7 Deadly Coding Sins

Presented By: Gary Rattray
Mobile Dev NJ MeetUp
November 9th (#11)

These will Kill you!

1. Duplication
2. Lack of Unit Tests
3. Coding Standards
4. Comments
5. Design Spaghetti
6. Complexity
7. Potential Bugs

Google them all...

- Programming
- Programmers
- Coding...

Nice Code Right?

```
-(void) setup {

    CGRect frame = CGRectMake(20, 100, 50, 22);
    UIButton *button1 = [[UIButton alloc] initWithFrame:frame];
    [button1 setTitle:@"Test1" forState:UIControlStateNormal];
    [button1 setBackgroundImage:[UIImage imageNamed:@"test1"]
                       forState:UIControlStateNormal];
    [button1 addTarget:self action:@selector(doSomething) forControlEvents:
     UIControlEventTouchUpInside];

    frame = CGRectMake(50, 100, 50, 22);
    UIButton *button2 = [[UIButton alloc] initWithFrame:frame];
    [button2 setTitle:@"Test2" forState:UIControlStateNormal];
    [button2 setBackgroundImage:[UIImage imageNamed:@"test2"]
                       forState:UIControlStateNormal];
    [button2 addTarget:self action:@selector(doSomething) forControlEvents:
     UIControlEventTouchUpInside];

}
```

Don’t Duplicate Code

```
+ (UIButton *)buttonWithTitle: (NSString *)title
                       target:(id)target
                       selector:(SEL)selector
                       frame:(CGRect)frame
                       image:(UIImage *)image
{
    UIButton *button = [[UIButton alloc] initWithFrame:frame];
    [button setTitle:title forState:UIControlStateNormal];
    [button setBackgroundImage:image forState:UIControlStateNormal];
    [button addTarget:target action:selector
     forControlEvents:UIControlEventTouchUpInside];
    button.backgroundColor = [UIColor clearColor];
    return button;
}
```

Create Unit Tests

```
#import "Testable.h"

@implementation Testable

- (NSUInteger)add4:(NSUInteger)value
{
    return value + 4;
}

@end
```

Example

```
#import "UnitTestable.h"

@implementation Testable

- (void)setUp
{
    [super setUp];
    // Set-up code here.
    test_subject = [[Testable alloc] init];
    STAssertNotNil(test_subject, @"Could not create test subject");
}

- (void)tearDown
{
    // Tear-down code here.
    [test_subject release];
    [super tearDown];
}

- (void)testExample
{
    STAssertEquals([test_subject add4:5], (NSUInteger)9, @"should add 4"
        );
}
```

Naming Methods

Code	Commentary
`insert:at:`	not clear; what is being inserted? what does "at" signify?
`remove:`	not clear; what is being removed?

Naming Methods

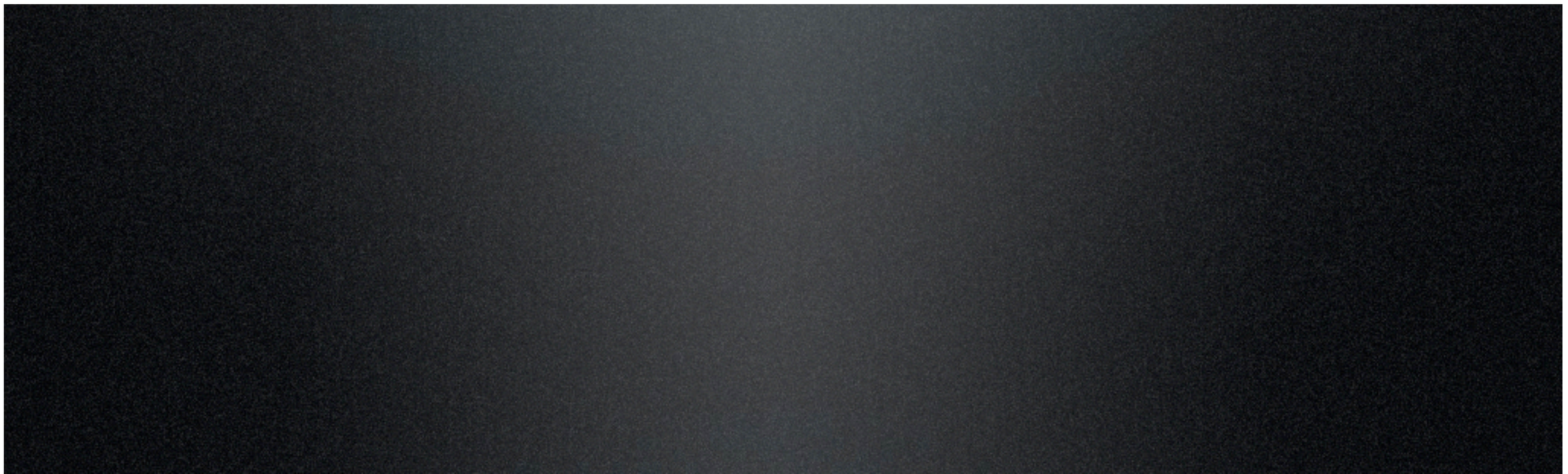
`- (void)setGlyphInfoAccepted:(BOOL)flag;`	wrong
`- (BOOL)glyphInfoAccepted;`	wrong

Naming Functions

`- (NSSize)calcCellSize;`	wrong
`- (NSSize)getCellSize;`	wrong

Naming Methods

`- (int)runModalForDirectory:(NSString *)path andFile:(NSString *)name andTypes: (NSArray *)fileTypes;`	wrong



Naming Functions

- (void)sendAction:(SEL)aSelector :(id)anObject :(BOOL)flag;	wrong

Naming Functions

`- (id)taggedView:(int)aTag;`	wrong

Comments

```
#pragma mark
#pragma mark Button with Image
#pragma mark
- (void)createImageButton
{
   // create a UIButton with just an image instead of a title

   UIImage *buttonBackground = [UIImage imageNamed:@"whiteButton.png"];
   UIImage *buttonBackgroundPressed = [UIImage imageNamed:@"blueButton.
      png"];

   CGRect frame = CGRectMake(0.0, 0.0, kStdButtonWidth, kStdButtonHeight)
      ;

   imageButton = [ButtonsViewController buttonWithTitle:@""
                        target:self
                        selector:@selector(action:)
                        frame:frame
                        image:buttonBackground
                        imagePressed:buttonBackgroundPressed
                        darkTextColor:YES];

   [imageButton setImage:[UIImage imageNamed:@"UIButton_custom.png"]
      forState:UIControlStateNormal];
}
```

What's this called?

```
10 i = 0
20 i = i + 1
30 PRINT i; " squared = "; i * i
40 IF i >= 10 THEN GOTO 60
50 GOTO 20
60 PRINT "Program Completed."
70 END
```

Here is the same code written in a structured programming style:

```
FOR i = 1 TO 10
    PRINT i; " squared = "; i * i
NEXT i
PRINT "Program Completed."
END
```

Spaghetti Code

Spaghetti code is a pejorative term for source code that has a complex and tangled control structure, especially one using many GOTOs, exceptions, threads, or other "unstructured" branching constructs.

Solutions

Rethink Control Structure, Classes, and Libraries
Use Modeling Tools, Mockups, Use Cases
Build Modular Apps

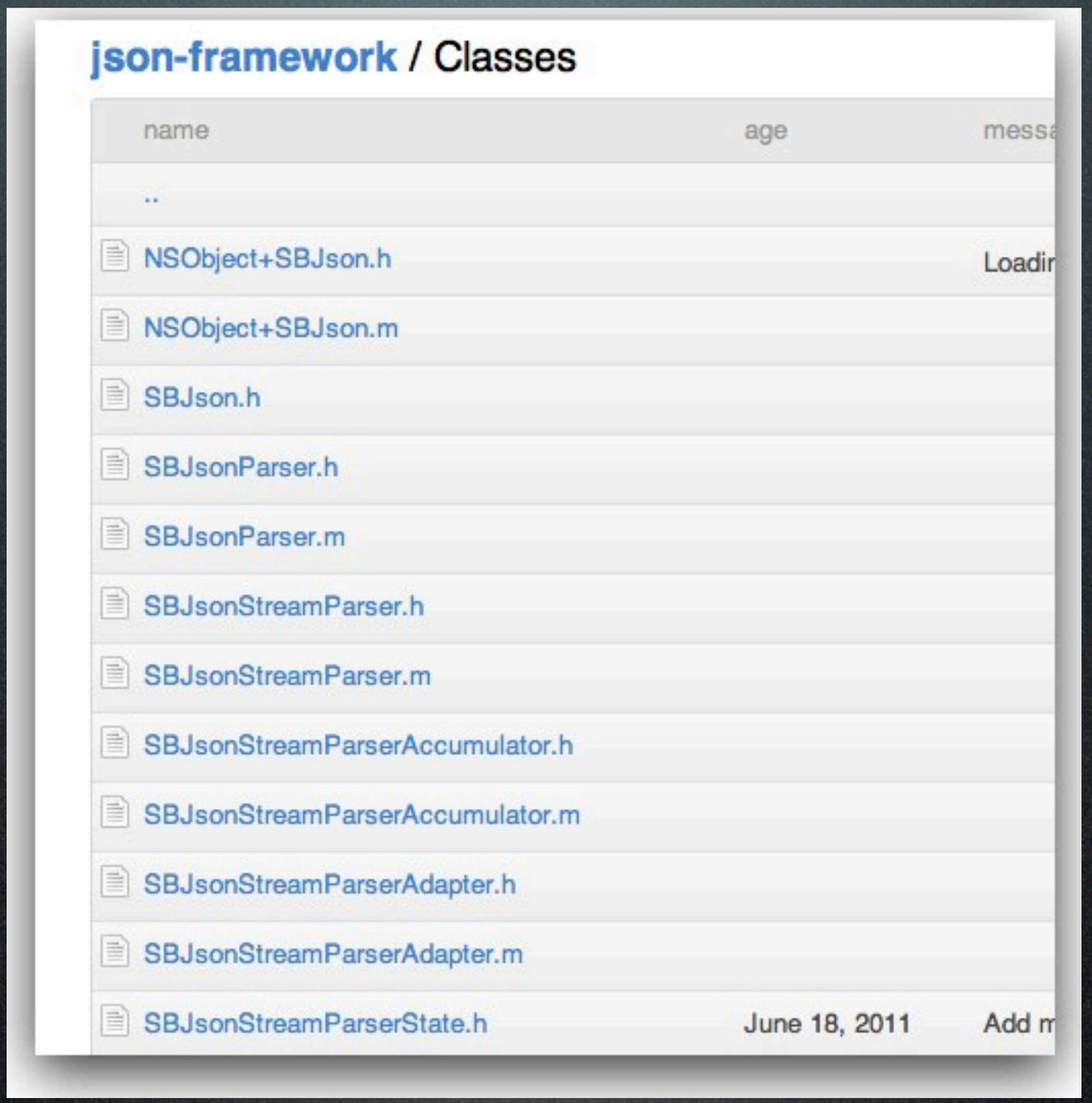
Ask Yourself:
What do I need this particular piece of code to do ?
What is this Class/Method responsible for?

Solutions

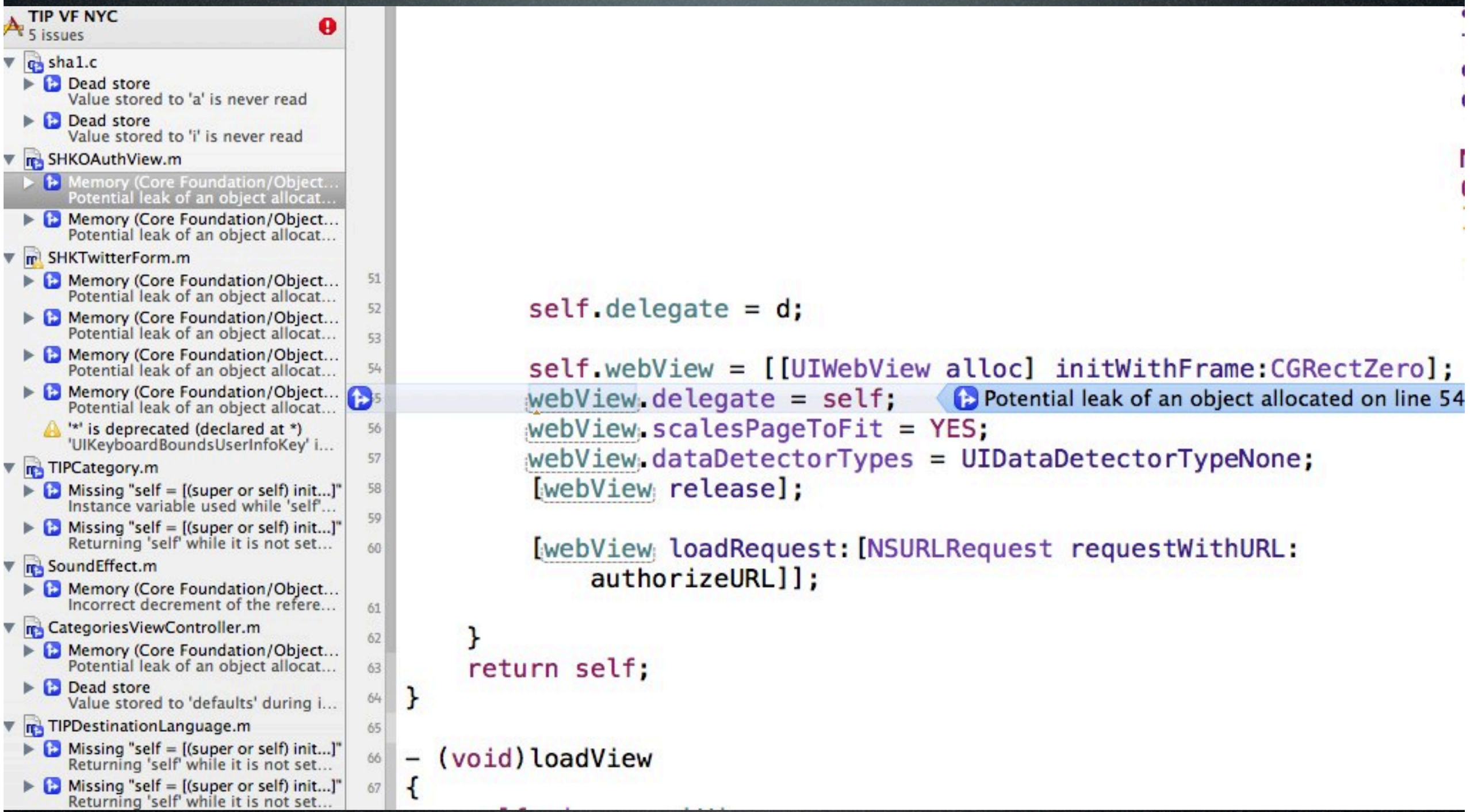
- ✓ Plan first, code only after you have a very good idea on how to solve the problem.
- ✓ Reuse first, create functional libraries if necessary.
- ✓ Use Object oriented inheritance and subclasses helps you add changes over time.
- ✓ Focus on encapsulation - e.g Button example.

Design Spaghetti

Complexity


json-framework / Classes
name
age
messa
..
NSObject+SBJson.h
Loadir
NSObject+SBJson.m
SBJson.h
SBJsonParser.h
SBJsonParser.m
SBJsonStreamParser.h
SBJsonStreamParser.m
SBJsonStreamParserAccumulator.h
SBJsonStreamParserAccumulator.m
SBJsonStreamParserAdapter.h
SBJsonStreamParserAdapter.m
SBJsonStreamParserState.h
June 18, 2011
Add m

Potential Bugs



Resources

- http://developer.apple.com/library/mac/#docu
- http://developer.apple.com/library/ios/#docu
Unit_Testing_Applications/unit_testing_applic
- QUESTIONS?

Resources

- http://developer.apple.com/library/mac/#documentatio
- http://developer.apple.com/library/ios/#documentatic
  Unit_Testing_Applications/unit_testing_applications.htr
- QUESTIONS?