# CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2014
Doc 2 Big Ball of Mud
Jan 28, 2014

## What Compsci textbooks don't tell you

What don't they tell you?

nearly every sample program in every textbook is a perfect and well-thought-out specimen, virtually no software out in the wild is, and this is rarely acknowledged

## What are the causes of bad Software?

bad programmers, too good programmers, bad laziness, time

## What is the simple fix?

Comments

## What is a Big Ball of Mud?

How many have worked on a Big ball of mud?
What caused it?
What was the impact of the Big ball of mud?

## What Forces Lead to Big Ball of Mud

Time, Cost, Experience, Skill, Visibility, Complexity, Change, Scale

## Patterns

Big Ball of Mud
Throwaway Code
Piecemeal Growth
Keep it Working
Shearing Layers
Sweeping it Under the Rug
Reconstruction

## Big Ball of Mud

You need to deliver quality software on time, and under budget.

Therefore, focus first on features and functionality, then focus on architecture and performance.

## Enemy of Big Ball of Mud

Top down design

Hire good architects

## Problems

Variable and function names
    uninformative

Functions themselves may make extensive use of
    global variables,
    long lists of poorly defined parameters.

The function themselves are
    lengthy and convoluted,
    perform several unrelated tasks.

The programmer’s intent is next to impossible to discern.

We built the most complicated system that can possible work

## Three ways to deal with BIG BALLS OF MUD

Keep it healthy - expansion then refactoring
Throw it away
Live with it

## Extreme Programming Practices

Pair programming
Planning game
Test driven development
Customer part of development team
Continuous integration
Refactoring or design improvement
Small releases
Coding standards
Collective code ownership
Simple design
System metaphor
Sustainable pace

## Throwaway Code

You need an immediate fix for a small problem, or a quick prototype or proof of concept.

Therefore, produce, by any means available, simple, expedient, disposable code that adequately addresses just the problem at-hand.

Why do we need throwaway code?

What the main problem with throwaway code?

## Piecemeal Growth

Users' needs change with time.

Therefore, incrementally address forces that encourage change and growth.

Allow opportunities for growth to be exploited locally, as they occur.

Refactor unrelentingly.

## What is the main problem with Piecemeal Growth?

## Keep it Working

Maintenance needs have accumulated, but an overhaul is unwise, since you might break the system.

Therefore, do what it takes to maintain the software and keep it going. Keep it working.

How do Piecemeal Growth and Keep it Working lead to a ball of mud?

How can we use Piecemeal Growth and Keep it Working and avoid the ball of mud?

Is it advisable to use Piecemeal Growth and Keep it Working?

## Shearing Layers

Different artifacts change at different rates

Therefor

Factor your system so that artifacts that change at similar rates are together

## Why?

Put things that change at different rates in different places?

Example?

## Sweep it Under the Rug

Overgrown, tangled, haphazard spaghetti code is hard to comprehend, repair, or extend, and tends to grow even worse if it is not somehow brought under control.

Therefore, if you can’t easily make a mess go away, at least cordon it off.

This restricts the disorder to a fixed area, keeps it out of sight, and can set the stage for additional refactoring.

## Reconstruction

Your code has declined to the point where it is beyond repair, or even comprehension.

Therefore, throw it away and start over.

"Plan to throw one away, you will anyway"

Fred Brooks

## Problems with Starting Over

Cost

Time

Reintroduce bugs

Few features