

Big Ball of Mud

Recommendation by Bar Biszick

Much of recent systems theory revolves around applying *ideal* software development patterns. Big Ball of Mud (www.laputan.org/mud/), in contrast, is for those of us who live

and work in the real world, where most systems emerge haphazardly from minimally controlled chaos under constrained development conditions.

Sound familiar?

Brian Foote and Joseph Yoder of The Refactory, Inc., believe most systems begin as Big Balls of Mud—with less time, resources, and domain knowledge than is needed to fully understand the problem and do the job right. Yet somehow those systems manage to survive and thrive, often longer than their well-planned counterparts. The site's authors find far more “barely planned” systems in service than those that were “well planned.” The writers believe that the study of such systems is as valid—and probably more useful—than the study of those that succeed by ideal process.

A system that never achieves a managed state is a failed project; we can all agree on that. But even a well-managed project can quickly become unmanageable—devolving into a Big Ball of Mud.

How quickly can things fall apart? For one memorable project our team worked on, it took only six months for us to find ourselves hip deep in mud. We had diligently followed a standard lifecycle model; we had hit all the milestone checkpoints for deliverables and reviews. We had brought excellent work in on time—deploying with only five substantial bugs against design and coding errors. But after we rolled to production, unexpected things began to happen. It was about this time that I stumbled on this Web site and brought it to the attention of my team.

Foote and Yoder explore pattern characteristics of systems that exceed expectations. They use creative analogies, contrasting shantytowns that survive and thrive for generations against the demolition of costly, well-planned civic buildings to illustrate their points. They expose several patterns that provide clear hints your muddy system is destined to fail. You can use these patterns to identify what's gone wrong and apply the corrections shown to succeed in similar systems to get your project back on track.

Surprisingly, they believe there are actually *advantages* to starting with a Big Ball of Mud! Over-planning at the start of a project, the authors argue, can restrict natural creativity, impede discovery of better approaches, and undermine future system flexibility. But in their opinion, how a project *begins* is less important than whether or not it eventually graduates to a managed state.

The trick with a Big Ball of Mud is to see it when it's coming and learn to control it. If your quick and dirty prototype was deemed good enough for production, chances are you're struggling with what the authors call the **throwaway code** pattern. In that case, instability must be systematically driven out through chunk rebuilding.

Most of us work in the **piecemeal growth** pattern (a.k.a. “Iterative Incremental Development”), in which complex development tasks are modularized and progressively added to a system. You're in trouble if you don't pause to consider the effects on other parts of the system. Becoming familiar with the **shearing layers** pattern (which derives from “Software Tectonics” theory) can help. Artifacts of a system naturally migrate to organizational layers—like data schema, system architecture, technology framework, code language, object models, and data—which accommodate change at different rates. Like tectonic plates, the faults between these layers can be stressed by making changes to the wrong layer—or even making changes to the *right* layer without considering the stress consequences to the adjacent layers. Catastrophic change can be avoided by considering the effects of change on these constructs in your project.

Constantly fighting production fires indicates you're in the **keep it working** pattern. You want to resolve

www.laputan.org/mud/

“You can't simply aspire to solve the problem at hand once and for all, because, by the time you're done, the problem will have changed out from underneath you.”

—Brian Foote and Joseph Yoder, *BIG BALL OF MUD*

Reference Point

customer dissatisfaction, so you drop in fast, creative fixes—resulting in a quagmire of undocumented workarounds. There is no time for comprehensive regression testing, so you have no way to determine if the fix itself—or its effect on untested areas—causes a new problem. If you don't step back and take a close look at what you've built, you'll probably end up in the **sweeping it under the rug** pattern. This might be called the "Chernobyl Effect," where, according to Foote and Yoder, a lethal problem is encased in a pretty block of cement. Project teams go dark, hiding system problems from Management and each other. Encouraging team review of all deliverables can help insure that no member feels any part of the system is their proprietary domain—or that its inadequacies are their personal failure.

The end of the line for a system is the **reconstruction pattern** (a.k.a. "Total Rewrite"), and it can happen either as a natural course of events or as an ugly scandal. Management can accept that changing market requirements can render a system unproductive, provided the system has recouped its investment. But if a project in its nascent stages has spun so far out of control that the investment can't be salvaged, you'll have a hard time justifying its replacement—at least not one that Management will trust YOU to build!

Inspired by the concepts laid out in this Web site, our team undertook a serious Architecture Review. We traced the root problems mostly to areas we had no control over—the business culture, changing corporate economics, and the undocumented limitations of a vendor product. Foote and Yoder's insights clarified the perils of the path we were heading down and gave us the courage to admit that given these constraints, a different solution needed to be found.

Letting hack demos and vaporware prototypes become production systems, sweeping fundamental problems under the rug, and creating Byzantine workarounds to keep a system running—these methods ultimately end up as Big Embarrassing Balls of Mud. You must constantly reevaluate your system, protect its most essential parts, identify and

shore up its structural weaknesses, and contain and rebuild its unstable parts.

Sometimes that means quarantining chunks of hazy, undocumented code that are too dangerous to rewrite, so they won't affect any other areas of the application as it continues to grow. Other times it means saying no to customers who want features added that could undermine your system's core functions. Sure, you can cook a steak in a toaster by jury-rigging a splatter pan to the bottom, but the next time you toast a piece of bread, you won't be happy about it. Remembering you built a toaster—not a barbecue pit—will insure that what you built will work for a long time, and consistently well.

Be ruthless in your analysis. Use checkpoints in these patterns to recognize potential failure points. Too often, we're so deep inside the code, or so pressured with deadlines, that we lose sight of the long-term business goals. To avoid this, make Architecture Reviews regular periodic events—not just responses to system crisis. Encourage frank, open discussion in a blame-free atmosphere. Ask the hard questions, and be prepared for the uncomfortable consequences. Recognize that doing otherwise is simply irresponsible.

In the end my team is admired for their courage in deciding to redirect efforts away from their original solution toward a more appropriate model. It's a mistake not to raise questions when we see that we're heading down a dangerous path. Foote and Yoder have given us the missing maps we need to navigate those paths and set the course of our projects back on the path to quality success. STQE

Bar Biszick is a certified CQA and Quality Assurance Systems Analyst for the Enterprise Systems division at SAFECO Insurance Corp. in Seattle, Washington, engaged in spearheading practical quality process management into a legacy IT environment. Prior to this she did testing for MCW Technologies, an independent software development group, while under contract to Microsoft for components and Wizards for Excel 97, Visual Inter-Dev 1.0, VB6, and Access 97.

Learn

SOFTWARE TESTING ESSENTIALS

training.prototest.com



- On the Web
- When it's most convenient for you
- Highly interactive content – templates, graphics, examples, sound files, exercises and labs
- Based on the ProtoTest Software Quality Process (PSQP)
- CSQE Instructor-supported
- Quizzes to assess your progress
- Great for new to intermediate-level testers
- No travel costs
- Try-before-you-buy

training.prototest.com

ProtoTest™

IMAGINE PERFECT SOFTWARE