# הגדרות בעולם הפיתוח

**סוגי מחשבים**

MAIN FRAME COMPUTER

מחשב ענקי (כמעט כבר לא קיים היום), עם שפה משלו, בעיקר בבנקים. מחשב על שמשמש כשרת על ואליו מתחברים שרתים ואל השרתים האלה מתחברים המחשבים. בלילה כל המידע משרת הסניף של הבנק עובר למחשב ה MAIN FRAME. זה נועד כדי להחזיק כמות מידע עצומה וגם לשמור על הנתונים.

גם מחשב נייד, סמארטפונים, מקררים חכמים, שעונים חכמים - כולם נחשבים למחשבים.

**מערכת תוכנה/אפליקציה**

אוסף של תוכניות מחשב שנכתבות באיזישהו קוד, שמשרתות איזישהו יישום מסויים (למשל, תוכנות שנועדו לעבד תמונה של פני תת-הקרקע ולתת תמונה כוללת).

תכניות מחשב = רצץ של הוראות בשפת תכנות כלשהי, שנועדו לבצע משימה מסויימת במחשב. מאז שהאינטרנט מתקדם יותר אז נוצרו יותר שפות תכנות.

גם אפליקציה היא מערכת תוכנה או תוכנית מחשב.

**סוגי מערכות** (כל מערכת נבדלת בפוקוס שלה)

- מערכות מידע (IT - Information Technology): הפוקוס כאן הוא המידע, לכתוב DATA, לעדכן DATA, למחוק DATA. למשל, CRM, מערכת בנקאית (רושמת פעולות, מעדכנת את המידע), מערכות ביטוחיות. רוב המערכות שנתקל בהן הן מערכות מידע.
- מערכת משובצת (Embedded): מערכת שמשלבת בין חומרה לבין תוכנה. למשל, מכשיר CT, רפחן, מל״ט.
- מערכות אינטרנטיות ואפליקציות (WEB/MOBILE): לפעמים זה נופל גם תחת מערכות משובצות.
- מערכת תשתית: למשל, WINDOWS, גם דפדפן זה סוג של תשתית. המערכות שעליהן מתקינים מערכות אחרות.
- מערכת ERP: מערכת מודולרית שמשלבת כמה מערכות ביחד. למשל, תיהיה לי מערכת שמנהלת לקוחות, מערכת שמנהלת עובדים וכ״ו ביחד.
- מערכות מדף: מערכות שלא נוצרו למשתמש ספציפי. למשל, OFFICE. מערכת שאנחנו מורידים מהמדף בלי שום שינוי.

גישת הבדיקות שלי תשתנה בין מערכת למערכת. למשל, טעות באפליקצית קניות לעומת טעות במערכת CT או מל״ט.

**מחזור חיי פיתוח חדש**

1. יוזמה - של הלקוח או של המפתח. היוזמה מגיעה עם איזשהן דרישות (למשל, בעל מחסן שרוצה מערכת שתנהל לו את המלאי במחסן).
2. מנתח מערכות - הדרישות עוברות אליו. הוא מתחיל לאפיין ולתאר את המערכת בקו כללי.
3. מעצב (ארכיטקט) - כוצתב אפיון מפורט יותר, איזה קוד, אילו פונקציות, אילו סוגי נתונים.
4. מפתח - המפתח לוקח את האפיון המפורט וכותב קוד לפי האפיון.
5. בודק - המערכת עוברת לבדיקה של הבודק. אם יש בעיות זה חוזר למפתח. ככה זה הולך וחוזר כמה פעמים.

**מחזור חיי פיתוח - תחזוקה**

למשל אם מוסיפים תכונות חדשות, מעדכנים גרסאות. אז עבור כל שינוי כזה עוברים את כל השלבים הנ״ל שוב.

**מודלים של פיתוח**

בתוך מחזור חיי פיתוח יש כמה מודלים

מודל המפל

זה המודל הקלאסי

דרישות

אפיון

קידוד

בדיקות

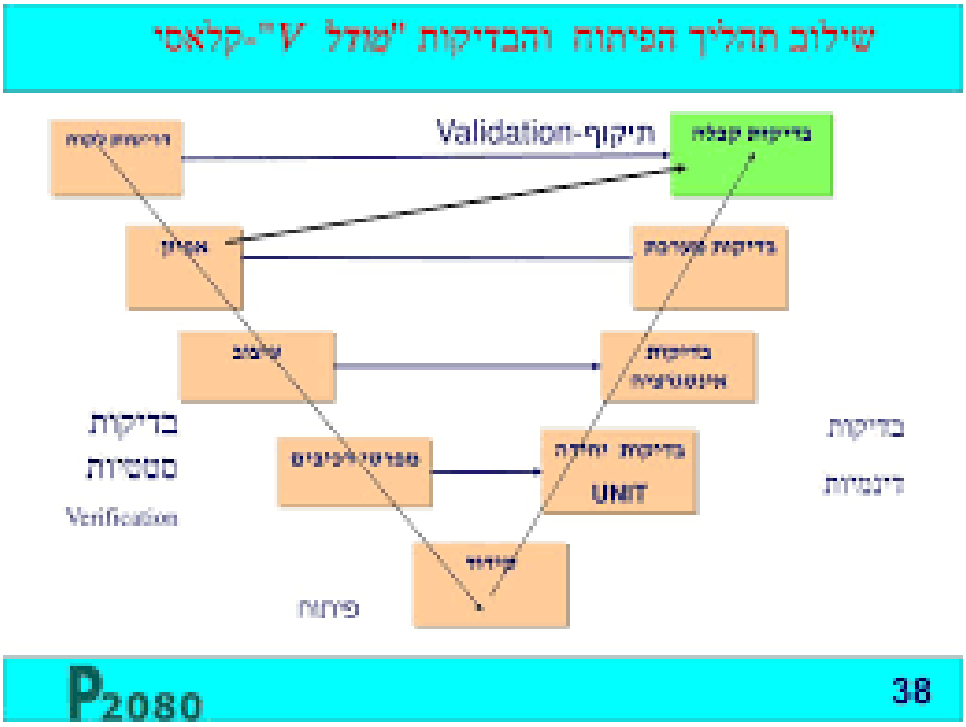
מסירה למשתמש

כל בעל תפקיד שמסיים את תפקידו מעביר את הטיפול הלאה. זה עובר לשלב הבא בתנאי שהשלב הקודם הסתיים.

זה טוב כשיש לנו מערכות קריטיות ואנחנו רוצים לשמור על מערכת מסודרת.

מודל V

בתוך המפל פיתחו גם מודל נוסף.




בזמן שכותבים את מסמך הדרישות מריצים כבר בדיקות על המסמך. בזמן שכותבים את מסמך האפיון מריצים בדיקות על המסמך. וכך עבור כל שלב. לא מחכים שכל השלבים ייגמרו ואז מריצים בדיקות רק על המוצר הסופי, מגלים את התקלות והבעיות עוד בשלבים המוקדמים. זה עוזר לחסוך כסף וגם לעלות על ביות בצורה יותר יעילה.

<u>פיתוח באיטרציות</u>

השיטה הכי נפוצה בפיתוח זה היא שיטת Agile. לא מפתחים את כל המערכת בבת אחת אלא כל פעם חלק מסויים מתוך המערכת ואותה מעבירים דרך מחזור חיי פיתוח קלאסי, דרך מודל המפל (כלומר מודל V).

זה מאפשר שחרור מהיר לשוק. כמו כן, תוך כדי ניתן לקבל פידבק מהלקוח על המערכת/תוכנה. כל פעם משחררים גרסא.

כאן הבדיקות מורכבות יותר.

**תוכניות מקוונות (On Line)**

תוכניות שיש למשתמש איזושהי תקשורת עם אותה תוכנית. למשל, כספומט (יש דו-שיח בין התוכנית למערכת). יש פעולה של המשתמש שגורמת לתגובה של המערכת. הפעולות האלה גם נרשמות במאגר. מדובר על תקשורת בין משתמש לבין התוכנה בזמן אמת.

**עיבודי אצווה (Batch Processin)**

אין שום תקשורת בין המשתמש לבין המערכת. למשל, בכספומט נרשמו כל הפעולות של המשיכה. בסופו של יום כל תנועות המשיכה נרשמות וחשבונות הבנק מתעדכנים ביתרות. אלה עיבודים שקורים מאחורי הקלעים. אין פה אפילו מסך.

**קלט/פלט - I/O (Input/Output)**

קלט: קליטה של נתונים למערכת תוכנה.

פלט: כל מה שיוצא אחר כך. נתונים היוצאים ממערכת התוכנה.

לחצתי על כפתור משיכה (קלט), מקבל מסך שבו אני צריך להקליד את הסכום (פלט).

כל התנועות והפעולות האלה הופכות להיות הקלט של עיבוד האצווה בסופו של יום. זה יכול להיות פלט של עדכון במערכות הארגון ויכול להיות חשבונות.

סכום תשלום (קלט) > חישוב מע״מ (עיבוד - כלומר חישוב או עדכון במאגרים) > סכום למע״מ (פלט)

**אינטגרציה (Integration)**

שילוב. למשל, שילוב של יחידות קוד. למשל, יחידת קוד שמבצעת משהו אחד ויחידת קוד שמבצעת משהו אחר ויש שילוב ביניהן.

למשל, יחידה לקליטת קוד מוצר וכמות ויחידה לחישוב המחיר.

לפעמים יש שילוב בין תתי מערכות. לדוגמא: במערכת עו״ש בבנק יש שילוב של תת מערכת למשיכות והפקדות ותת מערכת להעברות.

יכול להיות גם שילוב בין מערכות (בבנק: שילוב בין מערכת עו״ש לבין מערכת הלוואות לבין מערכת פיקדונות).

יכול להיות גם שילוב בין חומרה לבין מערכת/תוכנה (כמו מל״ט, CT).

**ממשקים (Interfaces)**

אוסף הדרכים שבהן ניתן לתקשר עם התוכנה. למשל, ממשק משתמש (UI - User Interface). זו הדרך של המשתמש להתממשק עם התוכנה.

יש גם התממשקות של רשת מחשבים (Network). יש התממשקות בין מחשב למחשב, יש תקשורת ביניהם. היכן שיש תקשורת בין גורמים אז יש ממשק, התממשקות.

תקשורת בין מערכות:

א. ממשק פנימי - התממשקות בין מערכות בתוך הארגון. בבנק למשל בין מערכת הלוואות למערכת העו״ש.

ב. ממשק חיצוני - למשל, בין מערכת רישום תושבים של משרד הפנים לבין מערכת הלקוחות של הבנק.

**הסבה (Conversion)**

תוכנה להסבת נתונים ממערכת קיימת לשימוש במערכת חדשה/מחליפה.
הסבה, למשל, של נתונים ממערכת מסויימת אחת למערכת אחרת.
מאגר נתונים מאופיין לפי:
סוג המאגר
מבנה המאגר
הנתונים עצמם

כשאנחנו רוצים לעשות הסבה אז יש לנו מערכת ישנה עם מאגר נתונים ישן ויש לנו מערכת עם מאגר נתונים חדש. תיהיה תוכנית הסבה של נתונים וצריכים לוודא שהנתונים במערכת החדשה זהים לנתונים שהיו במערכת הישנה.

**סביבת עבודה**

כוללת כל מיני רכיבי מחשב שמאפשרים לנו לבצע את העבודה

1. סביבת פיתוח - הסביבה של המפתחים. בסביבה זו יש כלי פיתוח, כלי Debugger (שבאמצעותו בודקים באגים) ועוד.
2. סביבת בדיקות - הסביבה של הבודקים. כלים לניהול וכתיבת הבדיקות.
3. סביבת הייצור - סביבה בה בודקים כבר את המערכת. זו סביבה לבדיקת המערכת כפי שהיא, המוצר המוגמר. זו כבר הסביבה של המשתמשים.

אם הבודקים היו עובדים בסביבת הפיתוח אז הפיתוחים שעשו היו עלולים לפגוע בבדיקות שלנו (כי זו סביבה בה אנחנו מפצחים קודים ומשנים קודים והיא לא מותאמת לסביבת בדיקות), ולכן חשוב להפריד בין הסביבות.

יש גם הפרדה בין בדיקות לבין סביבת הייצור - בסביבת הבדיקות יש העתק של המוצר ובסביבת הייצור יש סביבה של תנאי אמת, עם המוצר האמיתי. מה שלא טוב זה אם המשתמשים בסביבת הייצור מוצאים לפתע באגים ואז המערכת חוזרת לסביבת הפיתוח.

## הגדרות בבדיקות

**Validation & Verification**

2 נקודות מבט על המערכת שאנחנו רוצים לבדוק
א. Validation - וידוא שהמערכת אכן מספקת את הפונקציות והפעולות שהלקוח ביקש. שפיתחנו מה שהלקוח ביקש.

ב. Verification - לוודא שהמערכת עובדת בצורה תקינה.

**מסגרת העבודה (מהות הבדיקות)**

פעם היו מחפשים באגים וזהו. היום הבדיקות הרבה יותר מורכבות, גם המערכות בן יותר מורכבות, יש גם צורך בחשיבה מחוץ לקופסא. כל מערכת שונה זו מזו עם דרישות שונות. (מערכת אינטרנטית, מערכת משובצת, מערכת בנקאית). חשוב להכיר את המערכת איתה עובדים והיכן לשים את הפוקוס.
יד גם רכיבים בפרוייקט השונים זה מזה.
יש גורמים מעורבים שונים בין ארגונים. לנו יש תקשורת עם המפתחים, עם מנהלי המוצר אך אולי בחברה אחרת לא.
יש מגבלות ומחויבויות שהם שונים בין הפרוייקטים (דד-ליינים, למשל).

**פשרות**

לא נעבוד ונוציא מערכת שנקייה מתקלות ולכן נרצה לעשות את זה בצורה חכמה.
נרצה לעשות פשרות חכמות ולא שטוחות. למשל, נרצה לבדוק אתר קניות - נרצה לבדוק את כל המוצרים? כל הכמויות? לא.
פשרה שטוחה תיהיה למשל לבחור כל מוצר עשירי ולבדוק אותו. פשרה חכמה תיהיה למשל לבדוק מוצרים פופולריים.

**גישת הבודק**

מה יכול לדפוק את המערכת **(AIM TO KILL).**
למשל, ללחוץ על כפתור אנטר מאה פעם.
יצירתיות
שיטתיות - לעבוד בצורה שיטתית ומסודרת
תשומת לב לפרטים
חשיבה לעומק

לא להיות נעולים על הטכניקה, להתאים את שיטת העבודה לצרכים ולמגבלות של המערכת הנבדקת.

**שיטת הקופסא הלבנה (השקופה)**

אני יכול לראות את הקוד שיש.
אפשר לבצע בדיקות ישירות על הקוד. זה בדרך כלל עושה המפתח. המפתח יריץ בדיקות על הקוד שלו.
אבל זה מאוד טכני, הוא לא יתחיל להיות מאוד יצירתי. הוא יבדוק למשל 3 אופציות. המפתח הרבה פעמים רואה את מה שהוא מפתח, לא רואה את הקשר בין מה שהוא מפתח לבין שאר המערכת.

**שיטת הקופסא השחורה**

לא רואים את הקוד שיש מאחורה. אנחנו בתור בודקים נכניס כל מיני קלטים ונקבל כל מיני פלטים ונעבור על הפונקציות השונות.

למשל הפונקציה שאנחנו בודקים זה חישוב משכורת ע״י הכפלת שעות העבודה בתעריף העבודה לשעה.

אנחנו נתחיל כל מיני קלטים, נכניס את כל הקלטים האפשריים שעוברים על כל האופציות.

נכניס למשל:

| תעריף לשעה (קלט) | שעות עבודה (קלט) | משכורת (פלט) |
|---|---|---|
| 25 | 180 | 3900 |
| 25 | 30.5 | 762.5 |
| 0 | 180 | 0 |
| 25 | 0 | 0 |
| 9999 | 9999 | ERROR |
| 25 | -100 | ERROR |
| 12 | 100 | 1200 |

לדעת שבאמצעות עושר הבדיקות אני עובר את כל שורות הקוד מבלי באמת לדעת מה שורות הקוד.

**שיטת קופסא אפורה**

מצב שבו אנחנו לא יודעים את הקוד אבל כן יש לנו גישה למאגרי הנתונים מאחורה. למשל, אם אני בודק מערכת כספומט - לא רק שהכסף נמשך וכמה נמשך אלא גם שהמשיכה נרשמה במאגרי הנתונים.

**בדיקות סטטיות**

אנחנו עושים בדיקות על המסמכים. קריאה ביקורתית של המסמכים (אפיונים, דרישות הלקוח).

חשוב לבצע קודם כל בדיקות על המסמך עצמו (״בדר״כ״, ״לפעמים״, שוכחים מסכים שלמים, למשל) בצורה ביקורתית. זה עוזר לנו להבין לעומק את המערכת ולמצוא את ההשלכות והבעיות של המערכת.

לוודא את שלמות התוכן. תמיד לשאול ״מה אם...״.

זה עוזר למצוא בעיות באפיון בשלב מוקדם, מה שיתרום לכתיבת קוד יותר שלם ונכון ולפחות תקלות במערכת.

**בדיקות דינמיות**

בדיקות של המערכת עצמה או הגירסה. קלט/פלט. זה גורר לתיקון התקלות, עדכון מסמכי המערכת.

**סוגי נתונים**

בדיקות דינמיות דורשות נתונים (מאגרי מידע) בסביבת הבדיקות.

אתר קניות: מאגר הפריטים למכירה, מאגר הלקוחות. מערכת ניהול עוש: מאגר לקוחות, מאגר העברות וכו.

1. נתונים סינטטיים - נתונים הנרשמים במאגרים באופן מלאכותי, אשתמש בזה כשאין לי גישה לנתונים (למשל, מערכת חדשה). לפי צרכי הבדיקות.
2. נתוני "אמת" - העתקת נתונים מסביבת הייצור לסביבת הבדיקות. עדיף לעבוד עם נתוני אמת כי זה יוצר סביבה הרבה יותר קרובה לסביבת הייצור/אמת. זה מתאים יותר למערכות תחזוקה.

## רמות בדיקה

לא כל הארגונים מתייחסים לזה, אבל זה חלק מהמתודולוגיה הקלאסית שארגונים מסודרים יותר מתייחסים אליהם.

רמת בדיקה = אוסף בדיקות המתבצעות בהיבטים שונים.

הבודק צריך לראות את המערכת בצורה יותר רחבה, יש את הלקוח שבא מנקודת מבט של הדרישות, יש את המשתמש בקצה, יש את המפתח ויש את הבודק.

כל גורם משמש רמת בדיקה שונה.

רמות בדיקה המצריכות תכנון מקדים:

- בדיקות יחידה
- בדיקות מסירה
- בדיקות קבלה
- בדיקות אינטגרציה

חשוב לתכנן כדי להיות יותר יעילים, לחסוך בזמן. כמו כן, התיעוד במסמכי התכנון מאפשר לנו למחזר את הבדיקות עבור בדיקות אחרות - כדי להבטיח כיסוי ושימוש חוזר.

רמות בדיקה שאינן מצריכות תכנון מקדים:

- בדיקות "שפיות" ("עשן")
- בדיקות חוזרות ורגרסיה

- בדיקות MONKEY

הן פשוט מתבססות על חלק מהבדיקות שכבר תכננו.

העבודה העיקרית של הבודק היא הבנת המערכת ותכנון הבדיקות מראש.

1. **בדיקות יחידה (Unit Tests)**

המפתחים מבצעים את הבדיקות על יחידת קוד. שיטת הבדיקה היא קופסא לבנה. הבסיס לבדיקות (כלומר, על מה אני מבסס את התכנון של הבדיקות שלי) הוא הקוד עצמו או מסמכי עיצוב הקוד.

יתרונות: אלה הבדיקות היחידות שמבצעים המפתחים בשיטת הקופסא הלבנה. יכולים כבר לראות באיזו קוד הבעיה ויותר קל להם לתקן.

חסרונות: מאוד מצומצמות, לא לוקחות בחשבון את דרישות הלקוח והמערכת.

2. **בדיקות מערכת/מסירה (System/Delivery)**

בדיקות שמבצעים לפני שמוסרים את המוצר ללקוח, הבדיקות העיקריות שלנו.

בודקים את המערכת גם מבחינה פונקציונלית וגם טכנית.

- לבדוק כל פעילות במערכת
- לבדוק כל שדה במסך
- לבדוק כל מאגר נתונים
- לבדוק תקשורת

זה מתבצע על ידי הבודקים. שיטת הבדיקה היא קופסא שחורה או אפורה. אנחנו מבססים את הבדיקות שלנו

(כותבים אותם): על מסמכי האפיון או על מסמכי הדרישות.

יתרונות: המסה העיקרית של הבדיקות שמספקות את הבדיקות המקיפות ביותר. כמו כן, מאפשרת השוואה מול

דרישות הלקוח.

חסרונות: מסה עצומה של בדיקות שלא תמיד מאפשרת לבצע את כל הבדיקות ואנחנו נאלצים לבצע פשרות.

3. **בדיקות קבלה (Acceptance Tests)**

לוודא שהלקוח אכן מקבל את התוכנה שביקש בהיבט עבודת המשתמש.

לפעמים זה יהיה צוות בודקים מתוך צוות מסירה ולפעמים המשתמשים עצמם, לפעמים הלקוח יקים אצלו צוות לצורך בדיקות אלה.

זו רק קופסא שחורה.

הבסיס לבדיקות הוא מסמכי הדרישות, מה הלקוח ביקש.
יתרונות: משלים את בדיקות המסירה.
חסרונות: אלה לא בדיקות מעמיקות, לא נכנסים כאן למאגרי נתונים.

מה הקשר בין בדיקות המסירה לבין הקבלה?
בארגון שלנו יש גוף פיתוח ושם יש גם גוף בדיקות שעושה בדיקות מסירה/מערכת. אצל הלקוח יש גם גוף בדיקות והם עושים בדיקות קבלה - לפעמים גם הם עושים בדיקות מעמיקות, אבל לא כמו בודקים בבדיקות מסירה/מערכת. אין להם גישה למאגרי נתונים, לכלים למציאת באגים ובעיות.

דוגמא לבדיקות השונות (אתר קניות):
א. <u>בדיקות יחידה:</u>
חישוב סכום לתשלום (יחידה)
הפקה ושליחה של קבלה בסיום הרכישה (יחידה)

ב. <u>בדיקות מסירה:</u>
הכנסת פרטי אשראי שגויים או תקינים
ייצוא ורישום הפרטים במאגרים

ג. <u>בדיקות קבלה:</u>
בחירת פריט, מעבר ל"סל הקניות" וכ'ו.

**4. בדיקות אינטגרציה**

המפתחים יבצעו בדיקות אינטגרציה בין יחידות קוד שונות. כמו כן, יש שילוב בין יחידות, תת-מערכות, מערכות, שילוב בין מערכת תוכנה לבין חומרה.
נבדוק כל חלק בנפרד ונבדוק כיצד הם עובדים ביחד.
אם מדובר על אינטגרציה בין יחידות אז זה מבוצע על ידי המפתחים. שאר החלקים מבוצעים על ידי הבודקים או המשתמשים.

**5. בדיקות "שפיות" ("עשן")**

בודקים שהמערכת תקינה מספיק כדי לעבוד איתה.
נעשה את זה קודם בסביבת הפיתוח. לפני העברה של המערכת לסביבת הבדיקות יבוצעו בדיקות של מסירה בנבחרו באקראי (במדגם). אנחנו רוצים לוודא שהמערכת יציבה מספיק כדי להתחיל לבדוק אותה, זה למשל מאתר באגים יותר מדי בעייתיים.

העברת מערכת מסביבה לסביבה זו דרך לא פשוטה, יש למשל העתקת קבצים מאוד גדולה. לכן, כל הדברים האלה נועדו לצמצם בזבוז זמן.
לאחר מכן, הבדיקות מבוצעות על ידי הבודקים. נבצע את הבדיקות פעם נוספת אחרי שהמערכת עברה לסביבת הבדיקות, על מנת לוודא שהיא יציבה. זה בטרם שאנחנו מתחילים את בדיקות המסירה.
שיטת הבדיקות: קופסא אפורה. בסיס הבדיקות: מדגם מתוך מערכת הבדיקות (מסירה) שתוכננו.

### 6. בדיקות חוזרות (Repeated Tests)

אנחנו מצאנו בתור בודקים באגים, מוחזר למפתחים, עוברים תיקונים ומוחזר אלינו לסבב נוסף של בדיקות. נרצה לוודא שהבאגים תוקנו ולכן נחזור על אותן בדיקות של הבאגים.
הבדיקות החוזרות מבוצעות על ידי הבודקים בשיטת קופסא שחורה או אפורה. בסיס הבדיקות: בדיקות שבוצעו כבר, תקלות שתוקנו

### 7. בדיקות רגרסיה (Regression Tests)

לוודא שדברים שעבדו פעם לפני התיקון לא נשברו פתאום. שלא נמצא "תקלת רגרסיה". זה יכול לנבוע משינוי בקוד שבוצע בשל דרישה לתקלה שתוקנה או דרישה לפיתוח חדש.
אני אחזור על אותן בדיקות שעבדו לפני כן ואוודא שהדברים הללו עדיין עובדים. אז לאחר שהמוצר יחזור אלינו חזרה בעקבות תיקון, עושים בדיקות חוזרות וגם נדבוק שהדברים מסביב לאותה תקלה עדיין עובדים.
כמו כן, נבצע בדיקות אלה לפני השחרור של המוצר באופן סופי ללקוח ולמשתמשים.
מי שמבצעים את הבדיקות אלה הם הבודקים בשיטת קופסא שחורה/אפורה. הבסיס לבדיקות: מדגם מתוך הבדיקות שעברו בהצלחה לפני תיקון התקלות ומדגם של בדיקות המכסות את עיקרי המערכת.

### 8. בדיקות קהל (Crowd)

אנשים נרשמים לאתר בדיקות ומקבלים פרוייקטים. הם מקבלים כסף עבור כל באג שהם מוצאים.
בדיקות קהל מאפשר מגוון עצום של מכשירים, היום יש אינספור קומבינציות של מכשירים. מי שמבצע את הבדיקות הם הבודקים או כל אדם שנרשם כבודק בחברה המספקת את הבדיקות הללו.
אין שיטת בדיקות (למרות שזה מאוד תלוי בבודק). בסיס לבדיקות - לפעמים יש אפיון, לפעמים יש הסבר על אותה אפליקציה למשל. לפעמים אין.
יתרונות: יש מגוון אדיר של מכשירים, זה מאוד יעיל כי כל אחד מסתכל בצורה שונה ועושה בדיקות שונות.
חסרונות: אין פה עבודה מסודרת, לא תמיד ניתן לעקוב אחרי זה.

## 9. בדיקות Monkey

בדיקות בלי שיטה, ביצוע בדיקות לא שגרתיות, יצירתיות. זה אף פעם לא יבוא במקום הבדיקות השיטתיות שלנו. המבצעים: כל אדם עם גישה "הרסנית" למערכת. אין שיטת בדיקה. בסיס לבדיקות: גישת משתמש. מתאים במיוחד למערכות שלא ניתן להפנות לבדיקות Crowd Tests.