



**ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT**

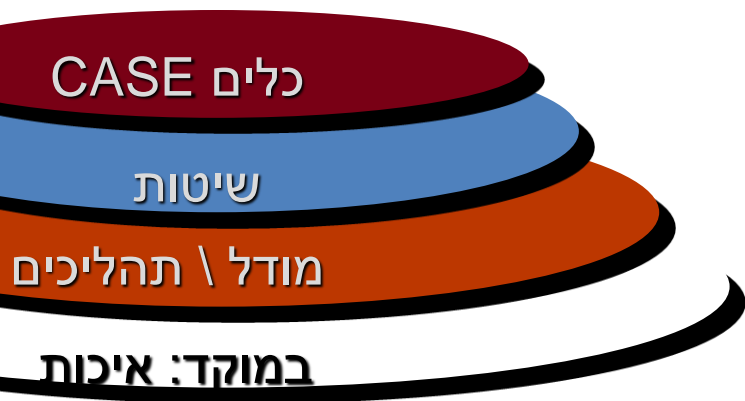
CODESMACK

הנדסת תוכנה

9. בדיקות - I

בדיקות יחידה

[Pragmatic Programmer Tip](#) : **Design to Test**
Start thinking about testing before you write a line of code.



השבוע

- בדיקות (מבוא)
 - בדיקות יחידה (Unit Testing), פיתוח מונחה בדיקות (TDD)
 - כלי בדיקה (למשל Junit, mocha)
- הדגמה
- מעבדת בדיקות יחידה
 - משימה אישית 4: TDD ו-בדיקות יחידה
 - השלמת git
- פרויקט: המשך\סיום סבב 1-MVP
 - הצגה, סקר קוד, משימות, רטרוספקטיבה ותכנון לסבב 2
 - פגישות וסקרים, ציון לפי איכות והתקדמות, **הערכת עמיתים**
 - פרויקט: משימת סבב 2 לדוגמא: בדיקות יחידה

איפה אנחנו בפרויקט (בקורס)?

- למה?
בעיה (פלט: הצעת פרויקט\חזון\SOW)
- מי?
צוות (Inception, אתחול\תכנון פרויקט)
- מה?
דרישות (SRS)
- איך?
תיכון (ארכיטקטורה) (SDS)
- מתי?
תכנון וניהול – (ZFR)

• בניה

- בקרת תצורה
- בדיקות
- קוד קיים\תיכון מונחה עצמים
- כלי הנדסת תוכנה
- שמישות





The Marker 25/11/12

- "הליכוד היו מוכנים לפער של מיליון שקל בין ההצעות אבל לא יותר, ואמן יועצים נבחרה. מערכת המחשוב הותקנה בחווה של ספק חיצוני – אחת מספקיות ההוסטינג בישראל. זה דבר **שלא נעשה עד כה** במערכות הבחירות. 1,400 תחנות קצה חוברו אל ספק ההוסטינג. לא בוצעו **בדיקות עומסים** לתוכנה, לא נעשו **בדיקות לגיבוי** של מערכות התקשורת במעבר מתקשורת קווית לתקשורת סלולרית. יש סניפים שגם בהם הצידוד עצמו היה **תקול**", הוסיף הבכיר.

מקורות

- Pressman ch. 16-17
- Beck, Test Driven Development by Example
- Osherov, The Art of Unit Testing
- Freeman & Pryce, Growing Object-Oriented Software Guided by Tests
- Rasmusson, Agile Samurai, ch. 12, 14
- Jenkins , [A Software Testing Primer](#), 2008
- Rasmusson, [The Way of the Web Tester](#)
Automate Your Tests, Reduce Bugs, Increase Quality,
2016

קישורים

- Introduction to Test Driven Development
<http://www.agiledata.org/essays/tdd.html>
- [Software Testing - How to Make Software Fail](#) (Udacity course)
- [Unit Tests Are FIRST](#), 2012
- R. Martin, Craftsman Series,
<http://www.objectmentor.com/resources/publishedArticles.html> -> Craftsman Series

Online Books

- Safari Books @ JCE Library
 - [Test-Driven JavaScript Development](#)
 - Amodeo 2015, [Learning Behavior-driven Development with JavaScript](#)
- Test-Driven Development - Extensive Tutorial. Open Source ebook
<https://github.com/grzesiek-galezowski/tdd-ebook>

סיום סבב: מעט על רטרוספקטיבה

- מטרה (ר' עוד בהרצאת סקראם)
- שיתוף ושקיפות, משוב ושיפור מתמשך
- שאלות

– במה הצלחנו?

– היכן היו קשיים?

– מה נרצה להמשיך לעשות?

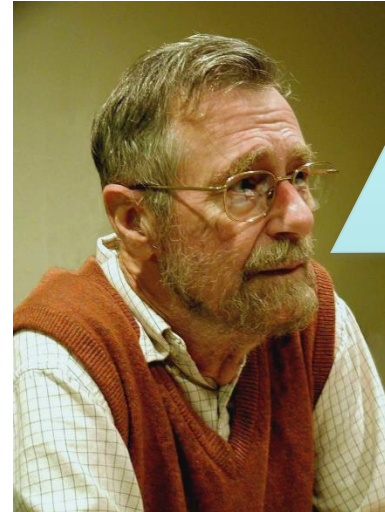
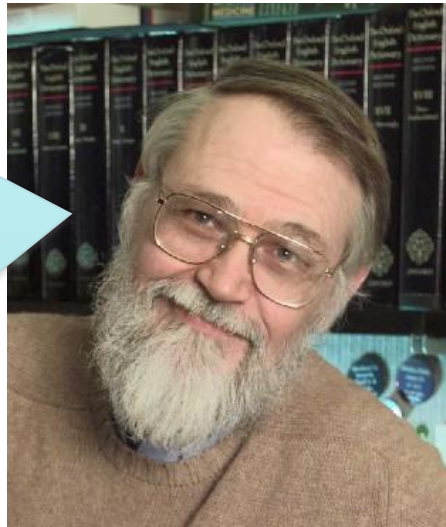
– מה להפסיק? מה לשנות?

- לוח לדוגמא



Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

se18a-yagel



Testing can never demonstrate the _____ of errors in software, only their _____

Source: ¹⁰[saas](#)

בדיקות תוכנה



- למה לבדוק?
- איך לבדוק?
- אילו בדיקות?
- מי בודק? מתי?
- איך? הדגמה ומשימה אישית
- כמה לבדוק? Boeing 777 wing break test
- האם זה באמת כדאי?
- מושגים נוספים
- למה עכשיו?

למה לבדוק?

• נכונות ותקפות Verification & Validation

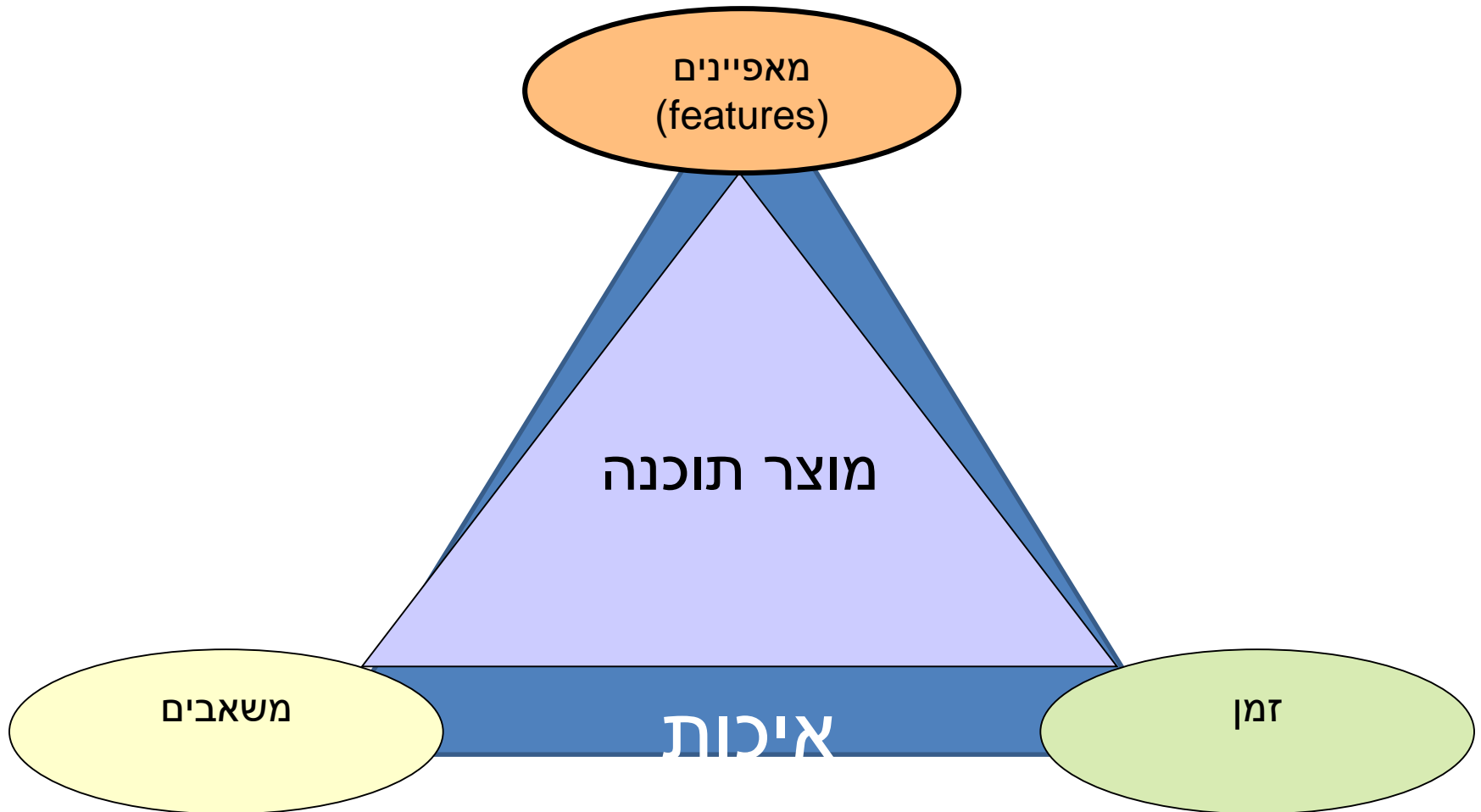
– איכות

– רגרסיה

– לאפשר שינויים

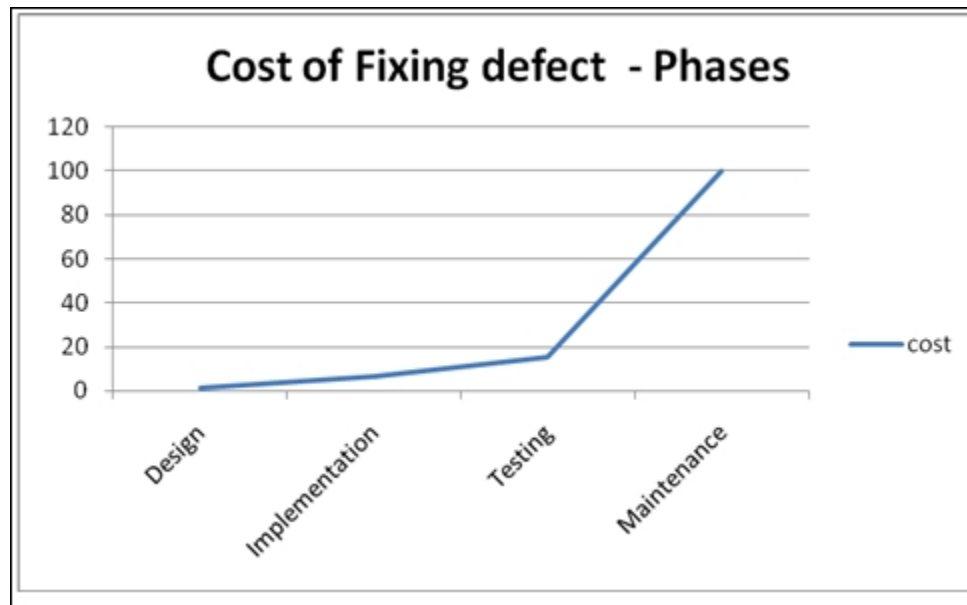


תזכורת: פרויקט תוכנה:



למה לבדוק?

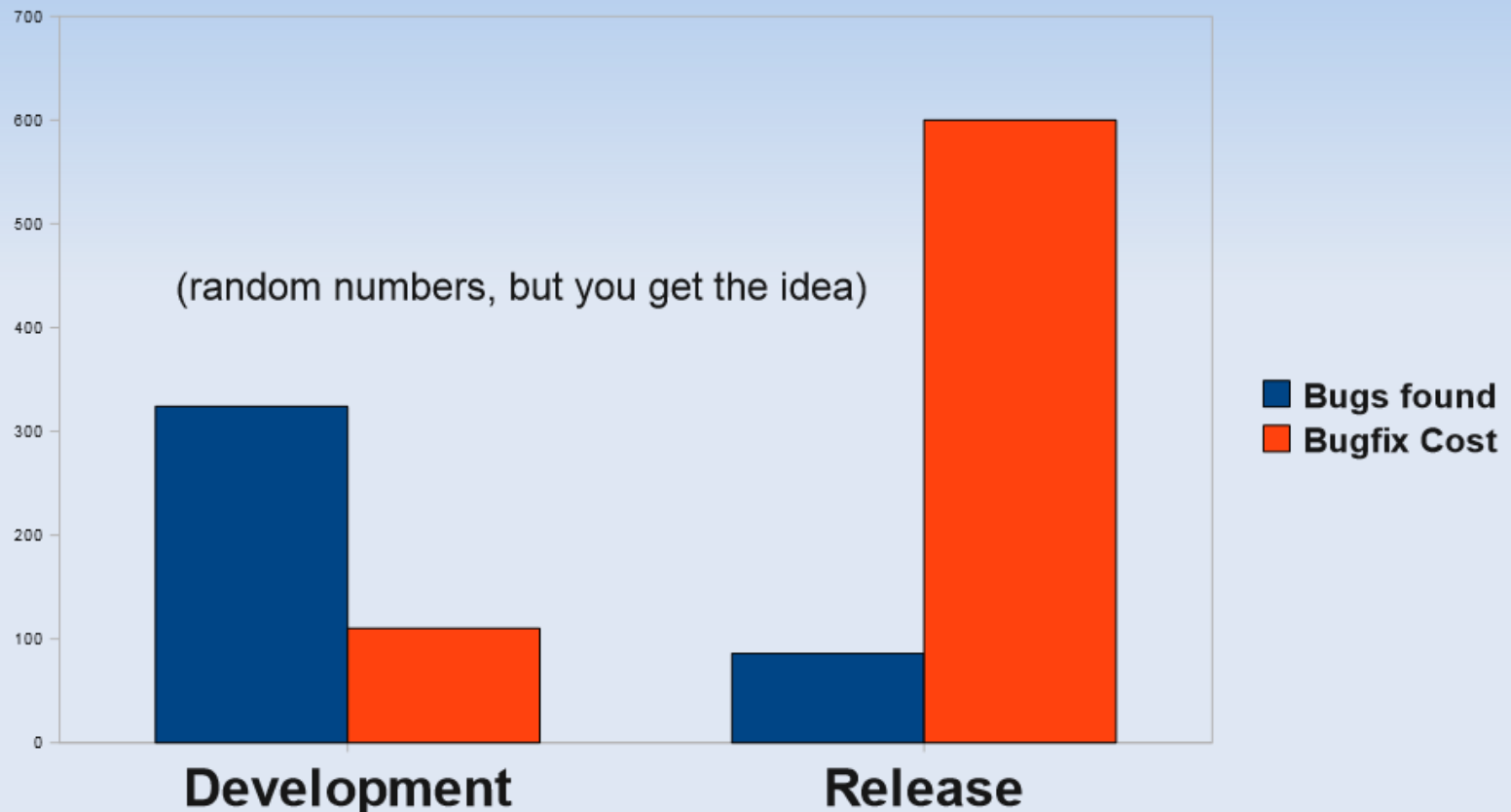
- NIST 2002 [Research](#):
 - software bugs cost the U.S. economy around \$60 billion annually
 - With better testing, more than one-third of the cost could be avoided.



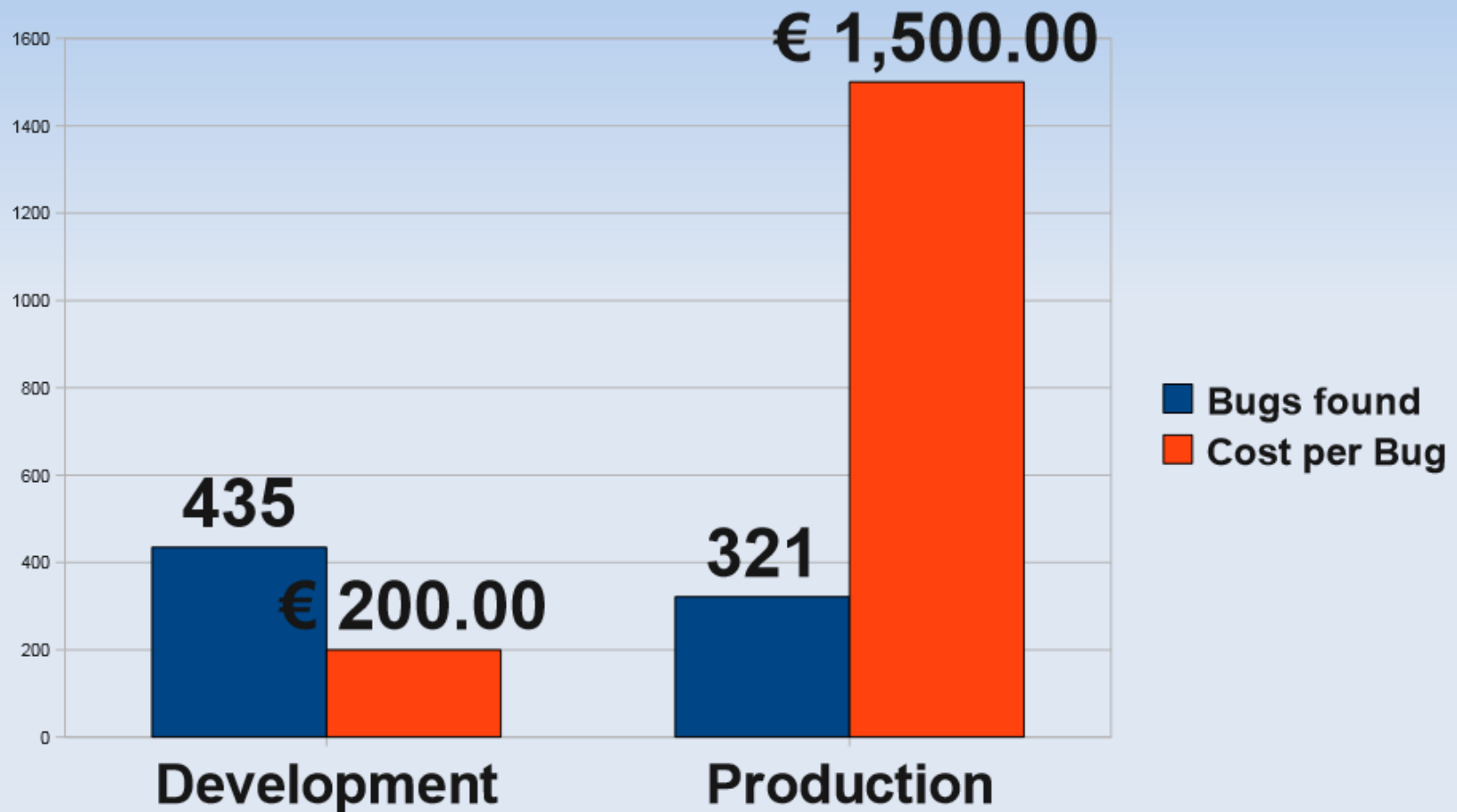
למה לבדוק?

(בכתום עלות לבאג אחד)

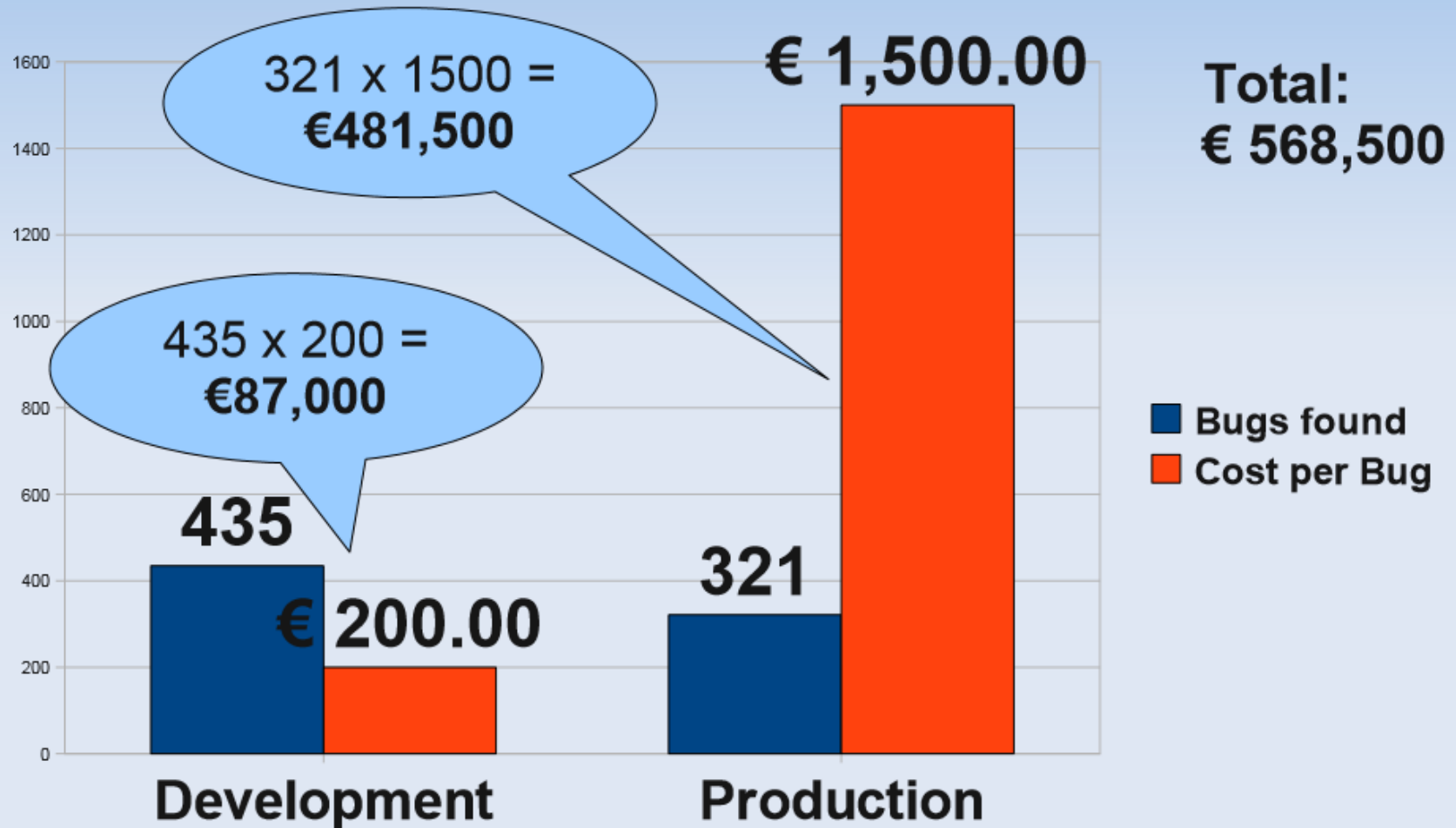
**Simplest case:
Development => Production**



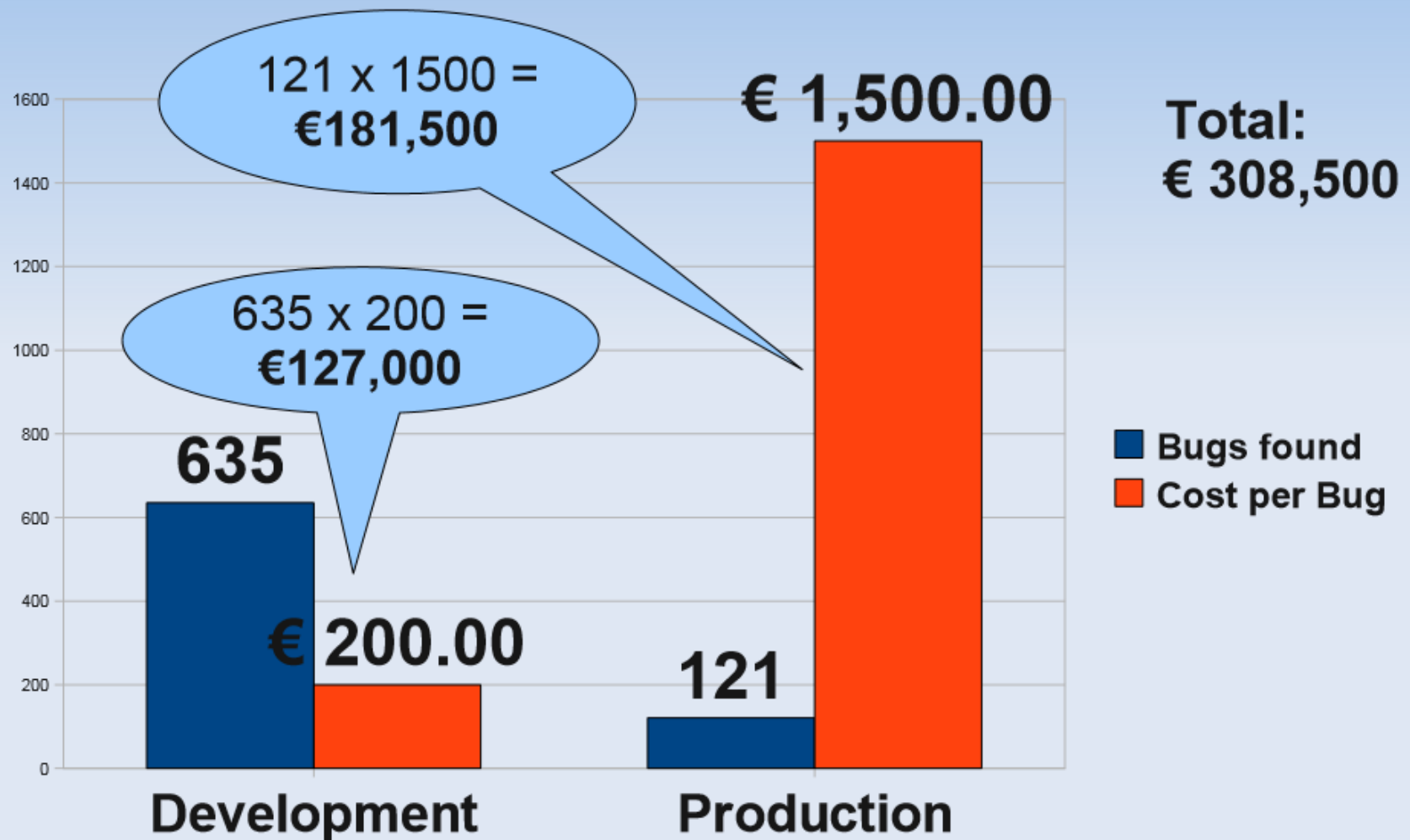
Real world (no testing)



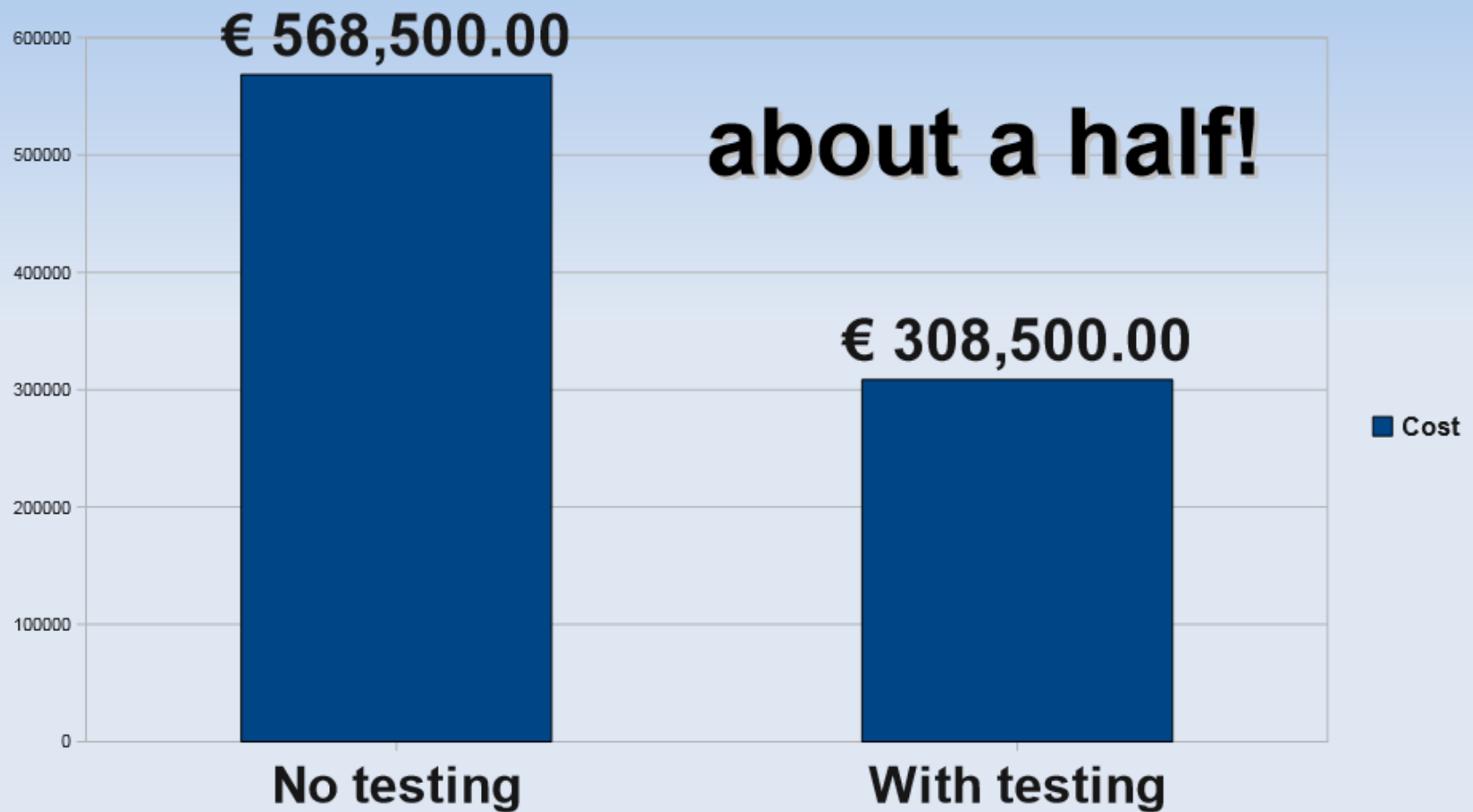
Real world (no testing)



Real world + testing



no testing vs + testing

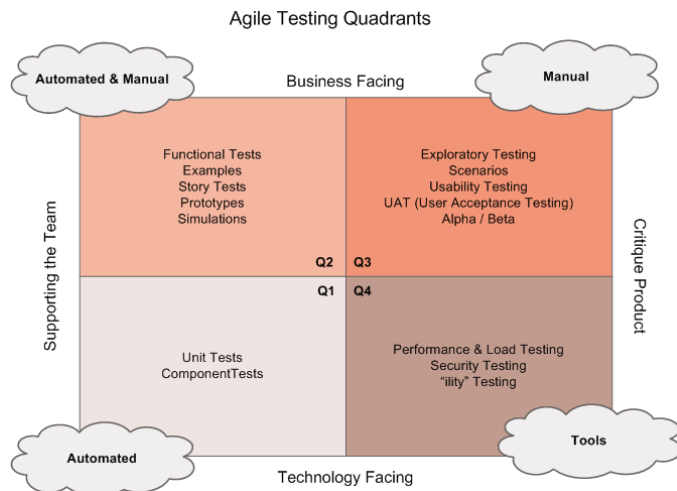


איך אתן בודקות את מוצר התוכנה שלך?

1. מריצה בעצמי ומדבגת \ `printf`
2. נותנת לחברה לבדוק (QA)
3. כותבת תכנית הדגמה שמריצה את הקוד שלי
4. לא בודקת \ אין לי באגים...

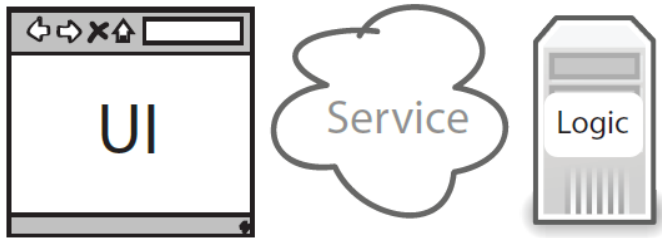
אילו בדיקות?

- 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...

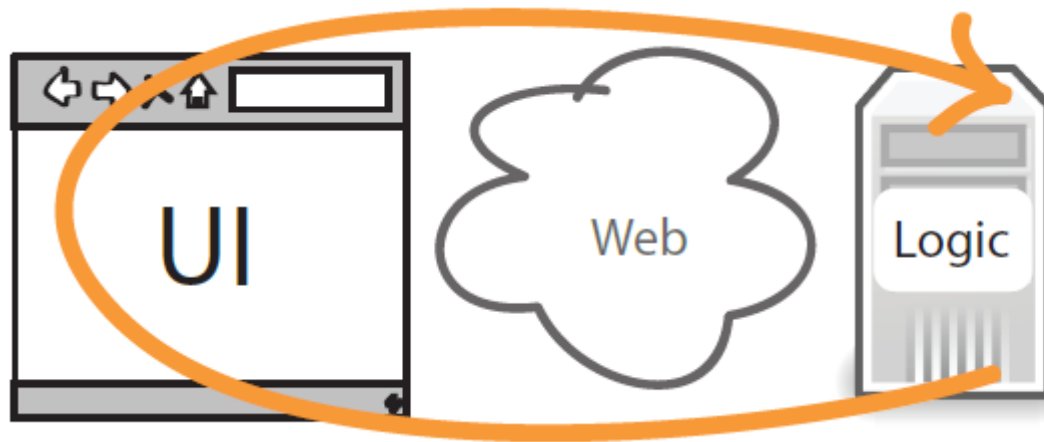


- [100 Types of Software Testing You Never Knew Existed](#)

ננסה להתמקד



- בדיקות מערכת
 - האם המערכת עובדת בשלמותה?
 - מנקודת מבט של המשתמש! ("קטמנדו")
 - **בפרויקט**: ניסוח בדיקה באמצעות תרחיש או סיפור
 - (קצה לקצה)\(ממשק) משתמש\קבלה\פונקציונליות)
- 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?



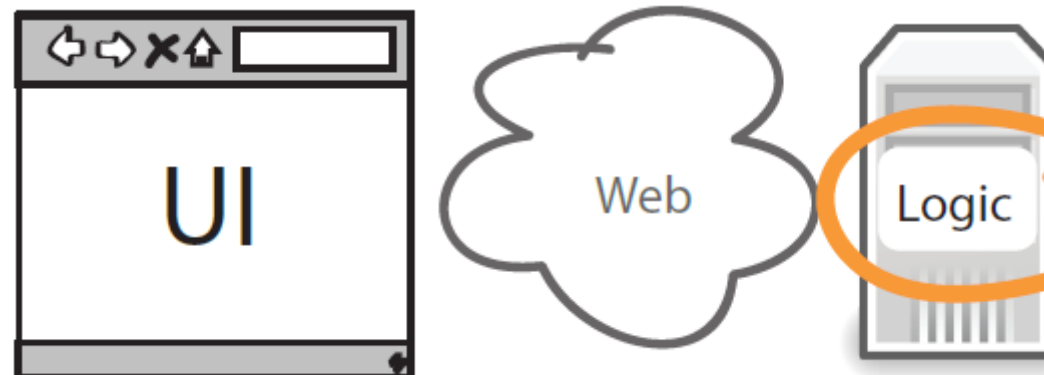
UI Tests

- ✓ Go end-to-end
- ✓ See what a user would see
- ✗ Expensive & slow



Integration Tests

- ✓ Web-services & APIs
- ✓ Connectivity
- ✗ Not most precise



Unit Tests

- ✓ Lightning fast
- ✓ Extremely versatile
- ✗ Miss integrations

הדגמת מבוא - FizzBuzz

- Problem [Definition](#) ([199/200](#) can't solve)
- Demo: [google sheets](#) appscript
- איזה סוג בדיקה זו?
- לקחים: הקדמת מפרט ובדיקות, אוטומציה, כיסוי קוד, Refactoring
- [FizzBuzz Test First Demo](#)
[FizzBuzzEnterpriseEdition](#) 😊

בדיקת יחידה

- הגדרה: בדיקת יחידה היא קוד שקורא לקוד אחר ובודק אח"כ נכונות של טענות מסוימות.
– "יחידה" היא רכיב "קטן" בד"כ מימוש של תרחיש מסוים (מצד המשתמש), מחלקה\פונקציה\מתודה (מצד המפתח)
– בד"כ נכתבת באמצעות framework (בהמשך)
- System Under Test (SUT) – הדבר שאותו אנחנו בודקים

A test is not a unit test if [Feathres]

- *It talks to the database*
- *It communicates across the network*
- *It touches the file system*
- *It can't run at the same time as any of your other unit tests*
- *You have to do special things to your environment (such as editing config files) to run it.*

האם כדאי להשקיע בזה?

- אולי מספיקות בדיקות אינטגרציה ומערכת?
- זהו קוד שגם מצריך תחזוקה!
- בעצם אולי כבר כתבתם עד היום כאלו דברים
- מה חסר?

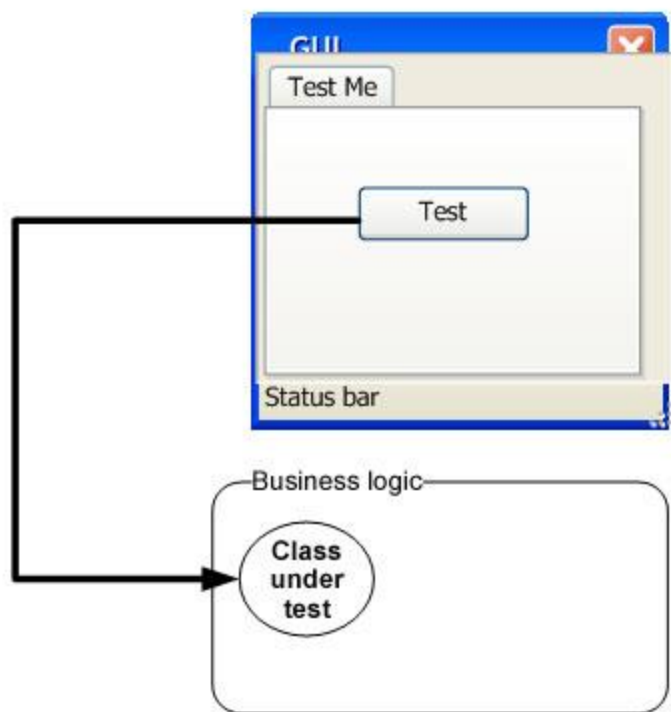
– אולי זו בדיקת אינטגרציה

– שיטה

– ביצוע חוזר

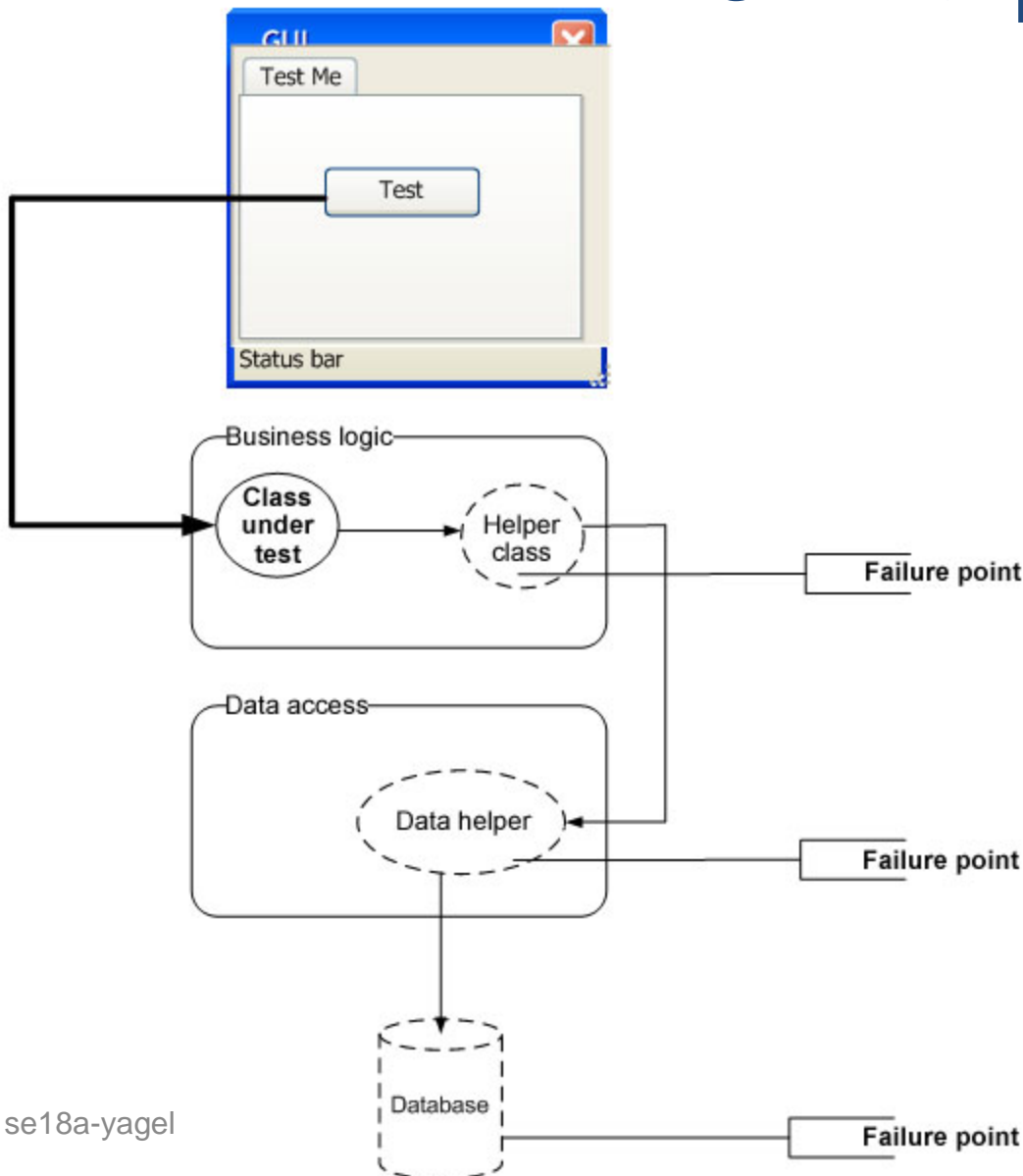
– לכל הקוד

תשתית - Framework ➤



בדיקת אינטגרציה

- בדיקת מספר רכיבים התלויים אחד בשני ביחד



בדיקות מערכת – ידני\אוטומטי

- בשנים האחרונות: Executable Spec. ,BDD ,ATDD
 - לפעמים משלבות בדיקות ממשק משתמש, כלים לדוגמא:
Capybara ,WatiR / N ,Selenium/Webdriver ,RobotFramework
- ```
[Test]
public void SearchForWatiNOnGoogle()
{
 using (var browser = new IE("http://www.google.com"))
 {
 browser.TextField(Find.By.Name("q")).TypeText("WatiN");
 browser.Button(Find.By.Name("btnG")).Click();

 Assert.IsTrue(browser.ContainsText("WatiN"));
 }
}
```

# מה מהבאים אינו יתרון של בדיקות יחידה על בדיקות אינטגרציה וקצה לקצה

1. ניתן להריץ בדיקות שביצעתי בעבר שוב ושוב  
(רגרסיה)

2. אפשר להריץ במהירות וכך לקבל משוב מהיר

3. קל לכתוב בדיקה בודדת

4. אוסף הבדיקות מהווה למעשה מהווה מפרט של  
המערכת

# מי בודק?

- בודקים או מפתחים?
- המטרה: מוצר בעל-ערך\איכותי

- יחס מפתח:בודק ( $\infty:1$ )

– מה משמעות גודל היחס?

Developer in testing –

Exploratory Testing etc. –

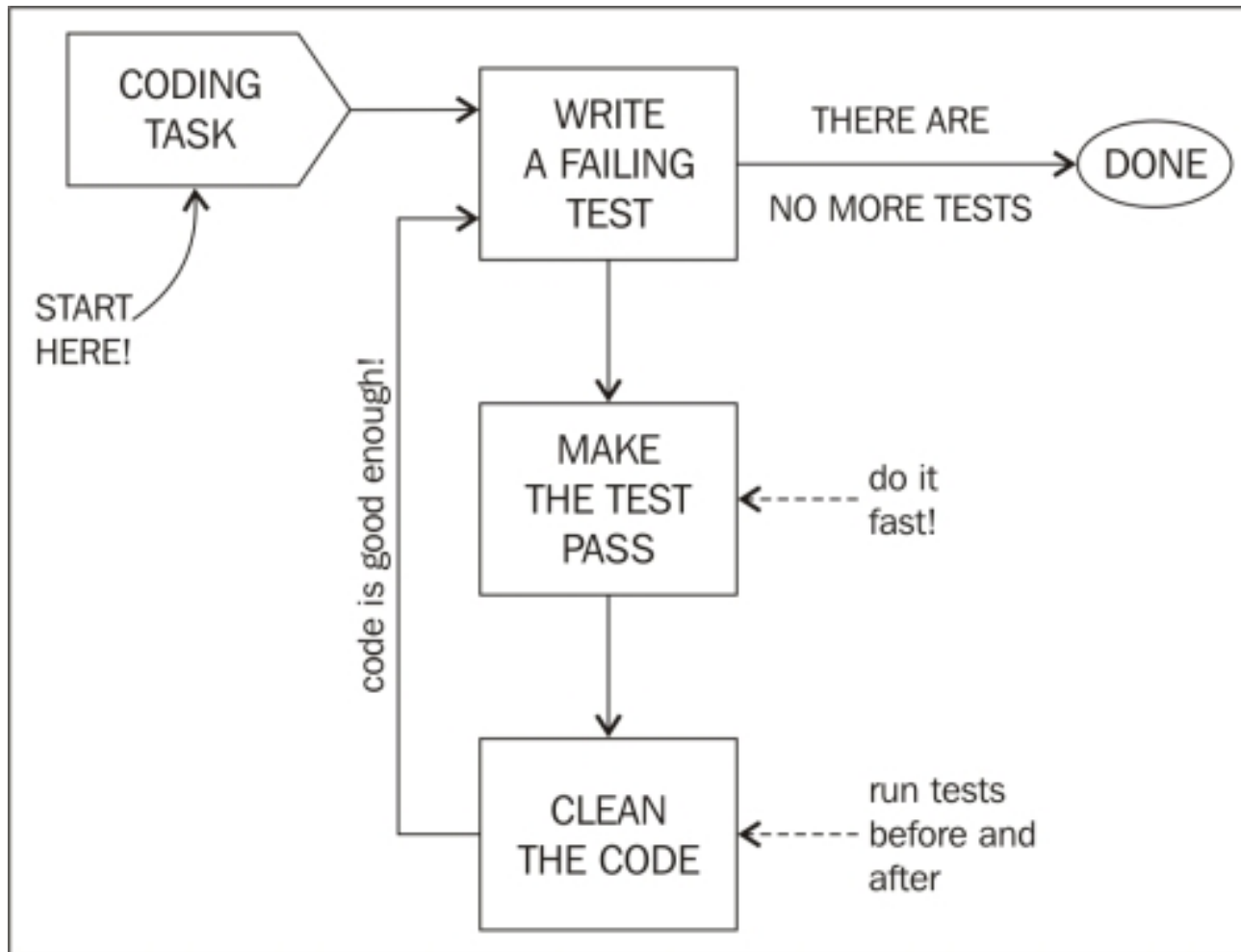
Why Facebook doesn't have or need testers –

# מתי כותבים?

- מסורתי: QA, Test **Last**  
– הנחה: כותבים קוד מושלם עבור דרישות לא משתנות
- אג'ייל: קודם!  
– **Test First**  
– **Test Driven Development**  
– מעקרונות XP, התקבל כנוהג כללי  
– משפחת xDD (Behavior, Feature, ...)



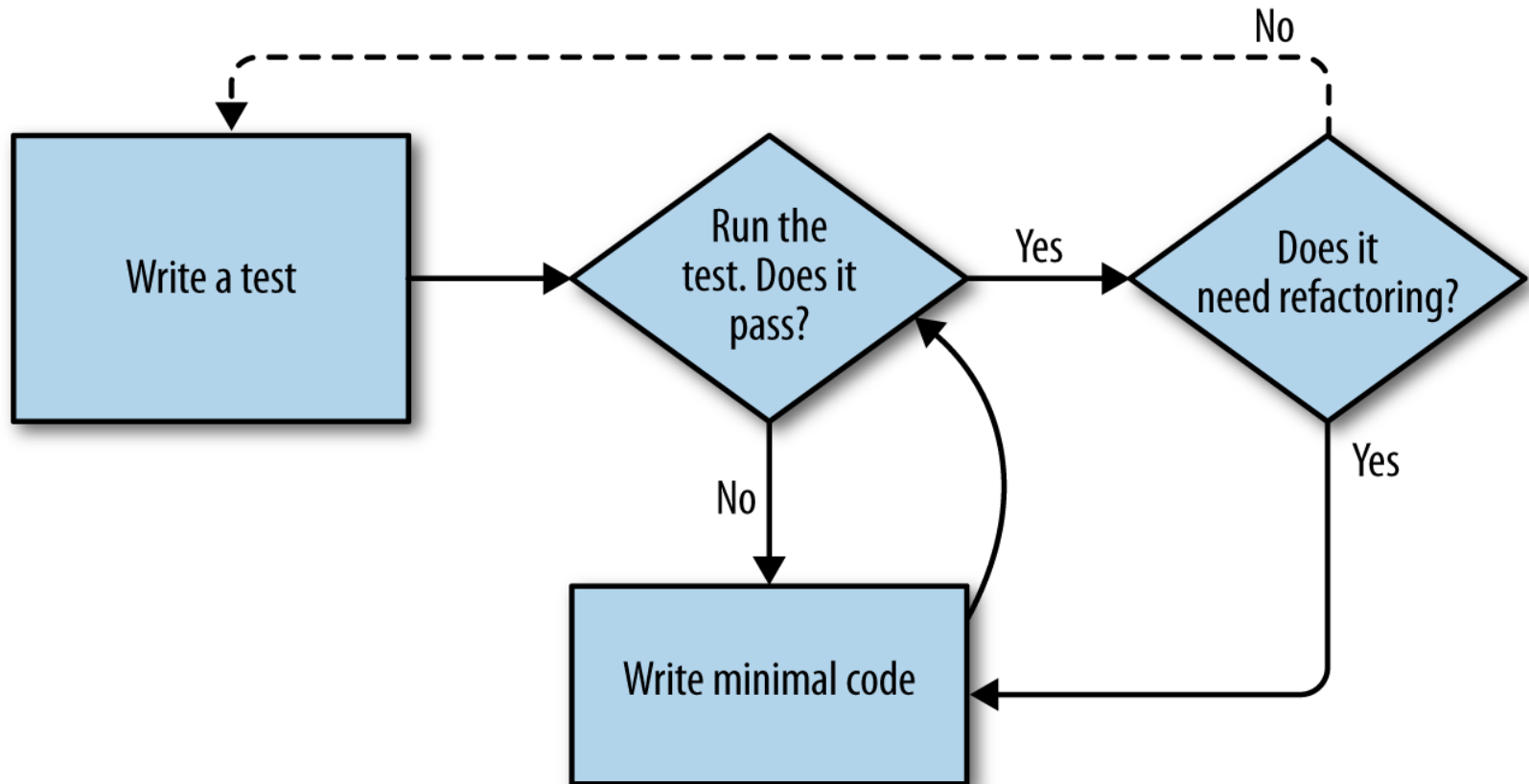
# Test First [[Amodeo](#)]



# Test First Rule Summary

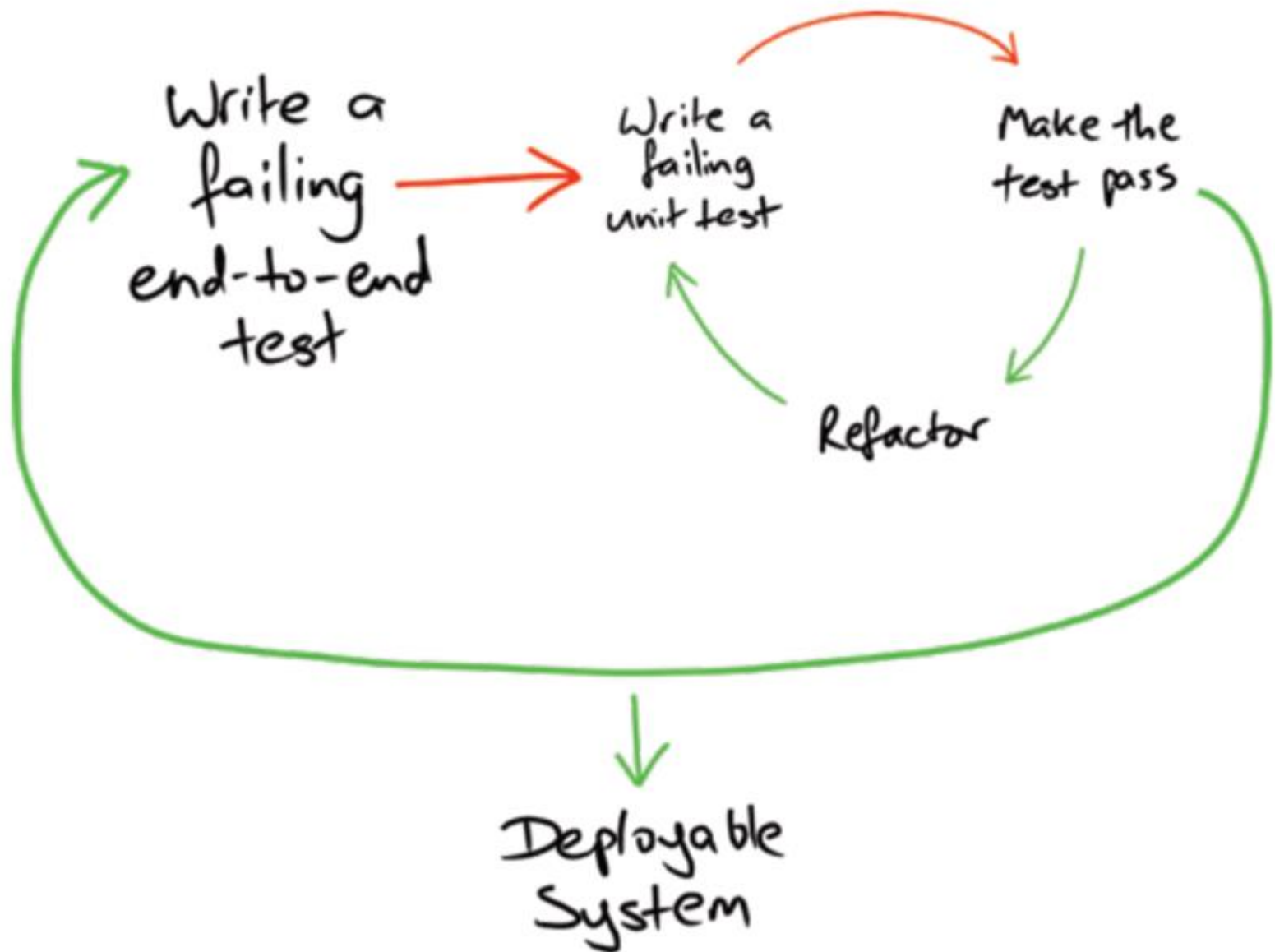
- Don't write any new tests if there is not a new coding task / a change in the product.
- A new test must always fail.
- A new test should be as simple as possible.
- Write only the minimum necessary code to fix a failing test, and don't bother with quality during this activity.
- You can only clean or redesign your code if all the tests pass. Try to do it in each cycle if possible.

# TDD Process

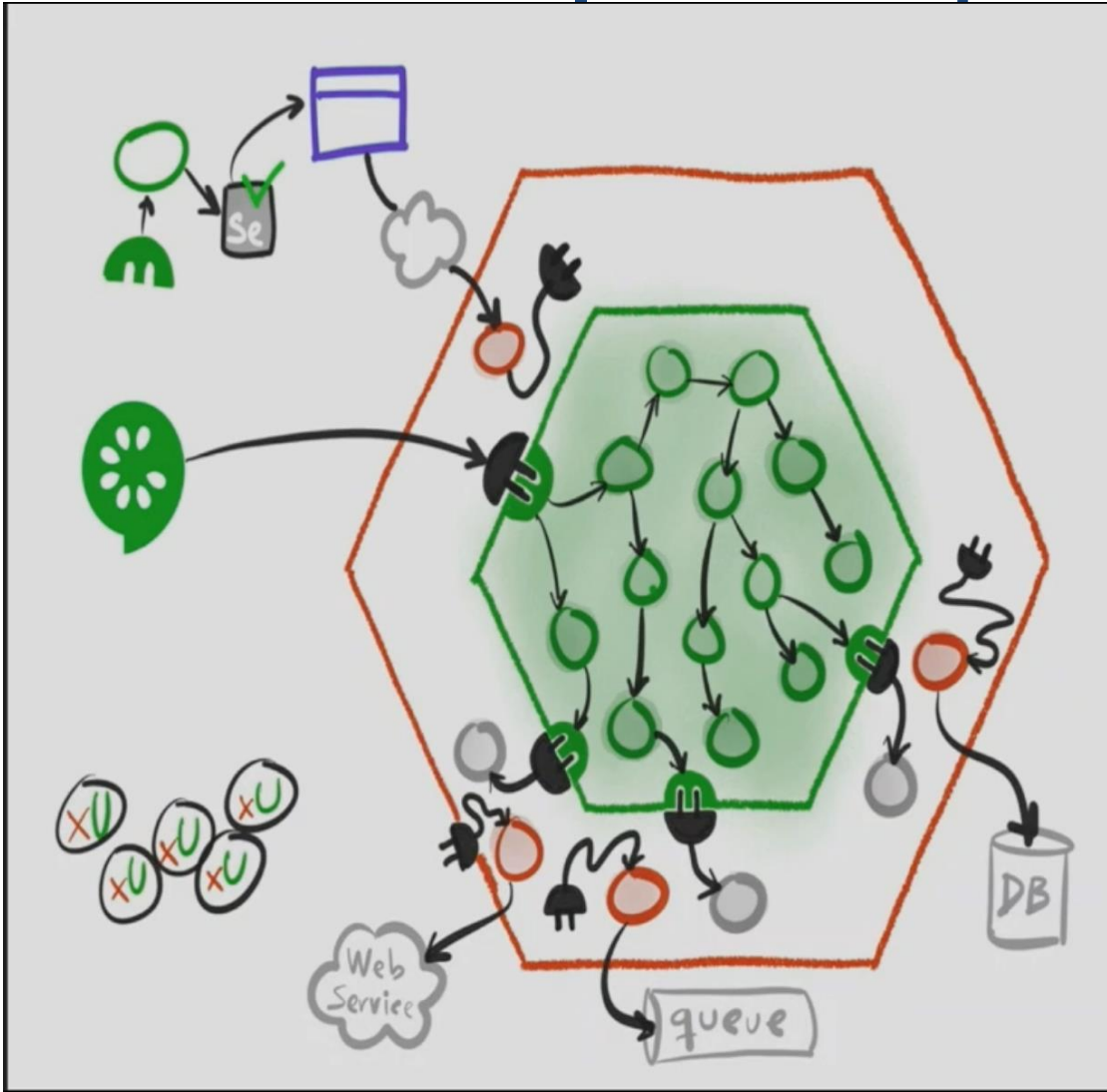


# Behavior Driven Development

- פיתוח של TDD
- דגש על בדיקת מאפיינים \ סיפורי משתמש
- בשילוב עם בדיקות יחידה



# בדיקות ותיכון



skills matter

CukeUp! 2016

## Keynote: Kind of Green

Aslak Hellesøy

14.04.2016

@skillsmatter  
skillsmatter.com



# Beck (XP): Simple Design

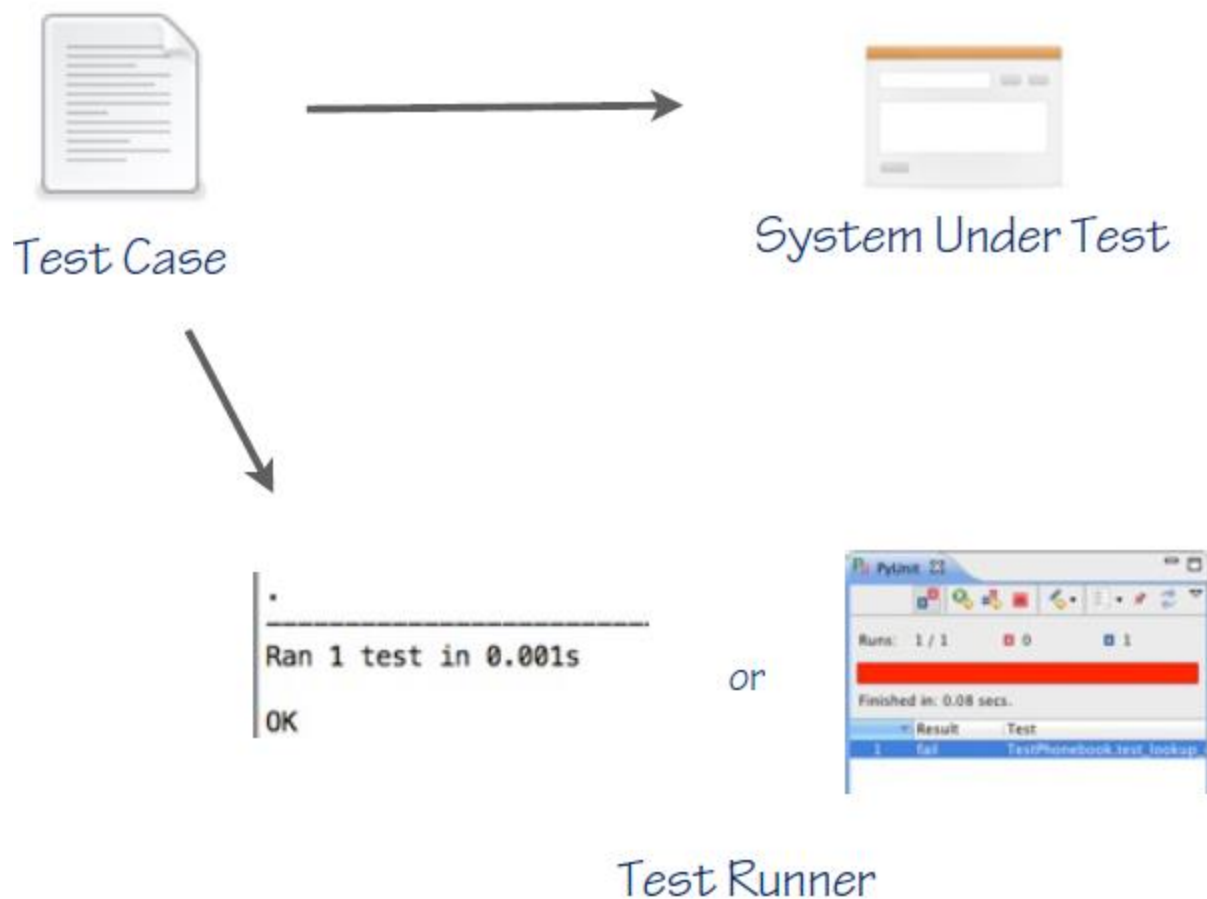
1. כל הבדיקות עוברות
2. ללא כפילויות (DRY)
3. ברור - מבטא את כוונת המתכנת
4. קטן - מינימום של מחלקות ומתודות

# כלים: xUnit Frameworks

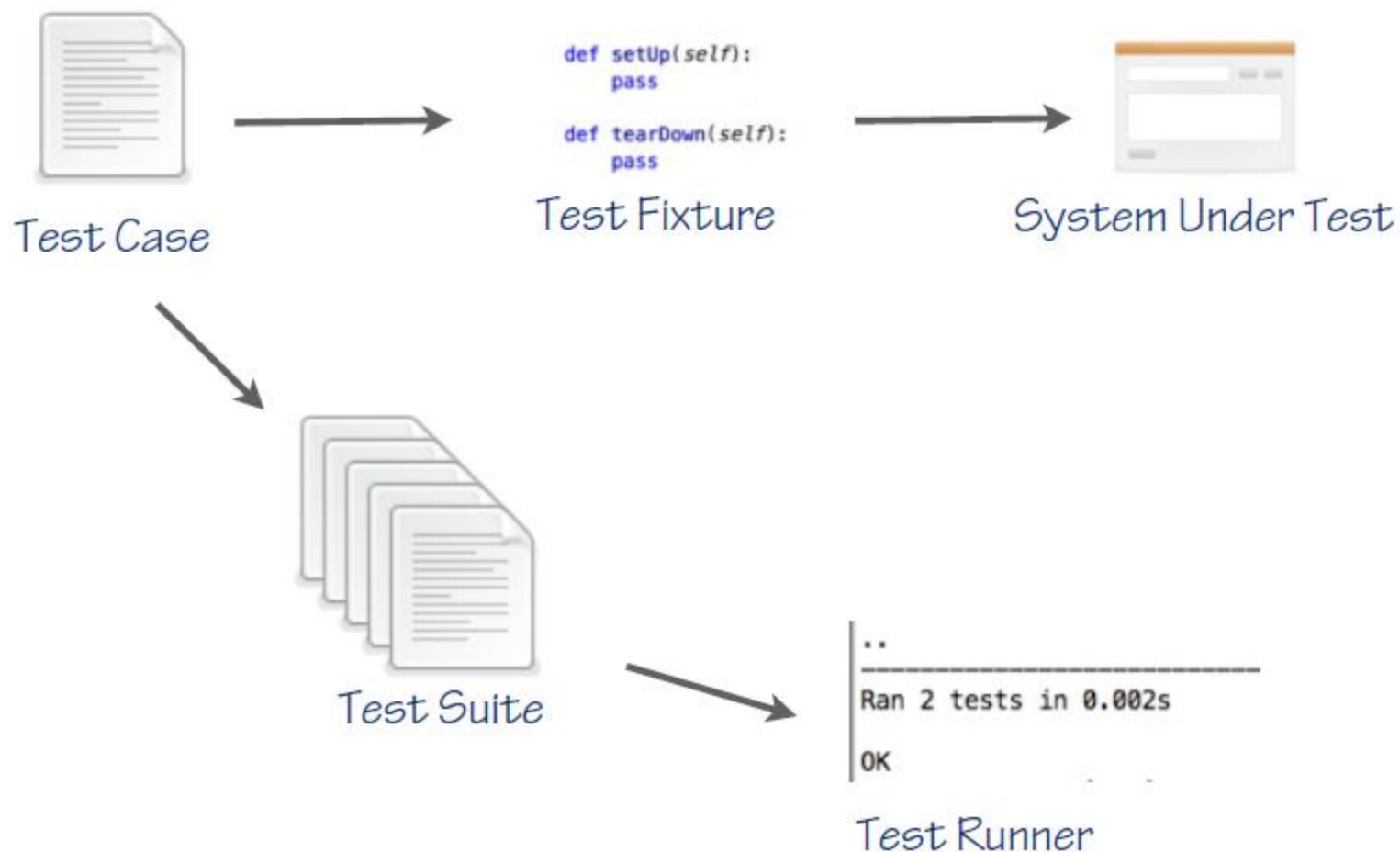
- '94, Kent Beck, SUnit– SmallTalk, [Rediscovering](#)
- [~'00](#), +E. Gamma, JUnit ("Test Infected")
- ייצוא לשפות רבות: CppUnit, PyUnit ועוד
  - <http://www.xprogramming.com/software>
  - [http://en.wikipedia.org/wiki/List\\_of\\_unit\\_testing\\_frameworks](http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks)
- ארכיטקטורה סטנדרטית בעיקר לבדיקות יחידה



# מונחים מקובלים

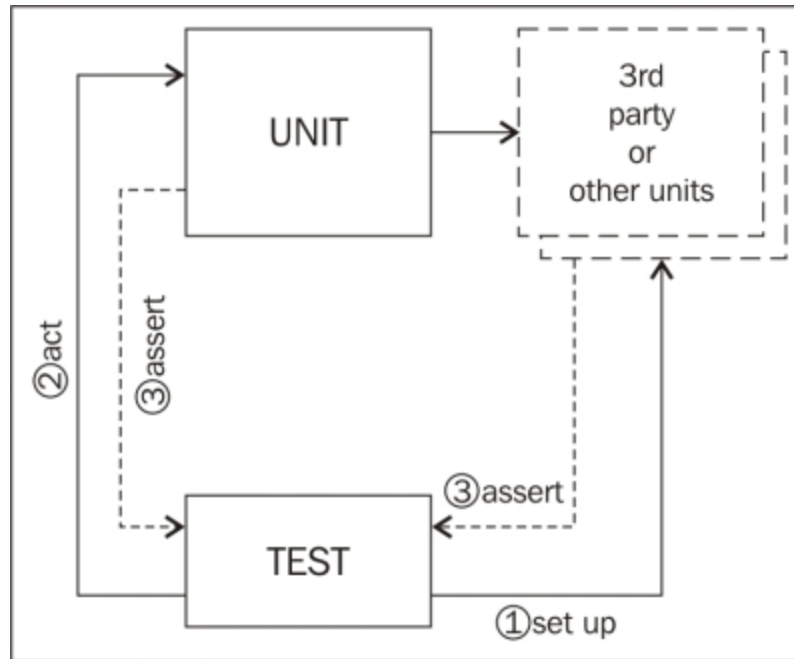


# מונחים מקובלים



# שלים בבדיקת יחידה

- Arrange (set up)
- Act
- Assert



# 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();
```

```
 int result = calc.add(2, 3);
```

```
 assertEquals(5, result);
```

```
 }
```

```
}
```

// SUT

```
public class Calc { public int add(int a, int b) { return a+b; } }
```

// Arrange

// Act

//Assert

# רכיבים עיקריים בקוד בדיקה (.Net NUnit)

```
//[TestFixture]
public class WhenUsingLogAnalyzer
{
 [Test]
 public void ValidFileName_ReturnsTrue()
 {
 //arrange
 LogAnalyzer analyzer = new LogAnalyzer();
 //act
 bool result = analyzer.IsValidLogFileName("whatever.slf");
 //assert
 Assert.IsTrue(result, "filename should be valid!");
 }
}
```

# 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()
```

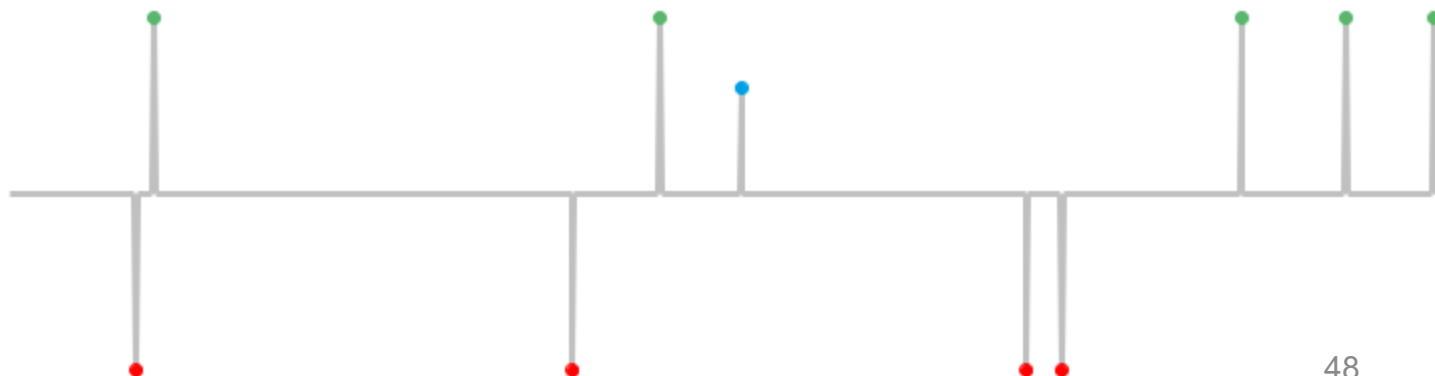
# רכיבים עיקריים בקוד בדיקה (js / node: mocha + chai)

```
var chai = require('chai');
var CartSummary = require('./../src/part1/cart-summary');

describe('CartSummary', function() {
 it('getSubtotal() should return 0 if no items are passed in', function() {
 // Arrange
 var cartSummary = new CartSummary([]);
 // Act
 var total = cartSummary.getSubtotal()
 // Assert
 expect(total).to.equal(0);
 });
});
```

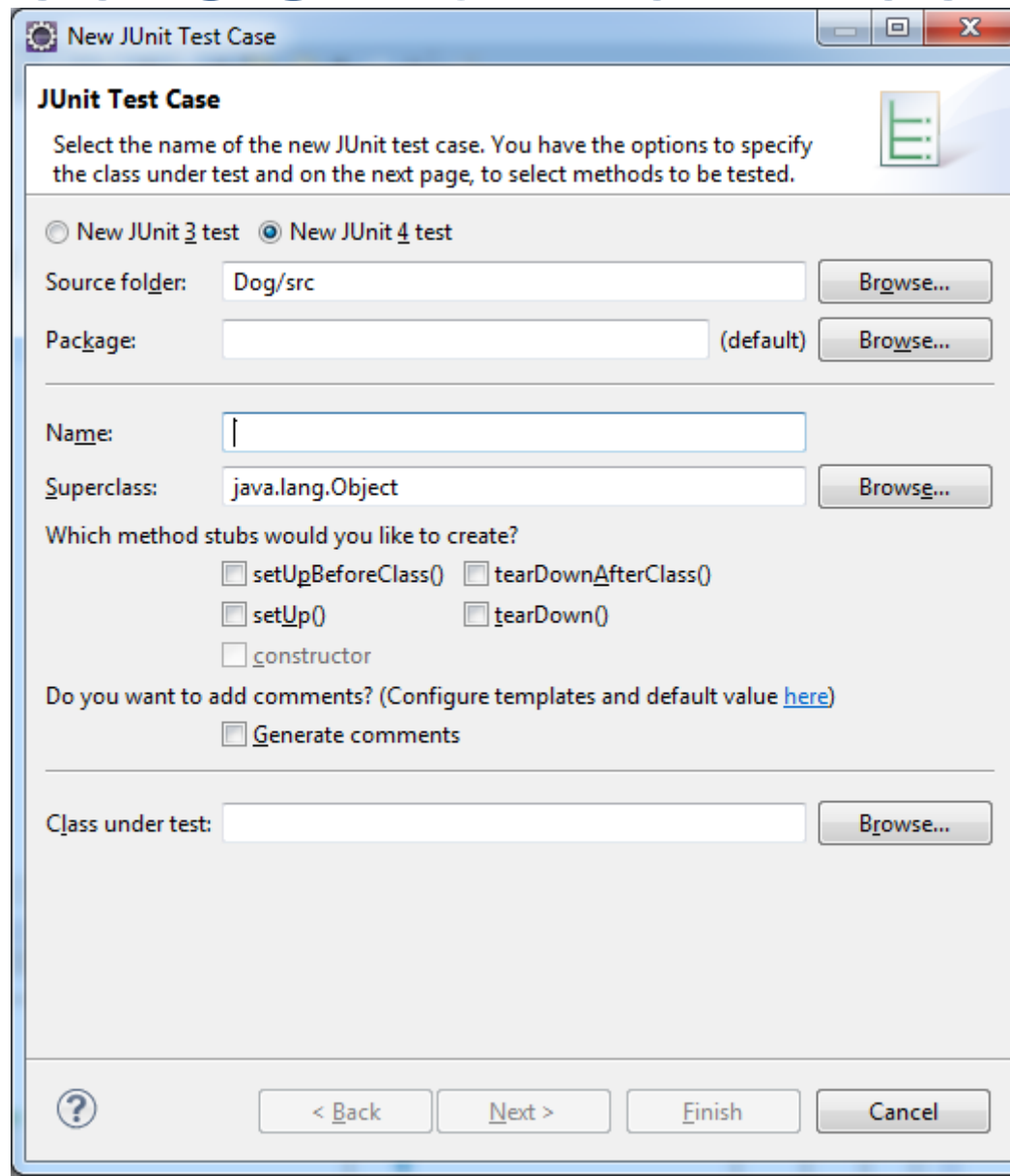
# כלים - Java

- Eclipse + JUnit (built in)
- Optional plug-ins:
  - Git/github: Egit, Mylyn
  - Gamification: [pulse](#), [TDGotchi](#) (Help->Install New Software)
  - Code Coverage: EclEmma





# Eclipse GUI for new TestCase



The screenshot shows the 'New JUnit Test Case' dialog box in Eclipse. The title bar reads 'New JUnit Test Case'. The main heading is 'JUnit Test Case'. Below it, a text box says: 'Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.' There is a small icon of a document with a green 'E' on the right. The dialog is divided into several sections. The first section has two radio buttons: 'New JUnit 3 test' (unselected) and 'New JUnit 4 test' (selected). The second section has a 'Source folder:' text box with 'Dog/src' and a 'Browse...' button. Below it is a 'Package:' text box with '(default)' and a 'Browse...' button. The third section has a 'Name:' text box with a cursor. The fourth section has a 'Superclass:' text box with 'java.lang.Object' and a 'Browse...' button. The fifth section is titled 'Which method stubs would you like to create?' and contains five checkboxes: 'setUpBeforeClass()' (unchecked), 'tearDownAfterClass()' (unchecked), 'setUp()' (unchecked), 'tearDown()' (unchecked), and 'constructor' (unchecked). The sixth section is titled 'Do you want to add comments? (Configure templates and default value [here](#))' and contains a 'Generate comments' checkbox (unchecked). The seventh section has a 'Class under test:' text box and a 'Browse...' button. At the bottom, there is a row of buttons: a help icon (?), '< Back', 'Next >', 'Finish', and 'Cancel'.

**New JUnit Test Case**

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder: Dog/src

Package:

Name:

Superclass: java.lang.Object

Which method stubs would you like to create?

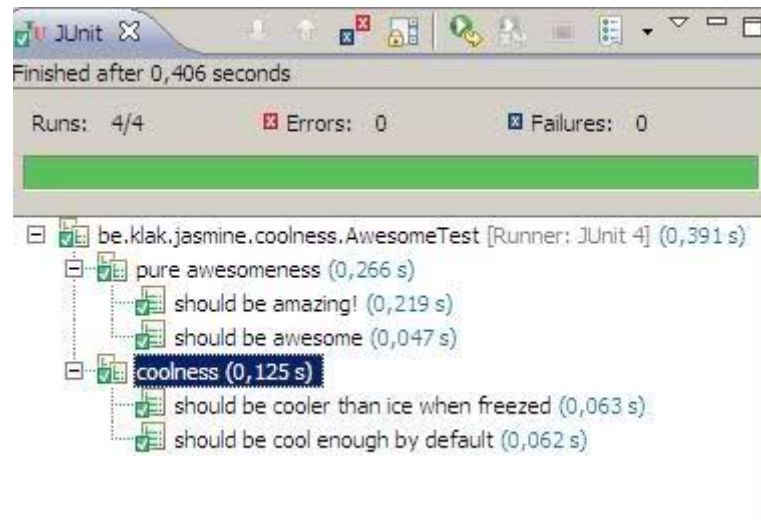
☐ setUpBeforeClass() ☐ tearDownAfterClass()  
☐ setUp() ☐ tearDown()  
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Class under test:

# Eclipse Runner



# JS / Node Tools

- Test framework / runner – [Mocha](#)
- Assertion library – [Chai](#)
- Many others

# הדגמה

- Tang, Unit Testing and TDD in Node.js – [Part 1](#), [Part 2](#) (future), [repo](#).
- Tools:
  - node + npm
  - npm install mocha -g (or local:)
  - npm install chai --save-dev

# משימה אישית 4 – בדיקות יחידה + TDD

- TDD: ה"בדיקות" מובילות את הפיתוח
- שכפול המאגר (fork / invitation + clone)
- אחרי כל צעד (red-green-refactor) יש לבצע git commit עם הערה שמתחילה בסוג הצעד (לדוג': RED: test missing name)
- אופציה: בזוגות מתחלפים (אחת בודקת אחת מממשת [pairhero](https://pairhero.com)), כולל ב-commits ((add origin)
- דחיפה חזרה ל-github

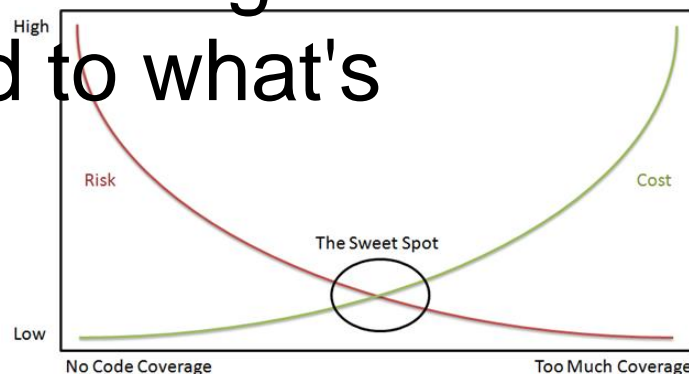
# Other: TDD Coding Katas

- String Calculator Kata  
<http://www.osherove.com/tdd-kata-1/> (up to stage 5)  
<http://www.21apps.com/agile/tdd-kata-by-example-video/>
- Many Others:
  - <http://www.codingkata.net>
  - <http://kata-log.rocks/tdd>
  - <https://github.com/garora/TDD-Katas>
  - Advanced: [GildedRose Kata](#) ([screencast](#))



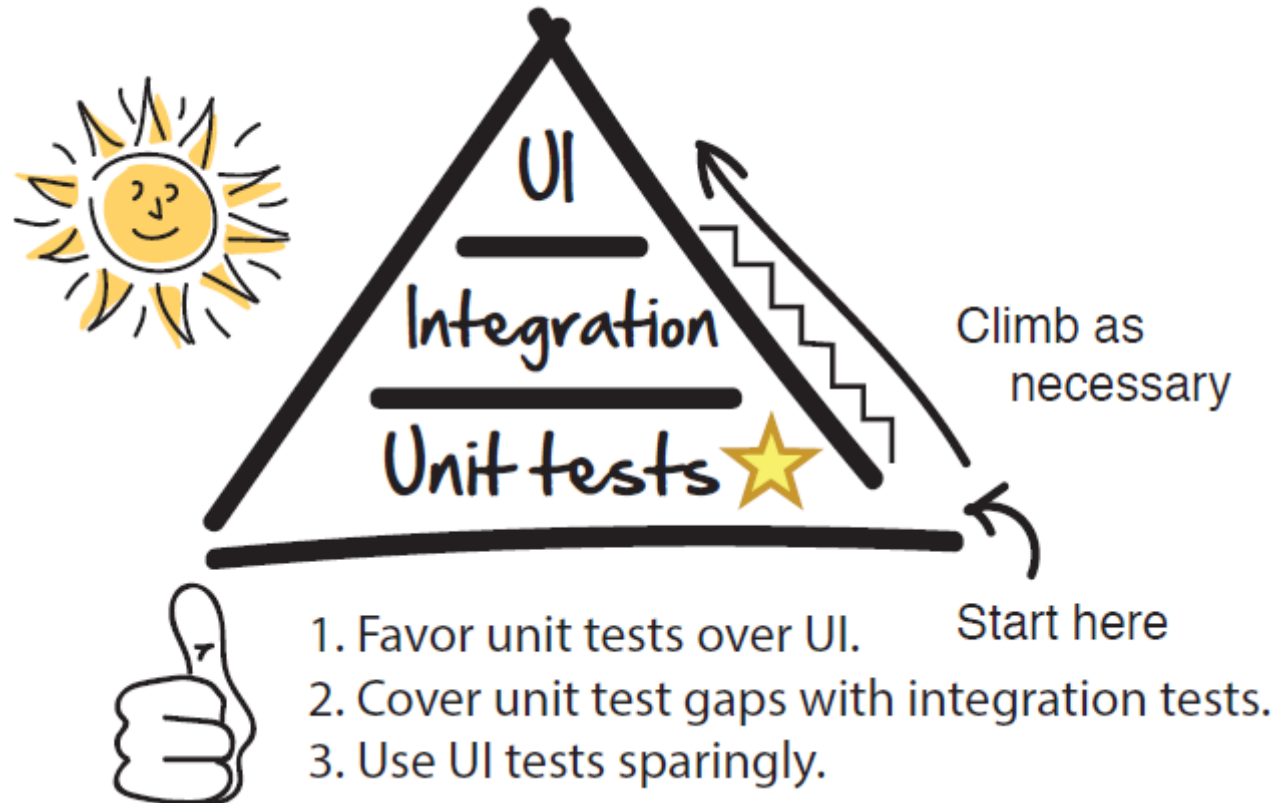
# כמה לבדוק?

- האם צריך תמיד כיסוי של 100% של בדיקות יחידה, 100% בדיקות אינטגרציה ו-100% בדיקות קצה?
- יחס קוד:בדיקות, למשל 40:60!
- Seth Godin: “Measurement is fabulous. Unless you're busy measuring what's easy to measure as opposed to what's important”



# The Forgotten Layer of the Test Automation Pyramid (also)

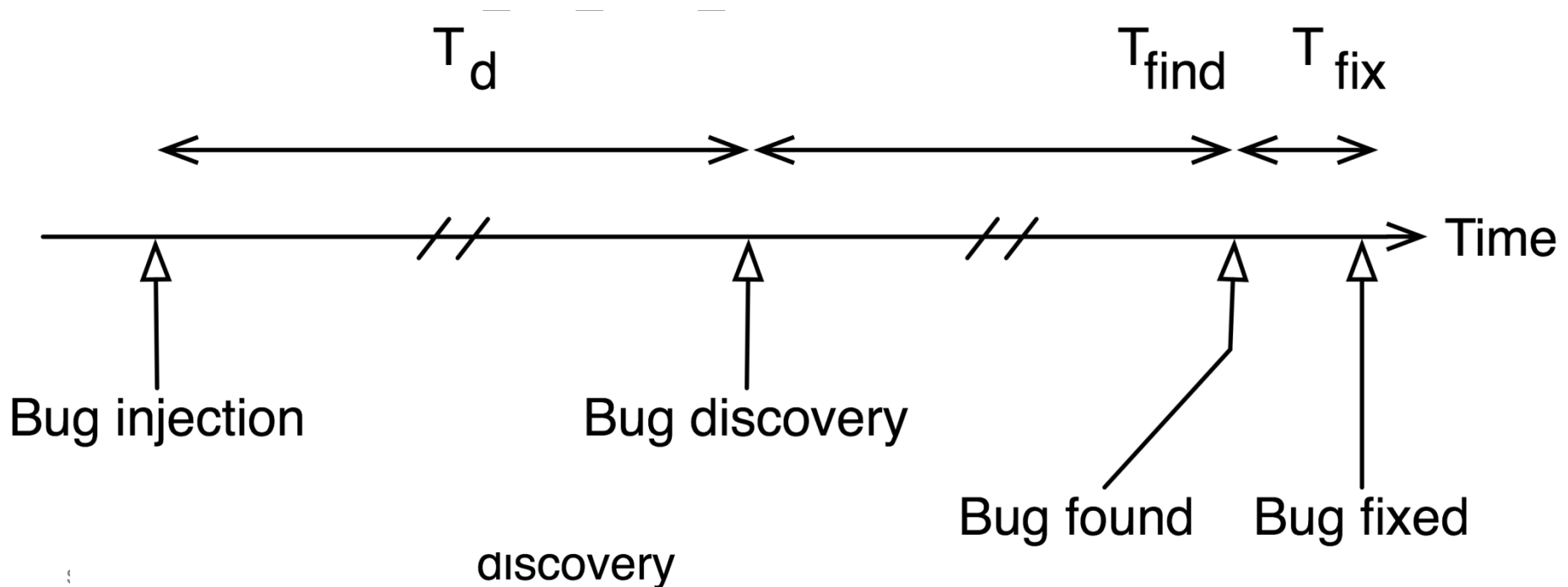
Google:  
small,  
medium and  
large





# האם זה משתלם?

- [Physics of Test Driven Development](#) (min. feedback)
- [How test-driven development works](#) (queuing)



# ROI for Selected Practices

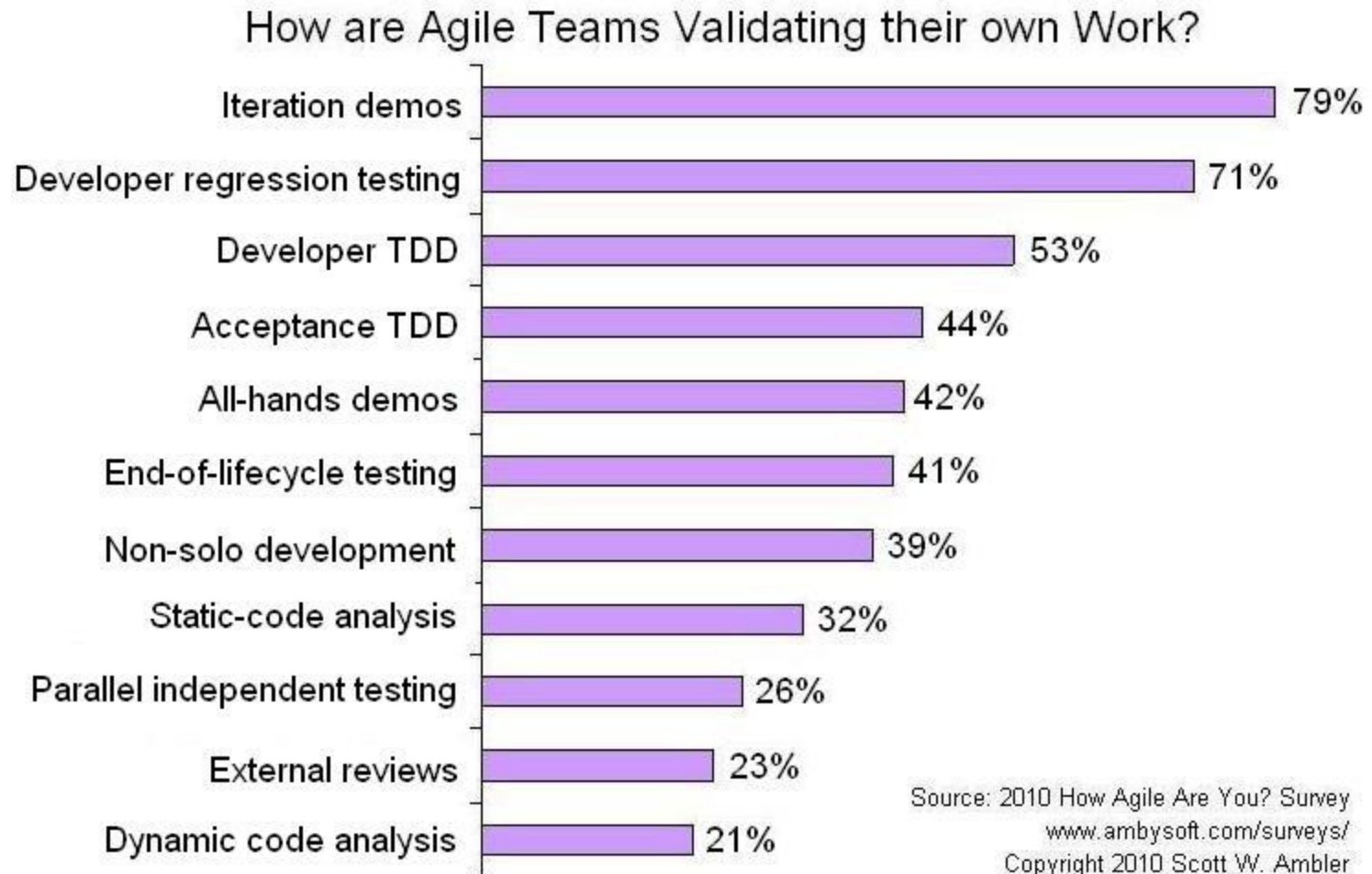
| Practice                    | 12-month ROI | 36-month ROI |
|-----------------------------|--------------|--------------|
| Test Driven Development     | -            | 1000%+       |
| PSP/TSP                     | -            | 800%         |
| Formal Inspections          | 250%         | 600%+        |
| Productivity Measurement    | 150%         | 600%         |
| Process Assessments         | 150%         | 600%         |
| Management Training         | 115%         | 550%         |
| Scrum                       | -            | 500%         |
| Process Improvement Program | -            | 500%         |
| Technical Staff Training    | 90%          | 500%         |

Sources: Rico, et al 2009; DACS 2007; McConnell 2004; Jones, 1994.

# האם כדאי?

- “The results of the case studies indicate that the pre-release defect density of the four products decreased between 40% and 90% relative to similar projects that did not use the TDD practice.”
  - *"Realizing quality improvement through test driven development: results and experiences of four industrial teams (2008)"*  
[http://research.microsoft.com/en-us/groups/ese/nagappan\\_tdd.pdf](http://research.microsoft.com/en-us/groups/ese/nagappan_tdd.pdf) (video)
- More: <http://biblio.gdinwiddie.com/biblio/StudiesOfTestDrivenDevelopment>  
<http://jamesshore.com/Blog/AoA-Correction-Test-Driven-Development.html>  
<http://langrsoft.com/jeff/2011/02/is-tdd-faster-than-tad/>

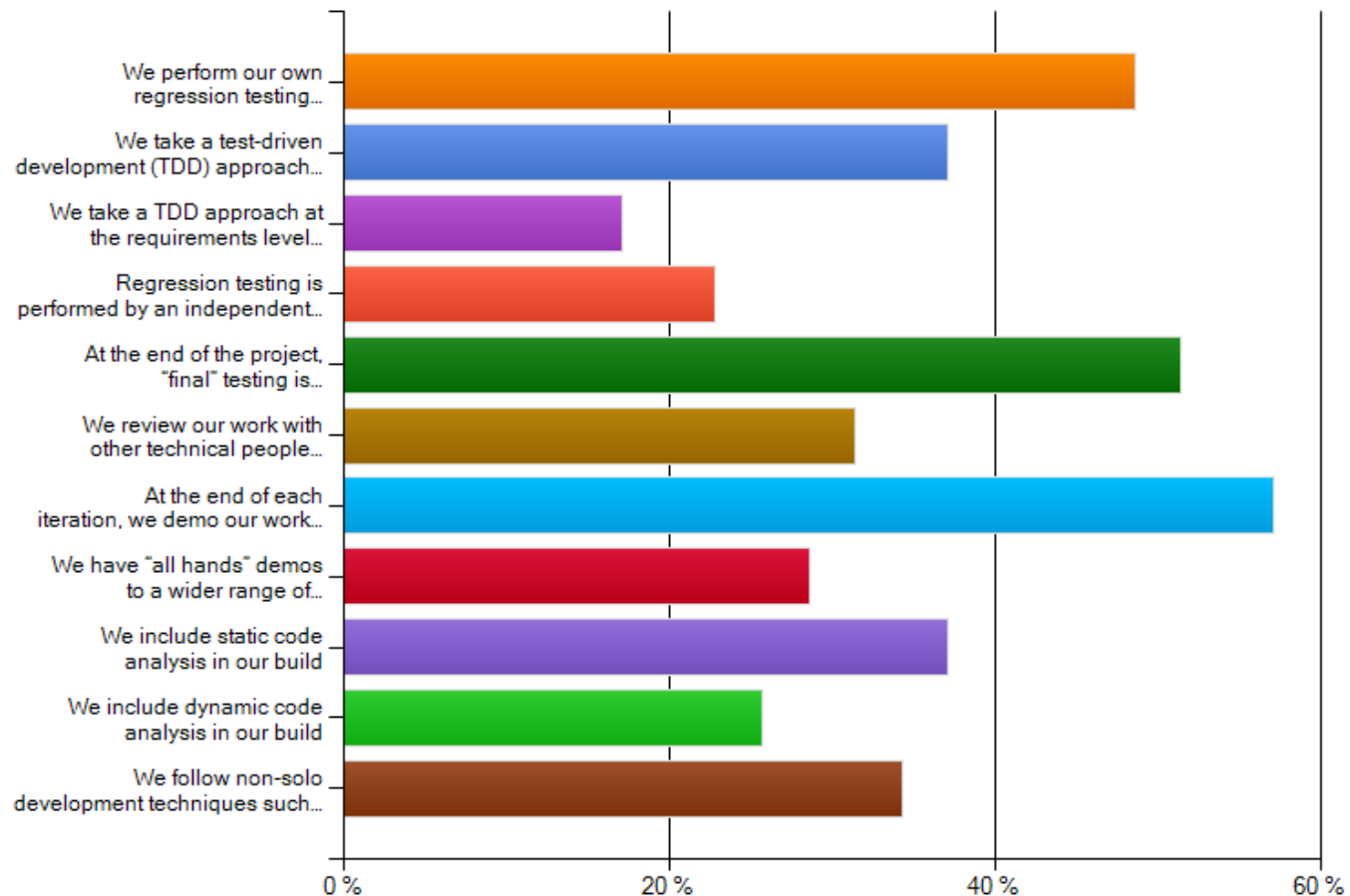
# How Agile Are You? 2010 Survey



Source: 2010 How Agile Are You? Survey  
[www.ambysoft.com/surveys/](http://www.ambysoft.com/surveys/)  
Copyright 2010 Scott W. Ambler

# How Agile Are You? 2013 Survey Results, S. Ambler

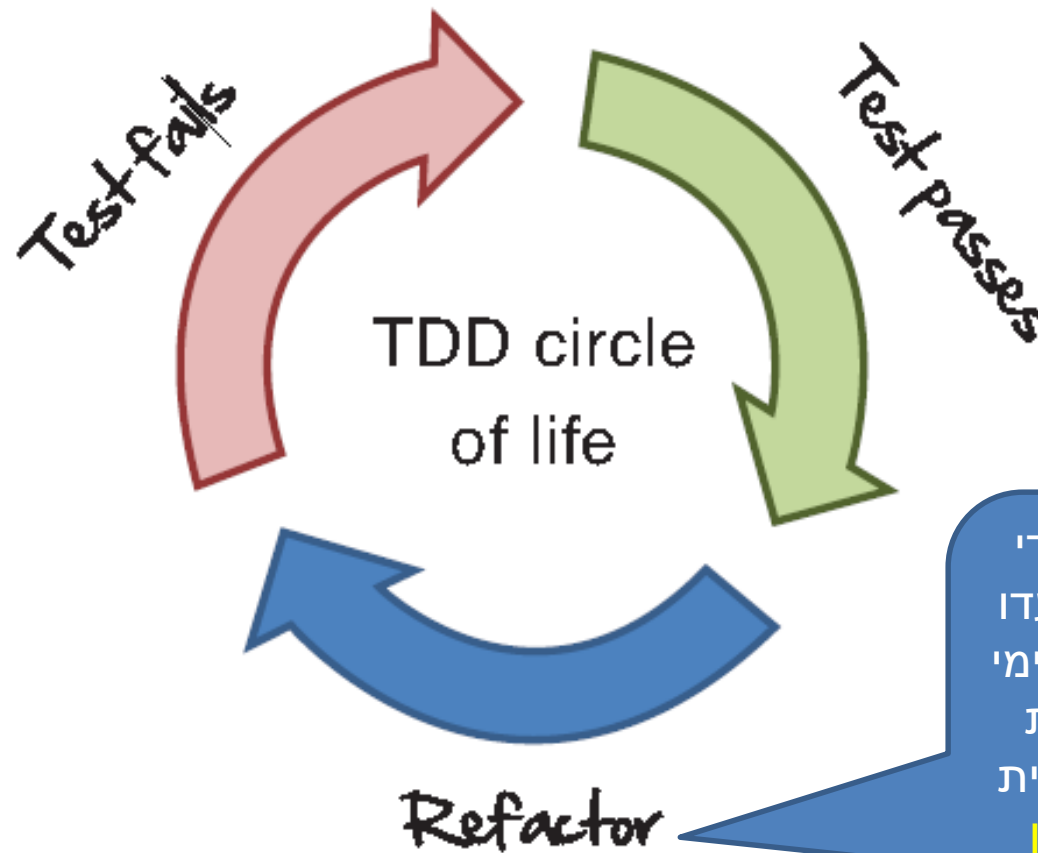
**What strategies does your team follow to validate their work? Please select all that apply (if any).**



# מצד שני

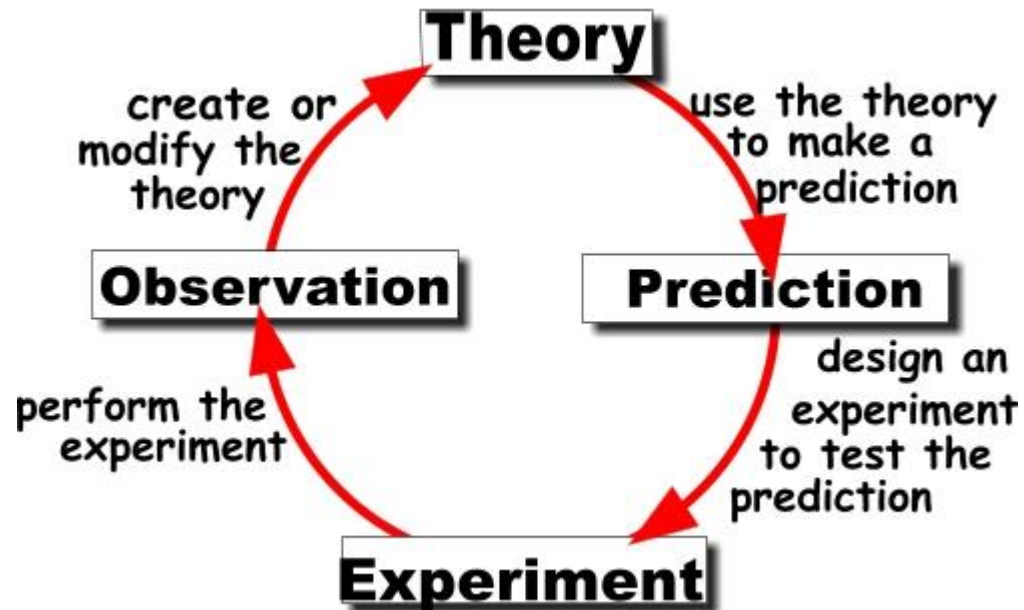
- “Does Test-Driven Development Really Improve Software Design Quality?” [[2008](#)]:
  - Smaller classes but not better at coupling and cohesion
- DHH (rails 2014) : [TDD is dead. Long live testing](#)
- [Coplien](#), Waste
- Ayende: “But I think that even a test has got to justify its existence, and in many cases, I see people writing tests that have no real meaning. They duplicate the logic in a single class or method.”
  - <http://ayende.com/blog/4217/even-tests-has-got-to-justify-themselves> ([refs](#))

# TDD = TFD + Refactoring



שיפור קוד קיים, על ידי שימוש בטכניקות שנועדו לשפר את המבנה הפנימי של הקוד מבלי לשנות את ההתנהגות החיצונית שלו (ויקיפדיה), **תיכון** מתמשך

# The Scientific Method [[Agile'03](#)]





# ?Refactoring כמה

- Refactoring vs YAGNI
- 4 rules of simple design?
- More later

## Two Refactoring Types\*

◆ Floss Refactorings—frequent, small changes, intermingled with other programming (daily health)



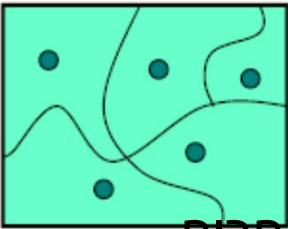
◆ Root canal refactorings — infrequent, protracted refactoring, during which programmers do nothing else (major repair)



# מושגים נוספים

- Regression vs. Retesting
- White-Box vs. Black-Box
- Test Selection / Coverage
- Alpha & Beta Testing
- Static & Dynamic Code Analysis
- Test Automation, Continuous Integration
  - [CI for HW4 repo skeleton](#)
- Exploratory / Usability / Perf. / Sec. / ...
- Test Management / Planning / Reporting / Metrics

# בהרצאת המשך \ נושאים מתקדמים



- בדיקות יחידה 2.0 למשל...
  - מאפיינים מתקדמים של xUnit: אתחולים, חריגות,
  - מחלקות שקילות, פרמטרים, כיסוי, תלות, אינטראקציה עם רכיבים אחרים, התנהגות מול מצב
  - בדיקות לניידים \ ענן \ רשת \ UI וכו'
  - כיצד למצוא את הבדיקה הבאה – טריאנגולציה, Transformation Priority Premise
  - בדיקות לקוד קיים.
- משימה אישית: TDD
- בפרויקט
  - בכל סבב: ניסוח בדיקת קבלת לתרחיש עיקרי
  - משימת סבב: בדיקות לרכיב מרכזי
- קריאה מומלצת להרצאת המשך:  
Using Mock Objects

# בפעם הבאה

- המשך בדיקות
  - TDD
- עבודה על תרגיל 4

# לסיכום



- בדיקות/בדיקות יחידה, פיתוח מונחה בדיקות, xUnit

“The project was a miserable failure because we let the tests we wrote do more harm than good” - [osherove](#)

- בדיקות ותיכון מתמשך
- Red-Green-Refactor
- מצריך לימוד מתמשך – אז למה עכשיו?
- האם בשימוש? שי ילין Wix: "כל מהנדס תוכנה שמגיע ל Wix לומד TDD"

<https://youtu.be/kN7PgIOtSC0?t=1104>