The Philosophy of Cocoa : Small is Beautiful and Lazy is Good

I believe a programming renaissance is upon us - and Cocoa, Apple's high level object-oriented framework is at the heart of it. By wrapping complexity inside of easy to use objects, Cocoa frees the application developer from the burden of the minutiae that so often drives developers crazy. Instead, one can focus on what's special about their application, and in a few lines of code, create a complete OS X application that interoperates with all the other OS X applications seamlessly. I should also mention that you get AppleScriptability for very little additional effort.

I've been living and breathing Cocoa and its various previous incarnations for 14 years now - and have noticed that my applications are getting more features with smaller amounts of code each year. This article will explore some of the truisms and gems hard earned by hanging in the trenches lo these many years.

Small is Beautiful

It's no coincidence that we use the term "architecture" for the overarching structure of an application. I received my baccaloreate in classical Architecture in the '70's when the visionaries of the time were rebelling against the huge concrete boxes that crushed the human scale and spirit. E. F. Schumacher, in "Small is Beautiful - A Study of Economics as if People Mattered":

"What I wish to emphasise is the duality of the human requirement when it comes to the question of size: there is no single answer. For his different purposes man needs many different structures, both small ones and large ones, some exclusive and some comprehensive. Yet people find it most difficult to keep two seemingly opposite necessities of truth in their minds at the same time. They always tend to clamour for a final solution, as if in actual life there could ever be a final solution other than death. For constructive work, the principal task is always the restoration of some kind of balance. Today, we suffer from an almost universal idolatry of giantism. It is therefore necessary to insist on the virtues of smallness - where this applies. ..." (p. 54)

I believe this thinking still rings true 30 years later in cyber-architecture.

From the programming classic "The Mythical Man Month" by Frederick P. Brooks, Jr., we learned that the more programmers you throw at a project, the less likely the project will ever be finished! From this follows that a project should have but one central architect, with rather fascist control over feature set and implementation, especially if you want to ship it in a timely fashion. Cocoa gives you the tools needed to build full featured, world-class applications with just a handful of programmers. For best effect, these programmers should be lazy...

Lazy is Good

Laziness is a virtue, believe it or not! I often describe my style of a computer scientist as someone who is so lazy that they'll spend days writing software to save a minute each time the task is performed from then on. There are two forms of laziness that Cocoa embraces - lazy loading of objects and just plain lazy programming.

Lazy loading lets full-featured applications like Stone Design's **Create**® - a three-in-one illustration, page layout and web authoring app - launch in just a few seconds. Compare that to a legacy Carbon application with half the features which takes minutes to launch! By using dynamically loaded bundles, you do not use memory or resources until the end user actually needs that particular feature and its related resources. Moreover, you can update and distribute just the tiny bundle instead of the whole application should, heaven forbid, a bug be found!

To use dynamically loaded bundles, you need to be able to compile your application without actually referencing the loadable object directly. Typically, the types of objects that do well being loaded dynamically are the numerous special editors and interfaces in a program - such as an arrow or pattern editor and the classes it needs to provides the interface. The following conditions make up a good candidate for a loadable bundle:

- Has resources that are not always used each session
- Doesn't contain core data model classes (these should be linked)

Lazy programming means **"Use the 'Kit, Luke!"** Every standard data structure and a complete set of API's are already available to you, so there is rarely a need to reinvent your own. Therefore, this lazy programming axiom has a corollary:

If it's hard to do or understand, it's wrong

By this I mean any coding solution that involves convoluted logic or going beneath the API (using undocumented methods) is probably not the right approach. Taking the time to understand what's already offered to you is well worth the effort, because Cocoa, and it's underlying frameworks, Foundation and AppKit, have evolved over 16 years to provide the basic building blocks of an object oriented solution. Many times when I'm adding a new feature I'll try one brute force approach, notice how cumbersome it is, re-read the AppKit or Foundation API and find a much a better solution involving much less code.

Your efforts should be focused on creating a mapping between the real world problems you are solving and the objects that represent them. Which brings me to my next point:

Put The Code Where It Belongs

One of the biggest challenges facing newcomers to Object Oriented Programming is placing code in the right object. Because many of us grew up with "functional" languages like Basic, Pascal, and C, and because old habits die hard, we need to let go of trying to tell things what do to do, and instead, let them figure it out for themselves. Let's say we have a document which is a list of pages, which contains a list of graphics, which are simple graphics or groups, which contain a list of graphics, which are graphics or groups which contain, etc... And let's say we want to set the "isVisible" state of all the graphics in the document.

The functional approach would be to assume absolute knowledge over this hierarchy, and you'd blythely code something like this:

```
@implementation MyDocument

// please don't do this!
- (void)setAllObjectsVisible:(BOOL)isVisible {
   unsigned int i, pageCount = [_pages count];
   for (i = 0; i < pageCount; i++) {
      Page *p = [_pages objectAtIndex:i];
      NSArray *graphics = [p graphics];
      unsigned int j, graphicsCount = [graphics count];
      for (j = 0; j < graphicsCount; j++) {
         Graphic *g = [graphics objectAtIndex:j];
         if ([g isKindOfClass:[Group class]) {
            NSArray *groupGraphics = [g graphics];
            unsigned k,groupGraphicsCount = [groupGraphics
count];
            for (k = 0; k < groupGraphicsCount; k++) {
          // since this only recurses one level, this code is wrong as
          // well as very hard to read and maintain!!!
          Graphic *groupedGraphic = [groupGraphics
objectAtIndex:k];
           [k setVisible:isVisible];
             }
         else [g setVisible:isVisible];
      }
   }
)

// the OO way:

@implementation MyDocument
- (void)setAllObjectsVisible:(BOOL)isVisible {
     [_pages
makeObjectsPerformSelector:@selector(setAllObjectsVisible:)
```

```
withObject:(id)isVisible];
}
...

@implementation Page
- (void)setAllObjectsVisible:(BOOL)isVisible {
     [_graphics makeObjectsPerformSelector:@selector(setVisible:)
withObject:(id)isVisible];
}
....
```

// groups need to recurse down the hierarchy until individual graphics
// are found...

```
@implementation Group

- (void)setVisible:(BOOL)isVisible {
    [_graphics makeObjectsPerformSelector:@selector(setVisible:)
withObject:(id)isVisible];
}


@implementation Graphic
- (void)setVisible:(BOOL)isVisible {
   // only do work if you absolutely have to - remember LAZY!
    if (_isVisible != isVisible) {
          // you'd probably do undo manager stuff here
          _isVisible = isVisible;
          // alert page we need to be redrawn
          [self tellMyPageToInvalidateMyBounds];
    }
}
...
```

Conclusion

The more you understand object oriented programming and Apple's implementation of Cocoa, the smaller and more reusable your applications will become. And they will load with lightning speed! But more importantly, you'll have less code to maintain which basically means less bugs, less headaches and more time to enjoy life.

Andrew Stone, CEO of Stone Design, www.stone.com, has been the principal architect of several solar houses and over a dozen Cocoa applications shipping for Mac OS X.

And now, more text, an middle schooler's perspective on the Animal Farm:

The Animals Of Animal Farm

By Lillie Stone

The animals from George Orwell's Animal Farm, all have a distinct personality to create a society like that of humans.The pigs are like greedy humans, and borne leaders who use there leadership skills for their own benefit. The sheep are (or animals) people who aren't quite aware of what is going on, and don't understand many things. While Boxer and Clover, the faithful horses, have there doubts, Ben, the donkey, thinks every thing is bad. Mollie and the Cat, unlike most the other animals, don't like working or the way things are.

The pigs seem to be natural leaders, but they use it for there own causes. "The pigs did not actually work, but directed and supervised the others. With there superior knowledge it was natural that they should assume the leadership." (pg. 45) Old Major was a leader, though he also had the animals help in decisions. Snowball wanted to carry out Old major's planes, but is not entirely clean on abusing the animal's ignorance. Napoleon and Squealer play off each other and take advantage of their brains. The pigs are animals of leadership,who take advantage of the other animals trust.

The horses, sheep, and donkey are animals who fallow, take orders, and never state there opinions. Boxer and Clover, the work horses, are very faithful, and Boxer is too trusting, like his two mottos, "I will work harder", and "Napoleon's always right." He'll believe any thing as long as Napoleon "sayes" it. Clover has her doubts of what is happing, but will never speak out. Ben, the gloomy donkey,thinks every thing is wrong, but anything will go. The sheep on the other hand haven't got a clue of what is going on, so they just fallow the others.

Mollie and the cat are the type of animals who are lazy and greedy and help cause animalism to fall. "It was soon noticed that when there was work to be done the cat could never be found." (Pg. 47) The cat likes eating the food ,but not working for it. Mollie just happened to like being a pet and better than working." Without saying any thing to the others, she (Clover) went to Mollie's stall and turned over the straw with her hoof. Hidden under the straw was a little pile of sugar lumps and several bunches of ribbon of different colours." (Pg.62) The two are the only animals who, other than the pigs, who shrike work.

The animals in Animal Farm represent a human society to show how humans interact, and the difficulties they have. They fight with each other and have head butts. Some of this is because of the incompatible nature of the personalities and some from the situation they are in.