



Migrating Legacy Applications To CFCs and OOP

Matt Woodward

cf.Objective() 2006



About Me

- Technical Architect and Lead Developer for Realty InfoLinks in Dallas, Texas
- ColdFusion Developer since 1996
- Adobe Community Expert for ColdFusion (formerly Team Macromedia)
- Also develop in Java, Flex, PHP, and a bit of C# on occasion

Agenda

- Legacy Applications
 - Definition
 - Challenges
 - Business Needs vs. Geek Desires
- Why migrate legacy applications to an OO architecture?
- Strategies for a sane approach to migration and architecture changes



Legacy Applications

- This does not necessarily mean “anything you wrote prior to a month ago”
- To me, legacy applications are ...
 - Applications that have been in service for several years
 - Applications that may have been duct-taped through several revisions over several years
 - Applications that aren't CF 7-ready
 - Possibly the bane of our existence as developers, depending on the application in question

Legacy Applications

- Legacy applications represent a significant investment of time and money
 - Typical developer attitude: “I could have done better!”
 - From a business standpoint if the app does the job and continues to do it well, it’s an important business asset
- Legacy applications are a fact of life!
 - If you do a good job developing an application today, it will be a legacy application in a few years
 - Be kind to those who follow you in the maintenance role
 - Be good stewards of applications that do their jobs day in and day out



Business Needs vs. Geek Desires

- As developers we always want to be using the latest, greatest tools and techniques
- The “suits” look at things completely differently
 - Legacy applications are “bought and paid for”
 - If the application still works, change for the sake of change doesn’t make business sense
- It may NOT make sense to migrate your application to an OO architecture!



Why Migrate?

- If the application is starting to show its age
 - Difficult and expensive to maintain
 - Difficult to interact with other applications, modern technologies
 - “Fragile” or intolerant to change
 - Lack of documentation and/or people who originally wrote it are no longer available
- “Have to” for regulatory or other legal reasons
- If the application no longer meets the business needs

Why Migrate?

- Migration is a business decision, not a technology one!
 - Nieman Marcus: still uses Pick database from the late 1960s (or at least did last I knew)
 - Banks, utility companies—massive applications that still work; more painful to change than not, not much ROI
- Tempting to go for the latest geek sheik, but consider things from a business standpoint
- There should be a payoff and relatively rapid ROI
- Don't forget opportunity cost!
 - If you're migrating a legacy application, you're *not* doing something else with your time



Why Migrate to OO?

- Benefits you've been hearing about throughout the conference
 - More standard way of doing things these days
 - Makes for low-impact maintenance, easier extensibility
 - Opens doors for integration with other technologies
 - Lays groundwork for easier large-scale overhauls to the application



So You've Decided to Migrate

- Great!
- NOW WHAT??!?!?!?
 - My application's been running for 7 years—where the heck do I even start?
 - The task is too daunting. I better just leave it as is and deal with the pain of maintenance.
- Take a deep breath ...

How do we get from this ...



... to this ...



... without looking like this?





Strategies for Migration

- It's likely not as bad as you think
- How do you eat an elephant?
 - Texas answer: with habaneros!
 - "One bite at a time"
- Evaluate the whole picture, grim as it may be
- Then make sensible decisions about where to start and create a realistic migration plan

A Tale of Two Applications

- It was the best of apps, it was the worst of apps
 - And they were both living in the same directory!
- Basic stats and features
 - 1500+ CF files between the two applications, all in one directory
 - Files for each application are distinguished with a prefix (mss_ or bsl_, and in some cases mss_bsl_ ???)
 - Includes in includes in includes
 - Includes + <cfexit> for version control
 - The two applications shared a database as well (300+ tables, not well organized, no foreign key constraints, inconsistent field names for the same field between tables, etc.)

It works, why migrate?

- Moving from version 1.0 to version 2.0 was PAINFUL
- Application starting to crack at the seams
 - Increasingly difficult to make changes without unforeseen side effects
 - Increasingly unstable, more and more anomalies, etc.
- Version 2.5 is on the horizon, and the application won't survive without beginning to move in a new direction
- Moving to new high-availability environment
 - Many changes flat-out necessary to get it working in a new environment, e.g. 800+ instances of *= in queries, which breaks in SQL Server 2005
 - Move to new data center is a good opportunity to re-evaluate



Pick Your Battles

- Where should we start?
 - At the beginning of course
- First we split out the two applications and the database
- Next task: login process & state management
 - Numerous redundant, unnecessary queries that run on every request to manage user state
 - Use of client variables (NO session variables ANYWHERE!)
 - Needed increased security for several reasons

General Grunt Work

- Things that had to be done before we could even *begin* to move to an OO architecture
 - DSN hard-coded throughout the application
 - Had to split the two applications and databases
 - Painful, painful, painful, but necessary
- Now running each of these two apps separately on independent, clustered instances of CF
 - Code at this point was more or less 100% as it was in the old applications, but we had two applications instead of one

State Management

- In need of serious cleanup
 - Mix of userinfo query in Application.cfm + client variables to manage session state
- Database getting hammered
 - “userinfo” query in Application.cfm runs on every request rather than caching the data after login
 - Elimination of client variables would lighten load on database
- No client variables in clustered environment: counterintuitive move?
 - When moving to a clustered environment, using client variables is a common option
 - In our case, sticky sessions are perfectly adequate

Step One: userinfo Query + Client Variables

- These need to be gathered into a session User object
 - Hit database once on login, populate User object, put this object in the Session scope
- 900+ instances of client.XXX throughout the application
 - This stuff is DRUDGE WORK, but there's a payoff on the other side

User Object

- Simple “bean” type object
 - Gathers together the userinfo query and client variables into a neat little package
 - Much easier to use and maintain
 - Hit the database once on login instead of several times per page for client variables
 - userinfo query in Application.cfm was run *on every request!!!*
 - Far easier to change behavior, add new attributes, etc.
- Turned out to be simpler than we thought it was going to be

Login Process

- Opportunity to introduce Mach-II into the environment
 - Mini Mach-II app that just handles login process
 - As we touch other areas of the application to add new features, etc., will migrate these to Mach-II one chunk at a time
- Have to start somewhere!
 - In a live, high-use application, often don't have the opportunity to gut it and start over
- Next on deck is a new subsystem in the application
 - All new code will be done in OO with Mach-II

The Decision to Migrate

- Not always an easy one
 - Significant risk involved with overhauling legacy applications—risk analysis is VERY important!
 - Evaluate each situation thoroughly from all angles
- Ask the right questions
 - Why?
 - What are the risks?
 - What's the ROI? When is the ROI?
 - What's the opportunity cost?
- The answer may be a resounding “YES!”
 - Go into the endeavor with your eyes wide open



General Migration Strategies

- First and foremost, do what's necessary
 - Legal requirements
 - Technology requirements
 - Business requirements
 - Future extensibility requirements
- Identify general opportunities for refactoring
 - Redundant code—rework for better reuse
 - Get logic completely out of your presentation layer
 - Organize and clean house—make the code completely transparent!

Steps for Migrating Large Apps

- Divide the app (even if only conceptually) into subsystems
- The next subsystem you have to modify becomes the first candidate for OO overhaul
- Preserve the functionality of the existing code, just do it using OO practices
- Test, test, test!
 - The rest of the system needs to not know or care about the change insofar as possible
 - This may involve tweaks to existing code
- Repeat until your whole app is OO
 - This may take a long time depending on the size of the app—be patient!



Additional Migration Strategies

- Encapsulating code (even existing logic as is) into objects can pay off big time in the future
 - Encapsulation: “shock absorbers” for your application (Hal Helms)
 - Low-impact maintenance and extensibility
- A good MVC framework can help
 - Consistent way of doing things that anyone familiar with the framework can pick up

Creating a Roadmap

- Measure twice, cut once
 - Spend a LOT of time thinking and planning
 - On mission critical applications be conservative; don't try to change too much at once
- Be realistic
 - On a large application resign yourself to the fact that it may be “spaghetti with meatballs” (CFCs) for a while
 - Idealism != reality

Summary

- It's all about the thought process
 - ... and not panicking!
- Think about the business needs first and foremost
 - What's are the risks?
 - What's the payoff/ROI?
 - What's the opportunity cost?
- Take it one step at a time (subsystem by subsystem)
- Go into everything with your eyes wide open

Questions?

- E-Mail Me: mpwoodward@gmail.com
- Skype the ColdFusion Weekly Podcast!
 - Skype User: cfweekly
 - Phone: 469-293-3820
 - Come record a question here and we'll put it on the podcast!