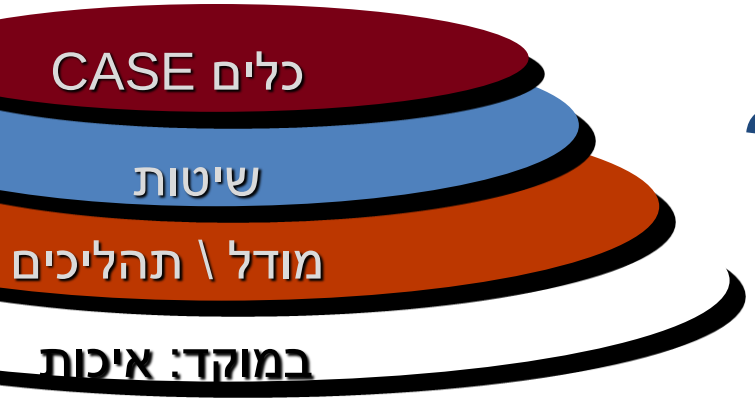
Azrieli
College of Engineering
Jerusalem

עזריאלי
מכללה אקדמית להנדסה
ירושלים

הנדסת תוכנה
8. בדיקות - II
הדגמה, בדיקות יחידה

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.

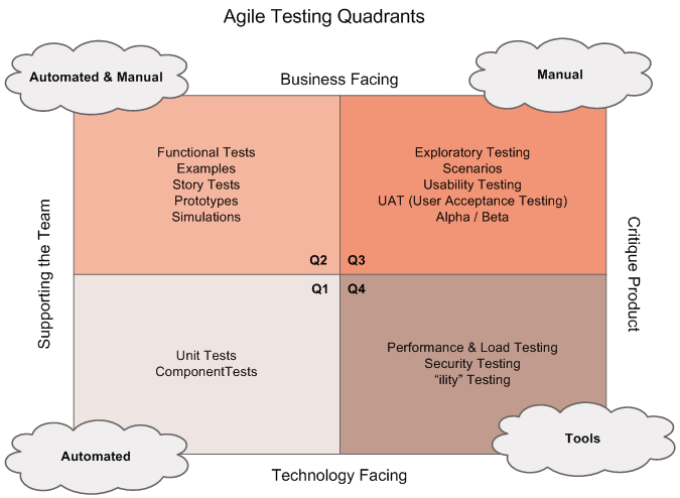


מה היום?

- בדיקות (חזרה)
- הדגמה
- פרויקט
  - מצגות סבב 1
  - מעבר לסבב 2

אילו בדיקות?

- דיבאג (ניפוי שגיאות)
- בדיקות יחידה (unit test)
- בדיקות עומס, בטיחות, גישוש, שמישות, A/B ועוד
- בדיקות אינטגרציה
- בדיקות קצה לקצה
- בדיקות מערכת
- בדיקות קבלה
- בדיקות רגרסיה
- סקרי קוד...



- 100 Types of Software Testing You Never Knew Existed

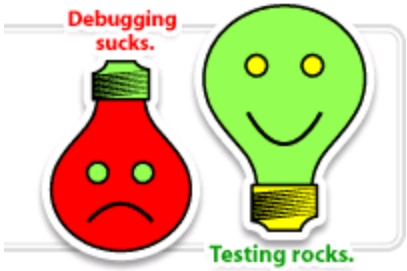
ננסה להתמקד

- בדיקות קצה לקצה (משתמש\קבלה\פונקציונליות)
  - האם המערכת עובדת בשלמותה?
  - בפרויקט: ניסוח בדיקה באמצעות תרחש או סיפור
- בדיקות אינטגרציה
  - האם הקוד שכתבנו עובד מול קוד אחר
  - האם אי אפשר להסתפק בסוג הראשון?
- בדיקות יחידה (מפתח)
  - האם המודולים עושים את הדבר הנכון? נוחים לשימוש? ע"י מי?

בדיקת יחידה

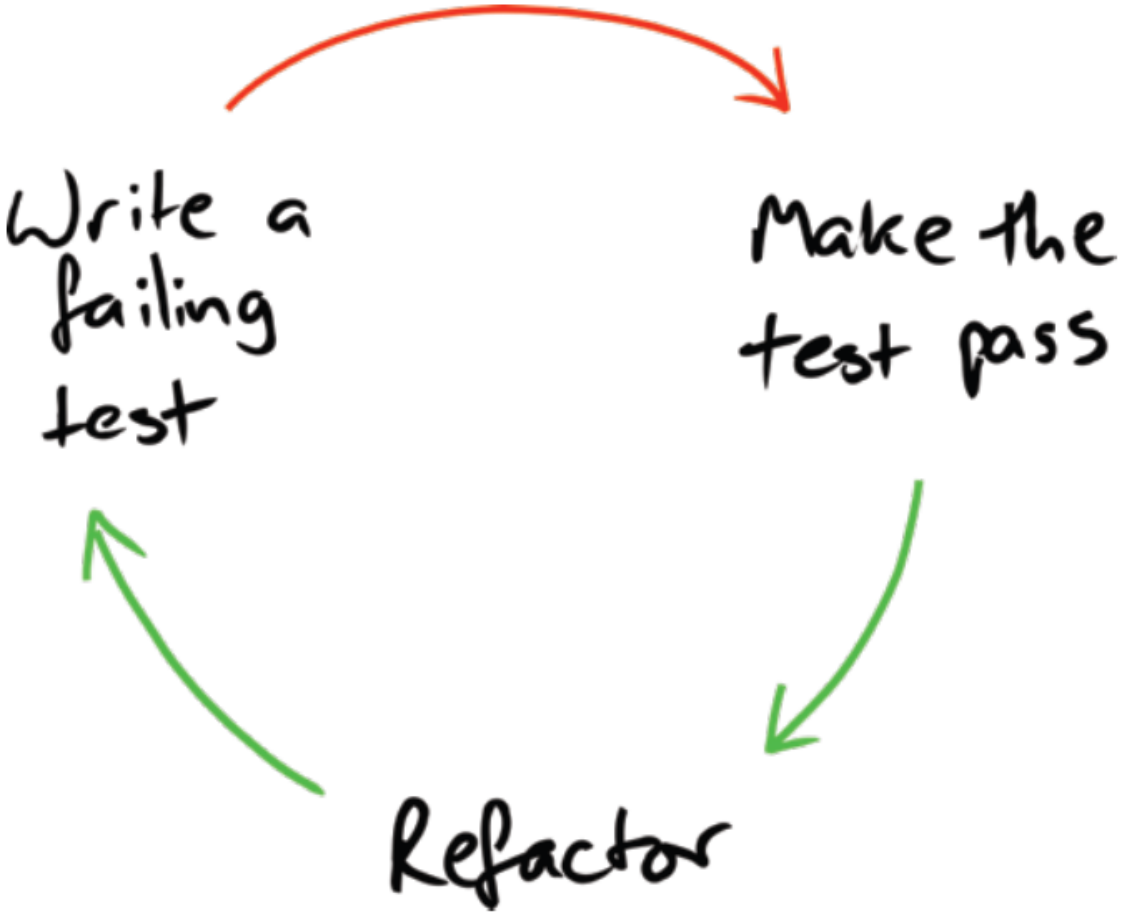
- הגדרה: בדיקת יחידה היא קוד שקורא לקוד אחר ובודק אח"כ נכונות של טענות מסוימות. "יחידה" היא "קטנה" בד"כ פונקציה, מתודה
  - בד"כ נכתבת באמצעות framework (בהמשך)
- System Under Test (SUT) – הדבר שאותו אנחנו בודקים

בדיקות יחידה (אוטומטיות)



- יתרונות
  - נכונות (ובמיוחד בשפות דינמיות)
  - פחות זמן ב-debugger, רגרסיה
  - תיעוד "חי"
  - לעומת בדיקות אחרות: קלות ומהירות
  - מאפשרות בדיקות ידניות משמעותיות יותר
    - הורדת עלויות \ אפשרות לשינויים ...
- חסרונות
  - קוד (תחזוקה, תיכון, בדיקות – כיצד נמנע זאת?)
  - זמן לימוד, כתיבה והרצה (אולי נסתפק באינטגרציה וקבלה?)
  - יכולות לתת תחושת בטחון מזויפת
  - לא תמיד קל עבור קוד קיים (legacy)

TEST DRIVEN DEVELOPMENT



http://amzn.to/atddbook

יתרונות TDD

- כיסוי טוב יותר ואוטומטי (ופחות באגים)
- תיכון: Test Driven Design, פשטות, חשיבה כלקוח (ראשוני של הקוד API), התמודדות עם הטיית אישור
- תיכון מתמשך – מודולריות, צמידות נמוכה, YAGNI, אפשור שינוי
- דיבאג מוקדם (מה קורה עם משאירים לסוף? אס"ק)
- חסרונות? (לעומת Test After Development)

James Shore: “14 years now and I’m continually refining my understanding of how to do that well”

חסרונות TDD

- השקעה בממשק (api) על חשבון פונקציונליות?
- לא מכסה דרישות (ראו BDD)
- עקומת למידה, כולל שיטות משלימות
- מצריך שיתוף פעולה ועבודת צוות
- כמה להשקיע מראש? כיצד מודדים?
- לא מהווה תחליף לחשיבה...(cargo cult programming, " TDD doesn't create design. You do"), קשה יותר בבעיות חדשות


Write a failing end-to-end test
Write a failing unit test
Make the test pass
Refactor
Deployable System

מהי בדיקת יחידה טובה?

Unit tests should be FIRST

- **F**ast
- **I**ndependent / Isolated
- **R**epeatable
- **S**elf-checking/verifying
- **T**imely

Unit tests should be FIRST

(adopted from A. Fox, Berkeley)

- **F**ast: run (subset of) tests quickly (since you'll be running them *all the time*)
- **I**ndependent: no tests depend on others, so can run *any subset* in *any order*
- **R**epeatable: run N times, get same result (to help isolate bugs and enable automation)
- **S**elf-checking: test can *automatically* detect if passed (*no human checking* of output)
- **T**imely: written about the same time as code under test (with TDD, written *first!*)

xUnit Framworks

- כלים לבדיקות יחידה
- ‘94, Kent Beck, SUnit– Small Talk
- ~’00, +E. Gamma, JUnit ("Test Infected")
- ייצוא לשפות רבות: CppUnit, PyUnit ועוד
  - http://www.xprogramming.com/software
  - http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
- ארכיטקטורה סטנדרטית לבדיקות יחידה

רכיבים עיקריים בקוד בדיקה (Java JUnit)

```
// Unit Test
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalcTest {
 @Test
 public void testAdd() {
   Calc calc = new Calc();                    // Arrange
   int result = calc.add(2, 3);               // Act
  assertEquals(5,result);                     //Assert
 }
}
// SUT
public class Calc {  public int add(int a, int b) { return a+b; } }
```

רכיבים עיקריים בקוד בדיקה (python unittest)

```
import unittest

class CalcTest(unittest.TestCase):
    def test_add(self):
        result = Calc().add(2, 3)
        self.assertEqual(5, result)

# test runner
if __name__ == '__main__':
    unittest.main()
```

Python Tools

- Tools:
  - Library: unittest (2.7)
    - Other: PyTest, Nose, doctest
  - Runners: CLI*, IDSs (Eclipse, PyCharm), Continuous Testing: pytddmon, CI, …
- Start with simplest, e.g. Calculator

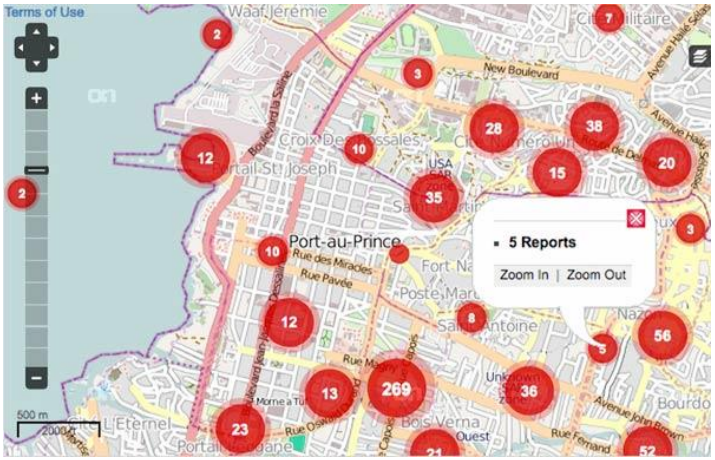
- * Also cloud based, e.g., pythonanywhere

Simple TDD

- Calc
- HW4 Fine a Person
- Nand2Tetris?
- (Nice TDD and Junit intro slides)

תרגיל – מיפוי ע"י חכמת המונים

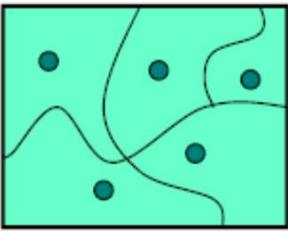
- Ushahidi: CrowdMap Basic Features:
  - **Create a post**
  - **Create a Map**
  - **Find Posts**
  - **Find a Map**
  - **Collaborate**
  - **Find a Person**: This feature will be added at a later date. Stay Tuned
  - …



Feature: Find a Person!

- Given a person name, return all posts (of a map) containing it’s name (in any of a post fields)
- Given a name, check if the map includes a location information (place or geo. location)
- Check if there are map inconsistencies, e.g., the same name with different locations.
- Example: Or (R.I.P) appears at [27°59′N 86°55′E] but also at [31°47′N 35°13′E]
- Updated version repo.

בהרצאת המשך \ נושאים מתקדמים



- בדיקות יחידה 2.0 למשל...
  - מאפיינים מתקדמים של xUnit: אתחולים, חריגות,
  - מחלקות שקילות, קופסא שחורה\לבנה, פרמטרים, כיסוי, תלות, אינטראקציה עם רכיבים אחרים, התנהגות מול מצב
  - כלים נוספים, אוטומציה, Continuous Integration
  - בדיקות לניידים \ ענן \רשת \ UI וכו'
  - כיצד למצוא את הבדיקה הבאה
  - בדיקות לקוד קיים...
- משימה אישית: TDD
- קריאה מומלצת להרצאת המשך: Using Mock Objects

לסיכום

- בדיקות, פיתוח מונחה התנהגות
- בפרויקט
  - בכל סבב: ניסוח בדיקת קבלת לתרחיש עיקרי
  - סבב 2: חליפת בדיקות לרכיב מרכזי (וכתמיכה בשלד המוצר)