



# הנדסת תוכנה

## 9. בדיקות III

### Test Doubles (Mock Objects)

Pragmatic Programmer Tip :

**Test Early. Test Often. Test Automatically.**

Tests that run with every build are much more effective than test plans that sit on a shelf.

# מה היום?

- ראינו: בדיקות <- בדיקות יחידה, Test Driven Development
- טיפול בתלויות
- הדגמה \ המשך שב 4
- הרצאה 3\תרגיל: סקרי סבב (+ בדיקות?)

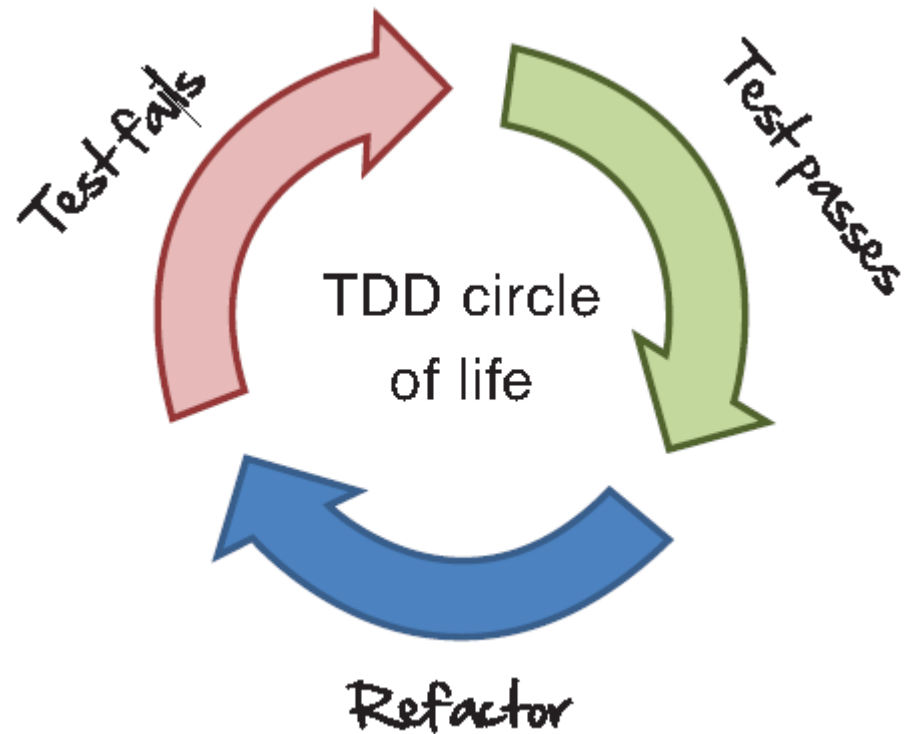
# מקורות

- Meszaros, xUnit Test Patterns: Refactoring Test Code, '07
- [Mock Roles, not Objects](#), '04
- Fowler, [Mocks Aren't Stubs](#)
- Osherove, "Interaction testing with mock objects" The art of unit testing, '09
- Unit Testing with Python, Pluralsight Course, module 5

# תזכורת: בדיקת יחידה טובה

- בדיקת יחידה היא קוד שקורא לקוד אחר ובודק אח"כ נכונות של טענות מסוימות על ההתנהגות הלוגית של מתודה או מחלקה.
- בדיקת יחידה תכתב בד"כ באמצעות framework
- קצרה ומורצת בקלות
- **FIRST**

# תזכורת: TDD Cycle

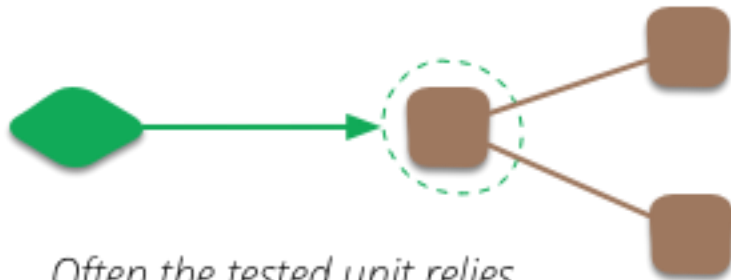


# איך בודקים כשיש תלות בגורמים חצוניים?

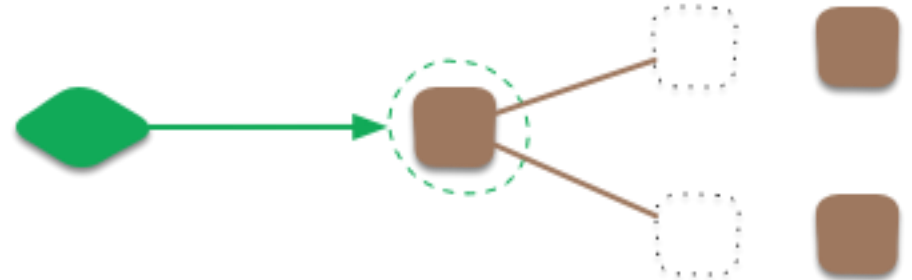
- מחלקות אחרות (שעוד לא קיימות \ BDD)
- גורמים חיצוניים (למשל File System, Database, Services): איטיים, לא עקביים



# Test Isolation



*Often the tested unit relies  
on other units to fulfill its  
behavior*



*Some unit testers prefer to  
isolate the tested unit*

<http://martinfowler.com/bliki/UnitTest.html>

# הדגמה

- מפרויקט?: תלות בבסיס נתונים של לקוחות...
- Tang, Unit Testing and TDD in Node.js – [Part 1](#), [Part 2](#) (future), [repo](#).
- המשך משימה אישית

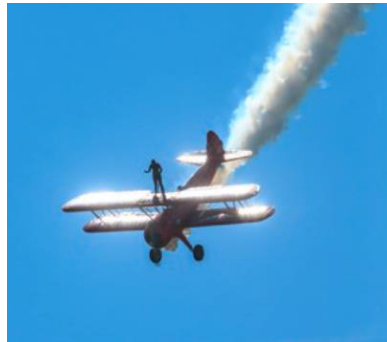
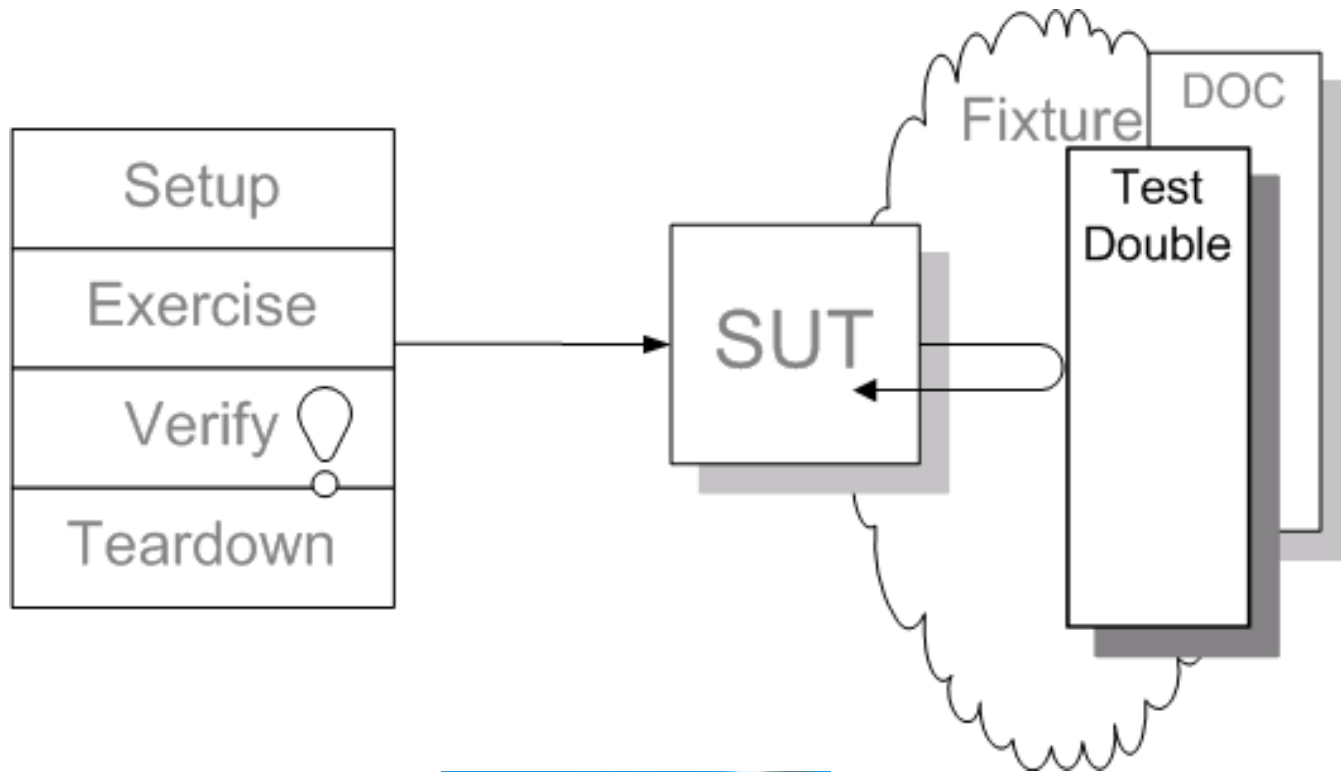
Insert Web Page

Preview

[Web Viewer Terms](#) | [Privacy & Cookies](#)

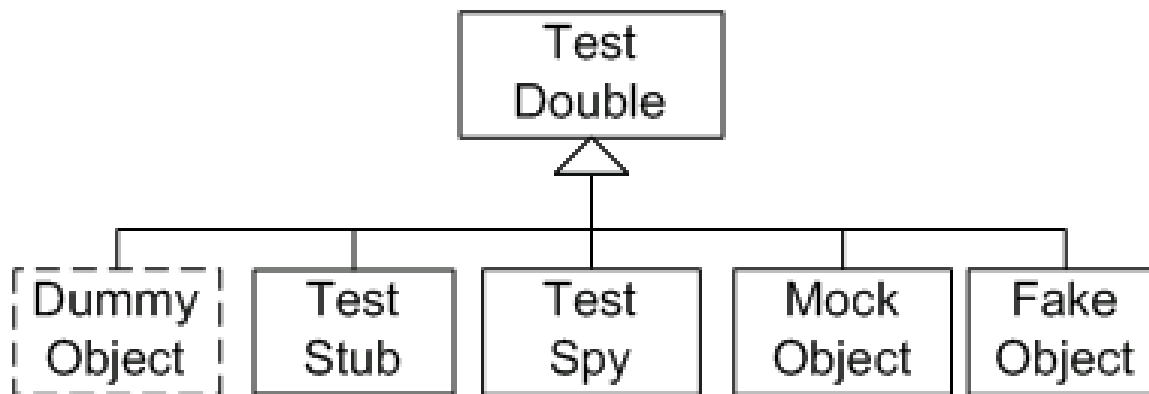


# Test Doubles

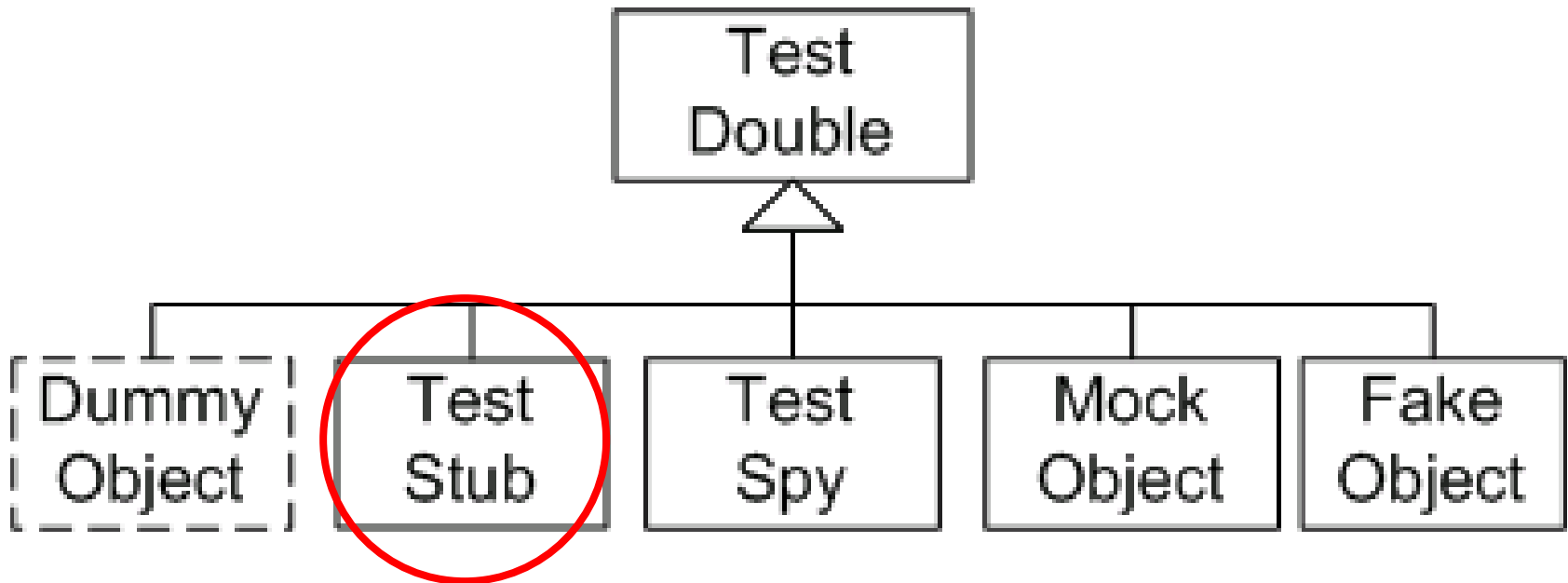


# Test Doubles

- By Gerard Meszaros ([xunitpatterns.com](http://xunitpatterns.com), [G tech-talk](#))
- Test Doubles – שם כללי לאובייקטים שמחליפים אובייקטים אמיתיים, לצרכי בדיקה

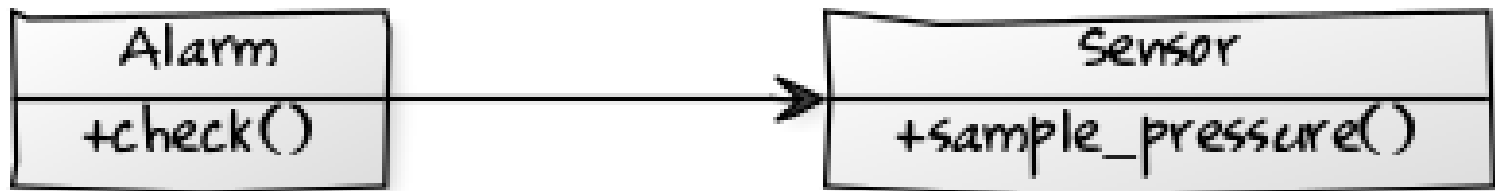


# Test doubles



# Stub – Car Example

- בדיקה של Alarm
- ללא חיישן אמיתי!



# Stub - Code

- <https://gist.github.com/robiny/79169db54c5b24f42fc1>

# Stub

```
public class StubRepo : IOwnerRepository
{
    public IOwner FindById(int id){}

    public IOwner Save(IOwner owner)
    {
        return new Owner();
    }

    public void Delete(IOwner owner){}
}
```

# Fake

```
public class FakeRepo : IOwnerRepository
{
    IList<IOwner> _owners = new List<IOwner>();
    int _idCounter = 0;

    public IOwner Save(IOwner owner)
    {
        owner.Id = _idCounter++;
        _owners.Add(owner);
        return owner;
    }

    public void Delete(IOwner owner)
    {
        var ownerToDelete = _owners.FirstOrDefault(o => o.Id == owner.Id);
        _owners.Remove(ownerToDelete);
    }
}
```

# Spy

```
public class SpyDefaultView : IDefaultView
{
    public SpyDefaultView()
    {
        ShowWasCalled = false;
    }

    public void Show(DefaultVM model)
    {
        ShowWasCalled = true;
    }

    public void ShowError(string err)
    public void Redirect(string url){}

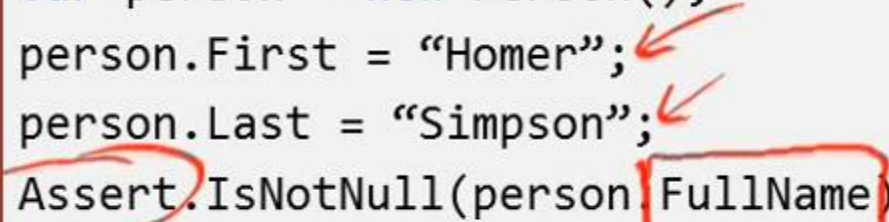
    public bool ShowWasCalled { get; set; }
}
```

```
Assert.IsTrue(spy.ShowWasCalled);
```



# Dummy

```
var person = new Person();  
person.First = "Homer";  
person.Last = "Simpson";  
Assert.IsNotNull(person.FullName);
```



# Mock Object (אובייקט מדומה)

- אובייקט הנוצר ע"י ספריה, ניתן לקנפג את האובייקט להחזיר ערכים על פעולות, **לודא שפעולות מסוימות נקראו ועוד.**

- בד"כ נרצה להשתמש בספריות, לדוגמא:

Java: mockito, jMock, EasyMock,  
.Net: Nmock, moq, RhinoMock, Isolator,  
Nsubstitute, FakeItEasy, NUnit ...  
Python: unittest.mock

- בד"כ יכולות לשמש ליצירת Test Doubles שונים
- (עוד בתיכון מונחה עצמים)

# מה אינה מטרה של mock objects?

1. לבדוק אם האובייקט הנבדק מתקשר נכון עם סביבתו

2. לאתחל ולהריץ את התלויות של אובייקט באופן אוטומטי

3. להגיע לכיסוי קוד גבוה ע"י דימוי סביבת האובייקט

4. לאפשר לבדוק גם כשתלויות עדיין חסרות

# סיכום הסוגים

- Stub – מחזיר תשובה צרובה לכל שאילתא, ללא לוגיקה
- Fake – מימוש אמיתי אך פשוט יותר
- Spy – מאפשר לבדוק מה קרה במהלך הבדיקה
- Mock – כולל את הקודמים ומשמש לבדיקת התנהגות
- Dummy – ערך שנדרש עבור קריאה לממשק, אך לא משמש את הבדיקה

# האם כדאי להשתמש?

- בעד

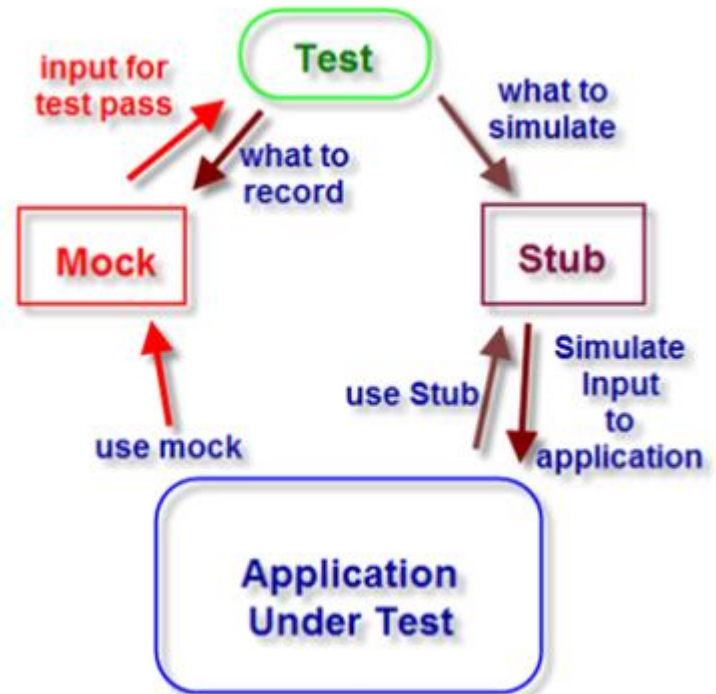
- מהירות ריצה, בידוד מתלויות
- מאפשר תיכון מתמשך (או כשחסרים חלקים)
- הפרדה וחלוקת אחריות

- נגד

- סיבוכיות וקריאות
- לא בודקים את הדבר האמיתי
- מצריך נסיון בתיכון
- בקוד קיים (legacy) לא תמיד אפשרי (אבל יותר קל בשפות דינמיות \ מנוהלות – monkeypatching)

# עוד מקורות

- Fowler, [Mocks Aren't Stubs](#)
- Roy Osherove, [Mocks and Stubs - The difference is in the flow of information:](#)



# דוגמאות נוספות

- Osherove, [TDD Kata 2 – Interactions](#)
  - Mocks and stubs
  - git init - [repo](#)
  - [Kata cast](#) (.net)

# נושאים נוספים

- מאפיינים שונים של Unit x (אתחולים, חריגות, ...)
- אינטגרציה\ממשק משתמש
- פרמטרים
- כיסוי
- Continuous Integration \ אוטומציה
- בדיקות לקוד קיים (Legacy Code)
- קוד מובייל \ ענן \ ווב
- כיצד להטמיע TDD בארגון?
- Katas, pexforfun
- עוד בקורס בדיקות תוכנה (אינטל)



# בשבוע הבא

- תיכון מתמשך (מבוא לתיכון מונחה עצמים)
- עוד גיט?
  - נושאים נוספים
- פרויקט – סבב 2
  - ~~סקר בדיקות (שבוע לפני סוף הסבב)~~



Sandro Mancuso

@sandromancuso



Following

I believe software design should be taught before TDD. TDD can't lead to good design if we don't know what good design looks like.



RETWEETS

241

FAVORITES

127



3:47 AM - 16 Apr 2015

# סיכום

- בדיקות בהינתן תלויות
- בדיקת מצב מול התנהגות
- למתחילים מומלץ להסתפק בערכים מוחזרים ומצב
- הקשר לתיכון
- לוקח זמן עד שמקבלים רווח
- תרגול ולימוד (<->) מתמשכים, ...Code retreats
- Google Code, 2012: [Stop Mocking, Start Testing \(video\)](#), “Mock objects tell you what you want to hear”
- בדיקות ואנחנו