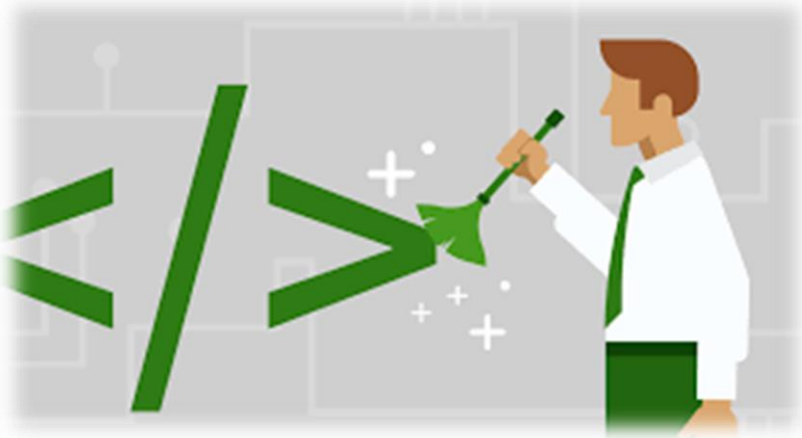
המכללה האקדמית להנדסה סמי שמעון

REFACTORING

תיקון/שיפור הקוד



ירדן קלדרון
315097527

ירדן שקד
206789885

ראם לוי
205866692

בס"ד

מהו תיקון/שיפור קוד - refactoring:

refactoring הוא תהליך שיטתי שמטרתו לבטל חובות טכניים ולטפל בסירחונות קוד וכתוצאה מכך נקבל קוד נקי יותר. על ידי שימוש בטכניקות שונות של refactoringאנו כמפתחים נשפר באופן איטרטיבי את איכות, הקריאה והתחזוקה של הקוד, ומבטיחים שהתוכנה תישאר עמידה לשינויים לאורך זמן.

חוב טכני - Technical Debt:

חוב טכני מתייחס לעלויות עבודה נוספות שנגרמות כאשר צוותי פיתוח תוכנה בוחרים בפתרון מהיר וקל ולא בגישה טובה יותר, אשר ייקח זמן רב יותר ליישום. זוהי מטאפורה שטבע וורד קנינגהם (Ward Cunningham) כדי לתאר כיצד קיצורי הדרך והפשרות הללו יוצרים "חוב" שיש לשלם מאוחר יותר באמצעות תהליכי פיתוח שנוספו.

קוד נקי – Clean Code:

קוד נקי הוא קוד שקל לקרוא, להבין ולתחזק. השקעה בקוד נקי משתלמת בטווח הארוך על ידי הפחתת באגים, הפחתת חוב טכני, שיפור הפרודוקטיביות ושיפור שיתוף הפעולה בין מפתחים. קוד נקי הוא לא רק פונקציונלי, אלא גם אסתטי ומאורגן היטב.

סירחונות קוד – Code Smells:

זוהי מטאפורה שטבעו קנט בק וורד קנינגהם (Kent Beck and Ward Cunningham) כדי לתאר קוד ש "doesn't quite feel right" ועשוי להצביע על אזורים לשיפור בקוד. בדיוק כפי שניחוח רע מרמז שמשהו לא בסדר מסביבנו, סירחונות קוד מצביעים על בעיות פוטנציאליות בתכנון התוכנה או ביישום התוכנה.

סירחונות קוד אינם באגים בעצמם, אבל הם לעתים קרובות יכולים להפוך לבאגים או להפוך את הקוד לקשה יותר להבנה, תחזוקה או להרחבה. הם משמשים כסימני אזהרה לכך שהקוד עשוי להיות בנוי בצורה גרועה, מורכבת מדי ושיהיה קשה לעבוד איתו.

מקורות לעמוד : 1,2,3,4,6

מתי נבצע refactor:

באופן אידיאלי, תהליך ה-refactoring צריך להיות תהליך מתמשך לאורך כול מחזור החיים של פיתוח הקוד, ולא אירוע חד פעמי.

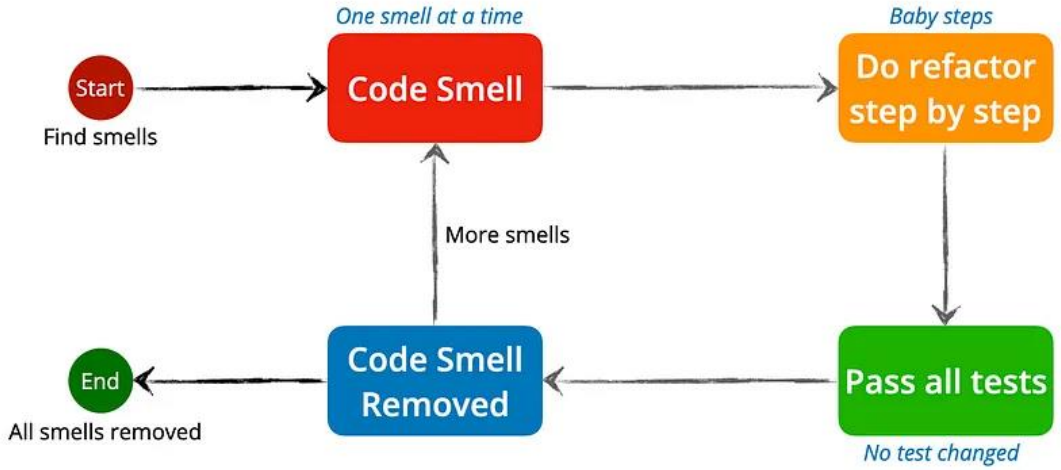
תרחישים שונים שכדאי לנו להשתמש ב- refactoring:

- **הוספת תכונות חדשות:** לפני הוספת תכונות חדשות, מומלץ לשנות את הקוד הקיים כדי להבטיח שהוא נקי, מודולרי ומוכן להתאים לשינויים.
- **תיקוני באגים:** בעת תיקון באגים, ננצל את ההזדמנות לשנות את הקוד הקשור לבאג כדי למנוע בעיות דומות בעתיד.
- **סירחונות קוד**: סירחונות קוד, כגון קטעי קוד כפולים, מתודות ופונקציות ארוכות, ועוד... הם אינדיקטורים לכך שייתכן שיהיה צורך בשיפוץ הקוד הקיים כדי לשפר את איכות הקוד ואת התחזוקה שלו.
- **מעקב אחר החוב הטכני:** refactoring מסייע להפחית את החוב הטכני, אשר מצטבר לאורך זמן כתוצאה מקיצורי דרך או פשרות שנעשו במהלך הפיתוח.
- **שינוי בצוות:** כאשר חברי צוות חדשים מצטרפים או משתנים תפקידים, refactoring יכול לסייע להם להכיר את בסיס הקוד.

מקורות לעמוד: 1,5,6

כיצד נבצע refactoring:

1. **זיהוי אזורים לשיפור:** שימוש בכלי ניתוח קוד או במעבר ידני על הקוד כדי לזהות אזורים בקוד שנצטרך לבצע בהם refactoring.
2. **כיסוי טסטים מלא:** לפני ביצוע תיקונים, נוודא שיש לנו כיסוי בדיקות מלא עבור הקוד שנשנה.
3. **תיקון קטן בכול פעם:** ביצוע שינוי אחד קטן בכל פעם . כל צעד צריך להיות פשוט ולהתמקד בשיפור חלק ספציפי של הקוד, כגון חילוץ מטודה, שינוי שם של משתנה, או הסרת כפילויות קוד.
4. **ביצוע טסטים:** לאחר כל שלב של תיקון בקוד, נריץ את הבדיקות כדי לוודא שהקוד עדיין פועל כמו שצריך . אם ניתקל בכשלים בבדיקות נבטל את השינויים ונבדוק את הבעיה.
5. **סקירה ואיטרציה:** נבדוק שהשינויים שנעשו תואמים לעקרונות העיצוב שאנחנו מצפים להם לאחר השלמת התיקון. נבצע לפי הצורך איטרציה של כול השלבים חזרה .
6. **תיעוד השינויים:** אם סיימנו את תהליך הrefactoring נתעד את השינויים שעשינו . טוב למעקב לשינויים שבוצעו.



מקורות לעמוד: 2,3,4

טכניקות Refactoring:

שתי דוגמאות לטכניקות תיקון קוד:

- **Extract Method:**

בעיה: יש קטע קוד שניתן לקבץ יחד.

פתרון: העברת קטע קוד זה לפונקציה חדשה נפרדת והחלפת הקוד הישן בקריאה לפונקציה.

לפני התיקון:

```
def printOwing(self):
    self.printBanner()

    # print details
    print("name:", self.name)
    print("amount:", self.getOutstanding())
```

לאחר התיקון:

```
def printOwing(self):
    self.printBanner()
    self.printDetails(self.getOutstanding())

def printDetails(self, outstanding):
    print("name:", self.name)
    print("amount:", outstanding)
```

השלבים:

1. יצירת פונקציה חדשה עם שם שיבהיר את פעולתה כראוי.
2. העברת קטע הקוד לפונקציה החדשה והחלפת הקוד בקריאה לפונקציה.
3. כול משתנה בתוך קטע הקוד שבפונקציה החדשה שהוגדר מחוץ לאותו קטע קוד יועבר לפונקציה כפרמטר.
4. אם יש משתנה מקומי שיש שינוי בערכו בתוך הפונקציה ויש בו צורך לאחר הפונקציה , אותו משתנה יוחזר בעזרת פעולת return .

מקורות לעמוד: 1,6

- **Inline Temp:**

בעיה: יש משתנה זמני שמכיל תוצאה של ביטוי פשוט.

פתרון: החלפת ההפניות למשתנה הזמני בביטוי עצמו.

לפני תיקון:

```
def hasDiscount(order):
    basePrice = order.basePrice()
    return basePrice > 1000
```

לאחר תיקון:

```
def hasDiscount(order):
    return order.basePrice() > 1000
```

השלבים:

1. מחק את ההגדרה של המשתנה הזמני ואת ההקצאה שלו לביטוי.
2. בכול מקום בו יש שימוש במשתנה נחליף את המשתנה בביטוי.

ביבליוגרפיה:

1. https://refactoring.guru/
2. https://www.sonarsource.com/learn/refactoring/#what-is-refactoring
3. https://medium.com/@heaton.cai/when-should-we-refactor-in-tdd-1da800034972
4. https://dzone.com/articles/what-is-refactoring
5. https://stackoverflow.com/questions/140677/how-often-should-you-refactor?rq=3
6. https://www.geeksforgeeks.org/refactoring-introduction-and-its-techniques/

* רוב העבודה התבססה על מקור 1 , ואפילו חלק משאר המקורות השתמשו בו כמקור.