

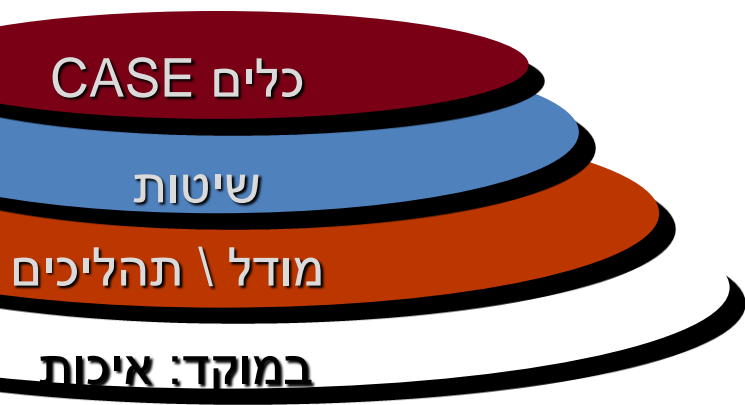


הנדסת תוכנה

8. בדיקות - II

הדגמה, בדיקות יחידה

[Pragmatic Programmer Tip](#) : **Test Early. Test Often.**
Test Automatically. Tests that run with every build are much more effective than test plans that sit on a shelf.

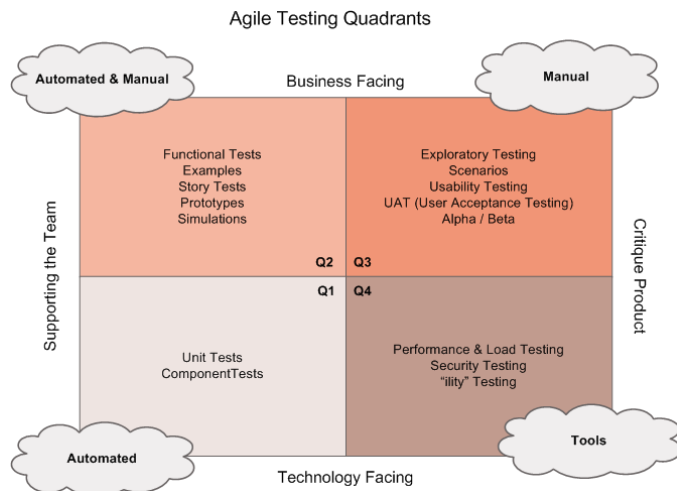


מה היום?

- בדיקות (חזרה ודיון)
- TDD - דיון
- הדגמה
 - שב4+
- פרויקט: סיכום סבב:
 - מצגת סבב
 - סקר קוד קצר
 - מצב המשימות
 - תכנון לסבב הבא
 - רישום לסקר
 - סקר עמיתים
- תזכורת: מאגר שאלות לבחינה

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?**

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

בדיקות יחידה (אוטומטיות)



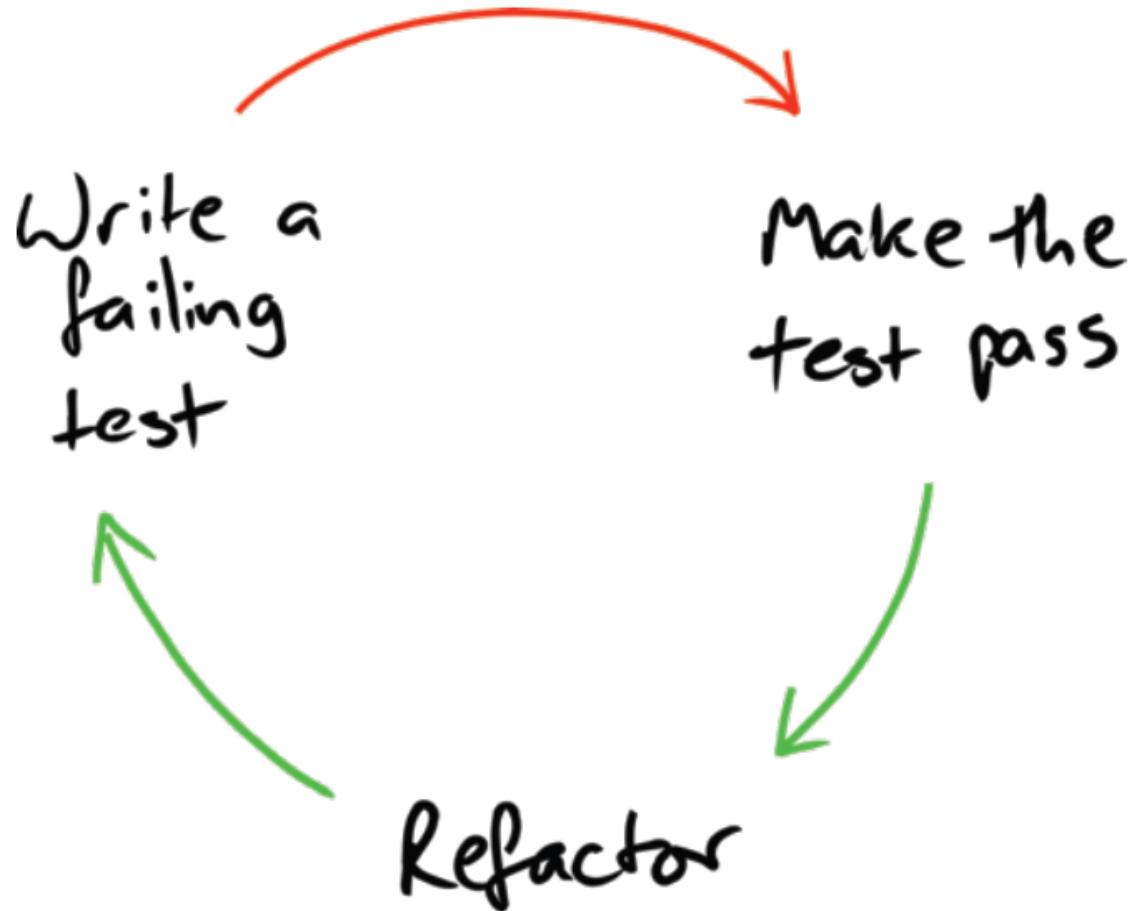
• יתרונות

- 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
- לא תמיד קל עבור קוד קיים (legacy)

TEST DRIVEN DEVELOPMENT



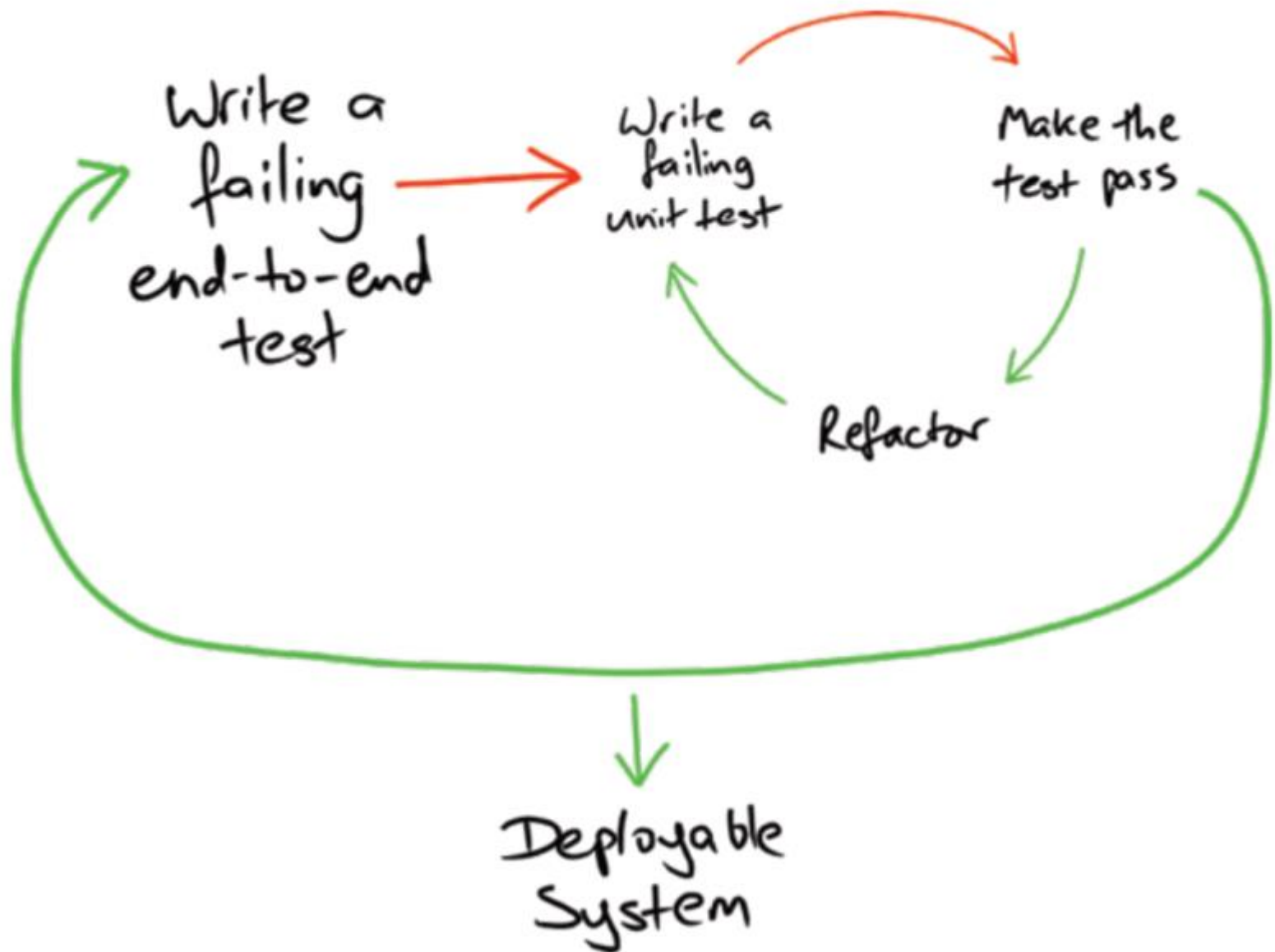
TDD יתרונות

- 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



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

- ארכיטקטורה סטנדרטית לבדיקות יחידה

לסיכום

- בדיקות, פיתוח מונחה התנהגות
- בפרויקט

– בכל סבב: ניסוח בדיקת קבלת לתרחיש עיקרי

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