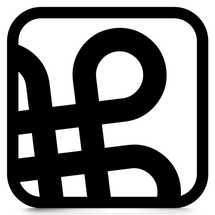


Macoun

Unit Tests für Totalverweigerer

Peter Hauke, Ingo Kasprzak

Hinweis

- Unit Tests für Totalverweigerer
- Spieleentwicklung mit Sprite Kit (Terrassensaal)
- B.L.I.N.K. (Kleiner Saal)

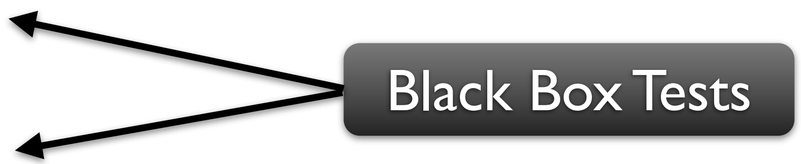
Ablauf

- Theorie
- Theorie mit etwas Praxis
- Praxis
- Demos

Testing

Varianten von Testing ...

- Verbreitete Tests:
  - Integrationstest
  - Systemtests
  - Akzeptanztest



- Problem bei allen: Debugging

... eine weitere
Unit Tests

- Test funktionaler Einzelteile (Module)
- Validierung auf statische Bedingungen
- Idee:
  - Einzelteile genügen Spezifikationen
  - Gesamtheit der Einzelteile genügt Spezifikation

Ja und nun?

Grundidee

Kleine Teile testen

- Tests ohne Abhängigkeiten von:
  - Benutzereingaben
  - Konfiguration des Host
  - Externen Daten (Datenbanken, Server)
  - anderen Tests
- Module liefern zu diskreten Eingaben reproduzierbare Ergebnisse

Ergebnis
Hohe Testabdeckung

- Vielzahl von Test, die
  - jederzeit wiederholt werden können
  - ständig wiederholt werden
- Stetige Validierung von existierendem Code
- Macoun 2013 „*96% Testabdeckung*“ (Maxim Zaks)

Ich versteh es immer noch nicht

Test Driven Development

Test First Development

- Ansatz:
  - Test schreiben
  - Code schreiben der Test genügt
  - Refactoring
- wenig überflüssiger Code & hohe Testabdeckung

Test Driven Development

Voraussetzungen

- Genaue Analyse der Anforderungen vor Entwicklung
- Kleinteilige Problemstellungen identifizieren (*Microfeatures*)
  - kleinere Microfeatures - einfachere Tests

Test Driven Development

Und so gehts ... (1)

- Analyse
- ein Microfeature identifizieren
- Randbedingungen für Microfeature erkennen
- Test schreiben

Test Driven Development

Und so gehts ... (2)

- Test starten
  - fail: Code fehlt noch
  - pass: Microfeature bereits implementiert
- Code schreiben
- Test starten

Test Driven Development

Und so gehts ... (3)

- Code so lang anpassen, bis Test erfolgreich
  - Just Barely Good Enough™
- Refactor
- Testen

Test Driven Development

Beispiel: Taschenrechner

- Analyse:
  - Methoden zur Berechnung
- Microfeature:
  - Addition zweier Zahlen
- Randbedingungen:
  - Eingabe numerisch und nicht `nil`

Änderung der Herangehensweise

- Weg von Gedanken „*Wie kann ich das debuggen?*“
- Über „*Wie kann ich einen Test dafür schreiben?*“
- Zu „*Wie kann ich beweisen, dass mein Code den Anforderungen genügt?*“

Vorteile

- Code wird kleinteilig
- Code wird kleinteilig tesbar
- Zeitnahe Problembehebung
- Weniger Code
- Sicherheit beim Refactor
- Spätere Änderungen am Code transparenter

Der lange Weg

- Unit Tests ausprobieren
- kleine Tests schreiben
- oder auch: erst Code, dann Test schreiben
- typisch: man schreibt Tests, die zum Code passen könnten
- Lernkurve enorm

Nachteile

- (naja) hohes Anforderungsverständnis
- nur erkannte Randbedingungen spiegeln sich in Tests wieder
- Testcode, der nicht in die App gelangt
- Tests garantieren keine Fehlerfreiheit
- Problem bei Datenbanken, Netzwerk, Nebenläufigkeit

Nettigkeiten

- Zwang, die Anforderungen zu verstehen
- Teamarbeit: Tester - Coder
- weniger überflüssiger Code
- Tests dienen der Dokumentation
- Last but not least: Viele Grüne Tests ergeben ein gutes Gefühl

Worauf teste ich?

Worauf teste ich?

XCTest/XCTestAssertionsImpl.h (1)

```
XCTFail(format...)
XCTAssertNil(a1, format...)
XCTAssertNotNil(a1, format...)
XCTAssert(expression, format...)
XCTAssertTrue(expression, format...)
XCTAssertFalse(expression, format...)
XCTAssertEqualObjects(a1, a2, format...)
XCTAssertNotEqualObjects(a1, a2, format...)
XCTAssertEqual(a1, a2, format...)
XCTAssertNotEqual(a1, a2, format...)
XCTAssertEqualWithAccuracy(a1, a2, accuracy, format...)
XCTAssertNotEqualWithAccuracy(a1, a2, accuracy, format...)
```

Worauf teste ich?

XCTest/XCTestAssertionsImpl.h (2)

```
XCTAssertThrows(expression, format...)
XCTAssertThrowsSpecific(expression, specificException, format...)
XCTAssertThrowsSpecificNamed(expression, specificException,
exception_name, format...)
XCTAssertNoThrow(expression, format...)
XCTAssertNoThrowSpecific(expression, specificException, format...)
XCTAssertNoThrowSpecificNamed(expression, specificException,
exception_name, format...)
```

Worauf teste ich?

XCTest/XCTestAssertionsImpl.h (3)

```
XCTAssertNil(a1, format...)
XCTAssertNotNil(a1, format...)

XCTAssertTrue(expression, format...)
XCTAssertFalse(expression, format...)
```

Worauf teste ich?

XCTest/XCTestAssertionsImpl.h (4)

```
XCTAssertEqualObjects(a1, a2, format...)
XCTAssertNotEqualObjects(a1, a2, format...)

XCTAssertEqual(a1, a2, format...)
XCTAssertNotEqual(a1, a2, format...)

XCTAssertEqualWithAccuracy(a1, a2, accuracy, format...)
XCTAssertNotEqualWithAccuracy(a1, a2, accuracy, format...)
```

Demo

Demos

- Ganz einfach
- Unit Test nachrüsten
- TicTacToe mit Überraschungen

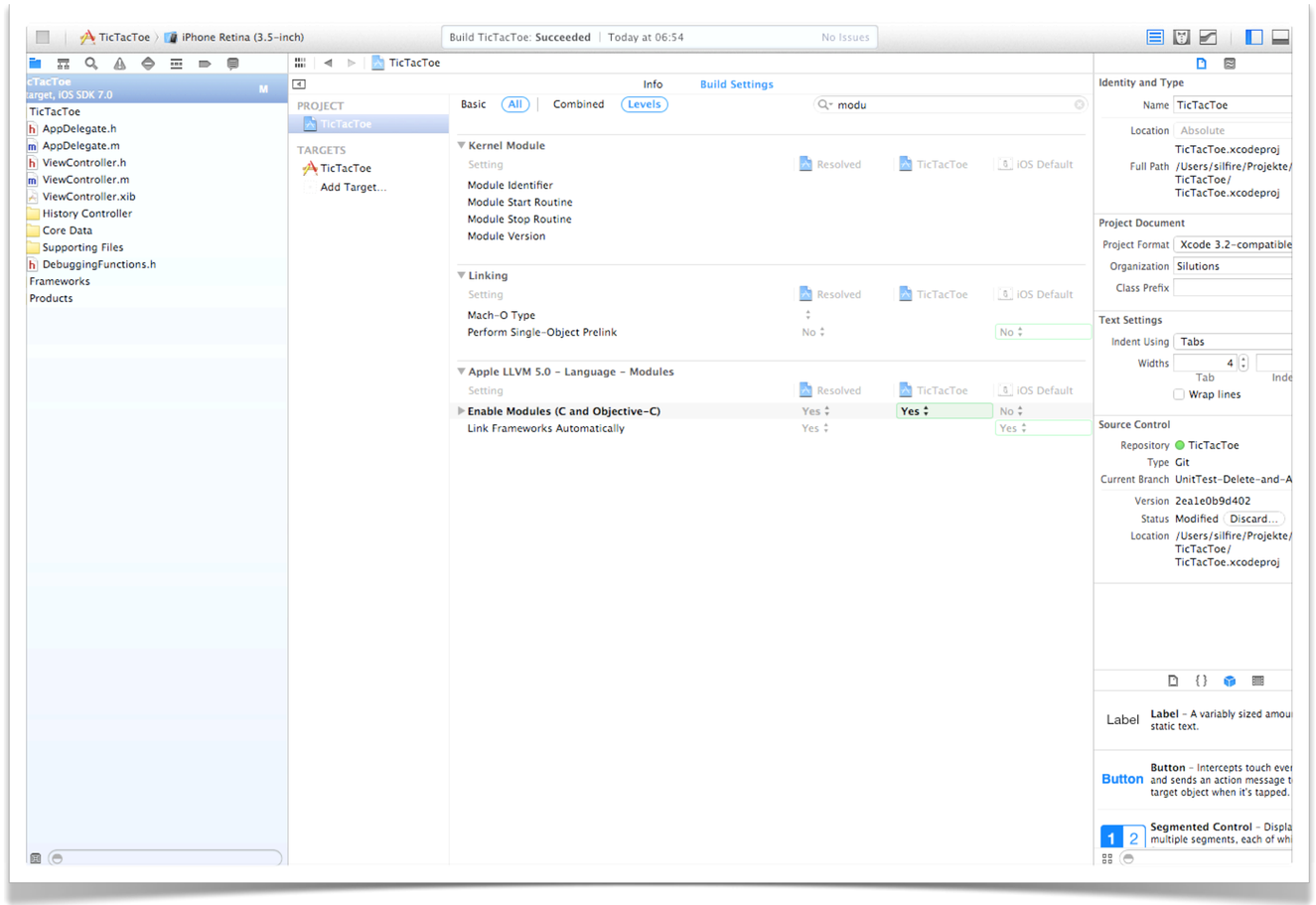
Ganz einfach

- Simple Calculator

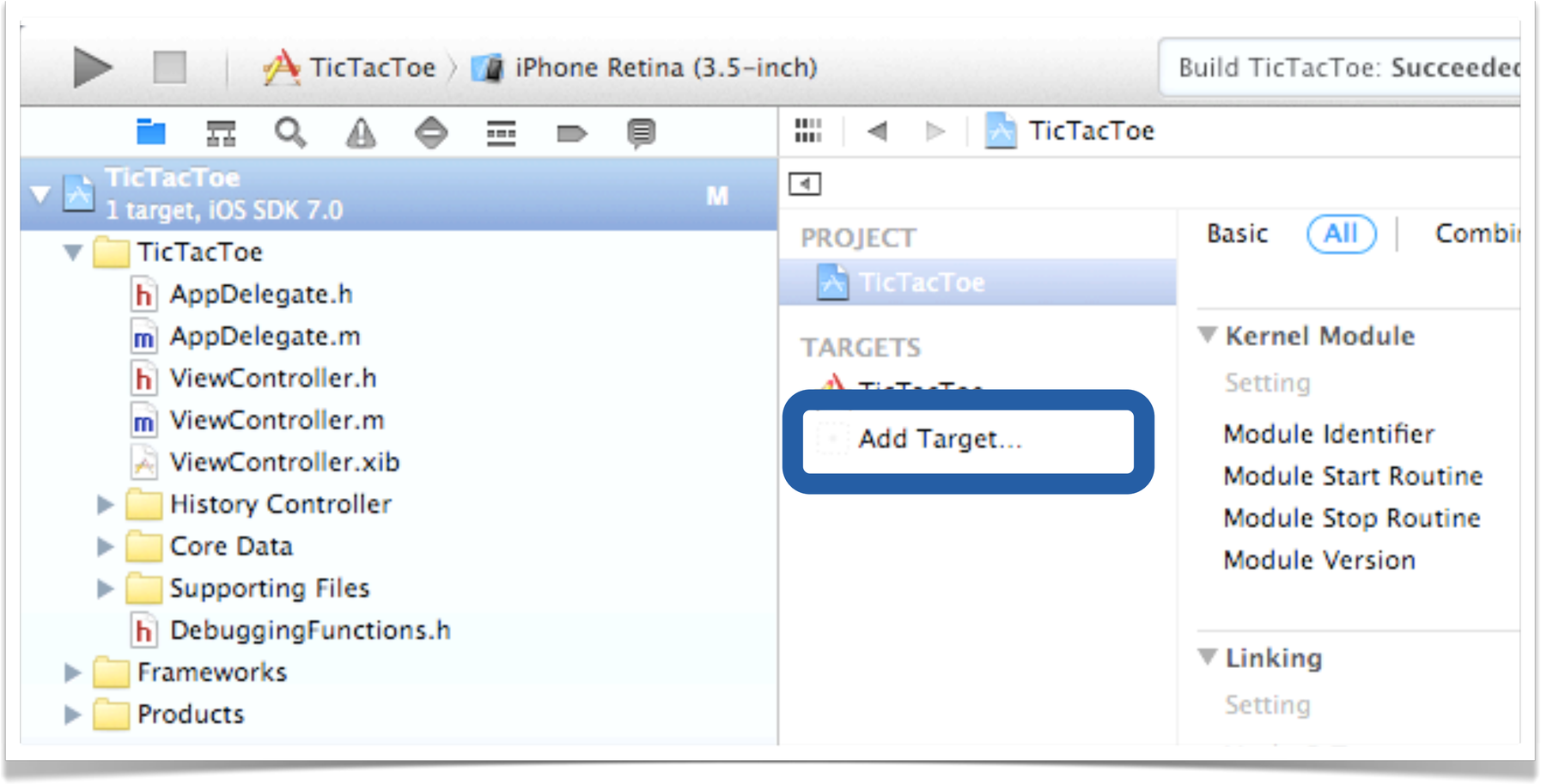
Unit Test nachrüsten

- Framework in Xcode 5: XCTest

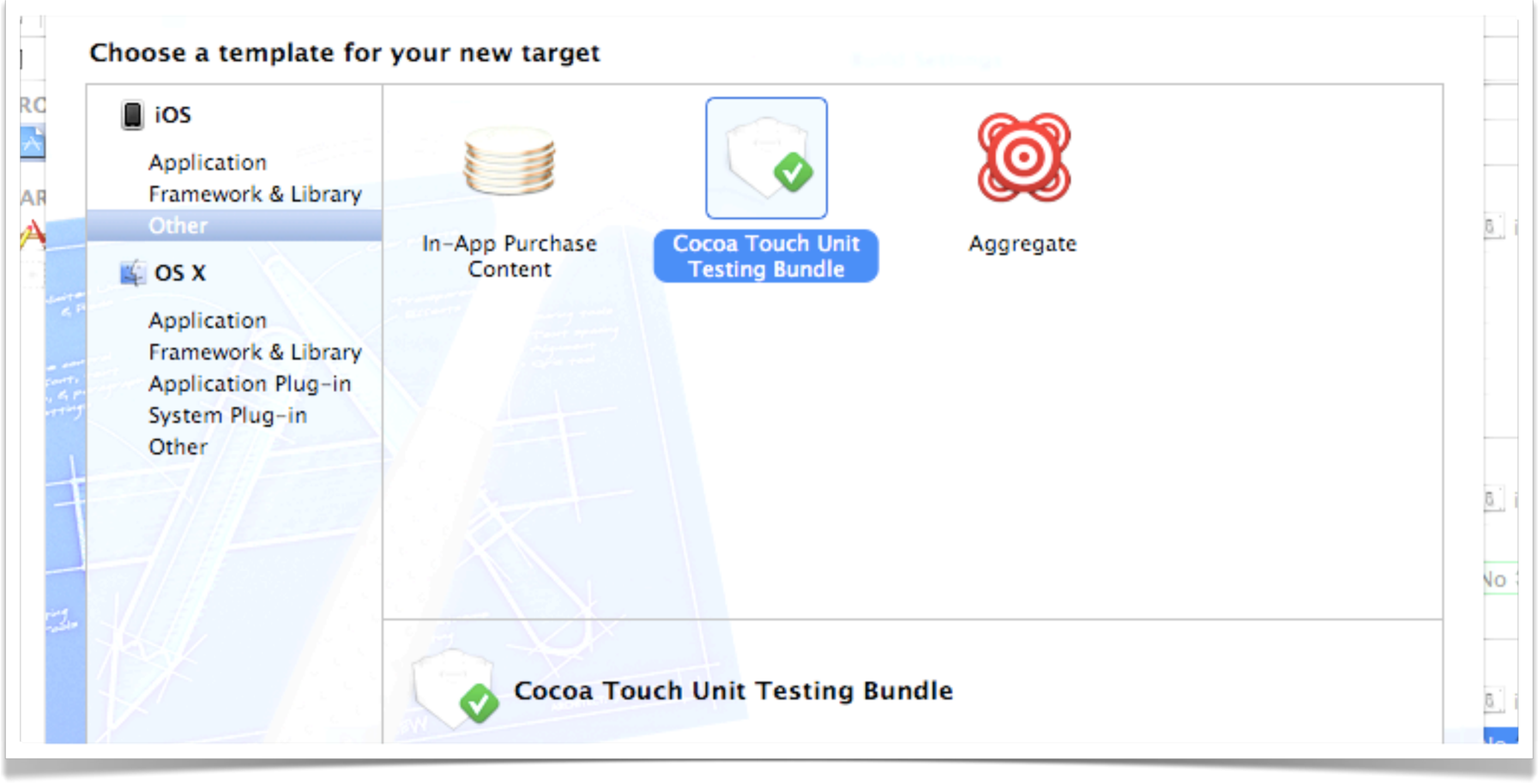
Projekt in Xcode 5



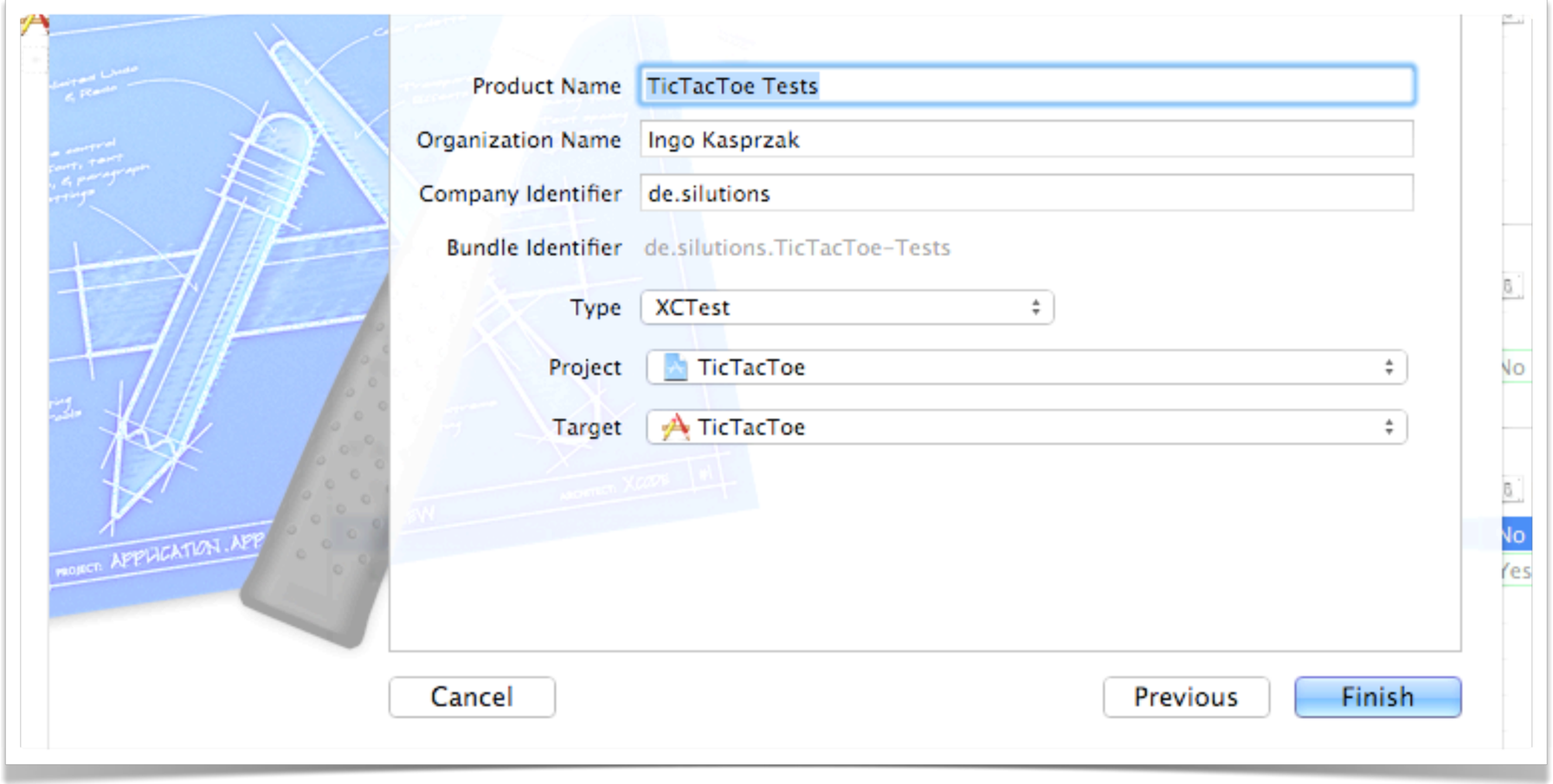
Target hinzufügen



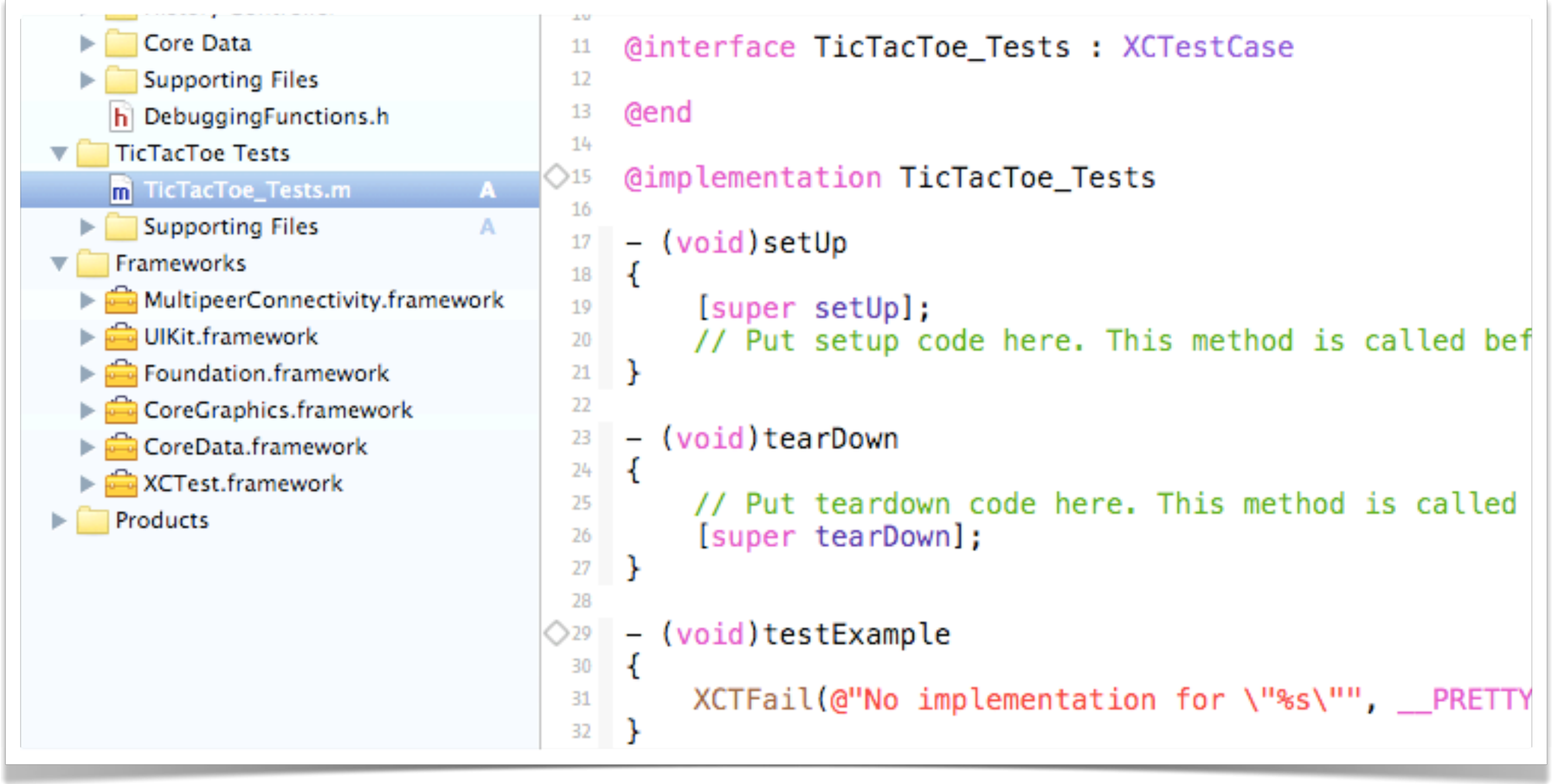
Testing Bundle



Target Namen wählen



Das war‘s ...


Core Data
Supporting Files
DebuggingFunctions.h
TicTacToe Tests
TicTacToe_Tests.m
Supporting Files
Frameworks
MultipeerConnectivity.framework
UIKit.framework
Foundation.framework
CoreGraphics.framework
CoreData.framework
XCTest.framework
Products
@interface TicTacToe_Tests : XCTestCase
@end
@implementation TicTacToe_Tests
- (void)setUp
{
[super setUp];
// Put setup code here. This method is called bef
}
- (void)tearDown
{
// Put teardown code here. This method is called
[super tearDown];
}
- (void)testExample
{
XCTFail(@"No implementation for \"%s\"", __PRETTY
}

TicTacToe

- Demo von Macoun 2012
- Unit Test nachgerüstet
- überraschende Ergebnisse

Fragen?

Tipp

Der wichtigste Test überhaupt ist der,

den man zuerst schreibt.

Buchempfehlung

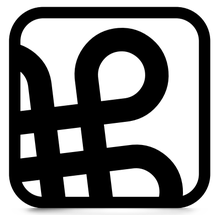
Graham Lee

„Test-Driven iOS Development“

One more thing

http://www.0x02100.de

Vielen Dank



Macoun