

## 1. מערכת | מודל | הפשטה:

- **מערכת** – אוסף יישויות ממשיות או מופשטות, המקיימות קשרי גומלין או תלות הדדית ויחדי יוצרות שלם המשרת מטרת משותפה. למערכת תמיד יש מטרה (אחת או יותר)!
- **מודל** – מודל הוא הבסיס לפיתוח תוכנה. ייצוג מופשט של תופעה או רעיון, של אלגורים, חישובים ותהליכיים.
- **הפשטה** – עיקר, תמצית. ניתן להגד שכל מערכת בנויה מהפשטה, בעלת גבולות, מבנה (OOP למשל), יחסים פונקציונליים – מקבלת קלט ומחזירה פלט (יש לכל מערכת התנהגות), ובעלת מטרה.
- **מערכת סגורה** – מערכת שאין לה קשור עם הסביבה החיצונית, היא רק בתיאוריה.

## 2. הנדסה לעומת אמונות כתהיליך יצירה:

הנדסה ואמונה עוסקות ביצירה, אז מה למשה ההבדל? הנדסה מtabסת על דיסציפלינות "ידועות" (כמו חוק הפיזיקה לדוג'), בעוד אמונה אינה בהכרח מtabסת על דיסציפלינות כלשהן. ואחרות. בתהיליך היצירה הנדסי אנו מנסים לצמצם את השונות- זאת אומרת לעשות את הדברים כמה שיותר כללים ע"פ כללי אכבע אלה ואחרים, לדוג- חברת פיסבוק מפתחת הרבה מוצריהם (הנדסיים) פיסבוק שואפת שהשונות בין תהליכי הפיתוח יהיו מינימליים כדי שיכלו בסופו של דבר לנכנר את כל התהיליכים. לעומת זאת באמונות ישנה שאיפה למגוון רב ולמעשה יש עידוד של שונות.

## 3. מאפייני הנדסת מוצר תוכנה וארגוני אופייניים:

- ראשית נפרט לגבי מוצר תוכנה, מוצר תוכנה מסווק ב-3 תצורות:
- **מוצר עצמאי**: מגיע בדיסק קשיח או בהורדה מהאינטרנט, מיועד לשימוש על מחשב המשתמש ומספק מערכת הפעלה ותוכנה תשתייתית. דוגמא: מערכת Office.
  - **מוצר משובץ**: (embedded) מגיע ביחיד עם החומרה המתאימה לו ספציפית, יש בו מערכת הפעלה ותוכנה תשתיית מוצר שלם. דוגמא: טלפון סלולרי.
  - **תוכנה כשירות (SaaS = Software as a Service)**: התוכנה עצמה אינה מגיעה לקו וAINה נשארת ברשותו, אלא הוא מקבל שירותים הדרושים באמצעותה. דוגמא: Gmail, Dropbox. מאפיינים:
    - מרכיבות הדורשת לבדוק צוות מספור דיסציפלינות (ידע הנלמד/מחקר באקדמיה).
    - אינטראקטיבית ותקשרות בין מגוון בעלי מקצוע.
    - דינמיות וחדשנות בכל אספקט אפשרי.

ארגוני אופייניים:

- **גורם אנושי** - חօר תקשורת, אינטראקטיבים מנוגדים, אי שימוש בסטנדרטים הנקבעים מראש, חօר מקצועית.
- **טبعו של עולם** - לחץ זמני ותקציב, דרישות משתנות ואי יציבות בהגדרת הצריכים.

פתרונות:

- קבלת המיצאות והתאמאה פרקטית אליה.
- שילוב שיטות תכנון אדרטיביות וכלי תומכים.

## 4. סוגי פעילות בהנדסת תוכנה

- **ייזום** – זיהוי בעיה והזדמנויות (תכלס לחשב על רעיון חדש).
- **הגדרת דרישות** – תיאור מדויק של הנדרש (למיושר הרעיון).
- **ניתוח** – כיצד ניתן לפתור את הבעיה (התעמקות).
- **עיצוב** – בחירה ותכנון הפתרון המתאים.
- **IMPLEMENTATION** – תרגום התוצאות למציאות.
- **בדיקות** – בדיקות התוצאות מול התכנון.
- **הטמעה** – התקנת המערכת והטמעה אצל המשתמשים.
- **תחזקה** – תהיליך מתמשך של ניפוי שגיאות והרחבות (תחזקה שוטפת של המוצר).
- **וגם** : דיוונים, הרכות סיכונים, תיעוד, תיעוד, ארכיטקטורה ועוד.. (דברים פשוטים עושים בתחזקה, זהה בתכלס מה שחברות עושות ביום יום נוראה).

**5. סוג תפקידים בהנדסת תוכנה****בעל עניין:**

- אדם בעל אינטראס חיבי או שלילי יכול להשפיע על התפתחות הפרויקט.
- גורם שמעורב "אחרוי הקלעים" בצורה אקטיבית לדוגמא: ל��וח, משתמש, משקיע, ספק, רגולטור.

**מנתח מערכות:**

- מנתח את המצב הנוכחי (מרחב הבעה) ומבין את הביעות הדרשיות.
- מגדיר אפיון למרחב הבעה וקיים מתאר לפתרון.
- תוחם ומחדד את מטרות המערכת ומגדיר את דרישותיה.
- אוסף ומוסוג דרישות מלוקחות ומשתמשים.
- תוצאות פורמליים - מסמך ייזום, ניתוח מצב קיימן, הגדרת דרישות, איפיון פתרון, דיאגרמות.

**מנהל פרויקט:**

- אחראי לעמידה בביצועים הנדרשים, בתקציב ובלו"ז.
- בקיא בתהליכי ניהול פרויקט מקובלים, מתאים את התהליכי לפרויקט הספציפי ודואג ליישום.
- נדרש ליכולות ניהוליות – יודע לארגן, לחשוב ולהניע אנשים.

**מנהל מוצר (PO):**

- דואג לאנטרסים של הלוקה.
- קולו של המשתמש בצוות הפיתוח – מסתכל מהצד של המשתמש.
- מביא את המוצר הנכון למשתמשים – מעין רפרנס בין הלוקה למפתח.
- קובע חזון לאחר חקר שוק, הבנת הלוקה וצריכיו והגדרת אופי הבעה.
- מגדיר תוכנית פעולה ומפת דרכיים לפיתוח איטרטיבי לטובות מימוש מלא של החזון.
- עובדcznie לצוות הפיתוח – מפקח על התוכולה, על הקצב ועל פתרון הביעות.
- כאשר הגרסה יוצאת תפקידי לעקב ולקבל פידבקים מהמשתמשים.
- בעל תפקיד אסטרטגי – קובע את הכוון, لأن החברה הולכת.

**הנדס מערכת:**

- אחראי טכנולוגית על הפיתוח.
- קובע את מיפוי הטכנולוגיות שברם המערכת תשמש.
- בעל ראייה מערכתי – בקיא ומכיר את כל התהליכי והקשרים ביניהם.
- יודע להסתכל על המערכת גם מבחינת חומרה וגם מבחינת תוכנות.
- משולב בפרויקטים הנדרדים רחבי היקף בכל שלביים.

**ארכיטקט תוכנה:**

- מי שמעצב ומיצר את המערכת.
- אחראי על High level design.
- בוחר את תשתיות התוכנה.
- אחראי על חלוקת יישום מורכב למודולים.
- מגדיר את כל העיצוב והפיתוח (איך מօספים רכיב חדש? איך מאפיינים רכיב חדש?) וublisher אותם למפתחים.
- פונקציונלי – מבין מה כל חלק עשוה ואת הקשרים והתלות בין כל חלק המערכת.

**ראש צוות:**

- אחראי על Low level design.
- בונה את הצוות – בוחר את האנשים.
- מנהל מס' אנשים בעלי תפקיד זהה.
- תיאום עבודות הצוות.
- קביעת נחיים, ביצוע code review.
- מתואם עם גורמי ניהול הפרויקט.

**מתמיע מערכת:**

- אחראי על כתיבת מדricsים טכניים למשתמש.
- מתקן את המערכת במחשבים השונים בחברה (משר זמן ההטמעה תלוי בגודל המערכת).
- נמצא בקרה מתמדת על פעילות המערכת.
- מעניק שירותים ולויו שוטפים לעובדים בתפעול התוכנות השונות.

**מומחה יישום:**

- נציג מטעם הלוקה אשר מהווה משתמש עיקרי של המערכת אשר נבחר כבר בשלב הייזום.
- אחראי ללוות את המערכת לאורך כל השלבים, לארגן את דרישות המערכת ובניהן בדיקות קבלה ומסירה.
- מאשר את קבלת המערכת.

## 6. סוגים ארטיפקטיים בהנדסת תוכנה

ארטיפקטים, אלו תוצריו הלואוי של פיתוח התוכנה.

לדוגמה: דיאגרמות UML של הרכבת סיכונים, Use Case, קוד המערכת.

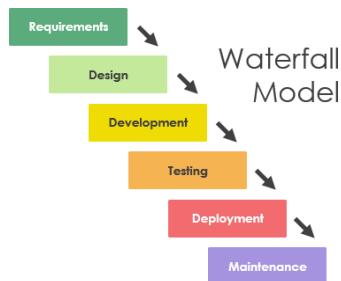
## 7. Stakeholders ופוטנציאלי מעורבותם:

**Stakeholders** - אדם בעל עניין או אינטרס שנמצא בתחום המערכת והרבה פעמים משנה את כל קבלת החלטות בשלב תכנון ויצירת התוכנה, מה שיכל להשפיע על ביצוע הפרויקט והשלמתו.

## 8. סוג תהילכים ומאפייניהם (لينארי, אינקרמנטלי, ניהול סיכונים):

## 1. ללא תהלים:

- קודד ותיקן (code & fix)** - מתודולוגיה ששמה דגש על מהירות, מציאת הביעות תוך כדי ניסיון התיקון בדרך של ניסיון וטעייה.



## 2. תהלים מתוכננים:

- מפל המים** - מתודולוגית פיתוח שדוגבת במחזור פיתוח אחד. לפני שמהלכים יש ליצור פורמליזציה נרחבות של כל הדרישות. שמים דגש רב על עיצוב מוקדם של התוכנה לפני שמקודדים, ורק בסוף יוצרים בדיקות.

יתרונות: תהלים מתואuden היבט, תחזקה קלה, מסודר.  
 חסרונות: צריך להציגם לתכנון הראשוני ואין מקום לשינויים.

## 3. מודל V - שם דגש על קשר הדוק בין רמת ההגדלה והפירוט לבין רמת המימוש והבדיקה.

- מאפייני התהלים: התהלים מגדר 4 רמות פיתוח, וכל אחת בדיקות מתאימות כוגן:

1. בדיקות יחידה - כדי להיפטר מבאגים בקוד.
2. בדיקות אינטגרציה - לוודא כי היחידות שנוצרו באופן עצמאי מתחברות.
3. בדיקות מערכת - מבטיחה כי הציפיות של הלוקוח מהישום מתקיימות.
4. בדיקות קבלת משתמשים - מודוא שמערכת שהובעה עומדת בדרישות המשמש.

יתרונות: (1) חסcoon בכיסף ובמשאבים עקב תחילת תהלים הבדיקות בשלב מוקדם של הפרויקט. (2) כיסוי מקסימלי של בדיקות.

חסרונות: (1)התארכות זמן הפיתוח. (2)חוסר ודאות. (3)סיכון גבוה. (4)אין גמישות לשינויים במהלך הסבב.

## 4. אב טיפוס - דגם של המוצר. משמש להדגמה בלבד, הוא לא המוצר הסופי!

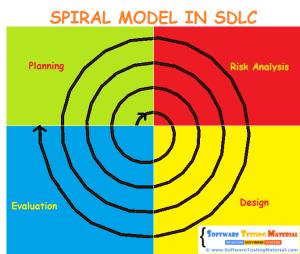
המפתחים בונים אב טיפוס בשלב ניתוח הדרישות. משתמשי הקצה מספקים מושב לתיקונים ולפי זה המפתחים משפרים את אב הטיפוס.

כאשר משתמשי הקצה מודים, משלימים את השאר כדי להפוך אותו למוצר סופי.

שימושי במיוחד כאשר דרישות המשתמש לא מלאות ונושאים טכניים לא לגמר ברורים.

יתרונות: (1)הילוקחות יכולות לראות תרגום הדרישות למשהו מוחשי עוד בשלב ניתוח הדרישות. (2)המפתחים לומדים ממשות המשמשים. (3)איתור דרישות ופונקציונאליות שלא היו צפויות. (4)אפשר עיצוב גמיש. (5)קל לראות התקדמות.

חסרונות: (1)ביטחעים לא-יעילים. (2)קיימת נטייה לאובדן מבנה התהלים. (3)ארכיטקטורה וראייה מערכתית אינם זוכים למקום מרכזי. (4)מתמקדים בראצוי מיידי ודרישות תחזקה נדחקות לשוליים. (5)קיימת סכנה לתהלים שלא נגמר (Creep Scope).



## 5. תהלים איטרטיבי / אינקרמנטלי:

**spirali** - פיתוח המערכת בגישה הסpirali נעשה בכמה סבבים (כל סבב יכול להימשך חודשים רבים בין שישה חודשים לשנתיים). הפיתוח מחולק לארבעה שלבים:

- הגדרת מטרות, חלופות ואילוצים.
- ניתוח חלופות, זיהוי סיכונים ומציאת פתרונות.
- פיתוח והטמעה (שחרור גרסה).
- תכנון הסבב הבא.

יתרונות: (1)ניתוח הסיכונים שקיים מעתים תוך כדי הפיתוח. (2)הפיתוח מתחילה מוקדם בתהלים.

חסרונות: (1)מתאים לפרויקטים גדולים כיוון שעלות הניתוח גבוהה. (2)הצלחת הפרויקט תלויות בעיקר בשלב ניתוח הסיכונים. (3)לא מתאים לפרויקטים קטנים.

## 4. תהליכי מסתגל:

- Scrum - מפורט למטה
- Agile - מפורט למטה
- Kanban - מודל מבוסס על שיטת agile הנוטן דגש לצד הויזואלי של התהילה- מה לייצר, מתי לייצר וכמה לייצר. השיטה מעודדת ביצוע שינויים קיטנים והדרוגתיים במערכת.



התהילה מגביר את הזמן צריך לעדכן את הולחן ואון בלוח מידע לגבי זמני המסירה של המשימות, אך כל הזמן צריך לעדכן את הולחן ואון בלוח מידע לגבי זמני המסירה. XP - מודל המבוסס על שיטת agile ומאפשר מעורבות גבוהה של הקהקה בתהילה הפיתוח. כאשר התוכנה תענה על צרכי הקהקה בצורה מיטבית. XP משתמש בפרקטיקות הבאות:

1. Programming Pair: זוג מתכנתים על אותו מסך מחשב, כתובים יחד את הקוד.

2. Design Simple: קוד פשוט. ללא קוד כפוף, עם קוד מובן למתכנתים, מכילה את המינימום הנדרש של מחלקות ושיטות.

3. Refactoring: שיפור קוד קיים על ידי שימוש בטכניקות שנעודו לשפר את המבנה הפנימי של הקוד מבליל לשנות את התנהלות החיצונית שלו.

4. TDD: בדיקת יחידה הנכתבת לפני כתיבת הקוד אותו היא בודקת.

## 9. הנדסת דרישות (ס'אג'ג דרישות, تعدוף)

**דרישה אינטואיטיבית** צריכה להיות בדידה, ברורה (מנוסחת בשפת הלוקוח), חד משמעות, שלמה, נצרכת, לא סותרת דרישות אחרות, ניתנת לבדוקה באמצעות טסטים, ניתנת למעקב, מתעדפת.

**דרישות משתמש** - משפטים בשפה טבעית המלויים בדיאגרמות, השירותים שהמערכת תספק ומסגרת אילוצים – התיאור הוא עבור משתמשים. דוגמ: על המערכת לספק דרך לקרו מסמכים שנוצרו במערכות אחרות.

**דרישות מערכת** - מסמך מבנה הפורס לתיאור מפורט של פונקציות המערכת, שירותיה ואילוצי התפעול, מתאר מה יש למשב באופן המאפשר להשתמש בו כבסיס לחוזה / פיתוח / בקרה / בדיקות. דוגמ: המערכת תאפשר למשתמש לבחור ולקשר בין סוג מסמך לעורףpecific. לכל סוג מסמך יופיע סמל אחר והמשתמש יוכל לקבוע מהו הסמל.

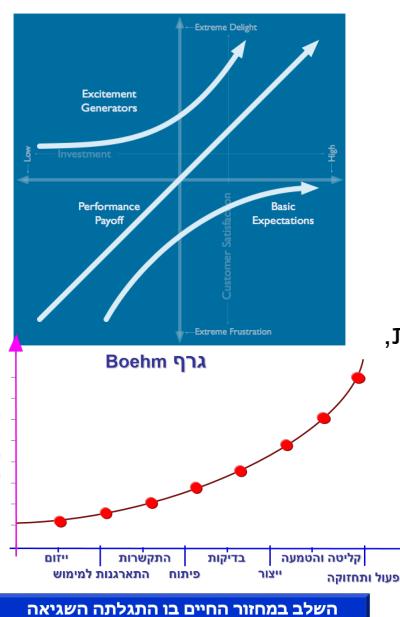
**דרישות פונקציונליות** - משפטים אודות השירותים שהמערכת תספק, כיצד המערכת תגיב לסוג קלטים ולמצבים נתונים ומהם הפליטים. בשורה התחתונה, מה המערכת עשו ולא איך היא עשוה זאת. דוגמ: על המערכת לספק דרך להזמין פיצה טעינה במיוחד.

**דרישות לא פונקציונליות** - אילוצים על השירותים או הפונקציות שהמערכת מספקת כגון חלונות זמן לTAGובה, נפחים מקסימליים, מספר משתמשים בו זמן. אילוצים כמו אבטחה וכו'. דוגמ: המערכת תתבסס על מחשב מסווג.....

**דרישה פעולהית - (OR = Operational Requirement)** - דרישת המתיחסת לתפעול, לאינטראקציה או להתנהגות של המוצר. דוגמ: המערכת תציג תמונה של המשתמש.

## דרישת מידע - (DR = Data Requirement)

- דרישת המתיחסת לשויות המידע ולנתונים בהן נדרש התוכנה לטפל (לקולות, לאחסן, לאבד, להפיק כפלט).
- לדוגמה: נתונים ומבנה נתונים, מאגרי מידע / בסיס נתונים, דרישות קולט/פלט.



**תעדוף דרישות:** הרעיון העיקרי הוא שהדרישות לאו דווקא מתועדות ע"פ שביעות רצון של הלוקוחות. **מודל Kano**: מוגדר כטבלה Kano: מערכת צירם של מודל Kano:

## 10. Separation Of Concerns/Coupling/Cohesion

עקרון עיצובי שבו כל יחידה (לדוגמ' יחידת קוד) היא בעלת תפקיד מוגדר ויחודי.

תוכנית המגלהת היבט את עקרון SoC נקראת תוכנית מודולרית.

**יעיפוד:** עיקרון עיצובי להפרדה של תוכנית מחשב למתקעים נפרדים. כל סעיף מתיחס לדאגה נפרדת, אוסף מידע המשפיע על הקוד של תוכנת מחשב. דאגה יכולה להיות כללית כמו "פרט החומרה של אפליקציה", או ספציפית כמו "השם של איזו מחלקה להפעיל". מודולריות מושגת על ידי קיבולת מידע בתוך קוד שיש לו משקל מוגדר היבט.

## 11. חוק/עיקומת בוהם, חשיבות שלבים (ນיתוח, עיצוב, פיתוח, בדיקות, תחזוקה)

עקרון בוהם - איתור שגיאה בשלב לאחר תגרור עלות תיקון גודלה יותר.

## 12. UML (מהו. הגדרות, סוג תרשימים)

הטיוי בשפה הוא גרפּי ומאפשר תיאור מופשט של מפרטי המערכת, בדרגות שונות של דיק.

בשל אופיה הוויזואלי, UML היא שפה קלה-יחסית ללמידה, בהשוויה לשפות מפרט אחרות. לרוב, כל בעלי התפקידים בצוות פיתוח תוכנה מכירים את הדיאגרמות העיקריות בשפה, והדבר מסייע לקשר את מפרט התוכנה בינם. ב-UML קיימים מספר סוגים תרשימים (דיאגרמות), המשמשים למטרות שונות, ומתחלקים לסוגים שונים. התרשימים השונים מורכבים משלשות אבני בניין עיקריות, גורמי יסוד, יחסיים בין גורמי היסוד ודיאגרמות - התרשימים שמאגדים את גורמי היסוד והיחסים לגורמים בעלי משמעות.

1. גורמי יסוד - גורמי היסוד (elements) מתחלקים לארבע קבוצות עיקריות:

- **גורמים מבנים (structural):** אלה הם "شمוט עצם" המרכיבים כל מודל ואשר בדרך כלל מתארים גורמים פיזיים או קונספטואליים. לדוגמה שחקן (Actor), שמוסמן על ידי דמות אדם.
- **גורמים התנהגותיים (behavioral):** אלה הם החלקים הדינמיים של מודל UML אשר מוגדרים כ"פעלים" של המודל. גורמים אלו מגדירים את ההתנהגות של גורם מסוים בזמן, או ביחס לגורמים אחרים במודל.
- **גורמים מקבצים (grouping):** אלה הם החלקים שמאפשרים ארגון טוב יותר של המודל. הגורם הנפוץ ביותר השיר לקבוצה הוא package. גורם זה הוא מעין ריבוע שמאפשר לארגן בתוכו גורמים מסוימים לקבוצה אחת שאיתה ניתן לקובץ גורמים התנהגותיים, מבנים ואף גורמים מקבצים נוספים. בין הגורמים השימושיים לקבוצה זו: מודול, מערכת (framework), תת-מערכת (sub-system).
- **גורמים מפרשים (annotation):** גורמים אשר בעזרתם ניתן להסביר ולפרש גורמים במודל או מודל UML. הגורם העיקרי בקבוצה זו הוא הערה (note). בהערה המשויכת לגורם, ניתן לרשום הערות בטקסט חופשי או אילוצים (constraints) המתיחסים לגורם ומגדירים עליו אי-allo הgalbotot.

2. יחסיים - היחסים (relationships) מגדירים את היחסים והקשרים בין האלמנטים. אף הם מתחלקים ל-4 סוגים:

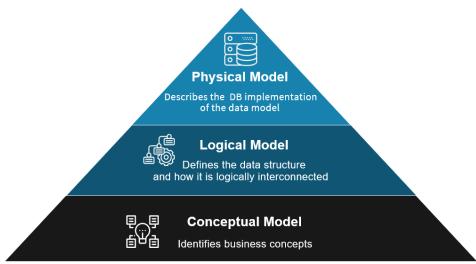
- **תלות (dependency):** יחסי סמנטי בין שני גורמים. משמעו הוא ששני גורמים תלויים זה אחד על השני. עשוי להופיע על העצם התלויה בו. תלות מסומנת באמצעות חץ מקוווק.
- **חיבור (association):** מתווגם לחבר בין שני גורמים. מסומן באמצעות חץ פשוט.
- **הכללה (generalization):** באופן אינטואיטיבי ניתן לחשב על יחסי זה כעל "הורשה" של מחלקות, כאשר האב הוא הגורם המכלי (generalized), והבן הוא הגורם "מיוחד" (specialized). מסומן באמצעות חץ עם מושולש בקצתו.
- **מימוש (realization):** זהו קשר סמנטי, שימושו גורם אשר פועלתו מבוצעת על ידי גורם אחר. בדרך כלל משתמשים ביחס זה בין ממשק (interface) לבין המחלקה המIMPLEMENTת אותו. מסומן באמצעות חץ מקוווק עם מושולש בקצתו.
- **composition:** יחסי הכללה חזק - אובייקט שלא יכול להתקיים בלי אובייקט אחר. יש בהם יחס הכללה חזק. מסומן עם מעין מלא. (לדוגמה, דירה ספציפית בתוך בניין, אם הבניין נהרס אין דירה).
- **Aggregation:** יחס הכללה חלש - כמו מוקדם רק שכן יכול להתקיים גם בלבד. מסומן עם מעין ריק (לדוגמה, איש וקבוצה של אנשים).

3. דיאגרמות - הדיאגרמות מאגדות את גורמי היסוד והיחסים לגורמים בעלי משמעות.

- **דיאגרמות מבניות:**
  - דיאגרמת מחלקה (Class diagram) - מתארת את מחלקות התוכנה ואת היחסים ביניהן.
  - דיאגרמת רכיבים (Component diagram) - מתארת את רכיבי המערכת כדוגמת טבלאות וקבצים.
  - Composite structure diagram
  - Deployment diagram - מתארת את פרישת המערכת באופן פיזי אצל הלוקו.
  - Object diagram - תרשים המציג תצוגה מלאה או חלקית של המבנה של המערכת ממודלת בזמן מסוים.
  - Package diagram
  - Profile diagram
- **דיאגרמות התנהגותיות:**
  - Activity diagram - מתארת תהליכי המתרחשים בתוכנה בהתאם לפעולה של המשתמש.
  - State diagram - מתארת את העצם במצבים שונים.
  - Use case diagram - מתארת את השימוש במערכת.
- **דיאגרמות אינטראקטיביות:**
  - Collaboration diagram (Collaboration diagram) - דיאגרמה זו חלופית לדיאגרמת רצף.
  - Communication diagram
  - Interaction overview diagram (Interaction overview diagram)
  - Sequence diagram (Sequence diagram) - מתארת קשר בין עצמי המערכת על ציר הזמן.
  - Timing diagram (Timing diagram)

הגדירה של Scope Creep היא שכאשר היקף הפרויקט משתנה, עובdot הפרויקט מתחילה להתרחב, או "לזחול", מעבר למה שהווסכם במקור. זה יכול להתறחש כאשר היקף הפרויקט אינו מוגדר, מתועד או מבוקר כהכל. זה נחשב בדרך כלל כמוין.

בנוסף Scope Creep גורמת לרוב חרגת עלויות. זה דומה אך שונה מ-feature creep, מכיוון ש-feature creep מתיחס לפיצרים, ו-scope creep מתיחס לכל הפרויקט. הסיבות השיכחות ביותר ליותר-ליותר-ל-scope creep הן: היקף פרויקט המוגדר בצורה גרועה, חוסר בשיטות ניהול פרויקטים, הוספה של פיצרים מיותרם, פער תקשורת בין בעלי העניין בפרויקט.



#### 14. שלושת רמות מדול נתונים ומה מייצג תרשימן ER?

##### מודל קונספטואלי -

- תיאור סמנטי "טהור" של תחום הבעה (משמעות).
- תיאור יישויות (Entities) (תכונות (Attributes) וקשרים (Relationships)).

##### מודל לוגי -

- יציג תחום הבעה במושגי טכנולוגיות, ניהול הנתונים (לדוגמ: טבלאות, רשומות ER).
- כיצד הנתונים נשמרים, במסגרת מאפייני ניהול RDBMS (מדוד נתונים רלוונטי כמו SQL).

הרעין בכלל הוא שהמודל הקונספטואלי מהוות בסיס למודל שלבי חשבנו, לאחר מכן אנו מארגנים את הרעיון בצורה מסודרת בטבלאות או רשומות (מודל לוגי) לבסוף שומרים את כל הנתונים במודל פיזי כלשהו כמו SQL.

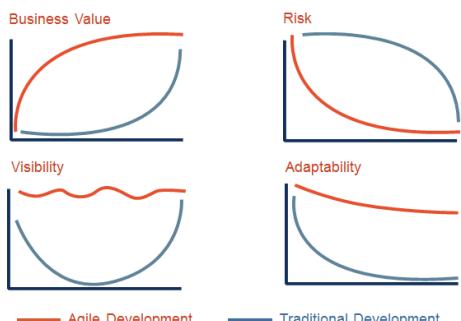
##### מודל ישות קשרים (ER – Entity Relationship):

מדובר על הגדרת הקשרים בין ישות מסוימות במערכת. זאת אומרת לאחר שהגדכנו את הישויות במערכת, ניתן לנרצה את הקשרים ביניהם. אפשר לעשות זאת באמצעות תרשימים כמו ERD.

קייפידי: דרך להציג מידע. הצגה עצת אפשרות תכנון מלמעלה-למטה של מערכות מסדי נתונים רלוונטיות. המודל אינו מתחשב בארכיטקטורת המחשב עליו יורץ מדד הנתונים אלא רק במבנה הלוגי הרצוי של מדד הנתונים באופן שיאפשר נוחות ויעילות בגישה למידע.

##### סוגי קשרים:(פרק דוגמאות)

1. יחיד לחיד: כיתה -> מורה מחנכת, מורה מחנכת -> כיתה (לכל כיתה יש מורה מחנכת אחת ולכל מורה מחנכת יש כיתה אחת שהיא המורה המחנכת שלה).
2. יחיד לרבים: כיתה -> תלמידים , בכיתה (יחיד) יש הרבה תלמידים (רבים).
3. רבים לרבים: מורה -> כיתות בהן הוא מלמד, כיתה -> המורים שמלמדים את הכיתה. לכל מורה יש מספר כיתות שבהן הוא מלמד לכל כיתה יש מספר מורים שמלמדים אותה.



#### 15. אגיליות : פילוסופיה, חשיבות, סוגים, קלים, תפקידים, פרקטיקות

##### מהי שיטת-agile?

שיטה ששם דגש מיוחד על ניהול פרויקט באופן שיאפשר יכולת תגובה לשינויי, יעילות, דרישות ואיכות התוכנה הן מבחינות המuna שנitin לצרכי הלוקוט והן מבחינות העדר תקלות ובאגדים. הגישה מאמיןה כי אי אפשר להגדיר או לחזות מראש את כל הצליכים של תוכנה לפני תחילת הפיתוח בפועל. הגישה מתאימה בעיקר לפרוייקטים בהם ישנה חוסר ודאות לגבי המוצר הסופי, מוחר והיא משלבת תהליכי למידה ושיפור מתמידים, תוך כדי התקדמות הפרויקט.

Agility - שילוב של גמישות וזריזות – השארת הקוד פשוט, עבודה על חלקים קטנים שתמיד עוברים תיקוף של הלוקוט ובודקوت – מצמצמים אי ודאות ולהבטיח אי-איכות.

Agility מرتبط גם באיכות הוצאות להיות גמיש זריז ובעל תקשורת בריאה וטובה – מעין צוות רב זרועי, כלי אחד המכיל הכל כמו אולר שווייצר וזו הסביבה והוצאות שנרצה ליצור. נctrkr פתרון מבוסס agility יכול פותר ומונע את הסיבות לכישלון.

##### עקרונות אגילים :

- הפשטה ממושכת ומקדמת ככל האפשר לתוכנה בעלת ערך.
- הפשטה גרסאות תקופה.
- תוכנה עובדת זה העקרון הבסיסי ביותר.
- תקשורת ישירה ללא מחסומים.
- צוות המורכב מאנשי מוכשרים ובעלי מוטיבציה.
- סומכים על צוות הפיתוח.

- עבודה בקצב סביר, לא להגיע למאכז של לחץ והתפוצצות.
- מתן סמכיות לצוות ולהתארגנות עצמאית.
- לא לפחות שהוצאות נכשל – מהכי שלונות נבנים ומפותחים (בנהה שהכי שלון לא נבע מטיפשות ורשלנות).
- פשוטות.
- חתירה למציאות.

#### תהליכי אג'יל:

- תהליכי איטרטיבי – איסוף אינפורמציה ע"י סקר ולמידת השוק, הערכת סיכונים, איסוף וארגון דרישות – כל הדברים שצורך לעשות, המרת הדרישות לפיצרים (לכל פיצ'ר נציג CASE USE) ותיעודם, יצירת ארכיטקטורה ב-level High ופיזול צוותים.
- תהליכי איטרטיבי וinemental – פיזול הפיצרים ל stories (יחידה פונקציונאלית שפיתוחה ובדיקה אותה לוחק כשבועיים), הערכת זמני פיתוח, דו שיח עם הלוקה באופן שוטף ותיעוד הפיתוח ע"י הלוקה, פיתוח מבוסס TDD, הערכת הסטטוס שלו על בסיס מצב הקוד ושקיופתו ללוקה.

#### פרקטיקות ורוצחות אג'יליות:

- מטאפורה – הלבשת רעיון חדש על ידע קיים – בסיס לחשיבה ופיתוח דיון אודות המוצר.
- User story – מייצג פיצ'ר אחד או יותר במערכת, קטן ומדויק.
- Sustainable Pace – דאגה לצוות רענן ואפקטיבי במשך כל זמן הפרויקט – הימנעות משחיקה.
- Customer Team Member – ראה מנהל מוצר בהרצאה 3.
- Planning Game – תהליכי ששותפים לו גם הלוקה וגם צוות הפיתוח כאשר המתכנתים מעריכים את משקל הפיצרים והлокה מתעדף את הפיתוח שלהם.
- Simple Design – השארת העיצוב כמה שיותר פשוט ונוח לשינויים ותוספות.
- Small Releases – שחרור גרסאות קטנות ולא גרסה אחת גדולת בבת אחת.
- Test Driven Development – לכל יחידת קוד נגידר סט בדיקות לפני כתיבתה בפועל (כתיבה מונחת בבדיקות TDD).
- Collective Ownership – הקוד באחריות כל הצוות, כולם אחוריים לשלאוט בו.
- Coding Standards – סטנדרטים בריאים לככיתבת קוד – אחידות ווגנון.
- Refactoring – המרת מבנה קוד קיים מבלי לשנות את הפונקציונליות, התהילך שומר על העיצוב המקורי.

#### 16. - Burn Down Charts (תרשים שריפה):

כל' המשמש את צוותי agile כדי לאסוף מידע על העבודה שהתבצעה עד עכשיו ועל העבודה העתידית שצריכה להתבצע בפרק זמן הנtent (צוותי-h-scrum מכנים אותה sprint). לעיתים קרובות, צוותים יכולים להשתמש בו ככל הצורך המאפשר להם לדמיין מתי הפרויקט שלהם יסתיים. הכל' מספק גם תובנות לגבי בריאות הספרינט שלהם. לאחר השתקפות של תריסים השריפה, הצוותים מקבלים תובנות לגבי צוואר' בקבוק בתהליכי ולאחר מכן הם מסוגלים לקבוע פרתרונות למושלים, המובילם לתוצאות משמעותיות. Dog: JIRA.

#### 17. Scrum :

##### מהי שיטת scrum ?

טכניקה לתהילך פיתוח זרי המבוסס על עקרונות agile. בטכניקה זו יש דגש על פיתוח איטרטיבי- עבודה בספריינט, עבודה מרוכזת עם שקיופת ושיפור.

משתמשים בשיטה זו בכל מקום החל מארגונים גדולים ועד סטראטפים קטנים.

שיטת זו מקדמת את הערכאים : מוטיבציה, עבודה לצוות ותקשורת, העצמת הפרט, תרבויות אירוגניות והגברת קצב הפיתוח. sprint: תקופה מסוימת בה מחליטים על משימות ויעדים שבסיומה נגייע אליהם. לאחר כל סיום sprint יגיע sprint חדש. sprint planning: הכנה לתקופה הקרובה והגדלת המשימות בהם הצוות יעסוק.

מה ניתן לספק בסיום התקופה? כיצד נספק את היעדים עד סיום התקופה?

daily scrum: פגישה יומיית של חברי הצוות בה מתכוונים את העבודה ליום הקרוב.

sprint review: פגישה המתמקדת בסיום כל sprint בה כל אחד מראה ומסביר על התוספות שהכניסו למוצר בתקופה שהסתירה. sprint retrospective: פגישה המתרכשת לפני-h-planning ואחרי-h-review ומטרה לסתן לצוות אפשרות לבדוק את עצמו ואת המוצר שלו ולתת פרטטיביה לגבי התכנון העתידי.

##### חרונות:

- לא ידוע מה הולך להיות התוצר הסופי.
- תפקיד-h-scrum master קשה.
- הטמעת השיטה מצריכה שינוי ברמת הארגן.

##### במודל זה יש מספר תפקידים:

- Scrum master – מעין מנטור שמנטב את הצוות להצלחה אך לא בהכרח בעל הכוח לקבל החלטות.
- Product owner, Developers, Testers הנהלה – כיוון שבמודל זה אין מנהל פרויקט והצוות מנהל את פעילותו אז תפקידו של הנהלה הוא ליצור מודל עסק, לנבד את החוקים והשיטה, לסייע בסילוק מושלים, לאתגר את הצוות למציאות.

## 18. דרכי להתמודדות עם מורכבות

- אבסטרקציה:** התעלמות מפרטים והתקדמות "בתמונה הגדולה".
- דינומיזציה:** פירוק הבעיה לאוסף תת בעיות בלתי תלויות (האם אי תלות באמת קיימת?).
- מודולציה:** הגדרת מבנים יציבים לאורך זמן.
- פרוייקציה:** התקדמות בזווית ראייה אחת כל פעם (View) בהתייחס ל תת בעיה הנבחנת.

## 19. גישות Bottom Up ו-Top Down לעיצוב

Bottom Up ו-Top Down הן אסטרטגיות לפיתוח תוכנה.

במודל **Top Down** מנוסחת תחיליה סקירה מערכת בקווים כלליים בלי להיכנס לפרטים של שם חלק منها. לאחר מכן מנוסח כל חלק של המערכת לפרטיו, ואז כל פרט חדש עשוי להיות מנוסח שוב תוך שmagדים אותו לפרטים נוספים עד שככל המפרט מפורט די כדי לתקף את המודל. מודל מעלה-מטה מעוצב לעיתים קרובות תוך הסתייעות ב"קופסאות שחורות" שמקלות את שימוש סקירת המערכת, אך באופן שאין בו די להבנת המנגנון שביסוד המערכת. ביגוד לזה, במודל **Bottom Up** מוגדרים תחיליה החלקים הפרטיים של המערכת לפרטיהם, ואז מצורפים החלקים יחדיו לריכבים גדולים יותר, שבתורם מצורפים גם הם עד להשגתה של המערכת השלמה. אסטרטגיות מבוססות על זרימת מידע מעלה-מטה עשוות להראות חחניות ומספקות עקב היזון מבוססות על ידיעת כל הגורמים שעשויים להשפיע על חלקיה היסודיים של המערכת.

## 20. עיצוב תוכנה ומודע הוא נדרש (שינויים יגעו, יש להתכוון בעיקר לשלב התחזקה)

עיצוב (תיכון) מבנה תוכנה :

ארQUITקטורת מערכת או ארQUITקטורת תוכנה:

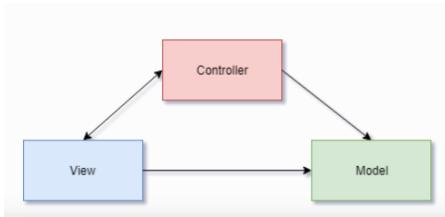
- ארQUITקטורת מערכת –** מציגה מודל שמכיל קונספטים שמתארים מבנה והתנהגות הכללים כתמי מערכות. בנוסף מודל זה מציג את החומרה ואת התקשרות.
  - ארQUITקטורת תוכנה –** מציגה מבנה level high (מבט מלמעלה בזום אוטו) לתוכנה הרצiosa המכילה סעיפים אב כר שלכל שימוש או כתמי מערכות יש אב (מעין עץ).
- בנוסף, הארכיטקטורה מכילה את הגדרות מטרות העיצוב, מודלים מרכזיים, תוכנות מרכזיות והקשרים ביניהם.

הסיבות לחשיבות שלב עיצוב התוכנה:

- התמודדות עם מורכבות:** תהיליך הפיתוח הוא דינامي ומשתנה כל הזמן וכן חשוב שייהיה לנו מבט מלמעלה שמרכז ומפרק את הכל ובכך יוצר יציבות, בנוסף שלב העיצוב מפרק לנו את הבעיות הגadolות לתתי בעיות ובלתי תלויות, בנוסף נבין כי המערכת מורכבת מהמן ישות שמסוגל לאלידוד מרוב מהם הרבה, שלב העיצוב נותן לנו את היכולת להבין את הקשרים ביניהם.
- התמודדות עם מחזור חי תוכנה ותנאי סביבת פיתוח:** כאמור לעיל יש שינויים תכופים כל הזמן אך באופןף נרצה שאורך שלב התחזקה שהוא אחד המרכיבים החשובים ביותר לא יהיה ארוך ומרכזי מעבר לתפקידו (לא נרצה לפתוח מוצר שני ולהשתמש בו חצי שנה) – מבנה ועיצוב נכון ימנעו את זה ויאפשרו אדרטיה מהירה לשינויים.

## 21. (הכוונים המרכזיים אליהם חותרים בעיצוב המערכת) Design Goals

.Architectural Design - הגדרת מודלים מרכזיים, גרנוולציה (רמת פרטים) נמוכה. (דוג: C) MVC מכונה גם .Detailed Design - הגדרת מודלים ברמת גרנוולציה גבוהה. (חלוקת למחלקות בקוד). מכונה גם High Level Design • Low Level Design •



:High Level Design

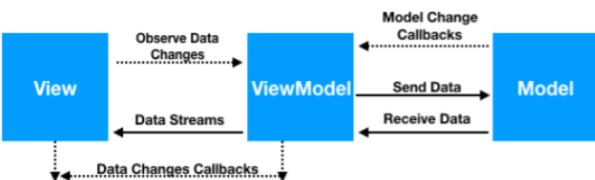
MVC - Model-View-Controller

הפרדה בין הלוגיקה (Model) לתצוגה (View) המודל אחראי רק על הלוגיקה של הנתונים (לדוג' כל המחלקות הרגילות של JAVA שמייצגות משתמשים כמו פרופיל מורה ועוד) והצוגה אחראית על הציגה של הנתונים.

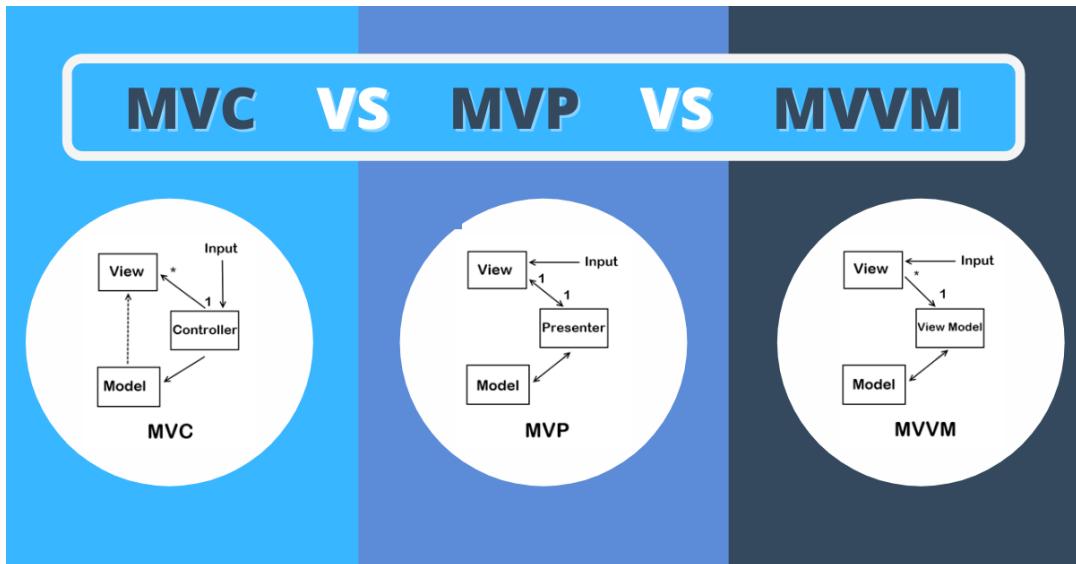
התפקיד של הבקר (Controller) הוא לקשר ביןיהן, איך זה נעשה בפועל? התצוגה מקבלת קלט מהמשתמש (לדוג' לחיצה על כפתור באפליקציה) התצוגה מיידעת את הבקר על מה שairע הבקר מעביר את הנתונים למודל והוא מעבד אותם (לדוג' מוסיף סטודנט לרשימת התלמידים של מורה מסוים).

MVP - Model-View-Presenter

הרעין הוא למעשה למשתמש איזשהו נציג בצוරה של ממשק שיקשר בין כל חלק המערכת. Presenter ייחסק את הממשקים של ה-View ושל ה-Model וכך יתקשר איתם. המודל ייחסק מציביע לממשק של Presenter וכאשר יהיו שינויים, הוא יעדכן אותו באמצעות המצביע (callback). חשוב לציין שזה כמו MVC רק שיש ממשק מקשר.

**MVVM - Model-View-ViewModel**

- מאוד דומה ל-MVC, עקרונות התבנית בניוות מ-Model – מתאר את האובייקטים שמהווים את ה-Data של התוכנית (לוגיקה).
- View – החלק האחראי על הגרפיקה ועל מסך המשתמש (UI).
- כמו כן הוא מחזיק מופע של-ViewModel – החלק אשר מהווה רכיב תקשורת בין החלקים ומחזיק רכיבי לוגיקה של UI ומופע של Model.
- הרעיון הוא הפרדה מוחלטת בין-View לבין-Model כשהתקשרות ביניהם היא באמצעות מנגנוני Commands להעברת פקודות מסרים, ו-Data Binding בין-View לבין-ViewModel.

**: Low Level Design**

- **OOD - ניתוח מונחה עצמים:** איתור הדרישות הפונקציונליות והלא פונקציונאליות של המערכת: סט הקלטים, ההתנהגות והפליטים הנדרשים למימוש לצד הchlוטות ואילוצים נלוויים.
- בגדוד: ניתוח המערכת בצורה של אובייקטים וארוגנה באמצעות תרשימים כמו ERD וכו'.
- **OOD - עיצוב מונחה עצמים:** עיצוב אפליקציה ועיצוב מערכת (לא מבחינת נראות) שימוש בתבניות ארכיטקטוניות ותבניות תכנון, שילוב עקרונותetc.
- בגדוד: תכנון של המחלקות בצורה של מונחה עצמים (מחלקות, ממשקים וכו') תוך שימוש בתבניות עיצוב.
- **OOD - תכנון מונחה עצמים:** התכלס, פרקטיקות מימוש המעודדת יצרת קוד מונחה עצמים "טוב": שימוש בממשקים (יצירת גרסאות קוד אבסטרקטיות), TDD וכו' ...

**: Scalability .22 (מדדויות):**

מדדיות, היא פיתוח תוכנה שמסוגלת לגודל לפי הצורך, היכולת של מערכת להתמודד עם כמות גדולה של עבודה על ידי הוספה משאבי המערכת. **דוגמה:** מערכת משלוחי חבילות היא מדדיות כיון שהיא ניתנת להרחבה, הוספה משאביים (רכבי משלוח) מאפשרת להתמודד עם כמות חבילות גדולה יותר.

**23. סימפטומים למערכת הנמצאת בבדיקה הגדרות עיצוב (шибירות, צמיגות, נוקשות, אי-MOVILITY):**

תכונות ומאפיינים למערכת הנמצאת בבדיקה הגדרות עיצוב:

- **נוקשות** – אין את היכולת לשנות בקלות, המערכת נוקשה ומגיבה רע לשינויים. יכול להתבטא גם בהשקה גדולה בשבייל שינויים קטנים.
- **шибירות** – שינוי קטן יכול לפרק את כל המערכת.
- **אי-MOVILITY** – חוסר יכולת לעשות שימוש חוזר(reuse), ככלומר המערכת לא מופרדת כמו שצරיך ולכן לא תוכל לקחת משנה ולהשתמש בו במקום אחר.
- **צמיגות** – חוסר גמישות, קשה מאד לזרז – כמות השלבים שתצטרכן לעבור בשבייל לעשות שינויים היא קטסטרופלית ומולואה ב비ורוקרטיה.

## :Dependency Management, Dependency Firewalls . 24

**ניהול תלות** (Dependency Management) הוא התהליך של ניהול כל המשימות והמשאבים הקשורים זה לזה כדי להבטיח שהפרויקט יוכל יתבצע בזמן ובתקציב. כאשר יש תלות שצריך לנהל בין פרויקטים, זה מכונה ניהול תלות הדדי בפרויקט.

הרענון בתלות בחומרות (Dependency Firewalls) הוא ליצור הפרדה בין מודלים שונים כך שלא יהיו תלויים אחד בשני, לדוגמה אם תיגרם שגיאה במודול אחד לא נרצה שזה ייפל את המודול השני. משק הוא דוגמה מצוינית ל-Dependency Firewalls כיוון שהוא יכול להיות גדר וחיציה בין מודלים ובכך למנוע מושגיהם להיכנס פנימה ולהקלר את המערכת.

## 25. חישבות ממשקים בכתיבת קוד

משק יכול להוות dependency firewall כשהוא מעין גדר וחיציה בין מודלים ובכך יכול למנוע מושגיהם לבבע פנימה ולהקלר את המערכת. בנוסף, תכנון מערכת על בסיס Interfaces היוצרים לצורך אבסטרקטית חיבורם ביניהם, מאפשרים לקבל מידע וחלוקת נכונה.

## 26. ארכיטקטורה ועיצוב תוכנה

ארQUITקטורה היא התהום העוסק בתכנון מערכות תוכנה. המונח ארכיטקטורה בהנדסת תוכנה פירושו ייצוג היבטים שונים של התוכנה באופן מופשט. ארכיטקטורה של מערכות תוכנה היא לפיך, תכנון מופשט של היבטים השונים של התוכנה, היחסים בין המרכיבים השונים של התוכנה והחוקים החלים עליהם.

## 27. תבניות ארכיטקטוניות

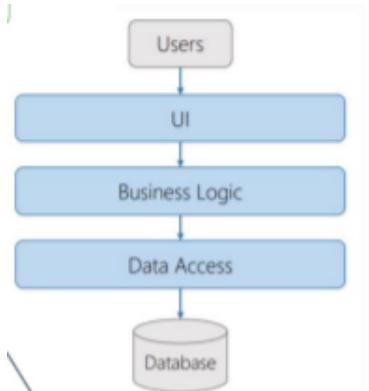
**MVC, MVP, MVVM** כפי שמפורטים בסעיף 21 ובנוסף:

## :Event Bus

דומה ל-susפט ממתקות הפעלה, סוג של אזור המשמש להעברת מידע בין תהליכיים. בתמונה ניתן לראות שישנם שני תהליכיים שמעודכנים שני ערוצים (1,2) בסעפם ושתי תהליכיים שמאזינים לאוטו מידע והכל קורא דרך ה-suspa.

## :Multi Layer

ארQUITקטורת שלוש השכבות, התבנית מורכבת מהשכבות הבאות:



## • שכבת תצוגה (UI):

זו ה层层ה העליונה של היישום. שכבת התצוגה מציג מידע הקשור לשירותים כדוגמת דפדף בין מօצרים, מידע אודiot הנקינה ותוכן עגלת הקניות. השכבה מתקשרת עם שכבות אחרות באמצעות העברת התוצאות לדפדף או לתוכנת ללקוח אחרית, וכל יתר השכבות ברשות.

## • שכבת היעשום (Business Logic):

נראית גם "שכבת הלוגיקה העסקית", "שכבת הגישה לנוטונים" או "השכבה האמצעית". שכבה זו שולחת בפונקציונליות של היישום על ידי ביצוע עיבוד הנתונים הפרטני.

## • שכבת הנתונים (Data Access):

שכבה זו מרכיבת מרשת בסיס נתונים אחד או יותר. מכאן המידע נשelfth ומואחסן. שכבה זו שומרת על הנתונים עצמאיים ובלתי תלויים בשארת היעשומים ובלוגיקה העסקית. אחסון הנתונים בשכבה נפרדת ממספר גם את הסילומיות והביצועים של היישום.

## השוואה בין Multi Layer לבין מודל ה-MVC:

מבט ראשון, מודל שלוש השכבות נראה דומה לתבנית העיצוב Model-View-Controller .

עם זאת, מבחינה טופולוגית הם שונים. ככל יסוד בארכיטקטורת שלוש השכבות הוא שלוקוח לעולם לא מתקשר ישירות עם שכבת הנתונים. במודל שלוש השכבות כל התקשרות חייבת לעבור דרך שכבה האמצעית, ולכן מבחינה רעיונית, ארכיטקטורת שלוש השכבות היא לא LINARית. לעומת זאת, ארכיטקטורת MVC היא מושלתת: המבט (view) שולח עדכונים לבקר (controller), הבקר מעודכן את המודל (model), והView מתעדכן ישירות מהמודל.

## :Ports and Adapters

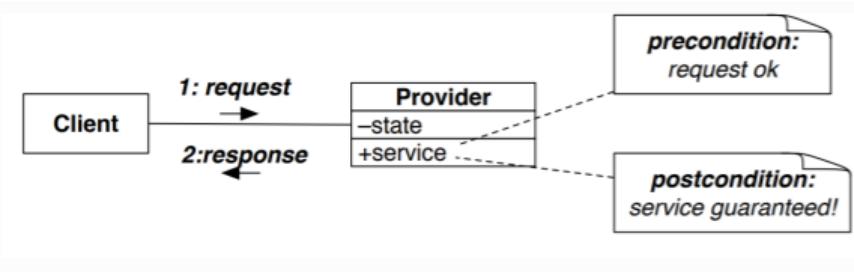
יצירת האפליקציה כרשותהיה מסוגלת לרוץ בלי UI ובלי DATA ביחס מקיים, זאת על מנת שנוכל להריץ טיסטים אוטומטיים על מערכת (לא קישור לדאטה ביחס) ממומש כאשר ה-DB אינו זמין מסיבות כללה ולמרות זאת אנו רוצים לבדוק את האפליקציה.

## 28. DBC (ציפיה לקיום תנאים מוקדמים, הבטחה למילוי תנאים ביציאה, שימור אינטראנסים)

**Design By Contract**: עיצוב ע"פ חוזה.

הגדרת ממשק עם תנאים מודיעין אשר יופיעו את חלק המערכת בצורה מדוקת. ממשק הוא למעשה סוג של חוזה, כי כל מי שמשמש אותו חייב למשש את כל הfonקציות שהוגדרו בממשק (חוזה).

- **preconditions**: תנאים שחייבים להתקיים לפני שימושים פונקציה (או תהליך).
- **Postconditions**: תנאים שחייבים לאחר התקיים לבצע פונקציה או תהליך.
- **Invariants**: תנאים שחייבים להתקיים לפני ואחרי לבצע פונקציה או תהליך.
- **Side effects**: צריך להימנע משינוי של משתנה המערכת מחוץ לתחום שבו הוא מוגדר.



תנאים מוקדים מחייבים את המשתמשים (שולחים קלט), תנאים מאוחרים מחייבים את המחלקות (שמעבדות את הקלט).

החוזה למעשה מניח שכל התנאים המוקדמים והמאוחרים מתקיימים רק כך המערכת עובדת כמו שצירף). החוזה מחייב את הלוקוח לבקש בקשوت תקפות (מודגר ע"פ החוזה) ומחייב את הספק לספק את השיווק בצורה נאותה (גם מוגדר ע"פ החוזה) אם אחד מהצדדים הפר את החוזה יהיה כשל המערכת ותיזרק שגיאה.

## Unit Testing vs TDD .29

**TDD** - עיצוב מונחה בדיקות: בדיקות יחידה הנכתבת לפני הכתיבה נכתבות קודם לכן. **Unit Testing** - כתיבת טסטים קטנים ורבים כאשר כל טסט בודק פונקציה פשוטה מאוד או התנהגות אובייקט מסוים. ההבדל המרכזי בין TDD לבין UT הוא ShTD הוא TDD והוא תהליך חשיבה שmbia לבדיקות יחידה, ו"thinking in tests" נוטה להוביל לבדיקות עדינות ומקיפות יותר ובנוסף הוא קל יותר להרחבה.

## Unit Tests and Mock Objects .30

בunit test, אובייקטים מודולים (mock objects) יכולים לדמות התנהגות של אובייקטים מורכבים ואמיתיים ולכן הם שימושיים כאשר אובייקט אמיתי אינו מעשי או שלא אפשר לשלב אותו ב-unit Test.

אם לאובייקט יש אחד מהמאפיינים הבאים, ניתן להשתמש באובייקט מודולם במקומו:

- האובייקט מספק תוכאות לא דטרמיניסטיות (למשל השעה הנוכחית או הטמפרטורה הנוכחית).
- יש לו מצבים שקשה ליצור או לשחרר (למשל שגיאת רשות).
- הוא איטי (למשל מסד נתונים שלם, שהוא צריך לאתחל לפני הבדיקה).
- הוא עדיין לא קיים או עשוי לשנות התנהגות.

דוגמא: תוכנית שעון מעורר שגורמת לפעון לצלצל בשעה מסוימת העשויה לקבל את השעה הנוכחית משירות זמן.

כדי לבדוק זאת, על הבדיקה להמתין עד למועד האזעקה כדי לדעת אם היא צלצלה נכון בפועל.

אם נעשה שימוש בשירות זמן מודולמה במקומם שירותים בזמן אמת, ניתן לתוכנת אותה לצלול הפעון (או כל זמן אחר) ללא קשר בזמן האמיתי, כך שניתן היה לבדוק את תוכנית השעון המעורר במנוטק.

## 31. מהו קידוד ויזואלי (בהקשר הקורס):

קידוד ויזואלי/חוותי מתייחס לתהליך שבו אנו, כבני אדם זוכרים תМОנות חוותיות.

דוגמא: הלוגו של מסטרקרארד. כשאתה רואה את עיגול כתום ואדם משלבלים, אתה יודע באיזו חברה מדובר. אתה לא מעביד עיגול כתום כאן, עיגול אדום וכך, הטעסט שם, וזה חשוב על חיבורו. אתה רואה את הסכום, לא את החלקים (תיאוריות הגשטאלט).

התרונות של קידוד חוותי: הפירושים של התמונות השלמות הללו הן מה שנאגר בזיכרון כמייצג של האובייקט או הגירוי שנכפה. קידוד חוותי יכול להגבר את הזיכרון ולוזר לזכור מידע עמוק יותר למשך הזמן.

בקשר לקורס נרצה להימנע במאגרי הזיכרון של המשתמש.

פרופיל משתמש (User profile) הוא אוסף הנתונים האישיים הנשמרים עבור משתמש קצר מסויים, לרוב בפלטפורמה דיגיטלית מקוונת. חשוב להפריד בין מערכות תגבות לבין מערכות פרופילים. פרופיל המשתמש נשמר על ידי כל מערכת המעוונית בכרך, שאיתה המשתמש בא ברגע, החל מישומים המעווניים לשומר את הגדירות משחק המשתמש, דרך מערכת הפעלה המעוונית לשומר את פעולותיו האחרונות של המשתמש לצורך בהפעלה נוספת, וכלה באתר אינטרנט המכיל מידע אישי הקשורו הוא מציג למשתמש בכל פעם שהוא גולש אליו. פרופיל משתמש אם כן הוא הייצוג הדיגיטלי של זהותו של אדם מסוים בעניינו תוכנה מסוימת וביעניו שאר המשתמשים באותו מימד.

### 33. מודל חמשת ה- S'ם

חמשת השכבות לחווית משתמש (UX) טוביה: Strategy, Scope, Structure & Surface

**Strategy** (אסטרטגיה) - מצא איזון בין היעדים העסקים לצורכי המשתמשים. *Booking*, למשל, שואפת למציניות בתחום החזמת הנטיות ומכירה בכרך שהצרכנים רוצים ושירותים איקוטי. כדי להשיג את האיזון הזה, *Booking* מציעה מידע וכלים המאפשרים למשתמשים לקבל החלטות ולהזמין בזמןן הנוכחי ובמחירו הנוכחי.

**Scope** (היקף/תחום) - הפיכת אסטרטגיה לדרישות, הגדרת מאפייני המוצר או השירות או כל מידע בעל ערך מוסף אחר שיוצג באתר או באפליקציה. פשנות היא המטרה הסופית. ערכו רשיימה של המידע, הכלים והאפשרויות הרצויות. עבור *Booking*, זה כולל נתונים על לינה, טיסות, השימוש של תעريف טיסה ומלוון, השכרת רכב, דירוג משתמשים וירידות מחירים.

**Structure** (מבנה) - כאן נכנס לתמונה עיצוב הפלטפורמה וממשק המשתמש (UI). מבנה של אתר או אפליקציה קובע כיצד המערכת יגיבו לפעולות המשתמש ומורכב ממשני הביטים:

1. **עיצוב אינטראקטיבי**. המטרה היא להגדיר התקדמות כדי להדריך את המשתמשים לאן אתה רוצה שהם יגיעו.

ב-*Booking*, הצעד הראשון הוא לגרום למשתמשים להזין את היעדים, התאריכים ומספר הנוסעים שלהם. משתמשים ממלאים טפס כדי לראות רשימה של אפשרויות זמינות, והם יכולים לראות פרטיים אודוט המלון לפני אישור ההזמנה שלהם.

2. **ארQUITECTורת מידע**. זה מפרט את התוכן, איך הוא יהיה בניו והיכן הוא יוצב על המספר. *Booking* בוחר להציג את המאפיינים המבדילים בין המלונות כבר בתוצאות החיפוש, וכך המשתמש לא צריך לחוץ על הקישור של המלון כדי לראות את מיקומו. לפיכך, זה עוזר לנוסעים לבחור (או לבטל) כל אפשרות מהר יותר, בהתאם על מה שהם מחפשים.

**Skeleton** (שלד) - בrama זו המבנה המדויק של האפליקציה/אתר נחתם, כולל היקן מקומותיים רכיבי הממשק על המסך וכיידם מקיימים אינטראקטיביה זה עם זה. מה שחייבנו כאן הוא עיצוב הממשק והנויוט (אלמנטים המאפשרים למשתמש לנוט בצורה אינטואטיבית, למצוא מידע או אינטראקטיביה).

הഫצה של *Booking* מבוססת על שלושה הزاد (שלפיו הנזפה ביותר באתר הוא החלק השמאלי העליון). בחלק העליון של העמוד יש כפתורים המ קישרים לשירותים העיקריים המוצעים, בעוד שחלונית משמאלי מכליה אפשרויות חיפוש ולאחר מכן מסננים. תוצאות החיפוש מופיעות במרכז העמוד.

**Surface** (משטח) - עיצוב חזותי אמר לעוזר למשתמשים להבין את המידע המופיע באתר ולהנחות אותו בשימוש בו באופן אינטואיטיבי, תוך שהוא מתאים מבחינה חברתית ותרבותית.

Booking תמיד מזכה את האפשרויות והמידע הרלוונטיים באותו מקום מיקומים כדי לאפשר ניוט קל. תחת שם המלון, למשל, הוא מציג תמיד הכתובת וקישור למפה, ומתחת הוא מציג תמונות, דירוגים ממוצעים והערות משתמשים.

### 34. צימוד ולכידות

בהנדסת תוכנה, מגד צימוד (Coupling) מייצג את רמת התלות בין מודולים שונים באותה מערכת. מונח נוסף אשר הומצא במסגרת שיטה זו הוא מגד הלכידות (Cohesion), המתיחס לחזק הקשר הפונקציונלי בין פעולות שונות תחת אותו מודול. רמת לכידות גבוהה, במרקמים רבים, היא סימן לרמת צימוד נמוכה ולהperf.

בharרכט איקוטו של קוד תוכנה, יעדף קוד בעל צמידות נמוכה, המהווה סימן לכך שהקוד תוכנן כהלהכה ובינוי היטב. על מנת להשיג צמידות נמוכה, יש לתוכנן כל מודול כיחידה עצמאית ככל הניתן, אשר תליה כמה שפחות במודולים נוספים במערכת. במצב זה, במידה שידרשו שינויים בקוד, אלו יבוצעו ביתר קלות כיון שינוי ששניינו במודול מסוים לא יאלץ שניים בכל המערכת.

שילוב של צמידות נמוכה ולכידות גבוהה הוא סימן לכך שהקוד בעל יכולת תחזקה גבוהה, מבצע על חסן, אמינות, יכולת שימוש חדש ויכולת הבנה גבוהה של הקוד.

עקרונות SOLID הם חמשה עקרונות בסיסיים בעיצוב מונחה- עצמים. המחשבה מאחוריה העקרונות היא שכאשר הם מיושמים ייחדי בפיתוח של מערכת תוכנה, סביר שהוא יהיה יותר קל להתחזק והרחבה לארוך הזמן.

- Single responsibility principle - לכל מחלקה צריכה להיות אחריות אחת ויחידה.
- Open for Extension, Closed for Modification - מחלקה צריכה להיות סגורה לשינויים ופתוחה להרחבה (למנוע שינויים ולאפשר ירשות).
- Liskov substitution principle - אובייקטים בתוכנה יכולים להיות מוחלפים על ידי מחלקות יורשות ללא שינוי תפקוד התוכנה בכללותה.
- Interface segregation principle - למחלקות יהיו ממשקים שונים אשר יתאימו לפוי צורכי המשתמשים.
- Dependency inversion principle - מימושים יהיו תלויים במושקים ובסטרטגיות ולא במימושים פנימיים.

SMART .36

מודל לקביעת יעדים:

- Specific - יעדים מוגדים ככל הניתן (ולא כלליים).
- Measurable - יעדים ניתנים למדידה.
- Attainable - יעדים ברิ השגה, ריאליים והגוניים.
- Relevant - יעדים המשרתים את הייעוד ואת האסטרטגיה הארגונית.
- Time-Bound - יעדים התחומים בזמן.