



**ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT**

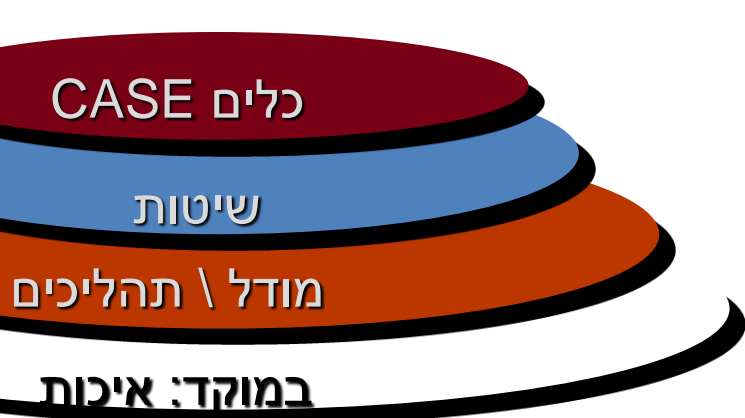
CODESMACK

הנדסת תוכנה

7. בדיקות - I

בדיקות יחידה, פיתוח מונחה מפרטים

[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, unittest)
- הדגמה
- מעבדת בדיקות יחידה
 - משימה אישית 4: TDD ו-בדיקות יחידה
 - השלמת git
- פרויקט: המשך\סיום סבב 1-MVP הצגה, רטרוספקטיבה ותכנון לסבב 2
 - פגישות וסקרים, ציון לפי איכות והתקדמות, סקר עמיתים?
 - פרויקט: משימת סבב 2 לדוגמא: בדיקות יחידה



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

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?

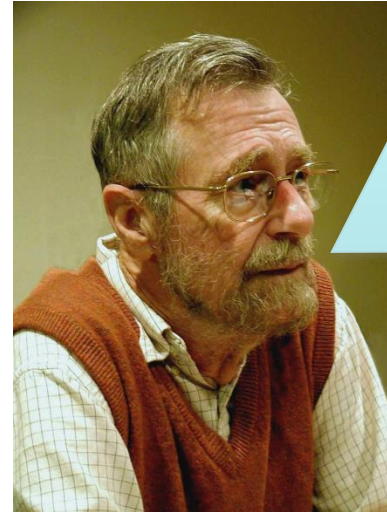
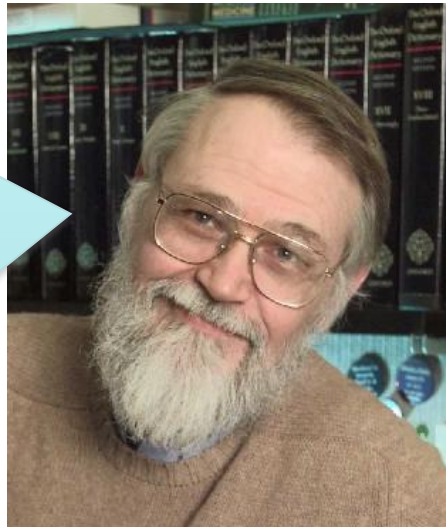
– מה להפסיק? מה לשנות?



- 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.

se15b-yagel



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

Source: [saas](#)⁷

בדיקות תוכנה



- למה לבדוק?
- איך לבדוק?
- אילו בדיקות?
- מי בודק? מתי?
- כמה לבדוק?

Boeing 777 wing break test –

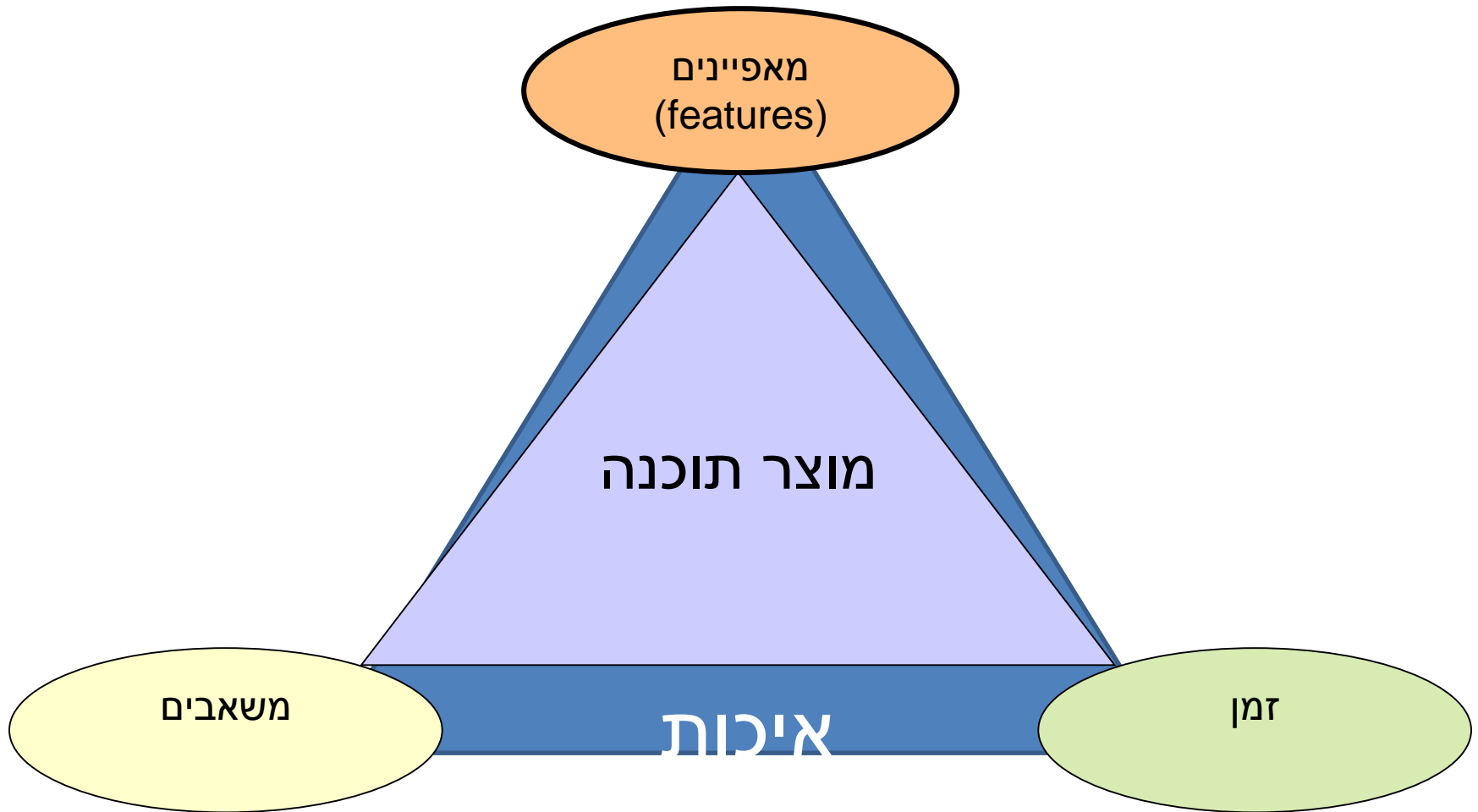
- האם זה באמת כדאי?
- למה עכשיו?

למה לבדוק?

- נכונות ותקפות
 - איכות בכלל
 - רגרסיה
- לאפשר שינויים
- הבוס\המרצה דורש..

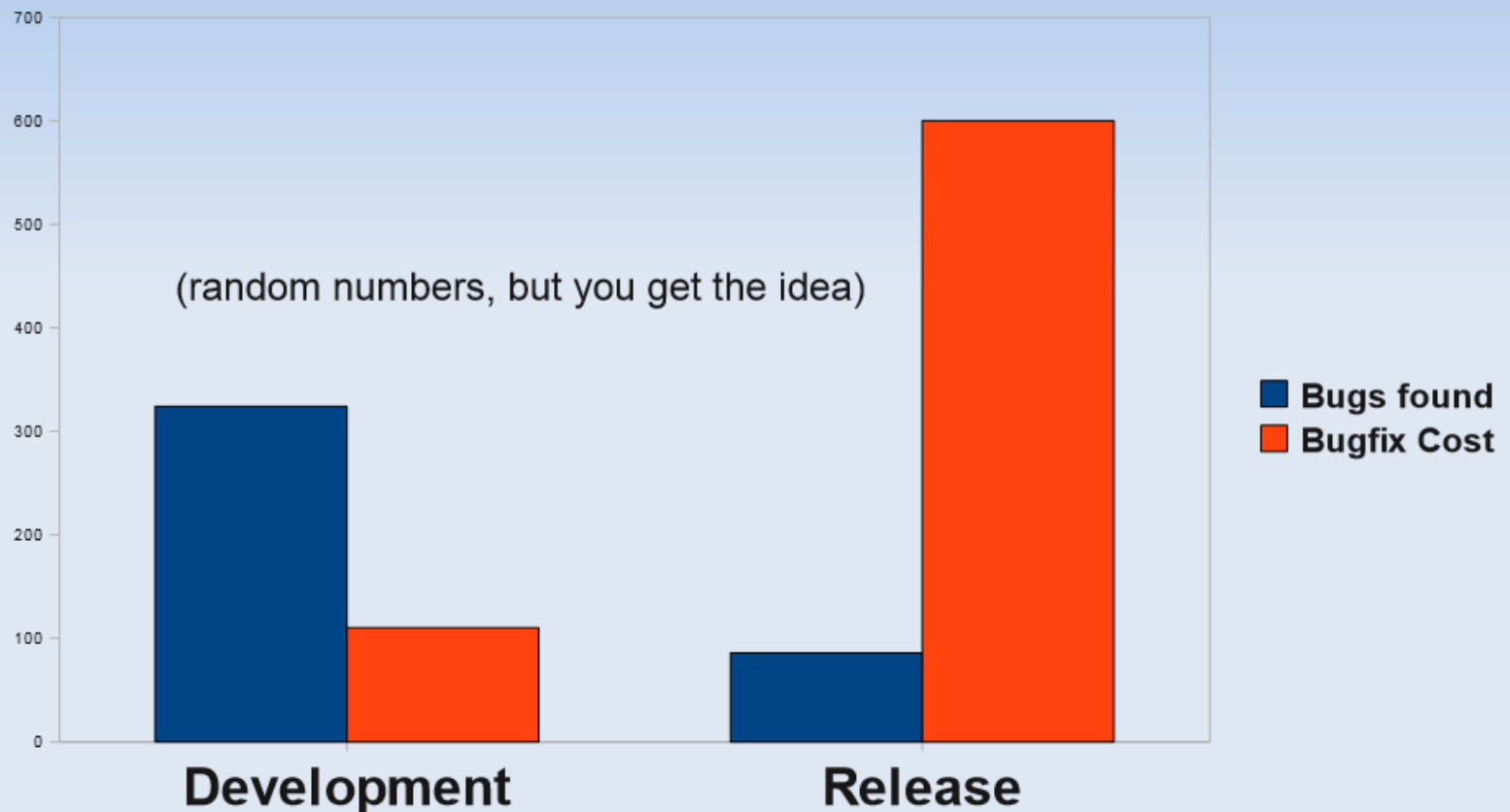


תזכורת: פרויקט תוכנה:

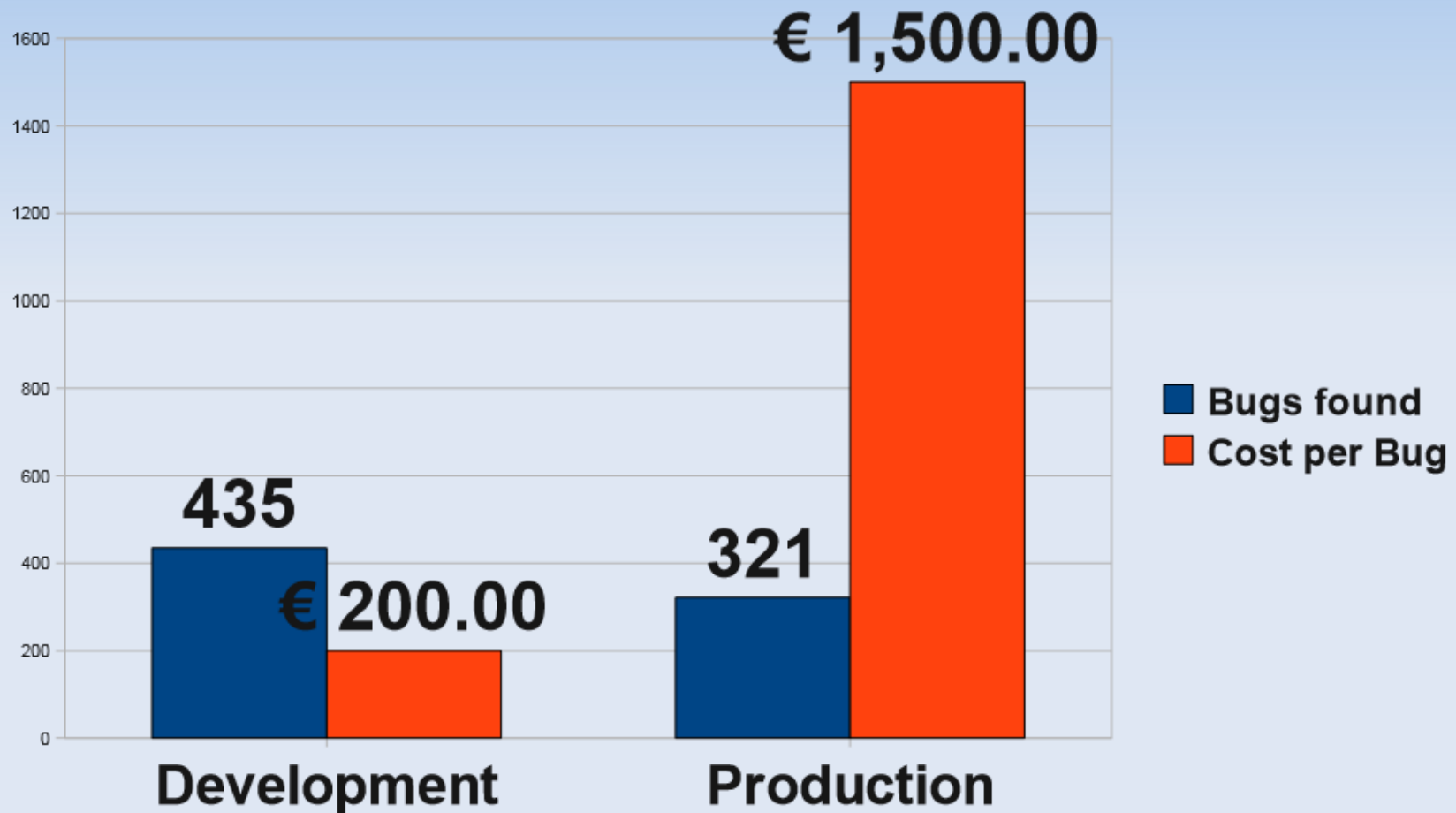


למה לבדוק? (באדום עלות לבאג אחד)

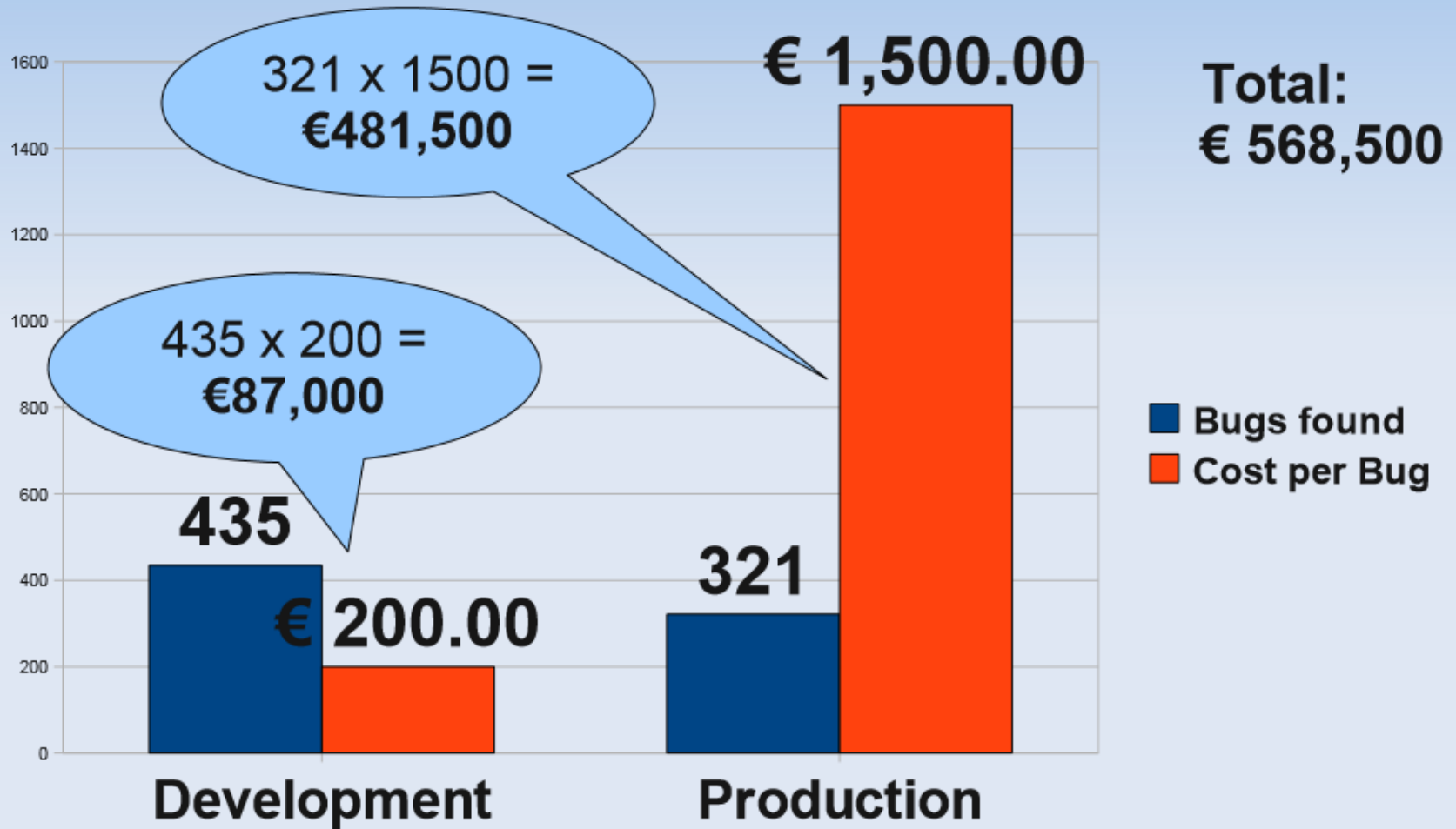
**Simplest case:
Development => Production**



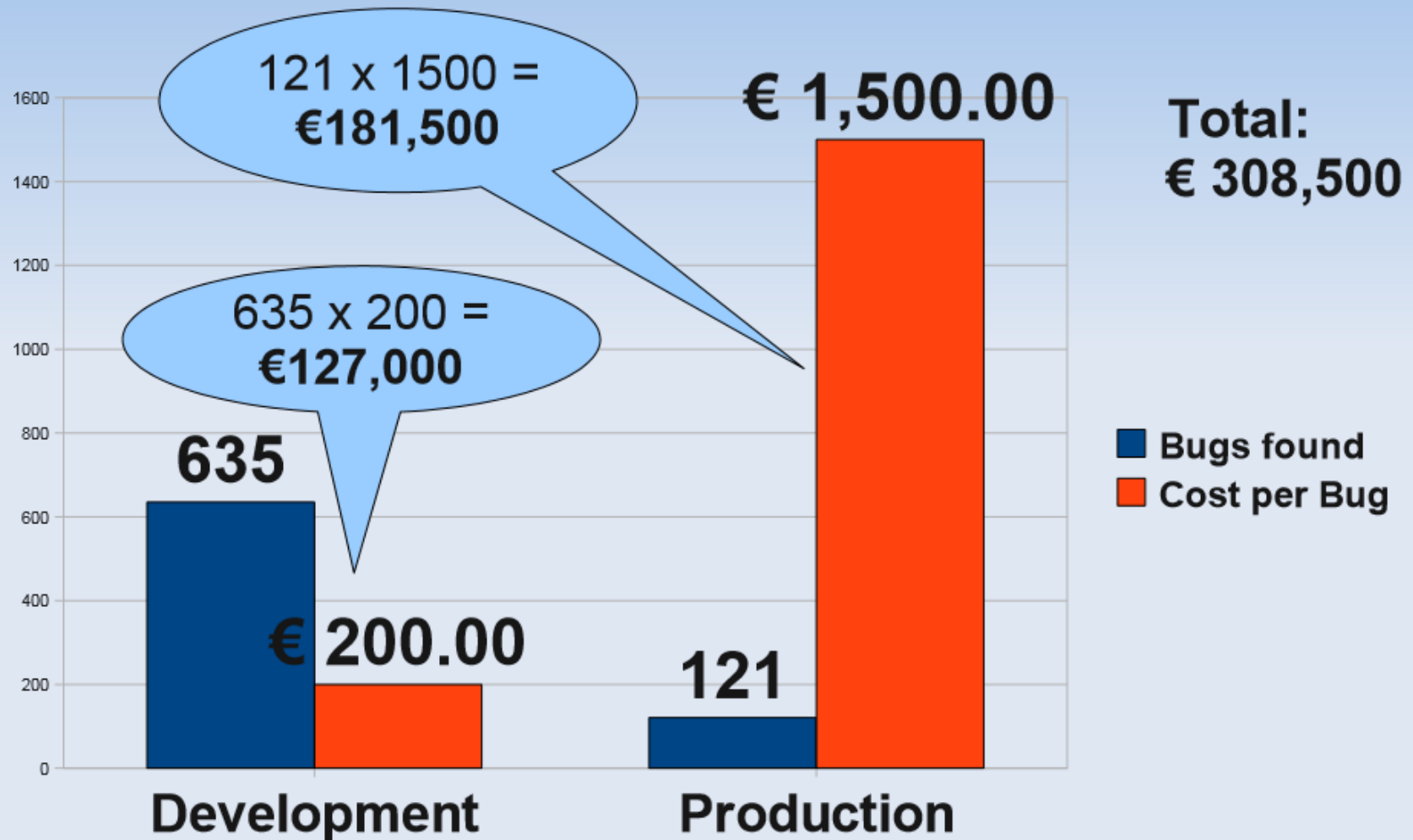
Real world (no testing)



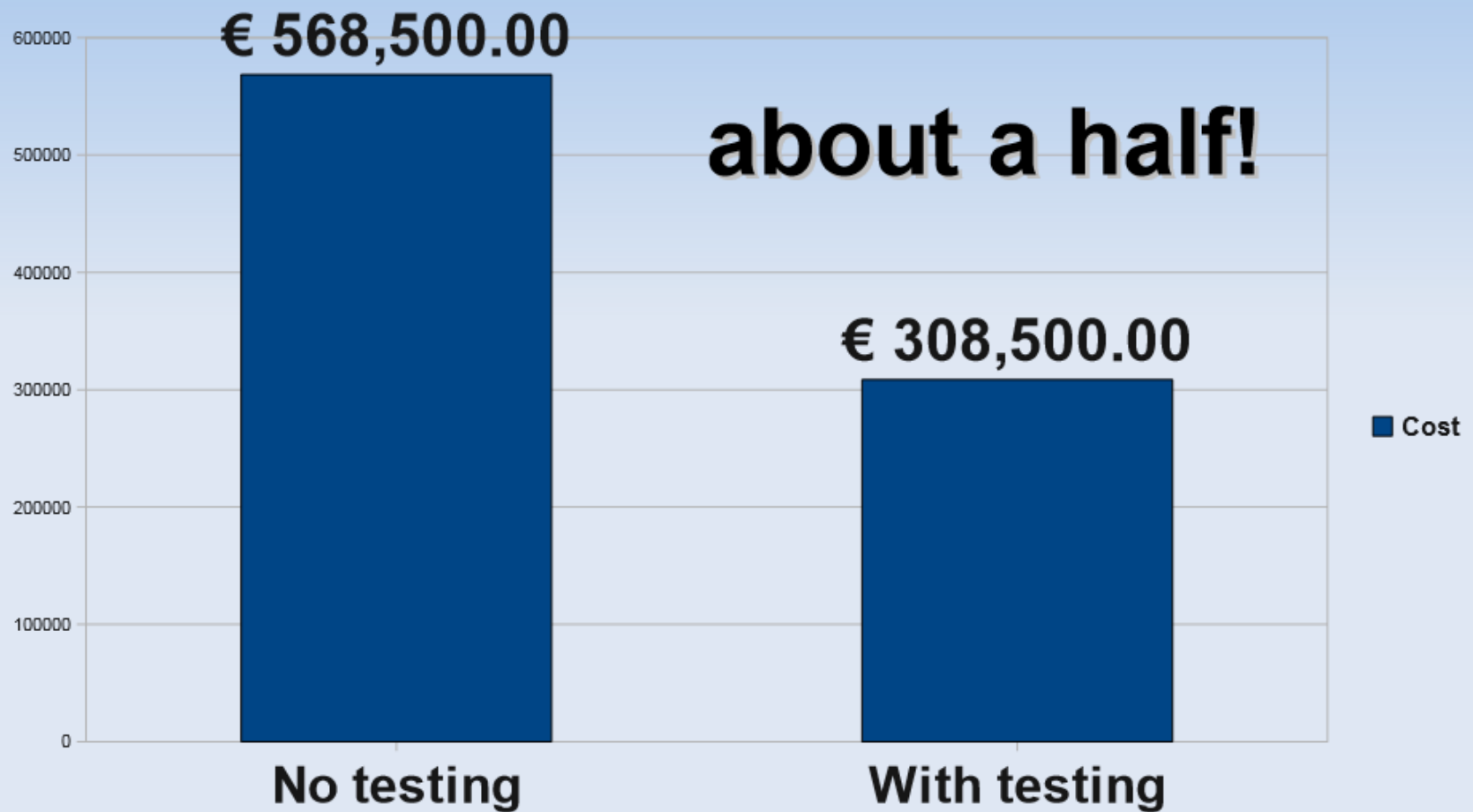
Real world (no testing)



Real world + testing



no testing vs + testing

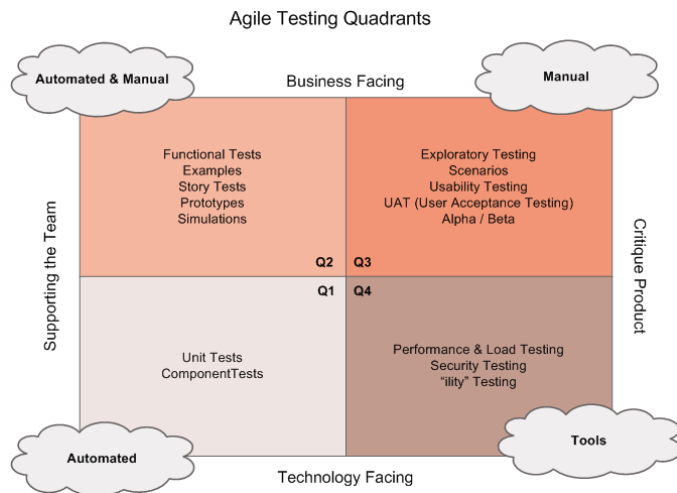


איך אתם בודקים את התוכנה שלכם?

1. מריץ בעצמי ומדבג \ `printf`
2. נותן לחברה לבדוק (QA)
3. כותב תכנית הדגמה שמריצה את הקוד שלי
4. לא בודק \ 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 קצה לקצה (משתמש\קבלה\פונקציונליות)

– האם המערכת עובדת בשלמותה?

– מנקודת מבט של המשתמש! ("קטמנדו")

– **בפרייקט**: ניסוח בדיקה באמצעות תרחיש 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?**

הדגמה קצרה

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

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

האם כדאי להשקיע בזה?

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

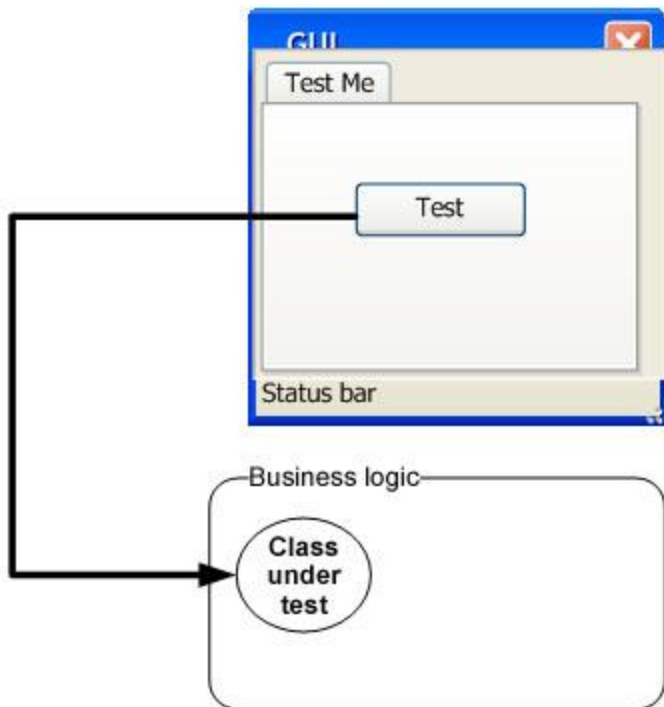
– אולי זו בדיקת אינטגרציה

– שיטה

– ביצוע חוזר

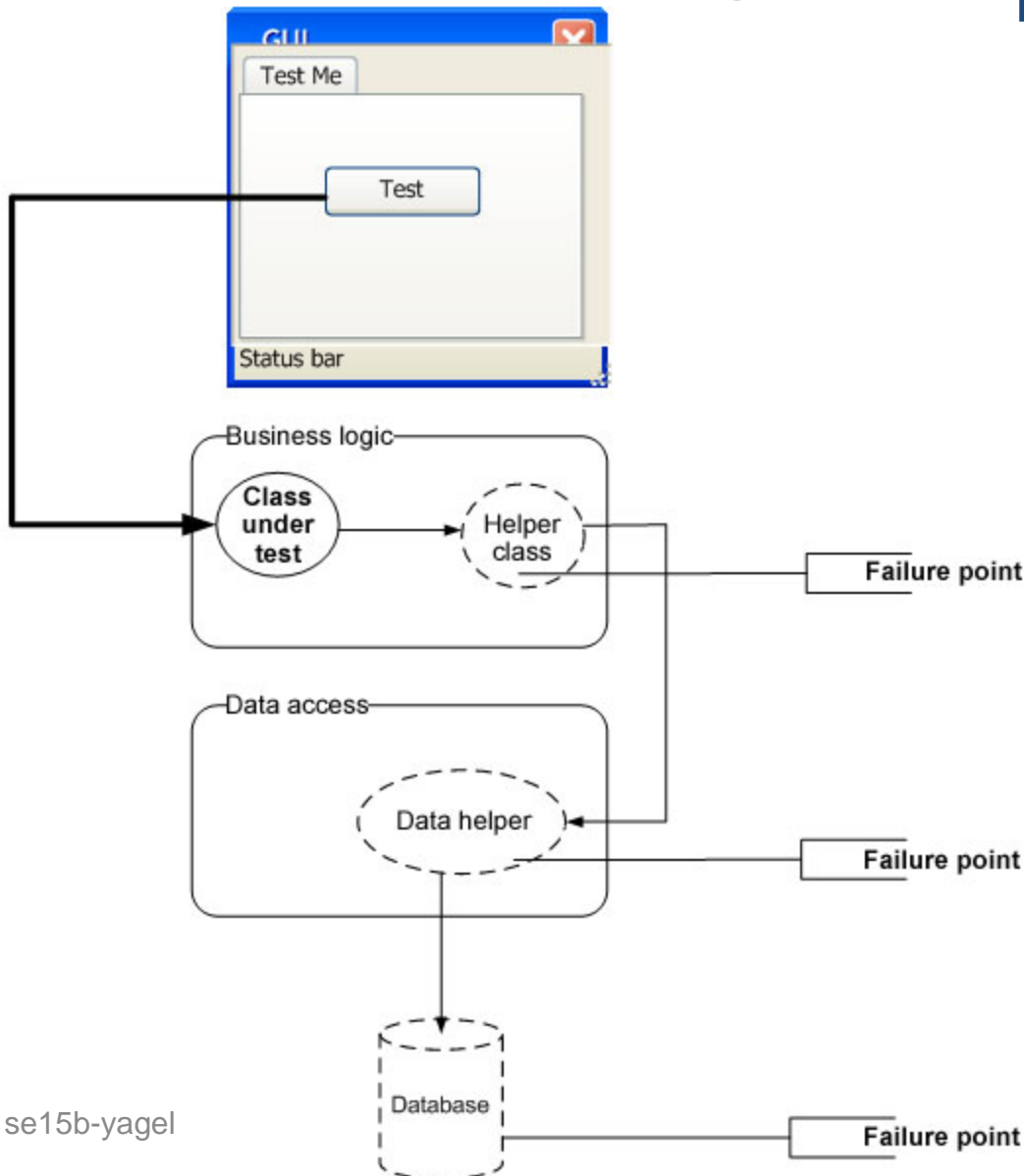
– לכל הקוד

תשתית - Framework ➤



בדיקת אינטגרציה

- בדיקת מספר רכיבים התלויים אחד בשני ביחד



בדיקות קצה לקצה (אוטומטיות)

- 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



# 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 \ בדיקת המובן מאליו \ על חשבון בדיקות אחרות
- לא תמיד קל עבור קוד קיימים (legacy)

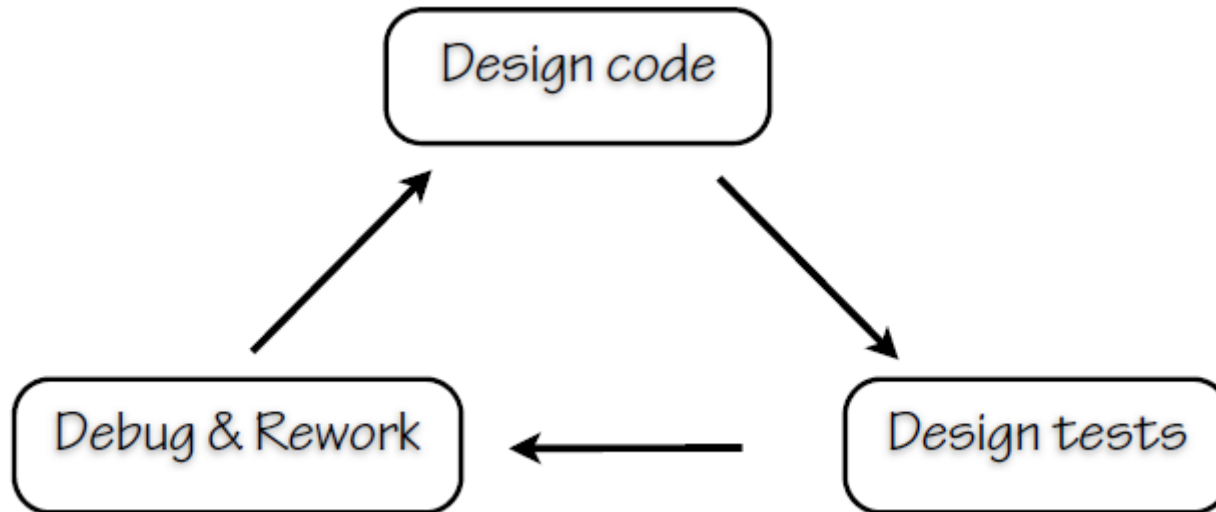
# מי בודק?

- בודקים או מפתחים?
- המטרה: מוצר בעל-ערך\איכותי
- יחס מפתח:בודק (e.g. Google vs. Microsoft)
  - מה משמעות גודל היחס?
  - Developer in testing
  - Exploratory Testing etc.
  - Why Facebook doesn't have or need testers

# מתי כותבים?

- מפל המים: Test Last, QA
- אג'ייל: קודם!
- Test First
- Test Driven Development
- מעקרונות XP, התקבל כנוהג כללי
- משפחת xDD (Behavior, Feature, ...)

# Test Last

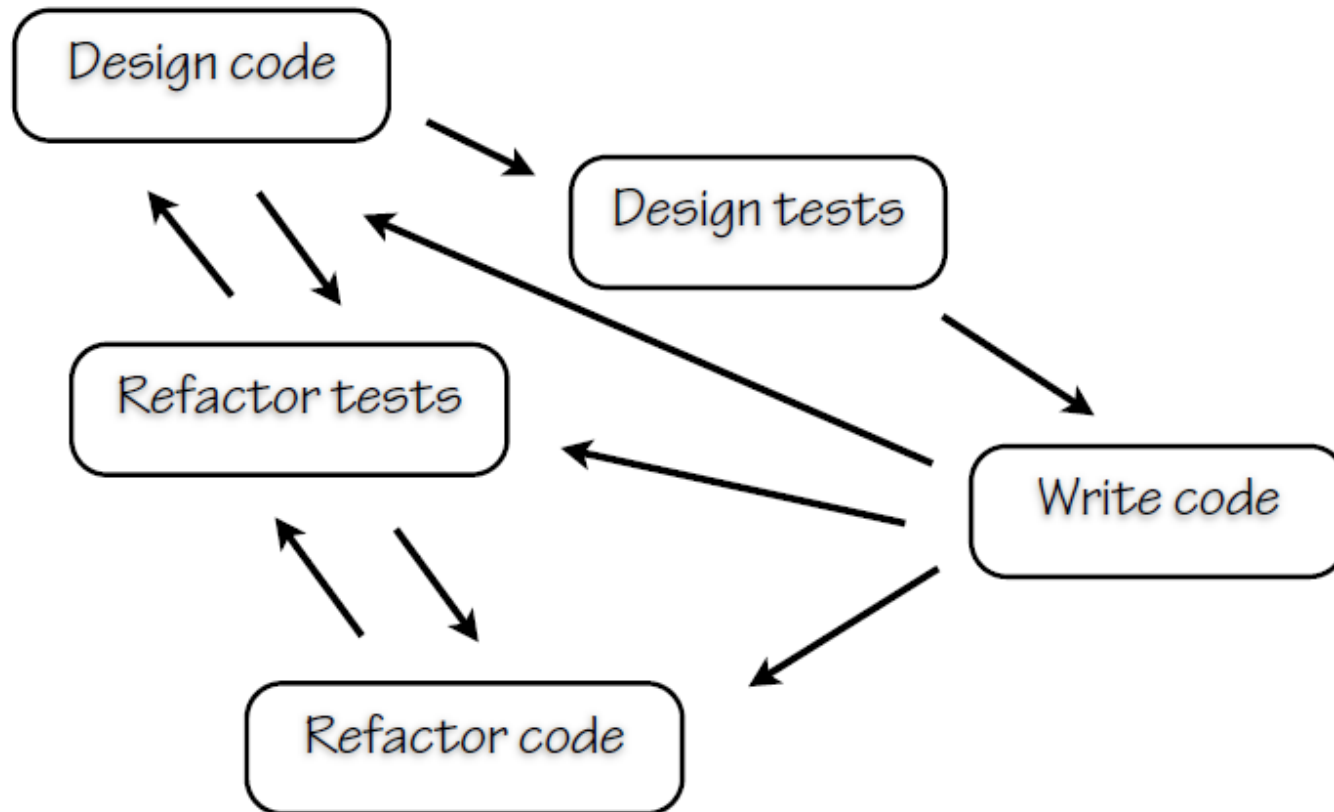


• סיכונים:

- גילוי של בעיות בדיקתיות ובאגים בשלב מאוחר
- לא נשאר זמן לבדיקות

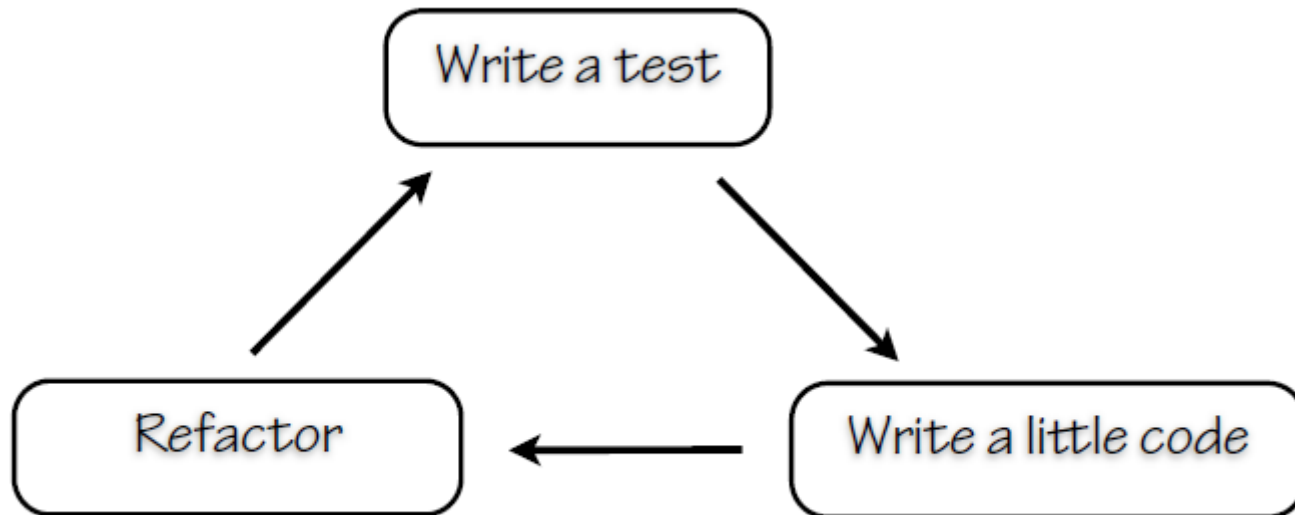
\*מתוך: <http://www.pluralsight.com/courses/unit-testing-python>

# Test First



• סיכון: עבודה מיותרת

# Test Driven



# TDD יתרונות

- 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, אפשר 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"

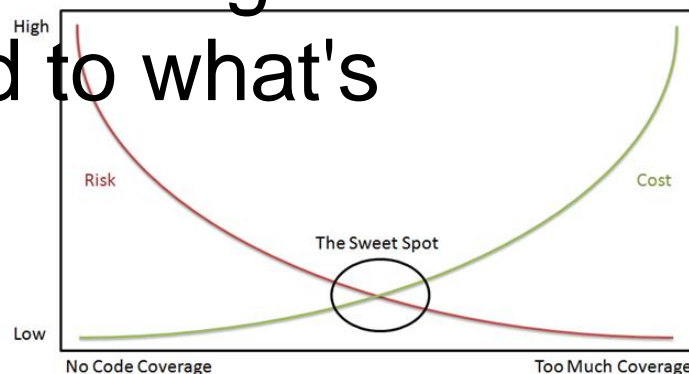
# TDD חסרונות

- 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



# כמה לבדוק?

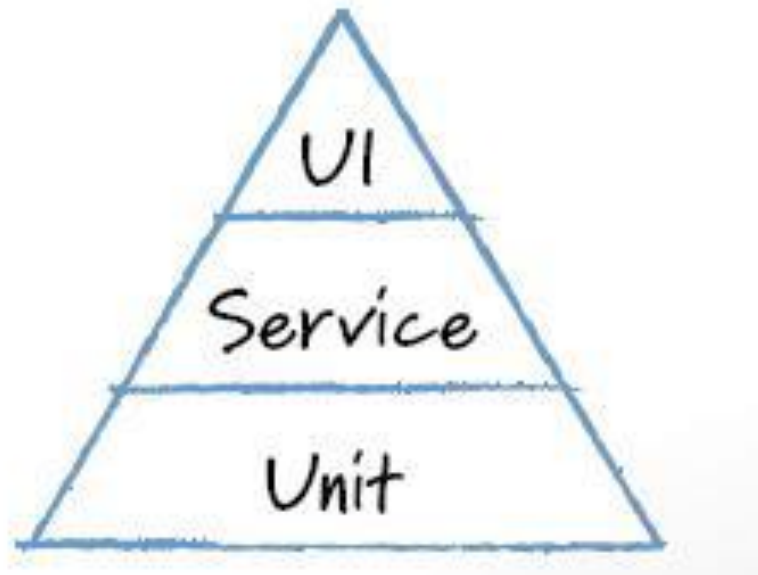
- האם צריך תמיד כיסוי של 100% של בדיקות יחידה, 100% בדיקות אינטגרציה ו-100% בדיקות קצה?
- יחס קוד:בדיקות, למשל 40:60!
- Seth Godin: “Measurement is fabulous. Unless you're busy measuring what's easy to measure as opposed to what's important”



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

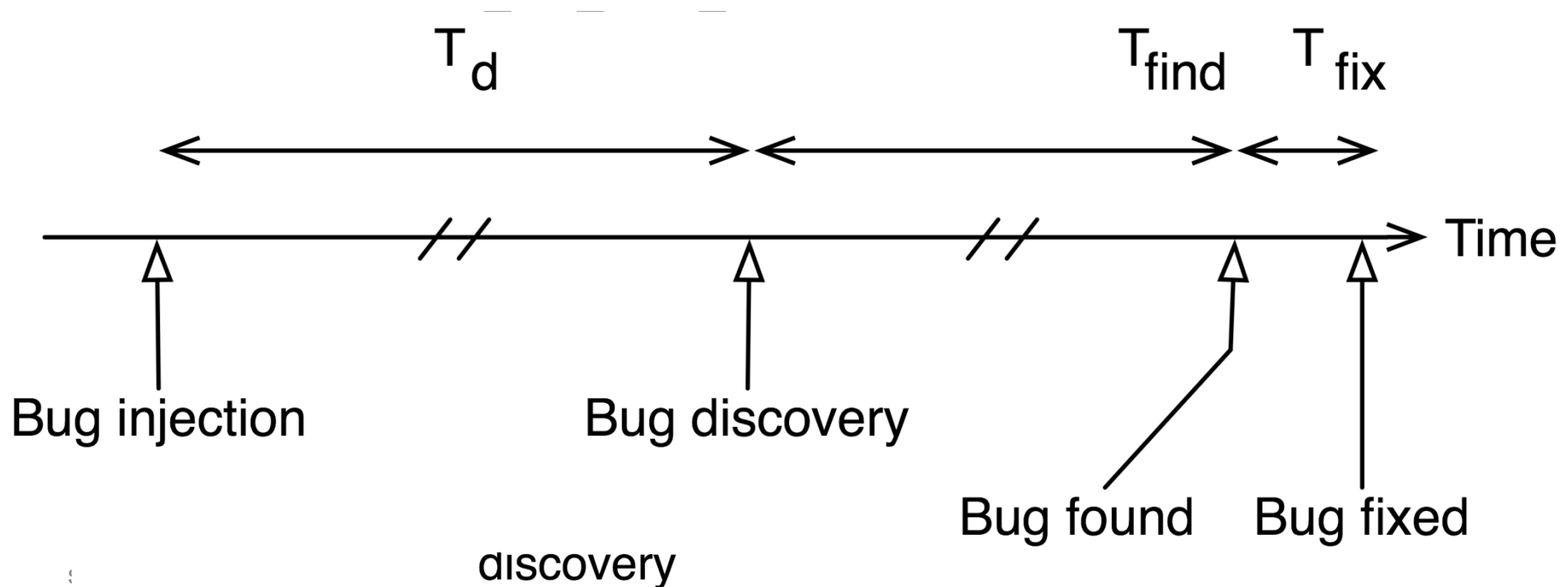
Google:

small,  
medium and  
large



# האם זה משתלם?

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



# ROI for Selected Practices

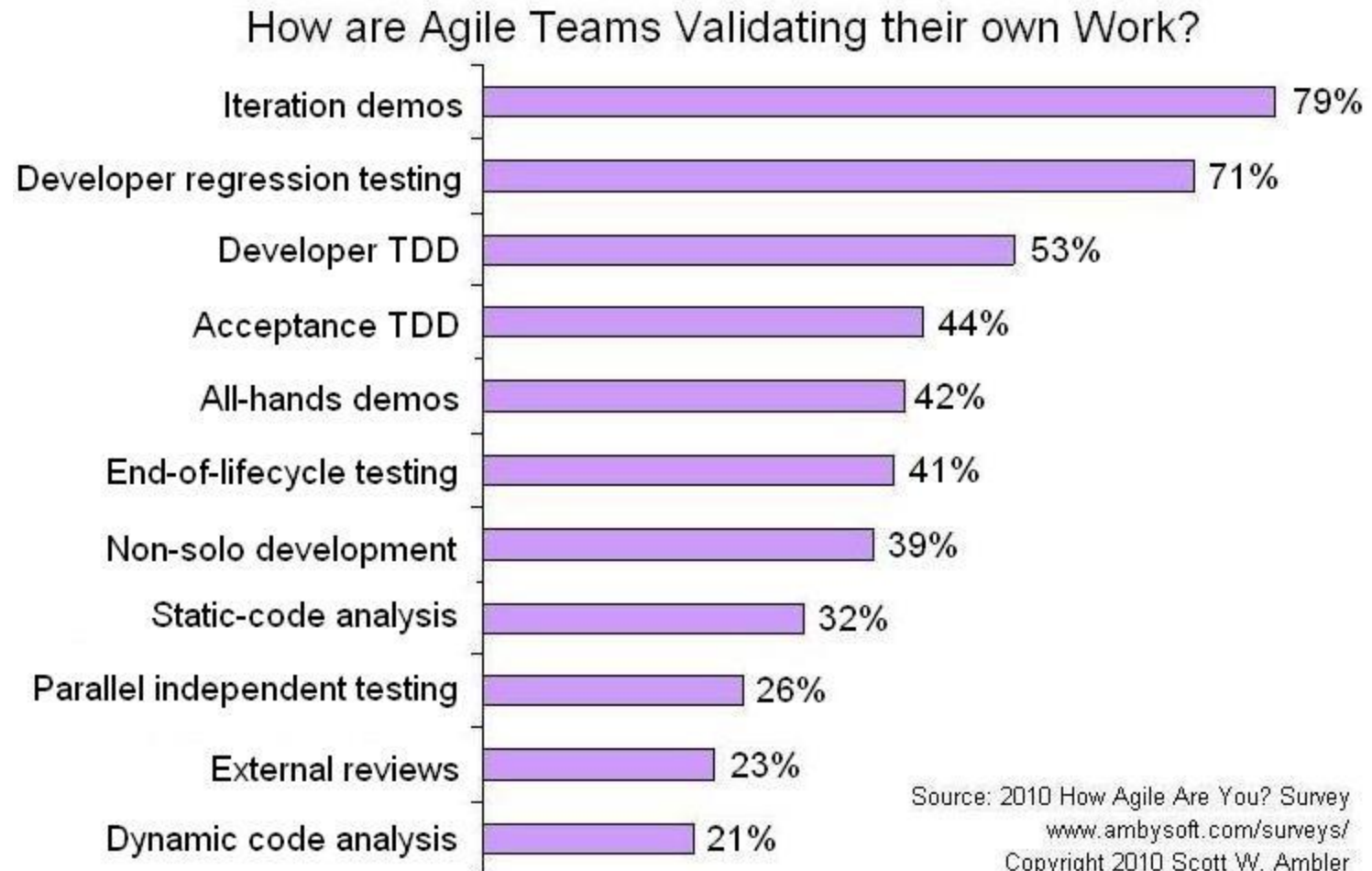
| Practice                    | 12-month ROI | 36-month ROI |
|-----------------------------|--------------|--------------|
| Test Driven Development     | -            | 1000%+       |
| PSP/TSP                     | -            | 800%         |
| Formal Inspections          | 250%         | 600%+        |
| 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/>

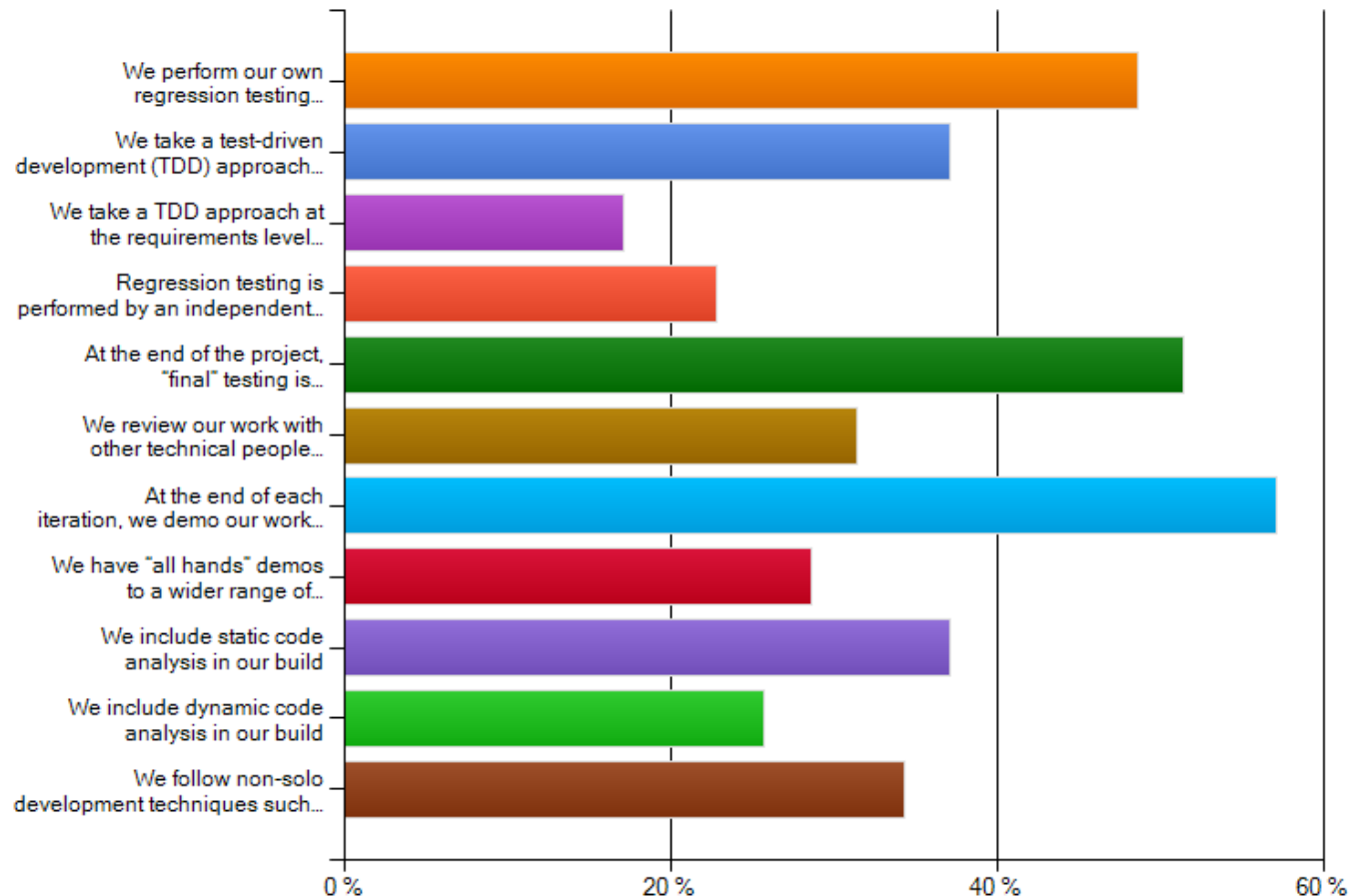
# How Agile Are You? 2010 Survey



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

# 2013

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

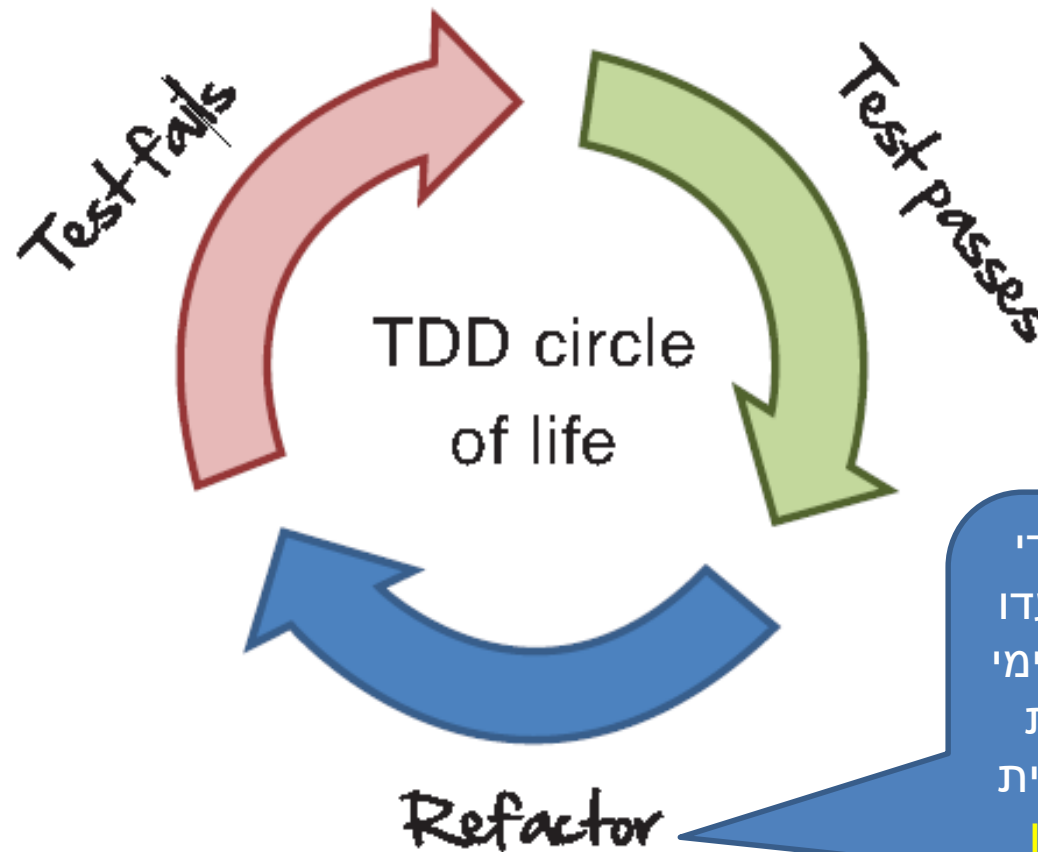


# מצד שני

- “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](#))

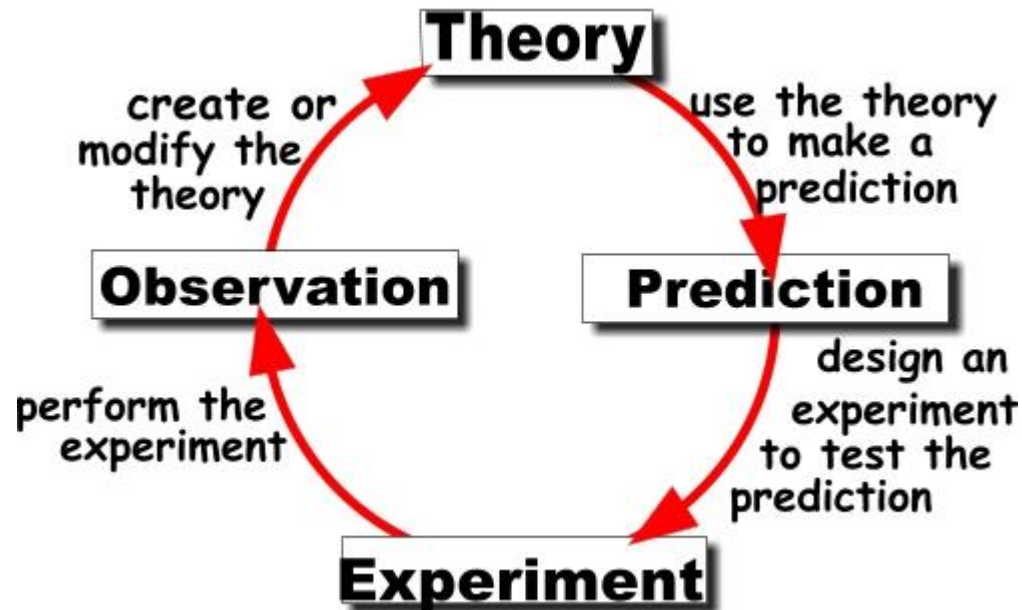


# TDD = TFD + Refactoring

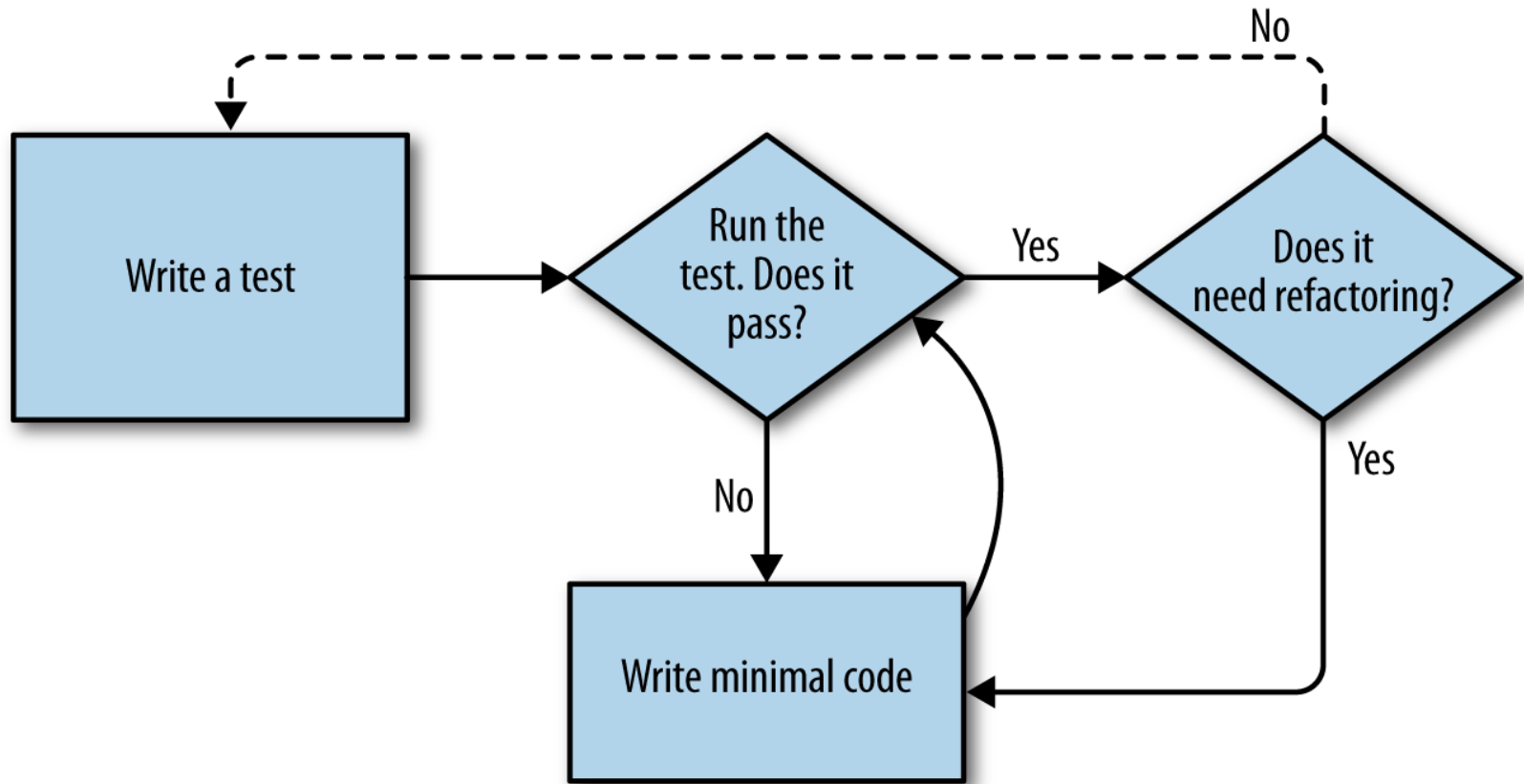


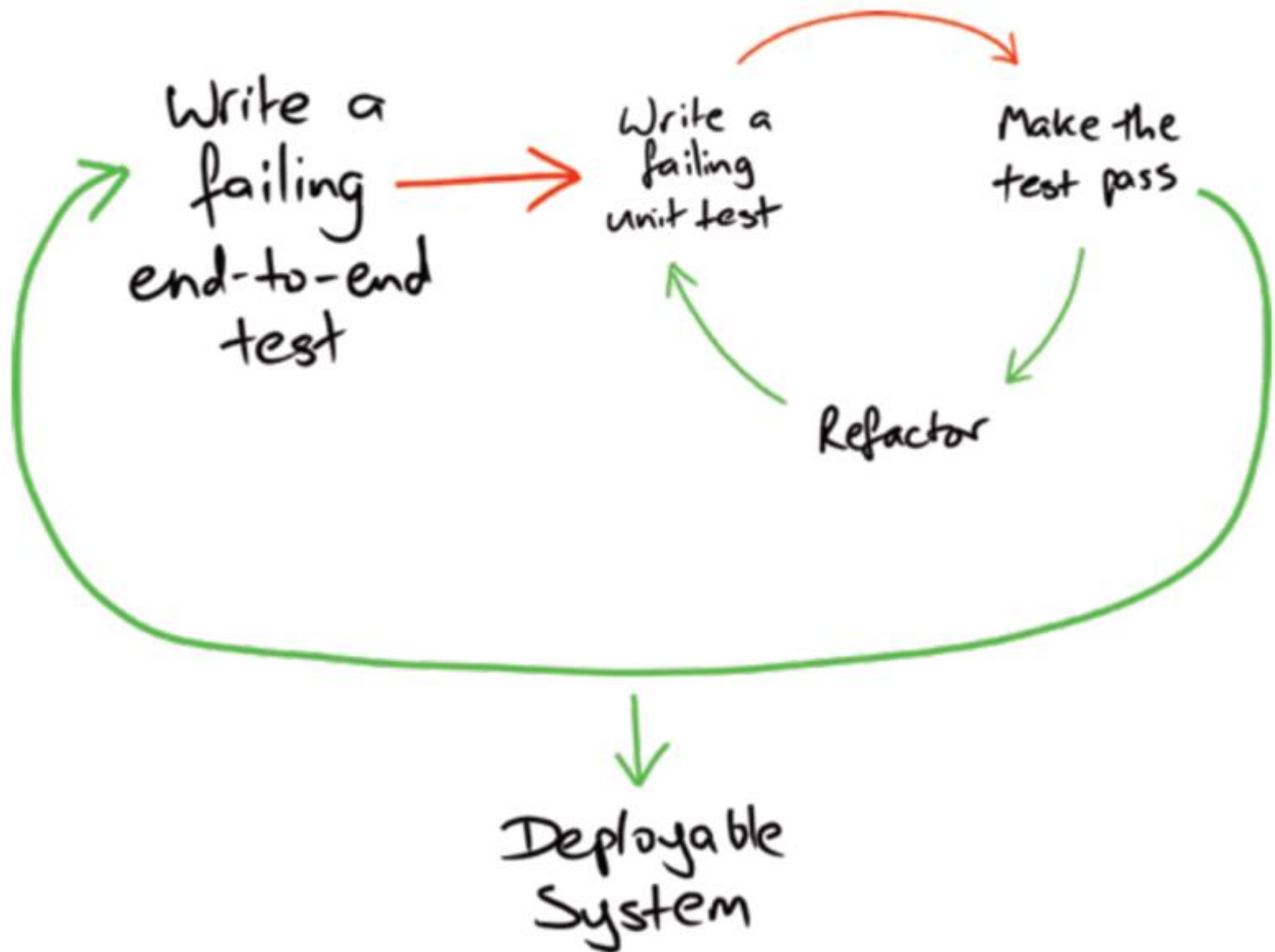
שיפור קוד קיים, על ידי שימוש בטכניקות שנועדו לשפר את המבנה הפנימי של הקוד מבלי לשנות את ההתנהגות החיצונית שלו (ויקיפדיה), **תיכון מתמשך**

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

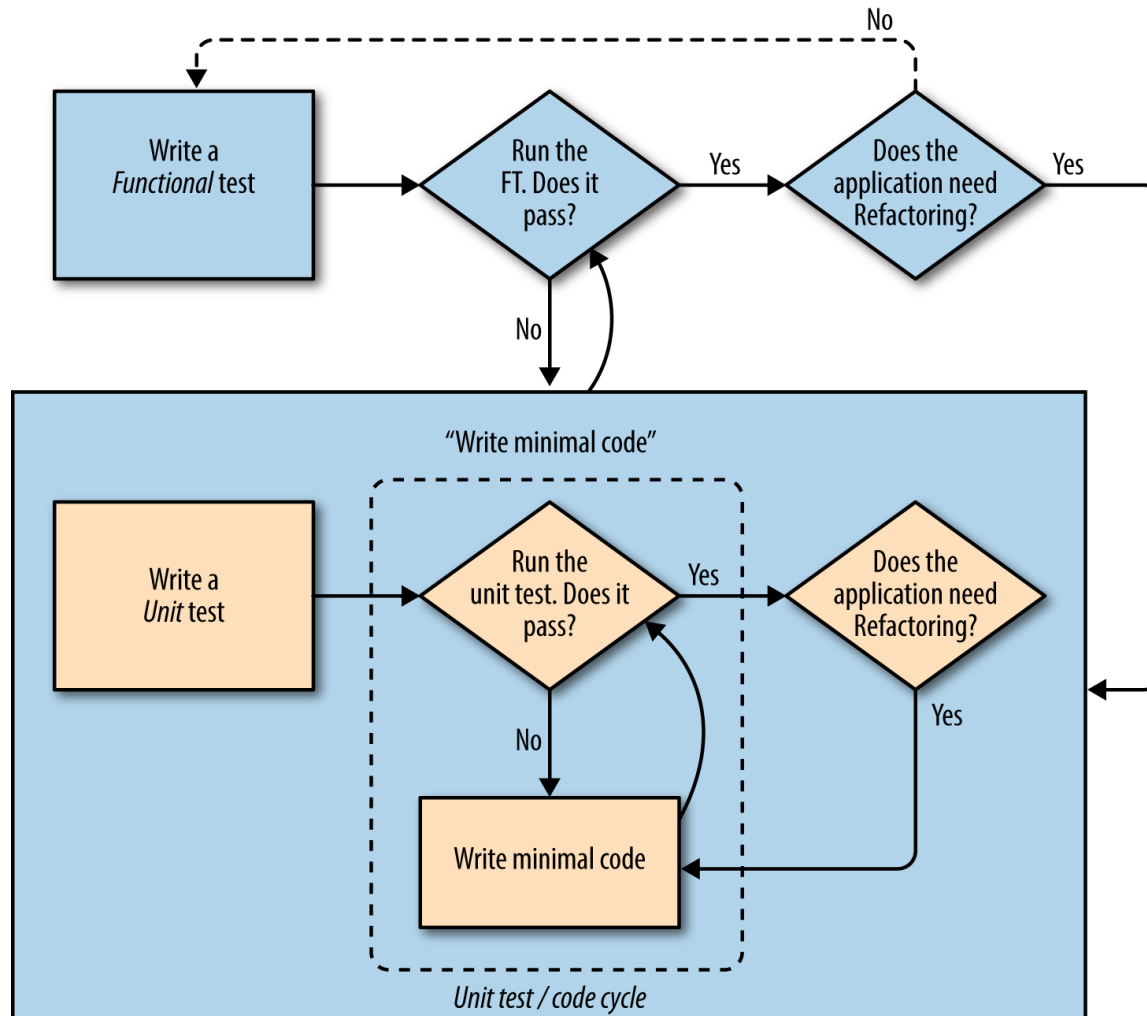


# TDD Process





# Combined FT & UT



# ?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)



מהי בדיקת יחידה טובה?

# Unit tests should be FIRST

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



# Unit tests should be FIRST

(adopted from A. Fox, Berkeley)

- **Fast:** run (subset of) tests quickly (since you'll be running them *all the time*)
- **Independent:** no tests depend on others, so can run *any subset* in *any order*
- **Repeatable:** run N times, get same result (to help isolate bugs and enable automation)
- **Self-checking:** test can *automatically* detect if passed (*no human checking* of output)
- **Timely:** written about the same time as code under test (with TDD, written *first!*)

# xUnit Frameworks

- כלים לבדיקות יחידה

- '94, Kent Beck, SUnit– Small Talk

- ~'00, +E. Gamma, JUnit ("Test Infected")

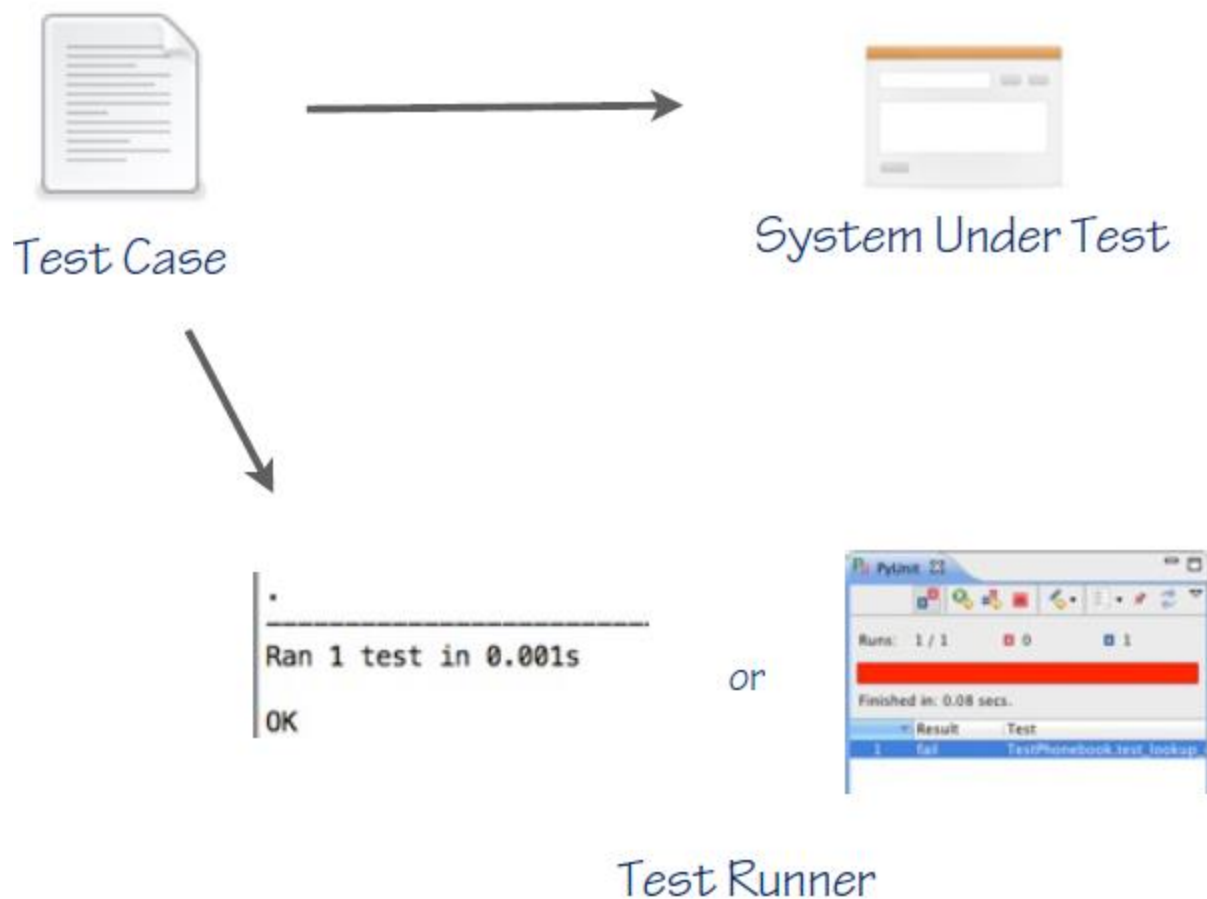
- ייצוא לשפות רבות: CppUnit, PyUnit ועוד

- <http://www.xprogramming.com/software>

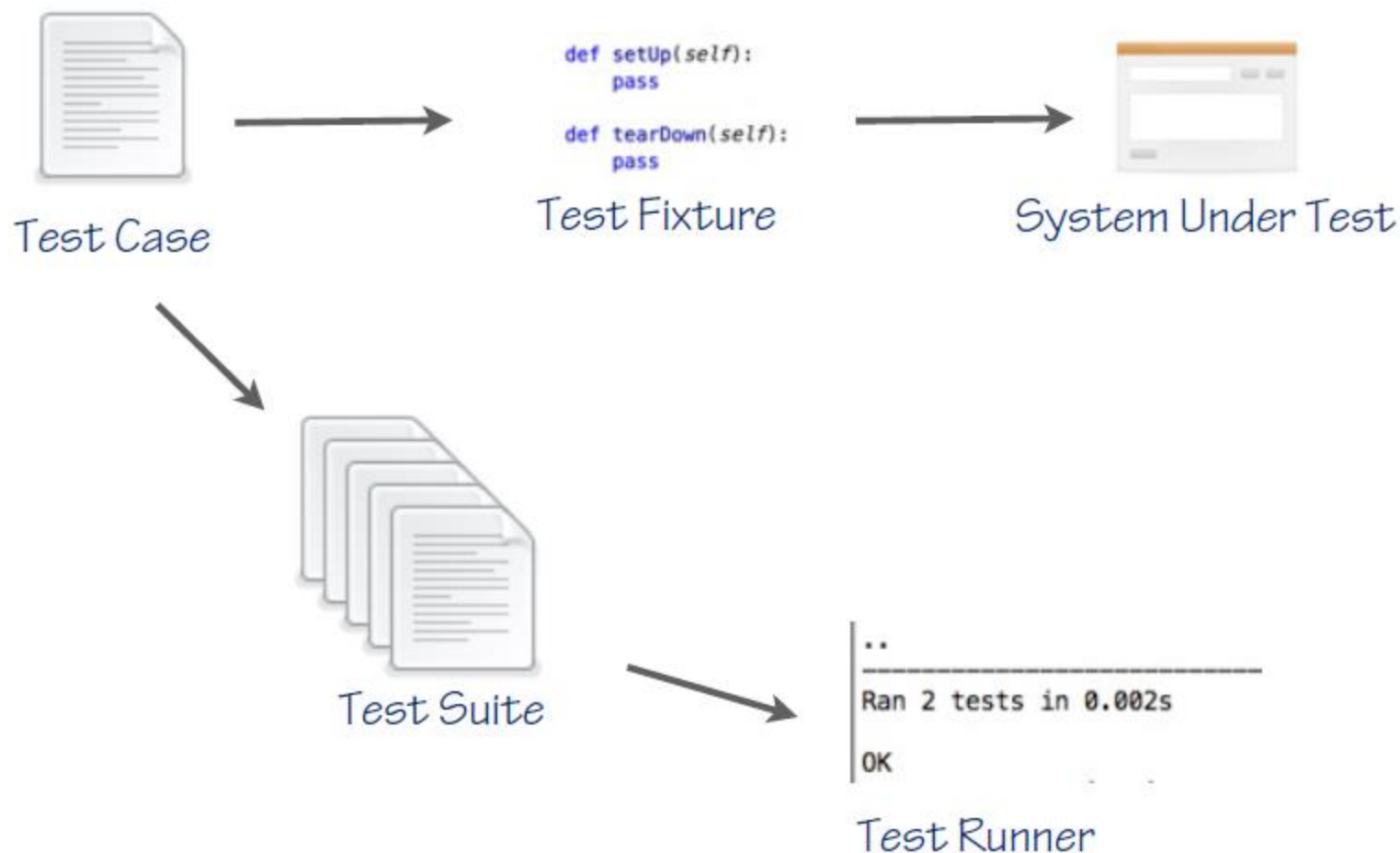
- [http://en.wikipedia.org/wiki/List\\_of\\_unit\\_testing\\_frameworks](http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks)

- ארכיטקטורה סטנדרטית לבדיקות יחידה

# מונחים מקובלים



# מונחים מקובלים



# 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() {
 int result = new Calc().add(2, 3);
 assertEquals(5,result);
 }
}
```

// SUT

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

# רכיבים עיקריים בקוד בדיקה (python unittest)

```
import unittest
```

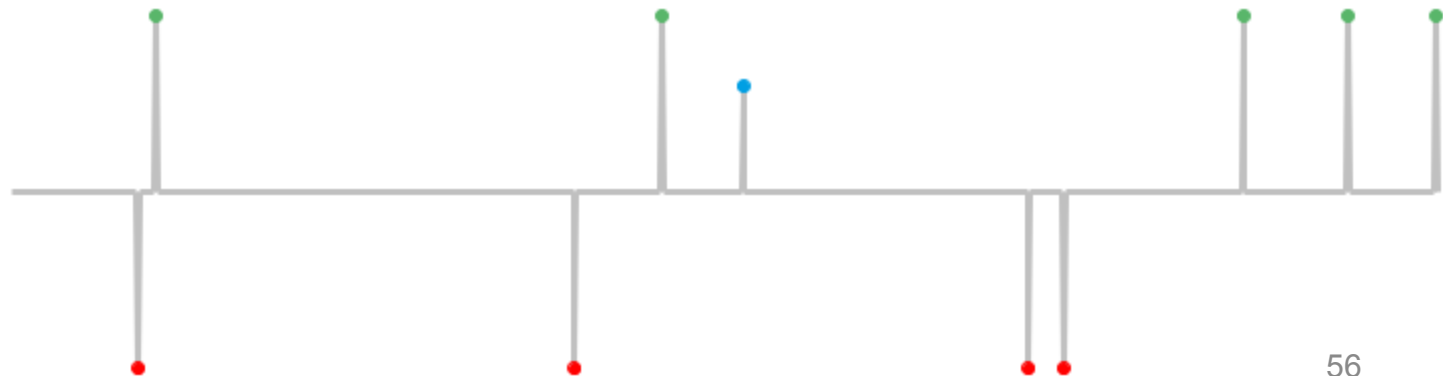
```
class CalcTest(unittest.TestCase):
 def test_add(self):
 result = Calc().add(2, 3)
 self.assertEqual(5, result)
```

```
test runner
```

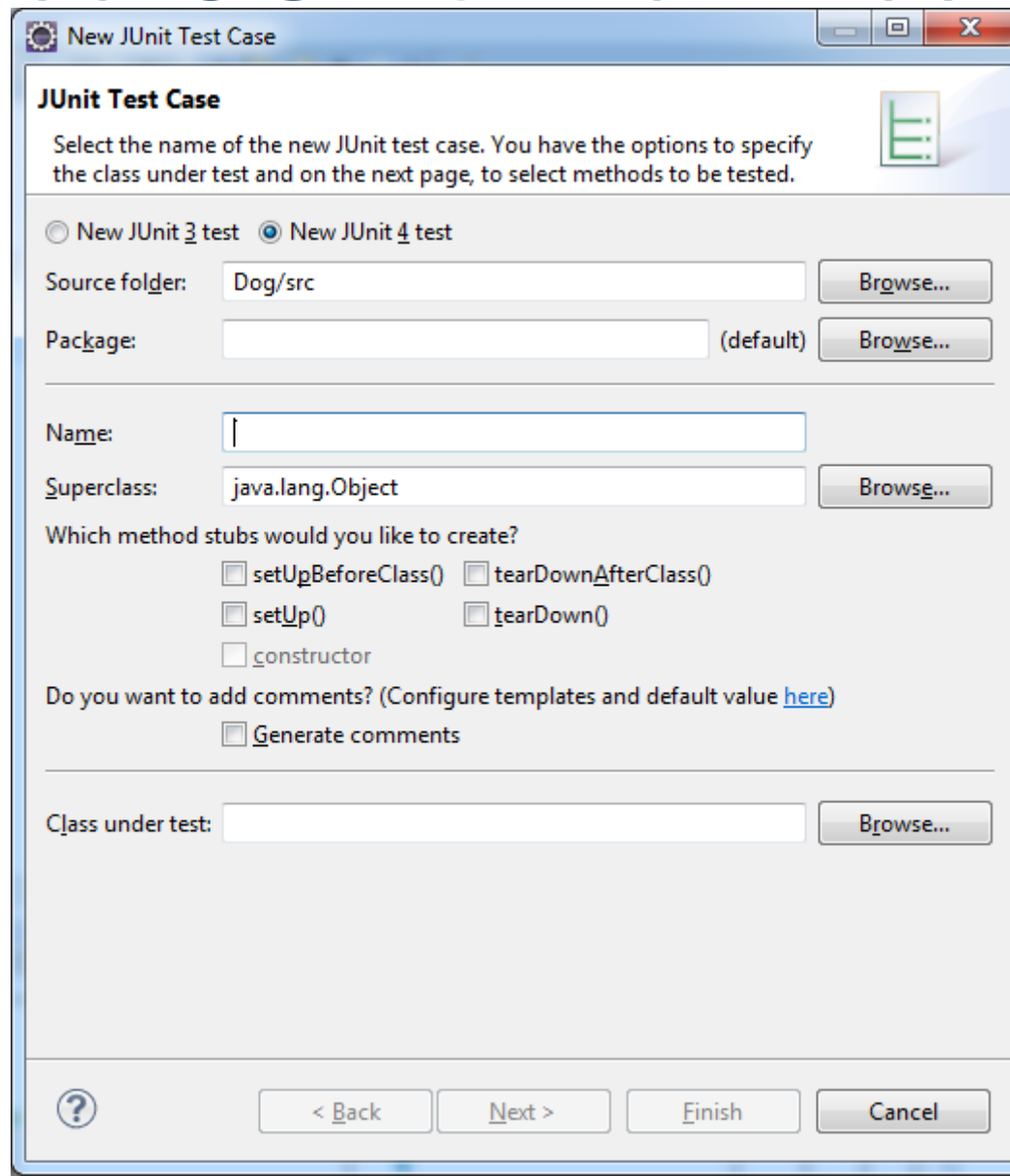
```
if __name__ == '__main__':
 unittest.main()
```

# 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.' To the right of this text is a small icon of a document with a green 'E'.

There are two radio buttons: 'New JUnit 3 test' (unselected) and 'New JUnit 4 test' (selected).

Fields and buttons:

- 'Source folder:' with text 'Dog/src' and a 'Browse...' button.
- 'Package:' with an empty text box, '(default)' text, and a 'Browse...' button.
- 'Name:' with an empty text box.
- 'Superclass:' with text 'java.lang.Object' and a 'Browse...' button.
- 'Which method stubs would you like to create?' section with five checkboxes:
  - ☐ setUpBeforeClass()
  - ☐ tearDownAfterClass()
  - ☐ setUp()
  - ☐ tearDown()
  - ☐ \_constructor
- 'Do you want to add comments? (Configure templates and default value [here](#))' with a checkbox for 'Generate comments'.
- 'Class under test:' with an empty text box and a 'Browse...' button.

At the bottom, there is a question mark icon, and four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.



# Python Tools

- Tools:
  - Library: unittest ([2.7](#))
    - Other: PyTest, Nose
  - Runners: CLI (pythonanywhere?), Eclipse, PyCharm

# 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!

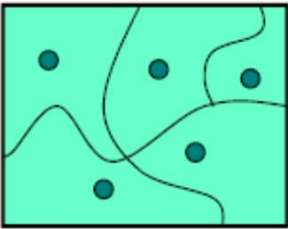
- Given a person name, return all posts (of a map) containing it's name (in any of a post fields)
- Given a name, check if the map includes a location information (place or geo. location)
- Check if there are map inconsistencies, e.g., the same name with different locations.
- Example: Or (R.I.P) appears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

# TDD Coding Katas

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



# בהרצאת המשך \ נושאים מתקדמים



- בדיקות יחידה 2.0 למשל...

- מאפיינים מתקדמים של UnitX: 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 (עמוד הבא)

- קריאה מומלצת להרצאת המשך:  
Using Mock Objects



# משימה אישית 4 – בדיקות יחידה + TDD

- FindAPerson  
אחרים, למשל: [String Calculator Kata 1](#) (עד ש' 5)
- TDD ה"בדיקות" מובילות את הפיתוח
- אחרי כל צעד (red-green-refactor) יש לבצע commit עם הערה שמתחילה בסוג הצעד (למשל: RED: test new line)
- דחיפה ל-github והגשת הקישור למאגר
- דרישות: כיסוי מלא, RGR, נכונות, איכות.
- בזוגות מתחלפים (אחד בודק אחד מממש ([pairhero](#)), כולל ב-commits) – אופציה: ?hangout
- הגשה ע"י pull request וכו' ראו הגדרות ש.ב. 4

# לסיכום

- בדיקות/בדיקות יחידה, פיתוח מונחה בדיקות, xUnit
- בפרויקט

- בכל סבב: ניסוח בדיקת קבלת לתרחיש עיקרי
- משימת סבב: בדיקות לרכיב מרכזי

“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>