תכנות מונחה בדיקות
(Test Driven Development)

תכנות מתקדם בשפת Java

אוהד ברזילי

אוניברסיטת תל אביב

המצגת מבוססת על הספר:

Test-Driven Development By Example

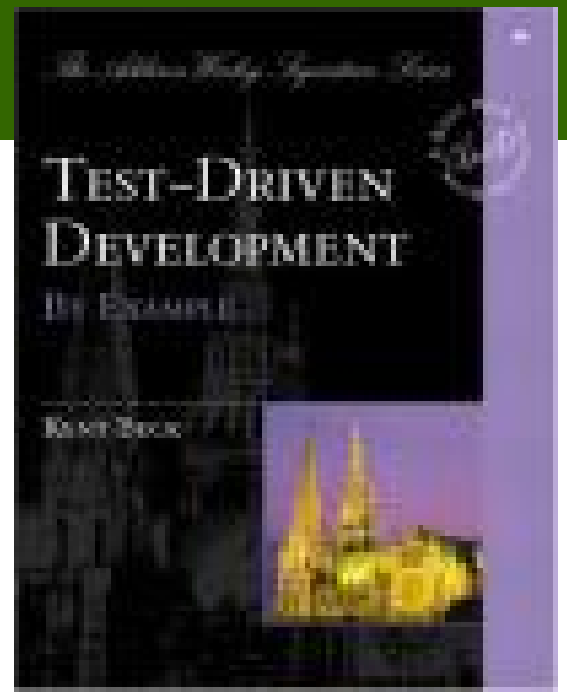
By Kent Beck

Publisher: Addison Wesley

Date: November 08, 2002

ISBN: 0-321-14653-0

Pages: 240

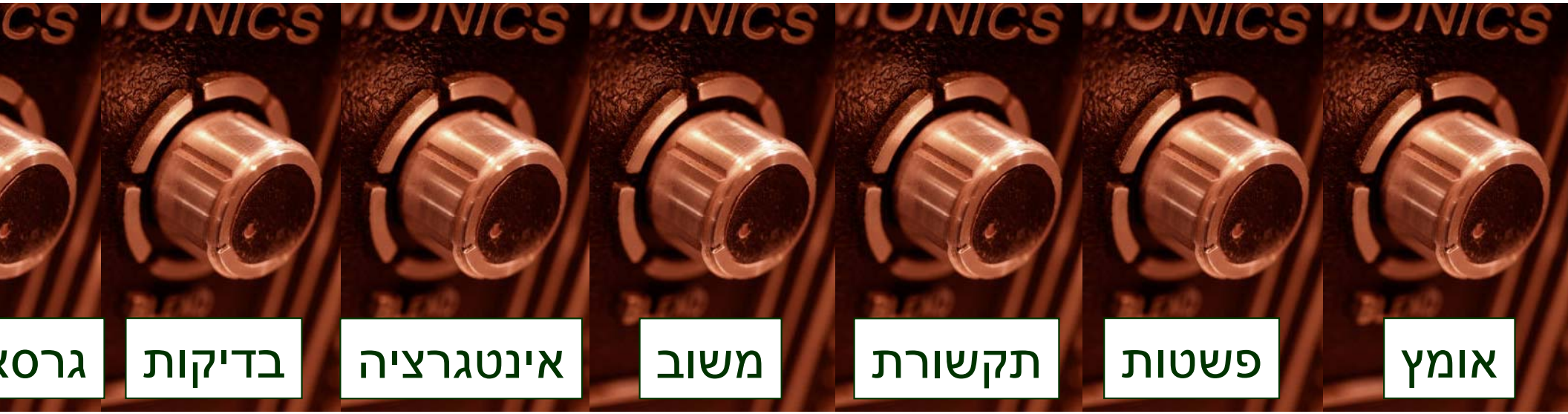


היום בשיעור

- XP כרקע לפיתוח גישת התכנות מונחה הבדיקות
- עקרונות הגישה
- דוגמא מודרכת - Fibonacci
- טעימה מתוך דוגמא מודרכת – Money
- סיכום

e**X**treme **P**rogramming

- ערכים ומיומנויות שהתגלו כאפקטיבים נלקחים לקיצוניות



4 ערכים

המתודולוגיה מושתתת על 4 ערכים:

- משוב (feedback)
- פשטות (Simplicity)
- תקשורת (Communication)
- אומץ (Courage)

מערכים אלו נגזרות 12 מיומנויות תוכנה

12 מיומנויות

Code/Technical Perspective	Human/Social Perspective
Refactoring	Collective ownership
Simple design	Pair programming
Coding standards	Sustainable pace
Test First	On-site customer
Continuous integration	Planning game
Small releases	Metaphor

The 12 XP practices

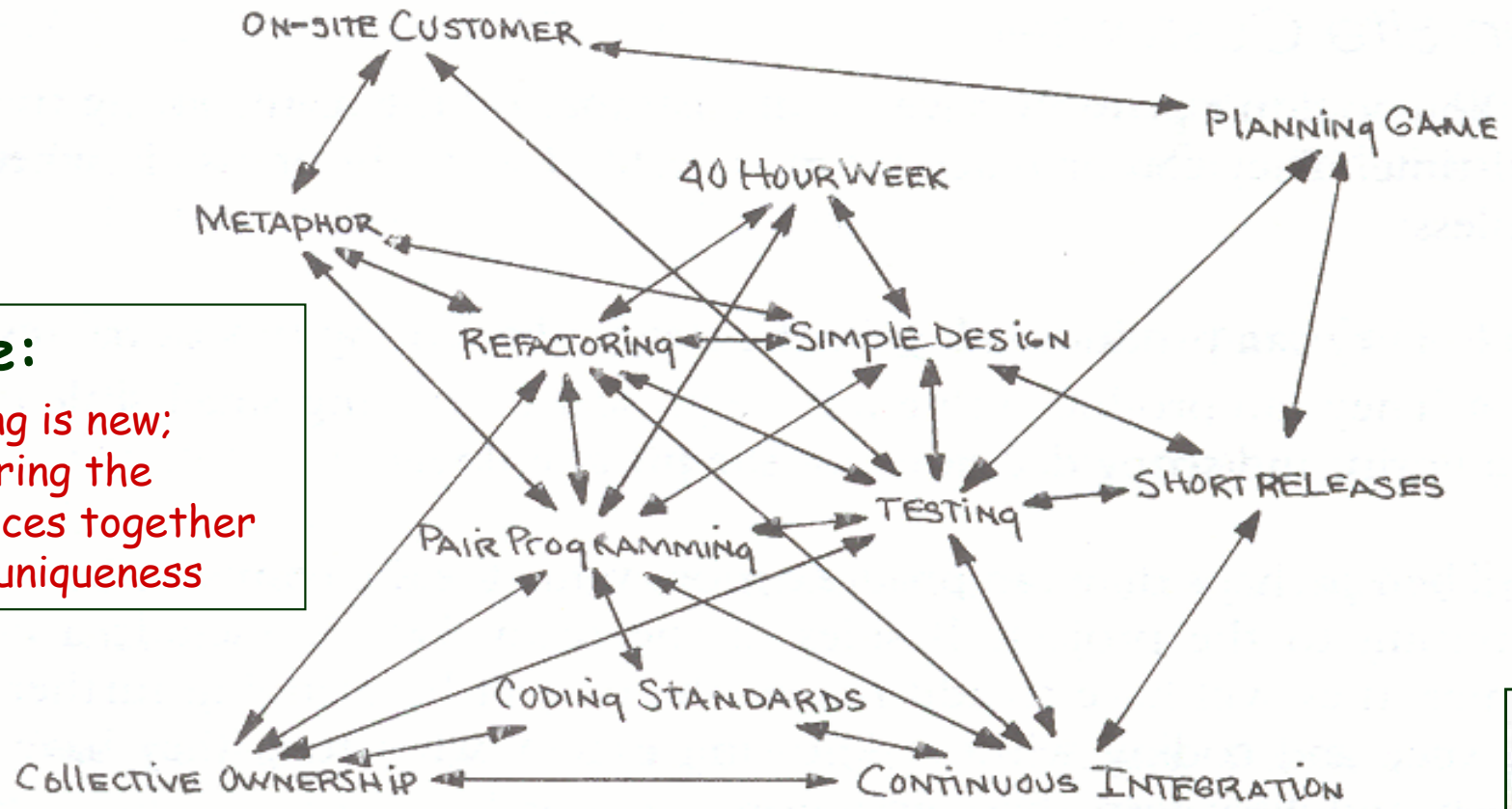


FIGURE 4. The practices support each other

Note:
nothing is new; gathering the practices together is XP uniqueness

Source: Beck, K. (2000). *eXtreme Programming explained*, Addison Wesley.

בדיקות תחילה

- קוד נקי שעובד (clean code that works)
  - קוד חדש נכתב רק אחרי שבדיקה אוטומטית נכשלת
  - הסרת כפילויות

- השלכות טכניות
  - התכן (design) מלווה בקידוד
  - המתכנת כותב את הבדיקות
  - קומפילציה מהירה
  - צימוד חלש בין רכיבים

אדום – ירוק - שכתב

- אדום
  - כתוב בדיקה שנכשלת (אולי אפילו לא עוברת קומפילציה)
- ירוק
  - תעשה במהירות שהבדיקה תצליח (תוך אולי שחיטת פרות קדושות של עקרונות תכנות נכונים)
- שכתוב (refactoring)
  - הסר את הכפילויות בקוד שכראה הכנסת בשלב הקודם

אומץ

If pain is nature's way of saying "Stop!" then fear is nature's way of saying "Be careful."

Fibonacci

- נתחיל בדוגמא פשוטה
- ברצוננו לכתוב פונקציה המחשבת איבר בסדרת פיבונאצ'י
- נכתוב את הפונקציה בגישה של Test First

שלבי העבודה

1. Quickly add a test.
2. Run all tests and see the new one fail.
3. Make a little change.
4. Run all tests and see them all succeed.
5. Refactor to remove duplication.

איך מתחילים?

- התכנות שלנו מונע מסיפורים (תסריטים)
  - "נרצה שאפשר יהיה לבצע במערכת..."
- מה נרצה שתעשה הפונקציה?
- ניצור מחלקה ששם יישב קוד הבדיקה

```
public class TestFib extends TestCase {
   public void testFibonacci() {
          assertEquals(0, fib(0));
   }
}
```

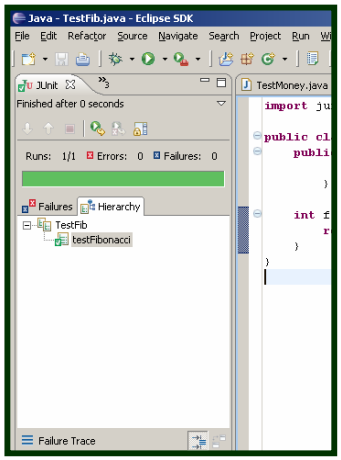
TODO List:
fib(0) == 0

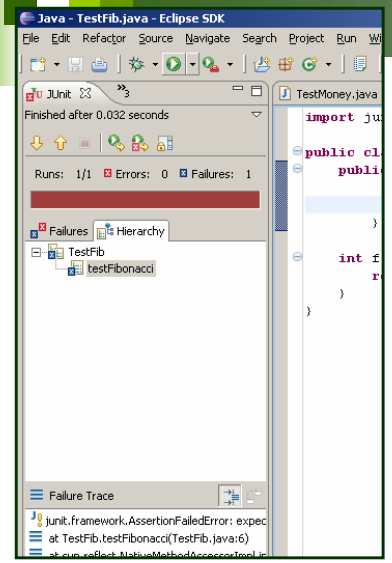
תעשה שיתקמפל

- נוסיף קוד מינימלי כדי לפתור את בעיית הקומפילציה

```
int fib(int i){
      return 0;
}
```

- נריץ... (את קוד הבדיקה)
- ירוק





נוסיף עוד בדיקה

- אפשר להוסיף עוד מתודת בדיקה חדשה:

```
public void testFibonacciOfOneIsOne() {
   assertEquals(1, fib(1));
}
```

- אנחנו נסתפק ב:

```
public void testFibonacci() {
   assertEquals(0, fib(0));
   assertEquals(1, fib(1));
}
```

- נריץ... אדום

TODO List:

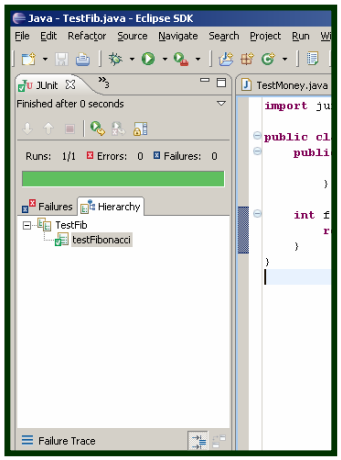
~~fib(0) == 0~~

fib(1) == 1

תעשה שיהיה ירוק

- נוסיף קוד מינימלי כדי להפוך את הפס לירוק

```
int fib(int n) {
   if (n == 0)
       return 0;
   return 1;
}
```



- נריץ... (את קוד הבדיקה)
- ירוק

TODO List:

~~fib(0) == 0~~

fib(1) == 1

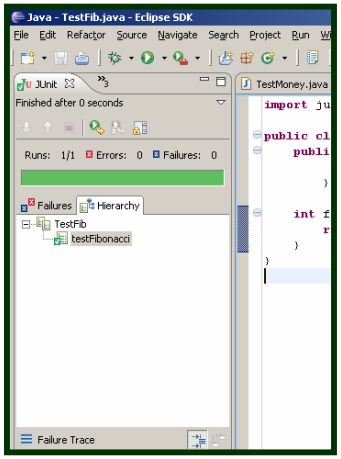
הסרת כפילויות

- הכפילויות הפעם הן בבדיקה (ולא בקוד) – נסיר אותן (refactoring)

```
public void testFibonacci() {
   int cases[][]= {{0,0},{1,1}};
   for (int i= 0; i < cases.length; i++)
       assertEquals(cases[i][1], fib(cases[i][0]));
}
```

- נוודא שלא הרסנו כלום
(או שלא גילינו באג חדש)...

ירוק

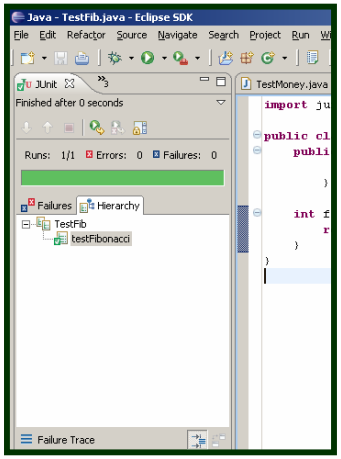


נוסיף עוד בדיקה

- קל להוסיף את הבדיקה בפונקציה הבדיקה המשוכתבת (6 הקשות מקלדת בלבד!)

```
public void testFibonacci() {
   int cases[][]= {{0,0},{1,1},{2,1}};
   for (int i= 0; i < cases.length; i++)
        assertEquals(cases[i][1], fib(cases[i][0]));
}
```

- נריץ...
- עדיין ירוק



TODO List:

~~fib(0) == 0~~

~~fib(1) == 1~~

fib(2) == 1

ואחת לשנה הבאה

- רק בשביל להיות בטוחים שסיימנו...

```
public void testFibonacci() {
   int cases[][]= {{0,0},{1,1},{2,1},{3,2}};
   for (int i= 0; i < cases.length; i++)
       assertEquals(cases[i][1], fib(cases[i][0]));
}
```

- נריץ... אדום (שזה טוב, גילינו באג!)

TODO List:

~~fib(0) == 0~~

~~fib(1) == 1~~

~~fib(2) == 1~~

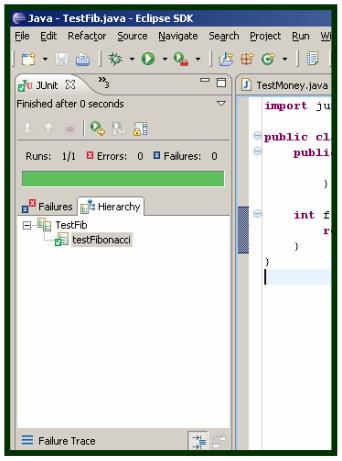
fib(3) == 2

תעשה שיהיה ירוק

- נוסיף קוד מינימלי כדי להפוך את הפס לירוק

```
int fib(int n) {
   if (n == 0) return 0;
   if (n <= 2) return 1;
   return 2;
}
```

שכתוב:
מאיפה הגיע ה-2 ?
זהו בעצם 1+1



תכנות מתקדם בשפת Java
אוניברסיטת תל אביב

- נריץ... ירוק

TODO List:
~~fib(0) == 0~~
~~fib(1) == 1~~
~~fib(2) == 1~~
fib(3) == 2

שכתוב

- קיבלנו:

```
int fib(int n) {
   if (n == 0) return 0;
   if (n <= 2) return 1;
   return 1 + 1;
}
```

- ה-1 הראשון הוא בעצם fib(n-1)

```
int fib(int n) {
   if (n == 0) return 0;
   if (n <= 2) return 1;
   return fib(n-1) + 1;
}
```

- ה-1 השני הוא בעצם fib(n-2)

שכתוב

- ה-1 השני הוא בעצם fib(n-2)

```
int fib(int n) {
   if (n == 0) return 0;
   if (n <= 2) return 1;
   return fib(n-1) + fib(n-2);
}
```

- נכליל עבור fib(2), וסיימנו

```
int fib(int n) {
   if (n == 0) return 0;
   if (n == 1) return 1;
   return fib(n-1) + fib(n-2);
}
```

דוגמא נוספת

- מערכת אמיתית לניהול תיק השקעות
  c2.com/doc/oopsla91.html
- נרצה לממש מחלקה שמייצגת כסף

Instrument	Shares	Price	Total
IBM	1000	25	25000
GE	400	100	40000
		Total	**65000**

דוגמא מודרכת

- לאחר כמה שנים נתבקש המתכנת לתמוך בעוד מטבעות (פרט ל Dollar)

Instrument	Shares	Price	Total
IBM	1000	25 USD	25000 USD
Novartis	400	150 CHF	60000 CHF
		Total	**65000 USD**

From	To	Rate
CHF	USD	1.5

TODO List:
$5 + 10 CHF = $10 if rate is 2:1
$5 * 2 = $10

מתחילים

- התכנות שלנו מונע מסיפורים (תסריטים)
  - "נרצה שאפשר יהיה לבצע במערכת..."
- בשלב ראשון נתאר את התסריט במונחים של עצמים מטיפוס Dollar

```
public void testMultiplication() {
    Dollar five= new Dollar(5);
    five.times(2);
    assertEquals(10, five.amount);
}
```

- ה- test מרמז על design נוראי:
  - public fields
  - side-effects
  - integers for monetary amounts

תרגיעי את הקומפיילר שלך

TODO List:

$5 + 10 CHF = $10 if rate is 2:1

$5 * 2 = $10

Make "amount" private

Dollar side-effects?

Money rounding?

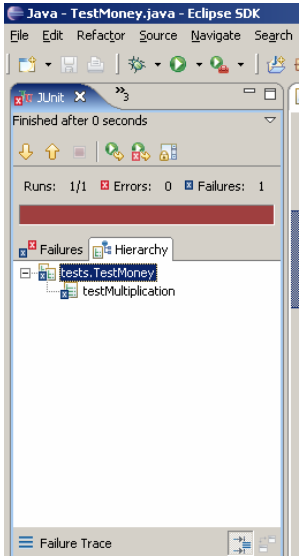
- ~~No class Dollar~~
- ~~No constructor~~
- ~~No method times(int)~~
- ~~No field amount~~

```
class Dollar {

  Dollar(int amount) { }

  void times(int multiplier)
  { }

  int amount;
}
```



ירוק עכשיו

TODO List:
$5 + 10 CHF = $10 if rate is 2:1
$5 * 2 = $10
Make "amount" private
Dollar side-effects?
Money rounding?

- תיקון מינימלי
- מה יש לשכתב כאן?
  - אין בקוד שכפול קוד
  - אבל יש תלות בין קוד הבדיקה והקוד הנבדק
- אם ה- 10 היה נכתב כ- 2*5 התלות היתה בולטת יותר

```
class Dollar {

  Dollar(int amount) { }

  void times(int multiplier)
  { }

  int amount = 10 * 2 ;
}
```

הסרת תלות

TODO List:
$5 + 10 CHF = $10 if rate is 2:1
$5 * 2 = $10
Make "amount" private
Dollar side-effects?
Money rounding?

- לאט, לאט
- התלות עברה ל times
- איך ניפטר מה- 5 ?

```
// Dollar.java
int amount;

void times(int multiplier) {
    amount= 5 * 2;
}

Dollar(int amount) {
   this.amount= amount;
}
```

הסרת תלות

TODO List:
$5 + 10 CHF = $10 if rate is 2:1
$5 * 2 = $10
Make "amount" private
Dollar side-effects?
Money rounding?

```
// Dollar.java
int amount;

void times(int multiplier) {
    amount= amount * 2;
}

Dollar(int amount) {
   this.amount= amount;
}
```

- לאט, לאט
- התלות עברה ל times
- איך ניפטר מה- 5 ?
- איך ניפטר מה- 2 ?

הסרת תלות

TODO List:
$5 + 10 CHF = $10 if rate is 2:1
$5 * 2 = $10
Make "amount" private
Dollar side-effects?
Money rounding?

```
// Dollar.java
int amount;

void times(int multiplier) {
    amount= amount * multiplier;
}

Dollar(int amount) {
   this.amount= amount;
}
```

- לאט, לאט
- התלות עברה ל times
- איך ניפטר מה- 5 ?
- איך ניפטר מה- 2 ?
- amount מופיע פעמיים

הסרת תלות

TODO List:
$5 + 10 CHF = $10 if rate is 2:1
~~**$5 * 2 = $10**~~
Make "amount" private
Dollar side-effects?
Money rounding?

```
// Dollar.java
int amount;

void times(int multiplier) {
    amount *= multiplier;
}

Dollar(int amount) {
   this.amount= amount;
}
```

- לאט, לאט
- התלות עברה ל times
- איך ניפטר מה- 5 ?
- איך ניפטר מה- 2 ?
- amount מופיע פעמיים

סיכום ביניים

- Made a list of the tests we knew we needed to have working
- Told a story with a snippet of code about how we wanted to view one operation
- Made the test compile with stubs
- Made the test run by committing horrible sins
- Gradually generalized the working code, replacing constants with variables
- Added items to our to-do list rather than addressing them all at once

בהקשר כללי יותר

- Write a test.
- Make it run
- Make it right

Write a test

- Think about how you would like the operation in your mind to appear in your code.
- You are writing a story. Invent the interface you wish you had.
- Include all of the elements in the story that you imagine will be necessary to calculate the right answers.

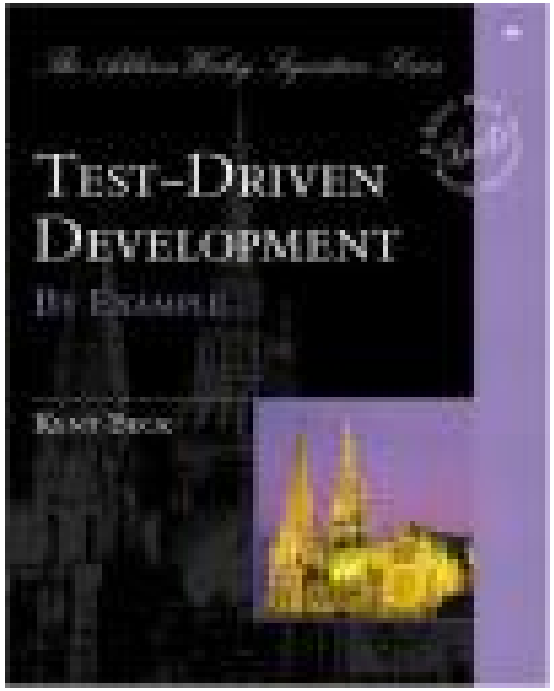
Make it run

- Quickly getting that bar to go to green dominates everything else
- If a clean, simple solution is obvious, then type it in
- If the clean, simple solution is obvious but it will take you a minute, then make a note of it and get back to the main problem, which is getting the bar green in seconds
- This shift in aesthetics is hard for some experienced software engineers
- They only know how to follow the rules of good engineering
- Quick green excuses all sins. But only for a moment

Make it right

- Now that the system is behaving, put the sinful ways of the recent past behind you
- Step back onto the straight and narrow path of software righteousness
- Remove the duplication that you have introduced, and get to green quickly

רוצים עוד?



עוד TDD

עוד XP