המכללה האקדמית להנדסה ירושלים


ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT

CODESMACK

# הנדסת תוכנה

## 7. בדיקות

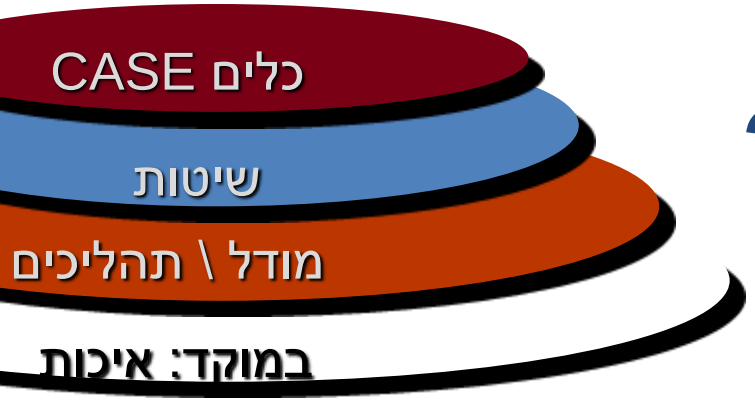
## בדיקות יחידה - I

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


Hey, have you tested out the new iPhone?
No.
Me neither.

# 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 הציוד עצמו היה תקול", הוסיף הבכיר.




# מה היום?

- בדיקות (מבוא)
  - בדיקות יחידה (Unit Testing), פיתוח מונחה בדיקות
    - כלי בדיקה (למשל JUnit)
- הרצאה 3: מעבדה
- פרויקט: המשך עבודה על סבב 1 (יעד: שבועיים) Minimum Viable Product, סיכום, רטרוספקטיבה ותכנון לסבב 2
  - סקר בדיקות לפרויקט עוד שבועיים
  - משימה אישית: בדיקות יחידה

Reid Hoffman (LinkedIn founder):
“If you are not embarrassed by the first version of your product, you’ve launched too late.”

# 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

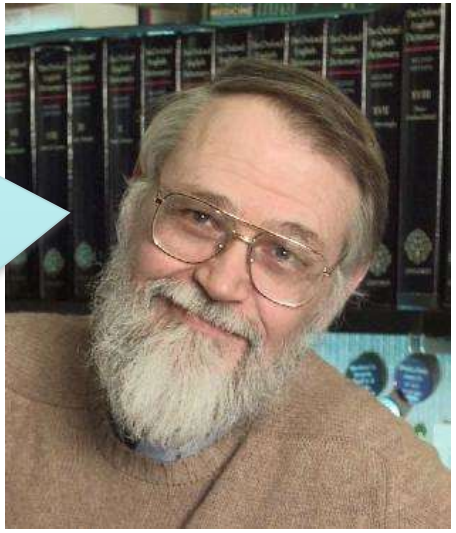
# Links

- MVC BDD testing lecture @mvcConf
- Jbraibns, ” The World’s Best Intro to TDD “(demo video) Outside-In Test-Driven Development, Pluralsight course (demo).
- Hendrickson, “Driving Development with Tests: ATDD and TDD"
- Software Testing - How to Make Software Fail (Udacity course)
- http://bit.ly/AgileTestTools spreadsheet
- TDD Hebrew post http://www.softwarearchiblog.com/2012/08/unit-testing.html
- Non-trivial and real-world feedbacks on writing Unit-Tests, post 2012 (addresses testing in .net with various tools and processes)
- Unit Tests Are FIRST, 2012
- Introduction to Test Driven Development http://www.agiledata.org/essays/tdd.html
- Shore, Let’s play TDD, video blog series

# מעט על רטרוספקטיבה

- מטרה (עוד בהרצאה קודמת)
- 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.



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

Source: saas

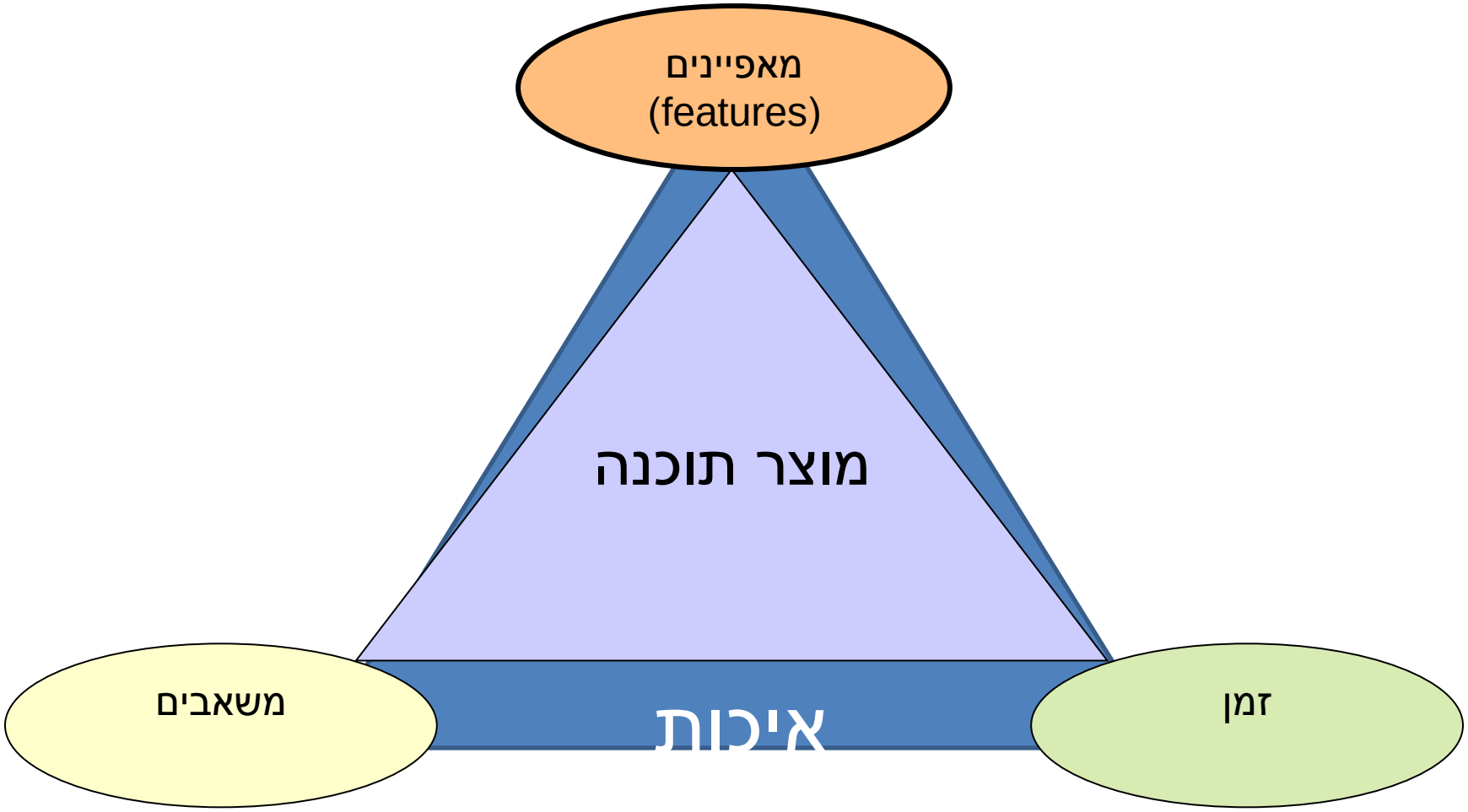
# בדיקות תוכנה

- למה לבדוק?
- איך לבדוק?
- אילו בדיקות?
- מי בודק?
- כמה לבדוק?
  - Boeing 787 wing break test
- למה עכשיו?

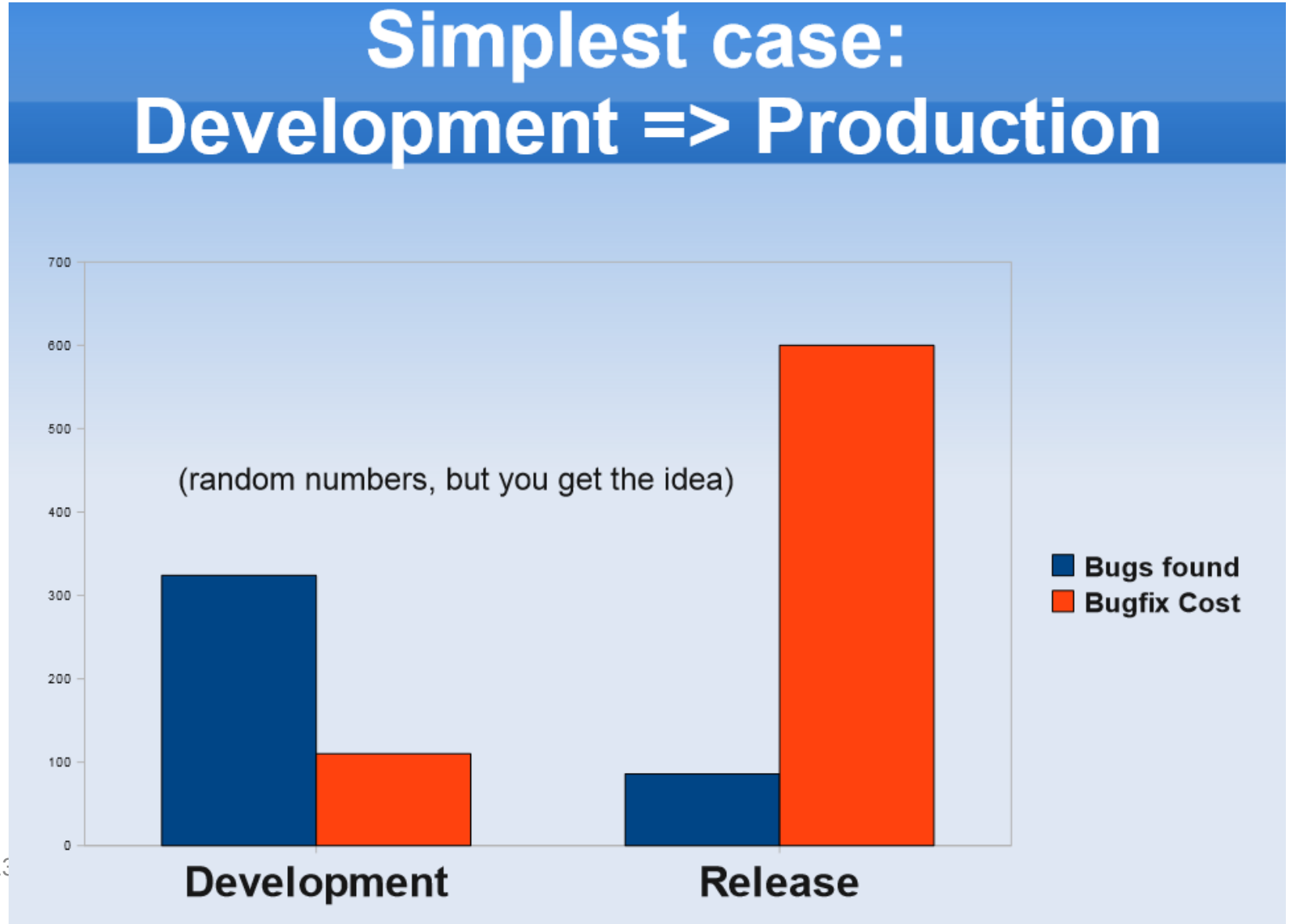
# למה לבדוק?

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

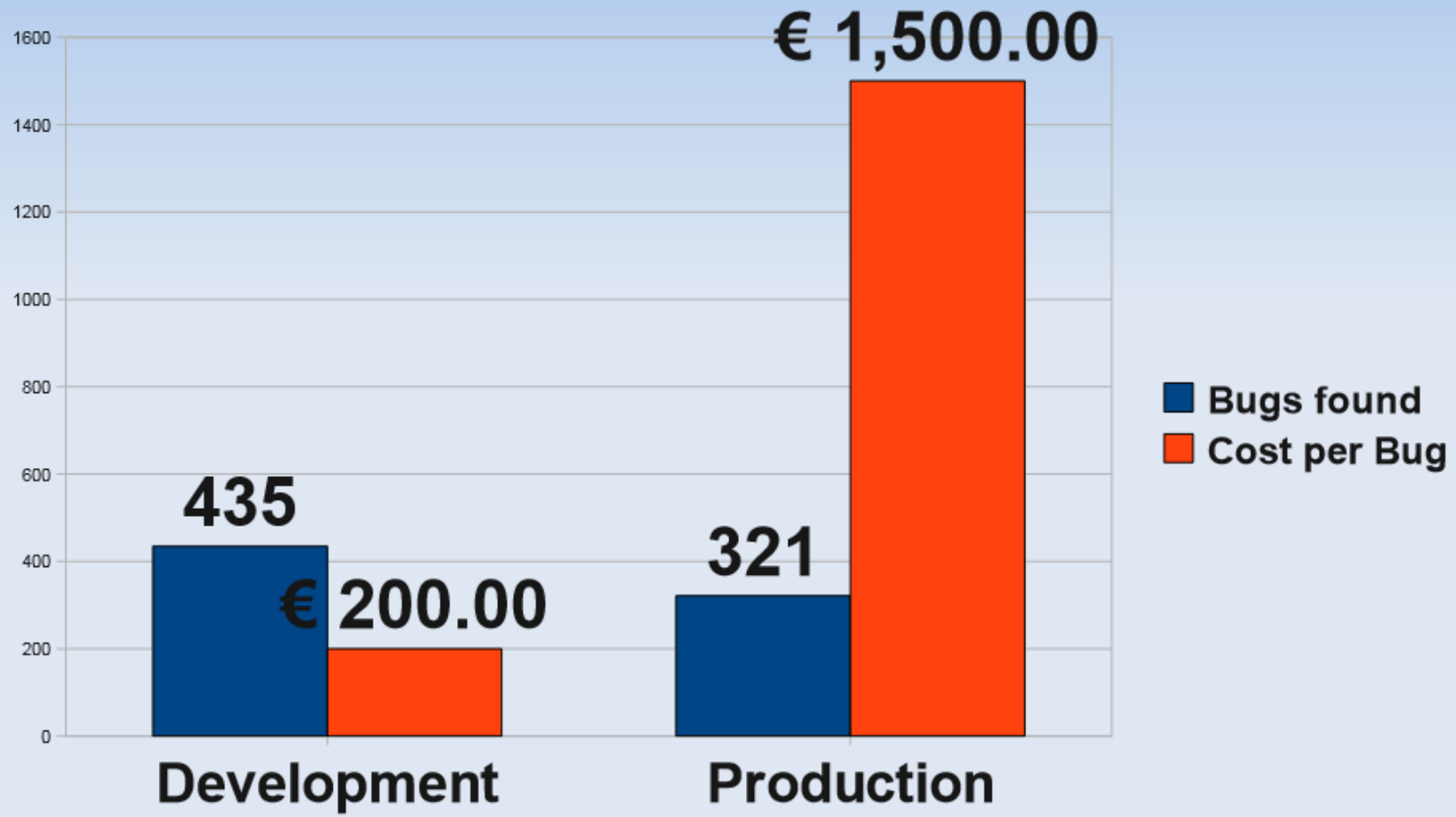
# תזכורת: פרויקט תוכנה:



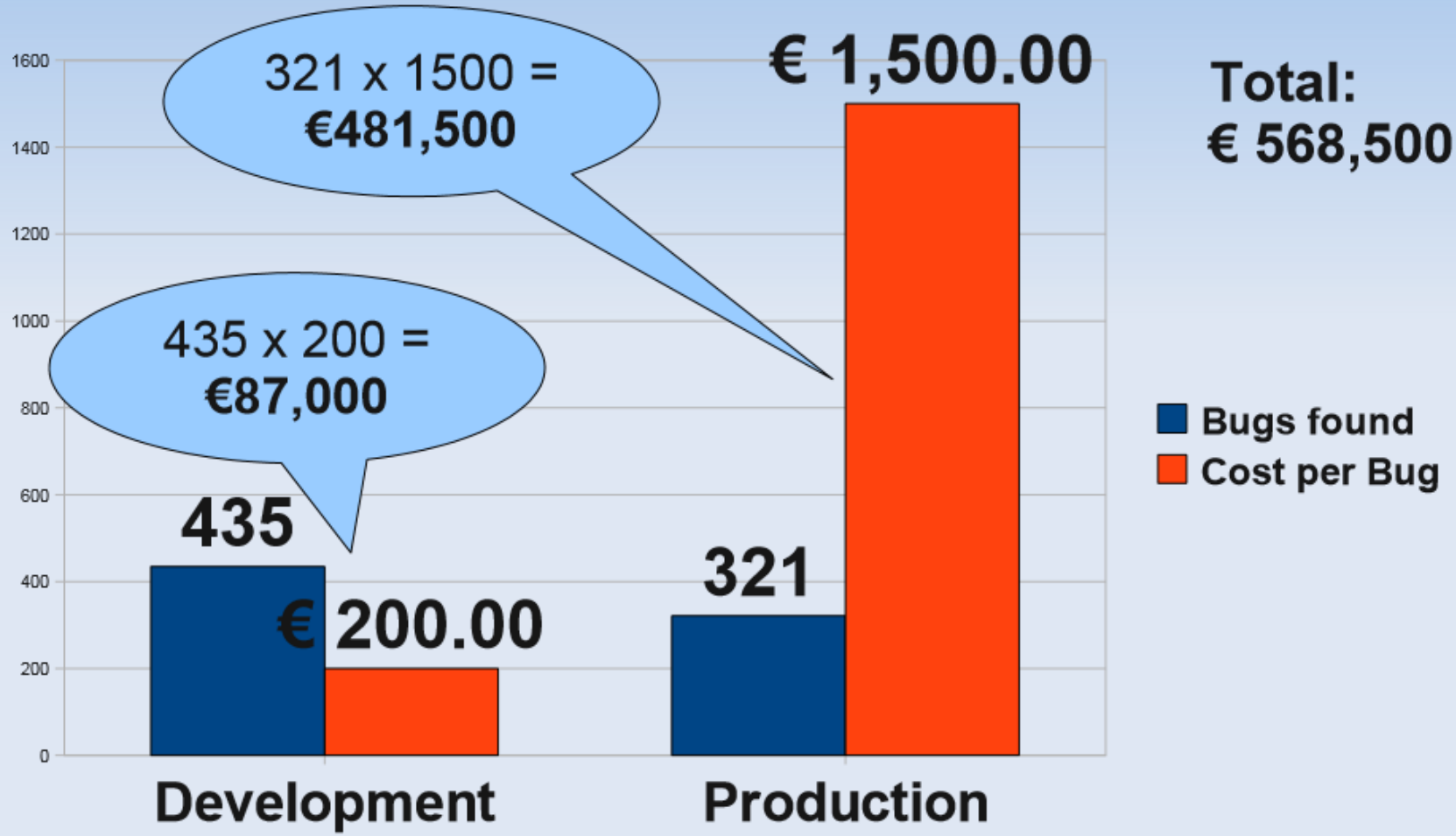
# למה לבדוק?


Simplest case:
Development => Production
700
600
500
400
300
200
100
0
(random numbers, but you get the idea)
Bugs found
Bugfix Cost
Development
Release

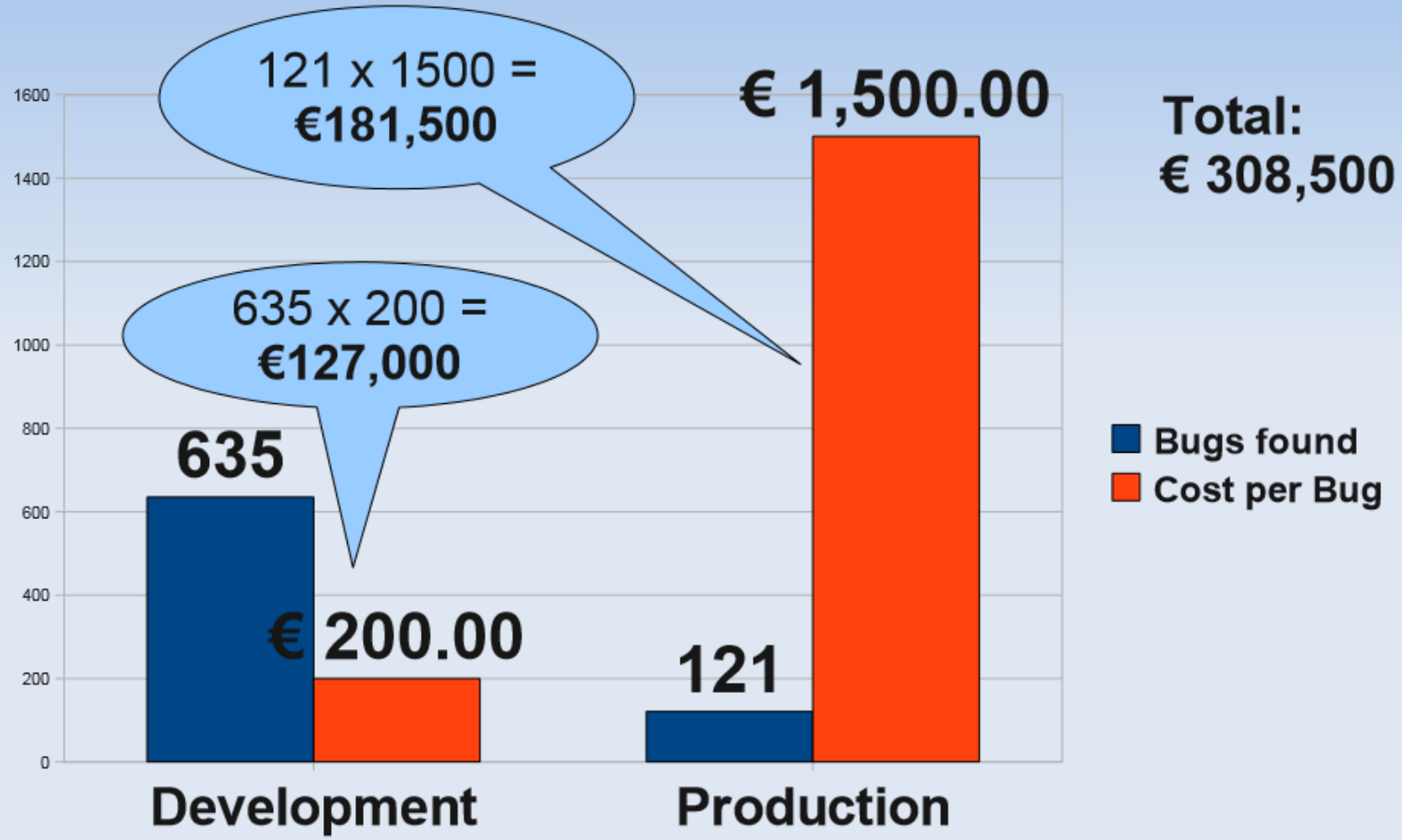
# Real world (no testing)



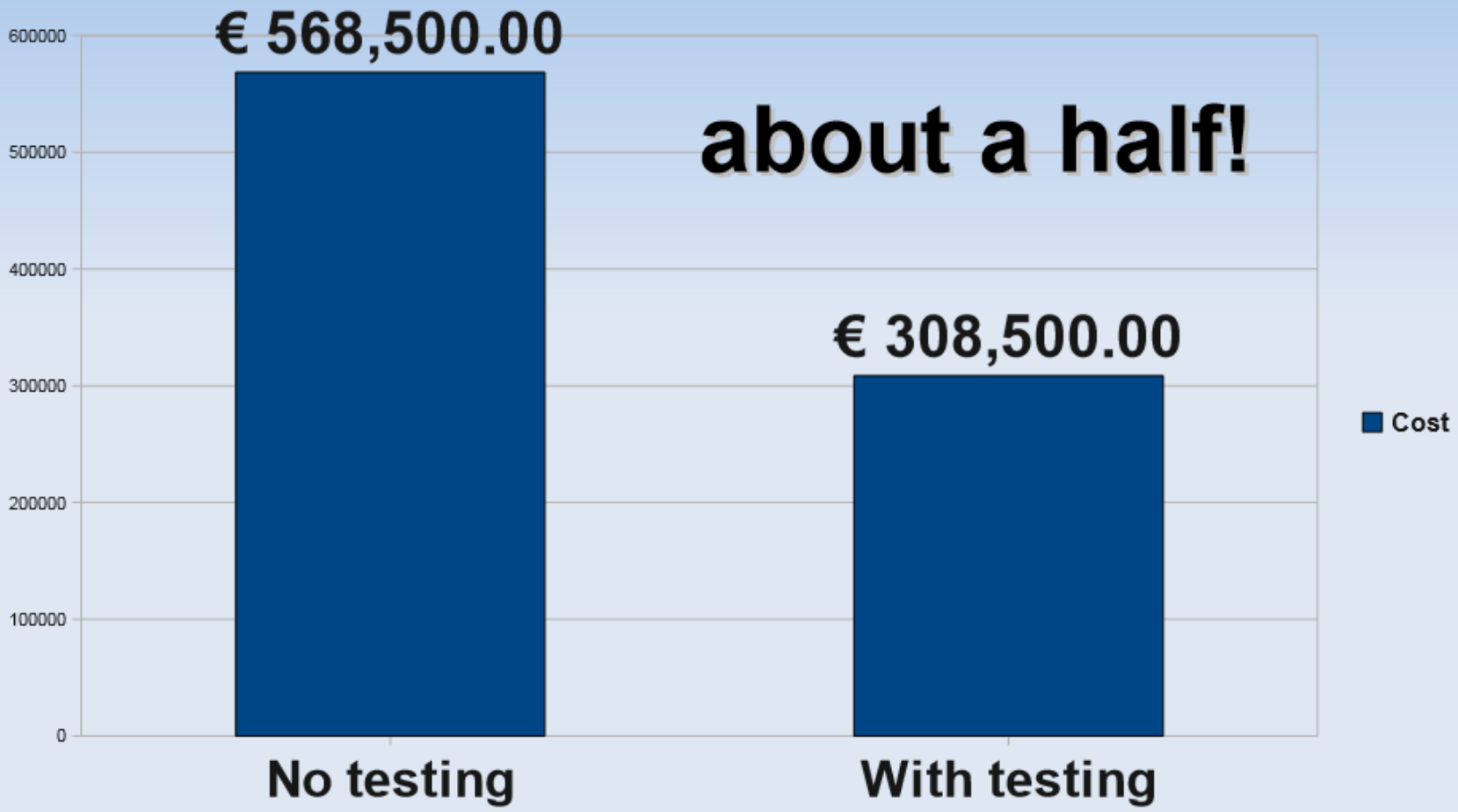
# Real world (no testing)



# Real world + testing



# no testing vs + testing



# איך אתם בודקים את התוכנה שלכם?

1. מריץ בעצמי ומדבג \ printf
2. נותן לחבר\ה 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...

# ננסה להתמקד

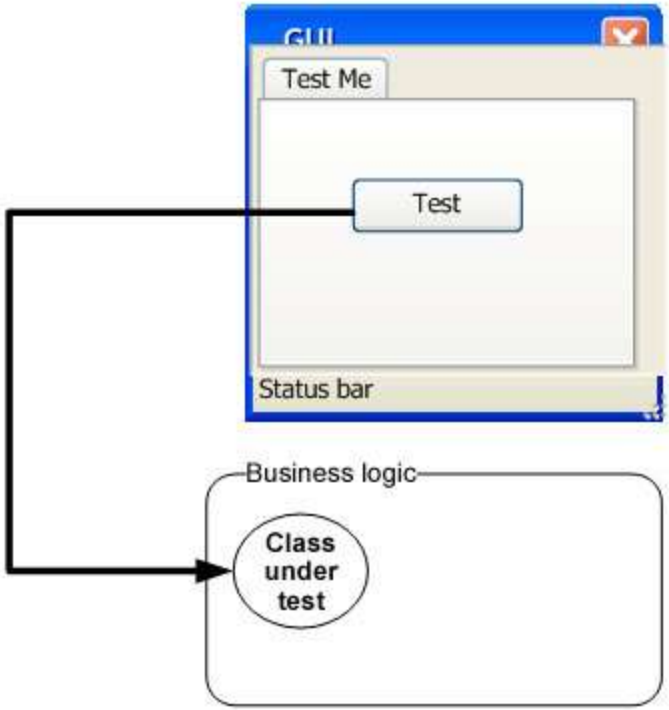
- בדיקות קצה לקצה (משתמש\קבלה\פונקציונליות)
  - האם המערכת עובדת בשלמותה?
- בדיקות אינטגרציה
  - האם הקוד שכתבנו עובד מול קוד אחר?
  - Jbrains, Integrated Tests are a Scam (lecture)
- בדיקות יחידה (מפתח)
  - האם המודולים עושים את הדבר הנכון? נוחים לשימוש? ע"י מי?

# בדיקת יחידה

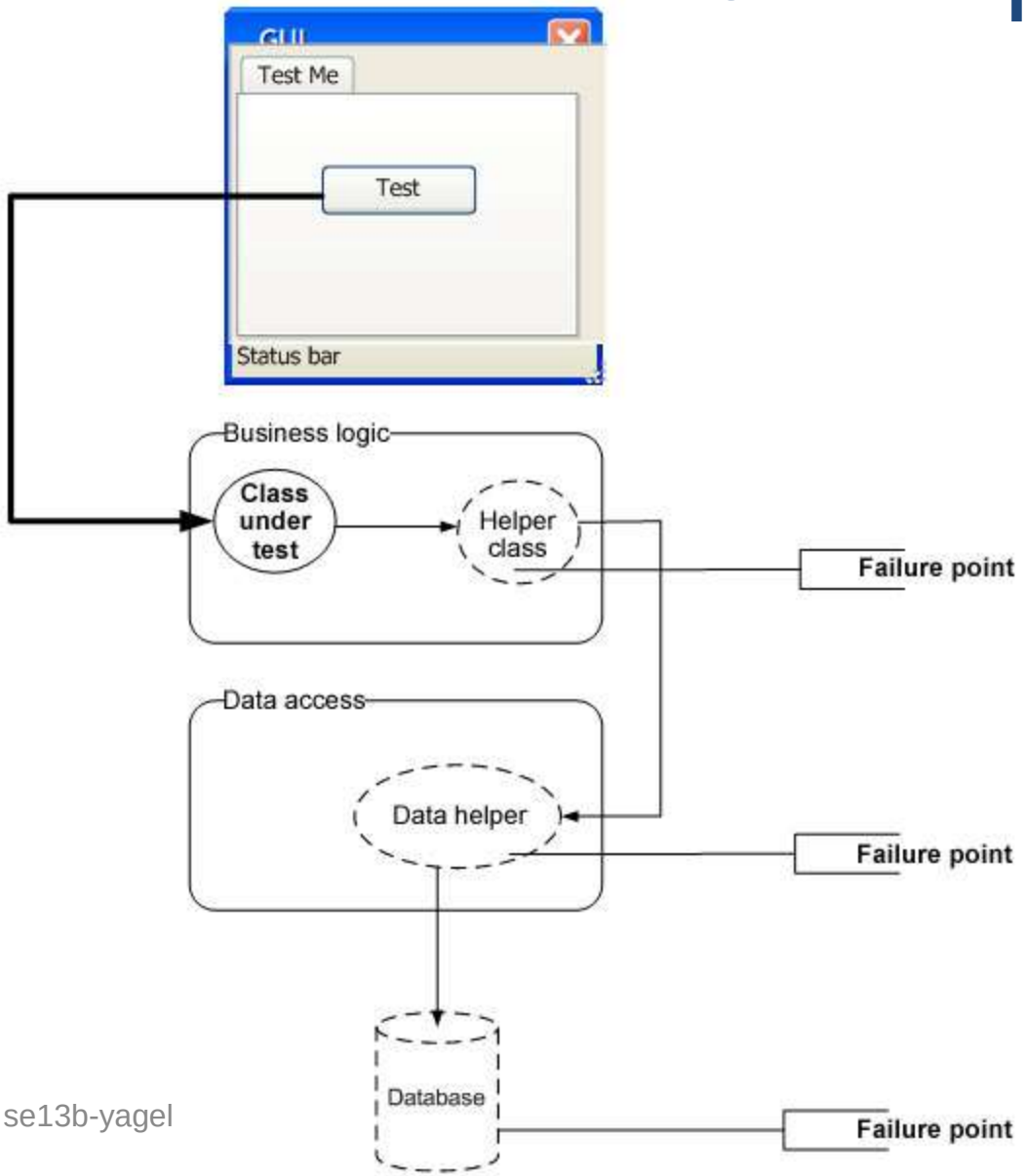
- נסיון להגדרה: בדיקת יחידה היא קוד שקורא לקוד אחר ובודק אח"כ נכונות של טענות מסוימות. "יחידה" היא "קטנה" בד"כ פונקציה, מתודה
- System Under Test (SUT) – הדבר שאותו אנחנו בודקים

# 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

- לפעמים משלבות בדיקות ממשק משתמש
- כלים לדוגמא: Fit, Selenium, WatiR / N, Capybara

```
[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"));
    }
}
```

# ATDD / BDD / Executable Spec.

```
Story: Animal Submission

  As a Zoologist
  I want to add a new animal to the site
  So that I can share my animal knowledge with the community

  Scenario: successful submission
  Given I'm on the animal creation page

  When I add a new animal

  Then I should see the page for my newly created animal
  And the notice 'Thank you for your animal submission!'
```

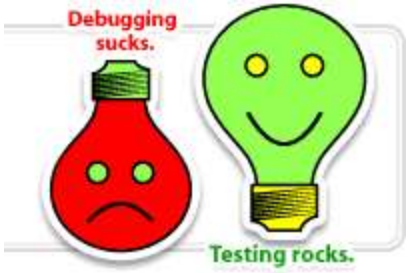
# ATDD / BDD / Executable Spec.

```
# animal_steps.rb
When "I add a new animal" do
  fills_in 'Name', :with => 'Alligator'
  selects 'Chordata', :from => 'Phylum'
  fills_in 'Animal Class', :with => 'Sauropsida'
  fills_in 'Order', :with => 'Crocodilia'
  fills_in 'Family', :with => 'Alligatoridae'
  fills_in 'Genus', :with => 'Alligator'
  checks 'Lay Eggs'
  clicks_button 'Create'
end
```

# 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...
- חסרונות
  - קוד (תחזוקה, תיכון, בדיקות?)
  - זמן (האם בדיקות אינטגרציה ומערכת לא מספיקות?)
  - 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

# מתי כותבים?

- מפל המים: בסוף, QA
- אג'ייל: קודם!
  - Test First
  - XP: Test Driven Development
  - התקבל כנוהג כללי
  - משפחת xDD (Behavior, Feature, …)
- הצדקה Brandon Satrom:

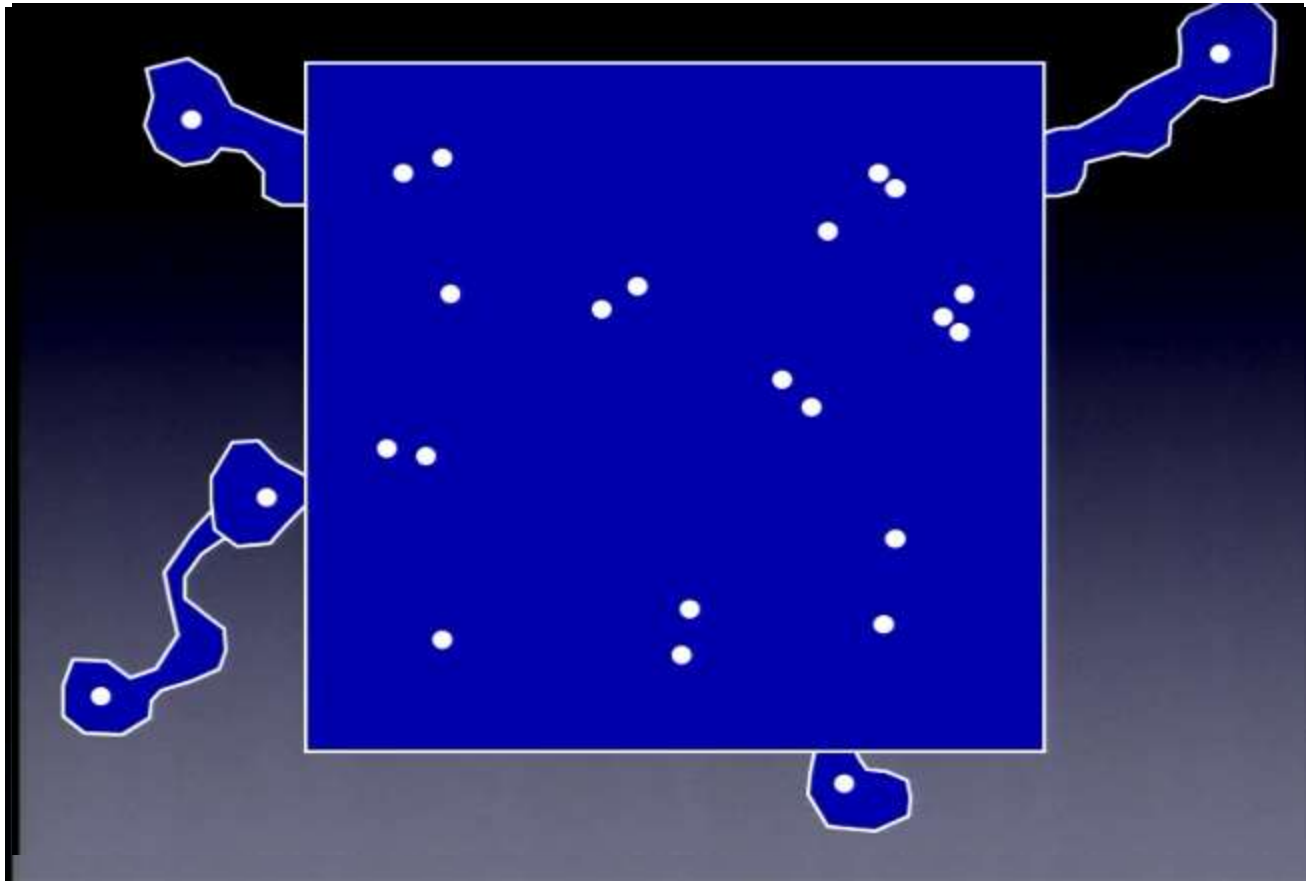
# יתרונות TDD

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

# חסרונות TDD

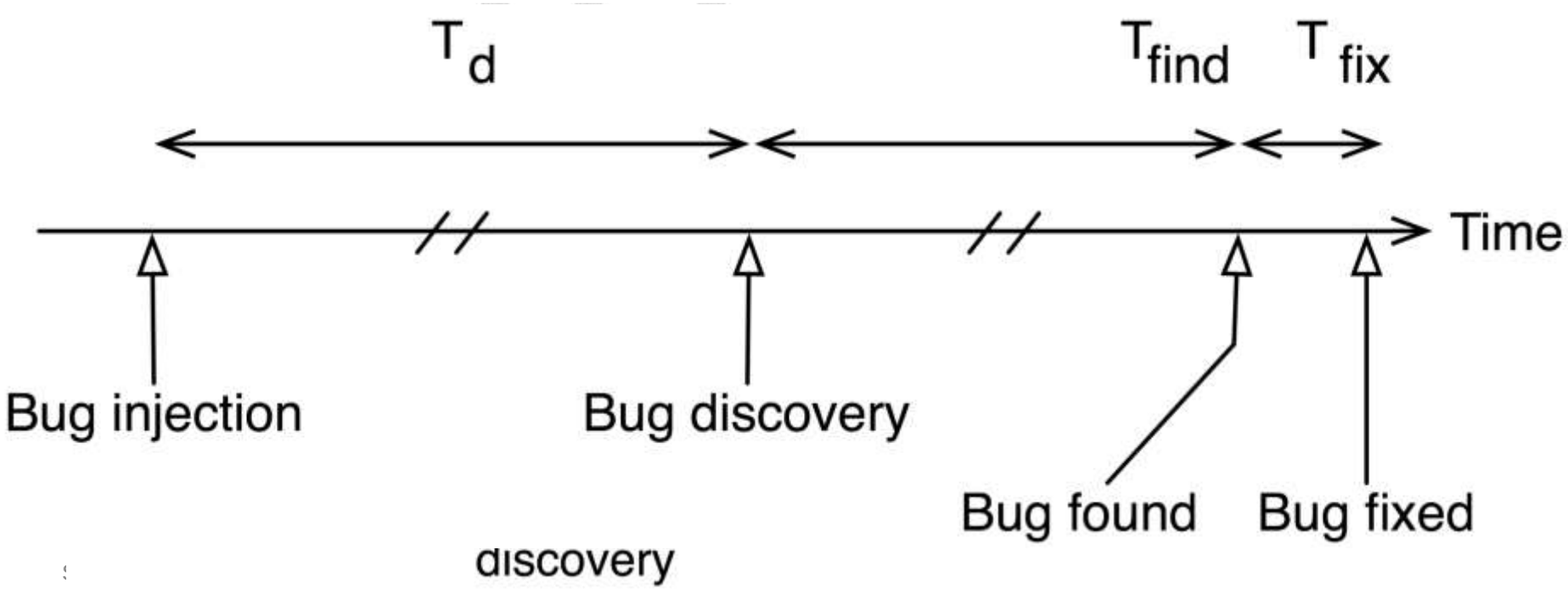
- השקעה בממשק (api) על חשבון פונקציונליות?
- לא מכסה דרישות (ראו BDD)
- עקומת למידה, כולל שיטות משלימות
- מצריך שיתוף פעולה ועבודת צוות
- כמה להשקיע מראש? כיצד מודדים?
- לא מהווה תחליף לחשיבה...
- תירוצים לא חסרים...
- ראו גם Pragmatic not Dogmatic TDD - Agile2012

# תיכון מתמשך



# האם זה משתלם?

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

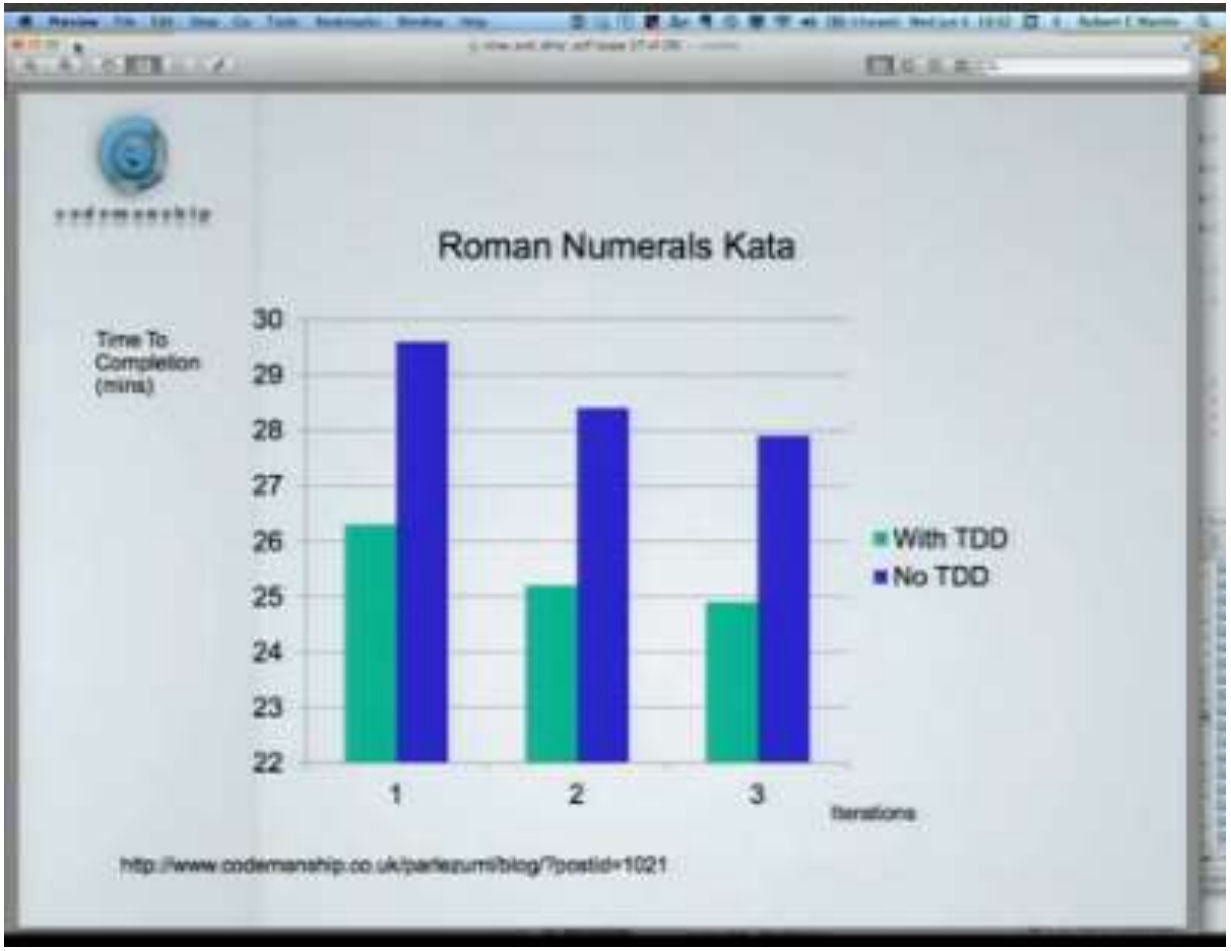


# איך לשכנע בכדאיות?

- "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/

# J. Gorman:

# "TDD felt slow but was actually faster"



# ROI for Selected Practices

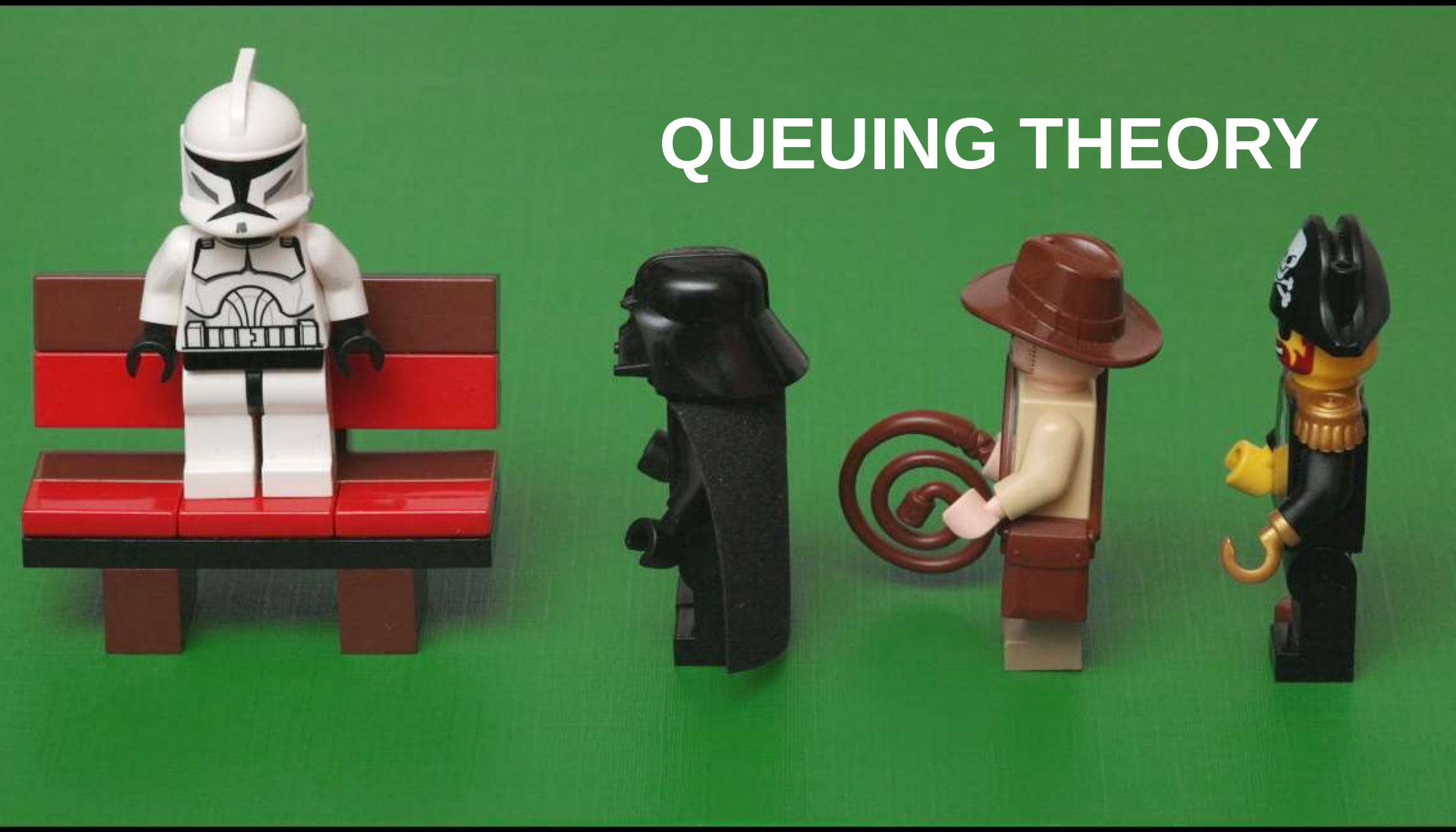
| Practice | 12-month ROI | 36-month ROI |
|---|---|---|
| Test Driven Development | - | 1000%+ |
| PSP/TSP | - | 800% |
| Formal Inspections | 250% | 600%+ |
| Productivity Measurement | 150% | 600% |
| Process Assessments | 150% | 600% |
| Management Training | 115% | 550% |
| Scrum | - | 500% |
|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

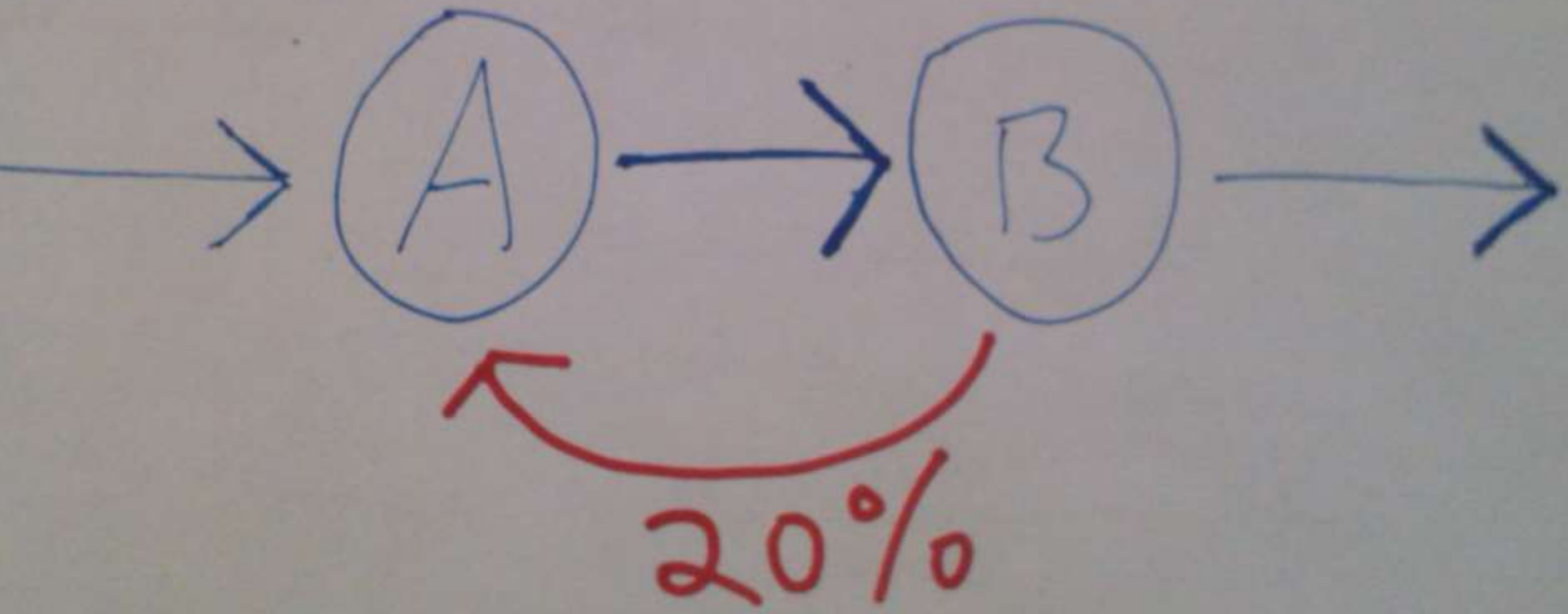
*Sources: Rico, et al 2009; DACS 2007; McConnell 2004; Jones, 1994.*

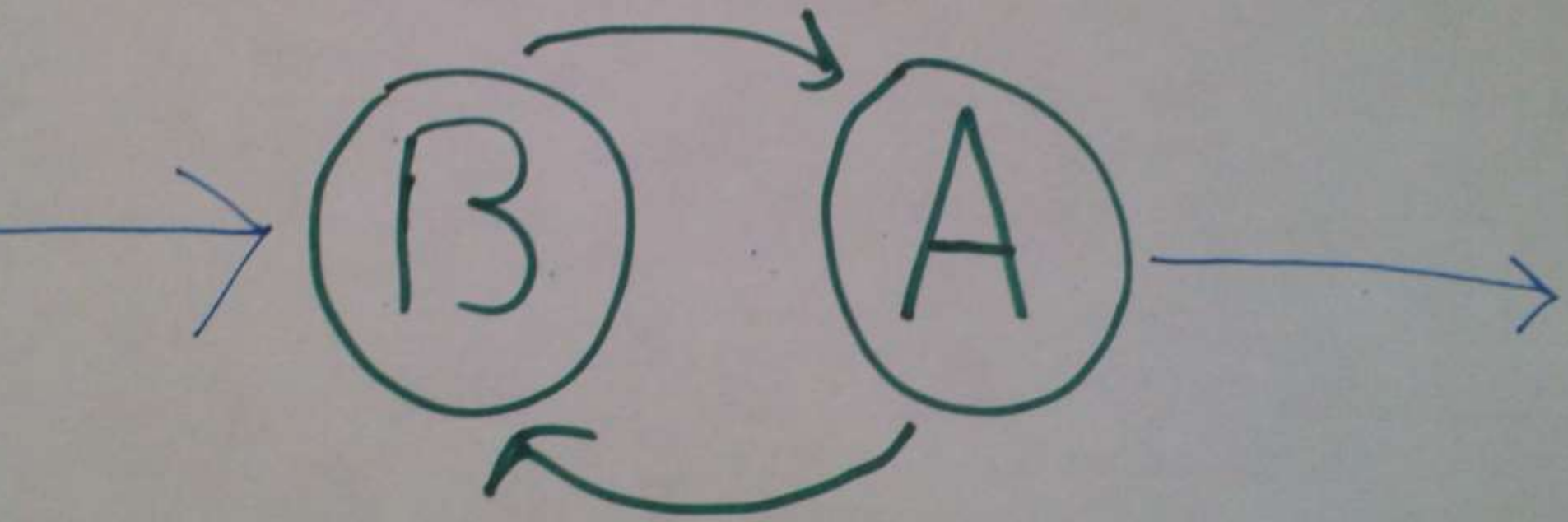
Construx

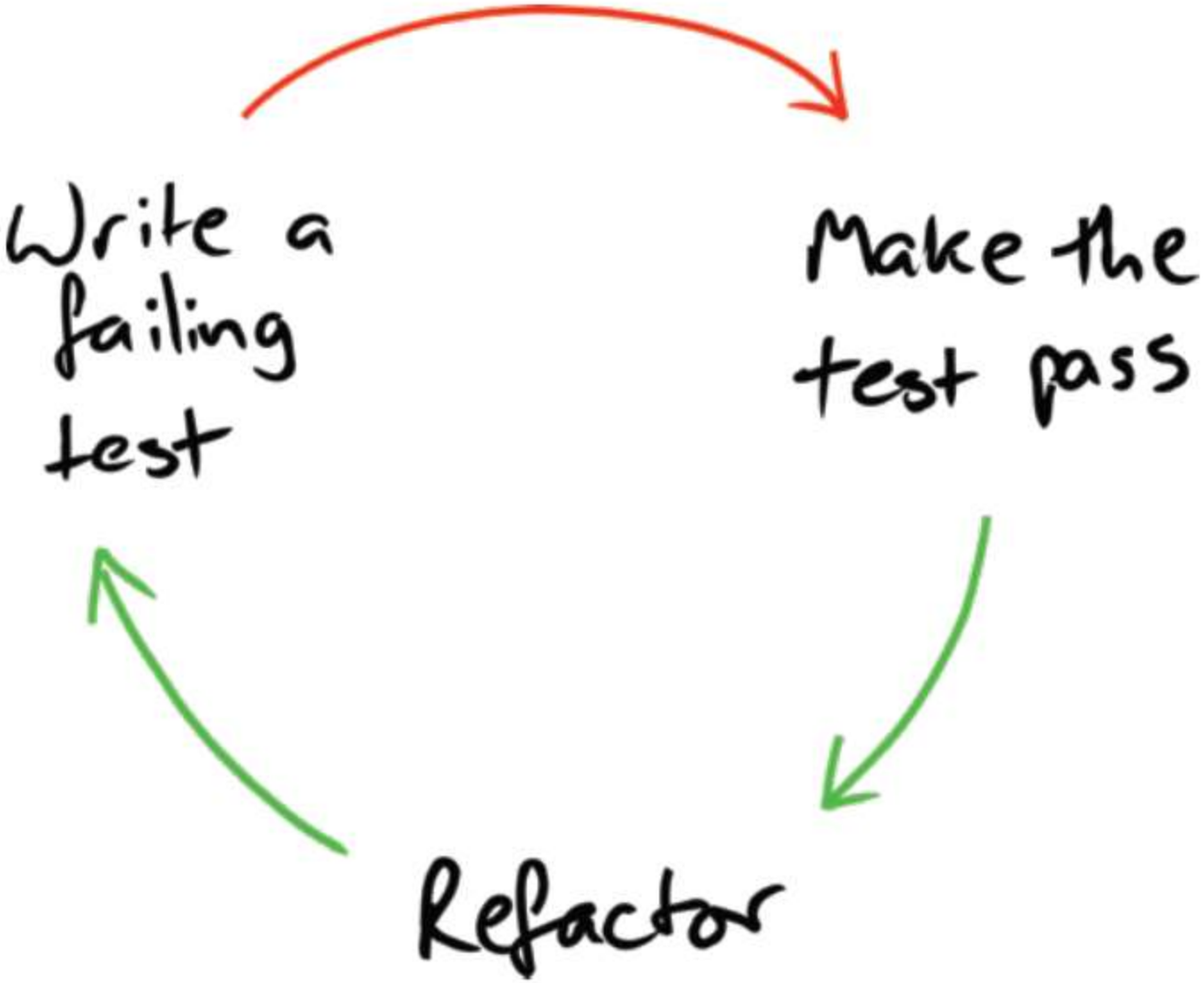
15

# QUEUING THEORY

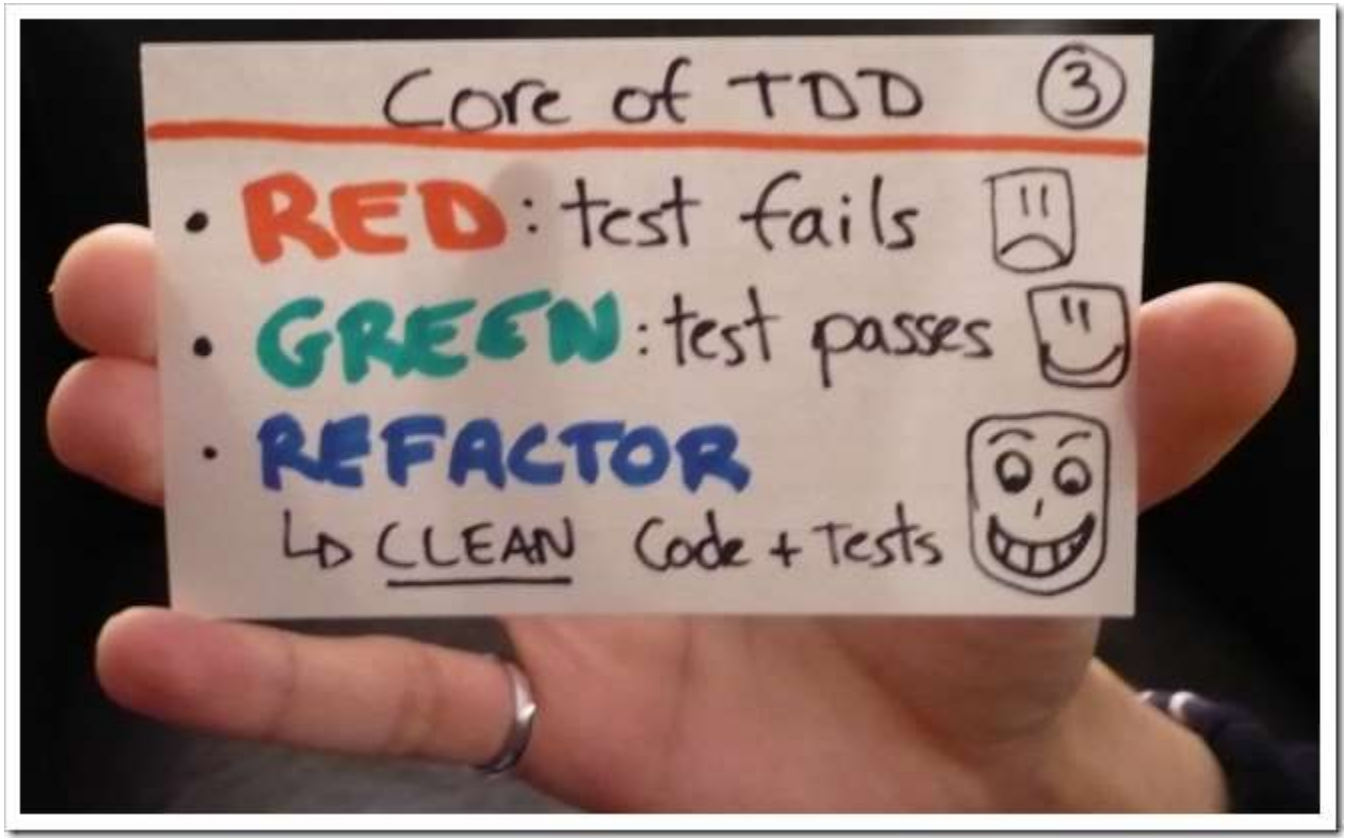



A
B
20%


B
A




http://amzn.to/atddbook

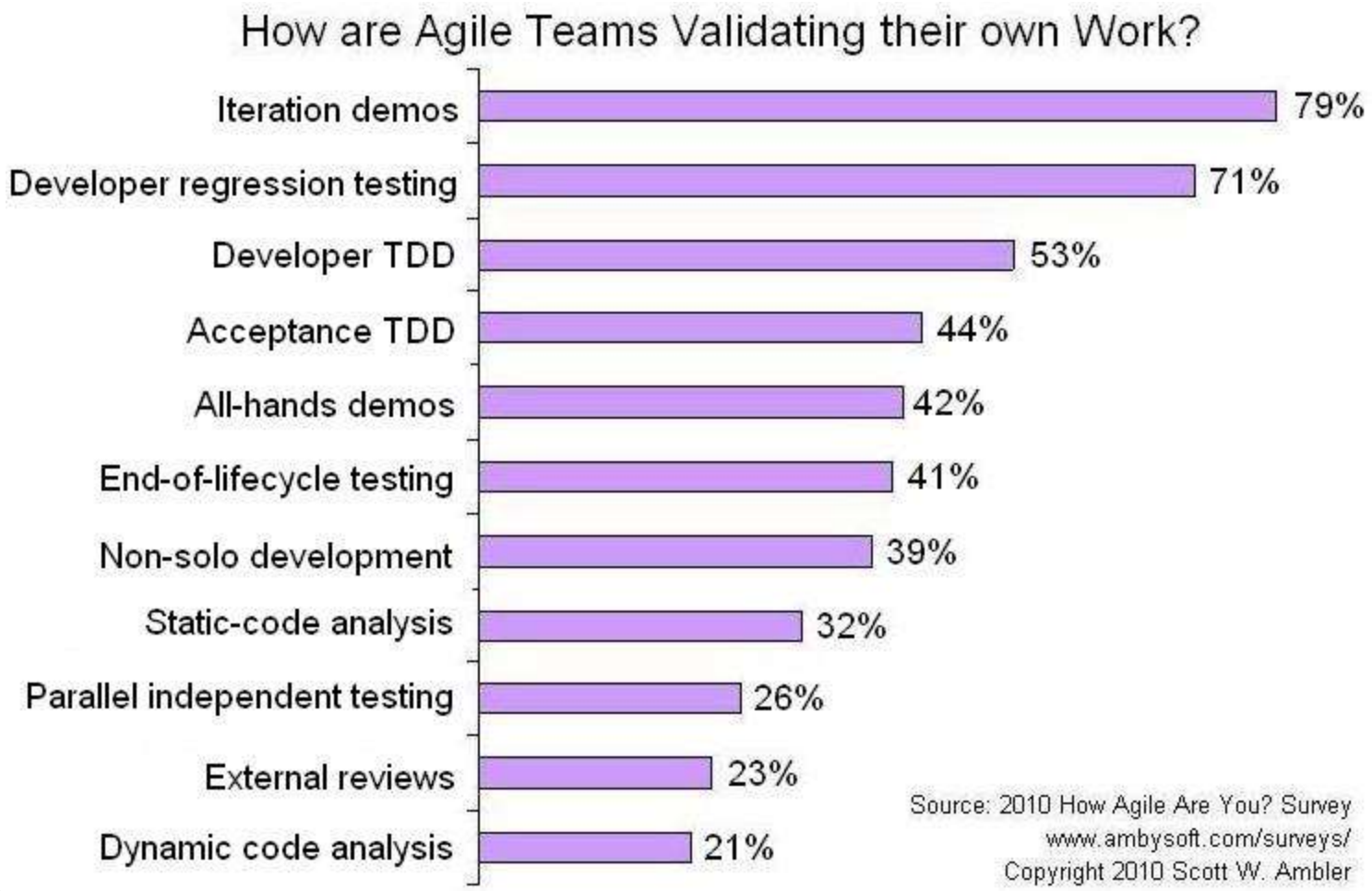



http://www.doolwind.com/blog/test-driven-game-development/

# מצד שני

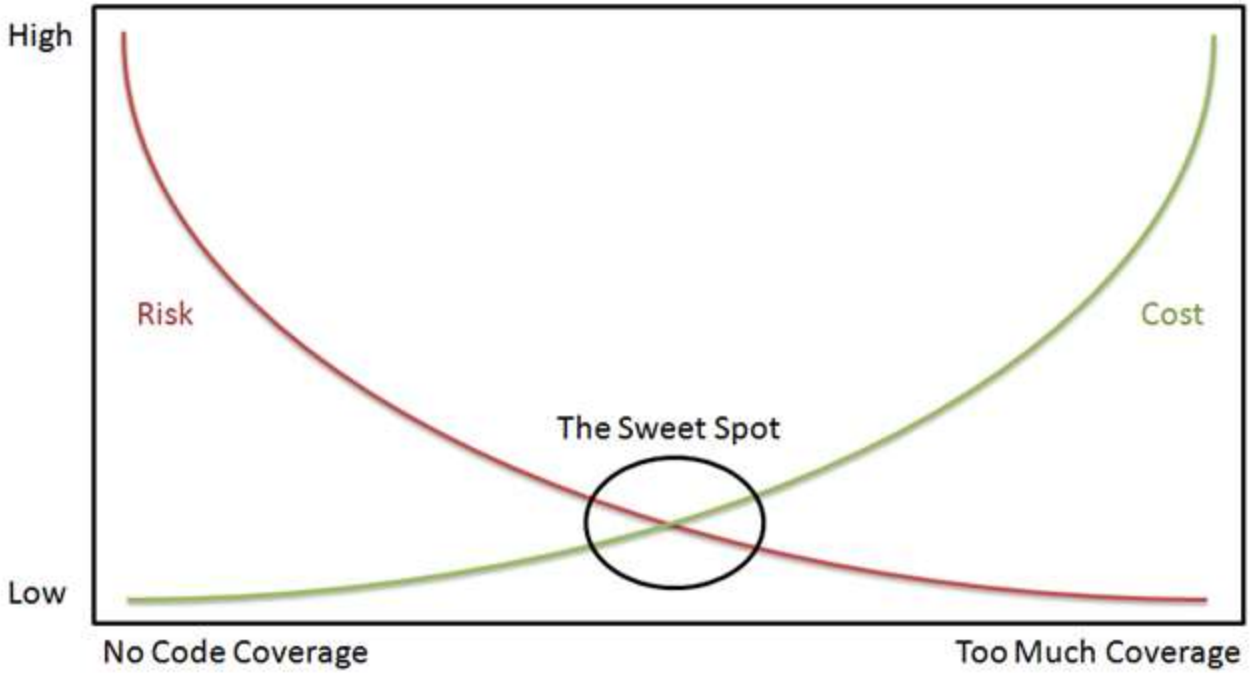
- Ayende: “But I think that even a test has got to justify its existence, and in many cases, I see people writing tests that have no real meaning. They duplicate the logic in a single class or method.”
  - http://ayende.com/blog/4217/even-tests-has-got-to-justify-themselves (refs)
- Fowler, CannotMeasureProductivity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

# How Agile Are You? 2010 Survey

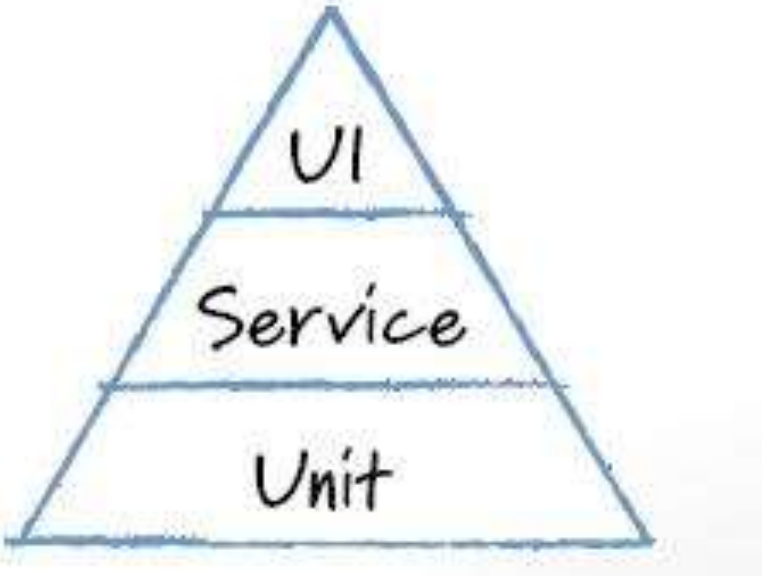


# Cost vs. Risk In Testing

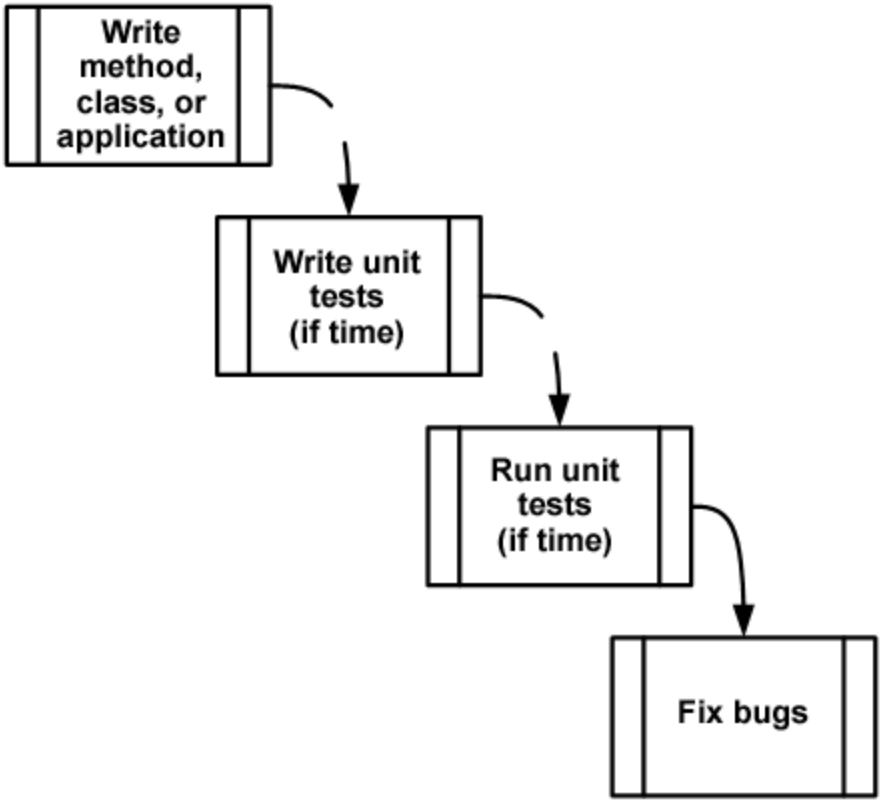
- האם צריך תמיד כיסוי של 100% של בדיקות יחידה, 100% בדיקות אינטגרציה ו-100% בדיקות קצה?



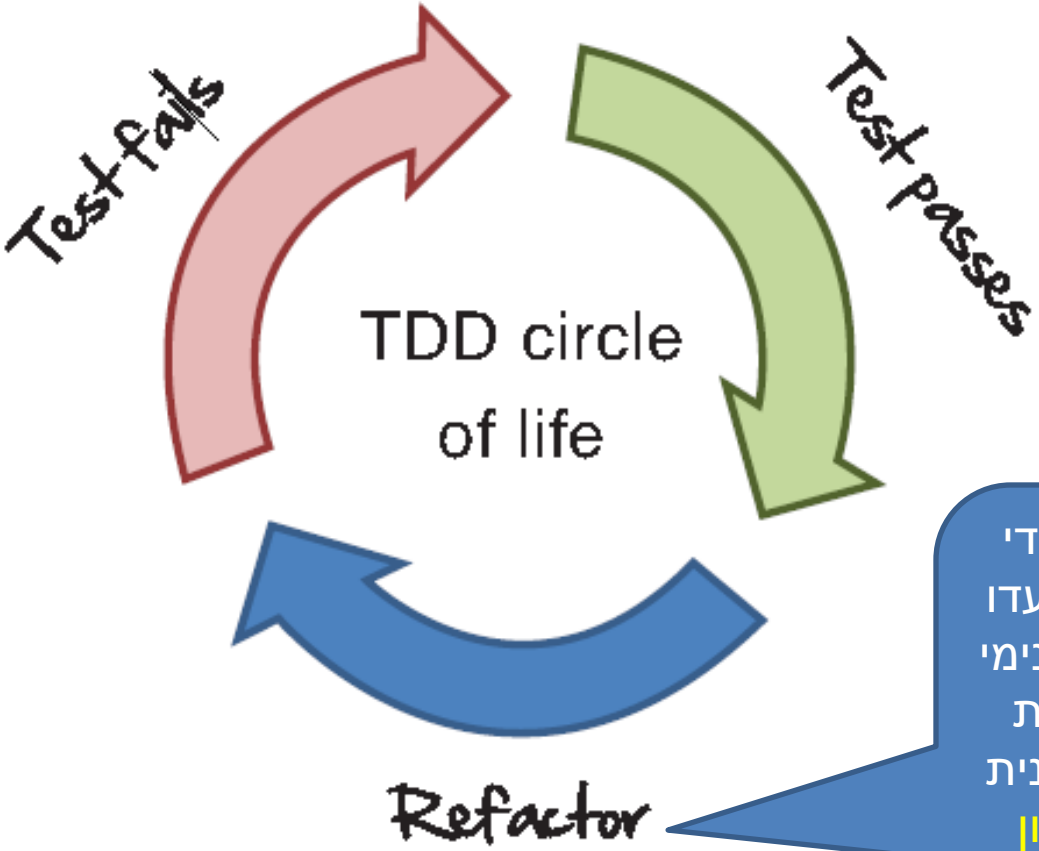
# The Forgotten Layer of the Test Automation Pyramid (also)



# הדרך המסורתית לבדיקות (יחידה)



# TDD = TFD + Refactoring



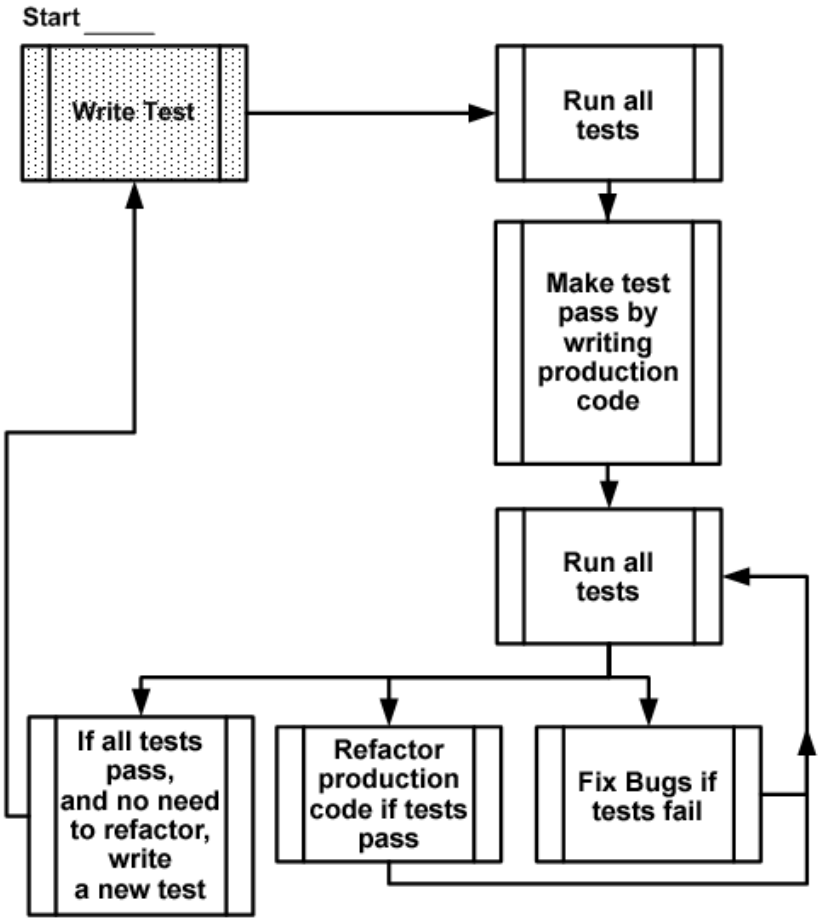

שיפור קוד קיים, על ידי שימוש בטכניקות שנועדו לשפר את המבנה הפנימי של הקוד מבלי לשנות את ההתנהגות החיצונית שלו (ויקיפדיה), תיכון מתמשך

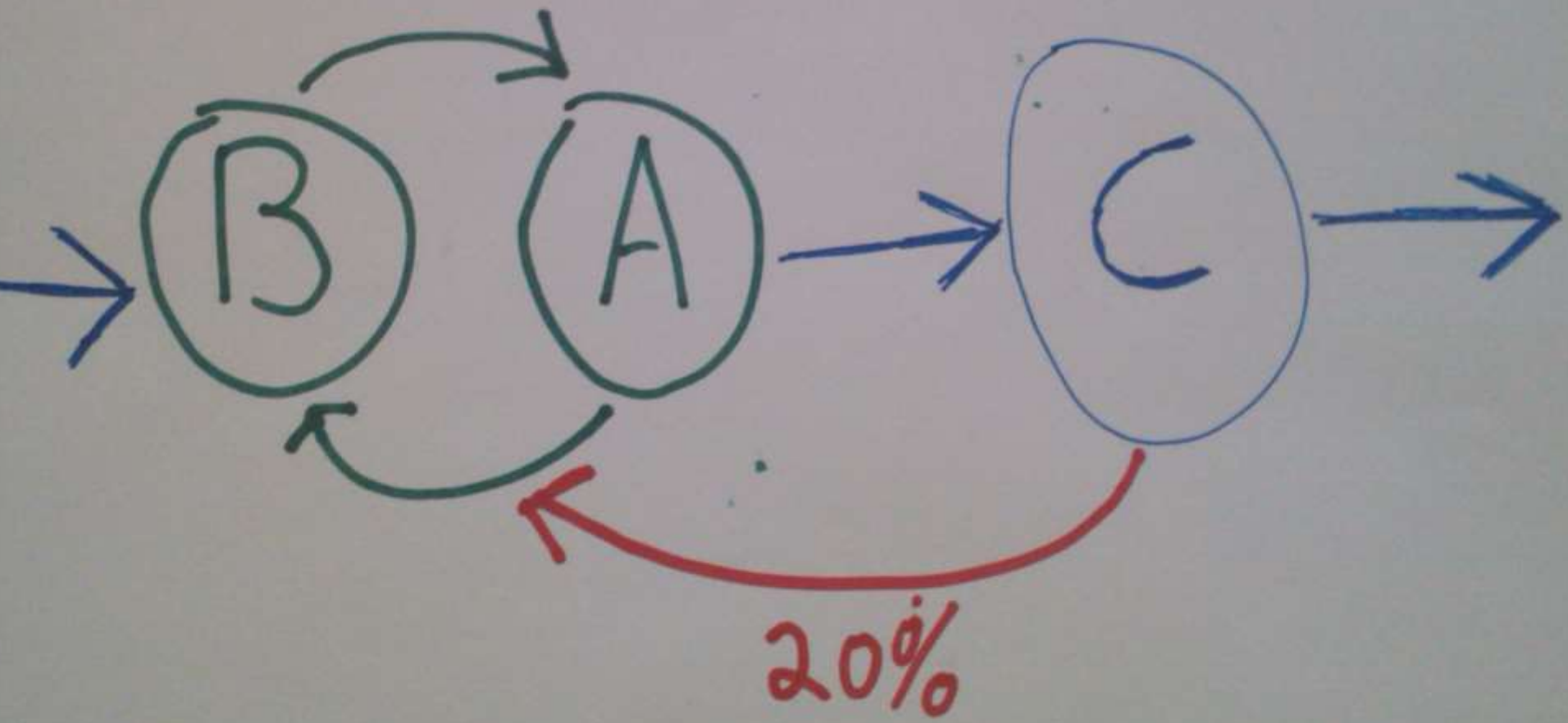
# TDD – R. Martin

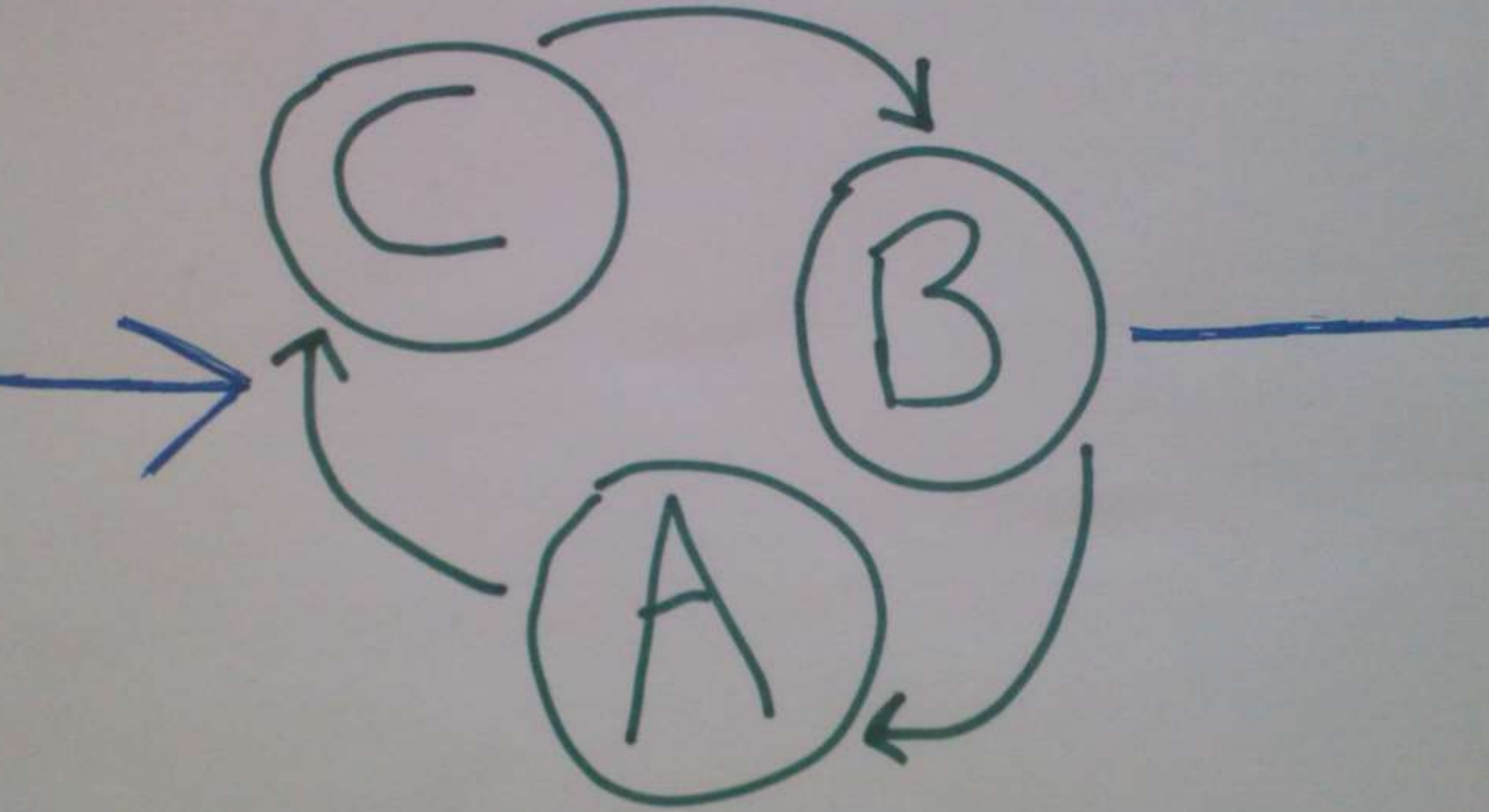
Over the years I have come to describe Test Driven Development in terms of three simple rules. They are:

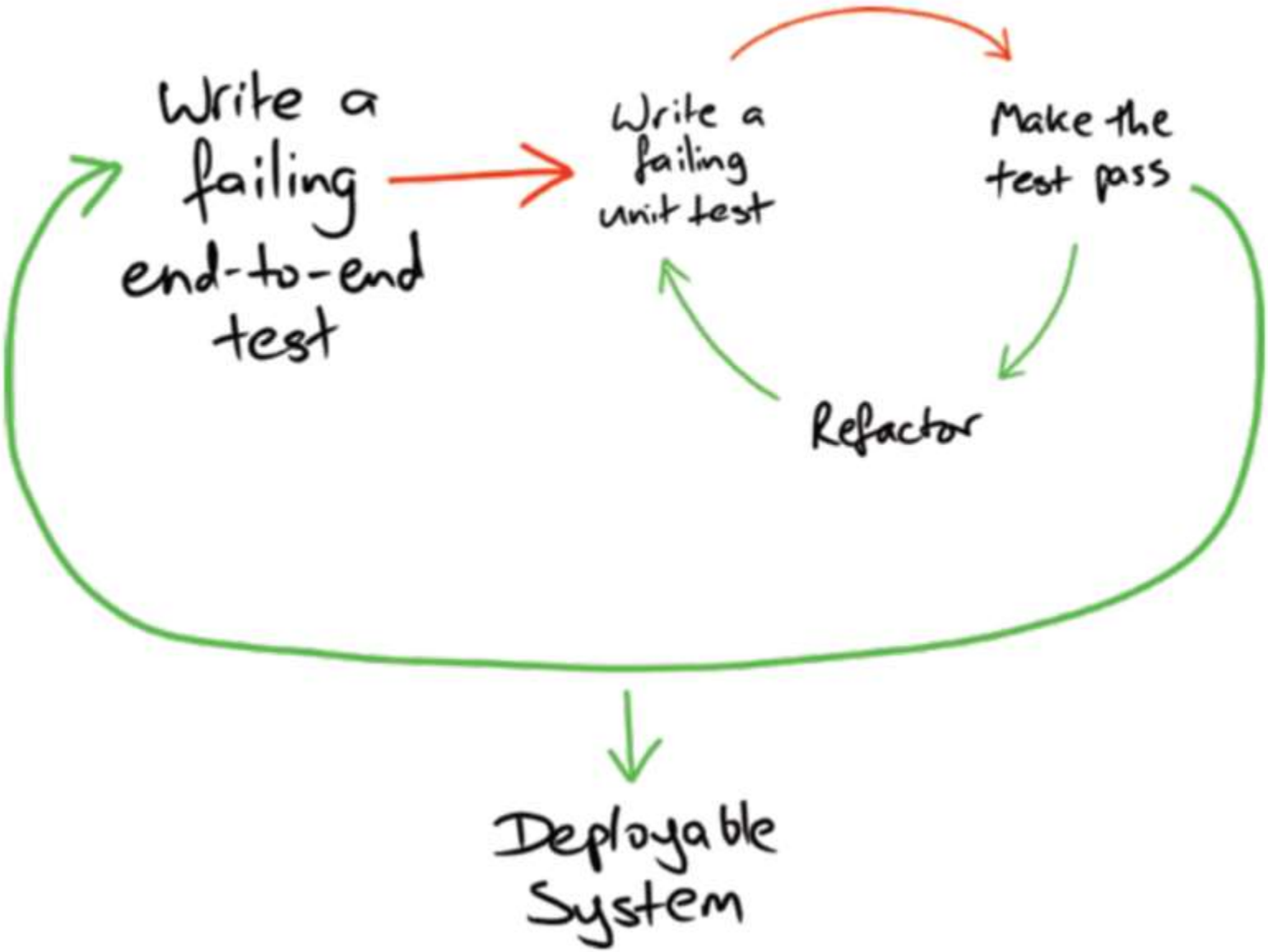
1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to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

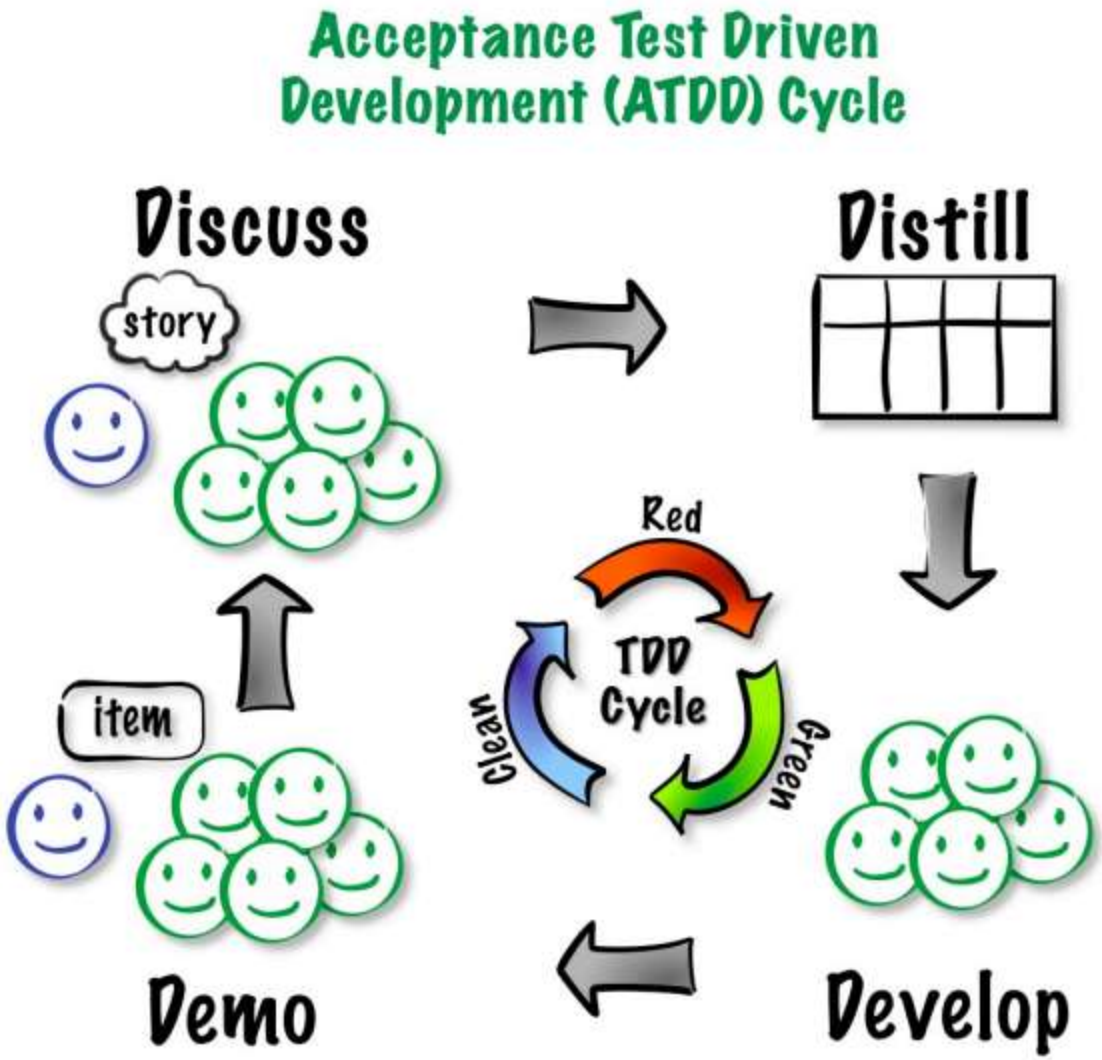
# TDD Cycle


Start
Write Test
Run all tests
Make test pass by writing production code
Run all tests
If all tests pass, and no need to refactor, write a new test
Refactor production code if tests pass
Fix Bugs if tests fail


B
A
C
20%

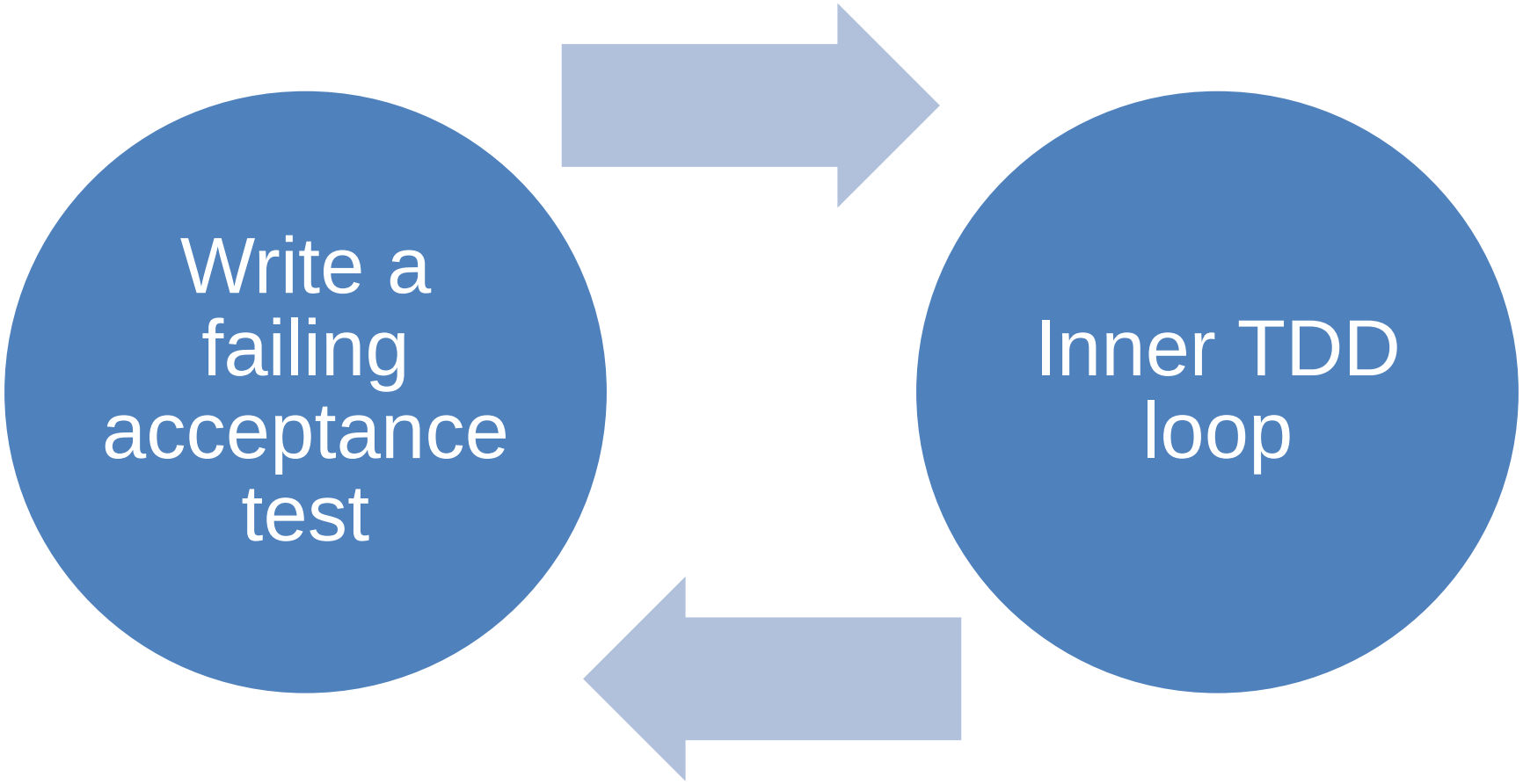

C
B
A


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




*(ATDD cycle model developed by James Shore with changes suggested by Grigori Melnick, Brian Marick, and Elisabeth Hendrickson.)*

# Inner & Outer feedback loops



# כמה Refactoring?

- 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)

* Emerson Murphy-Hill and Andrew Black in "Refactoring Tools: Fitness for Purpose" http://web.cecs.pdx.edu/~black/publications/IEEESoftwareRefact.pdf

# איכות פנימית וחיצונית

- בדיקות קצה-> חיצוני
- בדיקות יחידה -> פנימי
- בדיקות אינטגרציה – באמצע?

# בדיקת יחידה טובה (אושרוב)

- בדיקת יחידה היא קוד שקורא לקוד אחר ובודק אח"כ נכונות של טענות מסוימות על ההתנהגות הלוגית של מתודה או מחלקה.
- בדיקת יחידה תכתב בד"כ באמצעות framework
- קצרה ומורצת בקלות
- ניתנת לאוטומציה, אמינה, קריאה וקלה לתחזוקה

Readable, Trustworthy, Fast, Maintainable

# Unit tests should be FIRST

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

# Unit tests should be FIRST

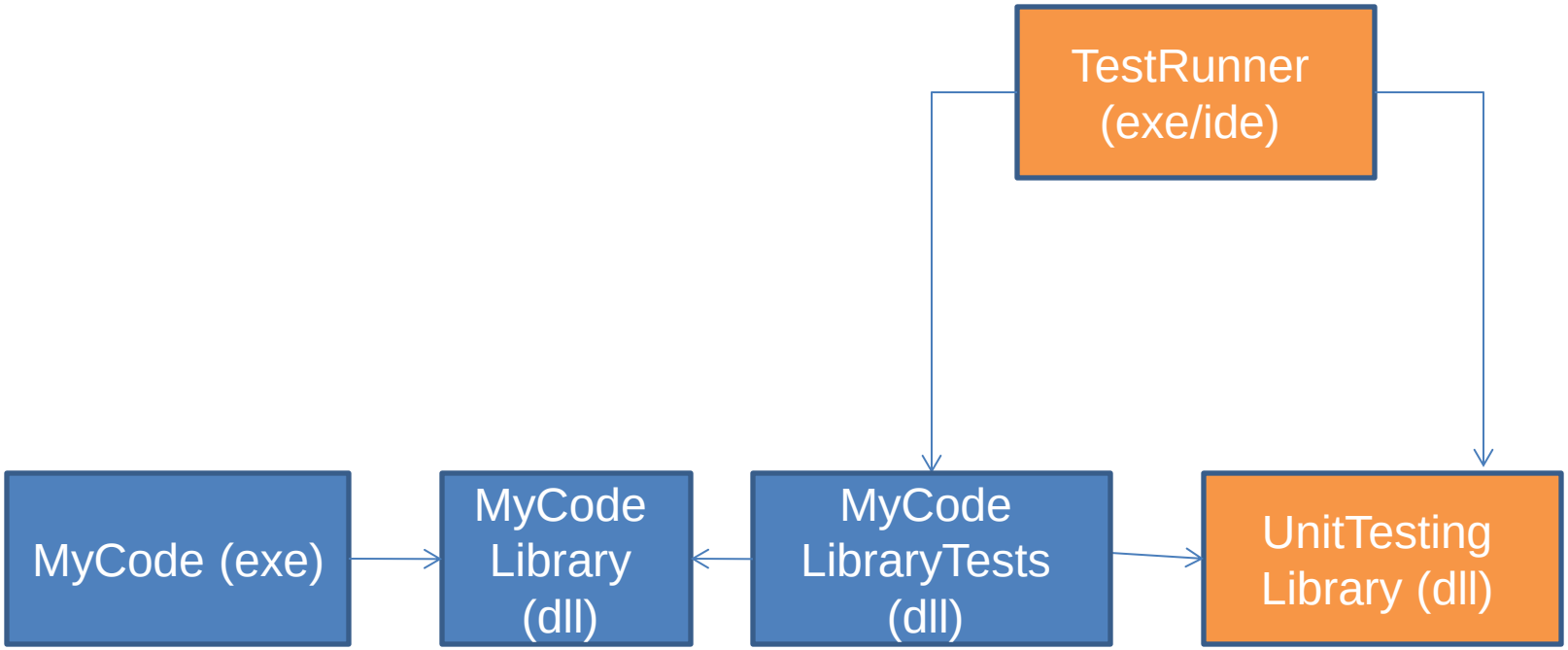
(adopted from A. Fox, Berkeley)

- **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
- 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)

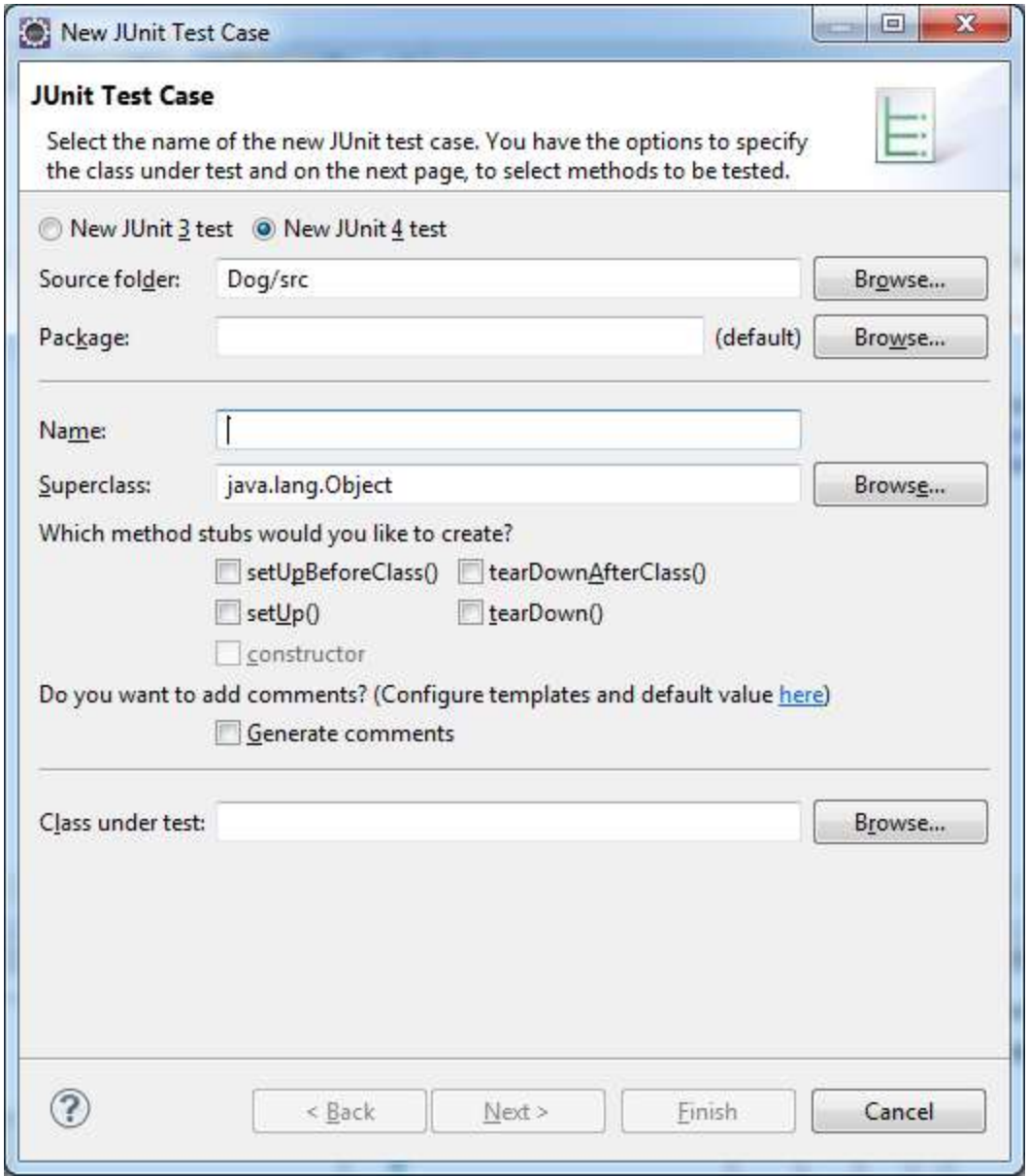
```
// SUT
public class Calc {
 public int add(int a, int b) {
  return a+b;
 }
}
// Unit Test
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalcTest {
 @Test
 public void testAdd() {
  int result = new Calc().add(2, 3);
  assertEquals(5,result);
 }
}
```

# רכיבים עיקריים בקוד בדיקה (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!");
    }
}
```

# הדגמה ראשונית



# ספרות רומיות

# Spec.

- https://gist.github.com/4187205
- באיזה סדר דוגמאות כדאי לעבוד?

- Roman Numerals Kata with Commentary using Transformation Priority Premise & the 4 simple design rules

# Java - כלים

- 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 (?)

# כלים - .Net

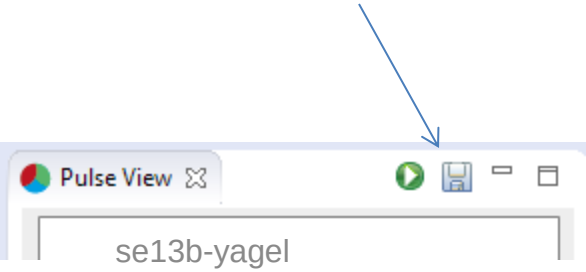
- DevEnv: MS Visual Studio 2012 (express), (nuget)
- UnitTesting: Nunit, (/MsTest), Runners: TestDriven.Net (/Resharper), Cont. Testing: AutoTest.Net, Giles
- תזכורת לבקרת קוד (בסביבת העבודה):
- VCS: git, gitextensions (VS: Tools->Options->SourceControl->Current) (without nuget packages)

# בפעם הבאה \ 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 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 2 (חובה) – בדיקות יחידה + TDD

- השלמת תרגיל הספרות הרומיות
- אופציה: שכפול מאגר הקוד (fork) מההרצאה (קישור ביומן) לחשבון שלכם ב-github, יבוא לסביבת הפיתוח (עם egit או עם לקוח git חיצוני ויבוא ממערכת הקבצים, אבל אפשר גם בסביבה\שפה אחרת)
- מומלץ לתת לבדיקות להוביל את פיתוח האלגוריתם
- אחרי כל צעד (red-green-refactor) יש לבצע commit עם הערה שמתחילה בסוג הצעד (למשל: RED: Deal with 9)
- דחיפה ל-github <u>והגשת הקישור למאגר</u> (בונוס: תיעוד pulse ו-TDGotchi)
- ציון: כיסוי, RGR, נכונות, איכות.
- אופציה: זוגות מתחלפים (קידוד ובדיקה)



# לסיכום

- בדיקות, בדיקות יחידה ופיתוח מונחה בדיקות
- סבב 2: חליפת בדיקות לרכיב מרכזי
- בדיקות ותיכון מתמשך
- Red-Green-Refactor
- xUnit + כלים
- מצריך לימוד מתמשך – אז למה עכשיו?
  - John Gall: “A complex system that works is invariably found to have evolved from a simple system that works.”

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