אוניברסיטת
אריאל
בשומרון

עיצוב (תכן) מבנה תוכנה

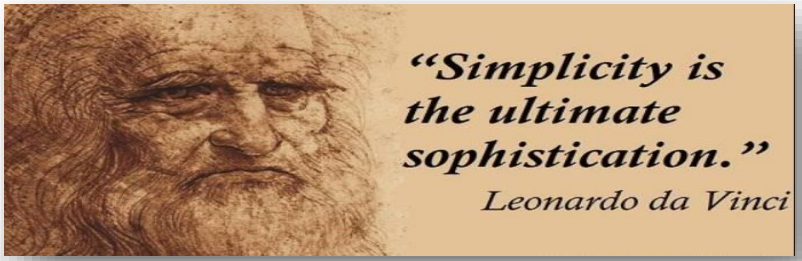
ארכיטקטורת מערכת vs. ארכיטקטורת תוכנה

- ארכיטקטורת מערכת: מודל קונספטואלי המתאר מבנה והתנהגות אוסף מרכיבים ותת מערכות הכוללות אפליקציות, מרכיבי רשת, חומרה, חיישנים ואלמנטים נוספים. ההסתכלות היא על כלל מרכיבי המערכת.

- ארכיטקטורת תוכנה: תהליך הגדרת מבנה High Level לתוכנה הכולל הגדרת מטרות העיצוב, מודולים מרכזיים ומנגנוני מימוש תכונות רצויות, בדגש על המרכיבים "הנראים" והאינטראקציה ביניהם. תכונות אלו כוללות:
  - Scalability
  - Security
  - modularity
  - Reusability
  - Extensibility
  - Maintainability

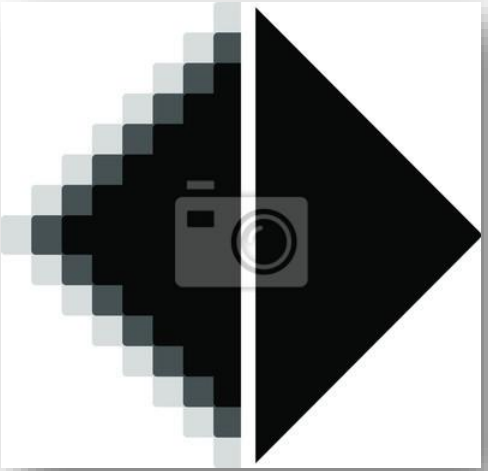
הסיבות לחשיבות שלב עיצוב התוכנה

- התמודדות עם מורכבות
  - אבסטרקציה: התעלמות מפרטים והתמקדות "בתמונה הגדולה"
  - דה-קומפוזיציה: פרוק הבעיה לאוסף תת בעיות בלתי תלויות (האם אי תלות באמת קיימת ?)
  - מודולציה: הגדרת מבנים יציבים לאורך זמן
  - פרויקציה: התמקדות בזוית ראיה אחת כל פעם (View) בהתייחס לתת הבעיה הנבחנת.
- התמודדות עם מחזור חיי תוכנה ותנאי סביבת פיתוח
  - שינויים תכופים
  - אורך שלב התחזוקה
  - מבנה "נכון" מאפשר אדפטציה מהירה לשינויים



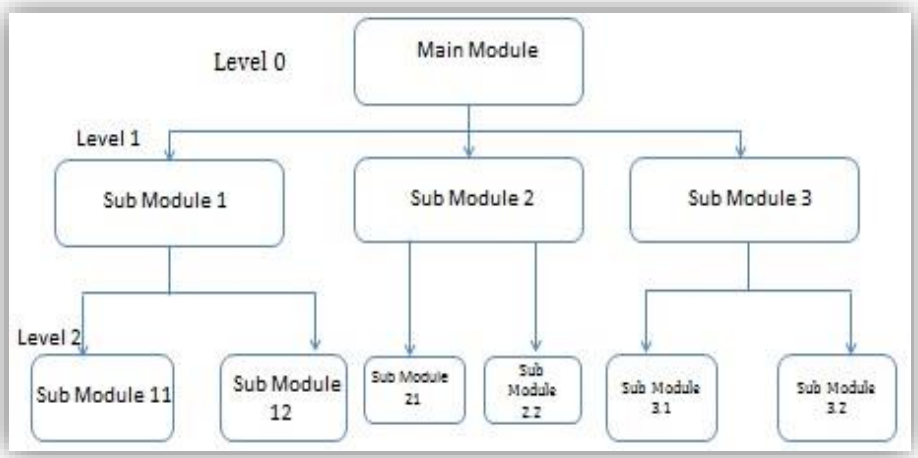
רמות עיצוב

- High Level Design: הגדרת מודולים מרכזיים ברמת גרנולציה (רמת פרטים) נמוכה. מכונה גם Architectural Design.
- Low Level Design:הגדרת מודולים ותוכנם ברמת גרנולציה (רמת פרטים) גבוהה. מכונה גם Detailed Design.

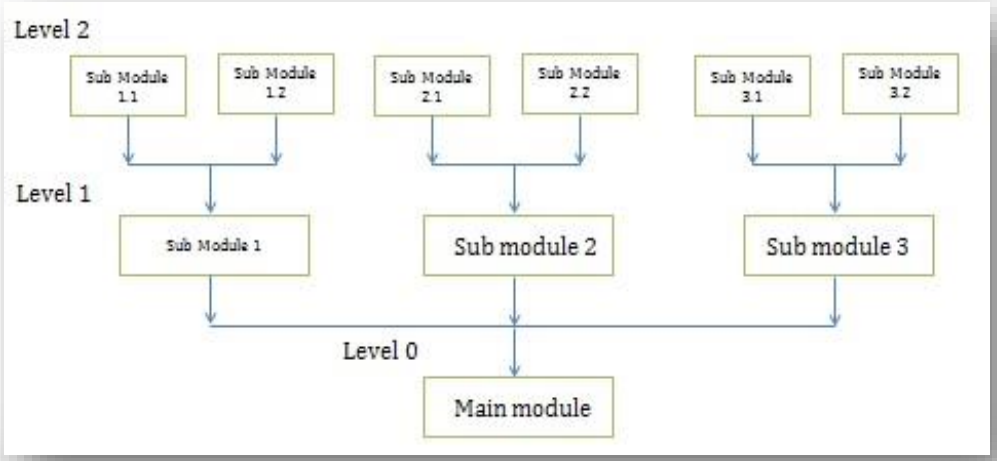


אסטרטגיות עיצוב

Top Down



Bottom Up



אוניברסיטת
אריאל
בשומרון

מה קורה למערכת במהלך מחזור חייה והקשר לאופן בו עוצבה

או

מה מאפיין מערכת בדעיכה ?

Robert C. Martin (Uncle Bob)

https://fi.ort.edu.uy/innovaportal/file/2032/1/design_principles.pdf

Software architecture

The overall shape and structure of software applications

→

The architecture that is specifically related to the purpose of the software application

→

The modules and their interconnections

Architecture and Dependencies

אוניברסיטת
אריאל
בשומרון

What goes wrong with software?

- Begins clean and elegant
- But then, the software begins to rot.

Symptoms of Rotting Design

- Rigidity
- Fragility
- Immobility
- Viscosity

אז איפה מתחילה ההידרדרות?

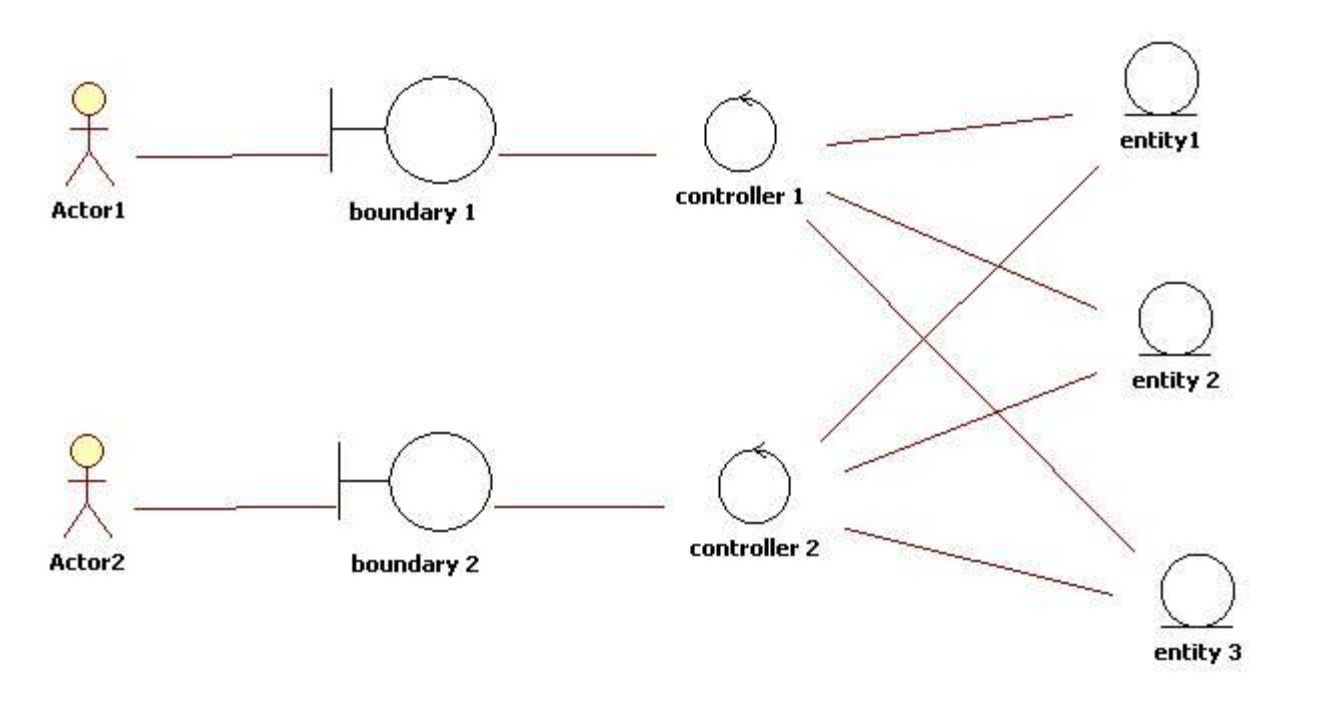
- Changing requirements
  - Changes that introduce new and unplanned for dependencies.
  - **Dependency firewall**

עקרון SoC ומודולציה

- SoC: Separation of concerns
- עקרון מנחה בעיצוב תוכנה הדוגל בהפרדת טיפול בנושאים וריכוזם ביחידות נפרדות.
- Module: יחידה נפרדת בתוכנה (פיזית או לוגית) המופרדת מאחרות בצורה "נכונה"
- מודולציה נכונה כוללת:
  - High Cohesion : מודולים ממוקדים בטיפול בנושא ספציפי
  - Loose Coupling: מודולים תלוים זה בזה באופן מינימלי

עיצוב תוכנה <=> הפרדת מרכיבי הפתרון למודולים וניהול התלויות ביניהם

Entity-Control-Boundary

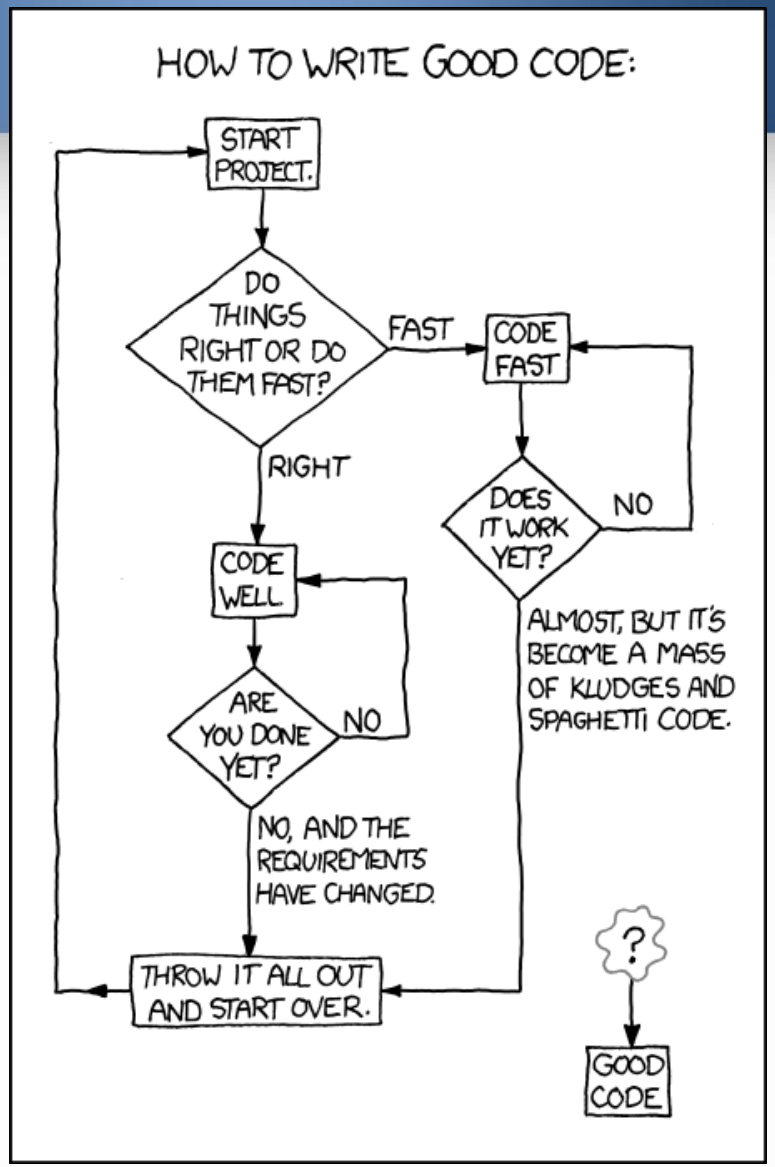


אוניברסיטת
אריאל
בשומרון

כתיבת קוד "נכון"

עקרון ה-kiss

- מהנדסי תוכנה נוטים לסבך את הקוד.
- Keep It Simple, Stupid
- פשטות פשטות פשטות
- הביטוי הוטבע ע"י קלי ג'ונסון (מהנדס מטוסים)



https://xkcd.com/844/

שמות משתנים ומחלקות

- שמות משמעותיים. לדוגמא- wagesPerHour לעומת, wph.
- לשמור על עקביות בשיטת מתן השמות.
- למשתנים זמניים (כמו מונה הלולאה) כדאי לקרוא בשם קצר. (i, j...)

שמות משתנים ומחלקות

IDENTIFIER	NAMING RULES	EXAMPLE

Variables	A short, but meaningful, name that communicates to the casual observer what the variable represents, rather than how it is used. Begin with a lowercase letter and use camel case (mixed case, starting with lower case).	`mass`
		`hourlyWage`
		`isPrime`

Constant	Use all capital letters and separate internal words with the underscore character.	`BOLTZMANN` `MAX_HEIGHT`

Class	A noun that communicates what the class represents. Begin with an uppercase letter and use camel case for internal words.	`class Complex`
		`class Charge`
		`class PhoneNumber`

Method	A verb that communicates what the method does. Begin with a lowercase letter and use camelCase for internal words.	`move()`
		`draw()`
		`enqueue()`

כתיבת הערות

- חשוב במידה
- הקוד אמור להסביר את עצמו, הערות נכתוב בשביל להסביר את ה-'למה'
- לא להשאיר קוד ישן בהערות
- בראש המסמך יש להוסיף הערה מי כתב את הקוד, מתי ומה הוא אמור לעשות

doc comments Java

- כלי שמגיע יחד עם ה- JDK
- מאפשר לנו ליצור תיעוד של הקוד שלנו בפורמט של HTML
- כותבים את ההערות בתוך הטווח שמתחיל עם **/ ומסתיים עם */
- משתמשים בתגים של HTML (כמו <H1>, <p>)
- ישנם מילים שמורות שאפשר להשתמש בהם בשביל לתאר דברים מסוימים-
  - @since
  - @version
  - ....
- שימושי למתכנתים אחרים שקוראים את הקוד, או לבעלי עניין אחרים.

אוניברסיטת
אריאל
בשומרון

The params of the function

The return value

Errors

```
/**
 * This method is used to add two integers. This is
 * a the simplest form of a class method, just to
 * show the usage of various javadoc Tags.
 * @param numA This is the first paramter to addNum method
 * @param numB  This is the second parameter to addNum method
 * @return int This returns sum of numA and numB.
 */
public int addNum(int numA, int numB)
{
    return numA + numB;
}

/**
 * This is the main method which makes use of addNum method.
 * @param args Unused.
 * @exception IOException On input error.
 * @see IOException
 */
public static void main(String args[]) throws IOException
{
    Ex2 obj = new Ex2();
    int sum = obj.addNum(10, 20);

    System.out.println("Sum of 10 and 20 is :" + sum);
}
```

doc comments Java

בתפריט הראשי בסביבת הפיתוח-

Tools→Generate Java Doc

ואז יווצר לנו מסמך HTML עם כל התיעוד

אוניברסיטת
אריאל
בשומרון

Method Detail

addNum

```
public int addNum(int numA,
                  int numB)
```

This method is used to add two integers. This is a the simplest form of a class method, just to show the usage of various javadoc Tags.

Parameters:

`numA - This is the first paramter to addNum method`

`numB - This is the second parameter to addNum method`

Returns:

`int This returns sum of numA and numB.`

main

```
public static void main(java.lang.String[] args)
                 throws java.io.IOException
```

This is the main method which makes use of addNum method.

Parameters:

`args - Unused.`

Throws:

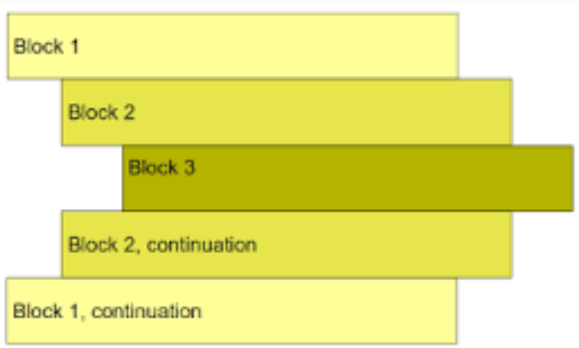
`java.io.IOException - On input error.`

See Also:

`IOException`

הזחות

- גם אם הזחה היא לא חובה בהגדרה של השפה שבה אנחנו כותבים, כדאי להקפיד עליה



K&R	`while (x == y) {`
	`  something();`
	`  somethingelse();`
	`}`

Allman	`while (x == y)`
	`{`
	`  something();`
	`  somethingelse();`
	`}`

GNU	`while (x == y)`
	`  {`
	`  something();`
	`  somethingelse();`
	`  }`

- יש שיטות הזחה רבות, יש לשמור על עקביות

https://gist.github.com/jesseschalken/0f47a2b5a738ced9c845

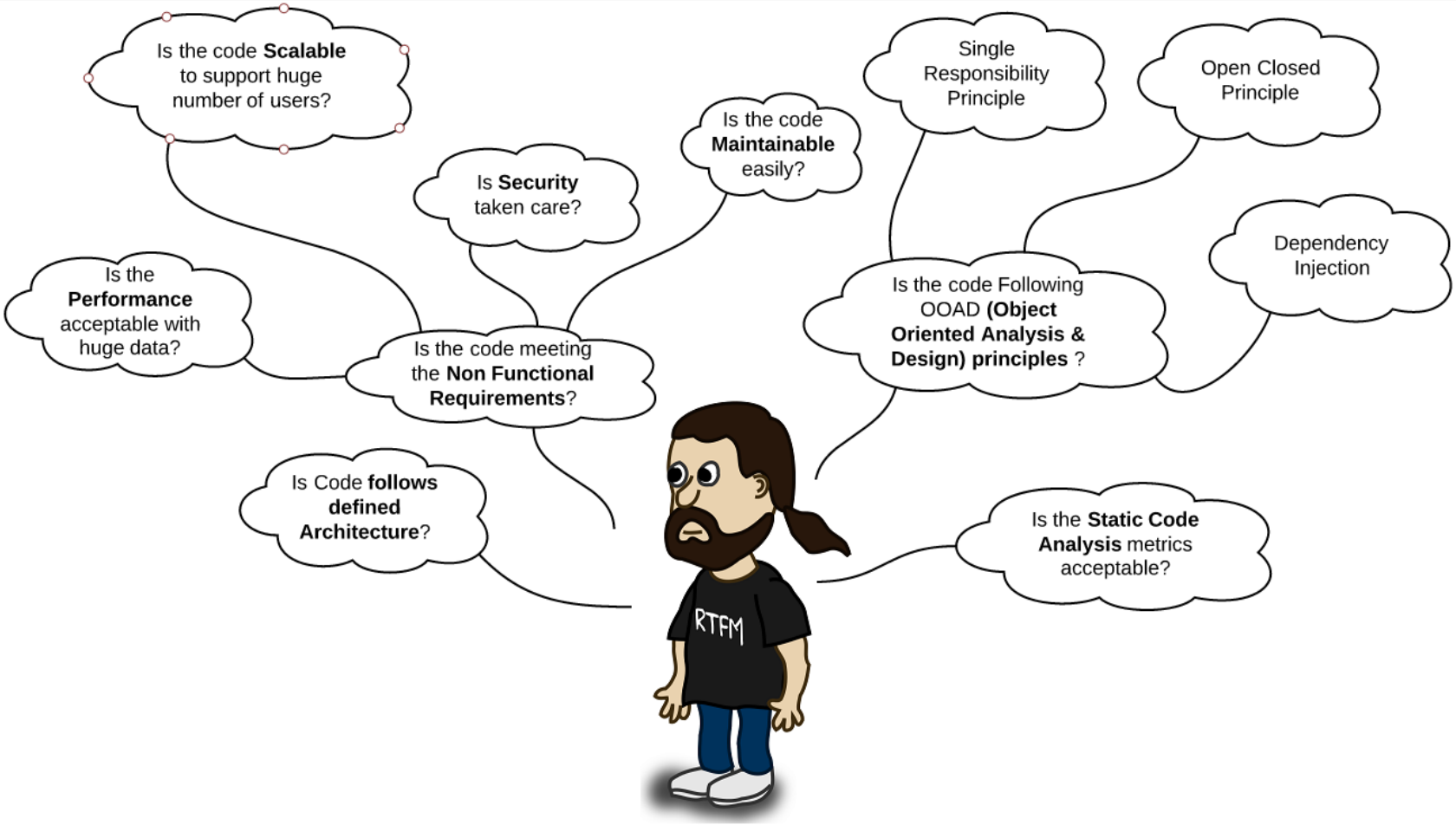
Code review

זוהי בדיקה שיטתית על קטע הקוד אשר נכתב לתוכנית או למערכת בכדי למצוא ולתקן בעיות שנמצאות בקוד עוד בשלבי הפיתוח המוקדמים.

הבדיקה נעשית ע"י עמיתים ולא ע"י המתכנת עצמו.

מטרות-

- מציאת בעיות ותיקונם.
- שיפור תהליכים בצוות הפיתוח.
- שיפור יעילות ומורכבות הקוד.
- עוזר לשיתוף מידע בתוך הצוות



Expert Code Reviewer

מה code review מאתר

- קוד מת.
- בעיות יעילות ומורכבות.
- שימוש חוזר בקוד.
- דליפת וזליגת מידע.
- דליפת זיכרון

יתרונות code review

- מעבר על הקוד ומציאת בעיות בשלבים מוקדמים במחזור חיי המוצר/ הפרויקט.
- מציאת בעיות שלא ימצאו על ידי בדיקות רגילות לדוגמא אי שימוש במשתנים.
- צמצום הבעיות כתוצאה מראייה נוספת אשר משפיעה על איכות המערכת.
- אזהרה מוקדמת לגבי היבטים וחלקים חשודים בקוד שניתן לאמוד אותם רק על ידי מדידה או מורכבות.
- שיפור רמת התחזוקה של הקוד לשינויים עתידיים.

Code review check list

- Code formatting
- Architecture
- Coding best practices
- Non Functional requirements
- Object-Oriented Principles

אוניברסיטת
אריאל
בשומרון

SOLID

Solid – עקרונות לתוכן יציב

- Single Responsibility Principle (SRP)
- Open Close Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

single Responsibility Principle

כל מחלקה אחראית על פונקציונליות אחת, ואחראית עליה בצורה מלאה

לכל מחלקה יש סיבה אחת להשתנות

- Martin, Robert C. "The single responsibility principle." *The principles, patterns, and practices of Agile Software Development* 149 (2002): 154

דוגמא

```
public class RectangleShape
{
    private int height;
    private int width;

    public int getHeight() {
        return height;
    }

    public int getWidth() {
        return width;
    }

    public void setHeight(int newHeight)
    {
        this.height = newHeight;
    }

    public void setWidth(int newWidth)
    {
        this.width = newWidth;
    }

    int Area()
    {
        return getWidth() * getHeight();
    }

    void Draw()
    {
        System.out.println("Drawing a Rectangle");
    }
}
```

פתרון

```
public class DrawRectangleShape
{
    private int height;
    private int width;

    public int getHeight() {
        return height;
    }
    public int getWidth() {
        return width;
    }
    public void setHeight(int newHeight)
    {
        this.height = newHeight;
    }
    public void setWidth(int newWidth)
    {
        this.width = newWidth;
    }

    void Draw()
    {
        System.out.println("Drawing a Rectangle");
    }
}
```

```
public class AreaRectangleShape
{
    private int height;
    private int width;

    public int getHeight() {
        return height;
    }

    public int getWidth() {
        return width;
    }

    public void setHeight(int newHeight)
    {
        this.height = newHeight;
    }

    public void setWidth(int newWidth)
    {
        this.width = newWidth;
    }

    int Area()
    {
        return getWidth() * getHeight();
    }
}
```

Open Close Principle

מחלקה צריכה להיות פתוחה להתרחבות אבל סגורה לשינויים

במקרה של שינויים, נוסיף תת- מחלקות המממשים אותם

- Martin, Robert C. "The open-closed principle." *More C++ gems* 19.96 (1996): 9.

דוגמא

```
public class Shape {
    public double calculateArea(Shape[] shapes) {
        double area = 0;
        for(Shape shape:shapes) {
            if(shape instanceof Rectangle) {
                //Calculate Area of Rectange
            }
            else if(shape instanceof Circle) {
                //Calculate Area of Circle
            }
        }
        return area;
    }
}

class Rectangle extends Shape {}

class Circle extends Shape {}
```

פתרון

```
abstract class NewShape {
    public double calculateArea(NewShape[] shapes)
    {
        double area = 0;
        for(NewShape shape:shapes) {
            area += shape.area();
        }
        return area;
    }

    abstract double area();
}
```

```
class Rectangle extends NewShape {
    @Override
    double area() {
        // Area implementation for Rectangle
        return 0;
    }
}

class Circle extends NewShape {
    @Override
    double area() {
        // Area implementation for Rectangle
        return 0;
    }
}
```

Liskov Substitution Principle

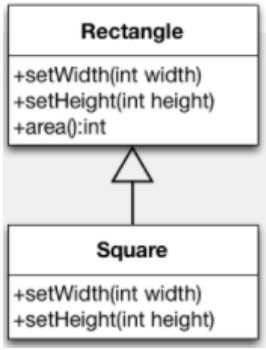
אם S היא תת-מחלקה (יורשת) של T אזי ניתן להחליף כל מופע של T במופע של S מבלי שההתנהגות תשתנה

מחלקה יורשת איננה יכולה לשנות את התנהגות מחלקת-האם

מי שפונה למתודה המוגדרת במחלקת האם איננו אמור לדעת לאיזו מהמחלקות-הבנות הוא פונה

דוגמא

```
public class Rectangle {
    private int height;
    private int width;

    public int getHeight() {
        return height;
    }
    public int getWidth() {
        return width;
    }
    public void setHeight(int newHeight)
    {
        this.height = newHeight;
    }

    public void setWidth(int newWidth)
    {
        this.width = newWidth;
    }

    public int getArea()
    {
        return this.height*this.width;
    }
}

public class Square extends Rectangle {
    public void setHeight(int newHeight)
    {
        super.setHeight(newHeight);
        super.setWidth(newHeight);
    }

    public void setWidth(int newWidth)
    {
        super.setHeight(newWidth);
        super.setWidth(newWidth);
    }
}
```



הבעיה

```
Rectangle r1 = new Rectangle();
r1.setHeight(4);
r1.setWidth(5);

Rectangle r2 = new Rectangle();
r2.setWidth(5);
r2.setHeight(4);

boolean match = r1.getHeight() == r2.getHeight() &&
r1.getWidth() == r2.getWidth();
System.out.println(match); // true
```

הבעיה

```
Rectangle s1 = new Square();
s1.setHeight(4);
s1.setWidth(5);

Rectangle s2 = new Square();
s2.setWidth(5);
s2.setHeight(4);

match = s1.getHeight() == s2.getHeight() && s1.getWidth() ==
s2.getWidth();
System.out.println(match); // false
```

הפתרון

```
interface Shape
{
    int getArea();
}

class Rectangle implements Shape
{
    protected int width;
    protected int height;

    protected Rectangle(int width, int
height)
    {
        this.width = width;
        this.height = height;
    }

    public int getWidth()
    {
        return width;
    }

    public int getHeight()
    {
        return height;
    }

    @Override
    public int getArea()
    {
        return width * height;
    }
}

class Square extends Rectangle
{
    public Square(int side)
    {
        super(side, side);
    }
}
```

דוגמא לריצה

```
Rectangle r1 = new Rectangle(4,5);
Rectangle r2 = new Rectangle(4,5);

boolean match = r1.getHeight() == r2.getHeight() && r1.getWidth() == r2.getWidth();
System.out.println(match); // true


Rectangle s1 = new Square(4);
Rectangle s2 = new Square(4);

match = s1.getHeight() == s2.getHeight() && s1.getWidth() == s2.getWidth();
System.out.println(match); // true
```

Interface Segregation Principle

אין להכריח לקוח להיות תלוי בממשק שהוא אינו משתמש בו

יש להחליף ממשק "שמן" בממשקים "רזים",
כל אחד מותאם ללקוח ספציפי

דוגמא

```
public interface ILogLocation {
    void setLogName(string _logName);
    string getLogName();
    void Log(string message);
    void ChangeLogLocation(string location);
}
```

```
public class DiskLogLocation implements
ILogLocation {
    private string logName;
    void setLogName(string _logName){
        this.logName = _logName;
    }
    string getLogName(){
        return this.logName;
    }
    public DiskLogLocation(string _logName) {
        logName = _logName;
    }
    public void Log(string message) {
        // do something to log to  a file here
    }
    public void ChangeLogLocation(string location){
        // do something to change to a new log-file
path here
}
}
```

המשך דוגמא

```
public interface ILogLocation {
    void setLogName(string _logName);
    string getLogName();
    void Log(string message);
    void ChangeLogLocation(string location);
}
```

```
public class EventLogLocation implements ILogLocation {
   private string logName;
    void setLogName(string _logName){
        this.logName = _logName;
    }
    string getLogName(){
        return this.logName;
    }
   public EventLogLocation() {
         logName = "WindowsEventLog";
    }
    public void Log(string message) {
        // do something to log to the event-viewer here
    }
    public void ChangeLogLocation(string location) {
        // we can't change the location of the windows event log,
        // so we'll simply return
        return;
```

הבעיה

```
ArrayList<ILogLocation> logLocations = new ArrayList<ILogLocation>() {
        new DiskLogLocation("DiskWriteLog"),
        new DiskLogLocation("DiskReadLog"),
        new EventLogLocation(),
        new DiskLogLocation("FileCheckLog")
};

for(ILogLocation logLocation : logLocations){
        logLocation.ChangeLogLocation("c:\" + logLocation.getLogName() + ".txt");
}
```

פתרון

```
public interface ILogLocation {
    void setLogName(string _logName);
    string getLogName();
    void Log(string message);
}


public interface ILogLocationChanger {
    void ChangeLogLocation(string location);
}
```

המשך פתרון

```
public class DiskLogLocation implements ILogLocation, ILogLocationChanger{
    private string logName;
    void setLogName(string _logName){
        this.logName = _logName;
    }
    string getLogName(){
        return this.logName;
    }
   public DiskLogLocation(string _logName){
      logName = _logName;
   }

   public void Log(string message){
      // do something to log to  a file here
  }

   public void ChangeLogLocation(string location){
      // do something to change to a new log-file path here
  }
}
```

המשך פתרון

```
public class EventLogLocation implements
ILogLocation
{
    private string logName;
    void setLogName(string _logName){
        this.logName = _logName;
    }
    string getLogName(){
        return this.logName;
    }
    public EventLogLocation(){
        LogName = "WindowsEventLog";
    }

   public void Log(string message){
      // do something to log to the event-viewer here
  }
}
```

Dependency Inversion Principle

1. מודולים ברמה גבוהה אינם צריכים להיות תלויים במודולים ברמה נמוכה

שניהם צריכים להיות תלויים באבסטרקציה

2. אבסטרקציות אינן צריכות להיות תלויות בפרטים

פרטים צריכים להיות תלויים באבסטרקציות

דוגמא

```
public class Message {
}

public class GmailBox {
    public Message[] getMessages() {
        return new Message[5];
    }
}

public class MailPrinter {
    private GmailBox mailBox;

    public MailPrinter(GmailBox gmailBox) {
        this.mailBox = gmailBox;
    }

    public void print() {
        for(Message message : mailBox.getMessages()) {
            System.out.println(message);
        }
    }
}
```

פתרון

```
public interface Message {

};

public interface MailBox {
    Message[] getMessages();
}

public class GmailBox implements
MailBox {
    @Override
    public Message[] getMessages() {
        return new Message[5];
    }
}

public class YahooBox implements
MailBox {
    @Override
    public Message[] getMessages() {
        return new Message[5];
    }
}
```

```
public class GoodMailPrinter {
    private MailBox mailBox;

    public GoodMailPrinter(MailBox
mailBox) {
        this.mailBox = mailBox;
    }

    public void print() {
        for(Message message :
mailBox.getMessages()) {
            System.out.println(message);
        }
    }
}
```