בדיקות תוכנה

שחר מעוז

בית הספר למדעי המחשב
אוניברסיטת תל אביב

איך יודעים שמודול או תוכנית נכונים?

- **אימות** (verification) הוא תהליך שמיועד לוודא באופן פורמאלי או לא פורמאלי נכונות של מודול או תוכנית ביחס לחוזה או אפיון (specification)

- **אימות פורמאלי אוטומאטי** אינו תמיד אפשרי (במקרה הכללי הוא לא כריע). למרות זאת קיימים כלים פורמליים, שלעיתים אינם מצליחים (הם sound אבל לא complete).

- **אימות פורמאלי ידני** יקר מדי לרוב המערכות פרט אולי למערכות שחיי אדם תלויים בהן ישירות (רפואיות, מוטסות, וכולי, אבל גם שם יש פחות אימות ממה שהיה ראוי ומכיוון שהוא ידני יכול להיות שיש בו עצמו טעויות)

- **בדיקות** (testing): ביצוע סדרת הרצות של התוכנה שמיועדות למצוא פגמים, אם יש, ולהגדיל את בטחוננו בנכונותה
  - לא מבטיח נכונות, אבל יותר טוב מכלום, **ומועיל מאוד באופן מעשי להקטנת מספר הפגמים**

אל תירה בשליח

- כאשר המכונית לא עוברת טסט, זה כמובן מעצבן, אבל זה בדרך כלל לא **כישלון** של מכון הרישוי שביצע את הטסט
- **כישלון** ו**הצלחה** של בדיקה הם נפרדים לחלוטין מאלה של הקוד הנבדק!
- בדיקה **מצליחה** אם היא מגלה פגם
- בדיקה **נכשלת** אם היא לא מגלה פגם או מדווחת על פגם לא קיים
- אם בדיקה מדווחת על פגם נאמר שהקוד לא עבר את הבדיקה, ולא נאמר שהבדיקה נכשלה
- דווח על פגם הוא אירוע חיובי (לא משמח אולי, אבל חיובי) כי הוא מספק אפשרות לתיקון פגם לפני שהוא גורם עוד נזק

שלושה סוגי בדיקות

- **בדיקות יחידה** (unit tests) בודקות מודול בודד (שרות, מחלקה אחת או מספר מחלקות קשורות)
- **בדיקות אינטגרציה** בודקות את התוכנית כולה, או קבוצה של מודולים ביחד; מתבצעות תמיד לאחר בדיקות היחידה של המודולים הבודדים (כלומר על מודולים שעברו את בדיקות היחידה שלהם)
- **בדיקות קבלה** (acceptance tests) מתבצעות על ידי הלקוח או על ידי צוות שמתפקד בתור לקוח, לא על ידי צוות הפיתוח
- גם לאחר כניסה לשימוש, התוכנה ממשיכה למעשה להיבדק, אבל אצל משתמשים אמיתיים; רצוי שיהיה מנגנון דיווח לתקלות ופגמים שמתגלים בשלב הזה, ורצוי לתקן את הפגמים הללו

קופסאות שחורות וקופסאות פתוחות

על כל מודול תוכנה צריך לבצע שני סוגים של בדיקות יחידה:

- **בדיקות קופסה שחורה** (black-box tests)
  - בודקים את הקוד מול החוזה שהוא מבטיח לקיים
  - בדיקות קופסה שחורה לא תלויות במימוש ולכן אותו סט בדיקות תקף לכל המימושים של מנשק מסוים, גם העתידיים, ובפרט לשינויים ותיקונים במימוש הנוכחי
- **בדיקות כיסוי** (coverage tests או glass-box tests)
  - דואגות שבזמן הבדיקות, כל פיסת קוד תרוץ, ובמקרים מסוימים, תרוץ יותר בכמה צורות
  - בדיקות כיסוי צריך לעדכן כאשר מעדכנים את הקוד

איך בודקים?

- בבדיקות מעורבים שני סוגי קוד: מנועים ורכיבים חלופיים
- **מנוע** (driver) הוא קוד שמדמה לקוח של המודול הנבדק וקורא לו
- **רכיב חלופי** (stub) מחליף ספק שמשרת את המודול הנבדק
- למשל מחלקה A משתמשת ב-B שמשתמשת ב-C
- בדיקת יחידה ל-B תדמה לקוח של B ותספק מחלקה חלופית ל-C, על מנת שניתן יהיה לבדוק את B בנפרד מ-A ו-C
- רכיב חלופי צריך להיות פשוט ככל האפשר
- לפעמים הרכיב החלופי לא יכול להיות משמעותית יותר פשוט מהמודול שאותו הוא מחליף, ואז כדאי להשתמש במודול האמיתי לאחר בדיקות יסודיות שלו

בדיקות רגרסיה

- בכל פעם שמגלים פגם בתוכנה, בכל שלב של חיי התוכנה (גם לאחר שנכנסה לשימוש) יש להוסיף **בדיקה שחושפת את הפגם**, כלומר שנכשלת בגרסה עם הפגם אבל עוברת בגרסה המתוקנת

- לפעמים הבדיקה תתווסף לבדיקות הקופסה השחורה ולפעמים לבדיקות הכיסוי (אם הפגם קשור באופן הדוק למימוש ולא לחוזה)

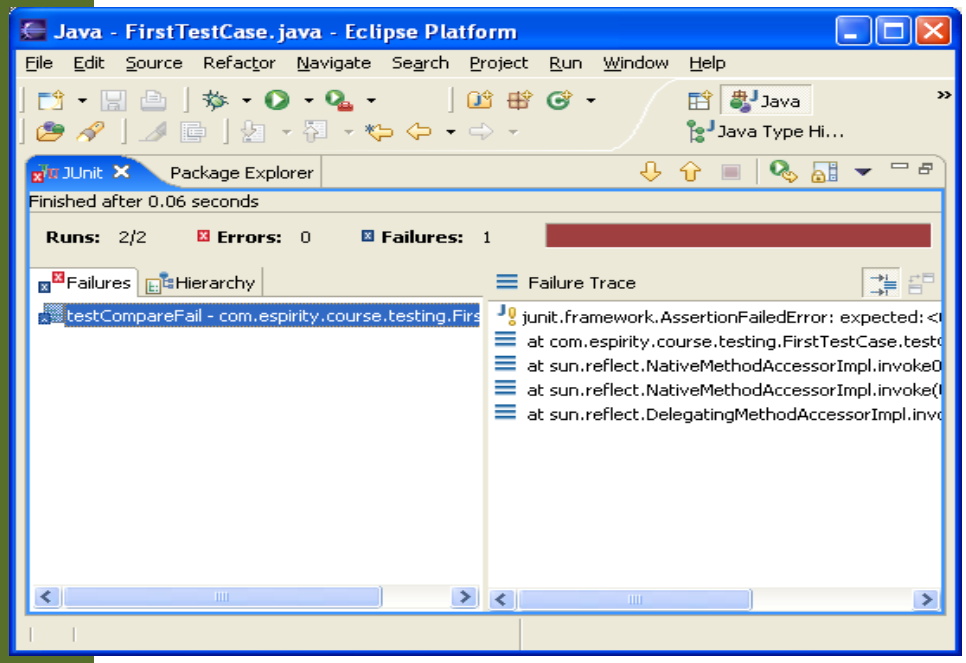
- את סט הבדיקות השלם, כולל כל הבדיקות הללו שנוצרו בעקבות גילוי פגמים, **מריצים לאחר כל שינוי** במודול הרלוונטי, על מנת לוודא שהשינוי לא גרם לרגרסיה, כלומר להופעה מחודשת של פגמים ישנים

- סט הבדיקות מייצג, כמו התוכנה המתוקנת, **ניסיון מצטבר** ויש לו ערך טכני וכלכלי משמעותי

בדיקות צריכות להיות אוטומטיות

- בדיקה שדורשת התערבות של אדם היא בדיקה לא טובה, כי קשה ויקר לחזור עליה אחרי כל שינוי בתוכנה
- לכן, כל בדיקה בדידה צריכה להיות **אוטומטית**
- צריך מנגנון (תוכנה) שמריץ את כל הבדיקות ומדווח על כל הפגמים שהתגלו
- לפעמים צריך להריץ אולי רק חלק, למשל אם ביצענו שינוי קטן בתוכנה; אבל אם הבדיקות מהירות כדאי להריץ את כולן

תמיכה בסביבת הפיתוח



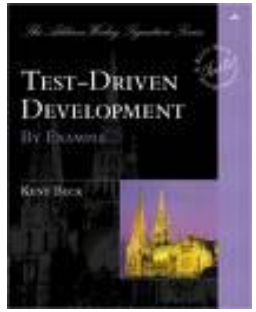
- כלים נוחים לבדיקות יחידה קיימים לכל שפות התכנות ולכל סביבות הפיתוח (JUnit, NUnit, CPPUnit), הכלים מגדירים את המושג Test Suite ליצירת סדרת בדיקות
- הסביבה מספקת מידע נוח לגבי אלו בדיקות בוצעו אילו עברו ואילו נכשלו
- קל לראות האם נזרקו חריגות ואילו
- קל לנווט בקוד ישירות למקור הבעיה
- אדום = בעיה

פיתוח מונחה בדיקות

מתודולוגיה ששמה דגש על הבדיקות כגורם המניע את התהליך.
חוזרים שוב ושוב על התהליך הבא:

- הוסף במהירות בדיקה.
- הרץ את כל הבדיקות וראה שהחדשה לא עוברת.
- בצע שינוי קטן בקוד.
- הרץ את כל הבדיקות וראה שכולם עוברות.
- בצע refactoring לביטול כפילות בקוד.

- Kent Beck, Test-Driven Development By Example, Addison-Wesley



פיתוח מונחה בדיקות

- הבדיקה **מקדימה** את המימוש!
- המימוש הנכתב הוא **מינימלי** - מטרתו לגרום לבדיקה להצליח
- היבט **פסיכולוגי**
- לכל מחלקה ולכל מתודה נכתוב מחלקת בדיקה ומתודת בדיקה.
  - לדוגמא את המתודה **`func`** של המחלקה **`MyClass`** נבדוק בעזרת המתודה **`testFunc`** של המחלקה **`TestMyClass`**