# ריפקטור(Refactor)

**מגישים:**

**קובי אלן - 318550985**

**מתן קחלון - 316550458**

**ליאור אנגל - 315006783**

# הגדרת תהליך הריפקטור

תהליך הריפקטור תהליך שמטרתו להפוך את הקוד לאיכותי יותר. זהו תהליך חיוני שמשפר את המבנה הפנימי של הקוד מבלי לשנות את אופן פעולתו. סביבת עבודה מבולגנת יכולה לגרום לקושי רב לעבוד ולהתמצא בה. קוד שלא עבר תהליך של ריפקטור יכול להיות מבולגן באותה מידה, מה שמקשה על ההבנה, השינוי והתחזוקה של הקוד. תהליך הריפקטור מתמודד עם חוסר ארגון זה על ידי ארגון מחדש, פישוט ואופטימיזציה של הקוד. זה יכול לכלול פירוק לוגיקה מורכבת לפונקציות קטנות יותר וניתנות לניהול, שימוש בשמות משתנים ברורים יותר ושיפור המבנה הכללי. היתרונות רבים: קריאה טובה יותר הופכת את הקוד לקל יותר לכולם (כולל את כותב הקוד) להבנה. שיפור התחזוקה תורם לכך שקל יותר להתאים את הקוד כאשר יש צורך בתכונות חדשות, פיצ'רים או תיקוני באגים, על ידי הפחתת המורכבות וחתיכות קוד מיותרות, ריפקטורינג עוזר גם לייעל את הקוד, שיכול להוביל לשיפור בביצועים. למעשה, ריפקטורינג הוא השקעה בבריאות ארוכת הטווח של בסיס הקוד. [2][4][5]

## מתי כדאי לעשות ריפקטור לתוכנה?

- כאשר הקוד קשה להבנה (ארגון לקוי, שמות גרועים) ריפקטורינג יכול לעזור בכך שיהפוך אותו לברור יותר וניתן לתחזוקה. [1]
- סיבה לבצע ריפקטורינג היא המהירות של ריצת הקוד, תהליך כזה יכול לגרום לכך שקוד יהיה יעיל יותר וירוץ בצורה חלקה וטובה יותר על ידי הורדת חתיכות לא רלוונטיות בקוד שמאטות אותו. [1]
- כאשר הקוד מלא בשורות כפולות או מחלקות גדולות מאוד מה שנקרא בתרגום פשוט "קוד מסריח"(code smells), תהליך הריפקטור יכול לנקות את הקוד על ידי יצירת פונקציות ומחיקת שורות מיותרות. [1][3]
- בקוד נורא חשוב עקרון ה"שימוש חוזר" (Reusability) לכן תהליך ריפקטור יכול לעזור לפירוק הקוד בצורה מודולרית לחתיכות קטנות שקל לעבוד איתן וכך תורם לעקרון החושב הזה בתיכנות. [1]
- כאשר ידוע על פיצ'ר חדש או שינוי בקוד שצריך לעשות, בשני המקרים האלו, ריפקטור יכול לעזור בכך שהוספת הפיצ'ר או השינוים יבוצעו בצורה קלה וחלקה יותר.
- הרבה פעמים בקוד יש לוגיקות מורכבות שלא לצורך מסיבות כאלו ואחרות. תהליך הריפקטור יכול לייעל ולפשט את הקוד. [1]

### חשיבות תהליך ריפקטור בקוד

לרוב תוך כדי כתיבת הקוד נראה שריפקטורינג הוא הליך מיותר, אך מנקה קוד "מלוכלך" אשר נהיה קשה לתחזוק, הבנה ועדכונים בהמשך פיתוח המערכת. הקוד המלוכלך מעט את הפיתוח העתידי של המוצר בגלל שהוא קשה להבנה ומורכב לשינויים. בטווח הארוך תהליך זה מונע את אפקט כדור השלג ובכך משאיר את הקוד מסודר, נקי וקל לעבודה לכותב הקוד עצמו למפתחים אחרים שיעבדו על הקוד, ובהכרח מוביל למוצר באיכות גבוהה יותר, ואף יכול לגרום לחיסכון בעלויות. ריפקטור עוזר בכך שתורם לפרוקיט שמתנהל בשיטה אג'ילית (Agile) בין היתר. [2][3][4]

## שיטות לריפקטור

- **הפשטה**

ריפקטור על ידי הפשטה היא טכניקה שבה אתה מזהה פונקציונליות משותפת למספר מחלקות ומתודות, מחלץ אותן לממשק נפרד או מחלקות מופשטות נפרדות. תהליך זה מסייע בהפחתת כפילויות בקוד, מקדם שימוש חוזר ומקל על הניהול והתחזוקה של הפונקציונליות המשותפת.
לדוגמה, שתי מחלקות עם מתודות דומות שמבצעות את אותם חישובים. על ידי חילוץ הלוגיקה המשותפת למחלקה מופשטת או לממשק, אתה יכול ליצור מימוש יחיד ששתי המחלקות יכולות לרשת או ליישם. הפשטה זו מפחיתה את כמות הקוד המשוכפל ומקלה על עדכון או שינוי הפונקציונליות המשותפת בעתיד. [1][4]

שלבים לביצוע ריפקטור על ידי הפשטה: [4]

- נעבור על הקוד ונחפש קטעים מיותרים אשר חוזרים או דומים אחד לשני. למשל פונקציות שונות שמבצעות את אותו התפקיד או לוגיקה אשר חוזרת על עצמה.
- לאחר זיהוי הקטעים המיותרים/כפולים צריך להבין איך ניתן לפשט אותן ללוגיקה אחת ופשוטה במידת האפשר
- יצירה של מתודות ומחלקות (עם שמות משמעותיים כמובן) אבסטרקטיות שיכילו את הלוגיקה הכפולה.
- החלפת הקוד הכפול או המיותר במתודות והמחלקות החדשות שיצרנו
- ביצוע בדיקות למחלקות\מתודות החדשות כדי לוודא שהן תקינות ומבצעות את העבודה הנדרשת אשר לא יפגע בפונקציונליות של המערכת.
- במידה וכל הטסטים עברו בהצלחה צריך לעדכן במידת הצורך את הדוקומנטציה של המערכת.

- **פישוט מתודות**

פישוט מתודות היא טכניקה אשר גורמת לקוד להיות ברור יותר, ניתן לתחזוקה ועבודה קלה יותר בכך שהוא מפחית את המורכבות של הקוד.
טכניקה זו כוללת בתוכה כמה גישות לפישוט הקוד, לדוגמא: הורדת מספר פרמטרים בפונקציות הפחתת תנאים לוגיקות עם מבנה פשוט יותר או פירוק מתודות לחתיכות קטנות יותר ובעלות תפקיד יותר מינימלי אשר תורם לעקרון הפרדת האחריות (SOC - Separation of Concern) במערכת. [1][5]

שלבים לביצוע הרכבה: [5]

- זיהוי מתודות ארוכות ומורכבות.
- תכנון פישוט מתודות אלה.
- פירוק המתודות שמצאנו ויצירת מתודות קטנות יותר מהן בעלי תפקיד מינימלי ויחיד.
- הטמעת המתודות החדשות והצבת הקריאות אליהן במקומות הרלוונטיים.
- פישוט תנאים מורכבים בסמוך לקריאות המתודות החדשות.
- הרצת בדיקות לקוד הכללי תוך כדי ווידוא שהפונקציונליות של המערכת לא נפגעה.

## כלי לביצוע ריפקטור

כלי מאוד נפוץ לביצוע ריפקטור הוא מערכות סביבות העבודה (IDEs - Integrated Development Environments) אשר מציעות למשתמש כלים מובנים לביצוע ריפקטור משמעותית מקלים על ייעול התהליך. כלים אלה יכולים להפוך משימות שחוזרות על עצמן לאוטומטיות כמו שינוי שמות של משתנים או עיבוד מחדש של בלוקי קוד ופונקציות לאורך הפרויקט. ישנם כלי IDE שונים כמו למשל: VSCode, Eclipse, PyCharm, IntelliJ IDEA אשר מספקים פונקציונאליות כמו למשל יישור הקוד וסידור מרווחים, שינוי ארגומנטים פונקציות בכל מקום בקוד בפעם אחת, סידור מודולריות ועוד. [2]

## ביבליוגרפיה:


[1] Code Refactoring: 6 Techniques and 5 Critical Best Practices (codesee.io)

[2] What is Refactoring? Code Restructuring Definition & Guide | Sonar ((sonarsource.com

[3] Lacerda, G., Petrillo, F., Pimenta, M. and Guéhéneuc, Y.G., 2020. Code smells and refactoring: A tertiary systematic review of challenges and observations. Journal of Systems and Software, 167, p.110610.

[4] Fowler, Martin. "Refactoring." TOOLS (34). 2000.

[5] https://refactoring.guru/refactoring