



**ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT**

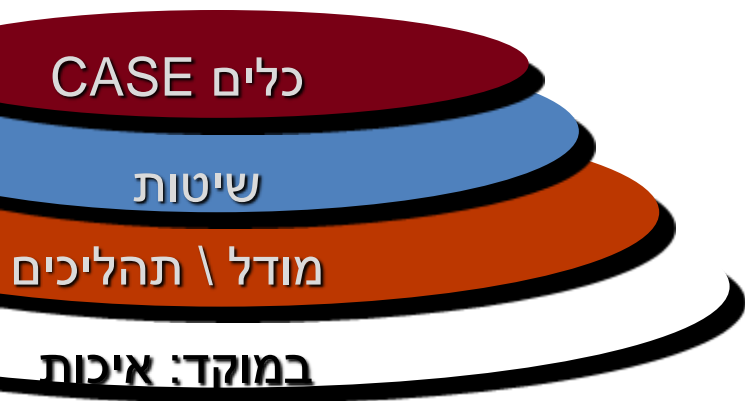
CODESMACK

הנדסת תוכנה

7. פיתוח מונחה מפרטים - I

בדיקות יחידה

[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 (למשל JUnit)
- הדגמה
- הרצאה 3: מעבדת בדיקות יחידה
- פרויקט: המשך עבודה על סבב 1-MVP הצגה, סיכום, רטרוספקטיבה ותכנון לסבב 2
 - פגישות וסקרים
 - משימה אישית: 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

מעט על רטרוספקטיבה

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

– במה הצלחנו?

– היכן היו קשיים?

– מה נרצה להמשיך לעשות?

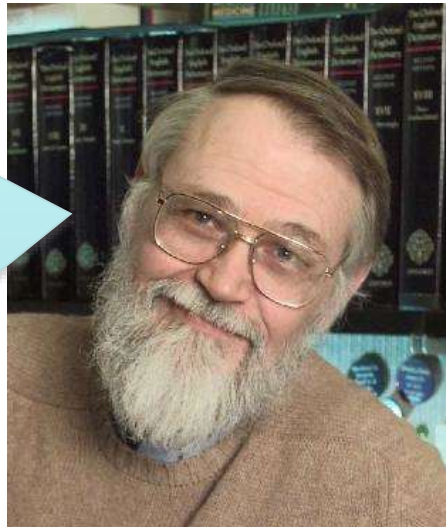
– מה להפסיק? מה לשנות?



- 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.

se13b-yagel



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

Source: ⁷[saas](#)

בדיקות תוכנה

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

Boeing 787 wing break test –

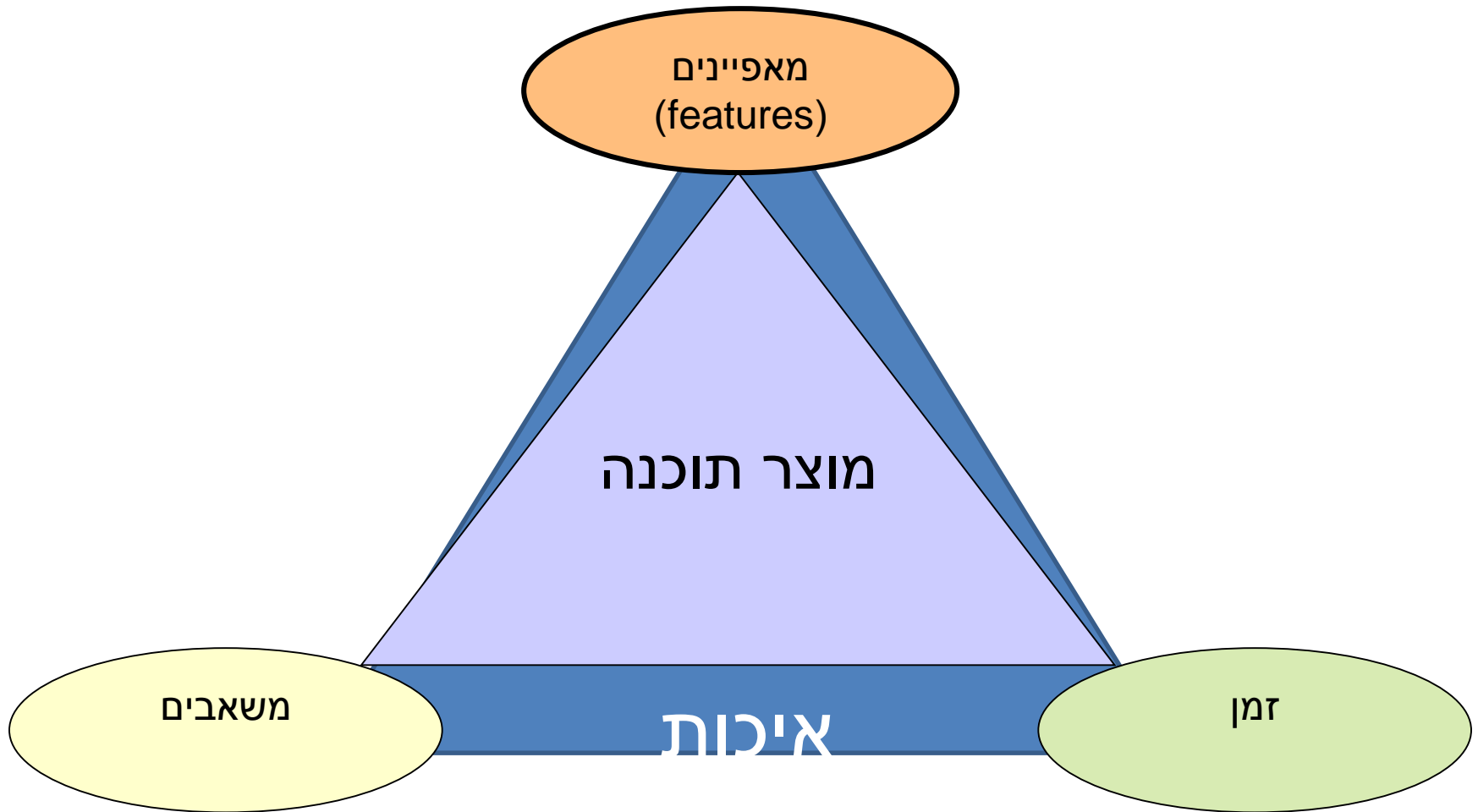
- האם זה באמת כדאי?
- למה עכשיו?

למה לבדוק?

- נכונות \ איכות בכלל
- אפשרות לשינויים
- לישון טוב בלילה

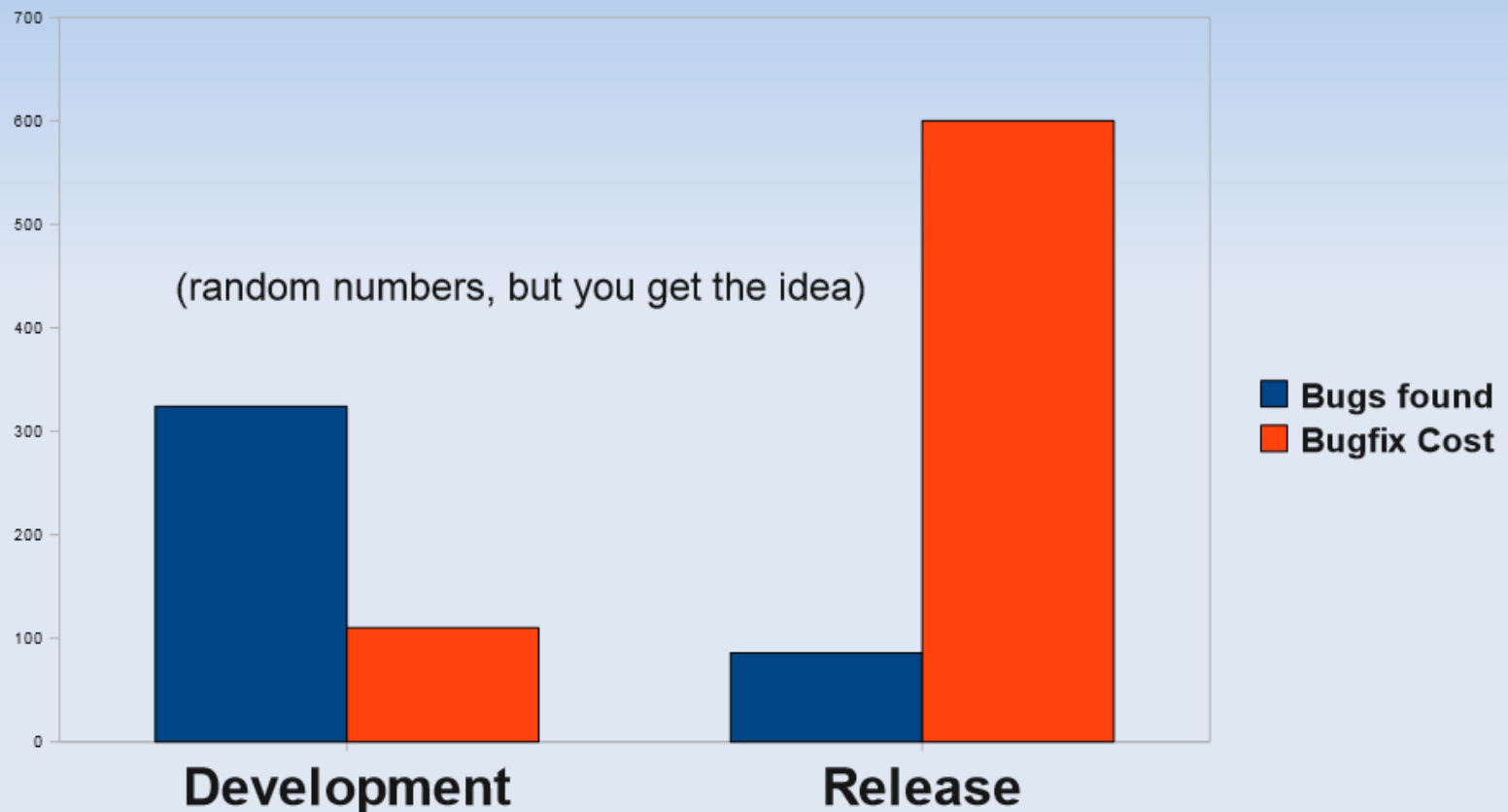


תזכורת: פרויקט תוכנה:

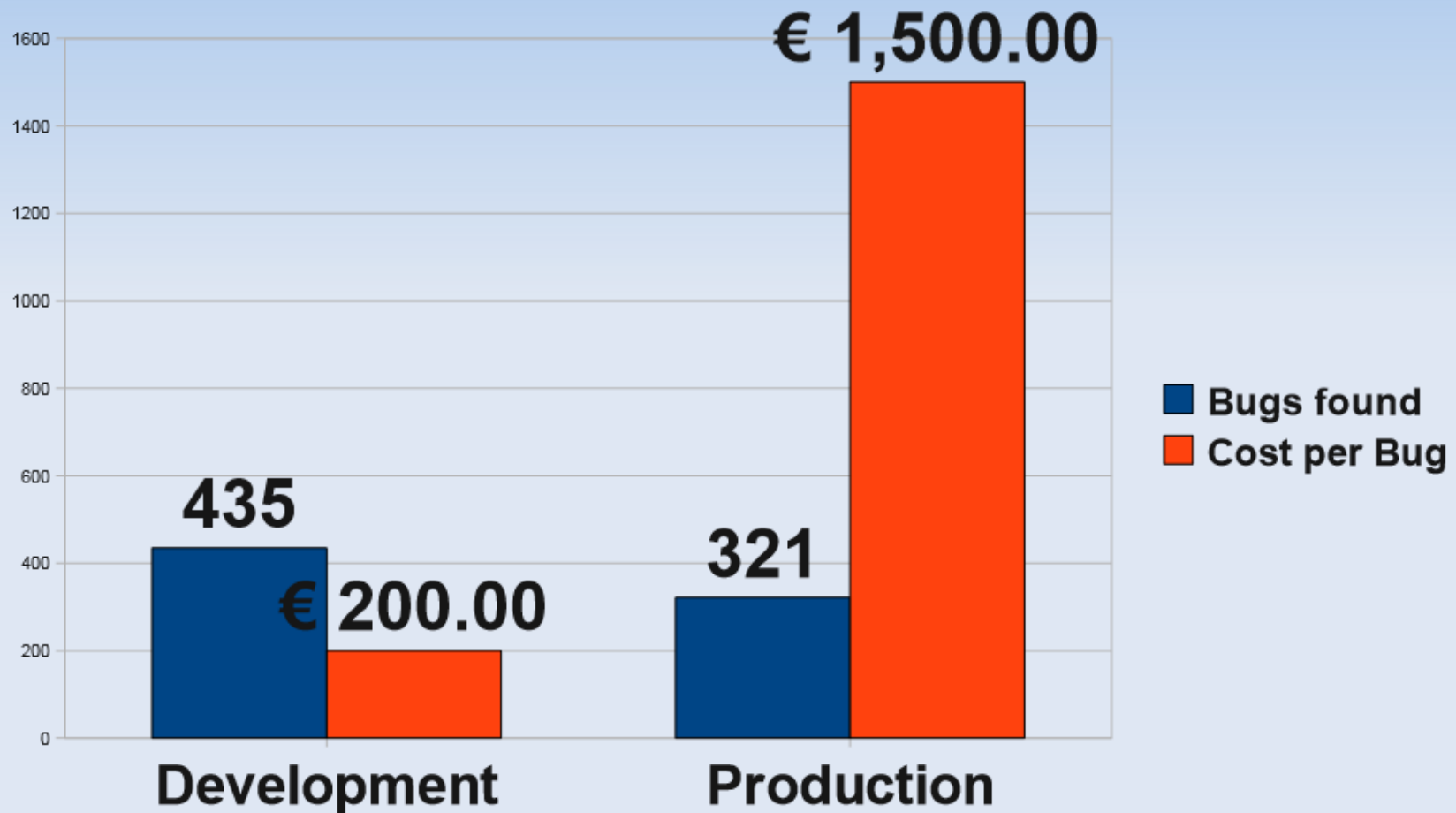


למה לבדוק? (באדום עלות לבאג אחד)

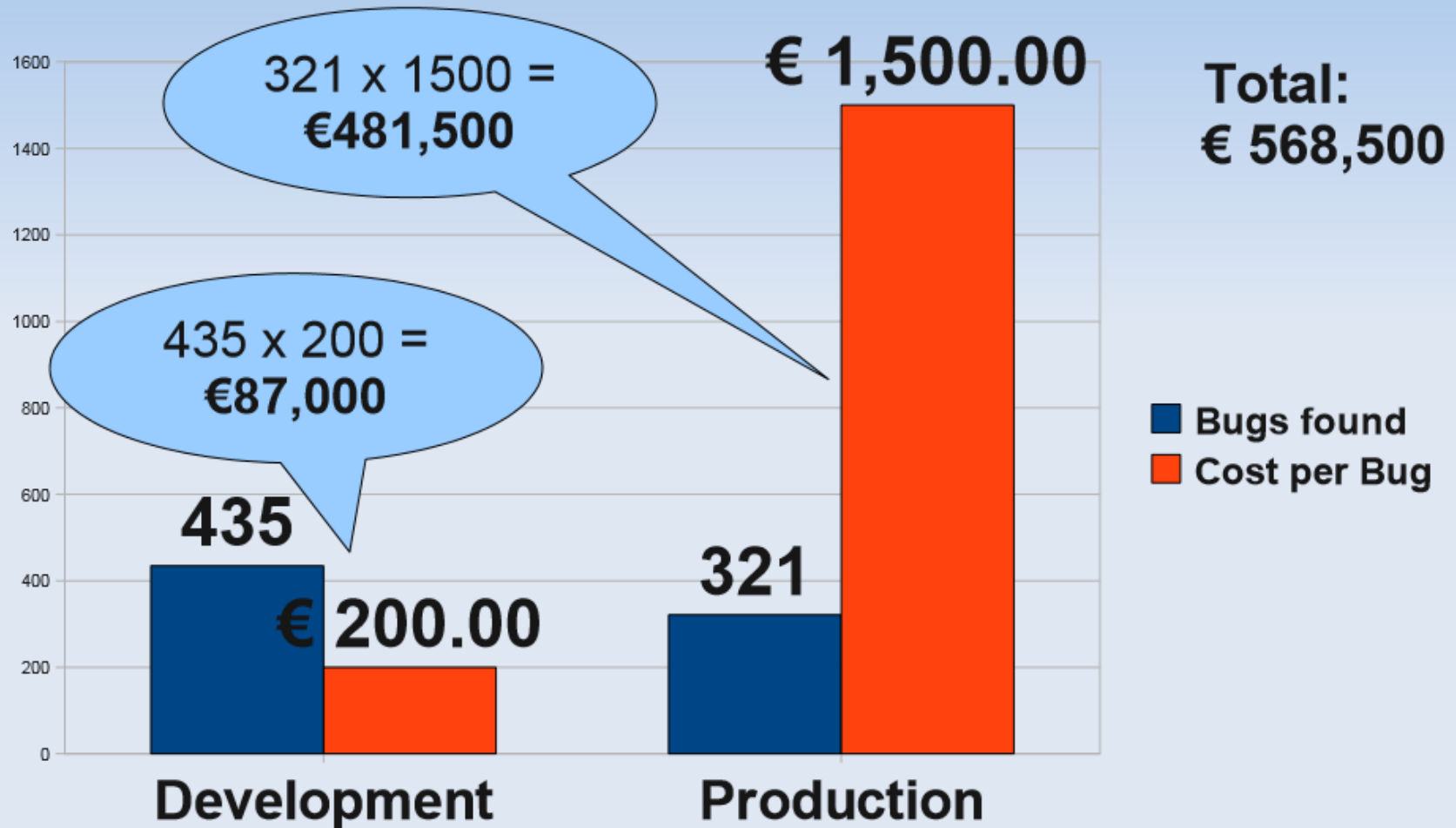
**Simplest case:
Development => Production**



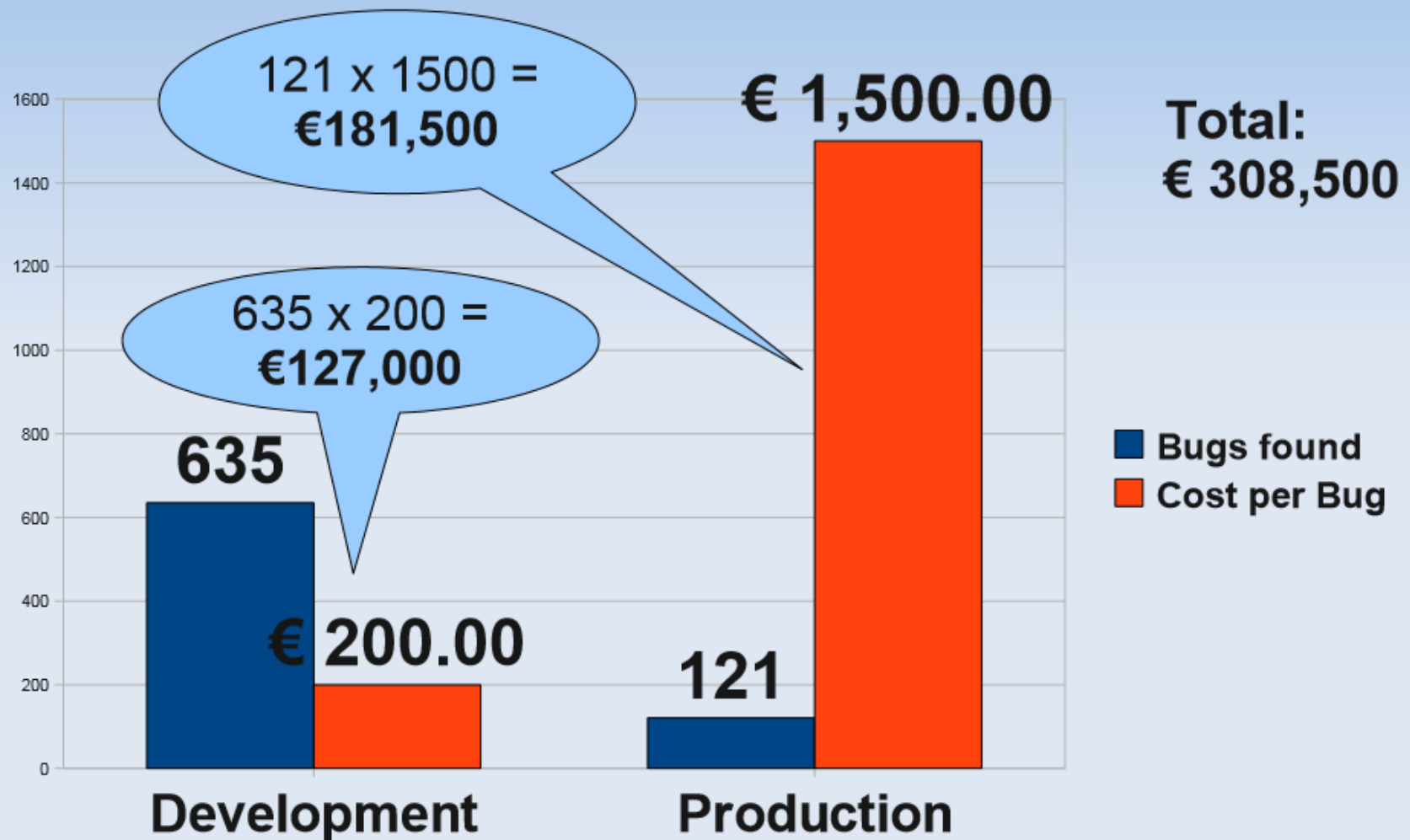
Real world (no testing)



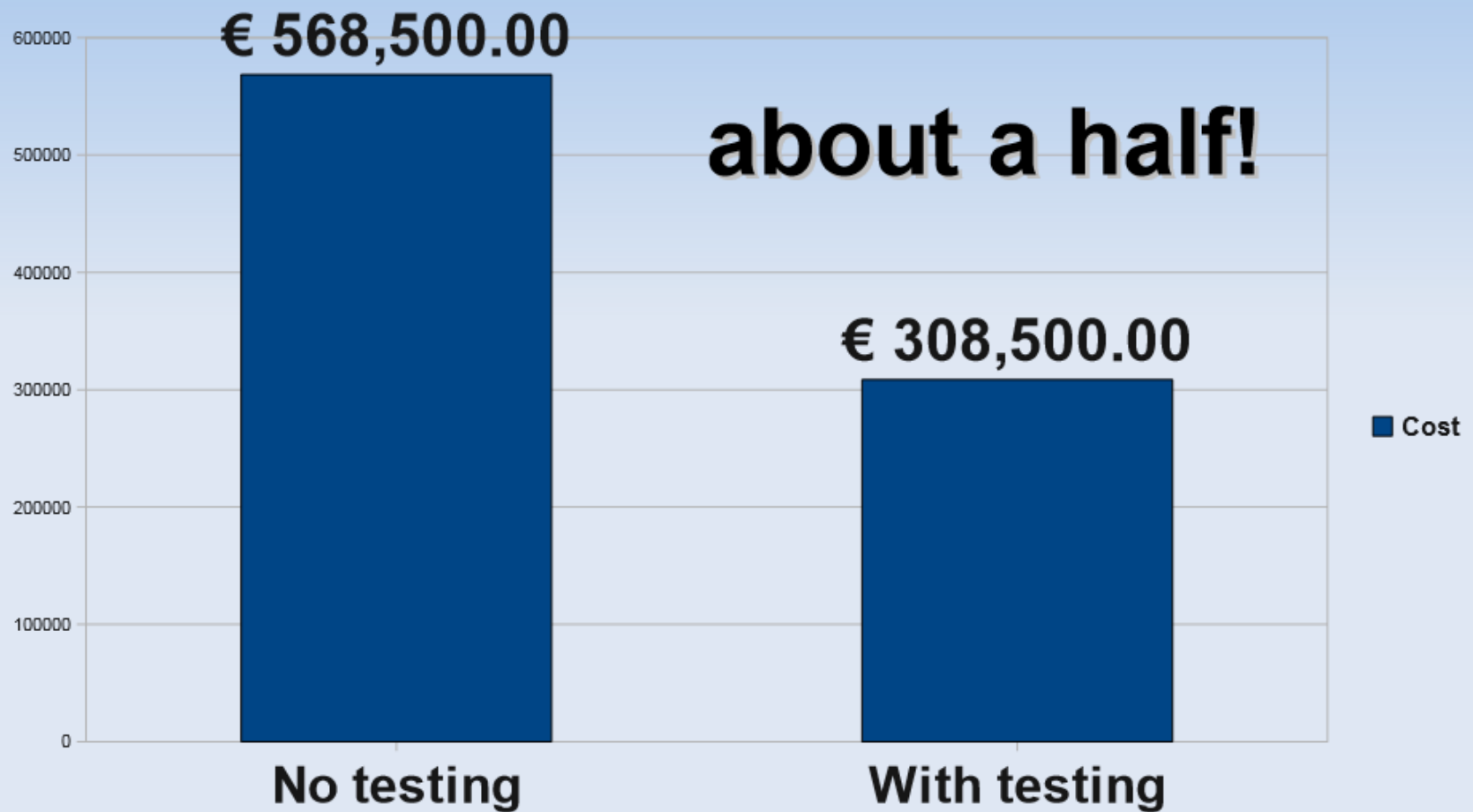
Real world (no testing)



Real world + testing



no testing vs + testing

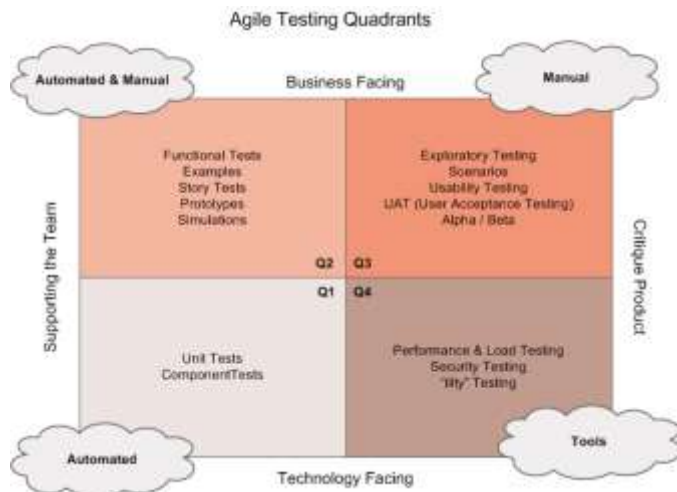


איך אתם בודקים את התוכנה שלכם?

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?
- בדיקות אינטגרציה
 - האם הקוד שכתבנו עובד מול קוד אחר?
- Jbrains, [Integrated Tests are a Scam \(lecture\)](#)
- בדיקות יחידה (מפתח)
 - האם המודולים עושים את הדבר הנכון? נוחים לשימוש? ע"י מי?

בדיקת יחידה

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

האם כדאי להשקיע בזה?

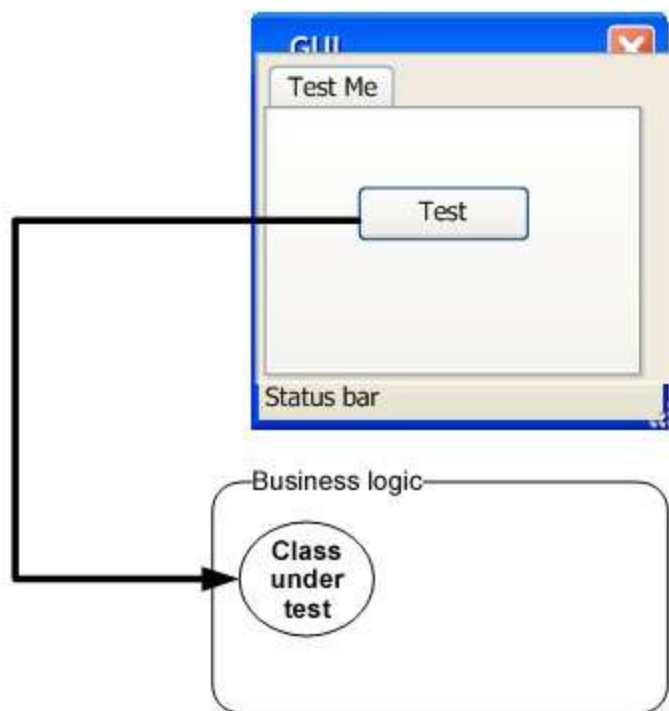
- אולי מספיקות בדיקות אינטגרציה ומערכת?
- זהו קוד שגם מצריך תחזוקה!
- בעצם אולי כבר כתבתם עד היום כאלו דברים

– הדגמה Dog

- מה חסר?

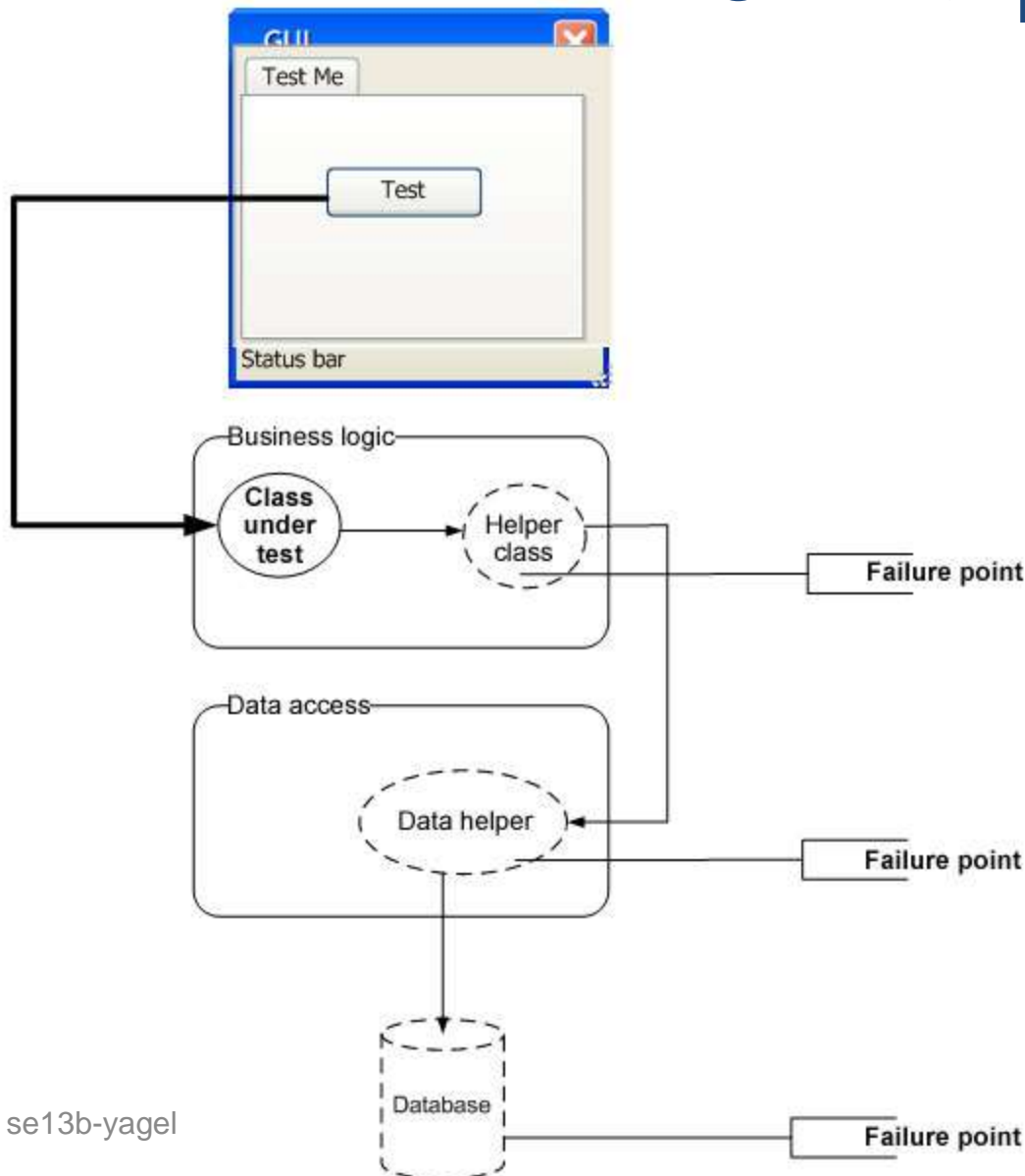
- אולי זו בדיקת אינטגרציה
- שיטה
- ביצוע חוזר
- לכל הקוד

תשתית - Framework ➤



בדיקת אינטגרציה

- בדיקת מספר רכיבים התלויים אחד בשני ביחד



בדיקות קצה לקצה

- 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



- יתרונות

- נכונות (ובמיוחד בשפות דינמיות)
- פחות זמן ב-debugger, רגרסיה
- תיעוד "חי"
- לעומת בדיקות אחרות: קלות ומהירות
- הורדת עלויות \ שינויים ...

- חסרונות

- קוד (תחזוקה, תיכון, בדיקות?)
- זמן (האם בדיקות אינטגרציה ומערכת לא מספיקות?)
- לא תמיד קל עבור קוד קיימים (legacy)

מי בודק?

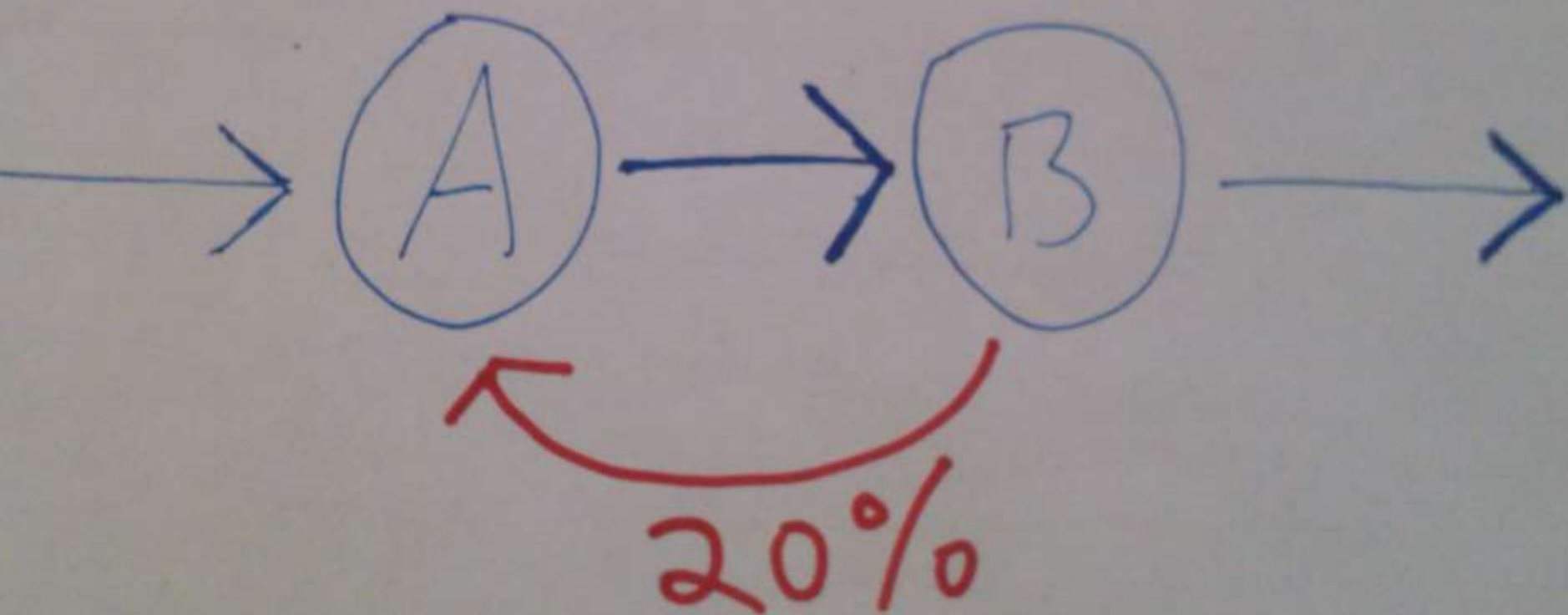
- בודקים או מפתחים?
- המטרה: מוצר בעל-ערך\איכותי
- יחס מפתח:בודק (e.g. Google vs. Microsoft)
 - מה משמעות גודל היחס?
 - Developer in testing
 - Exploratory Testing etc.
 - Why Facebook doesn't have or need testers

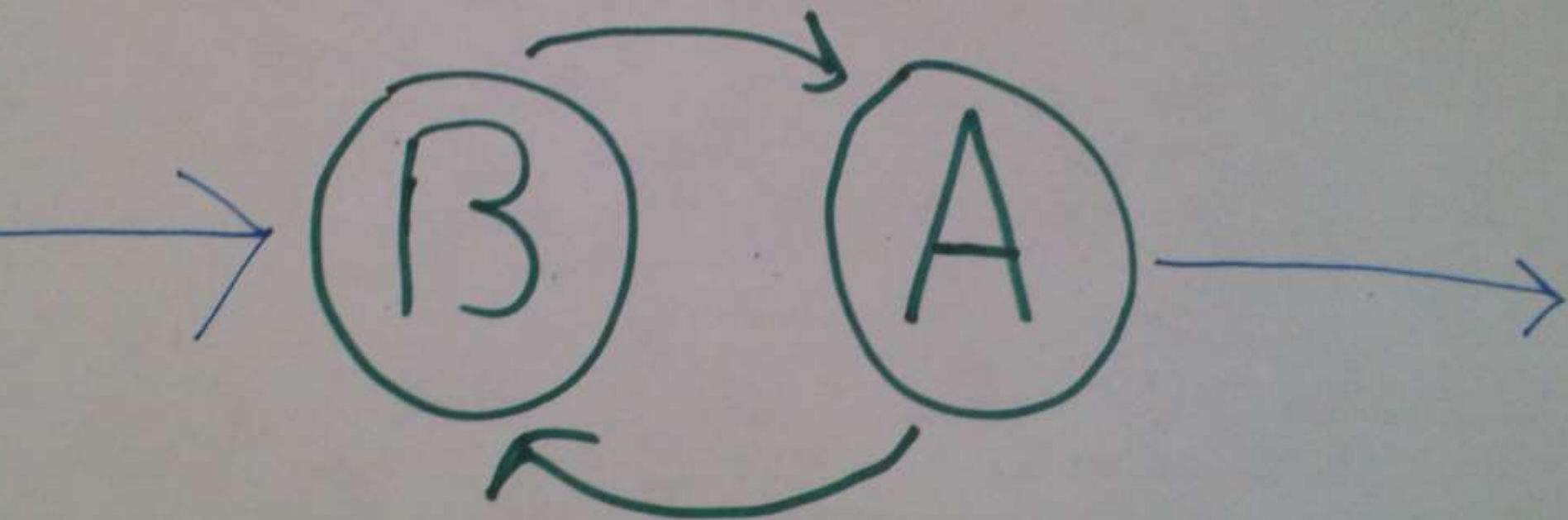
מתי כותבים?

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

QUEUING THEORY







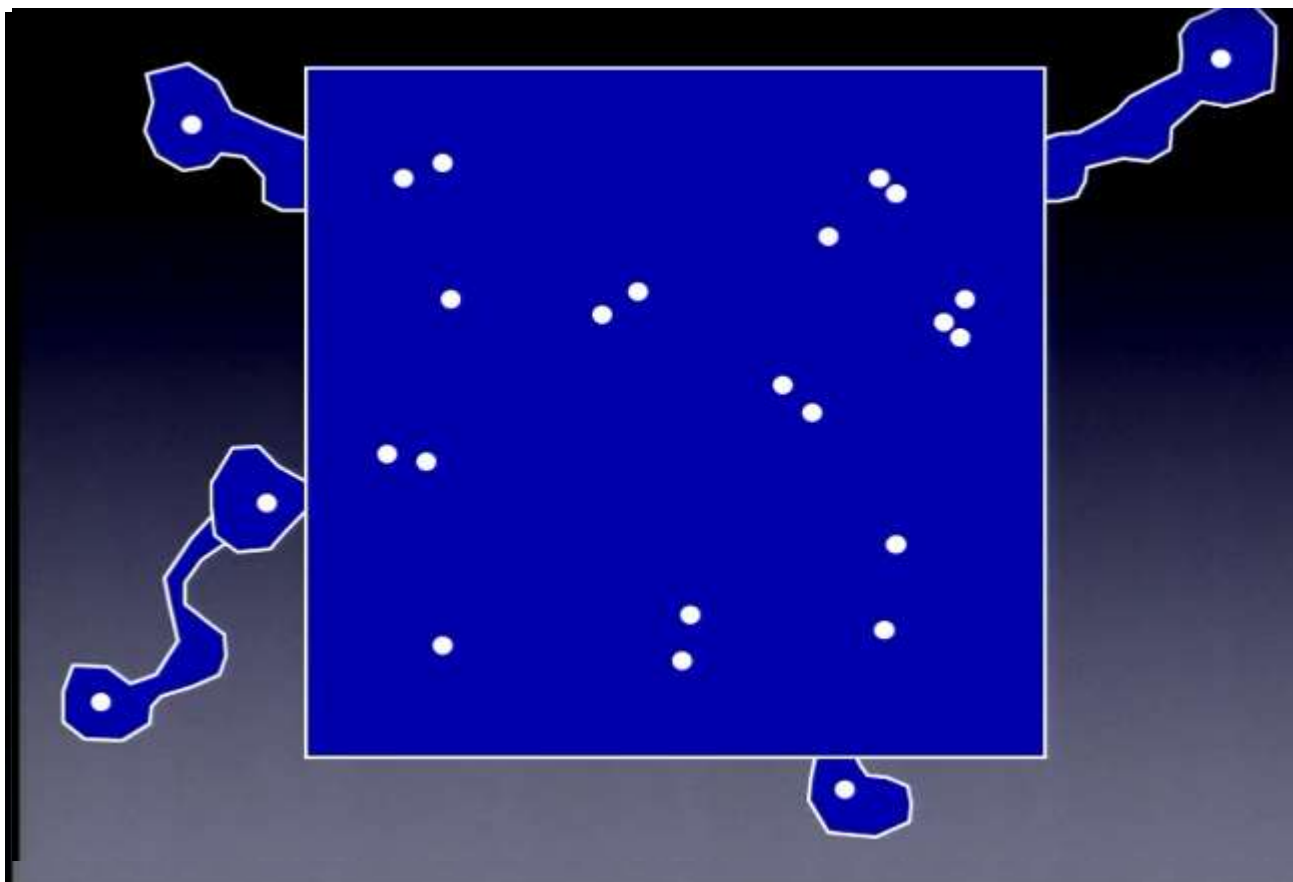
TDD יתרונות

- 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)

TDD חסרונות

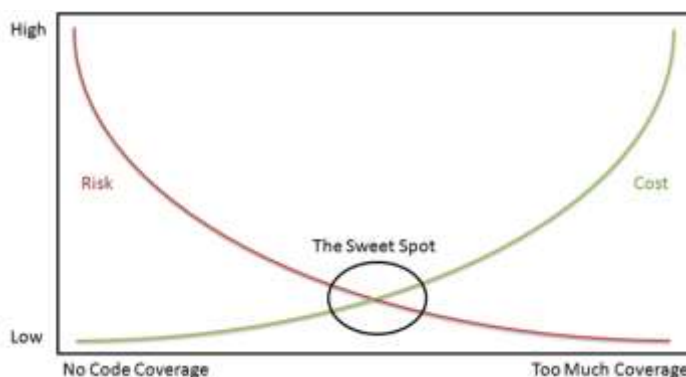
- השקעה בממשק (api) על חשבון פונקציונליות?
- לא מכסה דרישות (ראו BDD)
- עקומת למידה, כולל שיטות משלימות
- מצריך שיתוף פעולה ועבודת צוות
- כמה להשקיע מראש? כיצד מודדים?
- לא מהווה תחליף לחשיבה... (cargo cult programming)
- תירוצים לא חסרים...
- ראו גם Pragmatic not Dogmatic TDD - Agile2012

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!



The Forgotten Layer of the Test Automation Pyramid (also)

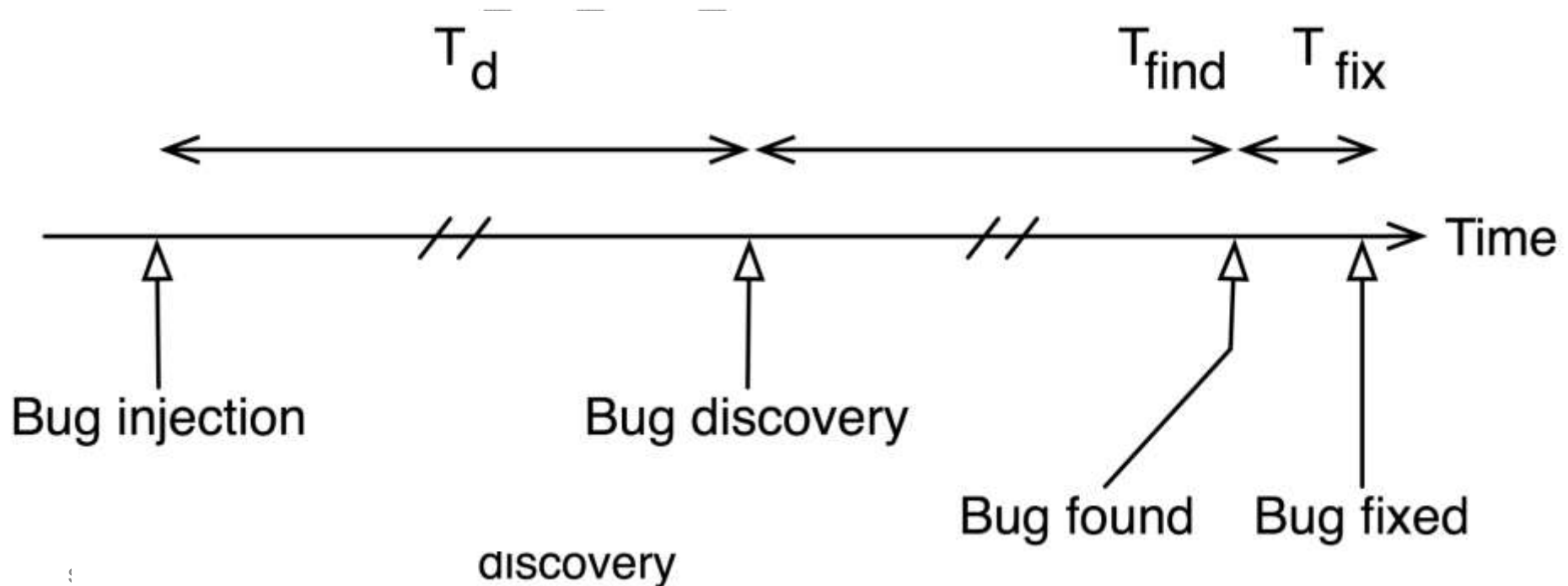
Google:

small,
medium and
large



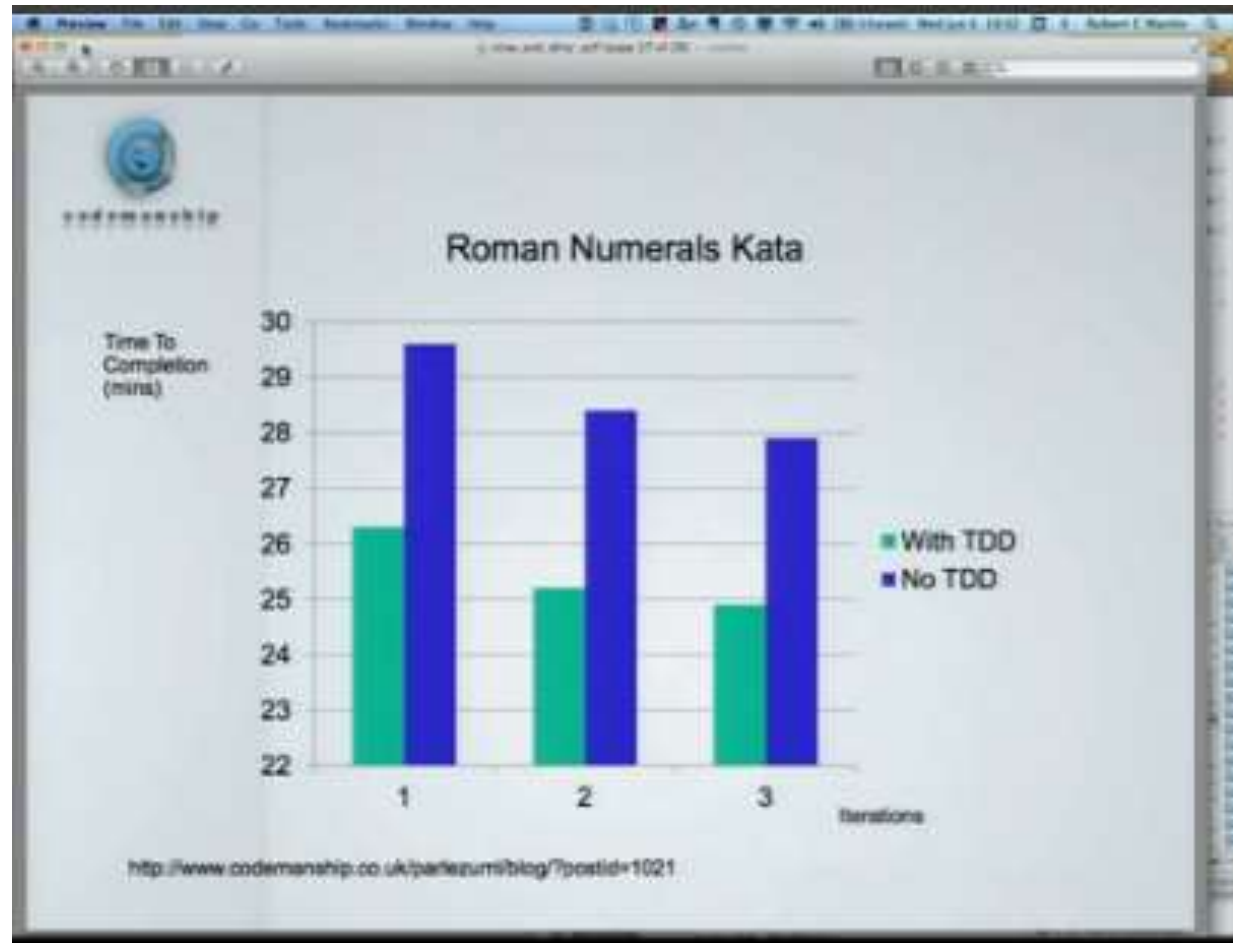
האם זה משתלם?

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



J. Gorman:

“TDD felt slow but was actually faster”



ROI for Selected Practices

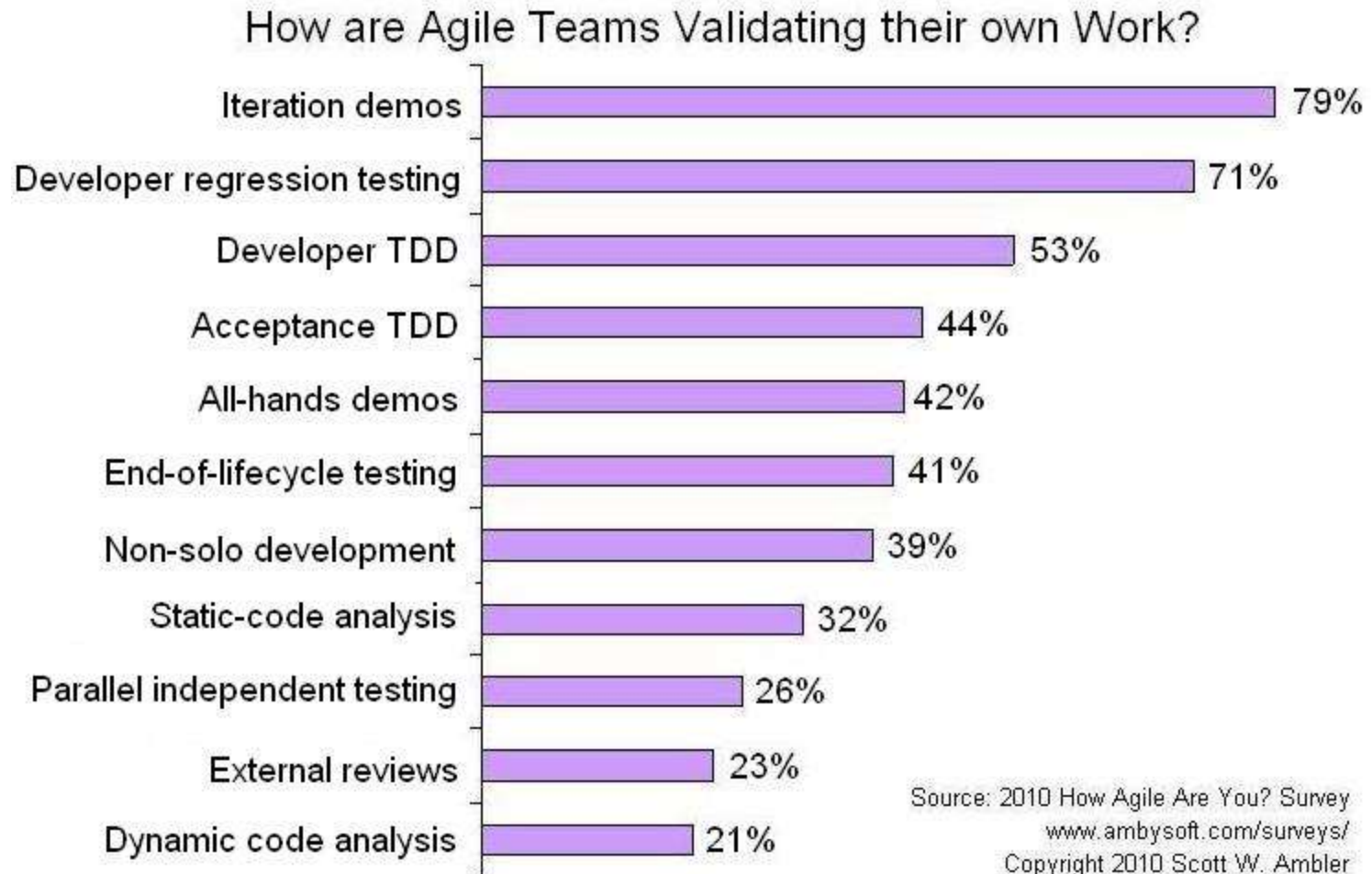
Practice	12-month ROI	36-month ROI
Test Driven Development	-	1000%+
PSP/TSP	-	800%
Formal Inspections	250%	600%+
Productivity Measurement	150%	600%
Process Assessments	150%	600%
Management Training	115%	550%
Scrum	-	500%
Process Improvement Program	-	500%
Technical Staff Training	90%	500%

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

איך לשכנע בכדאיות?

- “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 (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/>

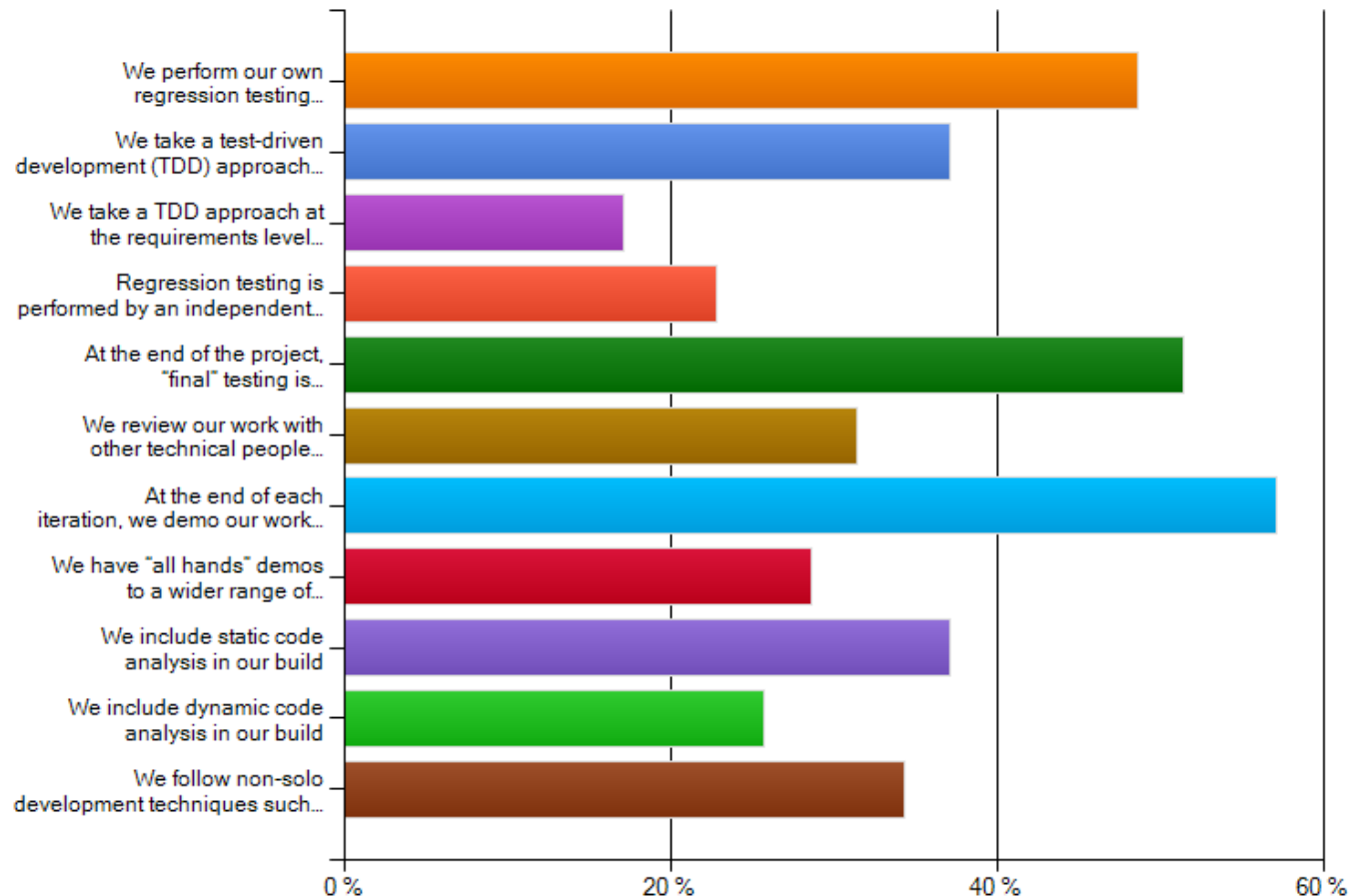
How Agile Are You? 2010 Survey



Source: 2010 How Agile Are You? Survey
www.ambysoft.com/surveys/
Copyright 2010 Scott W. Ambler

2013

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

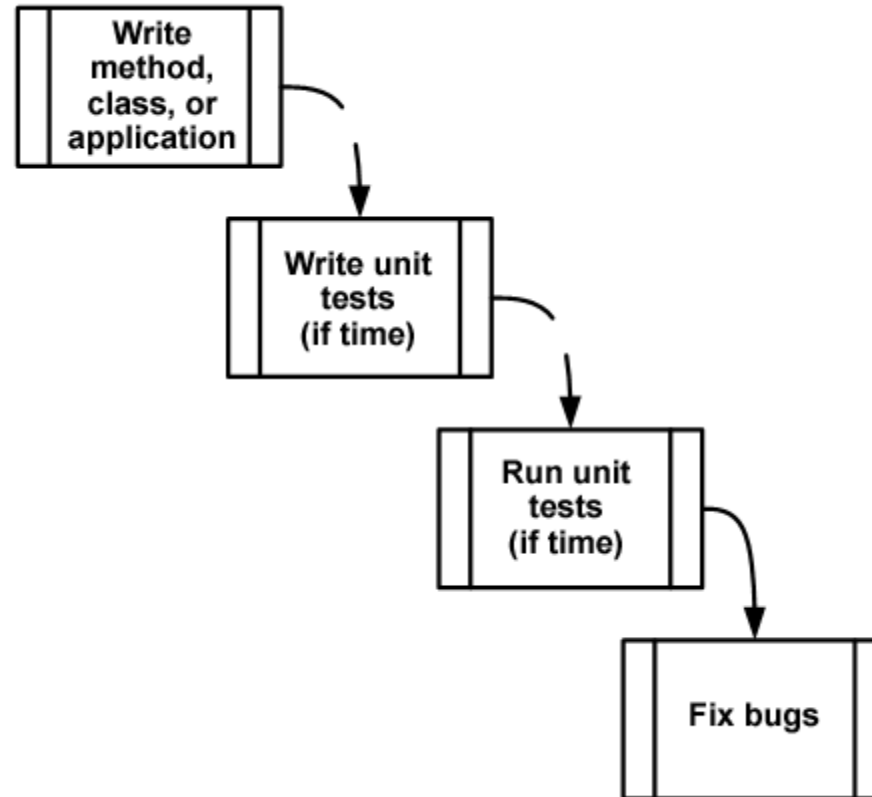


מצד שני

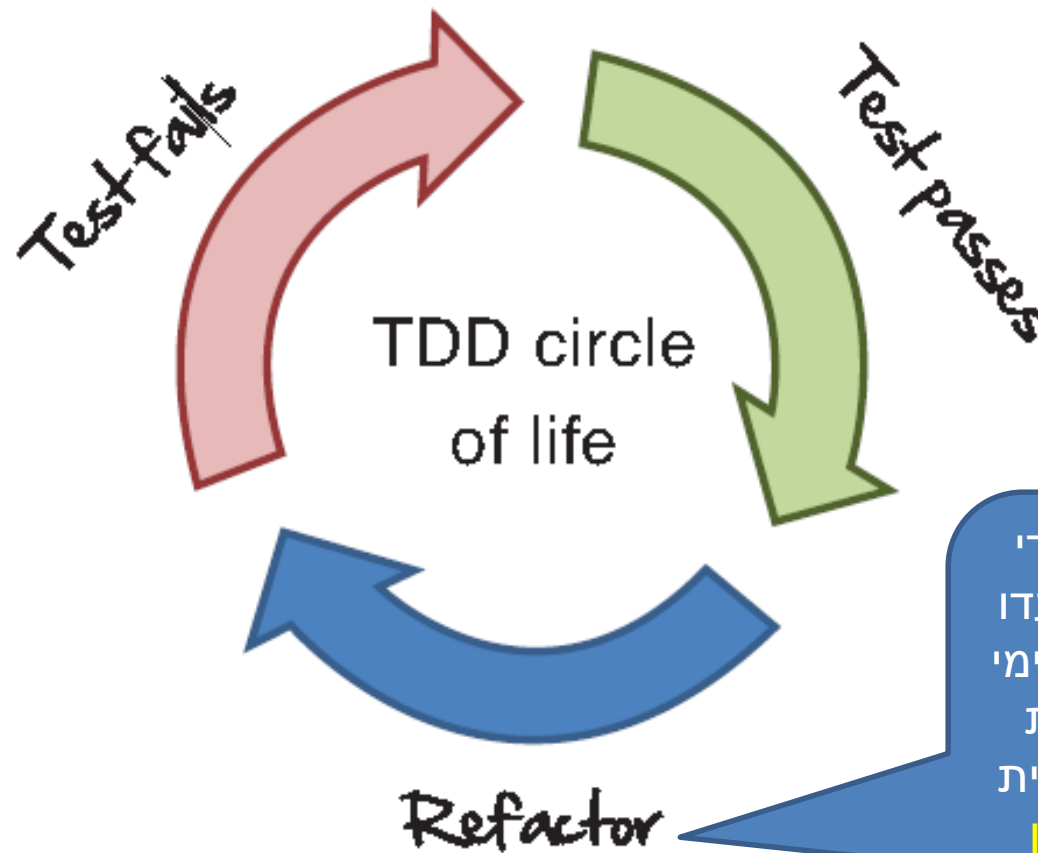
- Ayende: “But I think that even a test has got to justify its existence, and in many cases, I see people writing tests that have no real meaning. They duplicate the logic in a single class or method.”
 - <http://ayende.com/blog/4217/even-tests-has-got-to-justify-themselves> ([refs](#))
- Fowler, [CannotMeasureProductivity](#)
Dan North, [The Art of Misdirection](#)

TDD

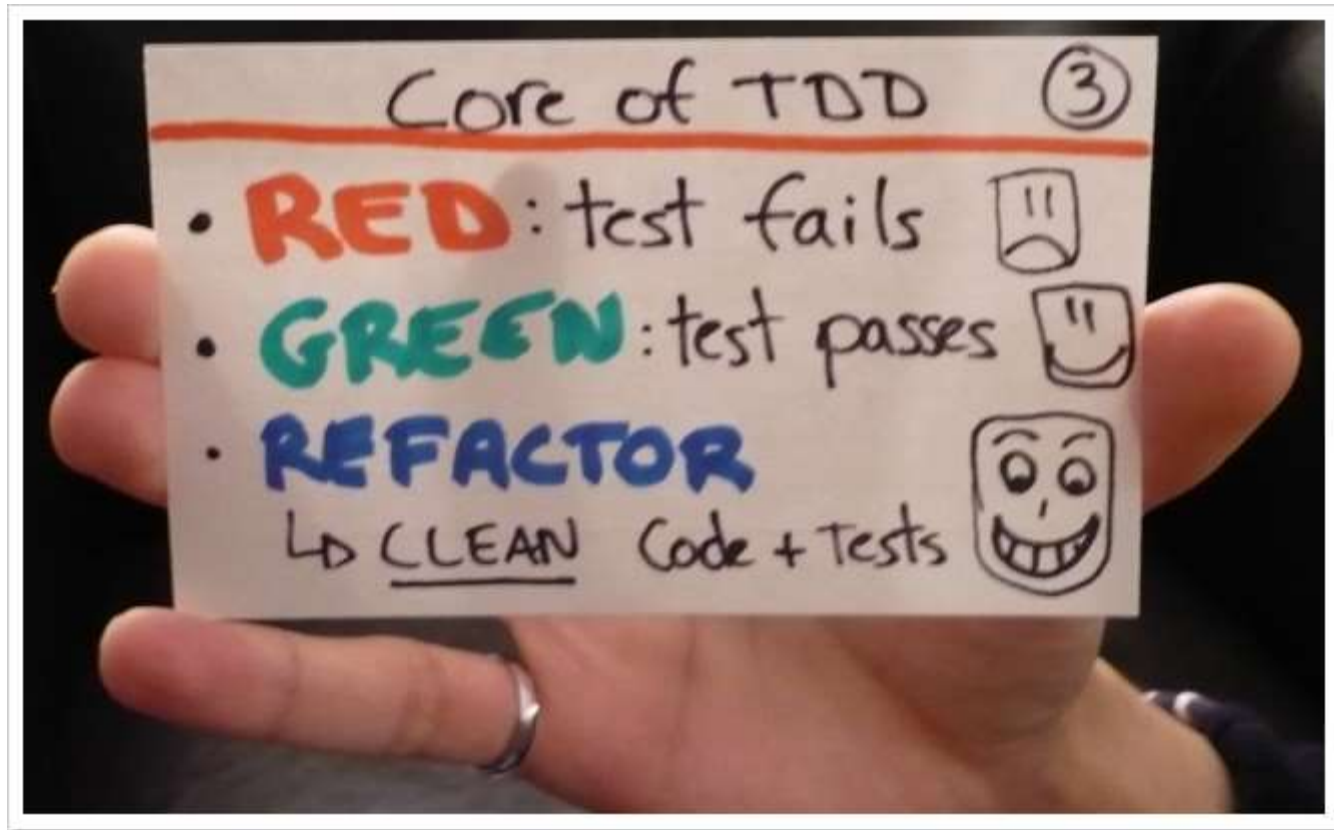
הדרך המסורתית לבדיקות (יחידה)



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**

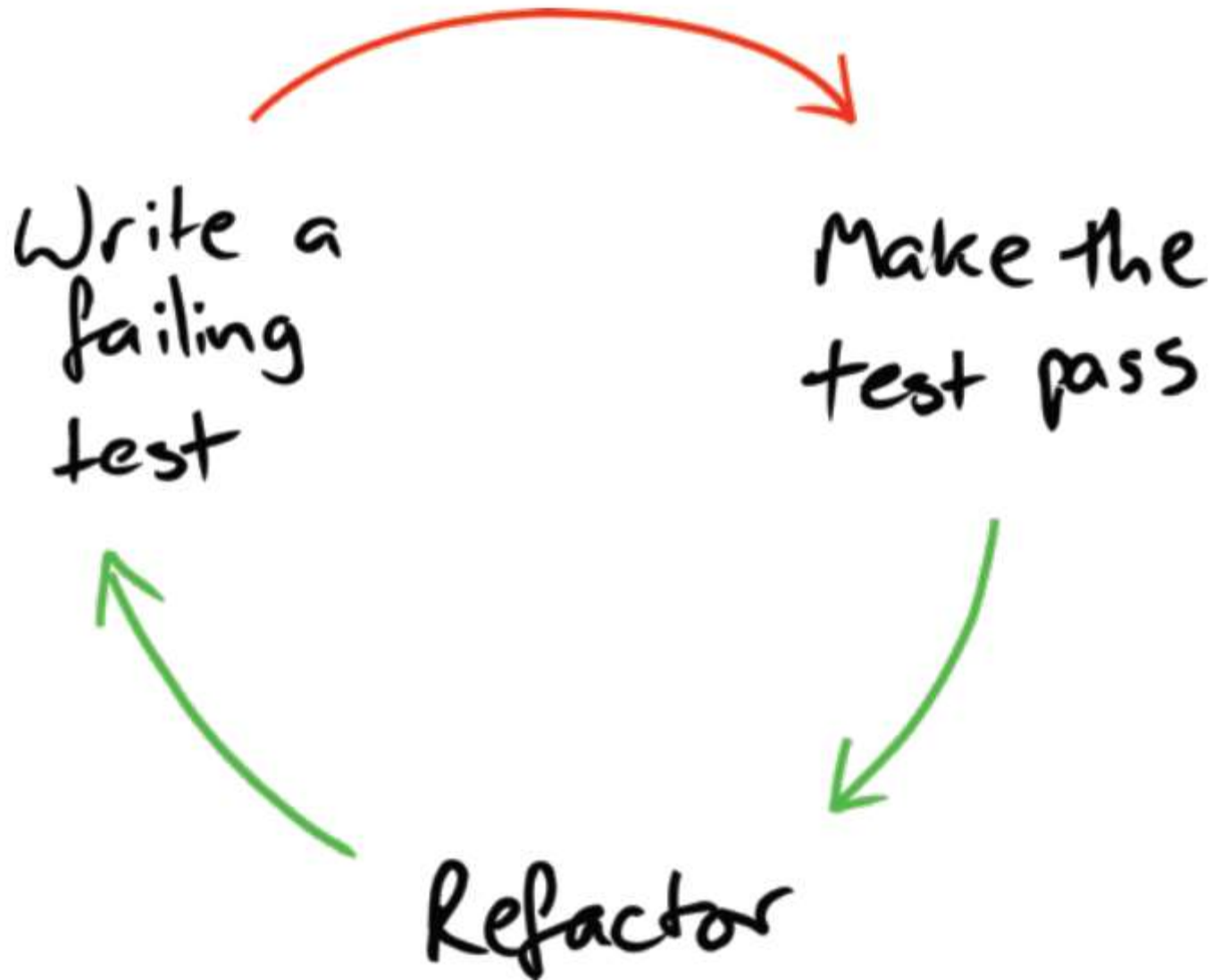


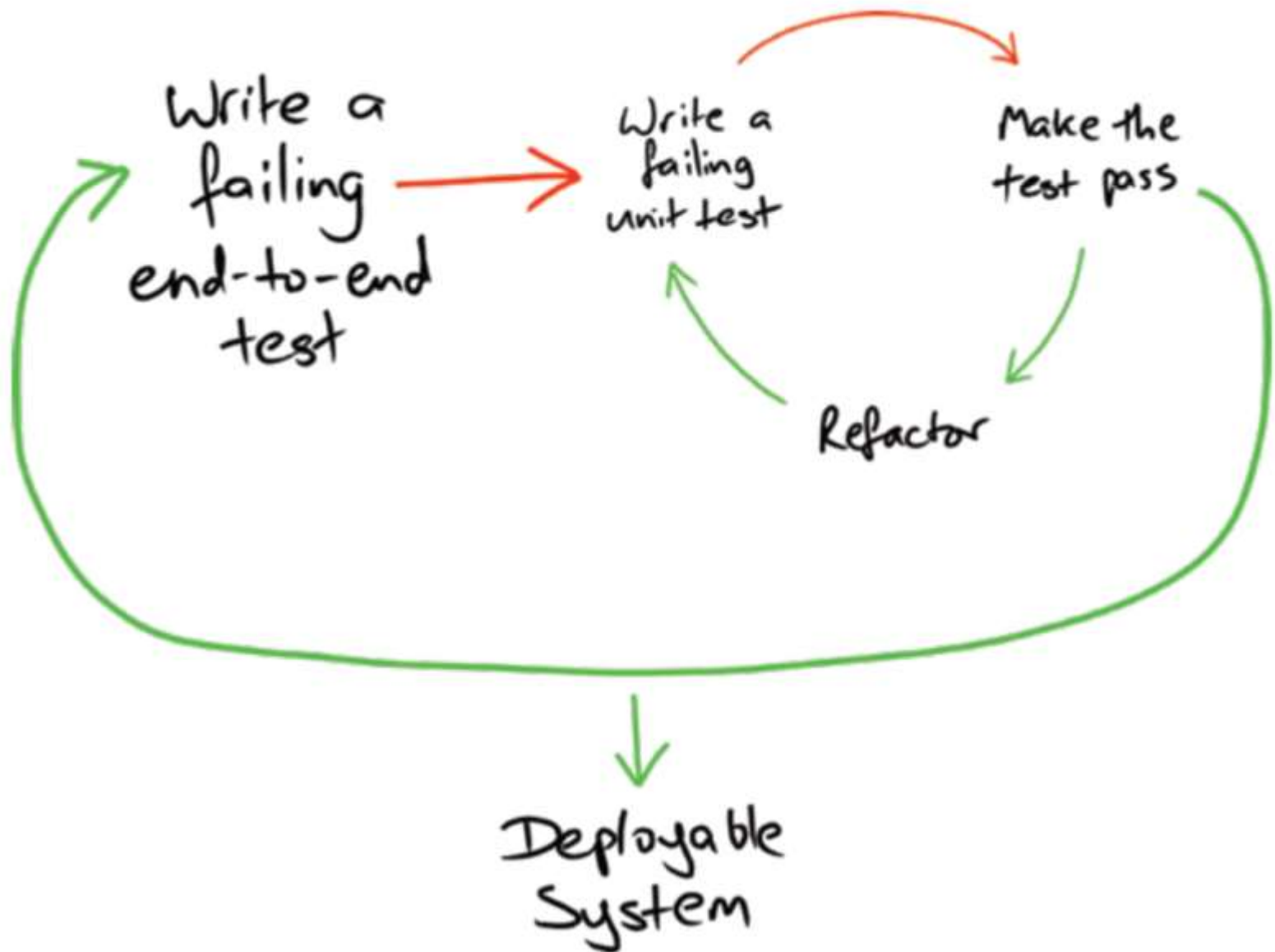
TDD – R. Martin

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.







?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

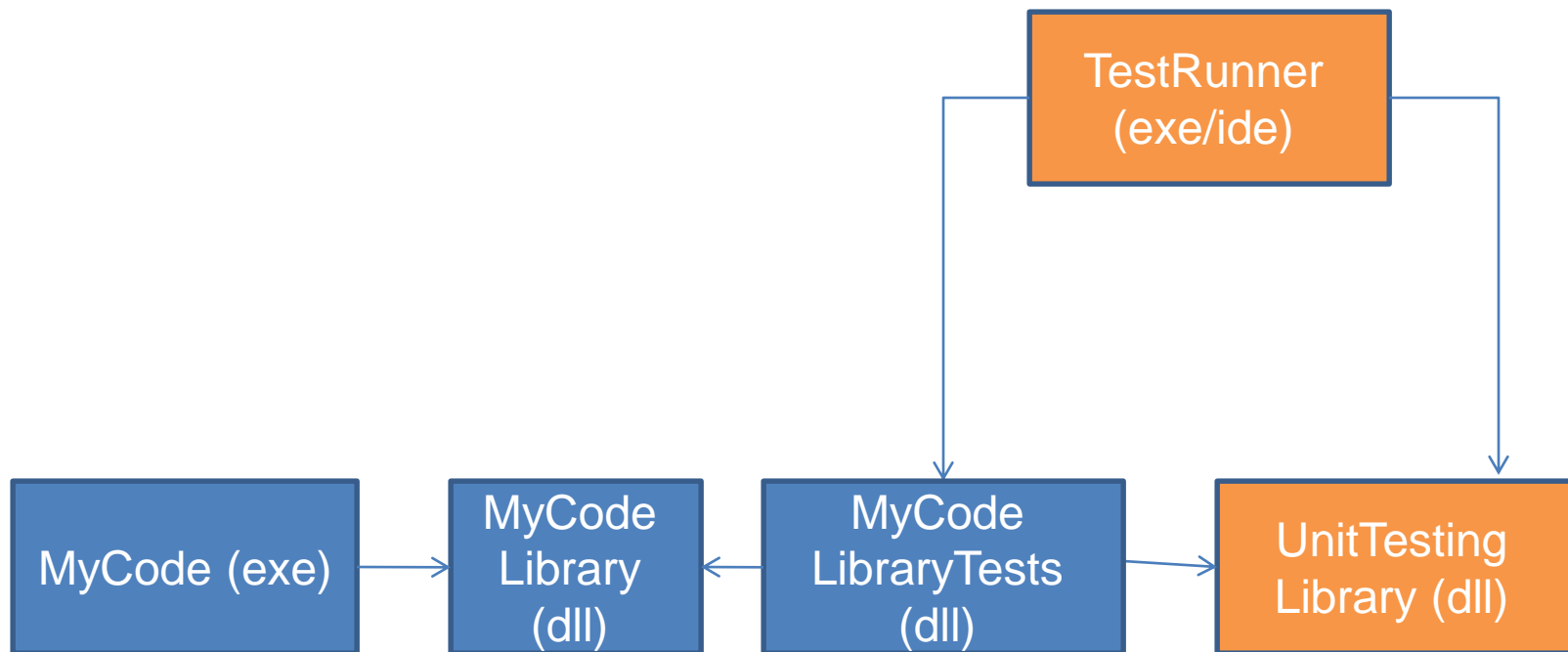
(adopted from A. Fox, Berkeley)

- **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
- Kent Beck & Erich Gamma – Small Talk
- ייצוא לשפות רבות: JUnit, CppUnit, PyUnit וכו'
- <http://www.xprogramming.com/software>
- 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: Dog/src Browse...

Package: (default) Browse...

Name:

Superclass: java.lang.Object Browse...

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: Browse...

? < Back Next > Finish Cancel

RPN Calculator Demo

- HP and OSx!
 - Spec??
 - Red-Green-Refactor, different hats
 - Commits (when?)
 - Code Smells: Feature Envy
 - BDD, test fixture
 - One assert per test
- Future: explore advanced ideas with this example
...
- Brett L. Schuchertn Getting Started with Tdd in Java using Eclipse <https://vimeo.com/10569751>

אחר: ספרות רומיות



Spec.

- <https://gist.github.com/4187205>
 - באיזה סדר דוגמאות כדאי לעבוד?
- Haines, [Roman Numerals **Kata** with Commentary](#) using Ruby, [Transformation Priority **Premise**](#) & the 4 rules of simple design

כלים - Java

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

TDD Coding Kata

- String Calculator
<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?



בהרצאת המשך \ נושאים מתקדמים

- בדיקות יחידה 2.0 למשל...
 - מאפיינים מתקדמים: אתחולים, חריגות,
 - סביבות שונות (.Net), קוד פתוח
 - פרמטרים, כיסוי, תלות, אינטראקציה עם רכיבים אחרים, התנהגות מול מצב ([Google ToT](#))
 - כלים נוספים, אוטומציה, Continuous Integration
 - בדיקות לניידים \ ענן \ רשת וכו'
 - כיצד למצוא את [הבדיקה הבאה](#)
 - כיצד להטמיע TDD בארגון? בדיקות לקוד קיים...
- הרצאה 3 + תרגיל: עזרה עם בדיקות יחידה
- משימה אישית: TDD (עמוד הבא)
- קריאה מומלצת לפעם הבאה (לא להגשה):
[Using Mock Objects](#)

משימה אישית 3 – בדיקות יחידה + TDD

- רועי אושרוב, [String Calculator Kata 1](#)
- 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 (בנוס: שיתוף pulse ו-TDGotchi בפורום)
- ציון: כיסוי, RGR, נכונות, איכות.
- אופציה: זוגות מתחלפים (אחד בודק אחד מממש [pairhero](#))



לסיכום

- בדיקות, בדיקות יחידה ופיתוח מונחה בדיקות
- בפרויקט

- בכל סבב: ניסוח בדיקת קבלת לתרחיש עיקרי
- סבב 2: חליפת 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
- 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.”

