TDD für iPhone OS

xpdays 2009

Tammo Freese

Inhalt

- Unit Testing für iPhone OS
- Mockobjekte für iPhone OS
- TDD für iPhone OS?

Unit Testing auf dem iPhone

- Vor iPhone OS 3.0:
  - kaum dokumentiert
  - nur auf dem Entwicklungsrechner
- Seit iPhone OS 3.0:
  - gut dokumentiert
  - auch direkt auf dem Device

Unit Testing für Mac OS X

- Gibt es schon lange!
- SenTestingKit (aka OCUnit)
  - Basiert wie JUnit auf SUnit
  - Release 1.0: Mai 1998
    (JUnit: Februar 1998)
  - Seit Mitte 2005 von Apple
    in Xcode integriert

Testfallklasse

- ExampleTests.h:

```
#import <SenTestingKit/SenTestingKit.h>
#import <UIKit/UIKit.h>
@interface ExampleTests : SenTestCase {}
@end
```

- ExampleTests.m:

```
#import "ExampleTests.h"
@implementation ExampleTests
- (void)testFail {
    STFail(@"I can haz fail?!");
}
@end
```

Assertions

- Unterschiede zu JUnit:
  - Test auf Gleichheit: Erwartung steht an 2. Stelle
  - Fehlgeschlagene Assertion bricht den Test nicht ab
  - Kommentar ist Pflichtparameter
  - Assertions sind C-Makros

Assertions

- STAssertEquals, STAssertEqualObjects
- STAssertEqualsWithAccuracy
- STAssertNil, STAssertNotNil
- STAssertTrue, STAssertFalse
- STAssertThrows, STAssertThrowsSpecific

Unit Testing für das iPhone

- Zwei Arten von Tests
  - Logic Tests
  - Application Tests

Logic Testing

- Isoliert Teile des Codes in einem eigenen Test Bundle
- Tests laufen auf dem Entwicklungsrechner
  - x86-Binary!
- Eigentliche Applikation läuft dabei nicht, weder im Simulator noch auf dem Device

Application Testing

- Tests werden in der Applikation ausgeführt
- Tests laufen nur direkt auf dem Device
- Die Applikation muss also laufen

Logic-Testing konfigurieren

- Unit Test Bundle Target "…Tests" hinzufügen (zum Beispiel "LogicTests")
- LogicTests in das Target der Applikation ziehen

Testfallklasse hinzufügen

- (Gruppe Tests erzeugen)
- Objective-C test case class
  zum Target LogicTests (!) hinzufügen
- Im Test: USE_APPLICATION_UNIT_TEST
  auf 0 setzen oder Template aufräumen

Logic Tests laufen lassen

- Durch Target-Abhängigkeit automatisch beim Build
- Fehlgeschlagene Tests ~~werden~~wurden bis Xcode 3.1 wie Compile-Fehler angezeigt
- Vorsicht: Tests werden nur beim Simulator-Target ausgeführt!

Application Testing konfigurieren

- Application Target duplizieren in Target "…Testing" (zum Beispiel "MyAppTesting")
- Unit Test Bundle Target "…Tests" hinzufügen (zum Beispiel "MyAppTests")
- MyAppTests in das Target der App ziehen
- MyAppTests.octest in Copy Bundle Resources von MyAppTesting ziehen

Testfallklasse hinzufügen

- (Gruppe Tests erzeugen)
- Objective-C test case class
  zum Target MyAppTests (!) hinzufügen
- Im Test: USE_APPLICATION_UNIT_TEST
  auf 1 lassen oder Template aufräumen

Application Tests laufen lassen

- Ausführung beim Starten der App mit Target MyAppTesting auf dem Device
- Fehlgeschlagene Tests werden textuell in der Debug-Konsole angezeigt
- Vorsicht: Application Tests werden bei Simulator-Target schon beim Build ausgeführt und schlagen daher meist fehl

Logic Tests vs Application Tests

- Logic Tests bevorzugen:
  Beim Simulator-Build ausgeführt, schneller, besser integriert
- Application Tests eher für Akzeptanztests und für Dinge, die nur auf dem Device gehen, z.B. Security-Framework
- Logic Tests auch zu Application Tests hinzufügen, wenn möglich

Mockobjekte für iPhone OS

- Seit 2004: OCMock
- Verwandt zu EasyMock
- Verwendet das Trampoline-Pattern
  - [mock expect] gibt ein Proxy-Objekt zurück
  - Proxy-Objekt erlaubt Definition der Erwartungen

OCMock konfigurieren

- OCMock.framework zu LogicTests hinzufügen
- Neue Copy Files Build Phase für Frameworks, OCMock.framework rein
- In LogicTests Build-Settings: Runpath Search Paths auf

  `${TARGET_BUILD_DIR}/${FRAMEWORKS_FOLDER_PATH}`

  setzen

OCMock-Beispielcode (sinnfrei)

```
#import <OCMock/OCMock.h>
#import "ExampleTests.h"
#import "User.h"

 @implementation ExampleTests
 - (void)testUser {
   id user = [OCMockObject mockForClass:[User class]];
   [[user expect] setName:@"John"];
   [[[user expect] andReturn:@"John"] name];

   [user setName:@"John"];
   STAssertEqualObjects([user name], @"John", @"");

   [user verify];
 }
 @end
```

Mockobjekte mit OCMock

- Für Klassen:

```
+ (id)mockForClass:(Class)aClass;
+ (id)niceMockForClass:(Class)aClass;
```

- Für Protokolle:

```
+ (id)mockForProtocol:(Protocol *)aProtocol;
+ (id)niceMockForProtocol:(Protocol *)aProtocol;
```

- Partieller Mock für ein Objekt:

```
+ (id)partialMockForObject:(NSObject *)anObject;
```

Expectations auf Mockobjekten

- Geprüft:
  `[[mock expect] methodenAufruf…];`
- Ungeprüft:
  `[[mock stub] methodenAufruf…];`
- Aufrufreihenfolge prüfen:
  `[mock setExpectationOrderMatters:YES];`
- Endabnahme:
  `[mock verify];`

Prüfen von Argumenten

- Beispiele für Argument-Checks für `[[mock expect] setName:…];`
  - Equals-Check: `@"John Doe"`
  - Alle zulassen: `[OCMArg any]`
  - Nicht nil: `[OCMArg isNotNil]`
  - Hamcrest-Matcher: `startsWith(@"John")`

Verhalten von Mockobjekten

- Das Übliche:

```
[[[mock stub] andReturn:@""] name];
[[[mock stub] andReturnValue:42] count];
[[[mock stub] andThrow:anException] doStuff];
```

- Nicht so üblich:

```
[[[mock stub] andPost:aNotification] doStuff];
[[[mock stub] andCall:aSelector
              onObject:anObject] doStuff];
```

Meine Meinung zu OCMock

- Trampoline-Pattern
  - Nachteile: Gewöhnungsbedürftig, keine intelligente Completion
  - Vorteil: kein [mock replay]
- Warum kein OCEasyMock?
  - Bisher keine wirklich bessere Syntax gefunden

TDD für iPhone OS?

- Pflicht erfüllt:
  - Unit Testing Framework
  - (Mockobjekte)
- Reicht das?

TDD

- Grün-rot
- Rot-grün
- Refactoring

Grün-rot

- OK: Neuen Test schreiben, Build
- FAIL: Bei neuen Klassen keine Completion ins Unbekannte: Klasse selbst erzeugen
- FAIL: Bei neuen Methoden nur Warnung
- FAIL: Bei neuen Methoden keine Completion ins Unbekannte: Signatur selbst in den Header schreiben und selbst leer implementieren

Rot-grün

- OK: Implementieren muss man immer noch selbst ;)

Refactoring

- Aktuell in Xcode 3.2.1:
  - Extract Method/Function
  - Rename Class/Method/Variable
  - Move Up/Move Down
  - Create Superclass
  - Encapsulate, Modernize Loop

Refactoring-Probleme

- Recht langsam
- Nicht so zuverlässig
- Eingeschränkt, es fehlen viele Refactorings
  - Extract Local Variable
  - Inline Method
  - …

Meine Meinung zu TDD für das iPhone

- Geht, ist aber mühsam
  - Erinnert an Java vor langer Zeit
  - Completion ins Unbekannte fehlt sehr
  - Refactoringunterstützung ist vielleicht nicht perfekt, aber immerhin vorhanden
- Wichtigster Punkt: Viel besser, als viele ahnen! ("TDD-feindliche Umgebung"?!)

Weitere Informationen

- Dokumentation: *Unit Testing Applications* im *iPhone Development Guide*
- Sample Code: *iPhoneUnitTests*
- OCMock: Suchen mit Google ;)

Die letzte Folie

- Vielen Dank für Ihr Interesse!

Tammo Freese
business@tammofreese.de
twitter: tammofreese