No artigo, o termo “*The big ball of mud*” ou “a grande bola de lama”, é usado para definir softwares desenvolvidos com uma anti-arquitetura que pode ser considerada casual e aleatória, e que não se dá através do design, mas sim através de conveniência e descaso. Nos é apresentado também, que apesar de arriscada para as soluções, essa arquitetura é a que predomina na prática.  
  
Esse nome, é uma analogia á bolas de lama, que são feitas (e crescem) de forma desordenada, desestruturada, misturando elementos aleatórios em uma grande bagunça; um claro exemplo é a reincidência e quantidade de reparos oportunos e emergenciais que funcionam como um tipo de remendo. Outra coisa bastante comum nesse tipo de arquitetura, são os ditos “códigos espaguete”, códigos altamente acoplados e difíceis de serem lidos, entendidos e modificados. Os “códigos espaguete” foram de certa forma, comparados a favelas, como algo que cresce sem ordenação sendo feitas com material de baixa qualidade e sem nenhuma arquitetura e cujo reparo muitas vezes é improvisado, apenas para manter de pé, ou em termos de *software*, manter funcionando. Muitas vezes, nas áreas de engenharia, um código pode acabar se tornando espaguete pelo prazo apertado, e o custo de manter uma arquitetura funcionando, com bibliotecas e outros recursos.

O artigo ainda apresenta outros 6 padrões que podem colaborar ou até mesmo originar uma bola de lama;

*O Throwaway Code,* um código que é feito com o propósito de ser descartado após o uso, não possui documentação, e nem um design eficiente, mas que pode se tornar uma bola de lama sob o pretexto de “se funciona, pode ser reutilizado”. Dessa maneira, o protótipo é mantido, e o código descartável não é de fato descartado e o sistema que já deveria estar maduro se torna algo grande e fragmentado. A produção de código descartável é uma prática quase universal, e é grande parte das vezes, influenciado pela falta de tempo. É possível também que haja alguma interferência da parte do cliente, que ao ver um código descartável funcionando, impulsione o profissional a seguir por esse caminho já que está entregando o esperado até aquele momento.

Piecemeal Growth caracteriza um crescimento desordenado da parte do código, apontando como exemplo a diferença de cidades que foram completamente planejadas, como Washington DC, e Brasília ou cidades que simplesmente vão crescendo. A grande diferença é que quando algo é criado sub algum modelo de arquitetura, os problemas que podem eventualmente surgir, são de certa forma mapeados. E que com auxílio de um feedback, durante a construção, o resultado final torna-se mais eficiente.

Keep it Working, segue a ideia de que o software funciona como uma cidade, e que uma vez estabelecido, deve ter seus serviços funcionando 24 horas por dia. E da mesma maneira, o cliente pode precisar que o software funcione sem parar, dessa maneira, é mais importante tomar mais tempo e custo no início do projeto e se certificar que o sistema não apresentará falhas que podem comprometer o uso como um todo, do que mantê-lo funcionando com reparos emergenciais.

No trecho sobre Shearing Layers ou Camadas de Cisalhamento, os escritores do artigo sugerem os componentes de um sistema devem ser organizados em camadas, com base na taxa de mudança esperada para cada uma. A ideia é que componentes que mudam com pouca frequência (como a infraestrutura) sejam colocados nas camadas mais baixas, enquanto componentes que mudam constantemente (como a interface do usuário) fiquem nas camadas mais altas. A analogia vem da geologia: as camadas da crosta terrestre se movem em diferentes velocidades.

Varrer para debaixo do tapete, (Sweeping it Under the Rug) segue o pensamento que se você não pode fazer uma bagunça desaparecer, pelo menos você pode escondê-la. Por razões estéticas, ou mesmo por falta de prazo, muitas vezes um programador pode apenas usar truques para esconder os problemas do código, que mais uma vez, não significam que eles não existam ou que não vão causar problemas futuros para o código.   
  
  
Apesar de todas as consequências e problemas, os autores não caracterizam a "Grande Bola de Lama" como algo terrível. Na verdade, é apresentado que circunstâncias, que podem exigir abordagens mais improvisadas para atender demandas críticas. Porém o profissional precisa discernir protótipo de projeto, e realizar refatorações quando necessário.

A arquitetura de software é exibida como a maior aliada para projetos robustos e

bem estruturados e como entender os possíveis caminhos para uma "Grande Bola de Lama" ajudam á evitá-los e optar por uma construção de sistema sustentável.

Em resumo, o artigo de Foote e Yoder oferece uma visão crítica sobre um problema recorrente no desenvolvimento de software e são apontados os riscos provenientes da falta de planejamento e o uso de uma arquitetura monolítica.É ressaltada também a importância da ciência de que seguir pelo caminho mais rápido pode eventualmente atrasar o progresso do projeto como um todo.