



**ALL CODE IS GUILTY  
UNTIL PROVEN INNOCENT**

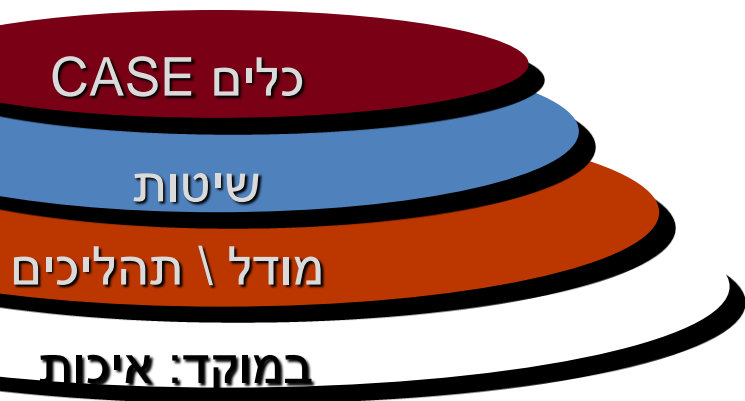
**CODESMACK**

# 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.



# מה היום?

- בדיקות (מבוא)
  - בדיקות יחידה (Unit Testing), פיתוח מונחה בדיקות (TDD)
  - כלי בדיקה (JUnit)
- הדגמה
- שעה אחרונה: מעבדת בדיקות יחידה
- פרויקט: המשך עבודה על סבב 1-MVP הצגה, רטרוספקטיבה ותכנון לסבב 2
  - פגישות וסקרים, ציון לפי איכות והתקדמות
  - משימה אישית 4: TDD ו-בדיקות יחידה
  - פרויקט: משימת סבב 2: בדיקות יחידה



# Remarks by Bill Gates

17th Annual ACM Conference on Object-Oriented Programming, Seattle, Washington, November 8, 2002



“... When you look at a big commercial software company like Microsoft, there's actually as much testing that goes in as development.

We have as many testers as we have developers. Testers basically test all the time, and developers basically are involved in the testing process about half the time...

# Remarks by Bill Gates

17th Annual ACM Conference on Object-Oriented Programming, Seattle, Washington, November 8, 2002



"... We've probably changed the industry we're in. We're not in the software industry - we're in the testing industry, and writing the software is the thing that keeps us busy doing all that testing."

"...The test cases are unbelievably expensive; in fact, there's more lines of code in the test harness than there is in the program itself. Often that's a ratio of about three to one (!)."

# 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?

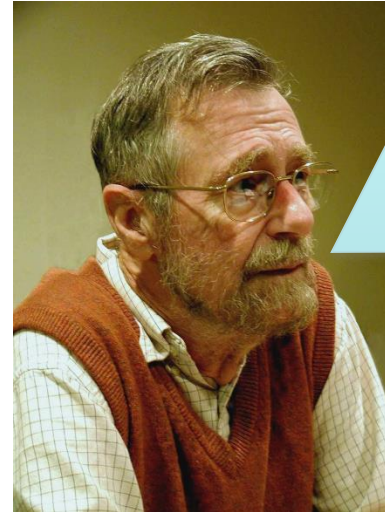
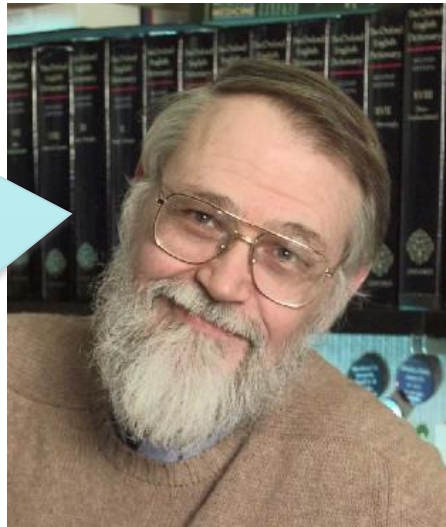
- לוח לדוגמא





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.

se14b-yagel



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

Source: <sup>9</sup>[saas](#)

# בדיקות תוכנה

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

Boeing 787 wing break test –

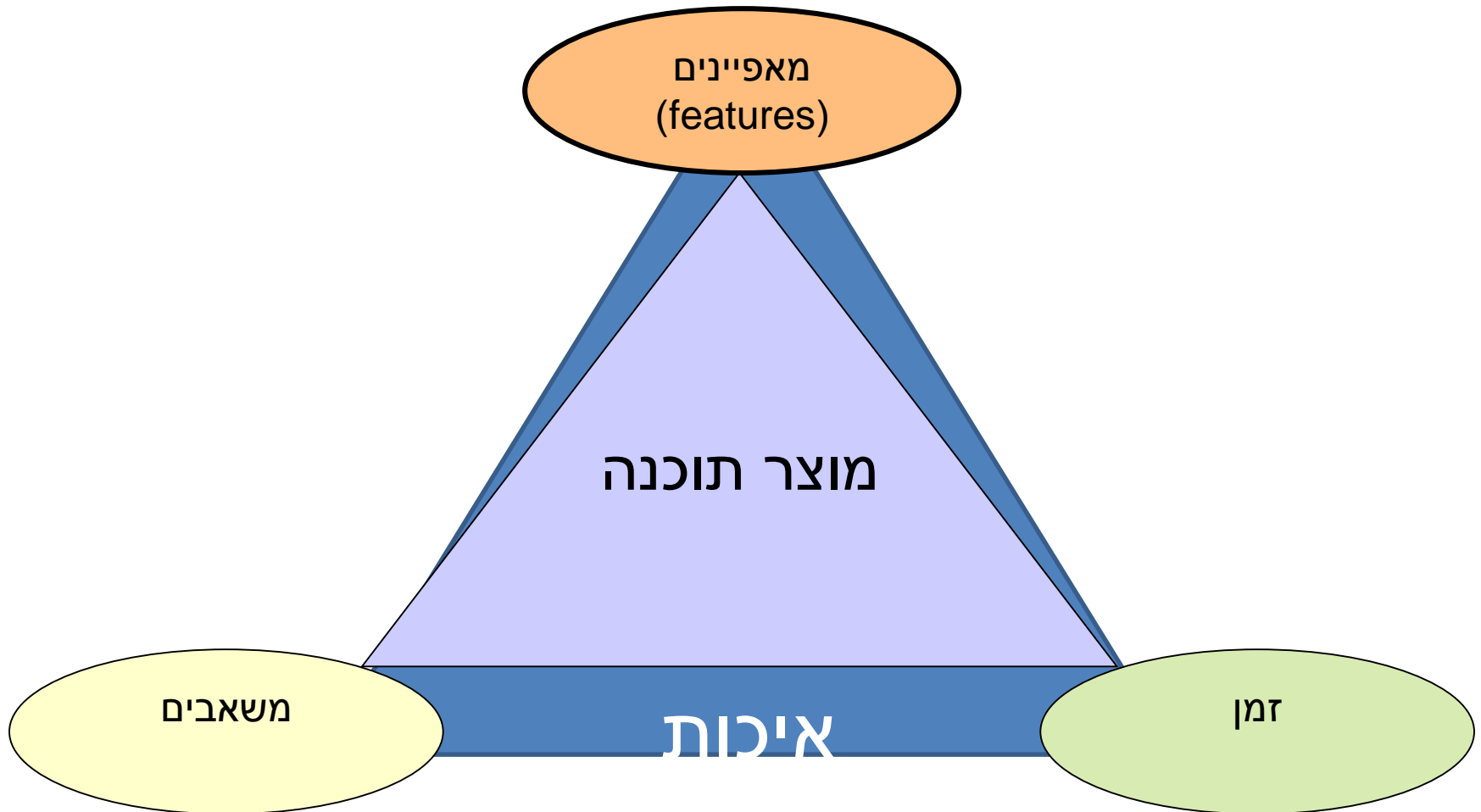
- האם זה באמת כדאי?
- למה עכשיו?

# למה לבדוק?

- נכונות \ איכות בכלל
- אפשרות לשינויים
- ...

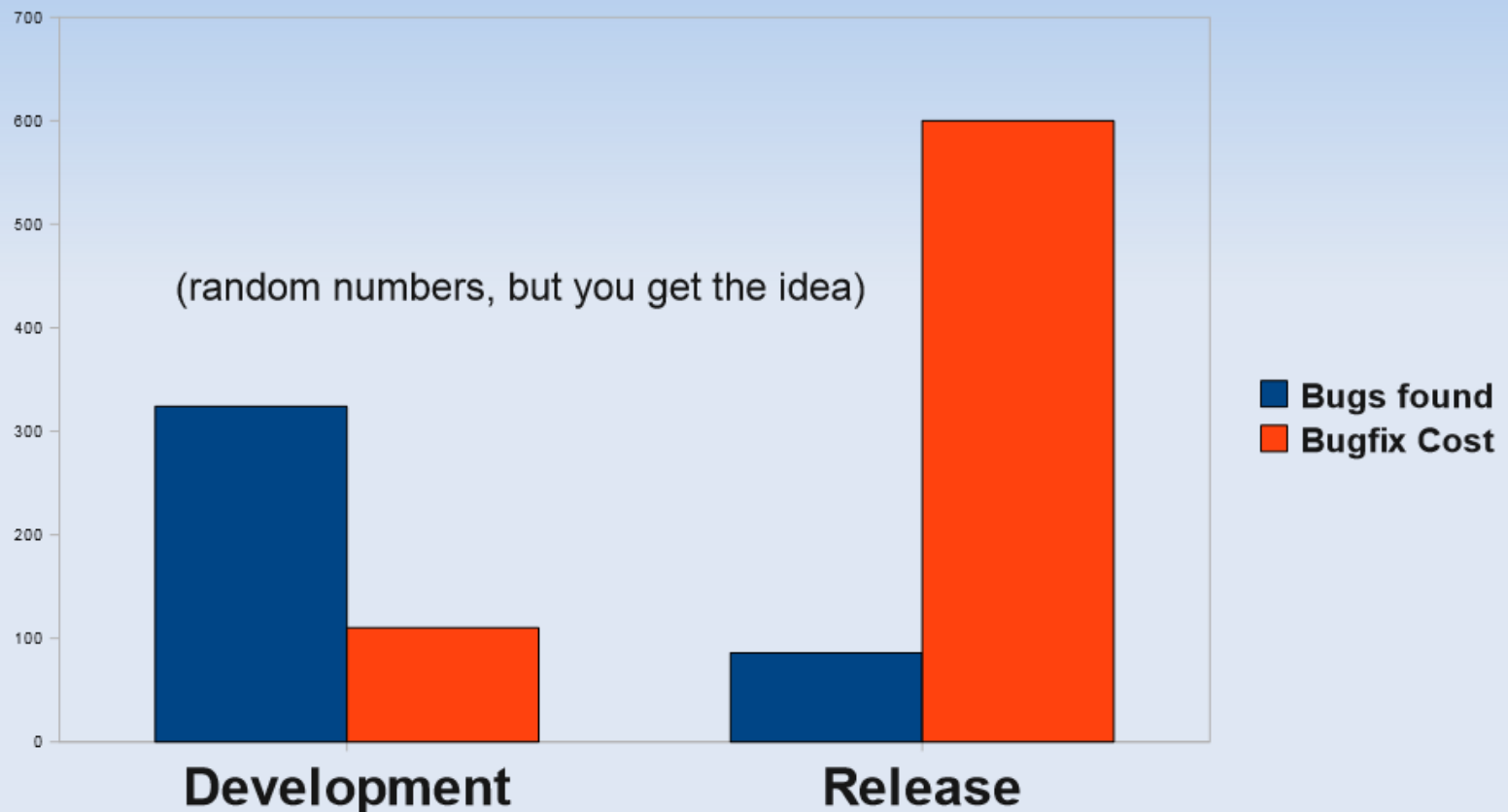


# תזכורת: פרויקט תוכנה:

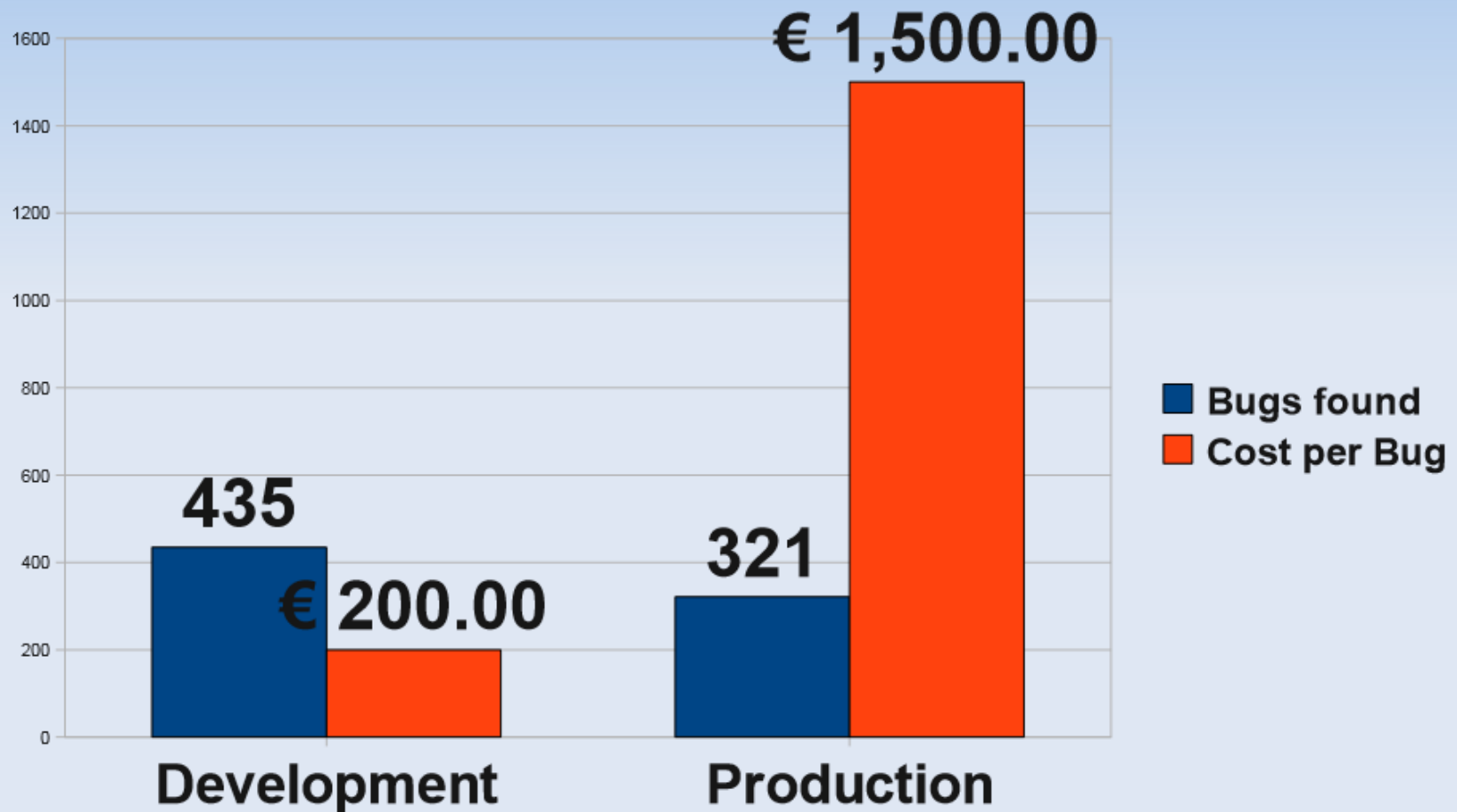


# למה לבדוק? (באדום עלות לבאג אחד)

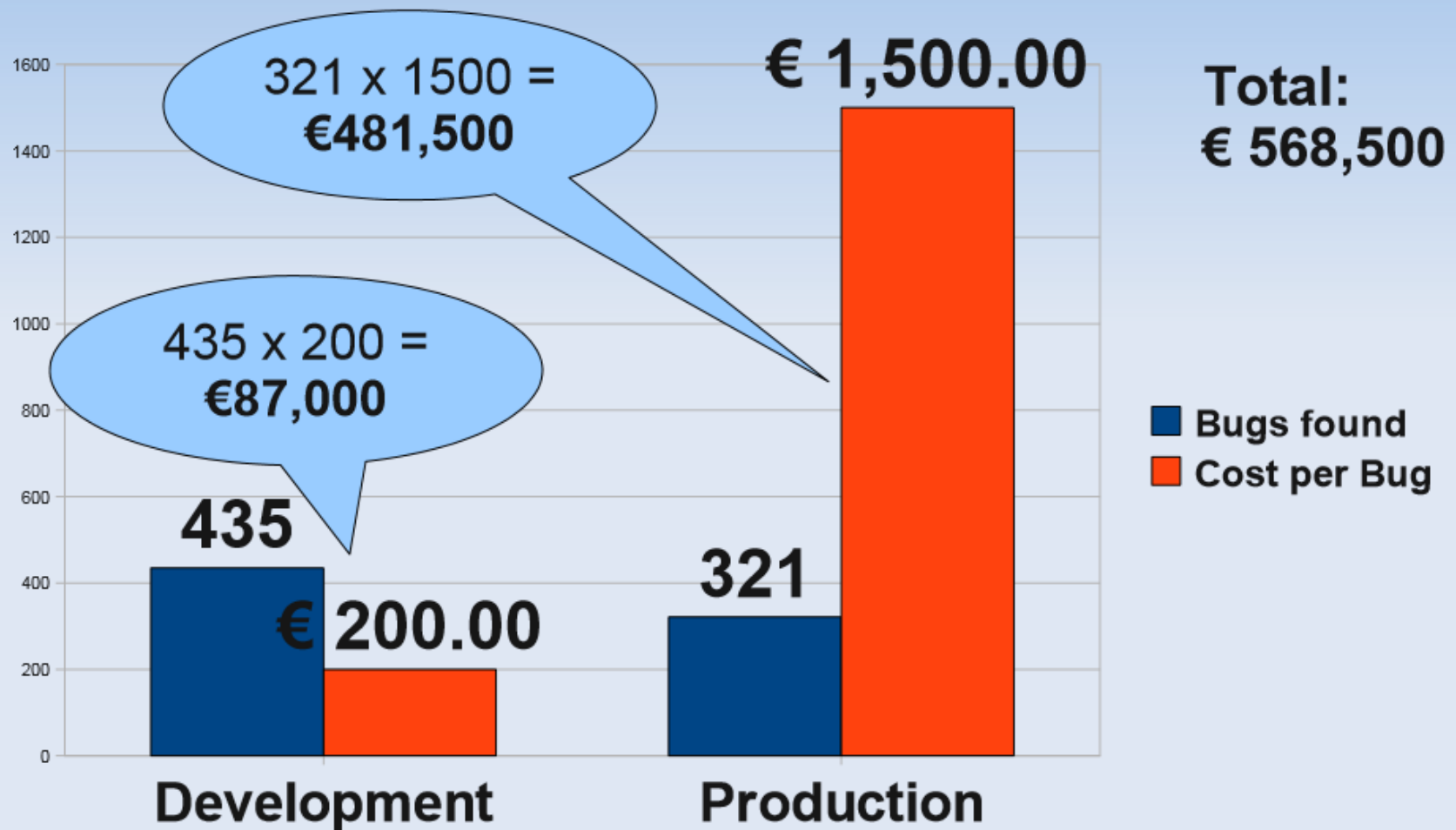
**Simplest case:  
Development => Production**



# Real world (no testing)

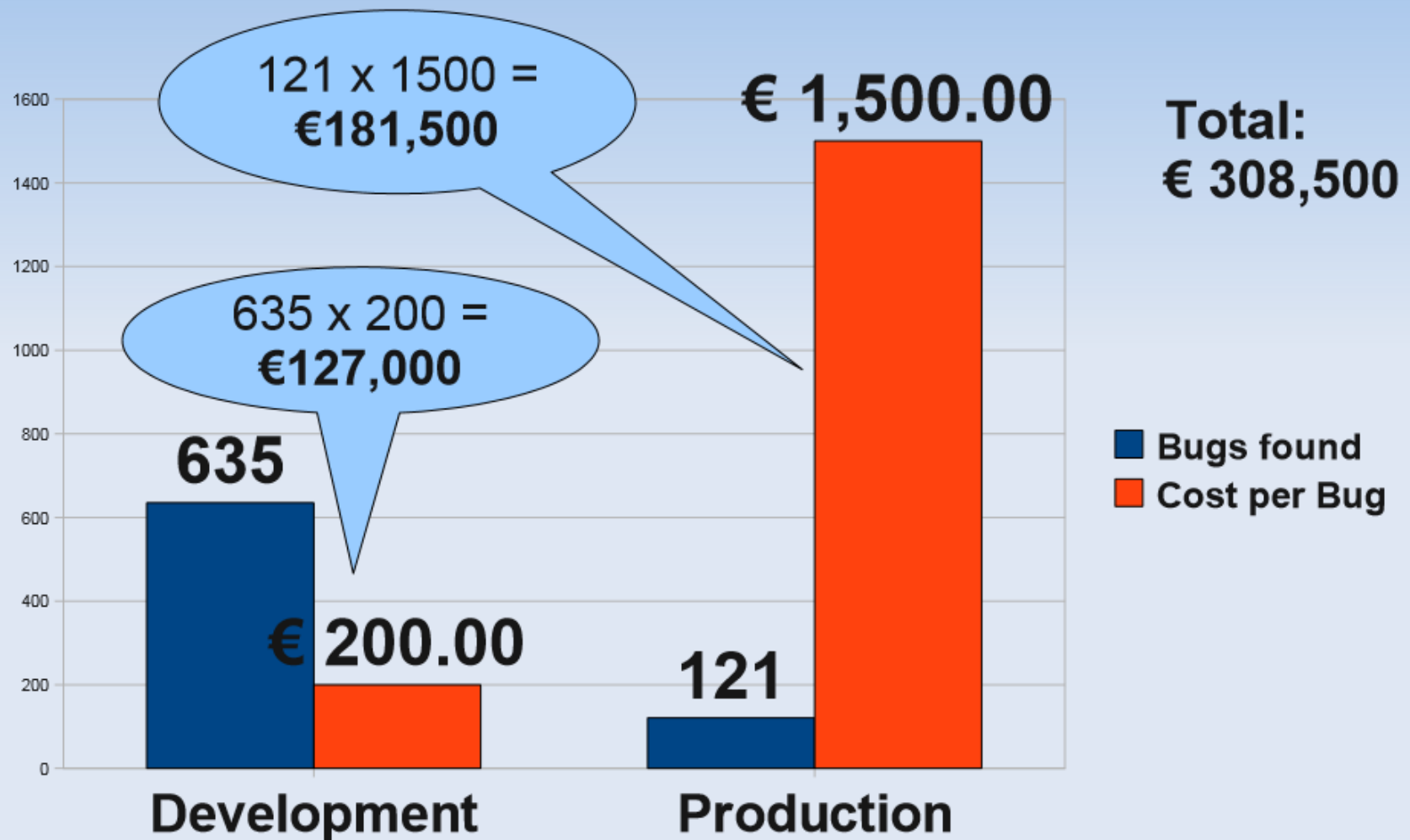


# Real world (no testing)

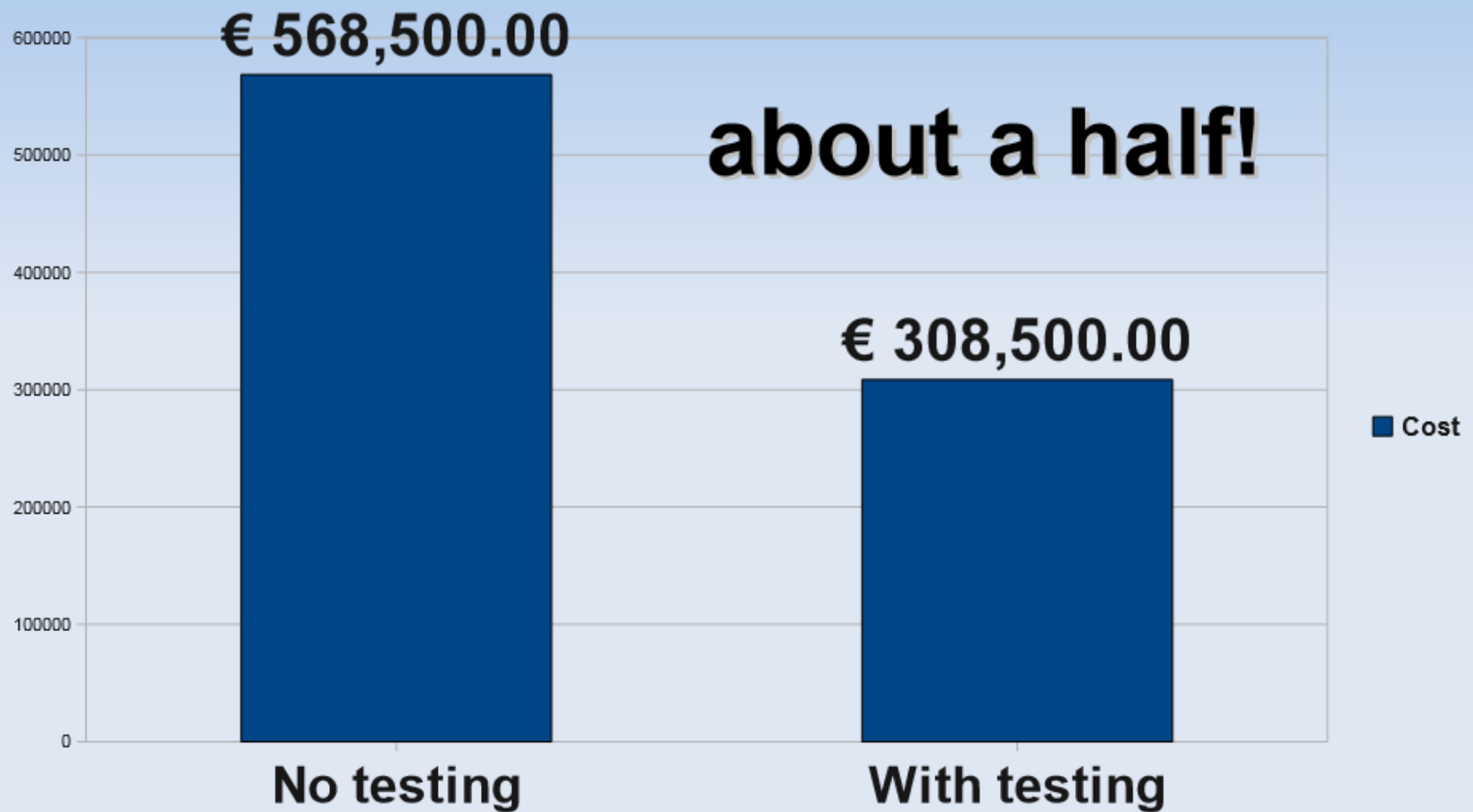




# Real world + testing



# no testing vs + testing

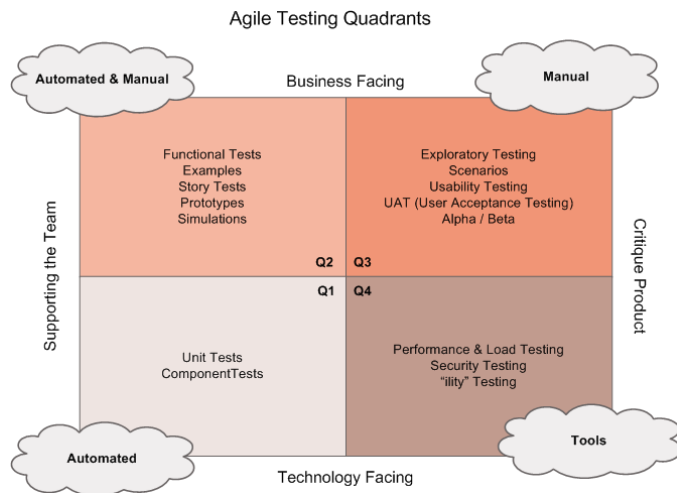


# איך אתם בודקים את התוכנה שלכם?

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?
  - **בפרייקט**: 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?**

# הדגמה

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

# 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?

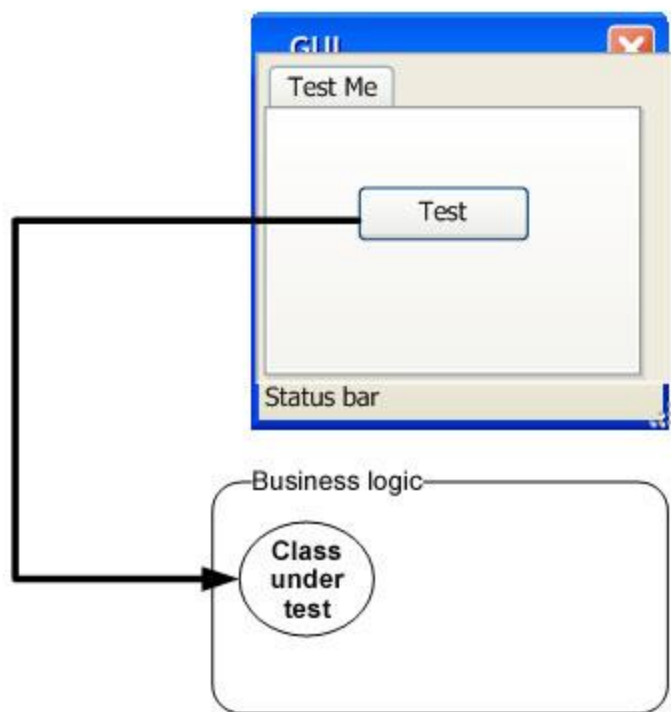
- אולי מספיקות בדיקות אינטגרציה ומערכת?
- זהו קוד שגם מצריך תחזוקה!
- בעצם אולי כבר כתבתם עד היום כאלו דברים

– הדגמה Dog

- מה חסר?

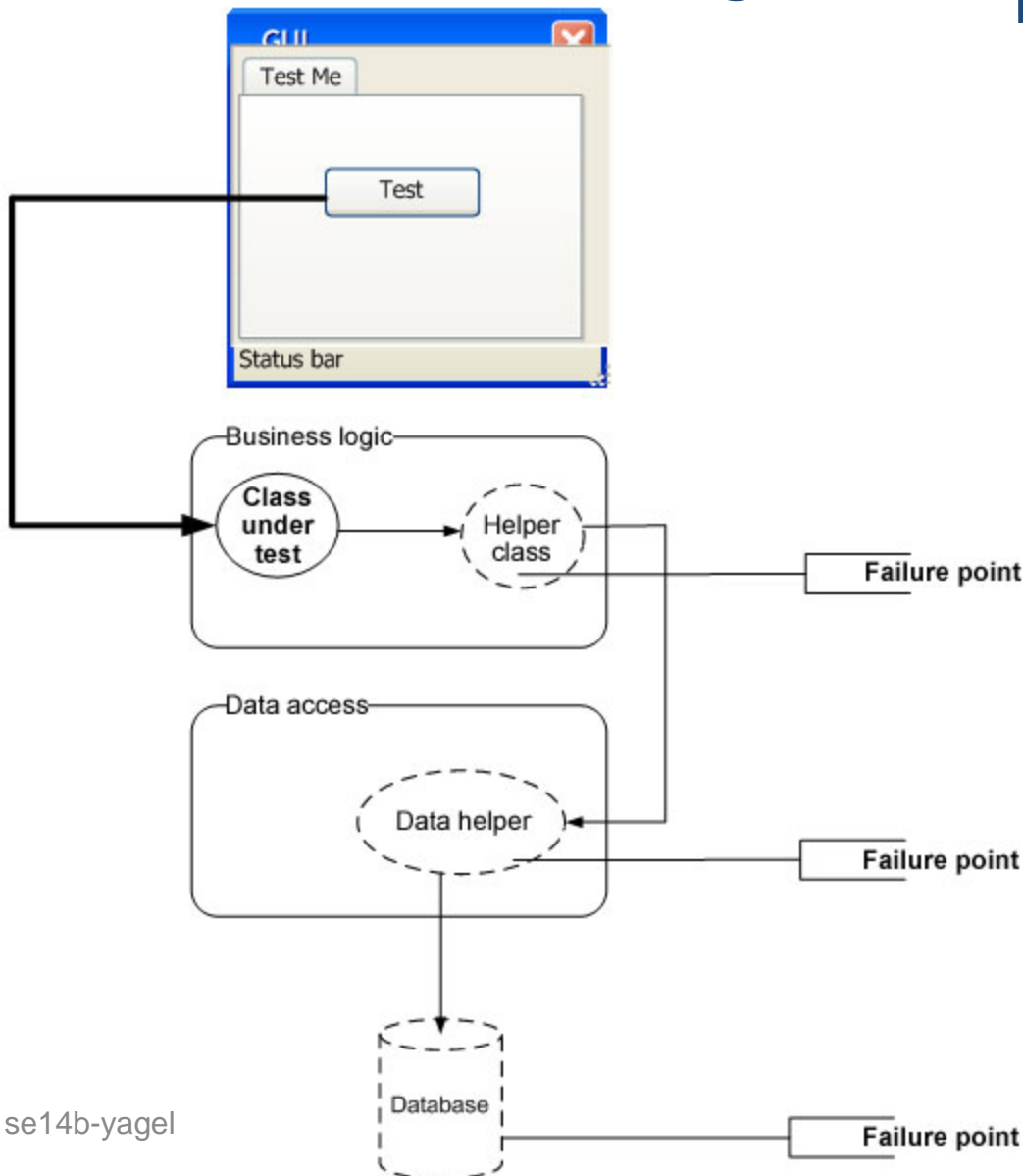
- אולי זו בדיקת אינטגרציה
- שיטה
- ביצוע חוזר
- לכל הקוד

תשתית - Framework ➤



# בדיקת אינטגרציה

- בדיקת מספר רכיבים התלויים אחד בשני ביחד



# בדיקות קצה לקצה (אוטומטיות)

- בשנים האחרונות: Executable Spec. ,BDD ,ATDD  
<http://www.infoq.com/articles/atdd-from-the-trenches>

- 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.ByName("q")).TypeText("WatiN");
        browser.Button(Find.ByName("btnG")).Click();

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

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

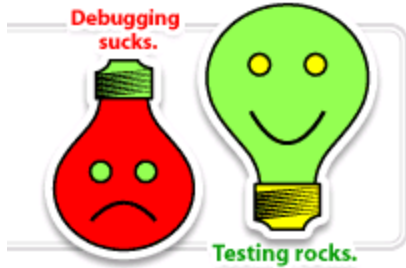
1. ניתן להריץ בדיקות שביצעתי בעבר שוב ושוב  
(רגרסיה)

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
- לא תמיד קל עבור קוד קיים (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

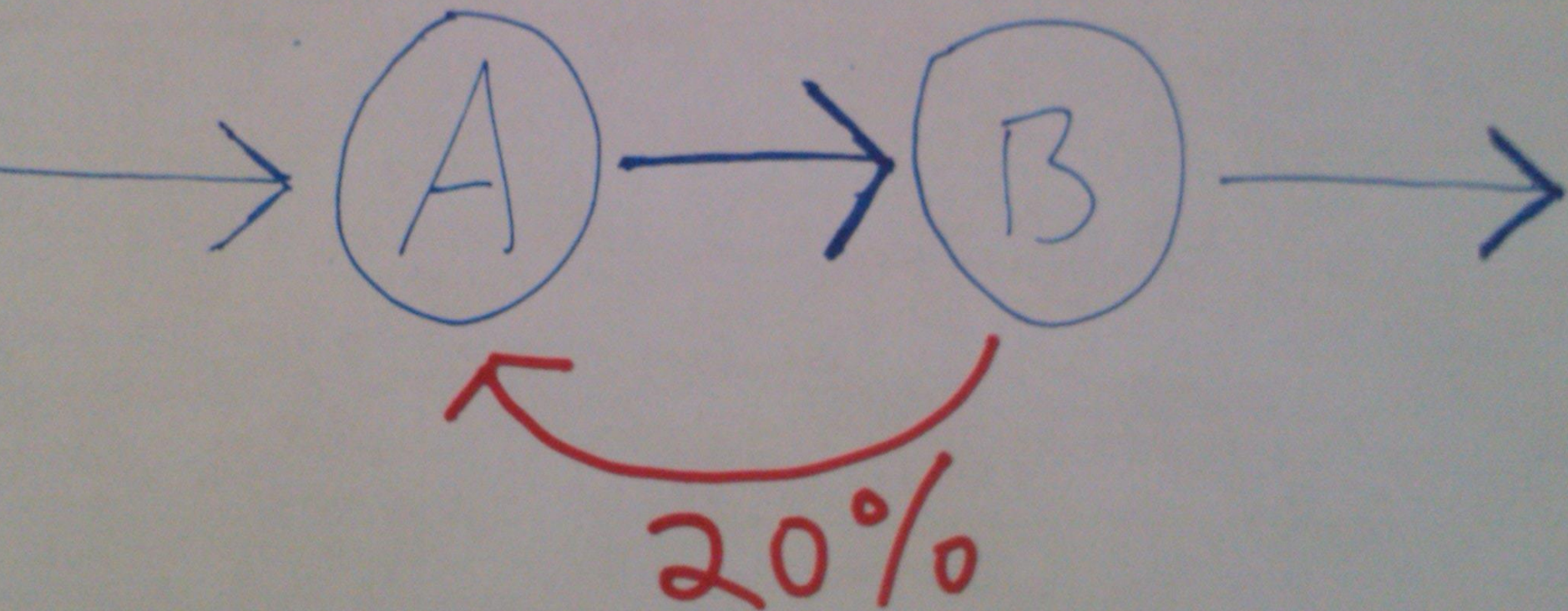
# מתי כותבים?

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

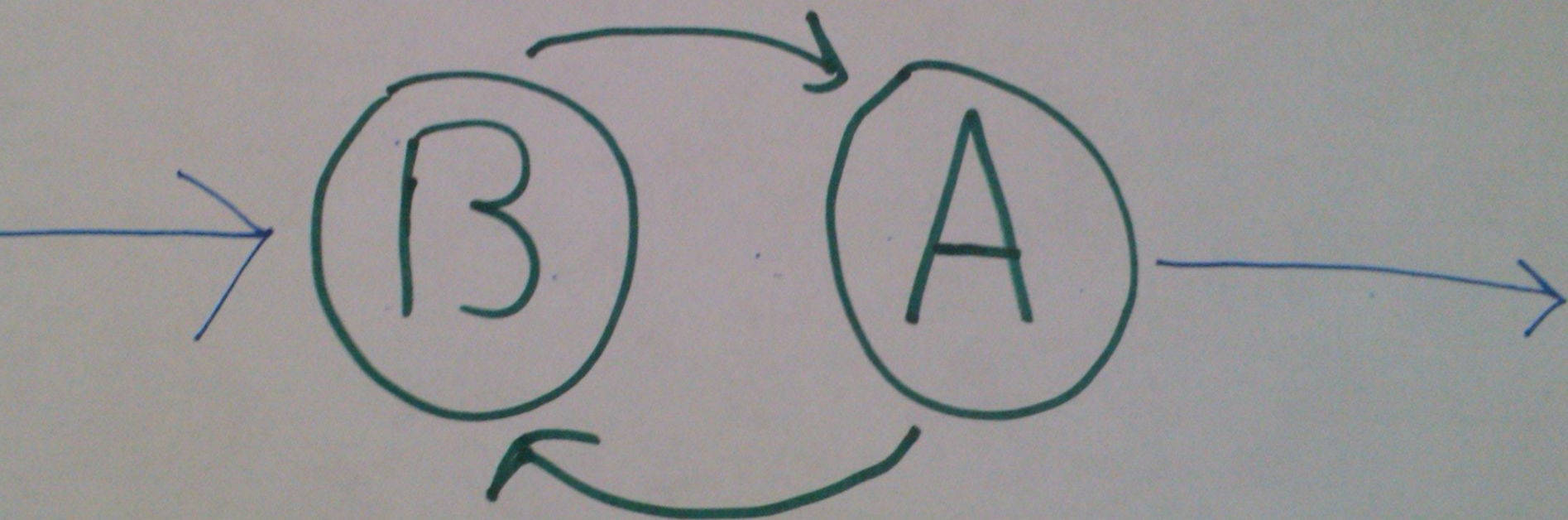


# QUEUING THEORY









# TDD יתרונות

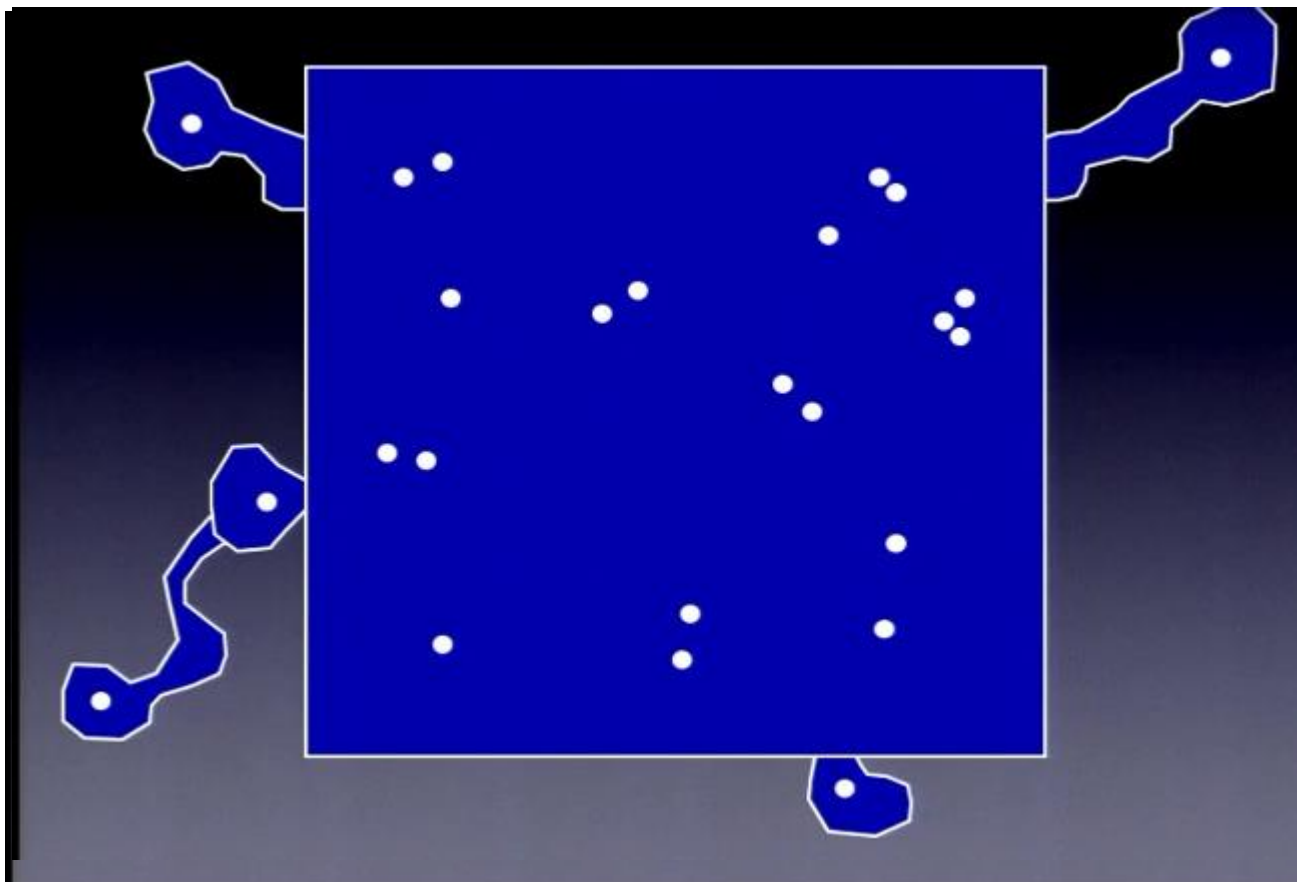
- כיסוי טוב יותר ואוטומטי (ופחות באגים)
- תיכון: Test Driven Design, פשטות, חשיבה כלקוח (ראשוני של הקוד API), התמודדות עם הטיית אישור
- תיכון מתמשך – מודולריות, צמידות נמוכה, 'YAGNI, אפשר שינוי
- דיבאג מוקדם (מה קורה עם משאירים לסוף? אס"ק)
- חסרונות? (לעומת Test After Development)

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

# TDD חסרונות

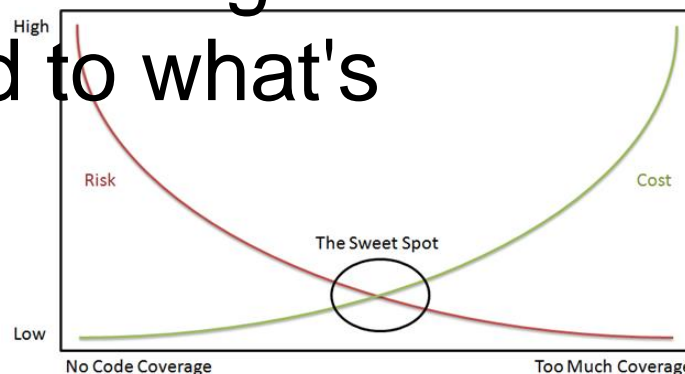
- השקעה בממשק (api) על חשבון פונקציונליות?
- לא מכסה דרישות (ראו BDD)
- עקומת למידה, כולל שיטות משלימות
- מצריך שיתוף פעולה ועבודת צוות
- כמה להשקיע מראש? כיצד מודדים?
- לא מהווה תחליף לחשיבה... (cargo cult programming), "TDD doesn't create design. You do", קשה יותר בבעיות חדשות
- תירוצים לא חסרים...
- Pragmatic not Dogmatic TDD - Agile2012 ראו גם

# תיכון מתמשך



# 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”

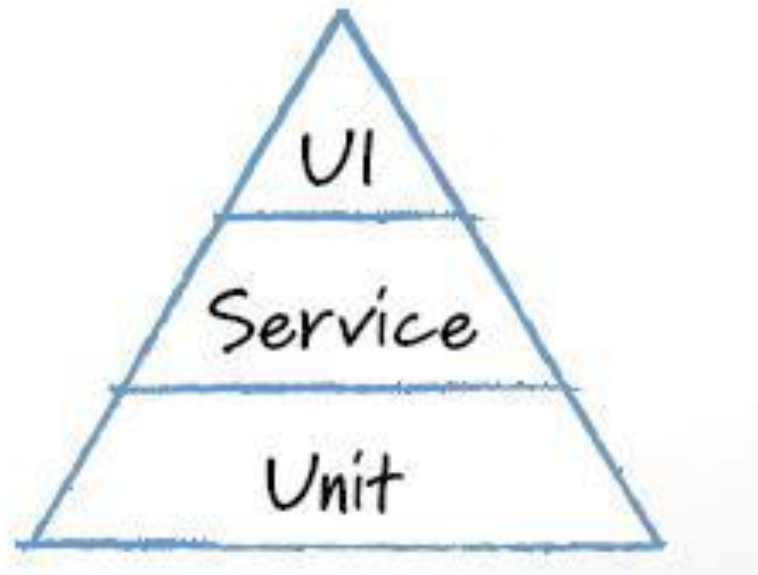




# The Forgotten Layer of the Test Automation Pyramid (also)

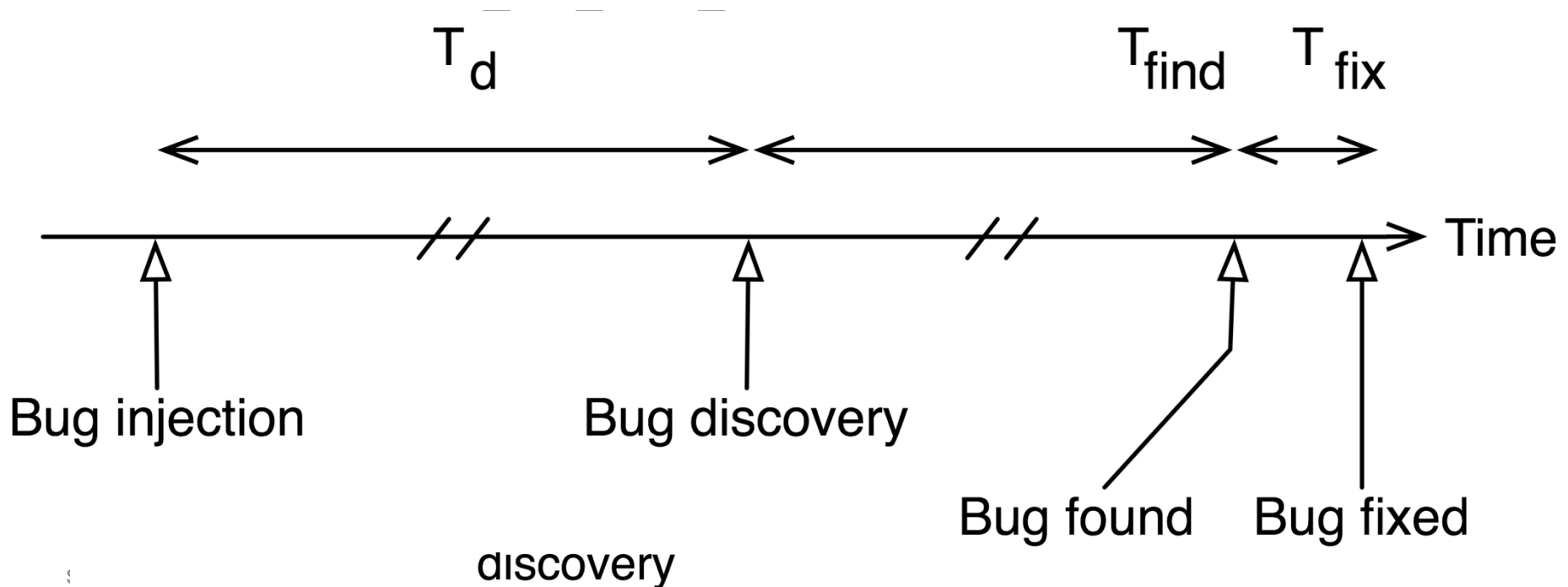
Google:

small,  
medium and  
large



# האם זה משתלם?

- [Physics of Test Driven Development](#) (min. feedback)
- [How test-driven development works](#) (queuing)
- Heins, [BDD in 5 minutes](#) (video)



# ROI for Selected Practices

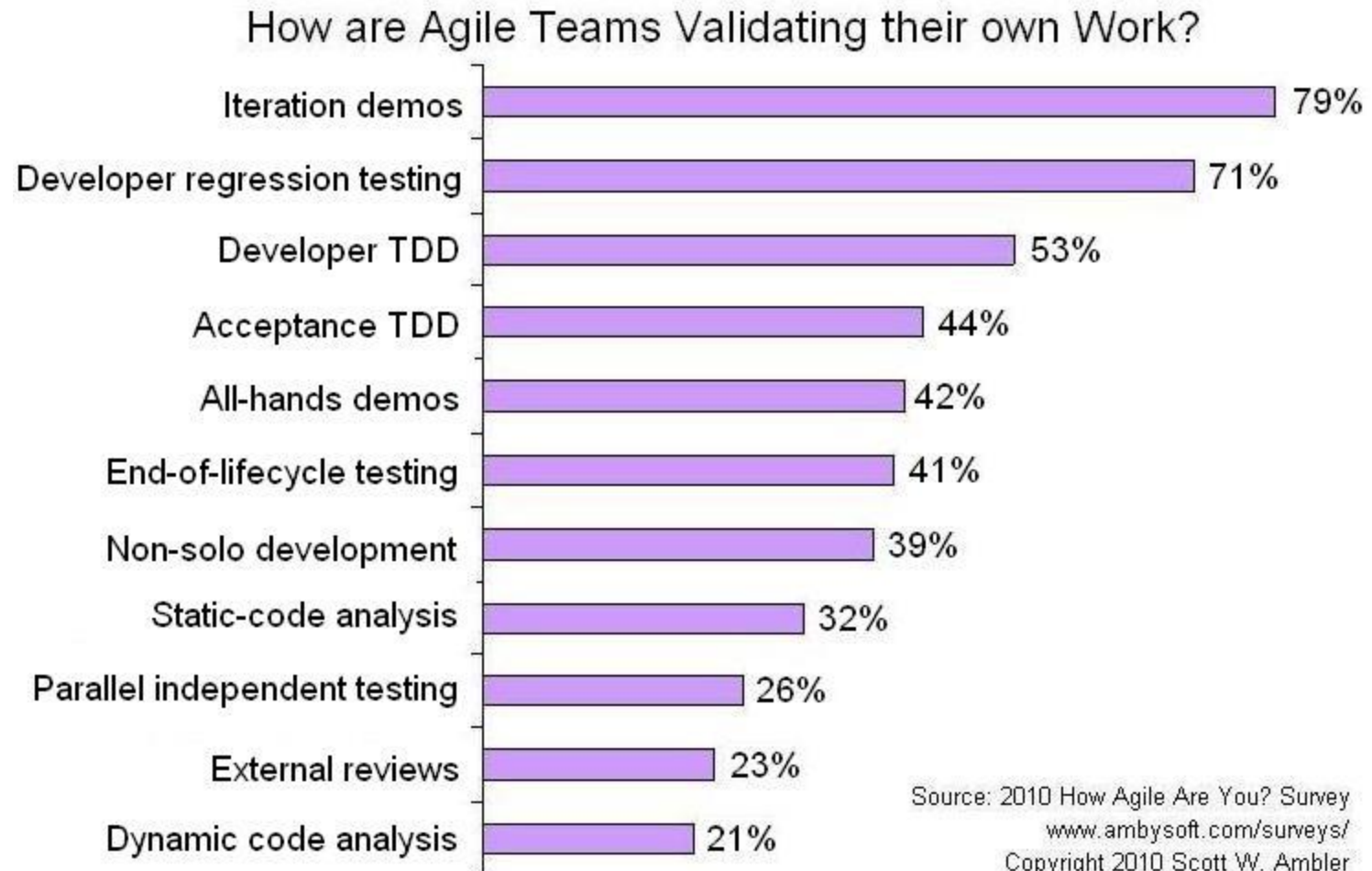
Practice	12-month ROI	36-month ROI
Test Driven Development	-	1000%+
PSP/TSP	-	800%
Formal Inspections	250%	600%+
Productivity Measurement	150%	600%
Process Assessments	150%	600%
Management Training	115%	550%
Scrum	-	500%
Process Improvement Program	-	500%
Technical Staff Training	90%	500%

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/>

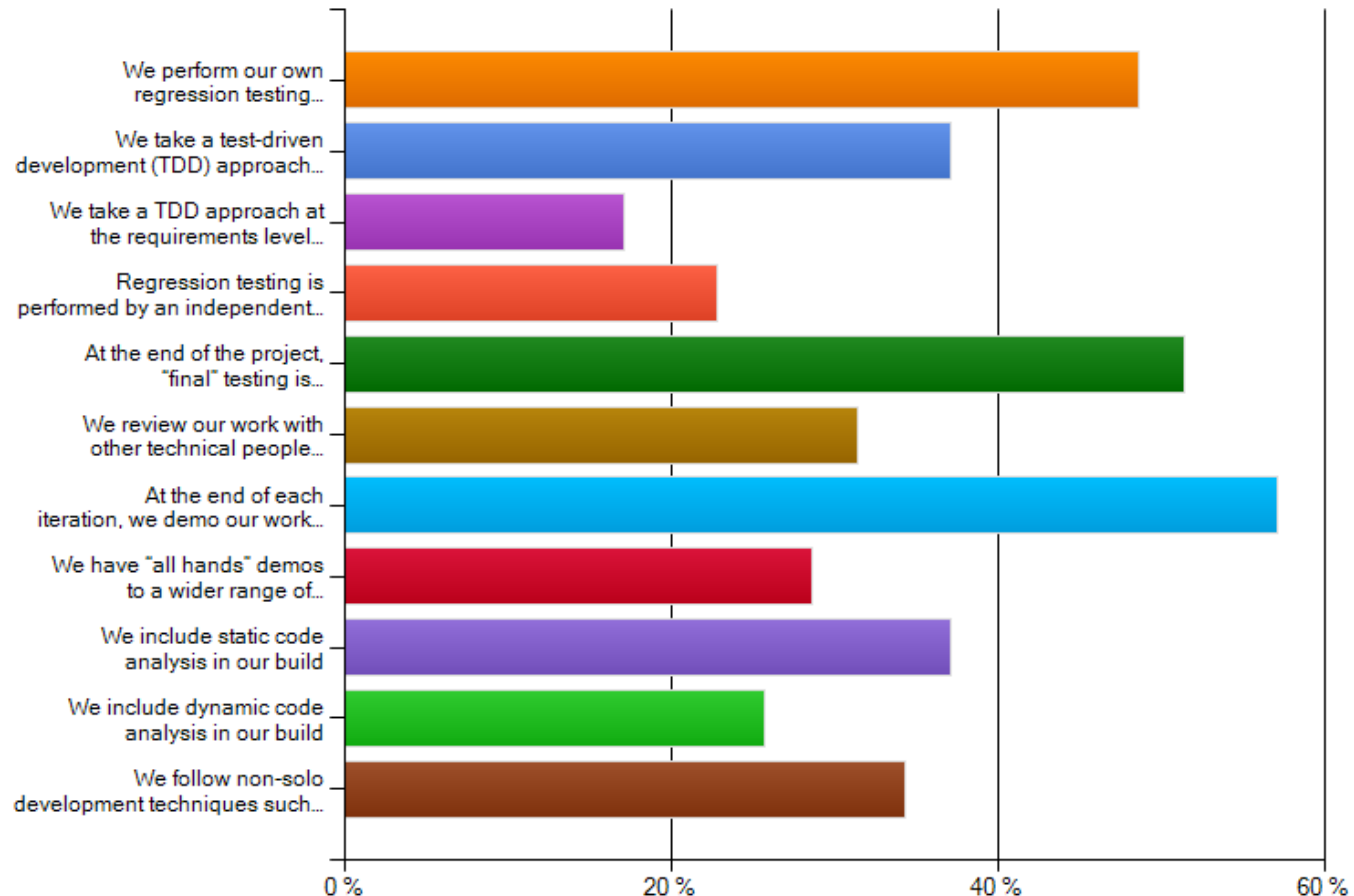
# How Agile Are You? 2010 Survey



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).**

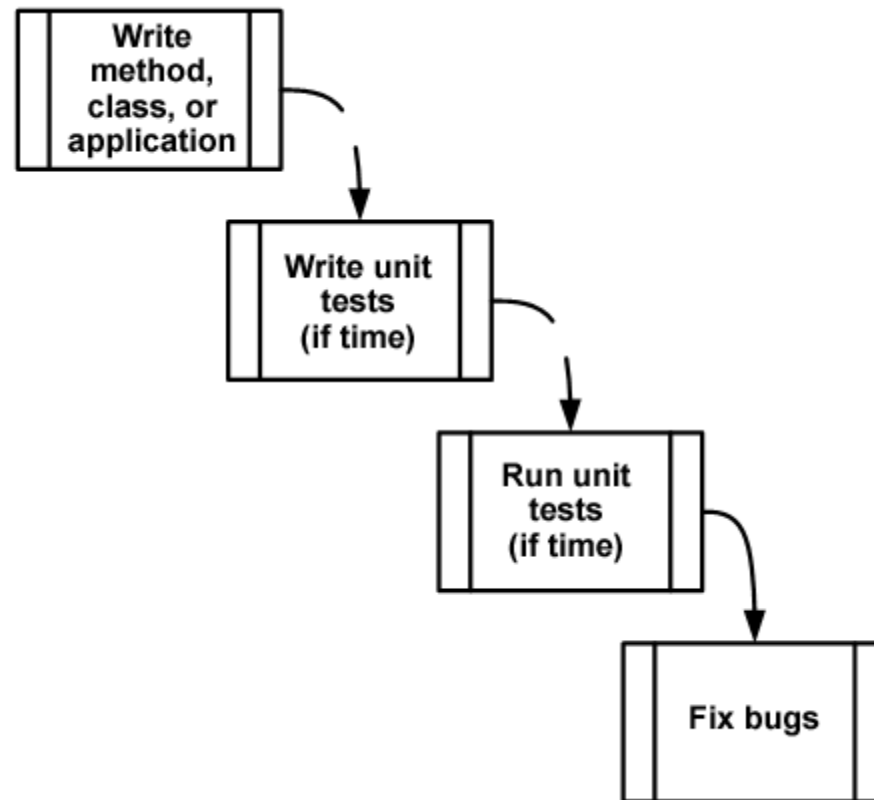


# מצד שני

- “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](#) (also [Coplien](#))
- 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](#))

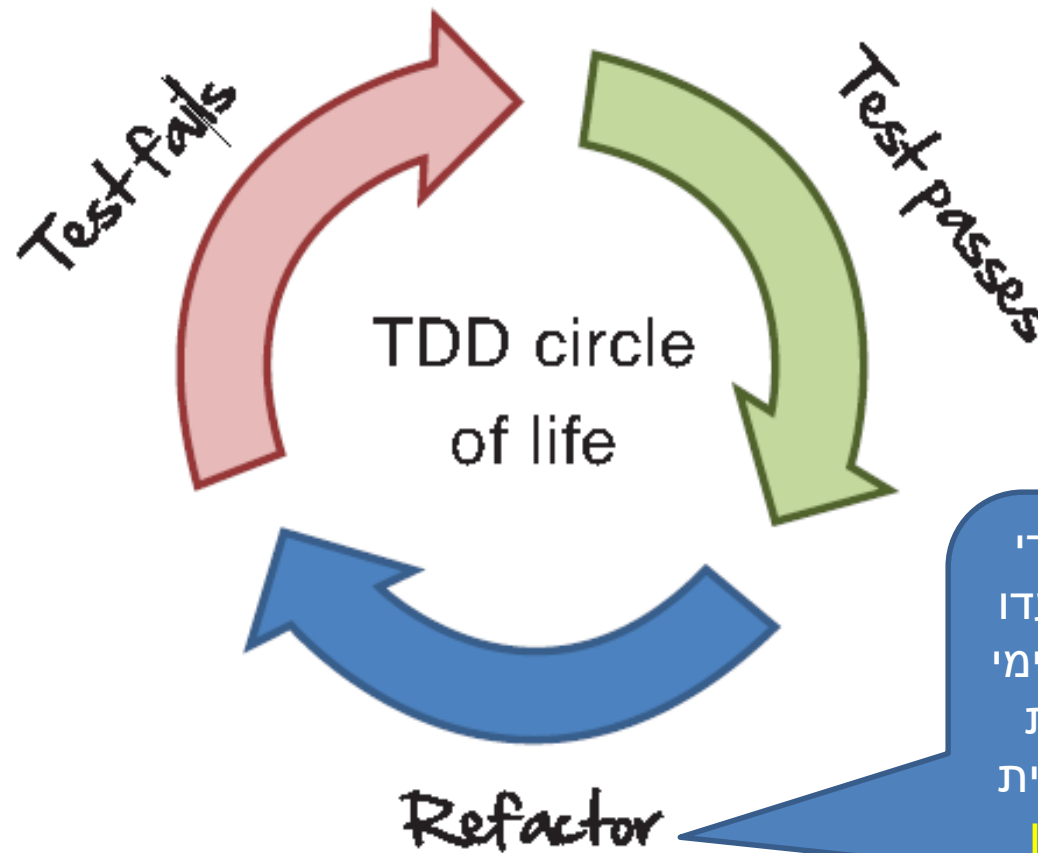
– רגרסיה: הוספת בדיקות כשמתגלים באגים + הרצה תקופתית לראות שמה שידוע שעבד עדיין עובד

# הדרך המסורתית לבדיקות (יחידה)

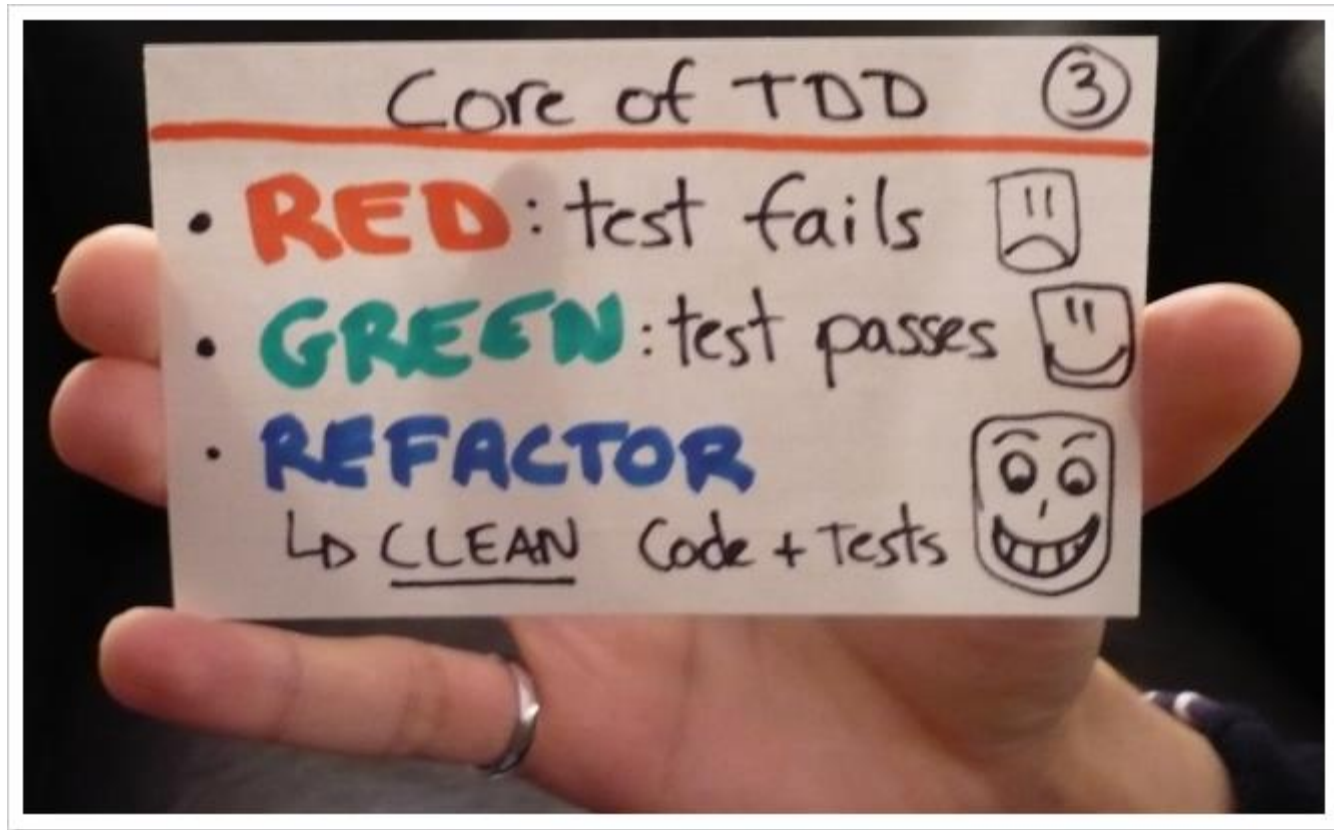




# TDD = TFD + Refactoring



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



# TDD – R. Martin

Over the years I have come to describe Test Driven Development in terms of three simple rules. They are:

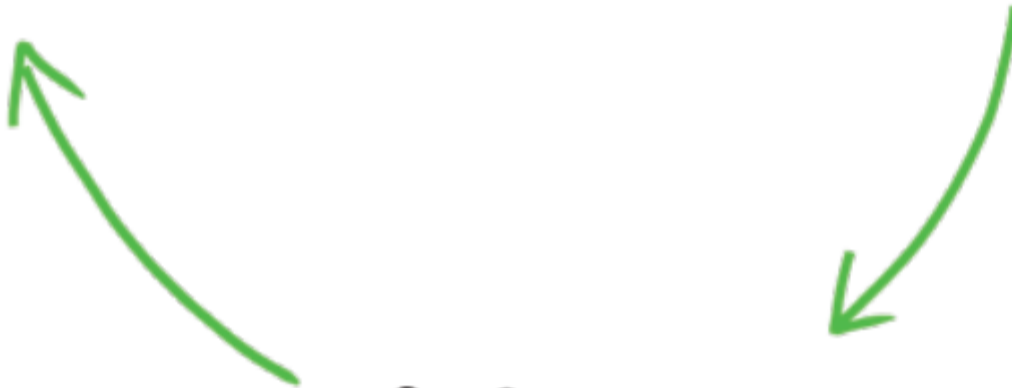
1. You are not allowed to write any **production code** unless it is to make a failing unit test **pass**.
2. You are not allowed to write any more of a **unit test** than is **sufficient to fail**; and compilation failures are failures.
3. You are not allowed to write any more **production code** than is **sufficient to pass** the one failing unit test.

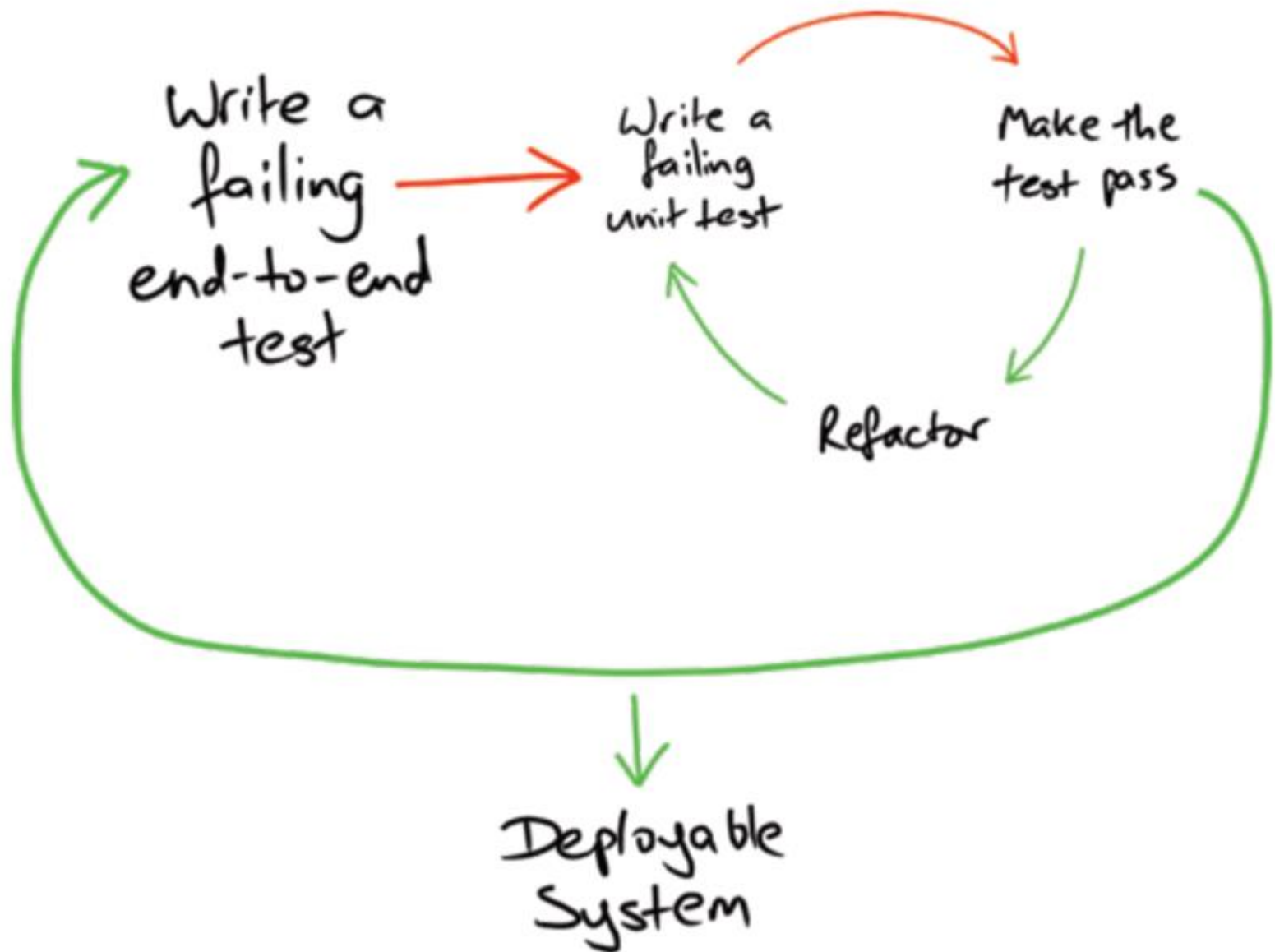


Write a  
failing  
test

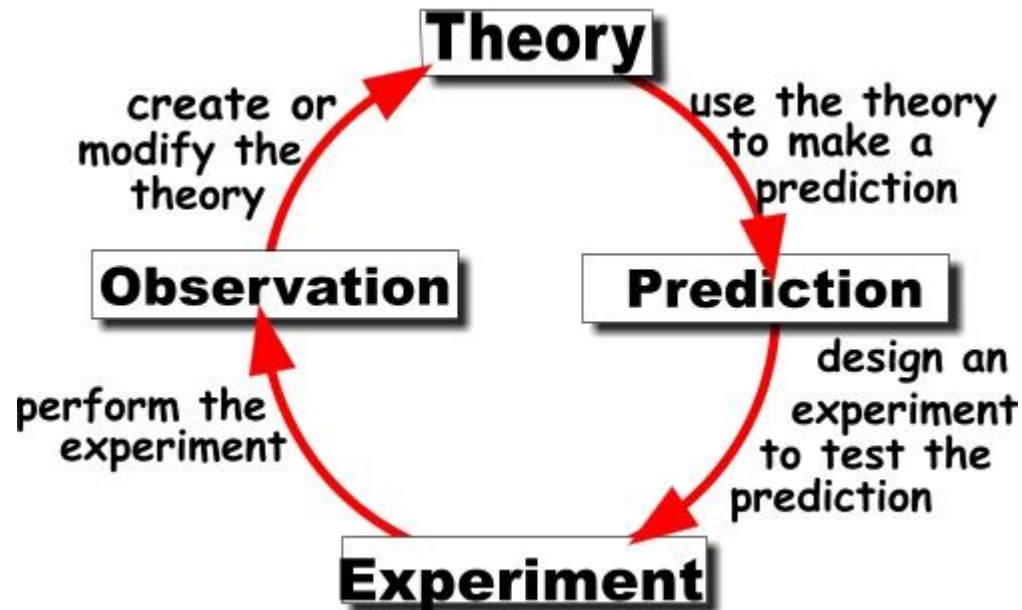
Make the  
test pass

Refactor





# 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)



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")

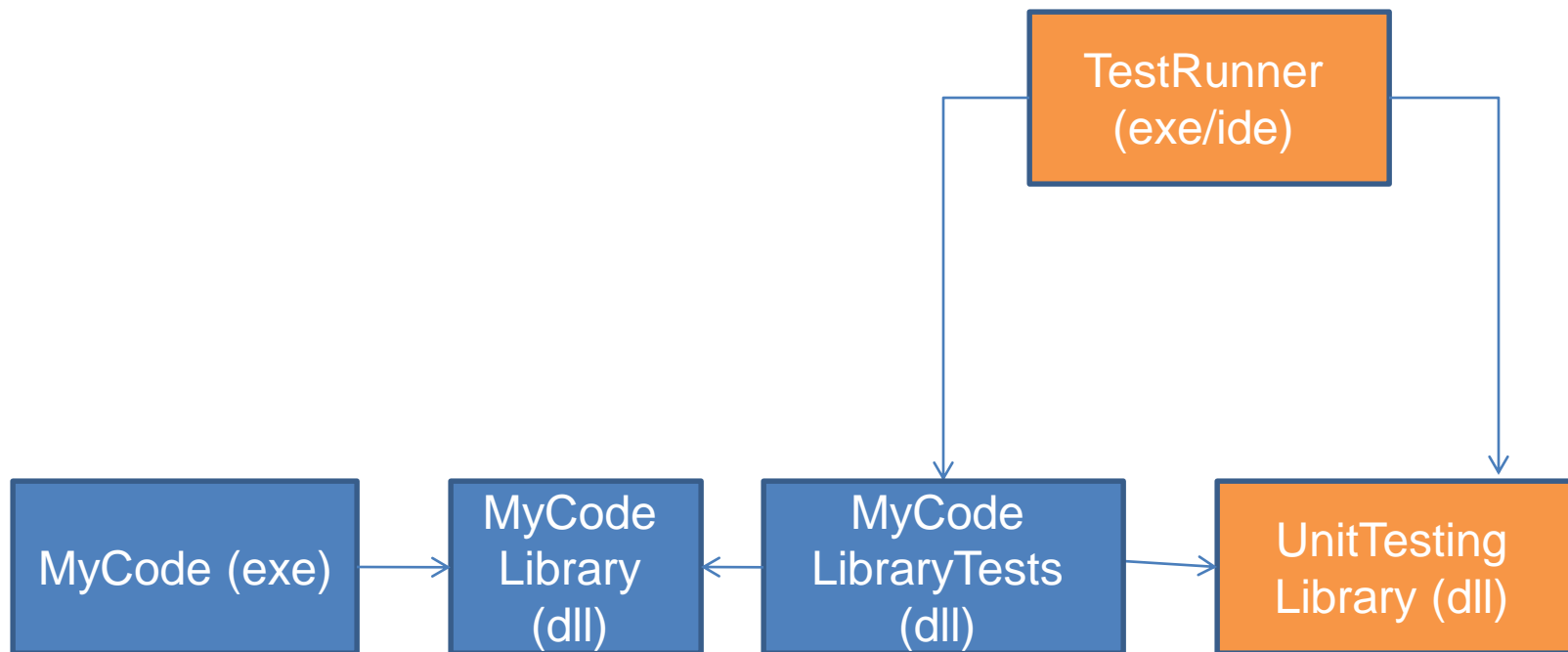
- ייצוא לשפות רבות: CppUnit, PyUnit ועוד

- <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

# רכיבים מקובלים



# רכיבים עיקריים בקוד בדיקה (JUnit)

// SUT

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

// 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);  
    }  
}
```

# הדגמה ראשונית

**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:

Package:

Name:

Superclass:

Which method stubs would you like to create?

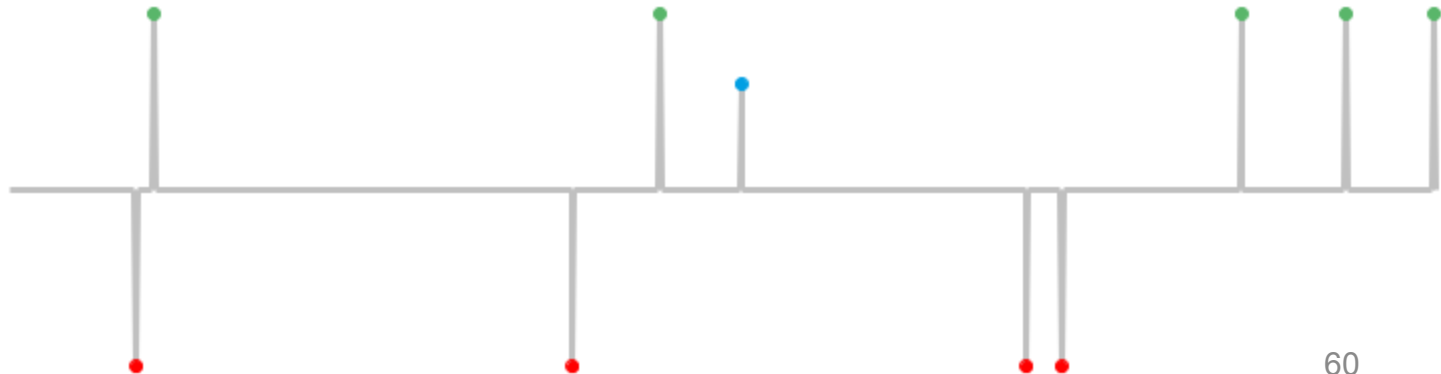
☐ setUpBeforeClass() ☒ tearDownAfterClass()  
☐ setUp() ☐ tearDown()  
☐ \_constructor

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

Class under test:

# כלים - Java

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



# Simple TDD

- FizzBuzz with Junit
- (nice TDD and Junit intro [slides](#))



# TDD Coding Kata

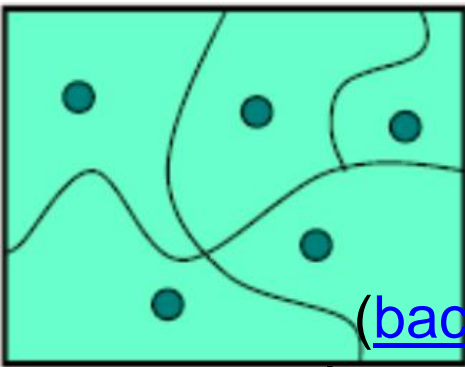
- String Calculator Kata
  - <http://www.osherove.com/tdd-kata-1/>
  - <http://www.21apps.com/agile/tdd-kata-by-example-video/>
- Many Others:
  - <https://github.com/garora/TDD-Katas>
  - Advanced: [GildedRose Kata](#) ([screencast](#))
- Yours?



בפעם הבאה

# RPN Calculator Demo

# בהרצאת המשך \ נושאים מתקדמים



- בדיקות יחידה 2.0 למשל...

- בדיקות קבלה
- מאפיינים מתקדמים של xUnit: אתחולים, חריגות,
- סביבות שונות (.Net), קוד פתוח (למשל ב [backbone.js](https://backbone.js))
- מחלקות שקילות, קופסא שחורה/לבנה, פרמטרים, כיסוי, תלות, אינטראקציה עם רכיבים אחרים, התנהגות מול מצב ([Google ToT](https://www.google.com/search?q=Google+ToT))
- כלים נוספים, אוטומציה, Continuous Integration
- בדיקות לניידים \ ענן \ רשת \ UI וכו'
- כיצד למצוא את [הבדיקה הבאה](#)
- בדיקות לקוד קיים...

- משימה אישית: TDD (עמוד הבא)

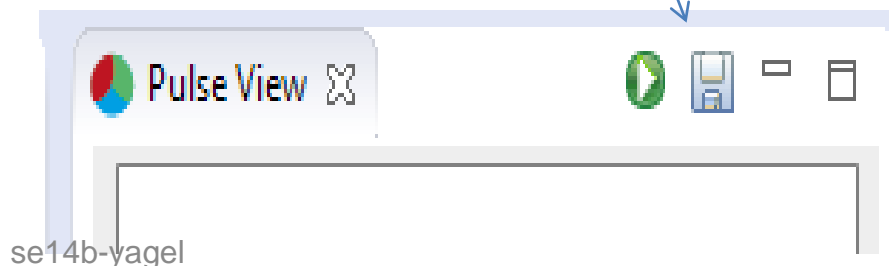
- קריאה מומלצת להרצאת המשך:  
[Using Mock Objects](#)



It's easier to ask forgiveness than it is to get permission.  
se14b-yagel

# משימה אישית 4 – בדיקות יחידה + TDD

- [FizzBuzz](#) או רועי אושרוב, [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 (עד להרצאה הבאה, בנוס: שיתוף pulse ו-TDGotchi בפורום)
- ציון: כיסוי, RGR, נכונות, איכות.
- אופציה: זוגות מתחלפים (אחד בודק אחד מממש [pairhero](#), כולל ב-commits) – אופציה: ?hangout



# לסיכום

“I get paid for code that works, not for tests, so my philosophy is to test as little as possible to reach a given level of confidence” –Kent Beck ([stackoverflow](https://stackoverflow.com/questions/102447/why-does-kent-beck-test-so-little))

- בדיקות, בדיקות יחידה ופיתוח מונחה בדיקות
- בפרויקט

– בכל סבב: ניסוח בדיקת קבלת לתרחיש עיקרי

– סבב 2: חליפת בדיקות לרכיב מרכזי (וכתמיכה בשלד המוצר)

“The project was a miserable failure because we let the tests we wrote do more harm than good” - [osherove](https://oshero.com/2012/01/20/when-tests-do-more-harm-than-good/)

- בדיקות ותיכון מתמשך
- Red-Green-Refactor
- xUnit + כלים
- מצריך לימוד מתמשך – אז למה עכשיו?

- R. Martin: “Specification, not Verification”
- John Gall: “A complex system that works is invariably found to have evolved from a simple system that works.”