro da arquitetura não seja imediatamente visível nas fases iniciais de um projeto, com o envelhecimento e a evolução do software, esse retorno se manifesta de forma indireta, por meio de um código bem estruturado e funcionalidades estáveis. Isso resulta em uma significativa redução de custos com manutenção e refatoração no futuro.

Embora o artigo explore os motivos pelos quais um sistema pode se transformar em uma "big ball of mud", ele presume que os envolvidos no desenvolvimento já tenham conhecimento prévio sobre os efeitos a longo prazo de suas decisões arquitetônicas. O texto não aborda adequadamente os sistemas legados criados em épocas em que técnicas avançadas e a conscientização sobre a importância da arquitetura ainda eram limitadas. Com a evolução contínua da tecnologia, sistemas mal projetados se tornarão mais comuns, o que fará com que a arquitetura ganhe mais destaque e seja mais valorizada, mesmo sem um retorno imediato.