iPhone OS

Applikationsentwicklung auf mobilen Endgeräten

Nicolas Neubauer
Universität Osnabrück, 22.04.2010

Agenda

Inhalt

Einführung in ...

- die Hardware-Grundlagen
- den konzeptuellen Plattform-Aufbau
- Objective C und Cocoa Touch
- den Umgang mit XCode
- den Umgang mit dem Interface Builder
- die Applikationsentwicklung mit iPhone OS

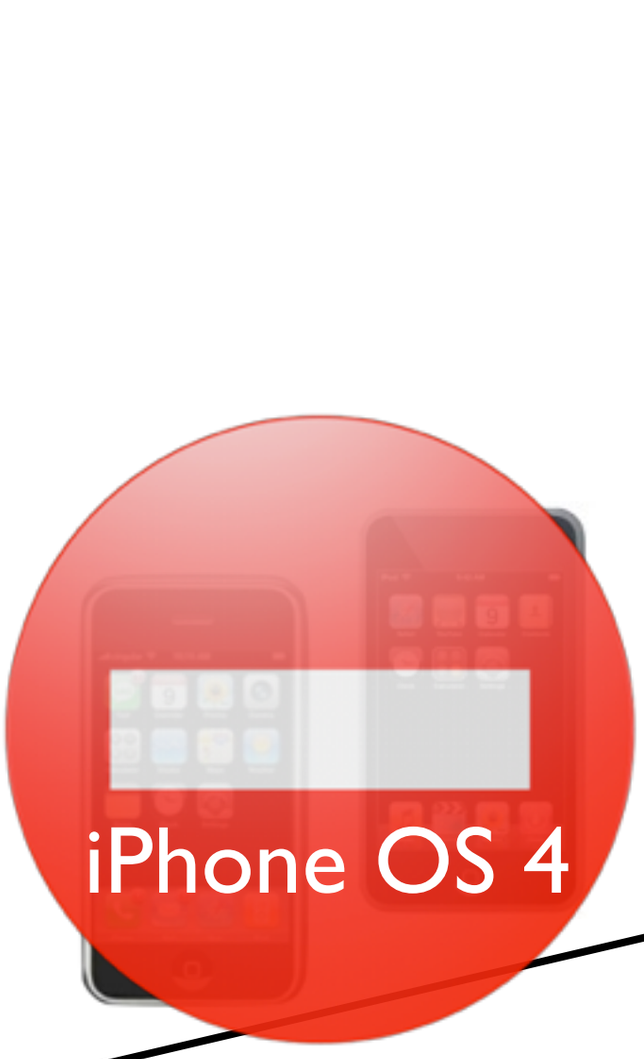
Bevor es los geht: Was brauche ich?

- Apple Mac (mit Intel-Prozessor)
- kostenlose XCode-Tools (Entwicklungsumgebung)
- kostenloses SDK
  developer.apple.com/iphone
- optional: mobile Hardware
- optional: iPhone Developer Program-Mitgliedschaft

iPhone OS-Hardware

- Hardware-Basis relativ homogen
- Bildschirmauflösung: 480x320 Pixel
- immer Multi-Touch, keine Tastatur, keine (API-nutzbaren) Hardware-Knöpfe
- Beschleunigungssensor
- OpenGL ES 1.0
- gleiche hardwareunabhängige APIs

iPhone OS-Hardware: Entwicklung



2007
iPhone (2G)
iPod touch (1G)

2008
iPhone 3G

+ UMTS
+ GPS

2008
iPod touch (2G)

2009
iPhone 3GS

+ OpenGL ES 2.0
+ verdoppelter
Arbeitsspeicher

2009
iPod touch (3G)

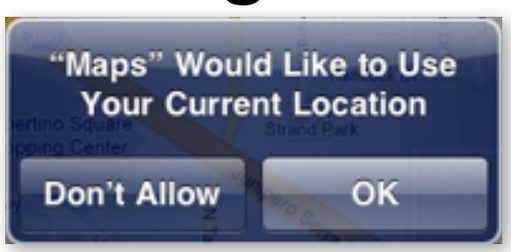
iPhone OS-Hardware-Beschränkungen

- allgemeine Beschränkungen in mobiler Entwicklung
- iPods ohne GPS (Ortung via Wifi möglich)
- OpenGL ES 2.0 nur mit iPhone 3GS und iPod 3G
- unterschiedliche Rechenleistung beachten
- Applikationen auf ~ 25 MB/~ 80 MB Speicher begrenzen (Geräte: 128 MB/256 MB RAM)

iPhone OS-Beschränkungen/Sicherheit

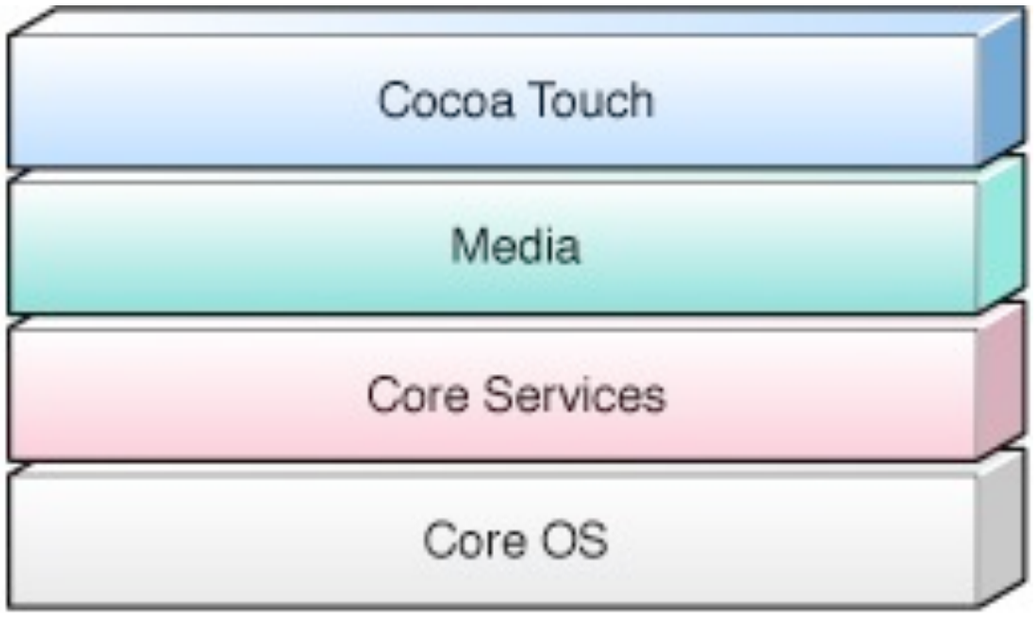
- kein Multi-Tasking (erlaubt)
- kein Hardware-Zugriff außerhalb der iPhone OS-APIs (erlaubt)
- Telefon-Funktionen geschützt

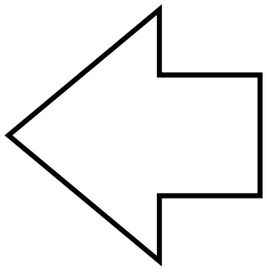
- Ortungsdienste geschützt



- Datenzugriff immer erlaubt

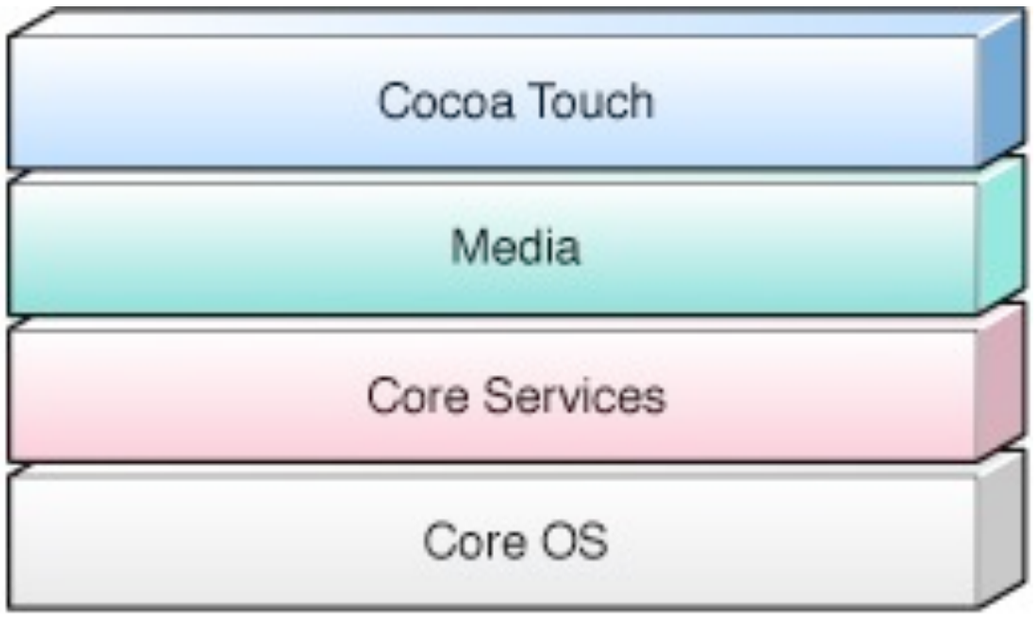
iPhone OS-Technologieebenen





- Hardware-Zugriff
- Treiberverwaltung
- ...

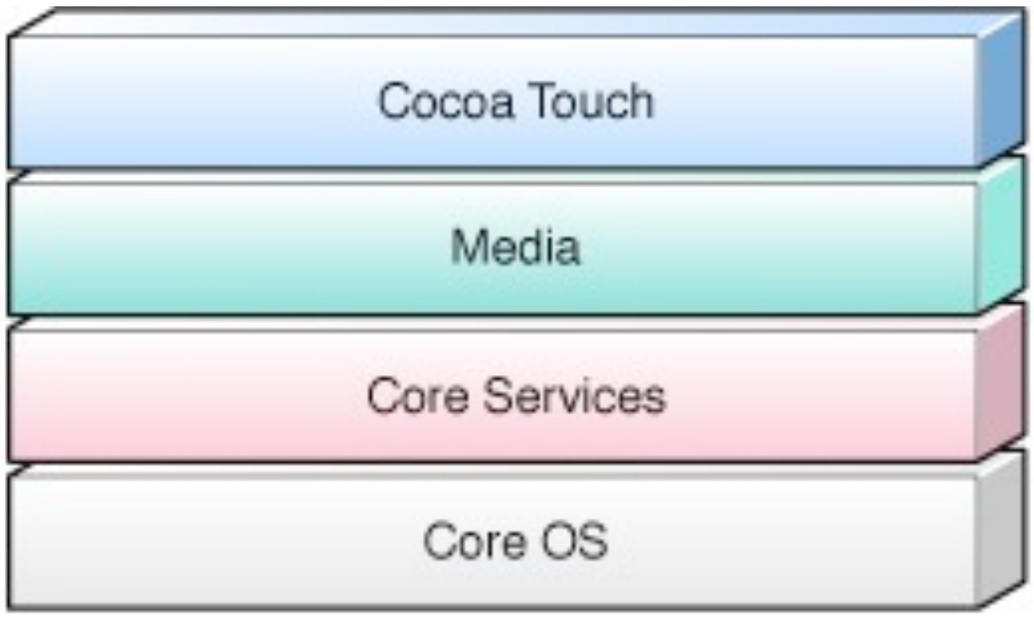
iPhone OS-Technologieebenen



Mach-UNIX-Kernel & Hardware-Treiber

- Threading
- Low-Level Netzwerk (Sockets)
- Dateizugriff (I/O)
- ...

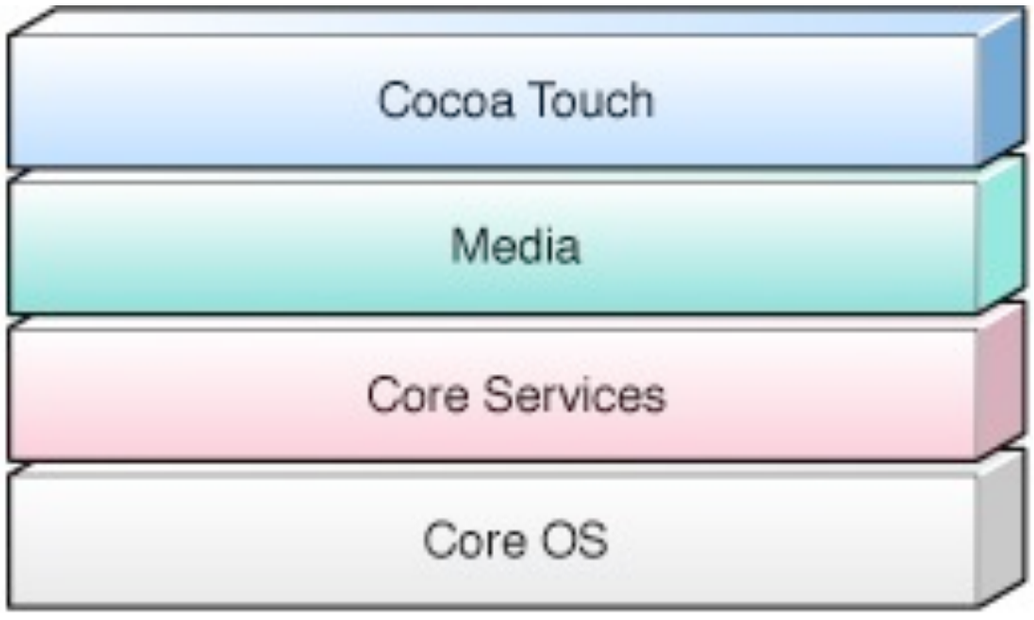
iPhone OS-Technologieebenen



Mach-UNIX-Kernel &
Hardware-Treiber

- Collections
- Strings
- Date/Time
- Core Data
- Core Location
- ...

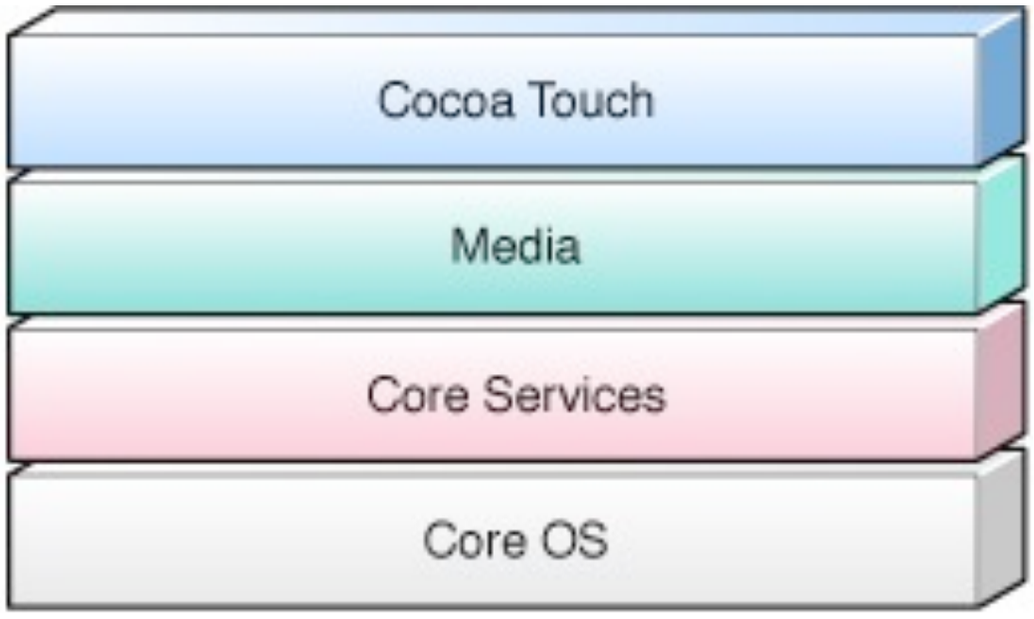
iPhone OS-Technologieebenen



- OpenGL
- Grafik/Video
- Audio
- ...

Mach-UNIX-Kernel &
Hardware-Treiber

iPhone OS-Technologieebenen



- UIKit
- MapKit
- Push-Notifications
- ...

Mach-UNIX-Kernel &
Hardware-Treiber

iPhone OS-Technologieebenen

- Core OS:
  sehr viel C, wenig Objective-C
- Core Services:
  viel C, viel Objective-C
- Media:
  viel C, (Grafik-)hardware-nah, viel Objective-C
- Cocoa Touch:
  wenig C, sehr viel Objective-C

Was ist Objective-C

- (komplexer) Aufbau auf C
- objektorientiert
- dynamisch
- kompiliert (nativ/hardware-nah)
- verfolgt (insbesondere mit den Apple-Frameworks) ähnliche Ideen wie Java
- ... ist trotzdem „anders“

Objective-C: Basics

- primitive Datentypen (z.B.:)
  - int
  - float, double
  - BOOL (YES oder NO)
- „komplexe“ Datentypen
  - Klassen und Objekte

Objective-C: Basics

- C-Kontrollstrukturen

```
if(i == 1)
{
      //i ist 1
}
else if(i != 2)
{
      //i nicht 1 und nicht 2
}
else
{
      //i ist 2
}
```

```
switch (i)
{
    case 1:
        //i ist 1
        break;
    case 2: case 3:
        //i ist 2 oder 3
        break;
    default:
        //i ist alles nur nicht 1, 2 oder 3
        break;
}
```

- Schleifen analog

```
for(int i = 0; i < 10; i++) {}

while(i < 10) {}
```

Objective-C: Basics

- Messages (Methoden)

```
[meinePerson sayHello]; //entspricht meinePerson.sayHello();
```

- Parameterübergabe

```
[meinePerson sayHelloTimes: 20]; //entspricht meinePerson.sayHelloTimes(20);
```

- Mehrere Paramter

```
[meinePerson setAge: 31 height: 180]; //entspricht meinePerson.setAgeAndHeight(31,180);
```

- Methoden schachteln

```
[anderePerson sayHelloTo: [meinePerson sayName]];
//entspricht anderePerson.sayHelloTo(meinePerson.sayName());
```

Objective-C: Eigene Klassen

- Trennung von Implementierung und Definition
- *Interface* (≠Java Interface) (.h für Header)
- *Implementation (.m für Implementation)*

Objective-C: Klassendefinitionen

- Person.h

```
#import <Foundation/Foundation.h>


@interface Person : NSObject {

}

@end
```

- Person.m

```
#import "Person.h"


@implementation Person

@end
```

Objective-C: Instanziieren

- alloc/init

```
#import "Person.h"

int main(int argc, char *argv[]) {
    Person *p = [Person alloc];
    [p init];

    Person *p2 = [[Person alloc] init];

    return 0;

}
```

- *alloc* besorgt Speicherplatz
- *init* Default-„Konstruktor“ (in NSObject definiert)

Objective-C: Pointer

- Pointer

```
Person *p = [[Person alloc] init];
//entspricht Person p = new Person();
```

- Pointer ist Variable, die Speicherbereich merkt

```
p = 1;
//hat in Java keine Entsprechung
```

- Tue so, als ob im Speicher an Adresse 1 eine Person gespeichert wäre.
- Wo ist die „echte“ Person hin? Leaked.

Objective-C: Speichermanagement (Teil 1)

- Speicherplatz besorgen

```
Person *p = [[Person alloc] init];
//belegt Speicher für eine Person, initialisiert Person und gibt Speicheradresse zurück
```

- Person zählt „wie oft sie noch gebraucht wird“

```
[p retainCount];
//ist nach alloc/init 1
```

- wenn retainCount 0 ist, wird Person gelöscht, Speicherplatz freigegeben (kein Leak)

- retainCount erhöhen und verringern

```
[p retain];
//retainCount +1
[p release];
//retainCount -1
```

Objective-C: Basics

- Systemklassen beginnen mit „NS“ z.B.:
- *NS*String

```
"Hallo, ich bin ein char-Array..."

@"Hallo, ich bin ein NSString."

NSString *str = [[NSString alloc] initWithFormat:@"Und ich auch!"];
```

- char-Arrays werden bei Mac/iPhone Dev kaum genutzt
- „konstante“ NSStrings mit @-Zeichen

Objective-C: Eigene Klassen

- Person.h

```
#import <Foundation/Foundation.h>

@interface Person : NSObject {
   int age;
   NSString *name;
}

-(id) initWithAge: (int) newAge name: (NSString *) newName;

-(int) howOldAreYou;

-(NSString *) sayHelloTo: (NSString *) name;

@end
```

Objective-C: Eigene Klassen

- Person.m

```
#import "Person.h"


@implementation Person

-(id) initWithAge: (int) newAge name: (NSString *) newName
{
   if(self = [super init])
   {
         age = newAge;
         name = newName;
   }
   return self;
}

-(int) howOldAreYou
{
   return age;
}

-(NSString *) sayHelloTo: (NSString *) name
{
   return [NSString stringWithFormat:@"Hello, %@!", name];
}

@end
```

Objective-C: Speichermanagment (Teil 2)

- Person hat einen Namen
- Was passiert, wenn jemand den Namen „löscht“ die Person aber noch darauf zugreifen will?
- Person muss sagen: „Ich brauche den Namen noch!“

```
-(id) initWithAge: (int) newAge name: (NSString *) name
{
   if(self = [super init])
   {
         age = newAge;
         name = name;
         [name retain]; //“Ich brauche den Namen noch!“
   }
}
```

Objective-C: Speichermanagment (Teil 2)

- Wann brauche ich den Namen nicht mehr?
- retainCount der Person ist 0
- Person wird gelöscht...
- dealloc wird aufgerufen (in NSObject definiert)

```
-(void) dealloc
{
   if(name) [name release];
   [super dealloc];
}
```

- dealloc nicht selbst aufrufen!

Objective-C: Speichermanagment (Teil 2)

- Was, wenn ich etwas zurück gebe?

```
-(NSString *) sayHelloTo: (NSString *) name
{
   NSString *retString = [[NSString alloc] initWithFormat:@"Hello, %@!", name];
   //jetzt ist retainCount 1
   [retString release];
   //jetzt ist retainCount 0
   return retString; //retString ist u.U. jetzt schon gelöscht
}
```

- autorelease-Pool benutzen

```
-(NSString *) sayHelloTo: (NSString *) name
{
   NSString *retString = [[NSString alloc] initWithFormat:@"Hello, %@!", name];
   //jetzt ist retainCount 1
   [retString autorelease];
   //jetzt ist retainCount immer noch 1 (aber gemerkt 0)
   return retString; //jetzt ist retainCount immer noch 1 (aber gemerkt 0)
}
```

Objective-C: Properties

- Setter und Getter nicht selbst schreiben, sondern synthetisieren (Person.h):

```
#import <Foundation/Foundation.h>

@interface Person : NSObject {
   int age;
   NSString *name;
}

@property(nonatomic, assign) int age;
@property(nonatomic, retain) NSString *name;

@end
```

- Person.m

```
#import "Person.h"

@implementation Person

@synthesize age, name;

-(id) initWithAge: (int) newAge name: (NSString *) name
{
    ...
```

Objective-C: Properties

- Zugriff mit Punkt-Syntax

```
Person *p = [[Person alloc] init];
p.name;
p.age = 31;
```

- Praktisch (wegen retain in Property-Deklaration):

```
-(id) initWithAge: (int) newAge name: (NSString *) name
{
   if(self = [super init])
   {
         self.age = newAge;
         self.name = name;
         //jetzt unnötig: [name retain];
   }
}

-(void) dealloc
{
   //if(name) [name release];
   self.name = nil;
   [super dealloc];
}
```

Objective-C: Collections

- NSArray (etwa List, Stack, Queue) und NSDictionary (etwa HashMap)
- Obacht: normalerweise nicht änderbar (kein add, delete, ...)
- NSMutableArray, NSMutableDictionary

```
NSArray* array = [NSMutableArray arrayWithCapacity:2];
[array addObjects: @"Erster"];
[array addObjects: @"Zweiter"];
[array addObjects: @"Dritter"]; //erhöhe Capacity automatisch
[array objectAtIndex:0]; //Erster
[array removeLastObject]; //“pop“

NSMutableDictionary* dict = [NSMutableDictionary dictionaryWithCapacity:1];
[dict setObject: @"Erster" forKey: @"ErsterKey"];
[dict objectForKey:@"ErsterKey"] //Erster
```

Objective-C: Protokolle

- Protokoll in ObjC ≈ Java Interface
- Konvention: nicht ... able sondern ...ing
  z.B. Laughing.h

```
@protocol Laughing

-(NSString*) laugh;

@end
```

- Implementierung ankündigen (z.B. Person.h)

```
@interface Person : NSObject <Laughing> {
   int age;
   NSString *name;
}
```

Objective-C: Hilfreich für den Start

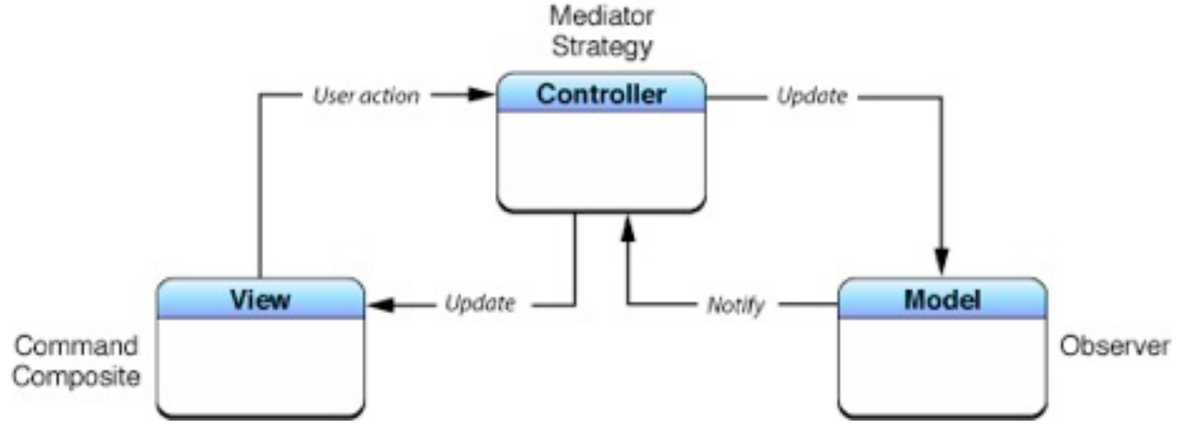
- Log auf der Konsole mit

```
NSLog(@"i ist jetzt %i, String ist: %@", i, meinString);
```

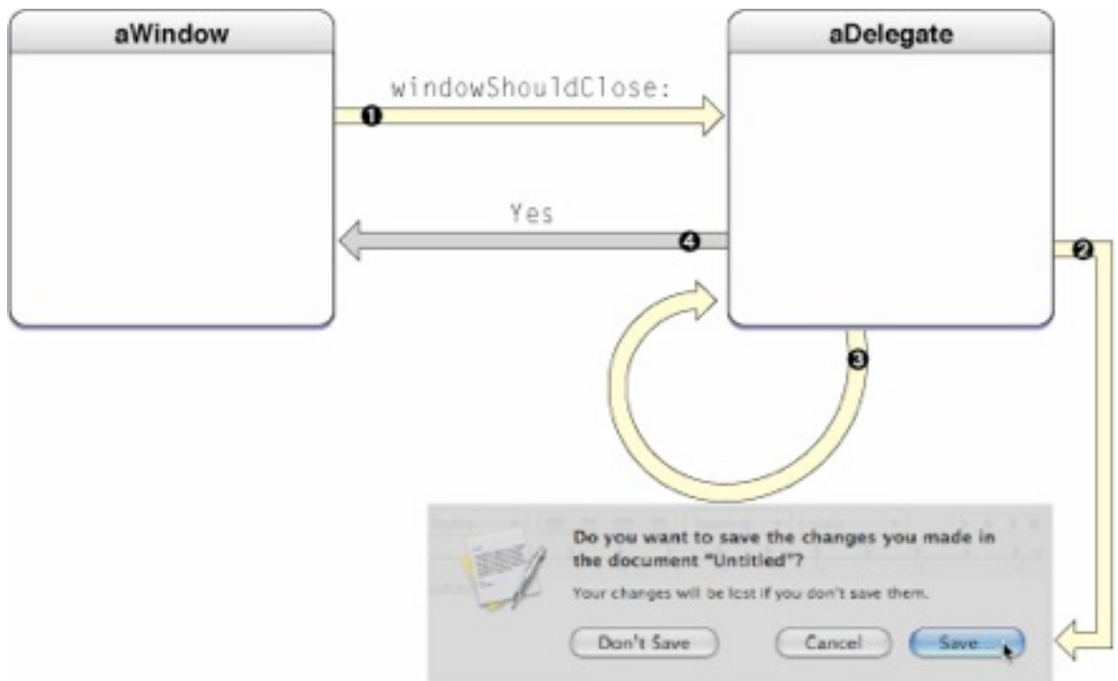
- toString heißt in ObjC description
- Memory-Management beachten (!)
- Dokumentation

Wichtige Design-Patterns

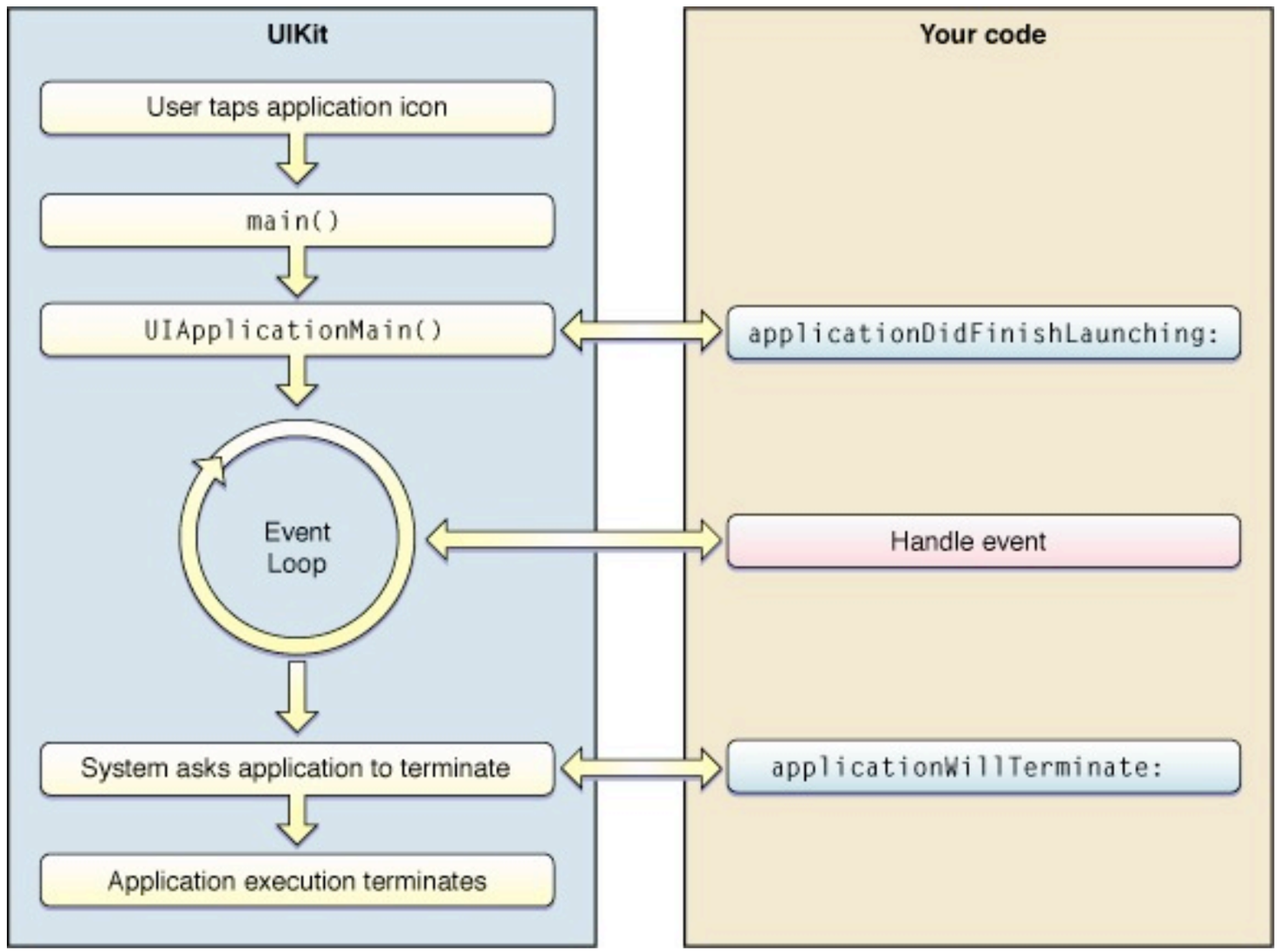
- Model-View-Controller



- Delegation

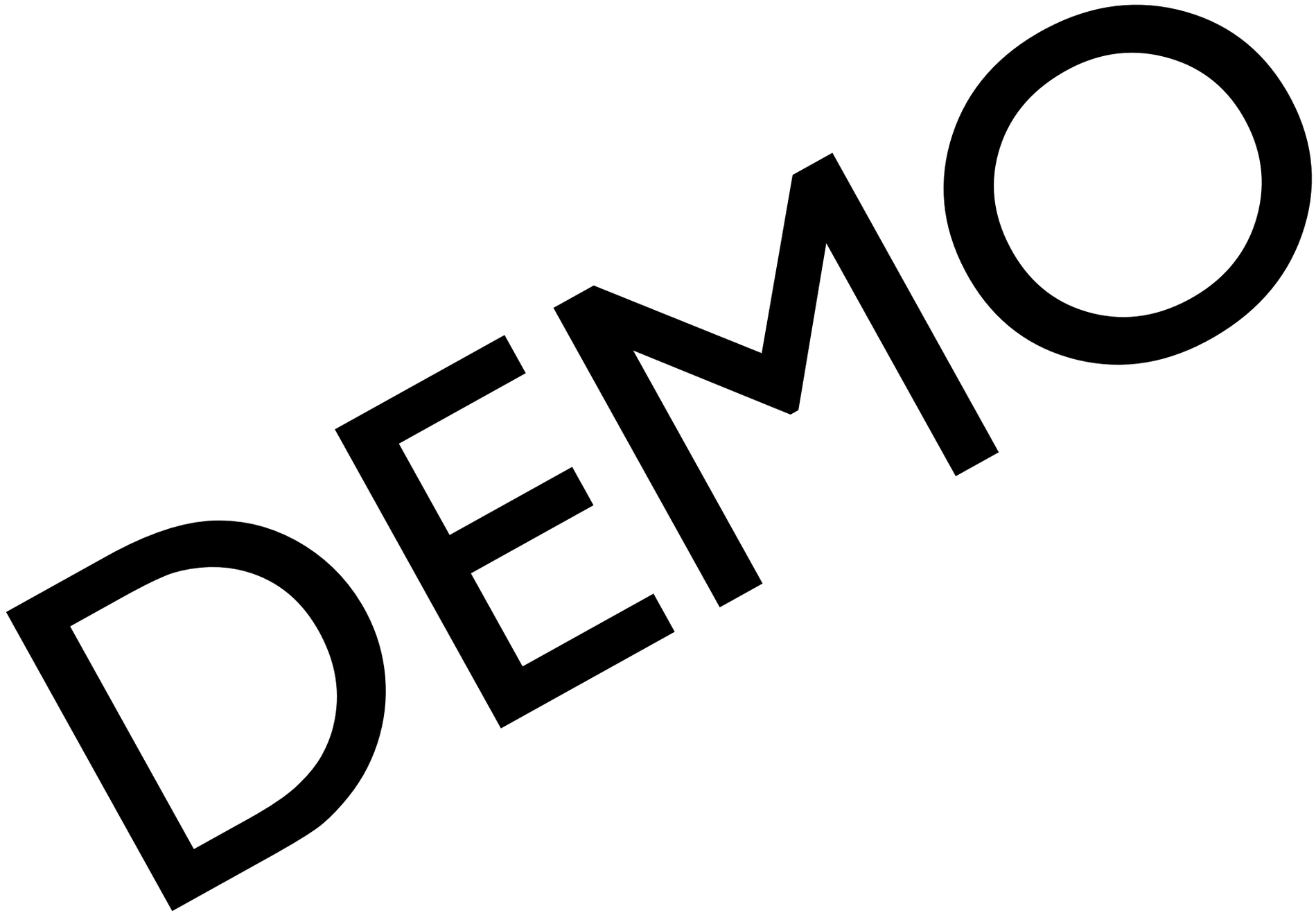


Applikationsentwicklung mit iPhone OS



Einführung in Cocoa Touch

Applikationsentwicklung mit iPhone OS



Demo-Zusammenfassung

- von X-Code-Templates lernen
- Interface-Builder für das UI nutzen
- mit MVC UI und View-Controller verknüpfen
  - Stichwörter: IBOutlet, IBAction
- testen auf dem Simulator (nicht Emulator!)
- testen auf dem Gerät
- testen mit Instruments

Literatur-Hinweise

- Apple‘s iPhone OS Reference Library
  (http://developer.apple.com/iphone/library/navigation/index.html)
  - Learning Objective-C: A Primer
  - The Objective C Programming Language
  - iPhone Application Programming Guide
  - Cocoa Fundamentals Guide
- D. Mark & J. LaMarche:
  Beginning iPhone Development: Exploring the iPhone SDK (Apress, 2009)

Fazit

- Einschränkungen fraglich
- Einstiegshürde: Objective-C
- native Ausführung, gute Performance, sinnvolle Konzepte und „schöner“ Code
- leistungsstarkes, ausgezeichnet dokumentiertes und sinnvoll aufgebautes SDK
- gut abgestimmte Entwicklungsumgebung