

## סיכום – הנדסת תוכנה

### הרצאה 1:

מערכת – אוסף יישויות ממשיות או מופשטות, המקיימות קשרי גומלין או תלות הדדיות ויוצרות ביחד תוצרת המשרתת מטרה (אחת או יותר) סופית.  
למרכיבי המערכת יחסים פונקציונליים המגדירים תהליכיים.  
למערכת התנהלות הכוללת קלטים (אנרגיה/חומר/ מידע) ופלטינם שהם תוצר עיבוד המשנה את הקלט בצורה כלשהיא.

תיאוריות מערכות:

- מערכת היא ישות השומרת על קיומה ע"י יחסי גומלין ופעולות הדדיות בין מרכיביה.
- Emergence (הופעה) – אפשרו בין האינטראקציות ההדדיות צורות תכונות שלא ניתן למצוא בנפרד. חקר המרכיבים בנפרד אינו מספיק להבנת המערכת.
- אינטראקציות אלו כוללות לולאות ומשובים המיצרים התנהלות לא לינארית ולא דטרמיניסטית המקשה על חיזוי התנהלותה.

מודל – יציג מופשט של תופעה, ישות או רעיון.  
הפשטה (אבסטרקציה) – השמתת פרטיים בצדדים לצור מושגים וככלים עקרוניים.

סוגי מודלים:

פיזי – כמו דגם של מטוס  
קונספטואלי (=קשרו לרעיון המרכזי) – מודל סטטיסטי  
 ועוד ..

בונים מודל כדי: לנתח, להבין, ללמידה, לבנות...  
למודלים המציגים אספקטים שונים בתהיל' היפותזה מוגדרים ארטיפיקטים סטנדרטיים (ארטיפיקטים בדומה לסמליים הם ביוטי מוחשי או פיזי גם כן לערכיים ונורמות בארגון)

תוכנה – אוסף שירותים המקובצים על גבי חומרת מחשב ומיעדים למימוש מטרות משתמש.

תכורות מוצר תוכנה:

1. מוצר עצמאי – מגע ישיר קשייח או בהורדה מהאינטרנט ומספק מערכת הפעלה ותוכנה תשתייתית (כמו תוכנת אופיס)
2. מוצר משובץ (embedded) – מגע ביחיד עם החומרה המתאימה לו ספציפית. יש בו מערכת הפעלה ותוכנה תשתיית כ מוצר שלם (כמו טלפון)
3. תוכנה כשירות (Software as a Service) – התוכנה עצמה אינה מגיעה ללקוח ואין נשארת ברשותו אלא הוא מקבל שירותים הדרושים באמצעותה (כמו Gmail, Dropbox) שבערךן התוכנה נמצאת על שרת ואנו מקבלים שם את השירותים

הנדסה:

- החלטת שיטות ועקרונות מדעיים לייצור טכנולוגיה ו מוצרים.
- לדוג' הנדסת מכונות, הנדסה אזרחית, הנדסת חשמל וכו'
- דיסציפלינות בנויות אלפי שנים (דיסציפלינה אקדמית היא תחום-ידע שבו מבוצע מחקר אקדמי)

הנדסת תוכנה – אוסף שלבים, פעילות, תפקדים ותוצרים המשרטים יצירת מוצר המורכב משירותי תוכנה במגוון "אריזות" (תכורות שונות של מוצר תוכנה כפי שהסבירנו לעיל)

תהליך הנדסי – המטרה היא להקטין את השונות – בתהילר/ב מוצר/בתכניב/בידע/במשאים  
באומנות – השונות רציה וمبرכת  
הנדסת תוכנה היא שילוב של שני הדברים הללו.

- שונות גבוהה בפרויקטים – באופי, בתכניב, בל"ז
- יחד עם זאת יש שאיפה להקטין את השונות בתהילר עצמו

בעלי עניין:  
מנתח מערכת  
ארכיטקט (מעצב את המערכת)  
תוכנinator (מממש את המערכת)  
בודק (בחינת התוצאות מול התכנון)  
מטמייע (התקנה והטמעה אצל המשתמשים)  
לקוח (МОМЧАЯ יישום)

#### האם פרויקטי תוכנה מצליחים?

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

דוגמה לכשל בתוכנה היא כsshur הדולר נקבע על 0 ולכן תקינה קרסה והלקחות אינן הצלחו לבצע עסקאות.

דוגמה נוספת היא התקלות של הפלפון Samsung Galaxy Note 7 בפראייריז במהלך העבודה

תוכנה, עם פילוח לפי שנים, לפי סוג מערכות ועוד.  
מסקנות כלליות מהחברה:

- כמעט מלחיצת מאוכלוסיות העולם השפיעה מהתקלות במהלך 2017 (3.7 מתוך 7.4)
- שיאי התקלות התרחשו בין החודשים אוגוסט לונובמבר
- אומדן ההפסדים מתקלות התוכנה הינו קרוב ל 2 טרילייארד דולר

דוגמה של כשל תוכנה: תרגום שגוי של כתוב שהוביל למעצר פלסטיני בחשד לתוכון פיגוע דרייסה

הסיבות לכשלים בפרויקט תוכנה:  
הגורם האנושי:

- תקשורת לקויה (לקוח-ספיק, הנהלה-צווות פיתוח, בעלי תפקידים בצוות)
- ניהול דוגמטי חסר מעוף, יצירתיות והבנת התחום
- תהליכי לא מתאימים לשיטות ולבאים
- אינטרסים מנוגדים של בעלי עניין (Stakeholders)
- אי שימוש בסטנדרטים ושיטות עבודה מוגדרות היטב
- אי שימוש במידדי תוכנה לקבלת החלטות על בסיס נתונים
- אי שילוב תהליכי בקרה איכות
- חוסר מקצועיות צוותים (ארכיטקטים, תוכנינגים, מנהלי מוצר)

- טבעו של עולם
- תהליים רוי אינטואוט מובנית
- מרכיבות, מרכיבות, מרכיבות (תhallיכים, כלים, פתרונות..)
- דרישות משתנות ואירועים בהגדרת הרכבים
- לחץ זמן ותקציב
- קשה לפתח תוכנה
- אין טכנולוגיה אחת או טכניקת ניהול אחת שאיתה בודדות ניתן לקבל שיפור בפרודוקטיביות [=יכולת לייצר ולהפיך] (בניגוד לחומרה שה�택חה באופן משוער)
- מרכיבות מהותית ומרקית
- מרכיבות מקרית ניתן לצמצום (קושי שנובע מהכלים שנבחרו להנדסת התוכנה)
- מרכיבות מהותית נובעת מעצם הבעה (קושי שנובע מהבעה ולא מהכלים)

#### תכונות מהותיות בתוכנה:

- סיבוכיות – Complexity
- Conformity (התאמה) - קונפורמיות, התנהגות בהתאם, תאימות [התאמת התוכנה גם לאחר שינויים, הוספה יכולות בחומרה או הוספה יכולות שאין בחומרה כמו שליחת סרטים]
- Changeability - ניתן לשינוי [כמו עדכון תוכנה]
- Invisibility - בלתי נראה [שימוש אינו חשוף]

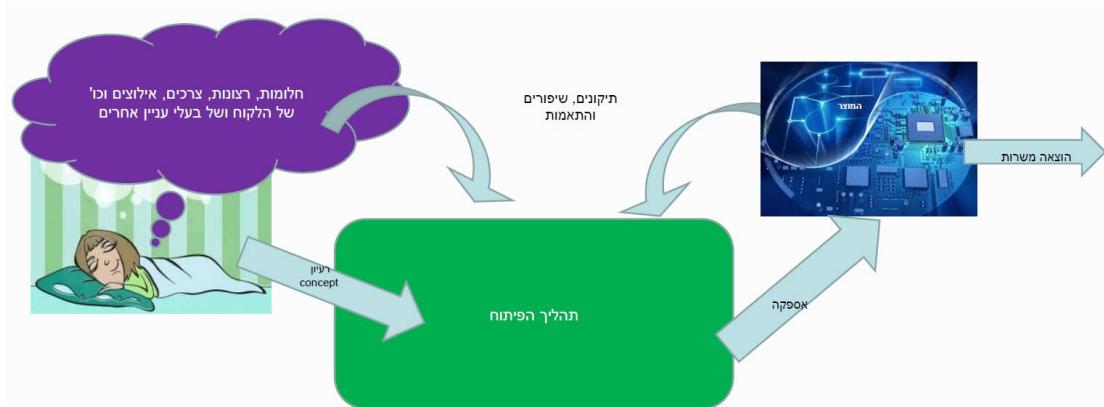
#### מצומן המרכיבות המקרית:

- שימוש בשפות עיליות – גורם לשיפור של כה פי 5 בפרודוקטיביות [=יכולת לייצר ולהפיך]
- שימוש במערכות מומחה – יכול להציג שיטות עבודה מומלצות. לדוגמה, יצירת בדיקות אוטומטיות
- תוכנות אוטונומי
- Program verification - מציאת באגים בתחום התהילה
- סביבות וכלים. לדוגמה- Eclipse , Visual studio ,
- שימוש בשיטות עבודה מתקדמות - דוגמת agile

#### מצומן המרכיבות המהותית:

- קניית תוכנת מדף
- חידוד הדרישות ובניית אפקטיבי
- למצוא מהנדסי תוכנה טובים יותר (יכול להגדיל את הביצועים פי 10)
- כתיבה<>>בנייה<הגדלת התוכנה – גישה בה ניתנת התיחסות לאידליה וה�택חות של התוכנה
- התיחסות כזו משפרת את הביצועים (נשתמש במודל עבודה שתומך בשיטה זו)

#### מחזור החיים של מוצר תוכנה:



**פעילות בפיתוח התוכנה:**

- ייזום – זיהוי בעיות והזדמנויות
- הגדרת דרישות – תיאור מדויק של הנדרש
- ניתוח – כיצד ניתן לפתור את הבעיה
- עיצוב – בחירה ותוכנן הפתרון המתאים
- שימוש – תרגום התוכניות למציאות
- בדיקות – בחינת התוצאות מול התוכן
- שילוב – התקנת המערכות והטמעה אצל משתמשים
- תחזקה – תהליכי מתמשך של ניפוי שגיאות ורחבות  
דינמיים, הערכת סיכונים תיעוד, תיעודף, ארכיטקטורה...

**עיצוב תוכנה:**

- אפיון והגדרת פתרון על בסיס דרישת ואילוצים
- מטרת ארכוט טווח: התמודדות מוצלחת עם דרישות משתנות
- הגדרת מבנה ומסגרת לקידוד
- הארכיטקט המקיים החלטות הנוגעות לעיצוב תוכנה ברמה הגבוהה
- עיצוב תוכנה ↔ מודולציה

**עקרון "הפרד ומשול" (Separation Of Concerns):**

- חלוקת התוכנה לחלקים כך שכל חלק מטפל בנושא אחר
- החלוקה נעשית באוצרה נכונה וכל חלק מהמערכת הוא מודול (מודול היא חלק מתוכנית מחשב שניית להפעלה בנפרד)
- תוכנה המחולקת למודולים היא מערכת מודולרית

**מודולים:**

- הפרדה נכוןה:
  - צימוד נכון (Coupling): מייצג את רמת התלות בין מודולים שונים באותה מערכת
  - לכידות גבואה: חזק הקשר הפונקציוני בין פעולות שונות תחת אותו מודול
  - עקרון SOC (Separation Of Concerns):  
המטרה הכלכלת של SOC היא להקים מערכת מאורגנת היטב שבה כל חלק ממלא תפקיד משמעותי אינטואיטיבי תוך מקסום יכולתו להסתגל לשינוי (תחזקה, הרחבה, שימוש חזוז)
- ארטיפיקטים (זה בעצם יציג המידע):** תוצר לוואי של פיתוח תוכנה  
דוגמאות לארטיפיקטים שונים: דיאגרמות UML, הערכת סיכונים, cases use, קוד המערכת וכו'

## הרצאה 2

### אלמנטים של תהליכי פיתוח תוכנה אגנדה

- בעלי עניין
- בעלי תפקידים
- שלב הייזום

בעלי עניין:

- מעורב בצורה אקטיבית בפרויקט (לדוגמה מנהל פרויקט ספונסר, לקוחות וכו')
- בעלי אינטרסים חיוביים\שליליים.
- מושפע או בעל יכולת השפעה על הפרויקט.

מייפוי בעלי עניין:

- המקרה הלא טוב הוא כאשר בעל העניין הוא עם רמת מחויבות נמוכה אך המערכת תוליה בו (לדוגמה תורם)

ניהול בעלי העניין:

- חתירה לעובדה משותפת ושיתופו מקסימלית
- מעקב אחרי התנהלות בעל העניין
- הדיפה – הורדת השפעת בעל עניין לא תומכים

בעלי תפקידים:

- מנהל פרויקט: בקייא בתהליכי ניהול הפרויקט ואחראי על עמידה בביצועים הנדרשים, בתכנון ובול"ז.
- מהנדס מערכת:
  - בעל חשיבות הגבוהה ביותר בהיבט הטכני בפרויקט.
  - בקייא בתהליכי פיתוח מוצר מקובלים ומתאים אותו לפרויקט עצמו.
  - מבין את הראייה המערכתית הכלולית וכייצד תתי מערכות מתמזגות.
  - יודע לתאר את המערכת מנוקודות ראייה שונות.
  - מסוגל לאתר בעיות ותקלות ולהפעיל שיקולים מגוונים כמו ניהול, ארגוניים, כלכליים וכו'

\*\* ההבדל בין מנהל פרויקט למהנדס מערכת הוא שמנהל פרויקט שם דגש על הספקת המוצר בזמן ובתקציב ומהנדס מערכת אחראי על הובלת המאמץ הטכני לפיתוח המערכת ושניהם חייבים לשתף פעולה.

- מנהל מוצר:
  - "קולו" של המשתמש בצוות הפיתוח.
  - תפקיד בו נגשים עולם העסקים, טכנולוגיה וחווית משתמש.
  - הבנת לקוחות ואופי הבעייה של המוצר.
  - הגדרת תוכנית פעולה ומפת דרכים לפיתוח איטרטיבי לקראות מימוש מלא של החזון.
  - עבודה בצדן לצוות הפיתוח (פיקוח על התכולה, הקצב ופתרון הבעיה)
  - עם יציאת גרסה – מעקב אחריו שימושו וקבלת משוב מליקות.

תכוונות: העצמה, הצבת מטרות משותפות, כוונה לנtinyת ערך לקוחות, תקשורת טובה, פיתוחות ושיתופו, שיפור מתמיד והקפדה על איכות, סדרי עדיפויות, יצירתיות וחדשנות.

- לפעמים נושא תפקיד זה הוא גם מנהל פרויקט או מהנדס תוכנה.
- יכולה להיות התנשיות בין מהנדס מערכת כאשר הוא חשוב על גרסה הבאה בזמן שמהנדס מערכת ינסה לספק הוראות טכניות מפורטות לצוות הפיתוח. (יש להימנע מזה).
- בניהול מוצר יש להימנע מקידום פתרונות מקומיים. (אלא פתרון כללי) בתום למידת הבעיה, הגדרה, תיאור ותיעוד, הבעיה מצטרף לצוות הפיתוח לצורך הבחרות ויצוב משותף לפתרונות.
- בעת תיעוד וקבלה החלטות יש לזכור את סוג הטיפוסים.

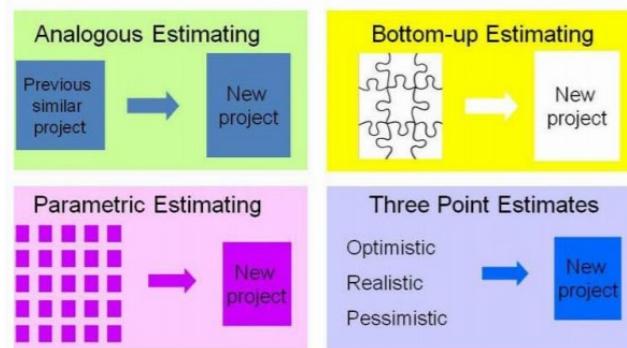
- **מנתח מערכת:**
  - אחראי על התוצרים הבאים:
  - מסמך ייזום, ניתוח מצב קיימ. הגדרת דרישות, איפון פתרון, דיאגרמות ופורמטים סטנדרטיים.
  
- **ארQUITקט תוכנה:**
  - איש פיתוח העוסק בעיצוב High Level ומודולציה פתרונות.
  - תחומי אחריות:
    - יצירה או בחירה של תשתיות תוכנה. (תשתיות כמו אנטן וירוס, מערכות הפעלה וכו')
    - חלוקה של יישום מורכב למודלים.
    - הגדרת כללי, עיצוב ופיתוח והעברתם למפתחים.
    - הבנת הפונקציונליות בכל חלק המערכת.
    - הבנת האינטראקציות והתלות בין רכיבים
  
- **ראש צוות:**
  - מנהל מס' אנשים בעלי תפקיד זהה.
  - אחראי על בניית הצוות ותיאום העבודות ביניהם
  - בקשר עם מנהלי הפרויקט.
  - ייצור Low Level Design
  - קביעת נהלים (code review, מסירה לבדיקות ועוד)
  
- **מתכנת**
- **בודק**
- **מטמייע:**
  - אחראי על כתיבת מדריך טכני למשתמש
  - התקנת המערכת אצל לקוחות
  - בקרה על פעילות תקינה של המערכות
  - מעניק שירות תמיכה ולויו שוטפים לעובדים שנתקעים בתפעול.

**שלב הייזום:** שלב ראשון בו מועלית דרישת כלילית למערכת חדשה, כאשר נרצה ליצור מערכת חדשה, או לשנות משהו במערכת קיימת.  
פועלות בשלב הייזום:

- הגדרת ליקחות
- בתוך הארגון
- מחוץ לארגון (הליקוח הפטונציאלי)

- מציאת מנהל מוצר
- ניתוח מצב קיימט, בדיקות מערכות דומות קיימות
- הגדרת מטרות (כלליות) ויעדים (ספציפיים)
- מטרה הופכת לרשימת יעדים.
- מודל SMART: (מודל הגדרת היעדים)
  - Specific (ספציפי) – יעדים מוקדים ככל הניתן, ולא כללים מדי.
  - Measurable (מדידה) – יעדים ניתנים לממדידה (?)
  - Attainable – יעדים ברิ השגה, ריאליים והגיוניים.
  - Relevant (רלוונטי) – יעדים המשרתים את המטרה והאסטרטגייה הארגונית.
  - Time-Bound – יעדים התוחמים בזמן.
- הגדרת אילוצים וויכונים
- הגדרת תועלות וחסכנות:
  - תועלות מוחשיות לא מוחשית.
  - חיסכון בכוח אדם/יעול תהילכים.
  - בדיקת ישימות וועלות/תועלות.
- בדיקת יישומיות: האם ניתן ליישם את המערכת (כגון: הגדרת יישום, מימוש המערכת, אבטחת מידע, הטמעת המערכת).
- בדיקת עלות/תועלות:
  - למה לבצע?
  - אפשרי (ניתן לעשות זאת)
  - עוזר בהשגת תקציב מראהי הארגון
  - כלי לבחינת האלטרנטיבות.
  - ניסיון הפיכת השערות והנחהות לעובדות ומדדיים

#### שיטות לשערור עלויות:



- **תבנית - Analogous Estimating:** שיעור על ידי פרויקט דומה
- **תבנית - Bottom-up-Estimating:** לרדת לפרטים הקטנים
- **תבנית - Parametric Estimate:** חלוקה בלי ירידת לפרטים הקטנים ביותר

- **תבנית Three Point Estimates**: עלות אופטימית, עלות פסימית ועלות ממוצעת (ממוצע משוקל בין שלושת הפרמטרים), ניתן למשוך על שאר התבניות

ערך הזמן של הכספי:  
:PV table

## Present Value Table

Periods	6%	8%	10%	12%	14%
1	0.943	0.926	0.909	0.893	0.877
2	0.890	0.857	0.826	0.797	0.769
3	0.840	0.794	0.751	0.712	0.675
4	0.792	0.735	0.683	0.636	0.592
5	0.747	0.681	0.621	0.567	0.519
6	0.705	0.630	0.564	0.507	0.456
7	0.665	0.583	0.513	0.452	0.400
8	0.627	0.540	0.467	0.404	0.351
9	0.592	0.500	0.424	0.361	0.308
10	0.558	0.463	0.386	0.322	0.270

## שיטת לבודיקת כדיאות כלכלית:

- **NPV (ערך הנוכחי נקי):**
  - הערך הנוכחי הנקי מורכב מהשווי הנוכחי של סך כל התקבולים (הכנסות) העתידיות של השקעה מסוימת פחות ההוצאות הצפויות. פועלה זו, אשר ממחשבת את הערך הנוכחי של סכום שיתקבל בעתיד נקראת היון והוא לוקחת בחשבון משתנים כגון ריבית אינפלצייה.
  - מהוונים כל הכנסה צפואה מהמערכת לערך הנוכחי (בו אנחנו משלמים על המערכת) וכן נוכל לבדוק האם המערכת כדאית או לא.

מס' שנים לחישוב  $N$   
 הכנסות\הוצאות באותה שנה  
 שיעור ההיון  $i$   
 מת' תגיאר ההכנסה  $t$

$$NPV(i, N) = \sum_{t=0}^{t=N} \frac{R_t}{(1+i)^t}$$

סה"כ	5	4	3	2	1	0		סה"כ
	1.6105	1.4641	1.331	1.21	1.1	100%		ערך היון
532,628	250,000	200,000	150,000	100,000	50,000	0	ערוך	
155,231	155,231	136,602	112,697	82,644	45,454	0	PV	הכנסות
468,220	468,220	457,042	444,748	431,230	416,362	400,000	PV	הוצאות
	18,000	18,000	18,000	18,000	18,000	400,000	ערוך	
	11,178	11,178	12,294	13,518	14,868	16,362	PV	
	468,220	468,220	457,042	444,748	431,230	416,362	PV	

- המרכיב שלנו צפואה במשך חמישה שנים להגדיל את מס' הלקוחות ב- 50 כל שנה.
- כל לקוחות מכיסים לקופת חולים 1000 ש"ח לשנה.
- ערך היון הוא 10%.
- מחיר עלות התוכנה הוא 400,000 בתחילת ואחר כך עליה 18,000 לשנה לתחזוקה ושיפורים
- אחרי 5 שנים PV של יהיה  $532,628 - 468,220 = 64,408$

\*\* חישוב ערך היון הסופי:  $(1 + i)^t$  כאשר  $i$  הוא ערך היון ו-  $t$  היא השנה עבורה מחשבים את הערך.

\*\*\* חישוב PV של השנה היא הכנסה/הוצאות של אותה שנה חלק ערך היון הסופי. בסופו של דבר נרצה שהPV יהיה גדול מ-0 בכך שהוא לנו רווחים.

- **ROI (return on investment):** מונח המציין את היחס בין הכספי והמשאים שהושקעו בעניות עסקתי לבין הכנסות שנבעו מהם (במקרה שלנו, פיתוח מערכת חדשה)

$$\text{החזר השקעה באחוזים} = \frac{\text{benefits} - \text{costs}}{\text{costs}} \cdot 100$$

דוגמה:

- $NPV = 64,408$
- בניית המערכת עליה לנו 400,000.
- PV חיובי.

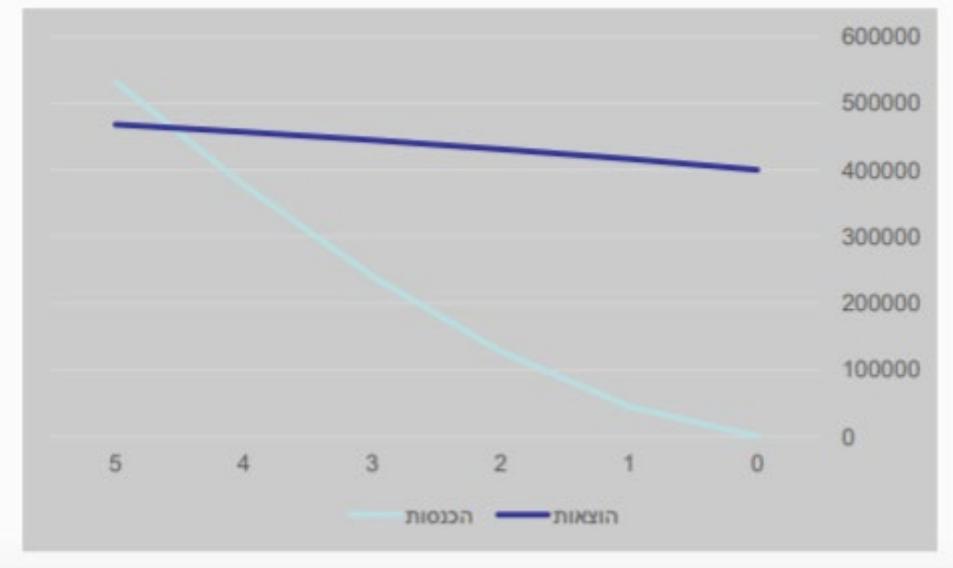
עכשו נחשב את ROI:

$$100 * \frac{\text{benefits} - \text{costs}}{\text{costs}} = \% \text{ הזר ההשקעה}$$

$$100 * \frac{532,550 - 468,220}{468,220} = 14\%$$

## • Payback analysis

- קביעת מועד הזמן שייקח למערכת להחזיר את השקעותיה או מתי המערכת מתחליה להיות רווחית (גרף המתאר הוצאות והכנסות)



## פעולות בשלב הייזום - המשך

- גיבוש ועדת הייגו: (?)
- נציגים מתחומי ידע שונים (מיחשוב, תשתיות, כספים, משתמשים וכו')
- ממשיכה לפקד עד אחרי העלאת המערכת.
- תפקידיו ועדת ההיגו:
  - קביעת מדיניות פיתוח
  - מעקב ובקרה אחרי הלוי"ז והעלויות של המערכת
  - פתרון בעיות במהלך הפיתוח
  - שינויים בשלב הפיתוח
  - קבלת החלטות בשלבים השונים של פיתוח המערכת
  - יצירת מעורבות ומחויבות אצל הלוקחות ואנשי הפיתוח
- קביעת לו"ז ומשאים:
  - לו"ז: סיום אפיון, סיום פיתוח, סיום בדיקות ומועד עליה לאויר.
  - משאים: משאבי חומרה ותוכנה + משאים אנושיים (לאפיון, לפיתוח, לבדיקות, להרצה ותחזוקה)

**הרצאה 3:**

אלמנטים של תהליכי פיתוח תוכנה

- הנדסת דרישות
- תהליכי עבודה

**הגדרת דרישות:** יצירת תשתיית למפרט התוכנה ע"פ צורכי הלוקו (זהו הבסיס להבנה משותפת בין הלוקו למפתח)

- הגדרת צורכי הלוקו
- הגדרת יכולות המוצר
- התנאים בהם המוצר נדרש לעמוד

חשיבות הגדרת דרישות:

- מקנה סיכון גבוה יותר שההטוצרים יענו על הדרישות ומקדים את תהליך האפיון והפיתוח של המערכת.
- משפר את יכולות לבצע שינויים במהירות תוך בקרה ובחינת השפעתם על דרישות אחרות.
- תיאום ציפיות בין ספק לлокו על בסיס מנותח, מוסכם ומואשר.
- שיפור יכולת לבצע בקרה על התקומות הפרויקט
- שיפור יכולת לביצוע אמידת עלויות ע"י ניתוח של עלות תועלת כל דרישة (או כולם)

\* כל בעל עניין רואה את המערכת בדרך שונה ולכן מסמך הדרישות באופן ברור

**הגדרת "דרישה":**

- תנאי או יכולת הדורשים ע"י בעלי העניין בשביל לפתרו בעיה או להשיג מטרה
- תנאי או יכולת שיש למלא (או למצוא פתרון) כדי לספק התcheinות, תקן, מפרט או מסמכים רשמיים אחרים
- דרישות אמרות לתאר "מה" המערכת אמורה לעשות ולא "איך"

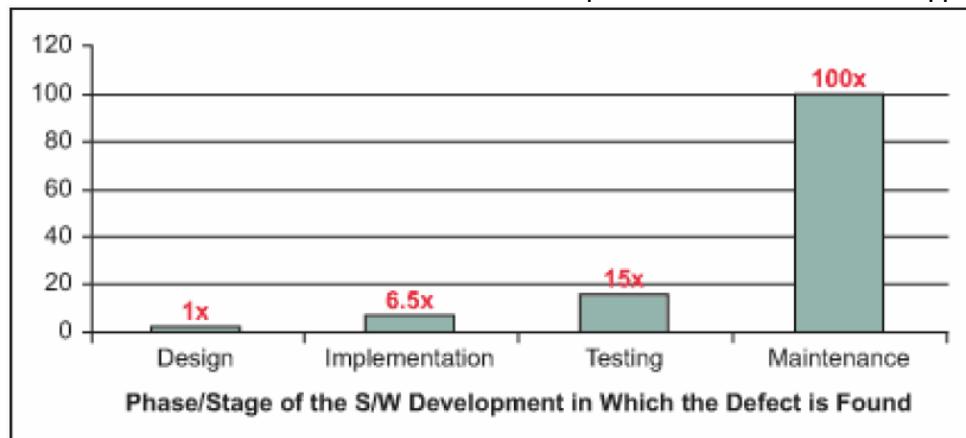
שיטות למציאת דרישות:

- ראיונות ושאלונים למשתמשים השונים
- סעור מוחות
- יצירת demo
- למידה ממוקד קיימת / למידה ממצב קיימ

בעיות אופייניות לשלב הגדרת הדרישות:

- המשמשים חושבים שהם יודעים מה הם רוצים עד שראים את התוצאה בעניין
- המנתחים מניחים מה המשמשים רוצים
- המפתחים חושבים שהם יודעים מה המשמשים רוצים
- אין אמון בין בעלי העניין (אי אמון אחד בשני לגבי הצלחת מימוש הדרישה)
- המשמשים לא יודעים להסביר מה הם רוצים ISO9001

**עלות תיקון שגיאות בשלבים השונים – עיקומת בוהם:**



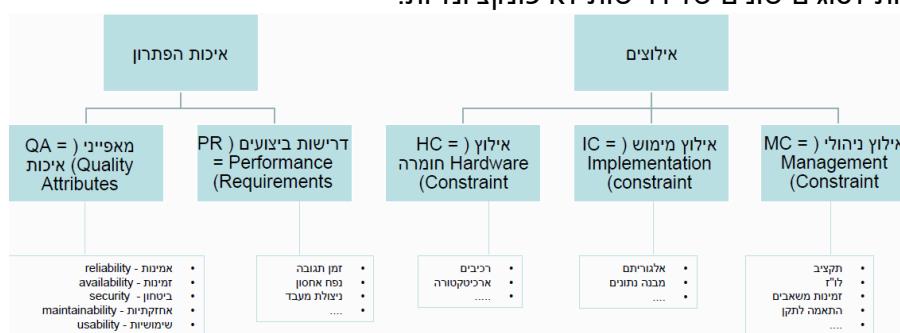
"מצעור פגמי קוד כדי לשפר את איכות התוכנה ולהזיל את עלויות הפיתוח." (בריסק')

Design and architecture	Implementation	Integration testing	Customer beta test	Postproduct release
1X*	5X	10X	15X	30X

עדוף דרישות – יש להשים לב להגדיר מה שצריך ולא מה שלא צריך. בנוסף, על דרישת ממשמעה עליונות כספיות, סיבוכיות המערכת וניהול של הבאים ולא נרצה לעשות זאת עבור דרישות לא שימושיות או איזוטריות.

סוגי דרישות:

- דרישות פונקציונליות (עסקיות): מה המערכת אמורה לעשות (פונקציות, שירותים)
    - ישנים שני סוגי של דרישות פונקציונליות:
      - דרישת פעולה – דרישת המתיחסת לתפעול או להתנהגות של המוצר
      - דרישת מידע – דרישת המתיחסת למסדי הנתונים / יישיות המידע בהן
  - התוכנה משתמשת
    - דרישות לא פונקציונליות (טכניות/איכות הפתרון): דרישות המגדירות תכונות נוספות של הפתרון שצריכות להתמלא תוך כדי מילוי הדרישות הפונקציונליות או דרישות ותנאים המגבילים את חופש בחירת ייוני הפתרון
      - (לדוגמה' זמן תגובה/ביצוע, נפח פעילות, אמינות, אבטחת מידע, אופני שימוש, תדרות ביצוע, עמידה בעומסים, שימושיות וכו')
  - דוגמאות לשני סוגי של דרישות לא פונקציונליות:



### מהי דרישת אינטואיטיבית?

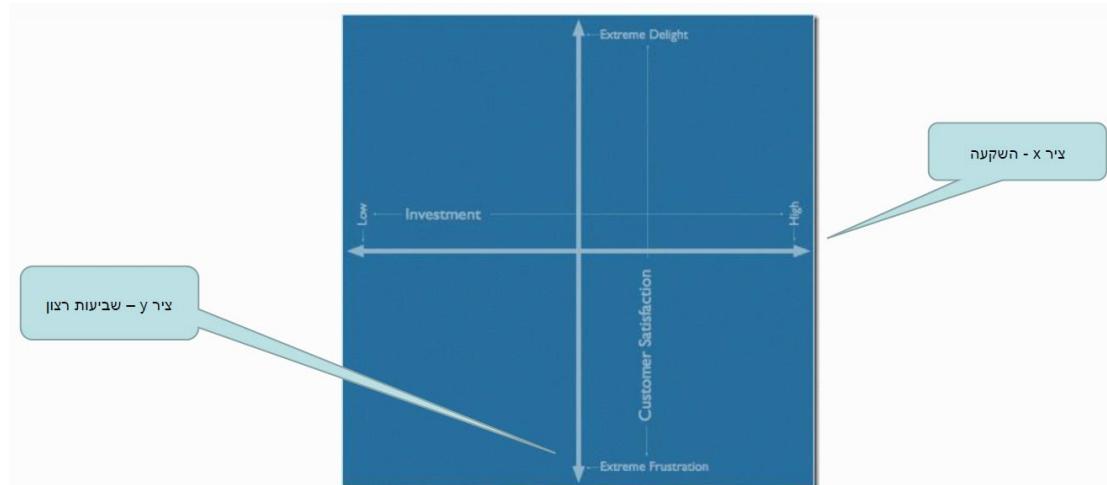
- - בדידה, מזויה חד ערכית, שייכות ברורה
- - מובנת (ברורה, מדוייקת) - מנוסחת בשפת הלוקו
- - לא עמויה (חד משמעית)
- - שלמה – Complete
- – הכרחית - בעלת תרומה משמעותית לשיפור תהליכי העבודה
- – עקבית (לא סותרת דרישות אחרות)
- - ניתנת לבדיקה באמצעות מבחני קבלה – Verifiable
- – עיקבה [ניתנת לbackward] (גם לדרישות ברמה גבוהה יותר וגם בהמשך האפיון)
- – מתועדת Prioritized

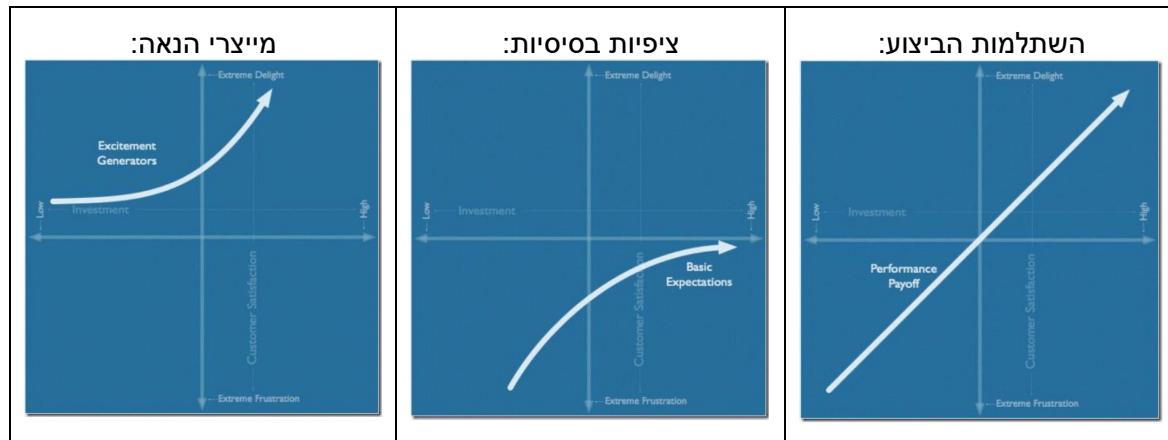
**דוגמאות:**

- חיפוש מתקדם שבו ניתן יהה לסנן גם לפי טקסט חופשי (לא אינטואיטיבית)
- זמן חיפוש סביר (לא אינטואיטיבית)
- המערכת תפחית את כמות הניריות בעלות שנתית של כ 100,23 ₪ (כן אינטואיטיבית)
- למנRAL האתר תהיה האופציה להוציא דוחות לפי חטכים שונים (לא אינטואיטיבית)
- במקרה שהמעלית נתקעה במהלך נסעה מזעיק הנוסף חילוץ באמצעות כפטור החילוץ (כן אינטואיטיבית)
- המערכת תאפשר חלוקת עבודה מאוזנת והוגנת בין העובדים (לא אינטואיטיבית)
- סטטיטיות (לא אינטואיטיבית)
- צמצום הוצאות החברה ב 27 מיליון ש"ח עד מחצית הראשונה לשנת 2018 (כן אינטואיטיבית)
- הציג היסטוריה של תלמיד בכינסה לאתר (לא אינטואיטיבית)
- ידידותי למשתמש (לא אינטואיטיבית)

### Kano Model: מודל קנו

- קנו קרא לתיגר על התפיסה שרמת שביעות הרצון של לקוחות מבוססת על "כל המרבה הרי זה משובח
- בעיני הלוקוחות יש הבדלים מהותיים בין התכונות השונות של מוצר שתורמות לשביעות רצון
- המודל מגדיר את הקשר בין רמת ההשקעה בדרישה לעומת רמת שביעות רצון הלוקוחות
- המודל משמש לתיעוד דרישות במערכת





עד כה דיברנו על הנדסת דרישות.  
כעת, נדבר על תהליכי עבודה.

שיטות עבודה:

#### פעליות בפיתוח התוכנה: (תזכורת מהרצאה 1)

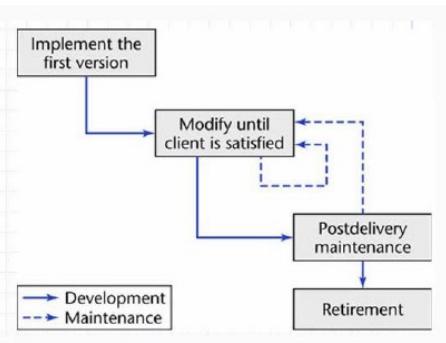
- ייזום – זיהוי בעיות והזדמנויות
- הגדרת דרישות – תיאור מדויק של הנדרש
- ניתוח – כיצד ניתן לפתור את הבעיה
- עיצוב – בחירה ותכנון הפתרון המתאים
- מימוש – תרגום התוכניות למציאות
- בדיקות – בחינת התוצאות מול התוכן
- שילוב – התקנת המערכת והטמעה אצל המשתמשים
- תחזוקה – תהליכי מתמשך של ניפוי שגיאות ורחבות

דינמים, הערכת סיכונים תיעוד, תיעודף, ארכיטקטורה...

Software Development Life Cycle  
נראה מספר מודלים של בניית מוצר תוכנה  
כל מודל מתאר גישה ותהליך = אוסף שלבים האמורים להתבצע במסגרת הפיתוח  
למודלים השונים מטרה זהה: התמודדות יعلاה עם א-זדאות ועמידה ביעדי עלות ותועלות

שלבי פיתוח התוכנה שונים משלבי הפרויקט עצמו  
SDLC != PCL (Project Life Cycle)

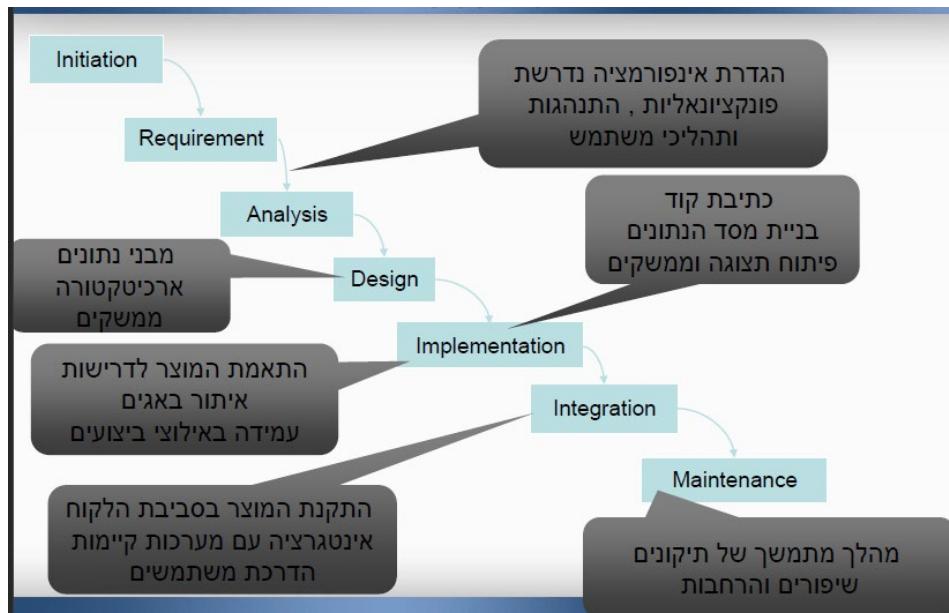
:Code and Fix



תכונות בשיטה זו:

- צוות תוכנה קטן או מתכוון בודד
  - ניתוח ותוכנן מינימלי
  - אין הבחנה ברורה בין השלבים
  - פתרון קטן העובד בנסיבות ובמינימום עליות
  - ללא תיעוד – התעלמות שיטתית שלבי התחזוקה
- בעיות
  - סינדרום ה-"משהו זמני"
  - אם מדובר במערכת קרייטית – מתכוון לאסון
  - עליות שניים ותחזקה מזנוקות עם הזמן
  - אנטיזה מוחלטת לרוח הקורס

### שיטת Waterfall:



#### יתרונות:

- מספק תהליך מובנה גם לחסרי ניסיון
- קל להבנה, פשוט לשימוש
- מספק יציבות לדרישות
- מקל על ניהול ושליטה בתהליך
- מכון לקראת איקות (ופחות אילוץ עלות או לוחות זמינים)

#### חסרונות:

- יש להכיר את כל הדרישות וramaesh
- תוצרת כל שלב אינם ניתנים לשינוי – הקפה גרסאות
- העדפת גישת התהילה המבונה על פני "פתרון בעיות"
- שלב העברה לייצור קורה בשלב אחד גדול בסוף ולא במדרג
- מעט ההזדמנויות או תחנות לקבלת פידבק מה לקוחות

#### מתאים כאשר:

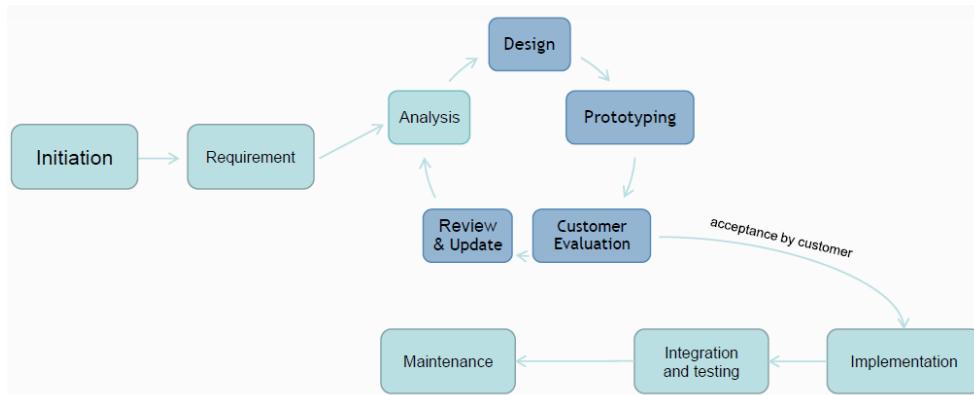
- הדרישות מאד ברורות וידועות
- הדרישות המוצר מאד יציבות
- הטכנולוגיה מובנת וידועה

- כאשר בונים גרסה חדשה ל מוצר קיימן
- כאשר מייבאים גרסה קיימת לפלטפורמה חדשה

**מודלים לנאריים נוספים**  
כניסו מונה לבניות של מודל מפל המים פותחו כמה שיטות עבודה נוספות

- אב טיפוס
- V model

#### מודל אב טיפוס מהיר:



#### שלבים מאפיינים:

- המפתחים בונים אב טיפוס בשלב ניתוח הדרישות
- אב הטיפוס עובר ביקורת של משתמשי קצה
- משתמשי קצה מספקים משוב לתיקונים
- המפתחים משפרים את האב הטיפוס
- כאשר משתמשי הקצה מרוצים, ממלאים את שאר הפרמטרים כדי להפוך אותו למוצר סופי

אב טיפוס כשמו כן הוא – דגם של המוצר בעל יכולות מוגבלות, אמינות נמוכה וביצועים לא ייעילים המשמש להדגמה בלבד. הוא לא המוצר הסופי!  
שימושי במיוחד כאשר דרישות המשתמש לא מלאות או שהנושאים הטכניים לא לגמרי ברורים

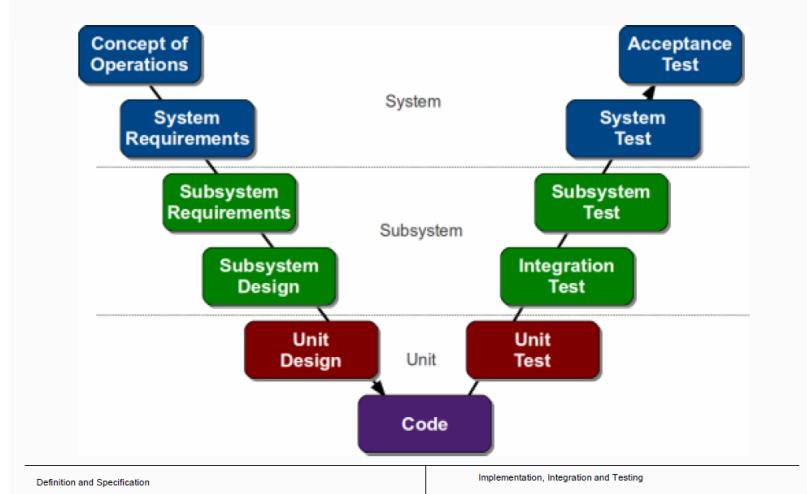
#### יתרונות

- הלקוחות יכולים לראות תרגום הדרישות למשהו מוחשי עוד בשלב האיסוף
- המפתחים לומדים ממשוב המשתמשים
- איתור דרישות וponeקציונאליות שלא היו צפויות
- אפשר עיצוב גמיש
- קל לראות התקדמות

#### חסרונות

- קיימת נטייה לאובדן מבנה התהילה (מה שייזמננו בהתחלה)
- ארכיטקטורה וראייה מערכתיות אינם זוכים למקום מרכזי
- מתמקד ברכז מידי ודרישות תחזקה נדחקות לשווים
- קיימת סכנה לתהילה שלא נגמר (Scope creep)

:V model



שיטת זו היא וריאציה על מודל מפל המים  
השלבים הראשונים של הפרויקט הם ניתוח ברמה כללית  
ובשלבים הבאים הניתוח נעשה מפורט עד רמת היחידה  
בנוסף, לכל שלב מוגדרת אספota בבדיקות מקבילה (בחילוק השני של המודל נמצא שלבי הבדיקות  
בסדר הפוך – מתחילה בבדיקות ברמת היחידה, דרך בבדיקות אינטגרציה ותתי הממערכות ועד  
בדיקות המערכת ובבדיקות קבלה סופיות)

[בדיקות אינטגרציה, הנקראות גם בדיקת שילוב, היא אחת מסדרה של בדיקות המבוצעות בהתקלך  
מחקר ופיתוח, במטרה לבדוק את ההשפעה שיש לחלק מוגדר במערכת עם חלקים אחרים במערכת  
ועם מערכות מקבילות ומשיקות].

#### יתרונות

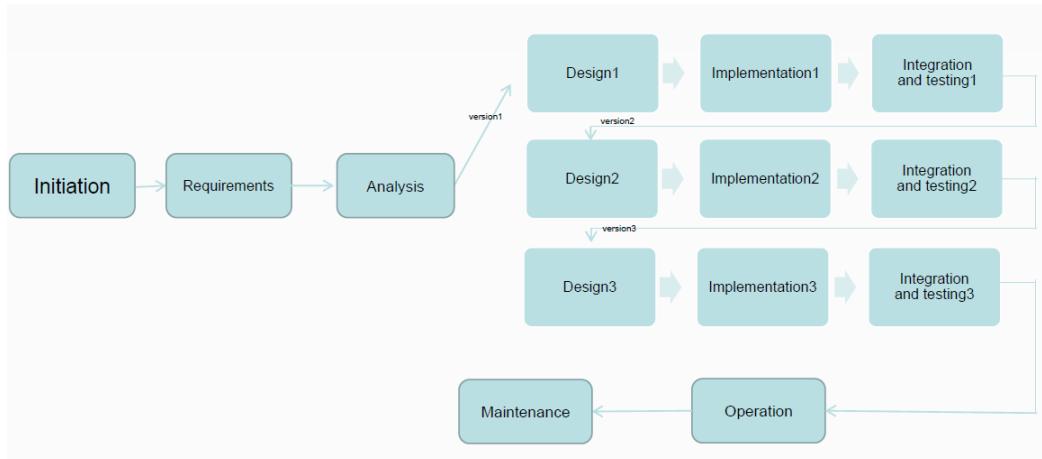
- דגש על אימות ותיקוף המוצר בשלבים מוקדמים של התהליך
- כל תוצריו הביניים חייבים להיות בני בדיקה
- מנהלי הפרויקט יכולים לבחון התקדמות על בסיס אבני דרך
- קל להטמעה ושימוש

#### חסרונות

- מקשה על טיפול באירועים בו זמנים (אי אפשר לבצע שלבים שונים במקביל)
- מקשה על התמודדות עם שינויים בדרישות

שיטת זו מתאימה כאשר המערכת דורשת אמינות גבוהה ביוטר כדוגמת מערכת רפואי.  
בנוסף, שיטה זו מתאימה כאשר הדרישות ידועות ומוגדרות מראש או שהטכנולוגיה מובנת וידועה.

**מודלים לינאריים (קוויים) לא גמישים לשינויים ולן עבור מודלים איטרטיביים**

**:Iterative model**

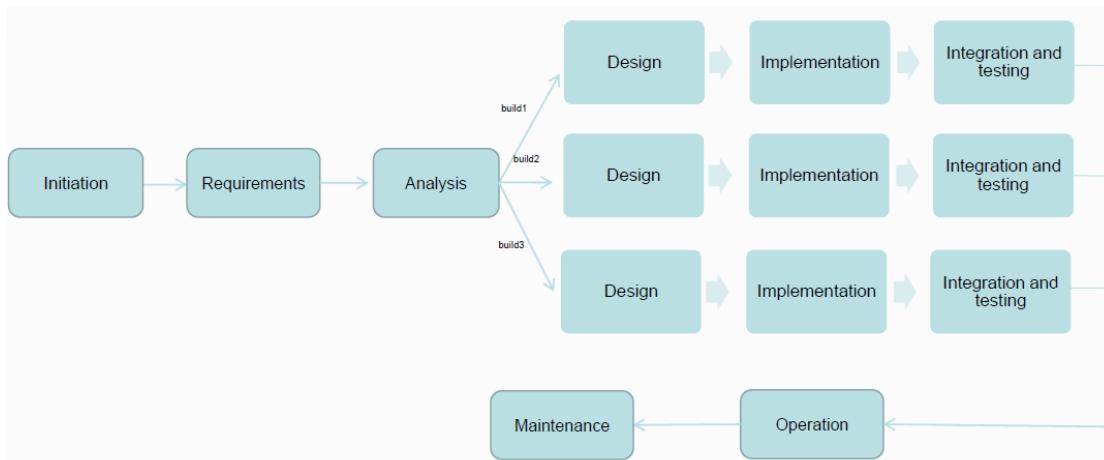
הרענון הוא למש בצורה כללית יותר ולאט לאט בכל איטרציה למקד את התוכנה

 **יתרונות:**

- ניתן לראות תוצאות מוקדם של התהילה
- שינויים בדרישות בעליים פחות
- יותר קל לבדיקה

 **חסרונות:**

- דרוש משאבים גבוהים
- אמנים עלויות השינויים נמנעים יותר, אבל עדין לא ממש מתאים לשינויים בדרישות
- קשה לניהול – דרוש ניהול צמוד וקפדי
- אין תמונה מלאה של המערכת הנדרשת לפני תחילת התהילה

**:Incremental model**

הרענון הוא למש חלקים נפרדים של התוכנה ואז לחבר את כל החלקים

 **יתרונות:**

- יצרת תוכנה עובדת מהר ובלב מוקדם של תהליך הפיתוח
- מודל גמיש יותר לשינויים, פחות יקר לשנות את התכוולה והדרישות
- קל יותר לבדוק איטרציות קטנות
- הליקוי יכול להגיב לכל מודול לחודש
- קל יותר לנוהל סיכונים, כי חלקים מסווגנים מוגדרים ומוטופלים באיטרציה שלהם

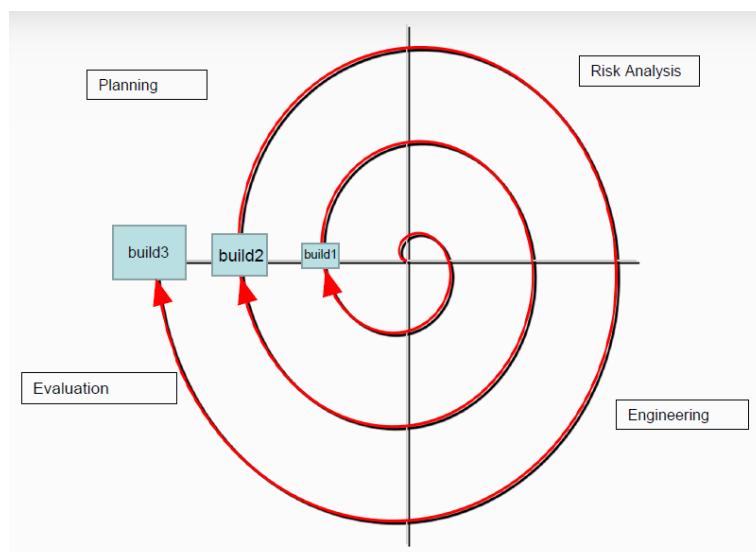
 **חסרונות:**

- דרישת תיכנון ועיצוב קפדיים
- יש צורך בהבahir ולהגדיר את כל המערכת לפני שאפשר לחלק אותה למודולים
- העלות הכוללת נראהתה גבוהה יותר ממודול מפל המים

**שיטת Spiral Model:**  
דומה למודל האינקרמנטלי אבל עם דגש לניתוח סיכונים  
**שלבי הפיתוח:**

Planning	-
Risk Analysis	-
Engineering	-
Evaluation	-

בתהליכי פיתוח תוכנה עוברים באיטרציות על 4 שלבים, כאשר על איטרציה מבוססת על איטרציה הקודמת.



**ניתוח סיכונים:**

ID	Risk	Probability	Loss	Risk Exposure	Risk Management Approach
1	Acquisition rates too high	5	9	45	Develop prototype to demonstrate feasibility
2	File format might not be efficient	5	3	15	Develop benchmarks to show speed of data manipulation
3	Uncertain user interface	2	5	10	Involve customer; develop prototype

Loss – זה רמת הנזק (כאשר 10 הוא המקסימלי)  
 Risk Exposure – רמת הסיכון (בין 0 ל-100) [זהה הכפלה של Loss עם Probability]

#### תرونנות:

- גישות "בעיה ופתרונה" – העדפת התמודדות מול הדרישות על פני מבניות
- פיתוח פונקציות מרכזיות או פונקציות בעיות בראשונה
- אבחן סוגיות סיכון בשלבים ראשוניים במינימום עלות
- מסירה מהירה של גרסאות – כל גרסה כוללת מוצר עבד
- הליקוי יכול להגיב ולספק משוב תוך כדי הפיתוח
- הקטנת עלויות פיתוח ראשונית
- התמודדות טוביה עם סיכון בשינוי דרישות תוך כדי הפיתוח

#### חסרונות:

- דריש יכולת ניתוח סיכונים
- העיצוב לא חייב להיות שלם מלכתחילה
- מודל מורכב, תקורות וסוגיות זמן משאבים
- נדרשת הגדרת ממשקים טוביה כמפתח לחיבור המרכיבים לארסה הסופית
- עלות כוללת סופית לא פוחתת בהכרח

#### מתאים כאשר:

- הגדרת סיכונים, מימון, לו"ג, מרכיבות או הערכת תועלות נדרשת כבר בתחילת הדרך
- הדרישות ידועות ומוגדרות אך צפויות להפתח ולהשתנות בהמשך
- בניית מוצר חדש
- פרויקטים בעלי סיכון גבוה
- הלוקחות אינטנסיביות מהם הדרישות
- הדרישות מורכבות
- יש מקום לפיתוח אבטיפוס

#### סיכום:

למדן להגדיר דרישות  
 ראיינו שיטות שונות בתהליכי הפיתוח:

- מפל המים – מבניות קשייה, איכות מקסימלית
- אבטיפוס מתפתח – הבהיר דרישות פרוגרטיבית
- מודל V – תיקוף ועמידה בתקנים נוקשים
- מודל ספירלי – ניתוח ובקרת סיכונים
- מודל אינקרמנטלי – הגעה ראשונית מהירה לשוקים

בהרצאה הבאה נלמד שיטות מתקדמות בפיתוח תוכנה.

## הרצאה 4

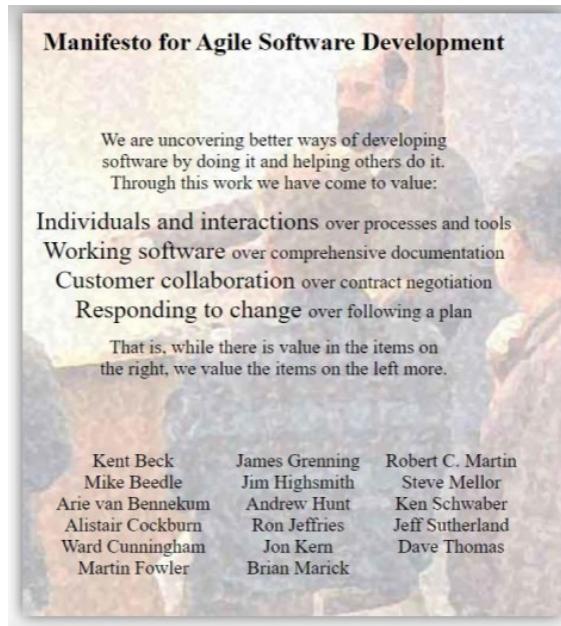
פיתוח אג'יל  
מתודולוגיות Agile:

- אוסף מתודולוגיות פיתוח תוכנה **זריזה וarterטיבית**
- נוצרה בגל היכלון עבור השיטות שקידשו תכנונים במקום גמישות.
- כל עזר שונים לניהול התהילה
- גישות שונות הניננסות לשילוב

הגדרה אג'יל:

גישה **איטרטיבית ואינקרמנטלית** (=מצטברת) הנעה בצורה **משותפת** בצד **להפק מוצר איכות**, עם דגש על **חיסכון בכיסף ובזמן** כאשר זה עונה על **שינויים בצריכים של בעלי העניין**.

המניפסט האג'יל



12 עקרונות אג'ילוֹת:

1. העדיפות הגבוהה ביותר שלנו היא לספק את הלקוח באמצעות אספקה מוקדמת ומתחמשת של תוכנות יקרות ערך.
2. פתוח לשינויים בדרישות אפילו בשלבים מאוחרים. תהליכי אג'ילם רותמים את השני לטובות היתרון התחרותי של הלקוח.
3. אספקת תוכנה עובדת בתדריות גבוהה, בין שבועיים לחודשים, עם העדפה לטוווח זמן קצר יותר.
4. בעלי העסקים והמפתחים ח"בים לעבודה ביחד במהלך הפROYיקט.
5. בניית פרויקטים תבצעו סביר אנשים בעלי מוטיבציה. ת策ר להביא להם את הסביבה והתמייה שהם צריכים ולסמן עליהם שיישו את העבודה כמו שציר.
6. השיטה היולה והאפקטיבית ביותר להעברת מידע לצוות פיתוח ובתוכו היא שיחה פנים אל פנים.
7. תוכנה שעבדת היא המדריך הראשוני להתקדמות.
8. תהליכי אג'ילם מקדמים פיתוח בר קיימת. נתני החסוט, המפתחים והמשתמשים צריכים להיות מסוגלים לשמור על קצב קבוע ללא הגבלת זמן.
9. תשומת לב מתחמשת למציאות טכנית ויציבות טוב משפרת את הזריזות.
10. הפשטות היא חיונית. (יכול להתגש עם עקרונות אחרים, הרעיון להיות פשוט איפה שאפשר)
11. הארכיטקטורת, הדרישות והעיצובים הטוביים ביותר מתקיימים בעזרת צוותים שמתחברים עצמם.

12. במרוחץ זמן קבועים, הוצאות משקף כיצד להפוך לאפקטיבי יותר, ואז מכון ומתאים את התנהגוותו בהתאם.

השוואה בין אג'יל למפל המים:

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

:Scrum

- גישת פיתוח המקדמת מיקוד, בהירות ושקיפות לתוכנו וIMPLEMENTATION פרויקטים.
- בשימוש של ארגונים גדולים ועד קטנים.
- סט ערכים, תפקידים, תהליכי איטרטיביים וכליים:
  - עבודה בצוות, תקשורת בין אישית, העצמת הפרט
  - Scrum Master, Product Owner, Developers, Testers
  - מחזורי תוכנן, תוכן, פיתוח אוינטגרציה.
  - Backlogs, Burn down Charts

תועלות צפויות מאימוץ Scrum:

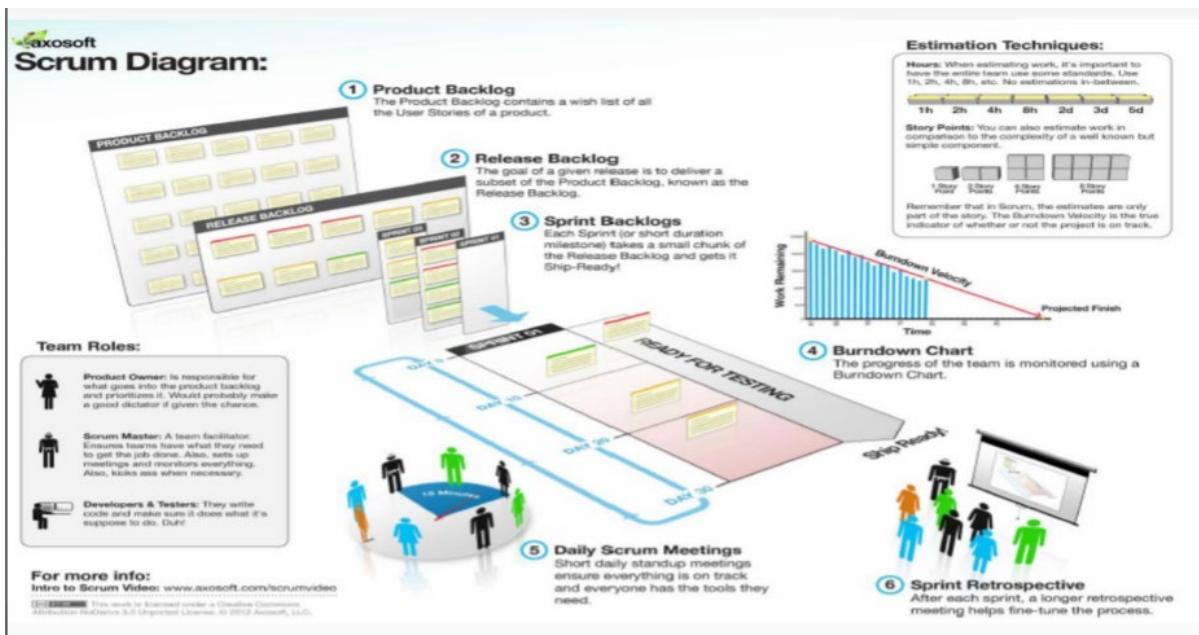
- הגברת קצב הפיתוח
- יישור קו ותאום מטרות הפרט והארגון
- ייצירת תרבות ארגונית (מושג כולל להתנהגות הארגון) לפי ביצועים
- תמיכה ביצירת ערך לבניין העניין
- ייצירת מנגנוני תקשורת וסנכרון יעילים בין כל המעורבים
- העצמת התפתחות אישית ואיכות חיים בין חברי הצוות

Product backlog: תוכנות שאתה רוצה לישם אך עדין לא נתת עדיפות לשחרור

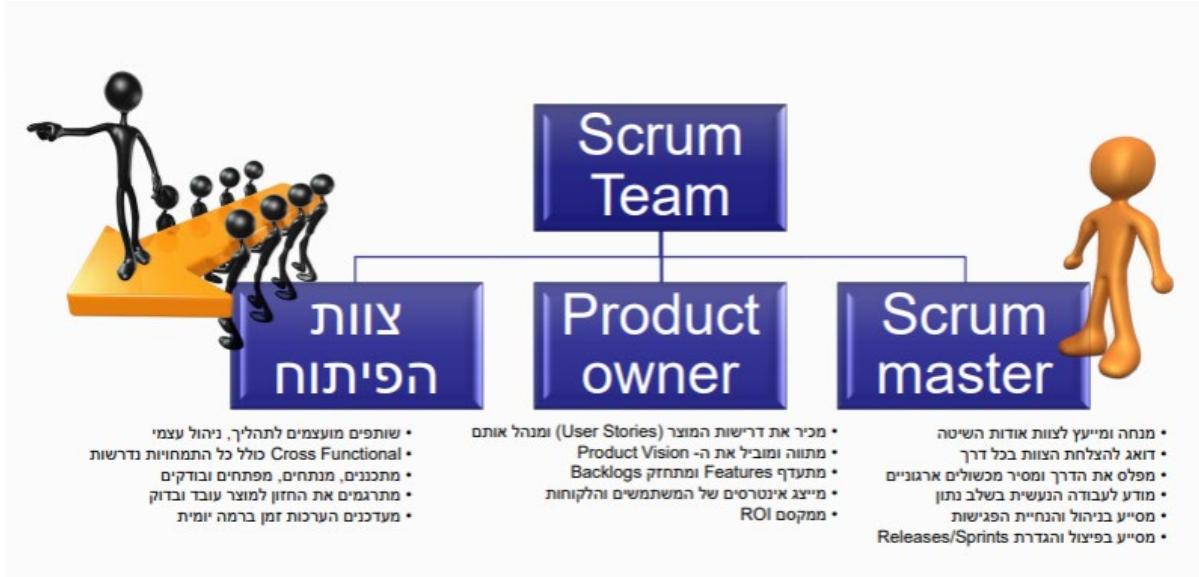
Release backlog: תוכנות שצריך לישם עבור מהדורות מסוימת

Sprint backlog: סיפור משתמש שצריך להשלים במהלך פרק זמן מסוים

GBT כולל של Scrum



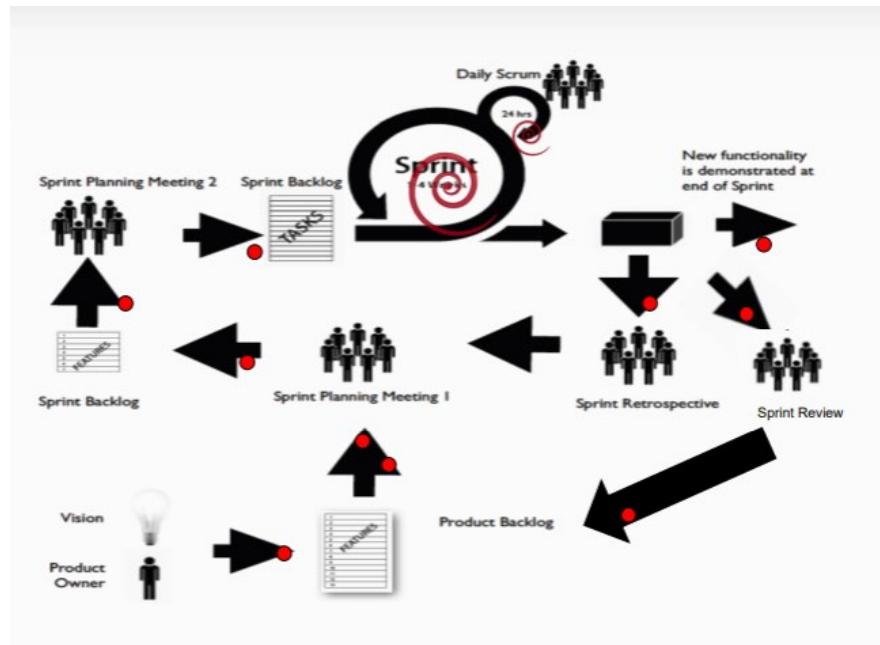
שחקנים – Scrum



הנהלה:

- אין מנהל פרויקט, הצוות מנהל את עצמו
- תפקיד הנהלה חשוב אך משתנה
- যিচ্রিম মডেল উকি মেস্পেক মশাবিম লেজেন্সি জোট
- מכבדים את החוקים ורוח השיטה
- מסיעים בסילוק מכשולים וסוגיות מפריעות
- מספקים הנחייה ותורמים מניס'ונם לצוות
- מתאגררים את הצוות למציאות
- מנהל ≠ אומנת, אלה גורומנטורקואוצ'ר
- לא מנהנית פתרונות, מתחאל ומכוון לקראת פתרון

התהילה:



:Scrum events

- Sprint Planning: ישיבת צוות עבור הפיתוח.
- Daily Scrum: ישיבת צוות יומית.
- Sprint Review: סקר עבור המוצר חלק מה מוצר המתקיים על ידי בעלי העניין.
- Sprint Retrospective: רטרוספקטיבה של הסPRINT היא מפגש חוזר שנערך בתום סPRINT המשמש לדין במה שהלך טוב במהלך הסPRINT הקודם ומה ניתן לשפר לkrarat הסPRINT הבא, מתקימת לאחר סיום הסPRINT ולפני תכנון הסPRINT.

תוצרים ב- :Scrum

### SCRUM ARTIFACTS



בעיות ב – Scrum

- חוסר בהירות בנוגע לתוכלה הסופית של הפרויקט.
- צוות שמנהל את עצמו ? (לא מתאים לצוות שאפשר לסמור עליו)
- חלוקה הפיצרים לחידות עבודה קטנות?
- אתגר הצוותים הקטנים
- דריש שינויים ארגוניים – ברמת הנהלה וגם ברמת הצוותים המפתחים
- תפקיד master - Scrum master - קשה לביצוע וקשה לאישוש.

## השוואה בין מודלים לינאריים לאג'יל:

Agile	ليناري	
מעט ידע מראש, הרוב נבנה בהמשך התהילה'	ידועות מוקаш	הדרישות בפרויקט
גמיש. מתබב בברכה	קשה. גורר עלויות נוספת	שינויים בתוכנות הפרויקט
נמוכה	גבוהה	רמת הסיכון להצלחת הפרויקט
גבוהה כל אויר תהילך פיתוח הפרויקט	גבוהה מאד בהתחלה, נמוכה בהמשך	עורבות הלקוח בתהילך הפיתוח
באיסטריות	בעומק אחות	מסירה ללקוח
לאנשים (לliquות ולבאות הפיתוח)	להתאייר	אוריגינטיצה של המודול
צוותים קטנים (3-5)	צוות גדול	גודל הצוות
הערך העסקי ללקוח	בעש כל הדרישות	מודדים להצלחה של הפרויקט

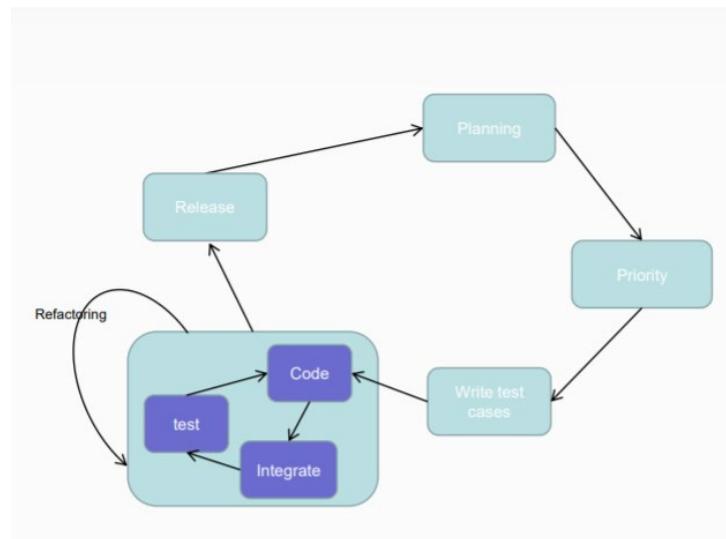
## מודל XP (Extreme Programming):

- טכניקה לתהילך פיתוח תוכנה זרייז המבוססת על עקרונות האג'יל
- נותנת דגש על **עורבות הלקוח** בתהילך פיתוח התוכנה, תקשורת טובה בתחום הלקוחות ועובדיה באיטרציות פיתוח.
- הערכים המרכזים עליהם מושתת XP:
  - תקשורת, פשוטות, משוב, אומץ, כבוד.

## 12 עקרונות של XP:

- .1. הרבה תכנונים :The Planning Game
- .2. לשחרר כל פעם קצרה :Small Releases
- .3. מטפורה :Metaphor
- .4. עיצוב פשוט :Simple Design
- .5. בדיקות מתמשכות :Continuous Testing
- .6. ארגון מחדש :Refactoring
- .7. זוגים על אותה תוכנה :Pair Programming
- .8. בעלות שיתופית :Collective Ownership
- .9. אינטגרציה מתמשכת :Continuous Integration
- .10. 40 שעות שבועית :40 hours per week
- .11. הלקוח תמיד נוכח :On-site Customer
- .12. קוד סטנדרטי :Coding Standards

המודל:



## יתרונות XP:

- מעורבות הלקוח מגדילה את הסיכוי שהתוכנה המוצרת תענה על הצרכים של המשתמשים.
- מקטין את הסיכון בפרויקט.
- בדיקות אינטגרציה מתמשכות מס' עות להגבר את איכות העבודה.
- זמן הפיתוח קצר יוטר, כי מכינים את הבדיקות בשלב ההגדרות.

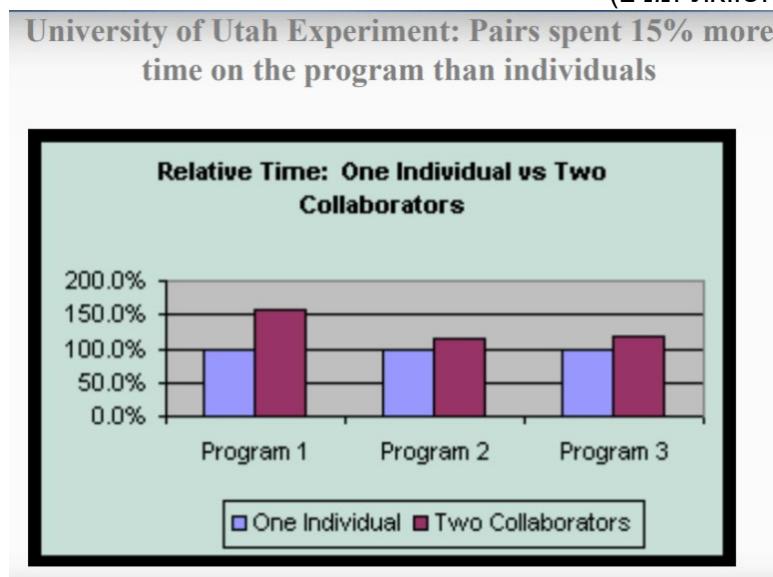
## חסרונות XP:

- XP מיועד לפרויקט אחד, שmpsota ומותחזק על ידי צוות אחד.
- XP פגיע במיזוג למפתחים סוליטרים אשר לא עובדים טוב עם אחרים / שחובבים שהם יודעים הכל / שאינם משתפים את הקוד "המעולה" שלהם
- XP לא יעבוד בסביבה שבה לקוחות או מנהל מתעקש על מפרט מלא או עיצוב לפני שהם מתחילה לתכנן.
- XP לא יעבוד בסביבה שבה מתכנתים מופרדים מבחינה גיאוגרפית (או לוחות זמנים שונים)

## Pair Programming:

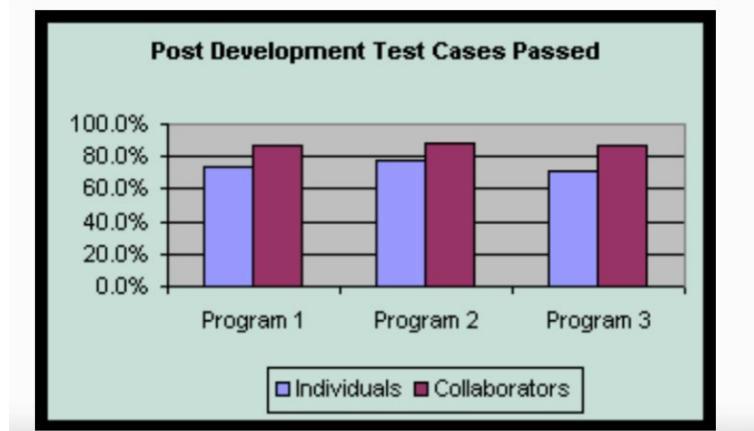
- 2 מפתחים יושבים ביחד באותו מחשב.
- זוגות דינמיים
- הרעיון הוא שני אנשים בוודאות י כתבו קוד יותר טוב יותר מאשר אחד (עם פחות באגים)
- 2 המתכנתים הם בעלי רמות שונות של ניסיון, אבל זה לא נועד לחניכה של מפתח חדש.

האם זה טוב? (השוואת זמנים)

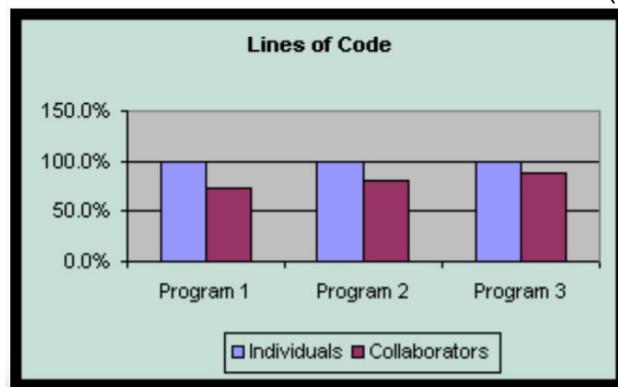


## (השוואה תוצאות הטסטים)

**University of Utah Experiment: Code written by pairs passed more test cases than code written by individuals**



## (השוואה בין שורות קוד)

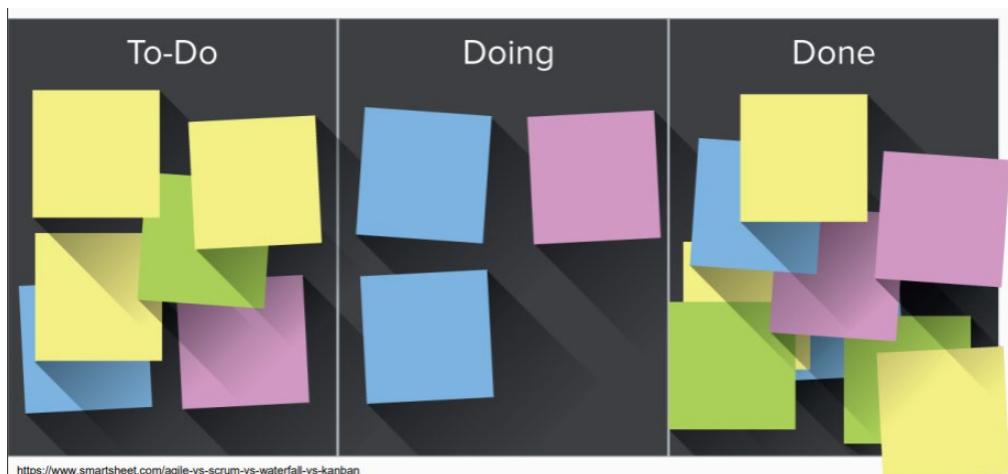


\* ע"פ מחקרים אין מסקנה חד משמעותית מה יותר טוב.

## מודל Kanban (סימן ויזואלי בפנים):

- טכניתה לתחילת פיתוח תוכנה זריז המבוססת על עקרונות Agile.
- Kanban מותנת דגש לצד הויזואלי של התהילה - מה לייצר, متى לייצר וכמה לייצר.
- השיטה מעודדת ביצוע שינויים קטנים ודרמטיים במערכת.

## לון Kanban:



## השוואה בין Scrum ל-Kanban:

Scrum	Kanban
לכל חבר צוות יש תפקיד מוגדר מראש, שבו Product Master משמש כマאַנְטָן אֲגִילִית של הצוות, Owner Product Manager מסרתו ויעדיים וחברי הצוות מבצעים את העבודה תוך עבודה צמודה עם ה-Product Owner.	"אין תפקידים מוגדרים מראש עבור צוות". מעדודים את הצוות לתקשר ולפעול ביחד אחד מאנשי הצוות תקוע או יחד עםם בעבודה
זמן אספקת תוכנה נקבעים באמצעות הנדרת ספרנים	מצרכים ומישות מסוימים בהתאם לביקור (on demand)
יש שימוש ב"תשיכת משימות", אבל ברמה של קבוצה של משימות הנמשכת ככל בכל ספרינט (קבוצת המשימות נקבעת בתכנון הספרינט)	חברי הצוות "מושכים" מישות חדשנות לאחר שהמשיכו בשלה הבוגר קסם מהחכם העליון שהוגדר עבור שלב זה
מידית תפוקה באמצעות הספק המבוצע בספרנים הקווים. והספק נמדד בדרך כלל על ידי מושג הנקרא Story Points או על ידי מספר המשימות שהותינו בספרינט	מידית תפוקה באמצעות "זמן מחזור" (זמן העוגר מתחילה טיפולו במשימה עד סיום הטיפול המשימה)
שוניים ביחס לספרנס אינם רצויים. הספרנט הוא "אורו זמן מונע" עבור הצוות	ניתן לבצע שינויים בכל נקודה בזמן

## יתרונות של Kanban:

- מגביר את הgemeישות לשימושים
- מצמצם בזבוז זמן - תמיד ממתינה למתקנת עוד משימה
- קללה להבנה ולהתמעה, ניתן להשתמש על גבי מתודולוגיה אחרת
- מקוצר את זמני המסירה של מישיות

## חסרונות של Kanban:

- כל הזמן צריך לעדכן את הלוח.
- אין בלוח מידע לגבי זמני המסירה.

## באיזה מודל נבחר?

- נבחון את הגורמים הבאים:
  - הארגון: מטרות, אילוצים, תשתיות..
  - הנהלה: פתיחות, הבנה מקצועית, רצון לשינויים..
  - העובדים: הכשרה מקצועית, יחס אנווש, פתיחות..
  - המוצר: מורכבות, ייחודיות, האם הדרישות ידועות..

Scrum ו-Kanban הוציאו ב-2018 מדריך משותף \*\*.

**הרצאה 5:**

ניתוח מערכת – מדרישות למודלים (חלק א')

מנתח מערכות ותוכרי הגדרת דרישות

תפקיד מנתח המערכת:

- מנתח מצב קי'ם ובוית הדרישות מחשוב
- מגדר אפיון למרחב הבניה וקוי מתאר לפתרון
- תוחם ומחדד מטרות מערכת
- אוסף ומוסוג דרישות מלוקחות ומשתמשים
- תוכרים ופורמלים:
  - מסמך יזום
  - ניתוח מצב קי'ם
  - הגדרת דרישות
  - מודלים, דיאגרמות ופורמלים סטנדרטיים
  - אפיון פתרון

שלב הניתוח:

- פונקציות (תהליכי) המערכת
- הקלטים של המערכת והמקורות שלהם
- הפלטים של המערכת והיעדים שלהם
- הנתונים שייאגרו בסיסי הנתונים למערכת

ובכלליות שלב זה הוא הגדרה מדוקית של ביצוע המערכת.

תרגומם דרישות למודלים – גישות שונות:

- **גישת התהליכים**
  - פירוק המערכת לתהליכי פונקציות
  - דגש על זרימה וטרנספורמציה של הנתונים
- **גישת יסודות הנתונים**
  - פירוק המערכת לחבילות נתונים המייצאות יסודות אבסטרקטיות או ממשיות
  - דגש על מבנה חבילות הנתונים וברחת ביצוע
- **גישות משולבות**
  - המערכת מייצגת ע"י יסודות, הגדרת יחסים מבנים בין היסודות, כל ישות משתתפת בתהליכי המשנים את נתונה

כעת, נפרט על כל גישה בנפרד

**גישת התהליכים** (Data Flow):

גישת מודל זרימת נתונים (Data Flow Modeling):

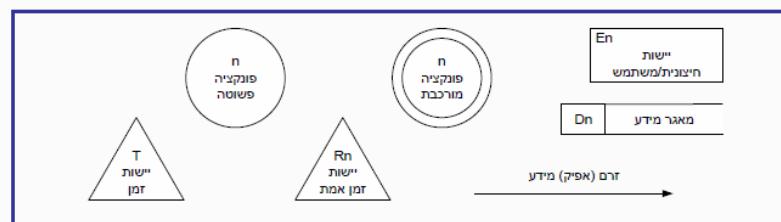
- תיאור התנהגותי של המערכת מנקודת מבט זרימת הנתונים, מהישויות החיצונית אל פונקציות המבצעות עליהם טרנספורמציות ועוד לאחסן => תרשימים DFD
- תיאור כל היסודות החיצוניים באינטראקציה עם המערכת => תרשימים קונטקטוט SCD
- מילונים מלאים את הפירוק (מילון פונקציות, מילון מאגרי מידע, מילון יסודות)

המודולוויות הדוגלוות ב- Structured Analysis (SSA) מtabססות על גישה זו.

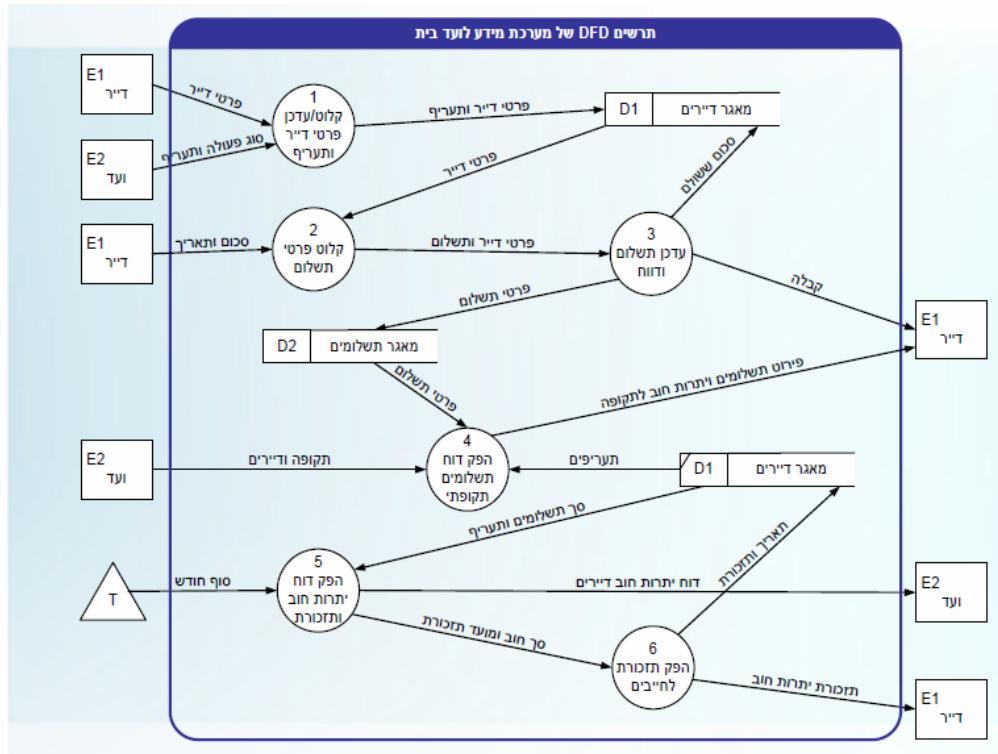
כiom השימוש בתרשיMI DFD - SCD איננו נפוץ במחוזותינו.

תרשים DFD מורכב מחייבים סמלים עיקריים:

- על גול** (יחיד או כפול) מצין פונקציה (פשוטה או מורכבת) שהמערכת מבצעת
  - מלבן** מצין ישות חיצונית או משתמש במערכת
  - מלבן מוארך** ובו משבצת מצד אחד ופתח מצד الآخر מצין מאגר מידע
  - משולש** מצין יחידת זמן או מכשיר הקשור למערכת הפעול בזמן אמיתי
  - חץ** מצין זרם מידע (אפיק מידע) בין פונקציות למרכיבים אחרים במערכת



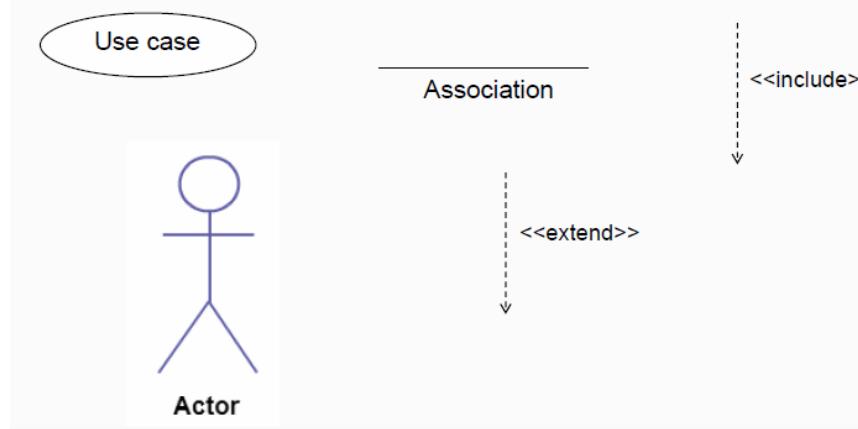
דוגמה של תרשימים DFD של מערכת מידע לוועד בית:



גישה תהליכיית מודרנית (Use Cases):

- הגישה החדשה למודול: Use Case Model
    - תיאור התנהוגות של מהלך שימוש במערכת ע"י משתמש (שחקן/תರחיש) או תיאור מהלך עסק' שלם
    - תיאור הcoilל הגדרת מטרה, משתמש, תנאים מקדים לתרחיש, טריגר לתרחיש, מהלך עניינים אפשרי ומה יהיה מצב המערכת בסיוםו.

## מרכיבי תרשימים Use Case:



שחקנים:

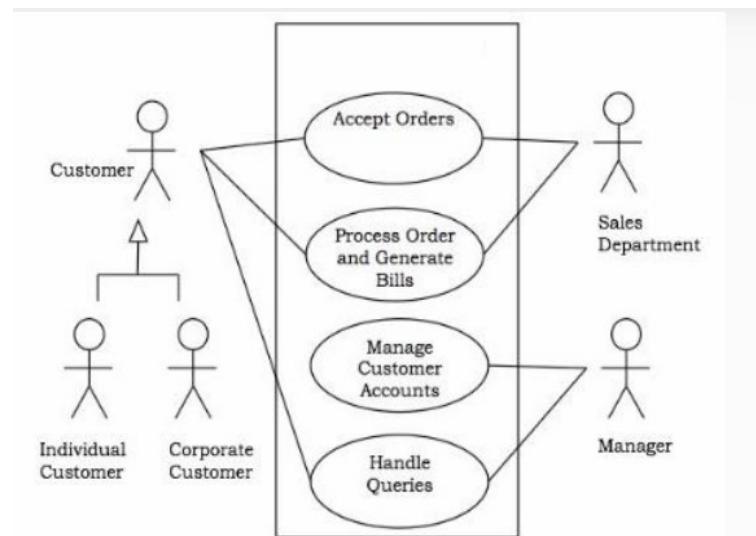
- ישות חיונית אשר יש לה קשר עם המערכת
- לכל ישות יש תפקיד ביחס למערכת
- השחקנים מייצגים את תיחום המערכת אך הם לא חלק ממנה
- שחקן יכול להיות ישות חיונית, מערכת אחרת או כל אמצעי מחשב
  - עובד, לקוח, אורח
  - ארגון – מדור בחברה, חברת משלוחים
  - התקן – קופה רושמת, מדפסת
  - מערכת חיונית – מערכת כספים מלאי, מערכת נוכחות עובדים

דריכים למציאת השחקנים: מי משתמש במערכת? מי מתקין את המערכת? מי מפעיל את המערכת?  
 מי מתחזק את המערכת? מי מכבה את המערכת? באילו מערכות אחרות משתמשים במערכת זו?  
 מי מקבל מידע ממערכת זו? מי מספק מידע למערכת? האם קורה משה באופן אוטומטי ברגע  
 הנוכחי?

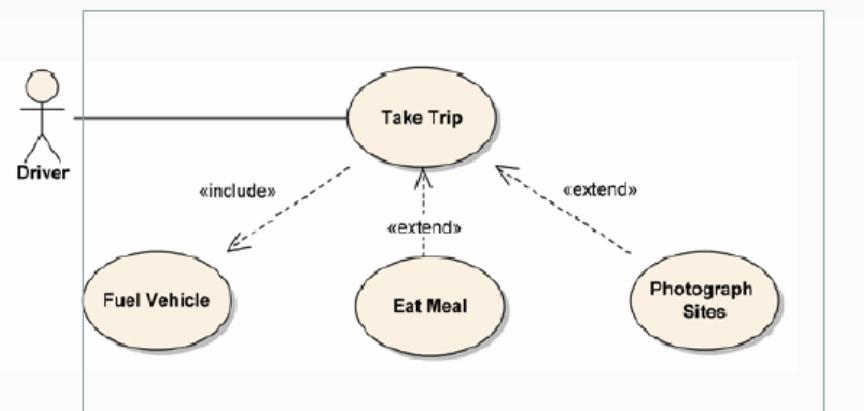
משיכתழמן מסופומט (שחקן) - לקוח של הבנק  
 הרשמה בכונסה למילון (שחקן) - אורח  
 הפתק דוח מכירות (שחקן) - מנהל המכירות  
 קניית מוצרים בסופרמרקט (שחקן) - לקוח של הסופר  
 שליחת דרישת תשלום (שחקן) - הספק  
 קביעת מסגרת אשראי ללקוח (שחקן) - מחלקת אשראי  
 דיווח תקלות במבנה (שחקן) - מהנדס/ מפקח בנייה

דוגמה: מערכת הזמן  
 מערכת המוחשבת של הזמן, ניהול מלאי ולקוחות:

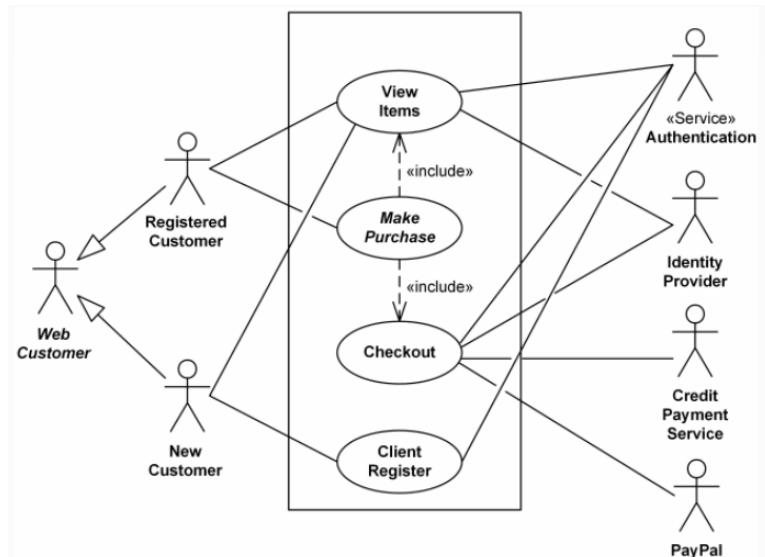
- הלקוחות יכולים להיות פרטאים או עסקאים
- הלקוח עושה הזמן, מחלקת מכירות מבצעת אותה ויוצרת חשבון – הלקוח מקבל את החשבון
- המנהל מנהל את חשבונות הלקוחות ועונה על שאלות ששוואים הלקוחות.



שימוש ב- << extend >> & << include >>



דוגמה נוספת:



**גישה ישוויות הנטונימט** (Entity Relations Model) גישה מבוססת נתונים נטוונים :

- Objects & Data Stores •
- תיאור המערכת כאוסף מאגרי מידע שאפשר להפעיל בצד עיבודים

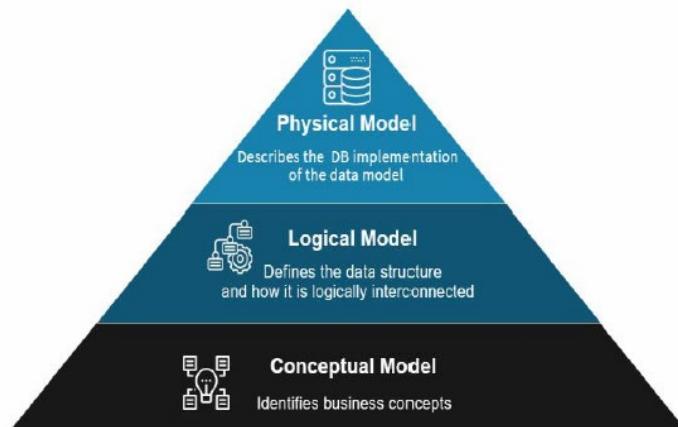
- פירוק המערכת לאובייקטים המעבירים מסרים זה לזה במהלך העבודה וمبקריםם שירותים זה מזה
- שימוש בתרשי Entity Relations Diagrams (ERD – Entity Relations Diagrams)
- שימוש מודרני בתרשי סטנדרטים UML כדוגמת Class Diagrams ו- Activity Diagrams
- מתודולוגיות הדוגלות בניתוח מונחה עצמים (Object Oriented Analysis OOA) מtabססות על גישה זו

**סיווג נתונים:**

- ללא מבנה: מסמכים בעלי טקסט חופשי, דפי HTML, או תמונות וסרטים.
- מובנים: הגדרה ברורה של מבנה (סכמה), בפרט נתונים בסיס נתונים רלוונטיים
- מובנים למחזאה: Mails, XML הכללים הגדרות המשורט הרככיה או משמעות (Meta Data)

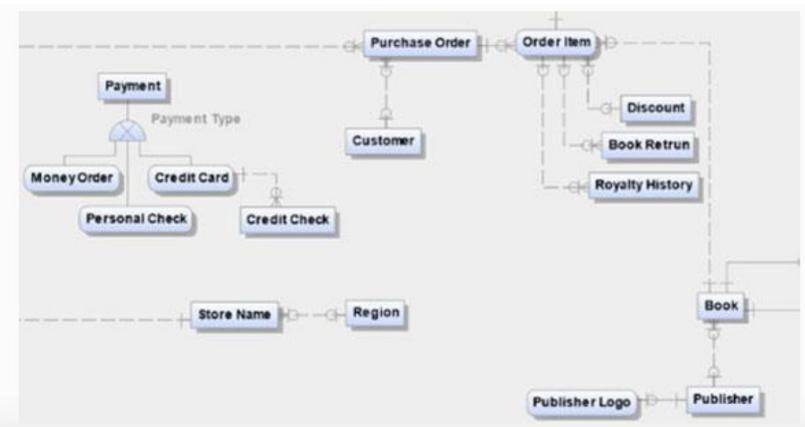
**שנים שלוש רמות מודלים:**

- מודל קונספטואלי:
  - תיאור סמנטי "טהור" של תחום הבעה (משמעות)
  - תיאור ישות (Entities), תכונות (Attributes) וקשרים (Relationships)
- מודל לוגי:
  - יציג תחום הבעה במושגי טכנולוגיית ניהול הנתונים (לדוגמה: טבלאות, רשומות)
- מודל פיזי:
  - כיצד הנתונים נשמרים ב-DB או במסגרת מאפייני ניהול RDBMS מסווגי ספציפי



**1. מודל קונספטואלי נתונים:**

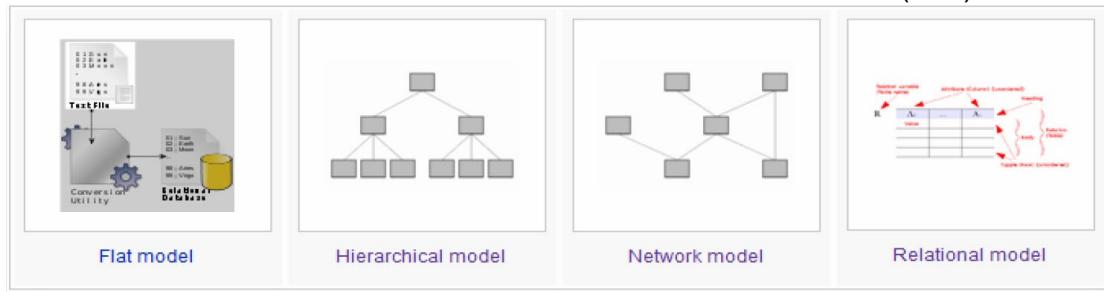
- תיאור מה מכילה המערכת ברמת מושגים, ישויות והיחסים ביניהם
- מספקת פרספקטיביה על מבנה הארגון באמצעות הגדרת יחסים בין הישויות העסוקיות הקיימות או מתוכנות



## 2. מודל לוגי:

- תיאור נתונים במושגי הטכנולוגיה וגישה ניהול הנתונים
- הבולט מבין שלושת סוגי המודלים, יתורגם פעמים רבות למודל רלציוני עקב תפוצת מסדי נתונים מבוססי SQL או למודלים מסווג SQL
- חסרון מרכזי: אובדן חלק ניכר ממרכיבי המשמעות במסגרת תהליכי הפיתוח יחסית למודל הקונספטואלי

גישה ליצוג (מודל נתונים לוגי – שטוח, היררכי, רשתית או טבלאי):



## 3. מודל פיסי לנתונים:

- תיאור האספקטים הפיסיים לאחסון הנתונים
- הגדרת מקום וחוקי הקצאת מקום, חלוקה פיסית של אחסון טבלאות וכו'
- לרבות פחות משמעותי בשלבי הניתוח והעיצוב
- אופטימיזציה של מימוש ע"י מומחים בתחום באמצעות כלים המלווים RDBMS מסחריים

סדר יצירה ומעבר בין המודלים:

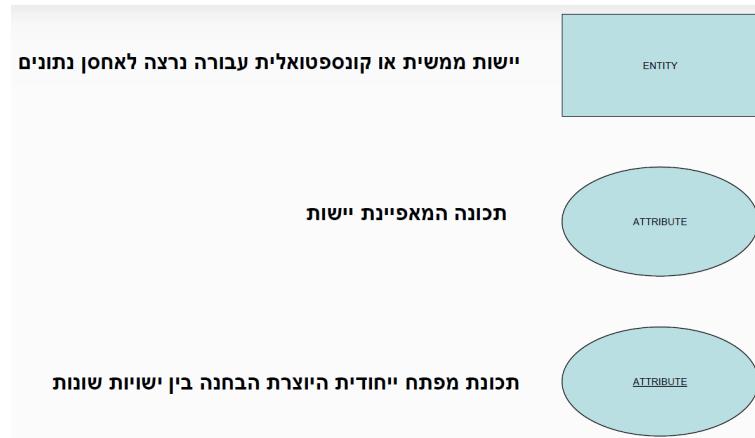
- פיתוח מודל קונספטואלי באמצעות E-R (Relations Entity) (Relations Entity-R)
- תרגום למודל לוגי לצורך מימוש, לרבות במסדי נתונים ובכלל זה גם תהליכי הנקרוא נירמול
- ביצוע התאמות למודל הפיזי מתוך הגדרות המודל הלוגי

## מודול מבוסס E-R:

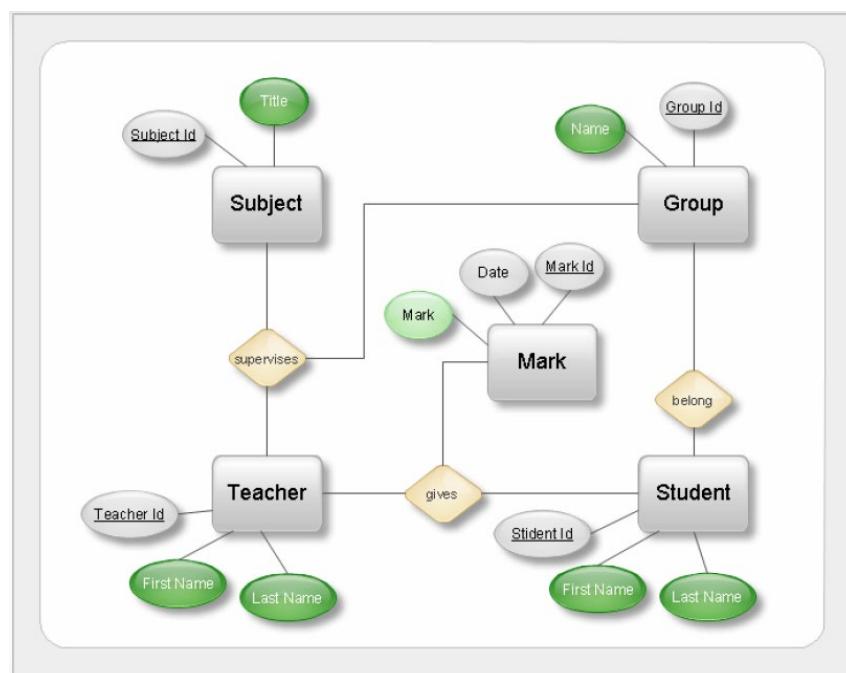
- תיאור קונספטואלי של נתונים ויחסים
- גישת Top-Down
- משמש כבר בשלב איסוף הדרישות לתיאור הנתונים שיידרשו או יאוחסנו במסדי הנתונים
- יכול לשמש לתיאור כל אוסף של רעונות ויחסים במסגרת תחום כלשהו
- יתורגם למודל לוגי כדוגמת המודל הרלציוני

- או יתורגם למודל אובייקטיבים

מרכיבי E-R Diagram:



יחסים E-R Diagram:



**גישהת מעורבות** (Object Process Methodology)

- מידול תהליכיים ומידול אובייקטים בכפיפה אחת
- כולל 2 סוגים אלמנטיים: ישויות וקשרים

סוגי ישויות:

- אובייקט – ישות המתקיימת במערכת בזמן נתון
- מצב – סיטואציה שבה יכול להימצא האובייקט
- תהליך – פעילות המשנה את מצבו של האובייקט

סוגי קשרים:

- קשר מבני – חיבור קבוע בין אובייקטים
- קשר פרודורי – חיבור אובייקט לתהליך לשינוי מצבו

בסיס לגישת מעורבות  
Model Based Systems Engineering

- מיועד לניטוח דרישות פונקציונאליות
- הדרישות מחוברות לאובייקטים באופן המשלב תהליכיים (Processes) עם נתונים (State)

בחינת תחום הבעה (Domain), התמקדות בתחום הליבה דורש התמחות וטיפול  
ספציפי

- יצירת מודל קונספטואלי לנוטונים בתחום המנותח
- אנקופוליצית השירותים העסקיים במערכת
- הניתוח מתחילה ממספר SRS או תהליך ישיר מול מראינים
- תוצרי הניתוח מתארים פונקציונאליות המערכת וכוללים:

דיאגרמות UC (Use cases)

דיאגרמות מחלקות (Class Diagrams)

דיאגרמת אינטראקציה כדוגמת Sequence Diagrams  
תיאור מסכים

דוגמה: ישות טיסה.

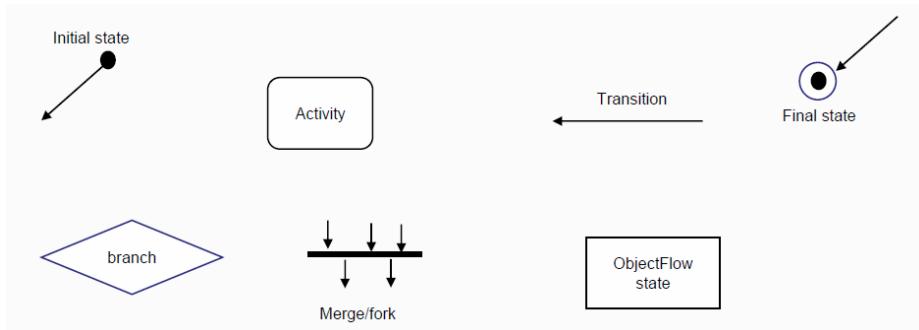
מאפייני הישות: קוד טיסה, תאריך טיסה, מוצא,יעד, סטטוס, תפוצה  
פעילות הישות: פתח, בטל, שנה יען, שנה תאריך טיסה, עדכן סטטוס

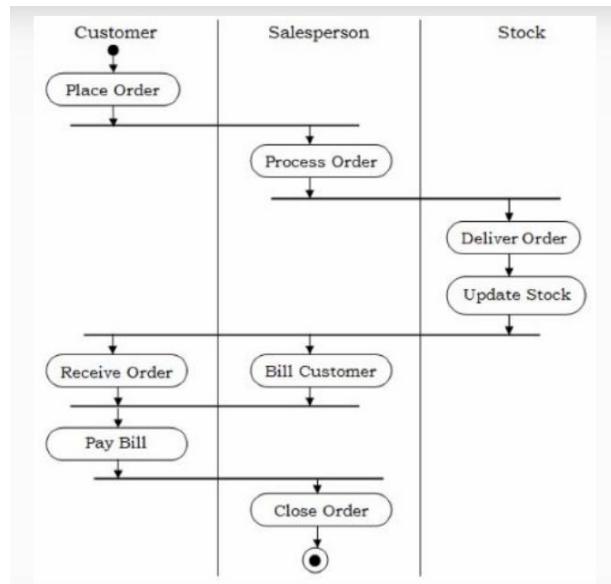
דוגמה של ישות טיסה + מצב: טיסה 232, תאריך טיסה 22.01.01, מוצא ת"א, יען נסיעה, סטטוס  
סגורה, תפוצה 400

כאשר סטטוס ותפוצה הם מאפייני המצב הנוכחי של הישות

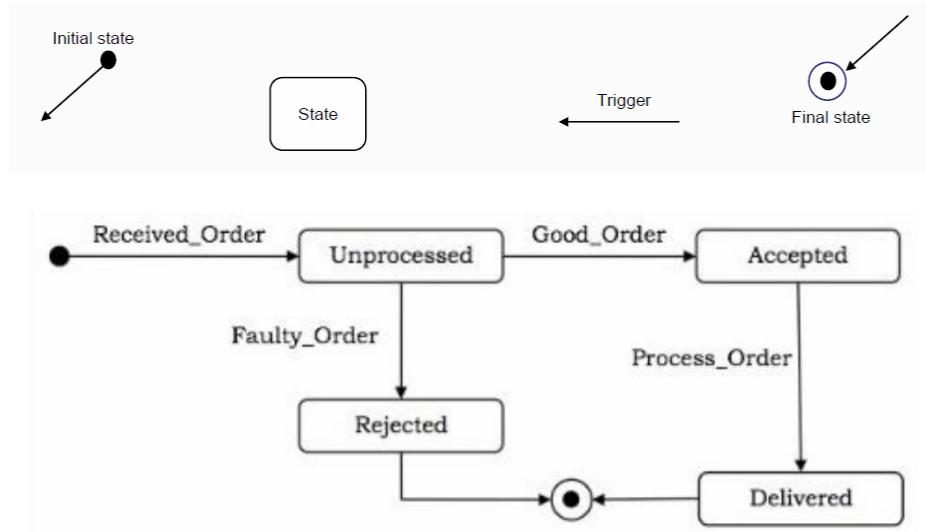
**Activity Diagram**

- תרשימים פעילות מציג את הזרימה מפעולות בתחום המערכת ובין השחקנים
- דרך להציג פעילות בתחום בדומה ל-work flow
- כל פעולה-base use case הופכת לתרשימים פעילות אחד



State Machine Diagram

- בתרשים מוצבים שמים את האובייקט במרכז
- התרשים מתאר את זרימת האובייקט ממצב אחד לשנהו ואת המצבים השונים שבו האובייקט יכול להיות לאורך ח' המ מערכת
- תרשים זה הוא מבוסס על האוטומט של פרופסור דוד הראל
- כל תרשימים מתאר מחלוקת אחת



התחלנו את שלב הניתוח בתהיליך פיתוח תוכנה  
ראינו שישנם 3 גישות ניתוח (גישת תהליכיים, גישת ישוות, גישה משולבת)  
ראינו דיאגרמות:

DFD (לא בשימוש כיום)	-
Use Case Diagram	-
ERD (מודל סוג 1)	-
Activity Diagram	-
State Machine Diagram	-

הרצאה 6

ניתוח מערכת - מדרישות למודלים ( חלק ב' )

نبיא כעט דוגמה שלפיה נבנה דיאגרמות:

מערכת לניהול חשבונות בנק:

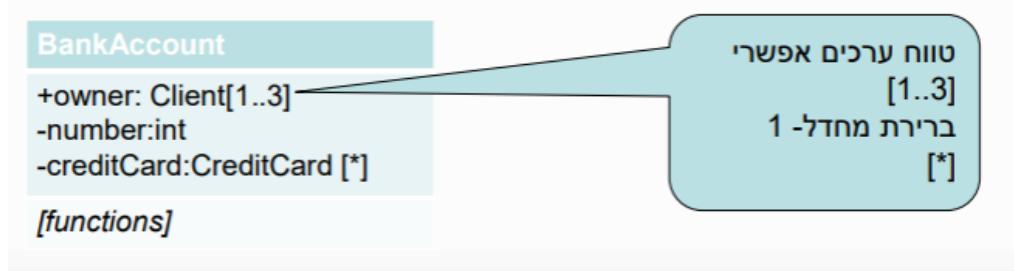
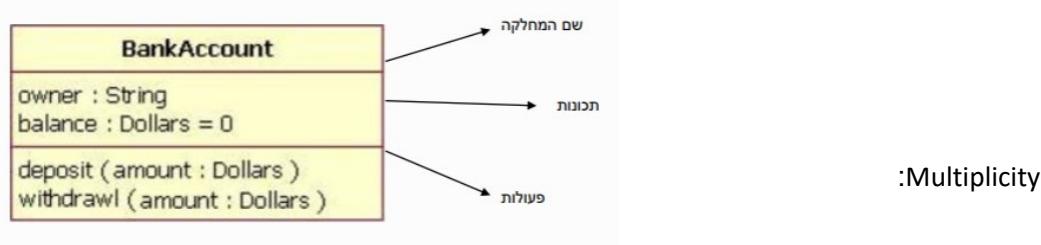
מערכת לניהול בנקים- סניפים, לקוחות, חשבונות וחלואות:

- בכל בנק יש הרבה סניפים.
- בכל סניף יש הרבה חשבונות
- חשבון בנק יכול להיות מ-2 סוגים- חשבון חיסכון וחשבון ע"ש
- לקוחות יש חשבון בנק אחד הסוגים או את שנייהם
- בכל חשבון יכול להיות רשום רק לקוח אחד
- החלואות משוכנות ללקוח ולסניף בנק (בנפרד מחשבונות הבנק)

:Class Diagram

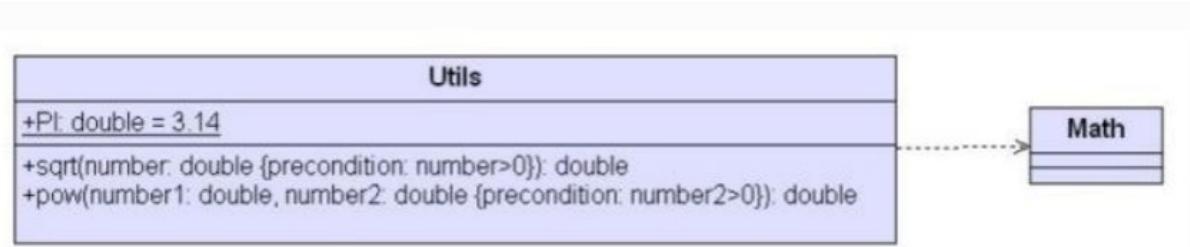
- מודל של מבנה המערכת ע"י מידול המחלקות, התכונות והפעולות.
- מודל UML class diagram הוא תכנית האב של המחלקות הדרישות לבניית התוכנה.
- מתכוונים מפתחים את המערכת בהתאם על מודל המחלקות שהוגדר
- זהו המודל הנפוץ ביותר ב-UML

סימון מחלקה:



קשרים בין מחלקות:

- |             |   |
|-------------|---|
| Dependency  | • |
| Association | - |
| Aggregation | - |
| Composition | - |

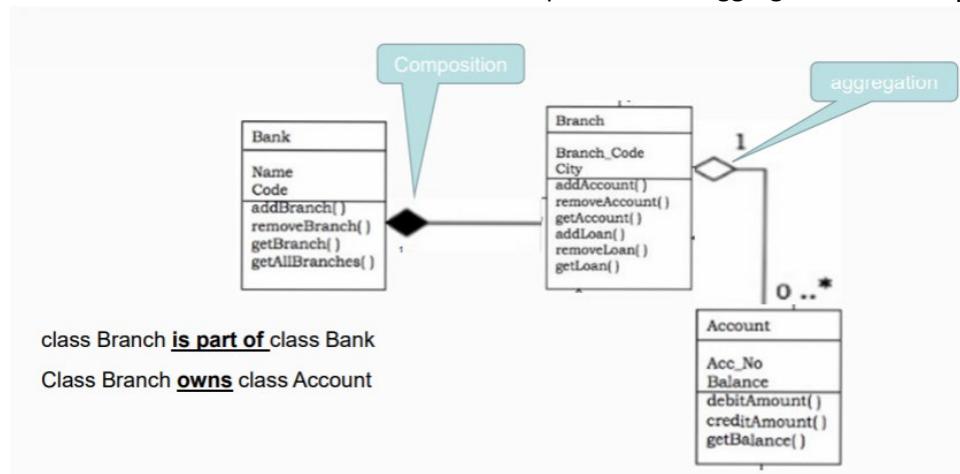


מחלקה UTILS משתמשת במחלקה MATH (קו מקווקוו)  
קשר של Association



למחלקה בנק יש מחלקה לקוחות (קו רגיל)

קשרים של Composition Aggregation ושל Composition



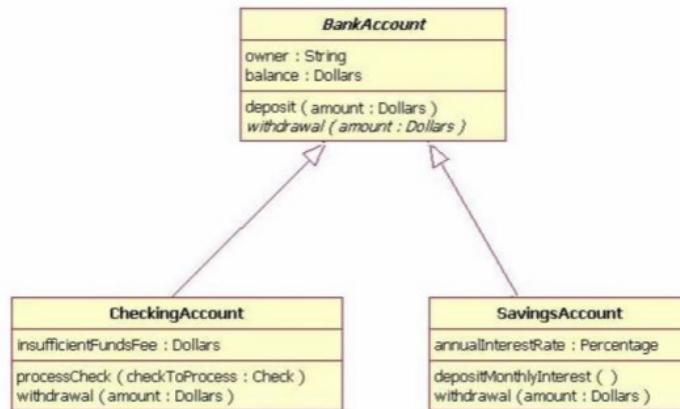
המשמעות המלא בשחוור מסמן שייכות ממש (Composition), לדוגמה גלאן חלק מרכיב.

המשמעות הריך מסמן על בעלות (Aggregation), לדוגמה החשבון הוא בעלות בנק.

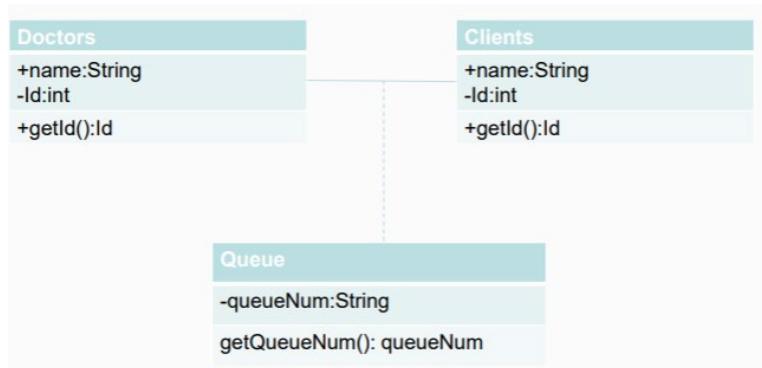
ברגע שסוגרים את הבנק לא צריך להலיך שיסגור את הסניפים (נסגרים אוטומטית בגלל הקשר)

אך אם סניף נסגר יהיה חייב לטפל בחשבונות.

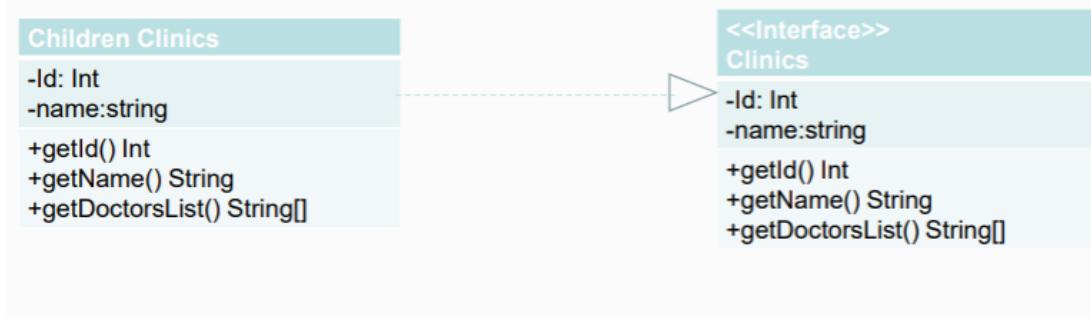
הורשה:



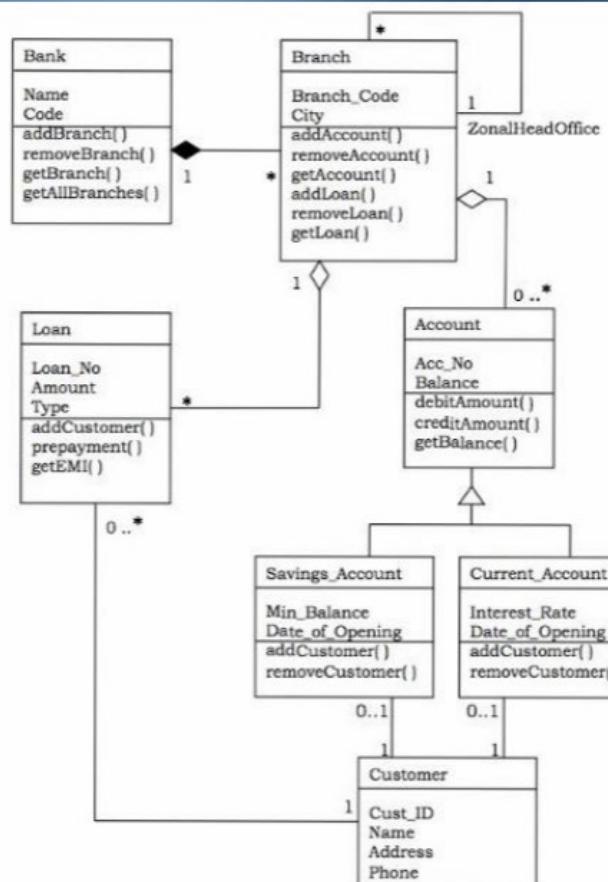
מחלקה קש:



לדוקטור יש **קשר** (ח' רגיל) עם המטופלים, שניהם **משתמשים** (ח' מקוווקו) בטור. תיאור Interfaces (ק' מקוווקו) ומחלקות שימושים אותם:

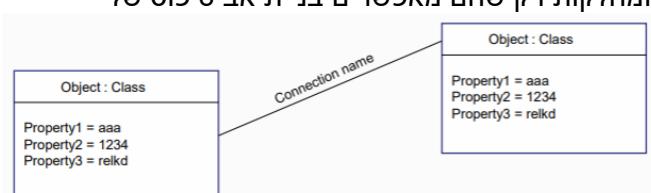


דוגמה ל-class diagram



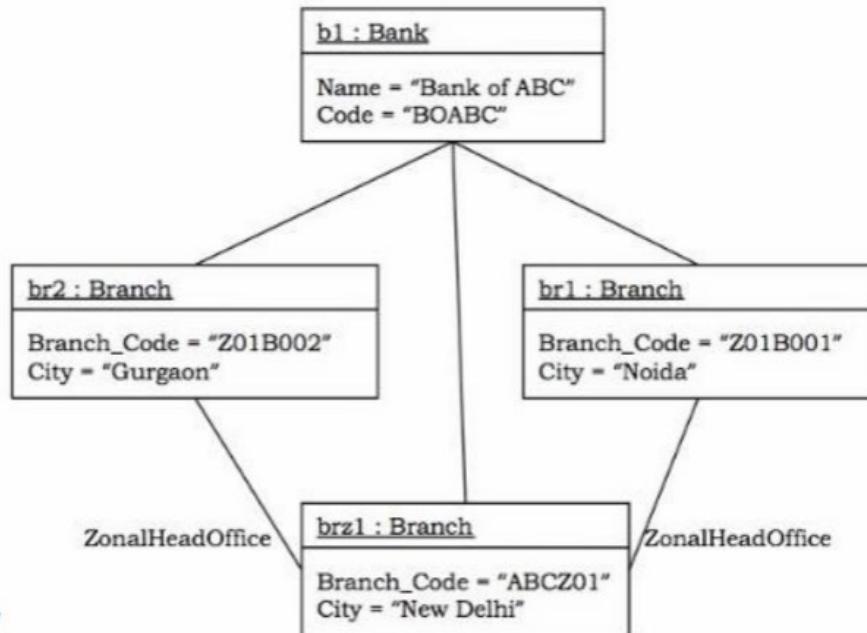
Object Diagram

- דיאגרמת אובייקטים מתארת מופע של מודל המחלקות, בדומה לתרחישים אמיתיים שעל בסיסם בונים את המערכת.
- דיאגרמת אובייקטים היא מודל סטטי (בדומה למודל המחלקות)
- השימוש במודל האובייקטים דומה למודל המחלקות רק שהם מאפשרים בנייה אבטיפוס של המערכת מנוקדת מבט מעשייה.



שיטות סימון:  
\* קו רציף זה קשר מסוים

## דוגמא ל - Object Diagram



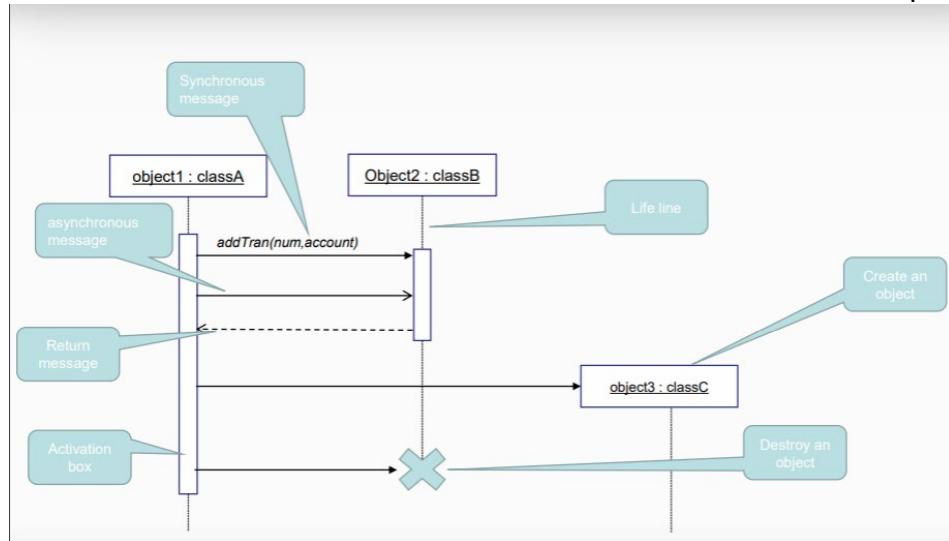
## הבדלים בין Class ל- Object

Object diagram	Class diagram
מבנה	מכיל 3 חלקים: שם, רשימת תכונות ורשימת פעולות
שם המחלקה	הפורמט מורכב משם האובייקט + נקודותים + שם המחלקה (Tom:Employee)
רשימת התכונות	מגדיר את התכונות של המחלקה
רשימת הפעולות	כלולות
קשרים	מוגדר הקשר בין מחלקות- שם הקשר וכמוות הקשר.

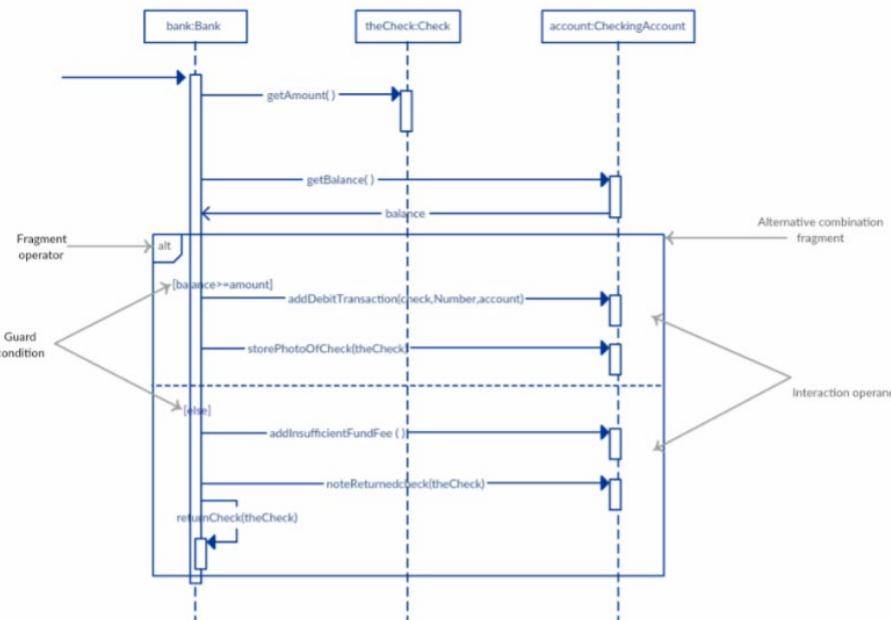
## Sequence Diagram

- תרשימים רצף הממחיש את סדר הפעולות ברצף של הזמן.
- תרשימים רשות נכתוב בצורה של תרשימים זו ממד:
  - על ציר ה-X נמצאים האובייקטים
  - על ציר ה-Y מוקומות ההודעות שהאובייקטים האלה שולחים.
- לכל פונקציונליות נכון תרשימים רצף נפרד.

## שיטת סימון:



שולח הודעה ומוחכה לתשובה.  
שולח הודעה בלי לחכות לתשובה



כאשר יופיע `alt` בקצה השמאלי העליון של הריבוע וקו מקווקו באמצעות זה אומר שהפונקציה.

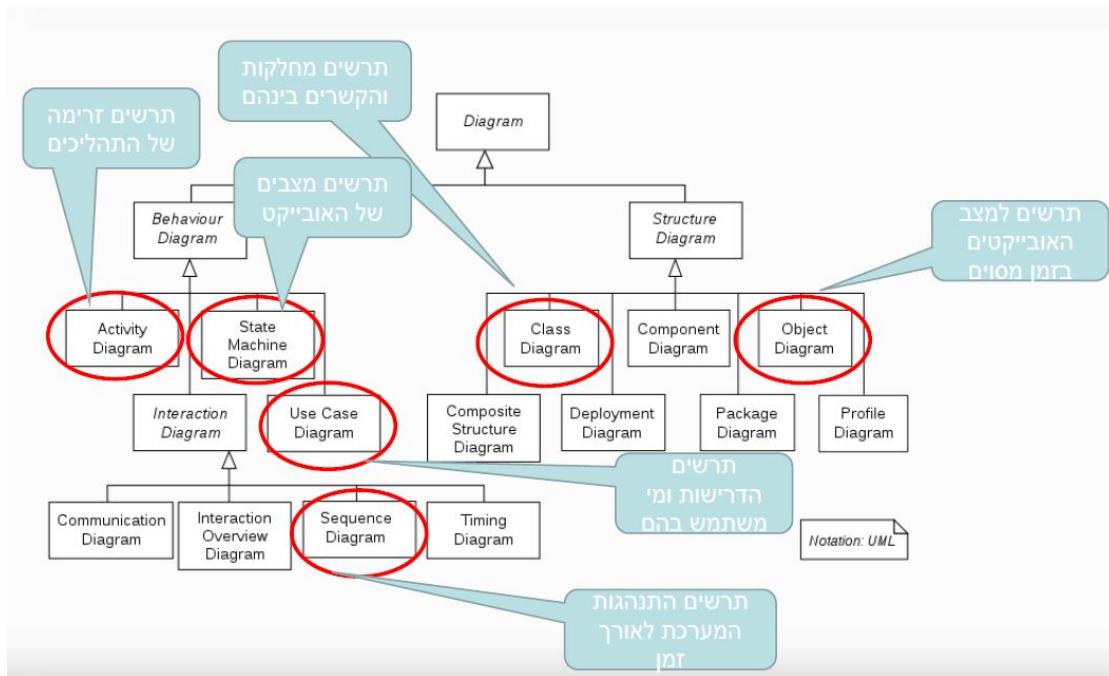
## הבדלים בין Sequence Diagram לבין Activity Diagram

Activity	Sequence
דיאגרמת התנהגות	דיאגרמת אינטראקציה
מתמקד בתהילך של האובייקטים ונוטן דגש לריצף ולתנאים שיש בתהילך	מתמקד בהודעות שמועברות בין ישויות במערכת.
סדר ביצוע הפעולות לא מודגש	נותן דגש לסדר ביצוע הפעולות
כללי יותר ופחות מדויק	התהילך יותר מדויק ומפורט

דיאגרמות ה-UML:

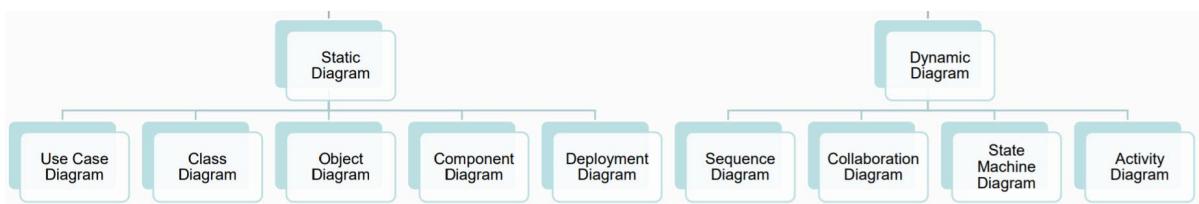
ניתן לחלק את הדיאגרמות הקיימות ב-UML ל-2:

- דיאגרמות המתארות מבנה
- דיאגרמות המתארות התנהגות או אינטראקציה.



חלוקת נוספת:

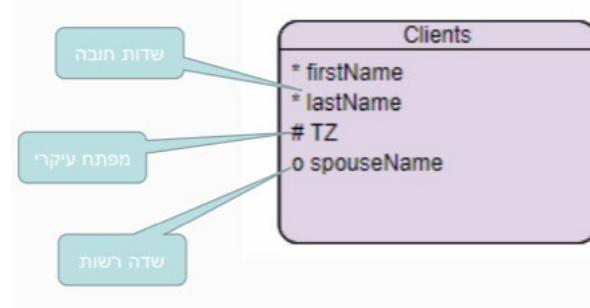
- דיאגרמות סטטיות.
- דיאגרמות דינמיות.



(קשר לשויות)

נכיר כי לממנו מודל של ERD מסווג 1 בהקשר של גישת השוויות. שאר הדיאגרמות הם בגישה המעורבת.

הציג ישות במודל:

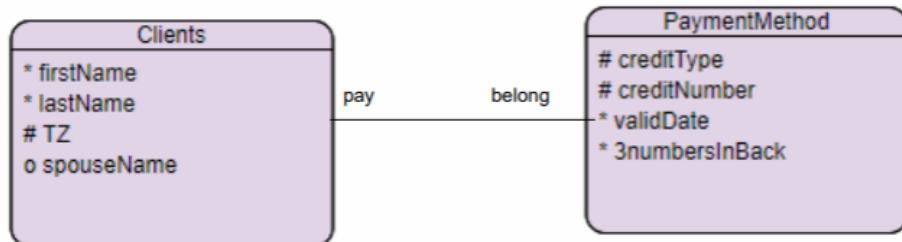


- (\*) – שדות חובה
- (#) – מפתח
- (o) – שדה רשות

קשרים בין ישות – Relationship:

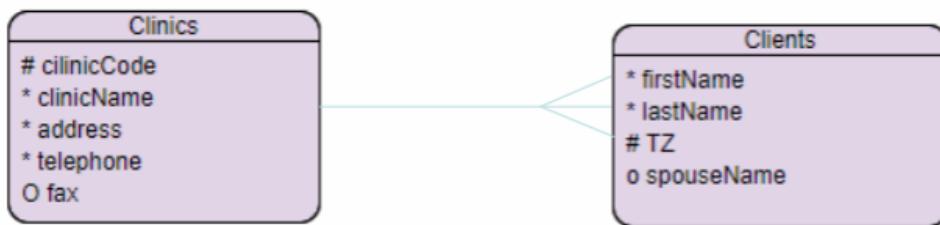
- קשר חד חד ערכי (1:1)
  - ישות אחת מתאימה לשות אחד אחרת.
  - קו רגיל

דוגמא: לכל לקוח מוגדר אמצעי תשלום ייחיד וכל אמצעי תשלום שייכת ללקוח אחד בלבד.



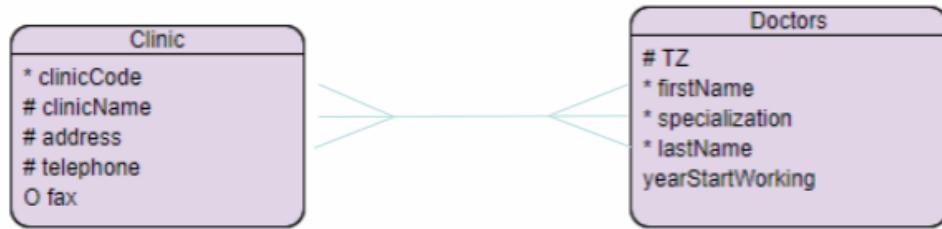
- קשר חד רב ערכי (M:1)
  - ישות אחת מתאימה לכמה ישויות בקבוצה אחרת.
  - קו רגיל עם שלושה קוויים בצדו האחד.

דוגמא: כל לקוח משוייך למרפאה אחת ובכל מרפאה יש הרבה לקוחות.



- קשר רב רב ערכי (M:N)
  - ישות אחת מתאימה לכמה ישויות בקבוצה אחרת, ושות בקבוצה האחרת מתאימה לכמה ישויות בקבוצה הראשונה.
  - קו רגיל עם שלושה קוויים בשני צדדי.

דוגמא: רופא יכול לעבוד בכמה מרפאות ובכל מרפאה עובדים הרבה רופאים.



- קרדינליות הקשר Relationship cardinality

- קרדינליות הקשר מוגדרת כמספר היחסות המינימלי והמקסימלי בקבוצת ישויות A
- הקשורות לישות אחת בקבוצת ישויות B.
- קו מקווקוו אומר שהישות לא עקרונית לישות אחרת.

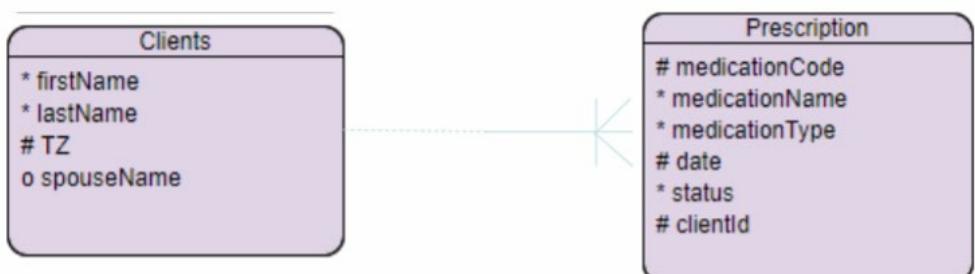
לדוגמה: רופא משפחה יכול להיות לטפל בעד 100 לקוחות



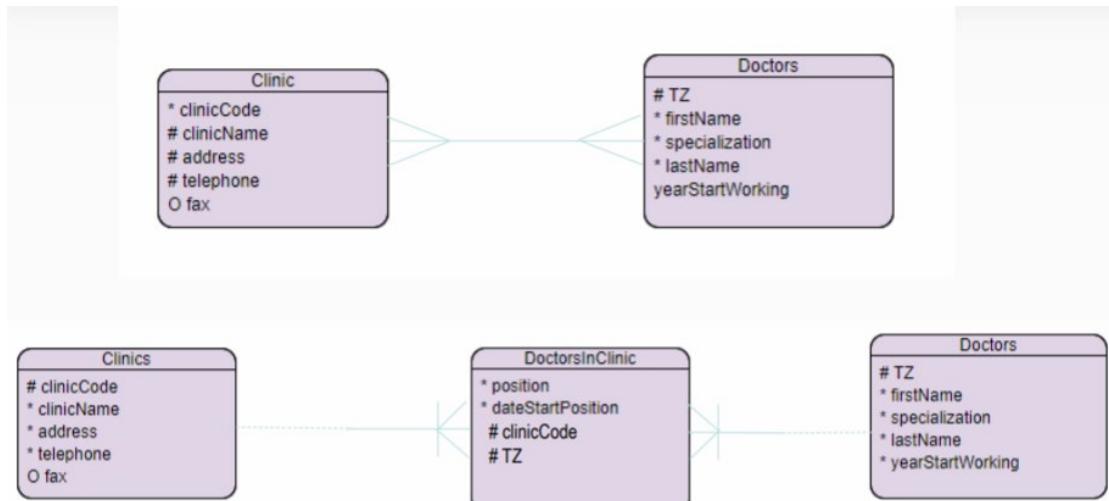
- תלות קיומית :existence dependence

- תלות קיומית בין קבוצה A לקבוצה B מוגדרת כמצב שבו קיומ ישות בקבוצת ישויות A מותנה בקיום ישות בקבוצת ישויות אחרת B.
- הקיום של הישות תלוי בקיום של הישות השנייה (קו חום)

דוגמה: לטבלה " מרשםים ללקוח" אין משמעות כשהלכו עבר את הקופה וכבר לא מבוטח שלא.



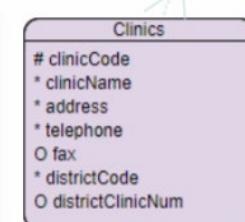
:From m:n to m:1 + n:1



קשר רקורטיבי:

- קשר רקורטיבי מוגדר כקשר המחבר בין קבוצה A לעצמה

לדוגמה: מרפאה מחוץ:



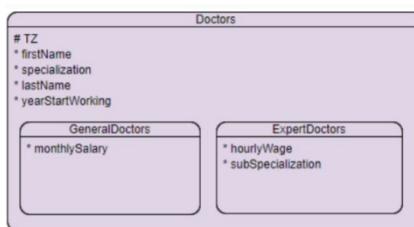
ישות בתוך ישות:

- תת ישות הינה סוג של מאפייני ישות העל

לדוגמה: כל רופא חייב להיות או מומחה.

ישות אשר יורשת את

רופא כללי או רפואי



קשרים, ישות על ותתי ישות:

- קשר של ישות העל הינו גם קשר של תת הישות.
- קשר של תת ישות מהוות קשר שלה עצמה בלבד.

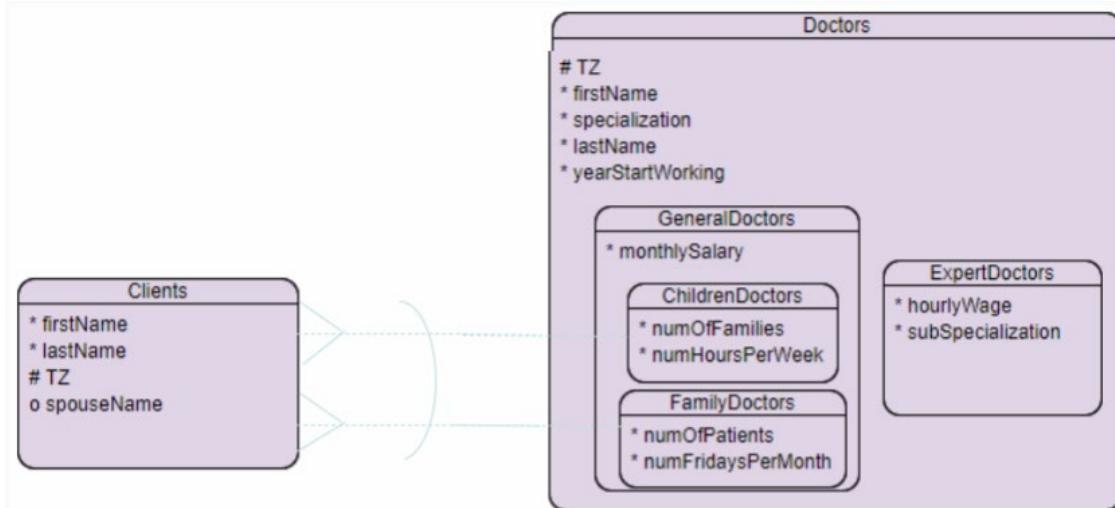
דוגמיה:



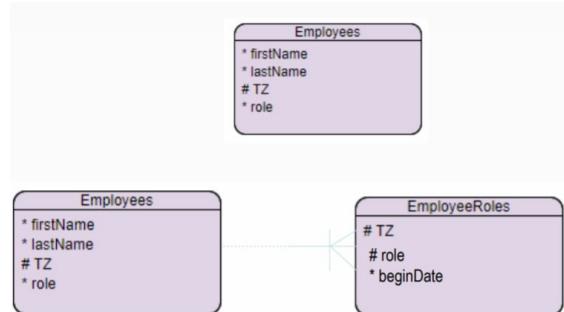
יחס בחירה בין קשרים:

- יחס בחירה בין מספר קשרים מוגדר כבחירה בקיים קשר אחד בלבד מתוך מס' אפשריים.

דוגמה:



חיי ישות לאורך זמן:



ניתן לראות שהוספנו עוד ישות כדי לשמר על היסטורית התפקידים של העובד

שלבים ביצירת מודל ER-שלון:

בהתנתק תיאור המערכת הרצויו רשיית הדרישות:

- זיהוי ישויות
- הגדרת הקשרים וסוגי הקשרים בין הישויות
- הגדרת קרדינליות (כמה חזק) הקשר
- זיהוי כל התכונות וסוג כל תכונה ותכונה
- יצירה ER ראשוני
- בדיקת רמת הנירמול של המודול ותיקונו על קבלת מודול בرمת הנירמול המבוקשת.

הרצאה 7:

UML  
ERD – תרשימים ישויות קשרים

נורמל – תהליך שמטרתו לייצר אוסף של טבלאות שאין כלולות תופעות לא רצויות הנובעות מכפילות נתונים או אנומליות.

NF1 – כל תכונה יכולה לקבל ערך אחד בלבד ללא קבוצות

NF2 – תכונה שאינה מופיעה באף אחת מהמפתחות לא יכולה להיות תלוייה בתת קבוצה ממש של מפתחות (סופר מפתח הינו מפתח שאינו בהכרח מינימלי [כלומר, יכול להכיל בתוכו מפתח מינימלי יותר])  
לדוגמא: אם {C,D} מפתח איזי התלות B  $\rightarrow$  C אינה מקיימת NF2.

NF3 – תכונה שאינה מופיעה באף אחת מהמפתחות לא יכולה להיות תלוייה בתת קבוצה שאינה סופר-MASTER (סופר מפתח הינו מפתח שאינו בהכרח מינימלי [כלומר, יכול להכיל בתוכו מפתח מינימלי יותר])  
לדוגמא: אם {C,D} סופר-MASTER איזי התלות B  $\rightarrow$  D אינה מקיימת NF3.

BCNF (3.5NF) – תכונה לא יכולה להיות תלוייה בתת קבוצה שאינה סופר-MASTER.  
לדוגמא: אם {C,D} סופר-MASTER איזי התלות C  $\rightarrow$  D אינה מקיימת NF3.5.

NF4 – ללא תלויות רב-ערךיות. כל ישות צריכה להחזיק 'עיוון' אחד בלבד.  
באופן דומה, אם לא ניתן להסיק קיום של רשותה אחרת על סמך רשומות אחרות.  
לדוגמא: אם ישן רשומות:

תק.ז	קבוצה ספורט	מחלקה לימוד
123456	ביסבול	מדעי המחשב
123456	כדורסל	הנדסת חשמל

איז אין קיימ של NF4 כי ניתן להסיק את הרשותות הבאות:

שם	מחלקה לימוד	קבוצה ספורט
123456	הנדסת חשמל	ביסבול
123456	מדעי המחשב	כדורסל

שלבים ביצירת מודל ה-ER (בהתאם לתיאור המערכת או/ו רשימת הדרישות):

- זיהוי היחסיות
- הגדרת הקשרים וסוגי הקשרים בין היחסיות
- זיהוי התכונות וסוג כל תכונה ותכונה
- יצרת ERD ראשוני
- בדיקת רמת הנורמל של המודל ותיקונו עד קבלת מודל ברמת הנורמל המבוקשת.

:FireBase  
שירות רשות היכול להחליף את כתיבת צד השרת במערכת.

- כולל בסיס נתונים (לא רלציוני [hirecii]) לאחסן מסמכים (Realtime)
- שירות hosting של נתונים וקבצים
- מערכת אוטנטיקציה (אימות) מובנת – אימייל/סיסמה או דרך google/facebook/twitter וועוד

- Firebase Analytics – מדידת אפליקציות, משתמשים, כינויים ועוד.
- המסד נתונים של הפירבייס הוא noSQL
- הנתונים שמאירים במבנה של json (key: value)
- ה-value יכול להיות מבנה של subvalue: key גוסף
- במקרה של ריבוי רשומות מאותו סוג, נדרש להשים לב שאנו יוצרים רשומות חדשות ולא דוחרים את הרשומה הקודמת

#### קינון עמוק מידי

- כל גישה לפירבייס צריכה בהפניה למיקום מסוים, המיקום כולל בתוכו את כל האlements שתחת המיקום שנבחר (כלומר, המסלול כולל)
- קימון עמוק מידי יגרום להחזקת הפניה גדולה שעלולה לפגוע ביעילות
- במקרה של צורך צורך בקינון עמוק יש להחליפו בצומת חדשה בرمה גבוהה יותר.
- ככלומר, נניח שישנם צומתים בرمה מסוימת המציגים תוכנה כלשהי (לדוגמה, נניח שברמה כלשהי ישנים ערכים של REPORTS). אז נוכל לפתח עץ חדש עבור אותה התוכנה (עץ של REPORTSvr) שנותן לגשת לכל דיווח באמצעות ה-IDI שלו כבר בرمה השנייה של העץ החדש וואז הגישה אליה תהיה עיליה יותר.

הרצאה 8

ניהול פרויקטים:

## פרויקט:

- מאמץ זמני שיש לו התחלה ועוד מוגדרים ועשוי למען מטלה מסוימת.  
מייצר מוצר ייחודי (אצלנו נתעסק במוצרים תוכנה)  
מורכב ממשימות התלוויות ביבחן  
PMI – ארגון ניהול פרויקטים משנהת 1969.

## פרויקט תוכנה: סיבות לכשלים בפרויקטים תוכנה.

- יעדים לאמציאותיים ולא ברורים
  - אומדן משאבים לא מדויק
  - דרישות מערכת שאין מוגדרות היטב
  - דיווח לKOי לגבי מצב הפרויקט
  - סיכונים לא מנווהלים
  - תקשורת לקויה בין KOוח, המפתחים והמשתמשים.
  - שימוש בטכנולוגיה לא בשלה
  - אי יכולת ניהול את מורכבות הפרויקט
  - פרקטיקות פיתוח מורשלות
  - ניהול KOוי של הפרויקט
  - פוליטיקה של בעלי עניין
  - להציג מסחריים

משולש האילוצים:



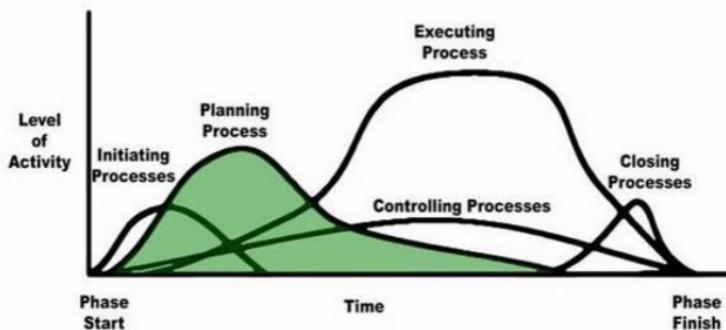
**תוכנות חשובות למנהל פרויקטים:**

- מיזמים תכנון וידע בשימוש בכלים תוכנן
  - חשיבה ביקורתית
  - ידע נרחב בנושא הפרויקט
  - יכולת ארגון
  - ניהול סיכונים
  - ניהול זמנים
  - יכולת ניהול צוות
  - מניגיות
  - ידוע לניהל משא ומתן בצורה טובה ובהרגשה נעימה ל'הצדדים
  - תקשורת טובה עם אנשים

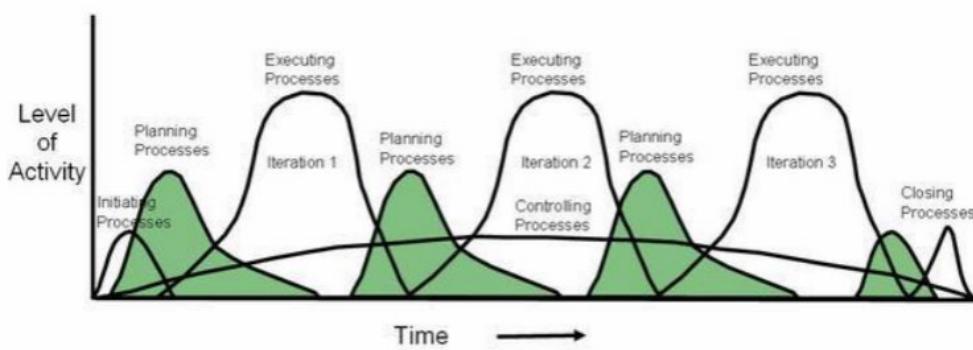
## שלבי ניהול פרויקט:

1. ייזום
2. תכנון
3. ביצוע
4. בקרה
5. סגירה

תרשים ניהול פרויקט מסורתי:



תרשים ניהול לפי שיטה אגילית:



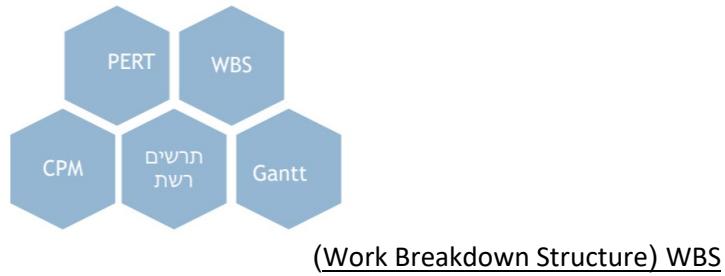
שלב הייזום:

- הכרה בצריך לביצוע: מה הרעיון \ איזה בעיה אמורה להיפתר?
- הגדרת מטרות העל של הפרויקט
- הגדרת ציפיות מלוקחות, הנהלה ויתר בעלי העניין בפרויקט
- הגדרת היקף הפרויקט
- בחירת הצוות הראשוני לביצוע הפרויקט

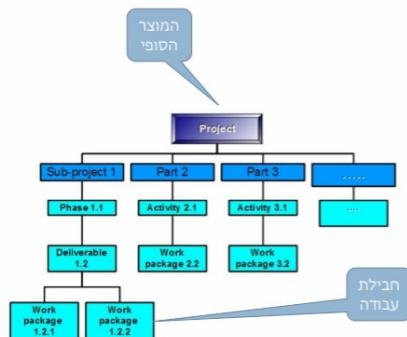
שלב התכנון:

- חלוקת מטרות העל ליעדים ניתנים למדידה
- חלוקת הפרויקט לתת-מערכות
- גישום וגיבוש צוות העבודה
- קביעת לו"ז לביצוע המשימות
- הערכת עלויות הפרויקט

נבחן כלים שונים בשלב תכנון הפרויקט: (WBS, PERT, Gantt, תרשימים רשות, CPM)



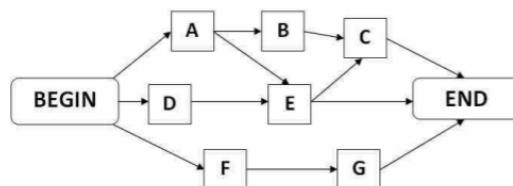
(Work Breakdown Structure) WBS



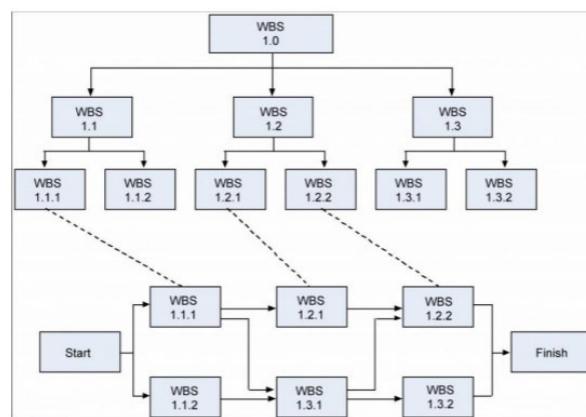
- חלוקת הפרויקט לתתי משימות הנוטנות ביחד את התמונה המלאה.
- אפשר לקבל תמונה מלאה של דרישות הפרויקט ברמת המשימות.

Project schedule network diagram (תרשימים רשות)

- כלי להצגה וניהול שרשרת המשימות בפרויקט
- תראה תלויות בין מטלות
- יכול לסייע לתכנון עיל, ארגון ובקרה הפרויקט.



WBS לתרשימים רשות:



תלות בין המשימות:

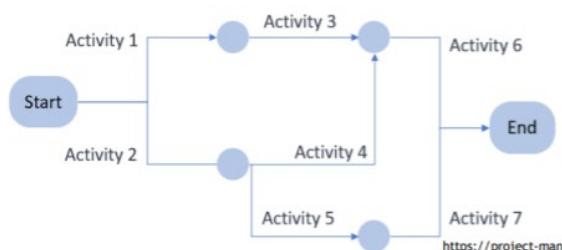


מקדמי בטיחות (הארכת זמנים):

- להאריך כל משימה (יכול להאריך את הפרויקט כולו – יכול להיות שבעליו העניין לא ירצה ללקחת את הפרויקט בשל כך)
- למתארח הארכה לפרויקט כולו
- לחלק את הפרויקט לשכבות, וכך שלב נבייא הארכה מסוימת (זאת אפשרות ביןיהם של 2 האפשרויות הראשונות)

#### תרשים PERT (Program Evaluation Review Technique)

- נוצר בשנות החמישים כדי לסייע בניהול ייצור כלי נשק בצי האמריקני
- מסיע בקביעת אומדן של עלות וזמן של משימות בפרויקט
- נקרא גם method (AOA) 'activities on arrows'



:PERT נוסחת

- תרשים PERT מטפל בנושא אומדן הזמן והעלויות
- עברו כל פעילות ניתנת למת 3 אומדנים:  
 1. אופטימי (shortest time)  
 2. פסימי (longest time)  
 3. סביר (likely time)

$$\frac{shotrest\ time + 4 \cdot likely\ time + longest\ time}{6}$$

הנוסחה:

תורנות:

- אפשר לנתח ולהעריך את הזמן, הוצאות והמשאים שדרושים לפרויקט כולו.
- נתן תמונה מלאה על הפרויקט למנהל הפרויקט ולשאר בעלי העניין
- אפשר מעקב אחריו ביצוע הפרויקט

### מציאת הנתיב הקרייטי CPM (Critical Path Method)

- הגדרה - שרשרת פעילותות הקשורות זו בזו, אשר כל שינוי בזמןם שלהם ישפיע על מועד סיום הפרויקט.
- בפרויקט בו כל המשימות תלויות זו בזו ולא מתבצעות פועלות במקביל, הפרויקט כולו נמצא בנתייב הקרייטי
- בפרויקט בו מתבצעות פועלות במקביל, ישנו מספר נתיבים, נוצרים מרוחקים (slacks) המאפשרים לדוחות פעילותות מסוימות בלי לדוחות את סיום הפרויקט
- משך הזמן של הנתיב הקרייטי הוא גם משך הזמן המינימלי של הפרויקט

התחלת וסיום של פעילות:

- **Early start** – התאריך המוקדם ביותר בו ניתן להתחיל ביחס לתלוויות.
- **Early finish** – התאריך המוקדם ביותר בו ניתן להסתיים בהתייחס לתלוויות
- **חישוב:** (משך התחילר) Early start + Duration
- **Late start** – התאריך המאוחר ביותר בו ניתן להתחיל בלי לדוחות את מועד סיום הפרויקט.
- **Late finish** – duration – duration
- **Late finish** - התאריך המאוחר ביותר בו ניתן להסתיים בלי לדוחות את מועד סיום הפרויקט.
- **חישוב:** ה-LS המינימלי מבין המשימות העוקבות (במשימה האחרונה LF שווה ל-EF)
- **Slack** – זמן שבו המטלה יכולה להמתין ללא עיכוב בפרויקט
- **חישוב:** Late start – Early start

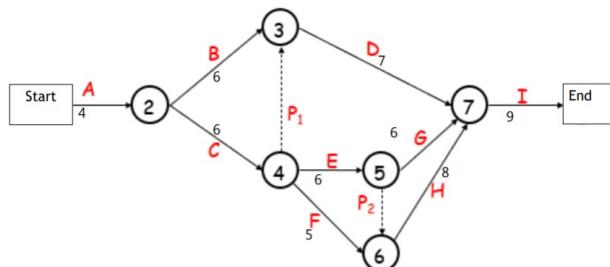
דוגמה:  
נתונה טבלה המטלות בפרויקט מסוים:

מספר	אומדן זמן			תלויות	תיאור	מטלה
	פסימי	סביר	אופטימי			
6	4	2	--		ביצוע סקר שוק	A
9	5	3	A		עיצוב איקונים	B
7	5	4	A		תכנון המערכת	C
10	6	4	B, C		עיצוב מסכים	D
7	5	4	C		קידוד מודול 1	E
8	4	3	C		קידוד מודול 2	F
8	5	3	E		קידוד מודול 3	G
10	7	5	E, F		קידוד מודול 4	H
12	9	4	D, G, H		אינטגרציה ובדיקות	I

## חישוב נסочת PERT

עיגול	זמן צפוי	אומדן זמן			תלוויות	תיאור	מטלה
		פסימי	סביר	אופטימי			
4	4.00	6	4	2	--	ביצוע סקר שוק	A
6	5.33	9	5	3	A	עיצוב איקונים	B
6	5.17	7	5	4	A	תכנן המערכת	C
7	6.33	10	6	4	B, C	עיצוב מסכים	D
6	5.17	7	5	4	C	קידוד מודול 1	E
5	4.50	8	4	3	C	קידוד מודול 2	F
6	5.17	8	5	3	E	קידוד מודול 3	G
8	7.17	10	7	5	E, F	קידוד מודול 4	H
9	8.67	12	9	4	D, G, H	אינטרגרציה ובדיקות	I

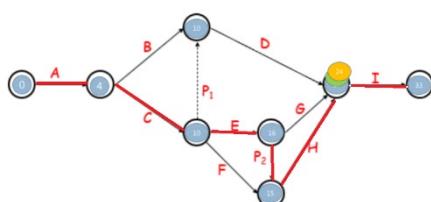
תרשים PERT



טבלת התחלת וסיום של הפעולות:

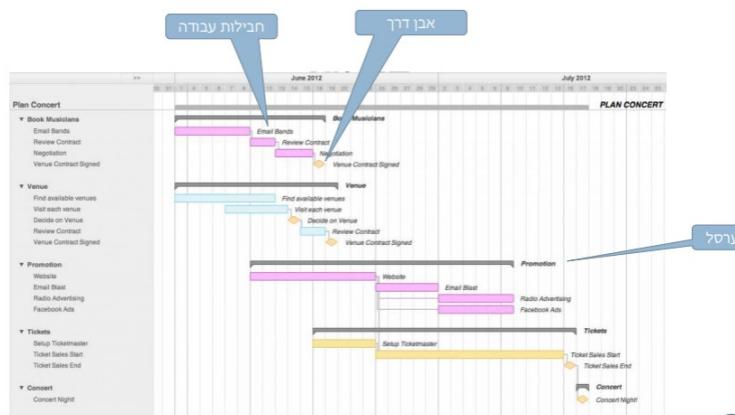
Slack	Late Finish	Late Start	Early Finish	Early Start	#	משך	תלוויות	מטלה
0	4	0	4	0	4	4	--	A
5.85	17	11	10	4	6	6	A	B
0	10	4	10	4	6	6	A	C
5.85	24	17	17	10	7	7	B, C	D
0	16	10	16	10	6	6	C	E
0.67	16	11	15	10	5	5	C	F
7.17	24	18	22	16	6	6	E	G
0	24	16	24	16	8	8	E, F	H
0	33	24	33	24	9	9	D, G, H	I

הנתיב הקritisי הוא הנתיב הארוך ביותר בכל הפרויקט.  
הנתיב הקritisי מסומן באדום.

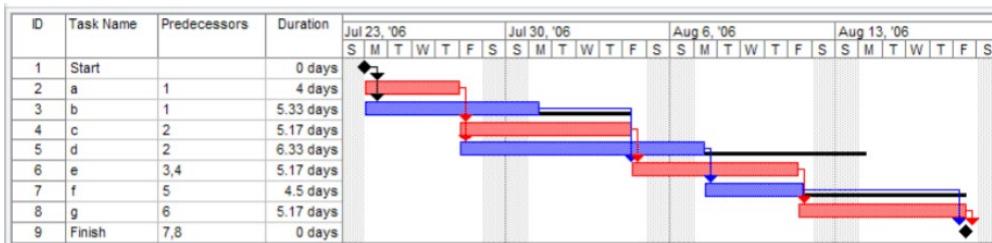


תרשים Gantt:

- השיטה הנפוצה ביותר להציג לוז'ז הפרויקט (הומצא בתחילת המאה ה-20)
- הציר האופקי מייצג את הזמן
- הציר האנכי מייצג את חבילות העבודה
- זה התרשים השימושי ביותר ע"י מנהלי פרויקטים



אבן דרך – רואים מה הסטטוס של המשימה  
חייבות עבודה – לכל דרישת על יש חבילות עבודה  
ערסל – דרישות על



שלב הביצוע:

- השלב הכי מקשור למנהל הפרויקט
- השלב שבו המנהל אחראי על:
  - סיפוק תוצר ללקוח
  - מיקוד חברי הוצאות במשימות
  - שמירה על המשאבים המוקצחים לפרויקט
  - עמידה על הוצאה לפועל של התיכנון
  - התהמה לכל חבר צוות את המשימה שמתאימה לו
  - הסבר על המשימות – לדאוג שהכל מובן אחרת להדרכה
  - יצרת קשר עם הלוקחות, חברי הוצאות והנהלה
- הביצוע תלוי במידה רבה בשלב התיכנון

שלב הבקרה:

- תהליך מתמיד של השוואה בין הביצוע לבין התכנון תוך נקיית פעולות מתונות על מנת להקטין פערים בלתי רצויים
- סוג בקרות בפרויקט:
  - בקרת תכולה
  - בקרת לו"ז
  - בקרת תקציב
  - בקרת סיכונים
  - בקרת איכות
  - בקרת שינויים

דרכים לצמצום הלו"ז:

- Re-estimation
  - בדיקה מחודשת של ההנחות, דברים שלא היו ידועים בזמןנו וכו'
- Crashing
  - הוספת יותר משאבים לנútב הקרייטי, תוך שמירה על התכולה
  - דורש הגדרה של תקציב
- Fast Tracking
  - ביצוע מטלות בנútב הקרייטי במקביל
  - מגביר סיכונים וסבירויות
  - לא אידיאלי, אבל אפשרי וimplementible לעתים
  - משולש האילוצים – להוריד באיכות או לצמצם תכולה.

ערך מזוכה:

- הביצועים הנוכחיים הם האינדיקטור הטוב ביותר לבחינת הביצועים העתידיים, ولكن באמצעות ניתוח המגמה, ניתן לחזות את עלות הפרויקט ואיחור בלו"ז בשלב מוקדם של הפרויקט.

- Method Value Earned (Earned Value) היא שיטת ערך מזוכחה (Earned Value) המשיטה הנפוצה ביותר היא שיטת ערך מזוכחה (Earned Value). בשיטה זו מתרגמים את תכולת העבודה למונחים כספיים (Costs) ביצוע מעקב אחר התקדמות הפרויקט ובדקה האם אנחנו מקדים או מאחרים בלו"ז של הפרויקט והאם אנחנו במסגרת התקציב או חורגים ממנו.

כיצד?

- בשינת הערך המזוכה תמיד נעשית סביר נקודת בקרה ספציפית, נקבעת נקודת חיתוך בזמן (למשל: היום), אשר מהו נקודת ייחוס איחוד לכל הפרמטרים השונים.
- השיטה למעשה תקופה רק לאחר שהפרויקט החל.
- כל החישובים נעשים ברמת הפעולות ונוסכים באופן אגרטיבי לרמת הפרויקט כולו.

נתונים ומשתנים:

- נתונים שמנהל הפרויקט מגדר וمعدכן לכל פעילות:
  - BAC – Completion At Budget – סך הכל הוצאות המתוכננת לפעילויות
  - AC – Cost Actual – סך הכל הוצאות שהצטברו בפועל (עד ליום הבקרה)
- פרמטרים המחשבים בהתאם לדיווח ההתקדמות בפועל של כל פעולה:

סימן	משמעות	איך מחושב
PV	Planned Value	% זמן הביצוע בתכנון * BAC
EV	Earned Value	% זמן הביצוע בפועל * BAC
CV	Cost Variance	EV – AC
SV	Schedule Variance	EV – PV
CPI	Cost Performance Index	$\frac{EV}{AC}$
SPI	Schedule Performance Index	$\frac{EV}{PV}$
EAC	Estimated At Completion	$\frac{BAC}{CPI}$
ETC	Estimated To Complete	EAC – AC
VAC	Variance At Completion	BAC – EAC

דוגמה:

הפרויקט – פיתוח פרויקט בעלות 4 משימות. בכל יום מבוצעת משימה אחת והוצאות המתוכננת ליום המשימה הנקה \$1000. התכנון הוא לפתח משימה אחריה משימה.

היום הוא סוף היום השלישי.

סטודנטים:

יום 1: המשימה הושלמה. הוצאות היו \$1000.

יום 2: הייתה מעת גלישה לבוקר היום השלישי. הוצאות למשימה היו \$1200.

יום 3: התחלו בפיתוח בשעות הבוקר המאוחרות. הושלמה ורק מלחצית המשימה. הוצאות היו \$600.

$$\begin{aligned} \text{BAC} &= 1000 + 1000 + 1000 + 1000 = 4000 \\ \text{AC} &= 1000 + 1200 + 600 = 2800 \end{aligned}$$

$$\begin{aligned} \text{PV} &= 1000 + 1000 + 1000 = 3000 \\ \text{EV} &= 1000 + 1000 + 500 = 2500 \\ \text{CV} &= 2500 - 2800 = -300 \\ \text{SV} &= 2500 - 3000 = -500 \end{aligned}$$

$$\begin{aligned} \text{CPI} &= 2500 / 2800 = 0.893 \\ \text{SPI} &= 2500 / 3000 = 0.833 \\ \text{EAC} &= 4000 / 0.893 = 4479 \\ \text{ETC} &= 4479 - 2800 = 1679 \\ \text{VAC} &= 4000 - 4479 = -479 \end{aligned}$$

שלב הסגירה:

- שלב זה קורה אחרי שמספקים ללקוח מוצר (או תוסף וכו')
- בשלב זה עושים ניתוח על התהיליך שעברו חברי הצעות:
  - מה עבד?
  - מה לא עבד? ולמה?
  - איך המנהל יכול לשפר
- כדי ליצור צוות וטהיליך יותר טוב להבא.

### הרצאה 9:

עיצוב (תוכן) מבנה תוכנה  
ארכיטקטורת מערכת מול ארכיטקטורת תוכנה

ארכיטקטורת מערכת – מודל קונספטואלי (=אפיוון הרעיון המרכזי) המתאר מבנה והתנהגות אוסף מרכיבים ונתת מערכות הכוללות אפליקציות, מרכזי רשות, חומרה, חיישנים ואלמנטים נוספים. ההסתכלות היא על כל מרכזי המערכת.

ארכיטקטורת תוכנה – תהליך הגדרת מבנה ברמה גבוהה לתוכנית, הכלל הגדרת מטרות העיצוב, מודולים מרכזיים ומנגנוניימוש תכונות רצויות, בדגש על המרכיבים "הנראים" והאינטראקטיבית ביניהם. תוכנות אלו כוללות:

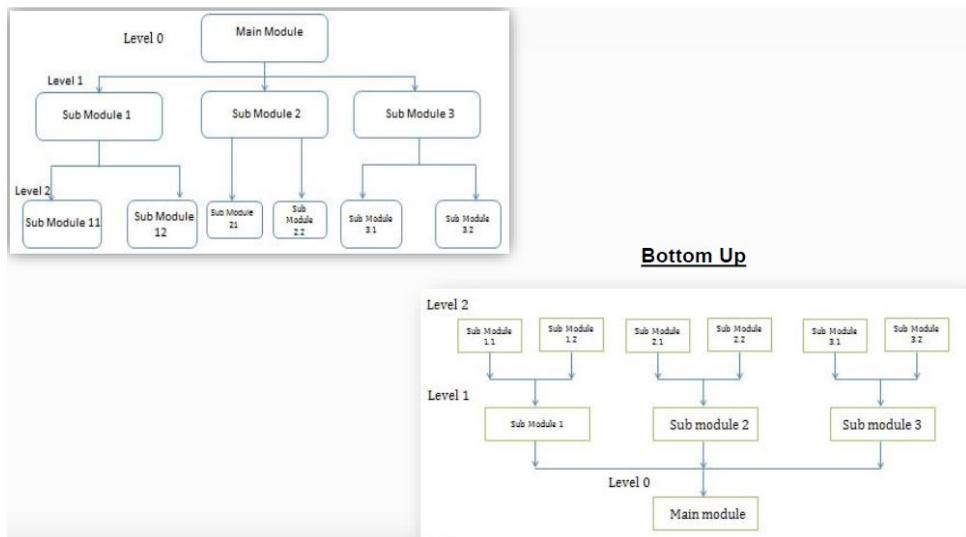
- Scalability (מדרגיות)
- Security (בטחה)
- Modularity (תוכנית המחלקת למודולים היא מערכת מודולרית)
- Reusability (שימוש חוזר)
- Extensibility (יכולת הרחבה)
- Maintainability (תחזוקה)

הסיבות לחשיבות עיצוב התוכנה:

- התמודדות עם מורכבות
- אבסטרקציה: התעלמות מפרטים והטמקדות "בתמונה הגדולה"
- דה-קומפוזיציה: פרוק הבעיה לאויסף תת בעיות בלתי תלויות (ההפק מקומפוזיציה)
- (עקרון הקומפוזיציה אומר שאם רוצים לתת לאובייקט את התכונות של אובייקטים אחרים אזי עדיף להשתמש בהכללה במקום בהורשה)
- מודולציה: הגדרת מבנים יציבים לאורך זמן
- פרויקציה: התמקדות בזווית ראייה אחת כל פעם (View) בהתייחס לתת הבעיה הנבחנת
- התמודדות עם מחזור חי תוכנה ותנאי סביבת פיתוח
- שינויים תכופים
- אורך של התחזוקה
- מבנה "גוף" מאפשר אדפטציה מהירה לשינויים

רמת עיצוב:

- רמת עיצוב גבוהה: הגדרת מודולים מרכזיים ברמת גראנוולציה (רמת פרטיים) נכונה. (מכונה גם עיצוב ארכיטקטואלי)
- רמת עיצוב נמוכה: הגדרת מודולים ותוכנם ברמת גראנוולציה (רמת פרטיים) גבוהה. (מכונה גם עיצוב פרטני)

**אסטרטגיית עיצוב:**

**שלבים כלליים בארכיטקטורת תוכנה:**

1. צורה כללית ומבנה כללי של יישומי התוכנה
2. הארכיטקטורה המווחדת (בהתאם) למטרה של יישום התוכנה
3. המודולים והקשרים הפנימיים שלהם

**סימפטומים של עיצוב נרקב:**

- קשיות (קושי לשינוי)
- שבירות (לא עמיד בשינוי)
- אי תחזזה (אי יכולת שינוי)
- צמיגות (=דביבות. כלומר, קשר חזק מדי)

ההידדרות מתחילה כאשר יש שינוי בדרישות (שינויים שמכניסים תלות חדשה ולא מתוכננת) שנו מושג של firewall dependency (חומר אש של תלילות) שם ישנו קשר מחייב בין רכיבים אחד לאניין לגעת באחד מבלי לשנות את השינוי

**עקרון SoC ומודולציה**

- Seperation of concerns
- עקרון מנחה בעיצוב תוכנה הדוגל בהפרדה טיפול בנושאים ורכיבים ביחידות נפרדות.
- Module: יחידה נפרדת בתוכנה (פיזית או לוגית) המופרدة מאחרות בצורה "נכונה"
- מודולציה נכונה כוללת:
- לכידות גבוהה (High Cohesion): מודולים ממוקדים בטיפול בנושא ספציפי
- צימוד רופף (Loose Coupling): מודולים תלויים זה בזה באופן מינימלי
- עיצוב תוכנה ↔ הפרדה מרכיבי הפתרון למודולים וניהול התלותות ביניהם

**גבול בקרת ישות:**  
כ舍דררים על מודולציה ישנים 3 סוגים ישות

1. Entity – ישות כללית יותר כמו סטודנט/מרצה
2. Control – ישות שהיא "דווגת" לפקודות
3. Boundary – רכיב/מודול שהוא בעצם מקשר בין הישות החיצונית למערכת

**כטיבת קוד "נכון":**  
 עקרון ה-s<sup>imple</sup>s<sup>tupid</sup> (Keep it simple, Stupid). הביטוי הוטבע ע"י קל' ג'ונסון (מהנדס מטוסים).  
 העקרון דוגל בימוש פשוט מבלי לסביר.

**שמות משתנים ומחלקות:**

- שמות משמעותיים. לדוגמה, wagesPerHour במקום whph.
- לשמר על עיקריות בשיטת מתן השמות.
- למשתנים זמינים (כמו מונה הלולאה) כדי לקרוא בשם קצר (כמו j).

**dagshim במתן שמות:**  
**שמות משתנים קבועים:** שימוש באאותיות גדולות בלבד  
**מחלקות:** שם משמעותי המתחיל באות גדולה לכל תחילה של מילה  
**פונקציות:** שם משמעתי המתחיל באות קטנה ואות גדולה לעל תחילת של מילה פנימית

**Dagshim בכתיבת העורות:**

- הקוד מסביר את עצמו, העורות נכתבות בשביל להסביר "למה".
- לא להשאיר קוד ישן בהערות
- בראש המסמך יש להוסיף הערה מי כתב את הקוד, מתי ומה הוא אמור לעשות

**:doc comments Java**

כל שמייע יחיד עם ה-JDK

אפשר לנשאלו ליצור תיעוד של הקוד שלנו בפורמט של HTML  
 מותבים את העורות בתוך הטוווח שמתחליל עם \*\* ומסתיים עם \*  
 משתמשים בתגים של HTML (כמו <k>, <H1>)  
 ישנו מילימ שומרות שאפשר להשתמש בהם בשביל לתאר דברים מסוימים:

@since  
 @version

....

שימושי לתוכנתים אחרים שקוראים את הקוד, או לבניין עניין אחרים.

**הפעלת doc comments Java:**

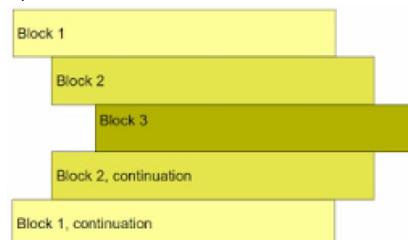
בתפריט הראשי בסביבת הפיתוח:

Generate Java Doc < Tools

ואז יוצר לנו מסמך HTML עם כל התיעוד.

**הזהות:**

גם אם זהה היא לא חובה בהגדירה של השפה שבה אנחנו כותבים, כדאי להקפיד עליה



ש שיטות זהה רבות, לכן, יש לשמר על עיקריות.

**:Code Review**

זהה' בדיקה שיטתייה על קטע הקוד אשר נכתב לתוכנית או למערכת בצד' למצוא ולתקן בעיות שנמצאות בקוד עוד בשלבי הפיתוח המוקדמים. הבדיקה נעשית ע"י עמיתים ולא ע"י המתכנת עצמה. מטרות:

- מציאת בעיות ותיקונם
- שיפור תהליכי ביצוע הפיתוח
- שיפור יעילות ומורכבות הקוד
- עוזר לשיתוף מידע בתוך הוצאות

**Code Review** מתרחשת: קוד מת, בעיות ייעילות ומורכבות, שימוש חוזר בקוד, דליפת זיכרון וזריגת מידע יתרונות של **Code Review**:

- מעבר על הקוד ומציאת בעיות בשלבים מוקדמים
- מציאת בעיות שלא ימצאו ע"י בדיקות רגילים (לדוגמה, אי שימוש במשתנים)
- צמצום הביעות כתוצאה מראייה נוספת נסופת אשר משפיעה על איכות המערכת
- אזהרה מוקדמת לגבי היבטים וחלקים חדשניים בקוד בגיןן לאמוד אותם רק ע"י מדידה או מורכבות
- שיפור רמת התחזקה של הקוד לשינויים עתידיים

**:Code Review של צ'אך ליסט**

- Code formatting (עיצוב קוד)
- Architecture (ארქיטקטורה)
- Coding best practices (שיטות עבודה מומלצות)
- Non Functional requirements (דרישות לא פונקציונליות)
- Object Oriented Principles (עקרונות מונחה עצמים)

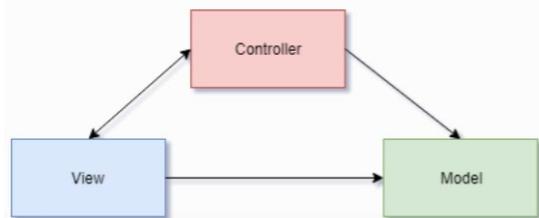
**:עקרונות SOLID**

- Single Responsibility Principle (למחלקה צריכה להיות אחריות אחת ויחידה)
- Open Close Principle (מחלקה צריכה להיות סגורה לשינויים ופתוחה להרחבה [למנוע שינויים ולאפשר ירשות])
- Liskov Substitution Principle (אובייקטים בתוכנה יכולים להיות מוחלפים על ידי מחלקות ירשות ולא שינוי תפקוד התוכנה בכללותה)
- Interface Segregation Principle (מחלקות יהיו מושגים שונים אשר יותאמו לפי צורכי המשתמשים)
- Dependency Inversion Principle (מיושרים יהיו תלויים במושגים ובսטרuktיות ולא במימושים פנימיים)

## הרצאה 10

עיצוב (תקן) מבנה תוכנה

(תבניות ארכיטקטוניות נבחרות)  
(Model-View-Controller) MVC



- במקור נועד לפיתוח line

התאמה לפיתוח Web

התאמה לפיתוח אנדרואיד

MVC באנדרואיד:

- MODEL – מחלקה נפרדת שלא ירושת משום מחלקה של אנדרואיד Activities (and fragments) – מחלקות של CONTROLLER, VIEW
- 

יתרונות חסריונות:

+ עיצוב מודולרי לאפליקציה

- תלות בין שכבות שונות (View + controller) View תלויים במודול

- הcontroller נהיה מסובך ומורכב במשך הזמן

(Model-View-Presenter) MVP

- התאמה לפיתוח Winforms

התאמה לפיתוח אנדרואיד

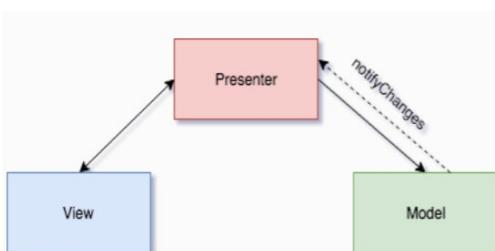
מימוש MVP באנדרואיד:

Model: מחלקה נפרדת שלא ירושת משום

מחלקה של אנדרואיד

View: Activities (and fragments)

Interface :Presenter



יתרונות חסריונות:

+ אין קשר בין רכיבי האנדרואיד

+ קל לתחזק ולבדוק: בגלל ההפרדה בין השכבות

-Presenter יש נטייה להתרחב

הבדלים בין MVC ל-MVP:

• דפוί MVC:

Controller מבודס על התנהוגיות וניתן לשתף אותו בין Views.  
יכול להיות אחראי לקביעת איזו View להציג.

• דפוί MVP:

View מקשרת בצורה רופפת יותר ל-Model.  
Presenter אחראי לחבר את Model ל-View.

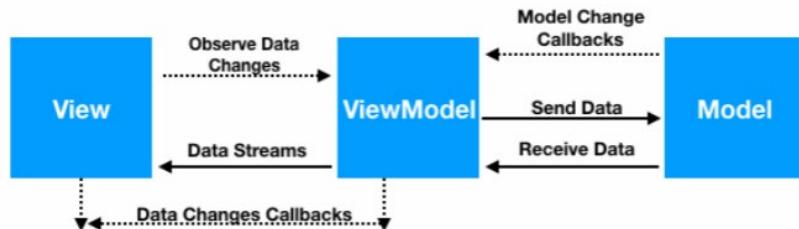
כל יותר לבודיקת יחידה מכיוון שהאינטראקטיה עם View היא דרך ממשך.  
בדרך כלל הצגת ה-View ל-Presenter אחד. לתצוגות מורכבות עשויות להיות  
כמה Presenters.

: (Model-View-ViewModel) MVVM

התאמת לפיתוח Web Client Side

- התאמת לפיתוח WPF

- התאמת לפיתוח אנדרואיד



:Event Bus

- מתאימה לפיתוח אנדרואיד

Event source, Event Listener, Channel & Event Bus

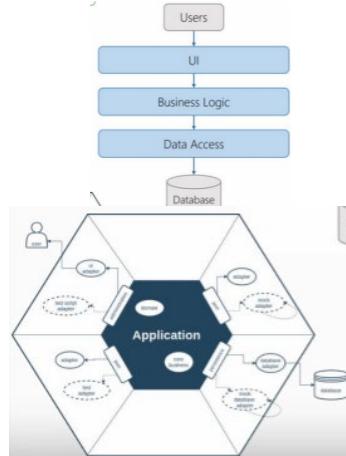
- מחברת בין & Event Bus

:Multi Layer

התאמת למערכות שלחניות ו애�ליקציות Web

:Ports and Adapters

Create your application to work without an UI or database, to which you can run automated regression tests, implement when the database is not available and connect applications without involving them. (Alistair Cockburn)



### Low Level Design

גישה מונחת עצמים:

- ניתוח מונחת עצמים – OOA

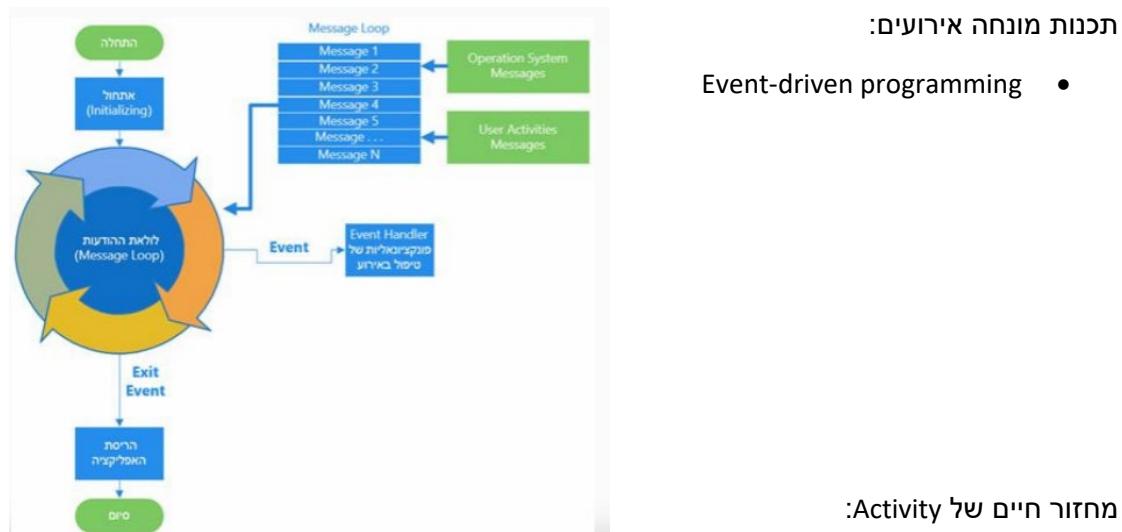
- איתור הדרישות הפונקציונליות והלא פונקציונליות של המערכת : סט הקלטים , ההתנהוגיות והפלטים הנדרשים לミושן לצד החלטות ואילוצים נילויים.

- עיצוב מונחת עצמים – OOD

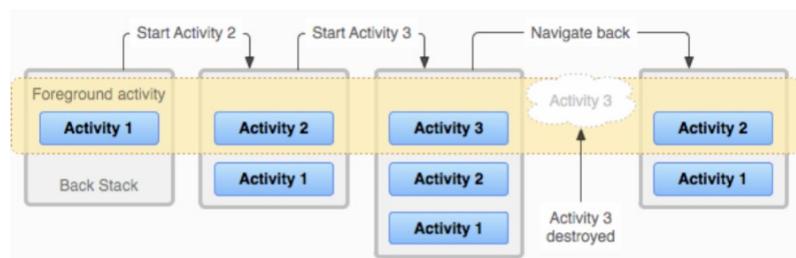
- עיצוב אפליקציה ועיצוב מערכת , שימוש בתבניות ארכיטקטוניות ותבניות תכנון .

- תכונות מונחת עצמים – OOP

- פרקטיקות מימוש המעודדות יצרת קוד מונחת עצמים "טוב" : ייצור גרסאות, קוד אבסטרקטיות, עקרון ותשתיות IOC, Refactoring, Program to Interface, מתmesh, TDD וכו'.



מחסנית:



**הרצאה 11 חלק א:**

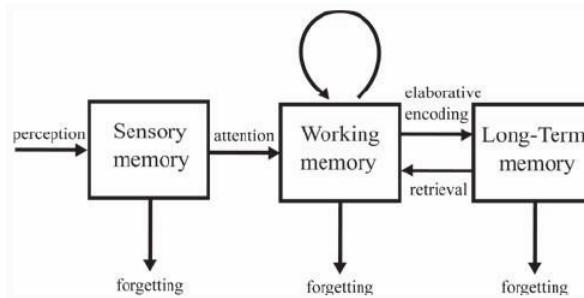
מבוא לעיצוב וחווית משתמש:

תחום בפסיכולוגיה העוסק בהבנת תהליכי מנטליים

- תהליכי מנטליים: חשיבה, קבלת החלטות, פתרון בעיות, שפה, קשב וזיכרון
- הגדרת ייחידת ידע (קונספט, אב טיפוס, סכימה)
- כיצד בני אדם לומדים – שיטות, מאפיינים ומגבלות
- התייחסות דומה למוח: בני אדם קולטים, מעבדים ומאחסנים אינפורמציה
- גישה הפוכה לביהוריزم [bihorizm] היא גישה פילוסופית-מדעית, העומדת ביסודן מחקר ההתנהגות ומדעי ההתנהגות. הגישה טוענת שניתן לחקור באופן ניסוי ומדעי את התנהגות היצורים החיים ואת התנהגות האדם בתוכם]

:Short-term memory

- היזכרון בו נשמרת אינפורמציה במהלך שימוש מודע
- זיכרון זמני
- חלקו מוקדש לאינפורמציה ויזואלית
- מוגבל בampie
- שימושיות אפשריות:
- אין להציג מידע בסיס אחד: פיצול או יצירת היררכיה
- יצירה תצוגה עקבית



:Visual Encoding

987349790275647902894728624092406037070570279072  
803208029007302501270237008374082078720272007083  
247802602703793775709707377970667462097094702780  
927979709723097230979592750927279798734972608027

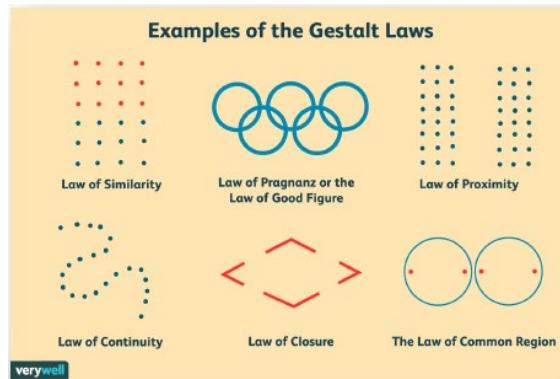
987349790275647902894728624092406037070570279072  
803208029007302501270237008374082078720272007083  
247802602703793775709707377970667462097094702780  
927979709723097230979592750927279798734972608027

כלומר, בצורה ויזואלית ניתן להבליט דברים מסוימים על מנת לשומר את תשומת לב המשתמש

פסיכולוגית גשלט:

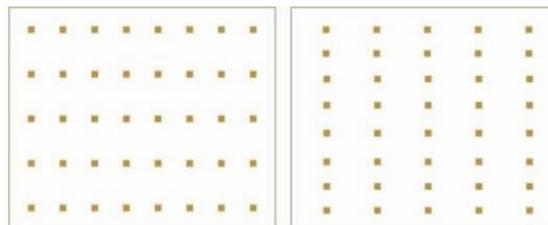
- גשלט=תבנית
- האופן שבו אנו תופסים תבניות וצורה, הדרך שבה אנו מארגנים את שמו רואים
- תובנות שימושיות לעיצוב תצוגה, בפרט לצירוף קשר בין פרטי מידע הפרדה בין פרטי מידע או הבלתי
- מציעה מספר עקרונות לעיצוב ויזואלי ובניהם:
  - Law of Proximity ○
  - Law of Closure ○

- Law of Similarity ○
- Law of Continuity ○
- Law of good figure ○
- Law of Common Region ○
- Law of Connectedness ○



נრחיב על העקרונות:  
:Law of Proximity

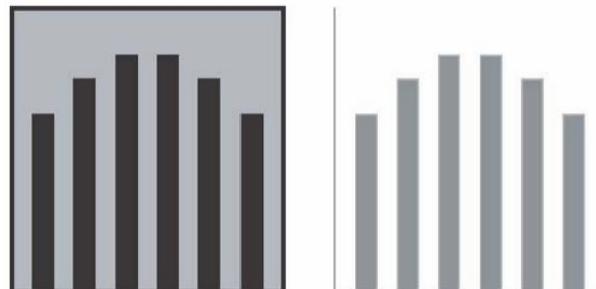
דברים שקרובים זה לזה נראהים קשורים יותר לדברים שמרוחקים זה מזה.



:Law of Closure

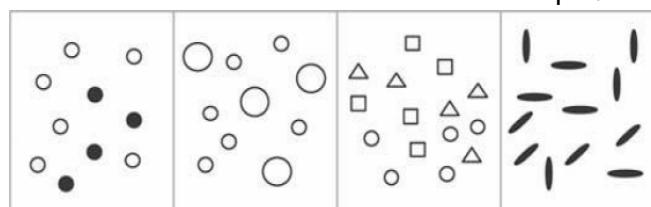
אלמנטים נתפסים כשייכים לאותה קבוצה אם נראה שהם משלימים ישות כלשהי (המוח שלנו לרוב מתעלם ממידע סותר וממלא פערים במידע)

לדוגמא, אם אנו מסמנים קבוצה מסוימת אז היא נראהית כישות אחת



:Law of Similarity

דברים דומים נוטים להופיע מקובצים יחד



:Law of Continuity

נקודות המוחוברות בקווים ישרים או מתעקלים נראות באופן העוקב אחר הנטייה החלק ביוטר כלומר, העין נוטה לראות יותר את הצורה החלקה (ישרה) יותר



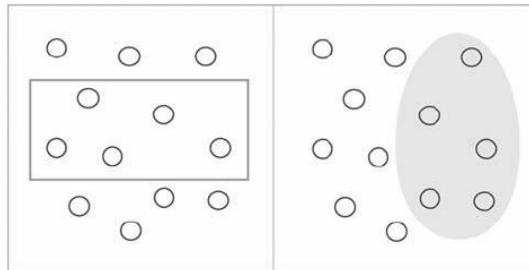
#### :Law of good figure

כאשר מוצגים עם קבוצה של אובייקטים מעורפלים או מורכבים, הם יתפרשו כך שייראו פשוטים ככל האפשר  
לדוגמה, העין נוטה לראות גביע/2 פרצופים בדוגמה הבאה:



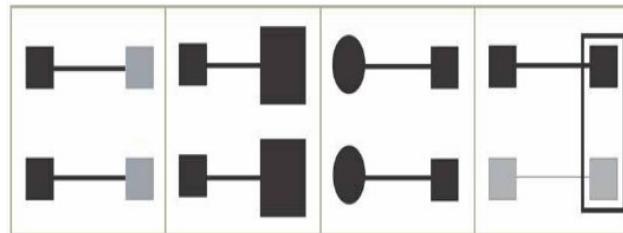
#### :Law of Common Region

אלמנטים הממוקמים באותו אזור סגור נטפסים כשייכים לאותה קבוצה



#### :Law of Connectedness

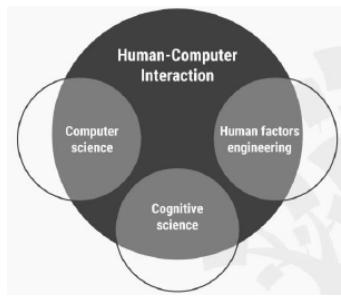
אלמנטים המוחברים זה לזה באמצעות צבעים/קוים/מסגרות, או צורות אחרות נתפסים כיחידה אחת בהשוואה לאלמנטים אחרים שאינם מקושרים באותו אופן.



#### **הנדסת UI: תחומיים ומוסגים:**

#### (HCI) Human Computer Interaction: (אינטראקציה עם מחשב אנושי)

אינטראקציית מחשב אנושית מתאפיינת בחקר האופן שבו בני אדם ומחשבים מתקשרים מנקודת VIH יכול להיות חוקר, מעצב, פסיכולוג או כל מי שעשי להתמקד באינטראקציה עם מחשב אנושי כחלק מהעבודה או הלימודים שלהם.

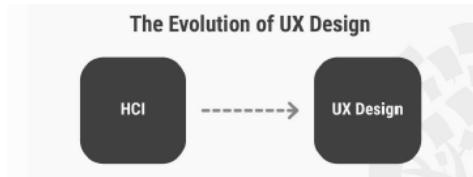


### (גורמים אנושיים: Human Factors)

תחום רב בתחום המשלב תרומות מפסיכולוגיה, הנדסה, עיצוב תעשייתי, עיצוב גרפי, סטטיסטיקה, חקר תפעול ואנתרופומטריה הינה מדעית של תוכנות היכולת האנושית

### (UX) (חוויית משתמש: User Experience)

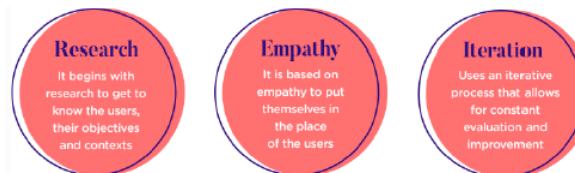
חווייה הכוללת שיש למשתמש עם מוצר



### (עיצוב ממוקד משתמש: User Centered Design)

מחלקה של מתודולוגיות שהן החלטות עיצוב מבוססות על מודל משתמש מוחשי כלשהו.  
עקרונות:

- אין טעם לעצב GUI ללא הבנה ברורה של פרופיל המשתמש המועד
- מעצבים צריכים לצאת בדרך ליצור מוצר מוצרי אשר משקף את צרכי המשתמש והעדפות
- אל תעצב עבורך, עצב עבור המשתמשים שלך
- UCD אינו ליניארי. צור מספר גרסאות של אותו מסך, בדוק עם משתמשים אמיתיים



### (שימוש: Usability)

היכולת של סוג מסוים של משתמש להיות מסוגל לבצע ביעילות משימה באמצעות מוצר

- חלק מרכזי בעיצוב UX
- נמדד בדרך כלל באמצעות בדיקות
- מספר נבדקי המבחן משקף את סוג המשתמש שישתמש באפליקציה
- מדידות בדיקות:
  - כמה מבצעים בהצלחה משימה.
  - בממוצע, כמה מהר הם משלימים את המשימה הרצו.
  - בממוצע כמה שגיאות משתמש נעשות תוך כדי ניסיון להשלים את המשימה.

### (ארQUITקטורת מידע ועיצוב אינטראקטיבי: Information Architecture & Interaction Design)

אלו שני היבטים נוספים של UX.

- עיצוב אינטראקטיביה: הבחירות הספציפיות של אינטראקטיביות משתמש שאנו חנו עושים כדי לאפשר למשתמשים לעמוד ביעדים שלהם בתוכנה.
- ארכיטקטורת מידע: מבנה מידע כך שייהי הכי טוב עבור הדרישה של משתמשי המידע שלהם.
- • השימוש הבאים נחברים מקבילים:
  - מעצב אינטראקטיבית מתמקדים באופן שבו אנשים משתמשים במחשבים על מנת להגשים עבודה
  - אדריכלי מידע מתמקדים באופן שבו אנשים מננים מידע לתמיכה במטרות

#### עיצוב חזותי (UI ו- UX)

עיצוב המראה החזותי של תוכנה או מוצר מסוים מinherits אחרים

- עיצוב משקל משתמש עוסק במשטח ובתחושים הכלליים של עיצוב.
- עיצוב חזותי הוא היבט UX Central
- עיצוב חזותי מתמקד יותר באסתטיקה
- מעצבים חזותיים לרוב איןיהם מעצבים מוקדי משתמש

הכר את המשתמשים:

- שחקן ומטרה: לעיתים קרובות שם תפקיד או השם הנפוץ לשוג המستخدم במערכת
- תפקיד משתמש: שם קוצר המתאר משתמש בمرדף אחר מטרה. משתמשים שונים
- תפקידים כהמורות שלהם משתנות
- פרופיל משתמש: מידע מסכם על סוג המשתמשים שממלאים תפקיד
- פורסונה: בחירת מאפיינים ספציפיים של אדם והרכבתם לתיאור ארכיטיפי של אותו אדם יוצר יעד עיצובי חזק

הערכת שימושיות:

- השימושיות נמדדת באמצעות בדיקת משתמשים המוצבת מול מערכת פוטנציאלית.
- מתרבשים לבצע משימות ספציפיות ללא הדראה או הדראה.
- בדיקות שימושיות טיפוסיות מודדות:
  - תדריות השלמת משימה
  - זמן השלמת משימה
  - שגיאות או שלבים שגויים
- אנשי מקצוע יכולים לתת את ההערכה הסובייקטיבית שלהם לגבי השימושיות, אבל אתה לא באמת יכול להיות בטוח עד שאתה בוחן זאת

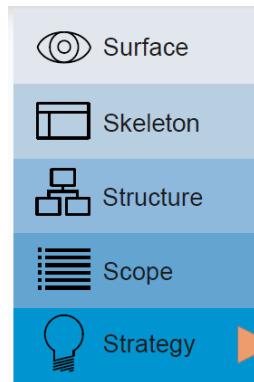
עיצוב ויזואלי חזק:

- השימוש בニアודיות, חזקה, ישור, קרבה וטכניקות מוצעות אחרות
- ליצור חזותיות אסתטית. למה? בגלל היבטים רגשיים

#### עיצוב שיפוט: (שליטה היבטים)

- קרבאים: מהי ההשפעה או המראה הראשוני של המוצר?
- התנהגותית: איך האובייקט מרגיש בשימוש?
- רפלקטיבי: על מה האובייקט גורם לך לחשוב? מה זה אומר על הבעלים שלו?

#### חוויות משתמש – מודל השכבות

:Strategy

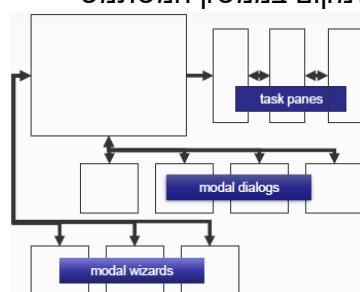
שייקול של אסטרטגיות: מטרות עסק, סוגים משתמשים, תנאים שימוש (משרד, סלולרי) וכו'

:Scope

כטיבת שימוש משתמש (הזמן מספרים, הזמן טקסט וכו')

:Structure

יצירת מבנה המתאר ניוט מקום למקום במרחב המשתמש

:Skeleton

יצירת שלד מתאר את פרישת המסר ותאים פונקציונליים במסך

:Surface

שכבות פנוי השטח מתארת היבטי עיצוב חזותי מוגמרים

כלומר, התייחסות למקור המוגמר והחזות שלן

## הרצאה 11 חלק ב

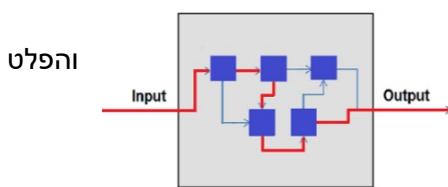
בדיקות (testing)  
הגדירה:

- בדיקות הן מכלול תהליכי, שיטות, פעילויות וכליים המלווים את פיתוח המערכת והמצרים לאורך מחזור החיים מתוכר כוונה לבצע אימות והוכחת תקיפות Verification & Validation (Verification & Validation) (להתאמת התוצרים לדרישות הלוקח, למפרטים הטכניים, לתקנים ולנהלים מחיבים).

Validation – האם אנחנו מפתחים נכון את המוצר  
Verification – האם אנחנו מפתחים את המוצר שנדרשנו לפתח  
למה לבדוק?

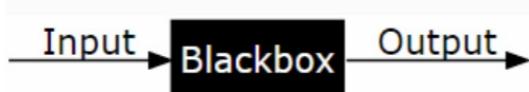
- לבדוק שהתוכנה עשויה מה שהיא אמורה לעשות (=דרישות פונקציונליות)
- לבדוק שהיא עשויה את זה עם כמה שפחות שגיאות (איכות)
- לבדוק שהיא עשויה את זה בצורה טובה מבחינת עמידה בדרישות ביצועים, ובעומסים ונפחים (=דרישות לא פונקציונליות)
- מטרת עקיפה: לאסוף רישום של שגיאות תוכנה לצורך מניעת שגיאות עתידות (פעולות מוגנות או פעולות מתקנות)

\*\* כולם בודקים: תוכנית, צוות הבדיקה, הלוקח.  
שיטות לבדיקה תוכנה:



- בדיקת קופסה לבנה:  
בדיקות של המימוש בנוסף לבדיקה הקלט

- בדיקת קופסה שחורה:



- בדיקות קופסה אפורות:  
אתה מכיר את הקוד כמו בקופסה אבל מנצל את זה בהשכלה לשימוש בקופסה כמו בקופסה שחורה.  
לדוגמה: לשנות את המסדר נתונים  
לבודק פلت מסויים

### (סוגי בדיקות נבחרים): Selected Types of Testing

- בדיקת יחידות  
• מבחן האינטגרציה
  - קח רכיבים שנבדקו ביחידת ובניו מבנה תוכנית שהכתב על ידי התכנון. בדיקת אינטגרציה היא בדיקה שבה משולבת קבוצת רכיבים לייצור פلت.
  - בדיקת קופסה שחורה לאיומות: מה הפלט?
  - בדיקת תיבת לבנה לאיומות: איך מושגים את הפלט?
  - מבחן רגסיבית
    - כל פעם שמתווסף מודול חדש המוביל לשינויים בתוכנית. סוג זה של בדיקות מודדא של רכיב עובד כמו שצරיך גם לאחר הוספה רכיבים לתוכנית השלמה

- בדיקת מערכת
- מבחן מאיץ
- מבחן ביצועים
- מבחן קבלה

**ידי (Manual):** תוכניות בדיקה, מקרי בדיקה, תרחישים קיירה או בדיקת תוכנה כדי להבטיח את שלמות הבדיקות

**אוטומטי (Automated):** כתיבת סקריפטים באמצעות תוכנה אחרת לבדיקת המוצר Unit Testing (בדיקה יחידה)

- סוג בדיקת תוכנה שבה נבדקות יחידות או רכיבים בודדים של תוכנה
- המטרה היא לאמת שכל יחידה של קוד התוכנה פועלת כמצופה
- בדיקת יחידה נעשית במהלך הפיתוח (שלב הקידוד) של אפליקציה על ידי מפתחים, ובכל זאת בעולם המשיי מהנדסי QA עושים גם בדיקות יחידות
- בדיקות יחידה מבודדות קטעה ומאמנתות את נכונותה
- $\text{tutu} \Rightarrow$  פונקציה בודדת, שיטה, נוהל, אובייקט, מודול כלשהו
- בדיקת יחידות היא טכניקת בדיקת WhiteBox

#### יתרונות בדיקות יחידה:

- המפתחים מקבלים תובנה על הפונקציונליות, יכולים להסתכל על בדיקות היחידה כדי לקבל הבנה בסיסית של ה-API של היחידה
- אפשר למתכנת לשחזר קוד במועד מאוחר יותר, ולודא שהמודול עדין פועל כהלה. הנוהל הוא כתיבת מקרי בדיקה לכל הפונקציות והשיטות, כך שבכל פעם ששינוי גורם לתקלה, ניתן לזהות ולתקן אותה במהירות
- בשל האופי המודולרי של בדיקת היחידה, אנו יכולים לבדוק חלקים מהפרויקט מבלי לחכotta לסיום אחרים

#### חסרונות של בדיקות יחידה:

- לא ניתן לצפות שבבדיקה יחידות תתפסו כל שגיאה בתוכנית. לא ניתן להעיר את כל נתיבי הביצוע אפילו בתוכניות הטריוויאליות ביותר
- בדיקת יחידות מעצם טבעה מתמקדת ביחידת קוד. מכאן שהוא לא יכול לתפס שגיאות אינטגרציה או שגיאות ברמת מערכת רחבה.

#### שיטות עובדה מומלצות לבדיקת יחידות:

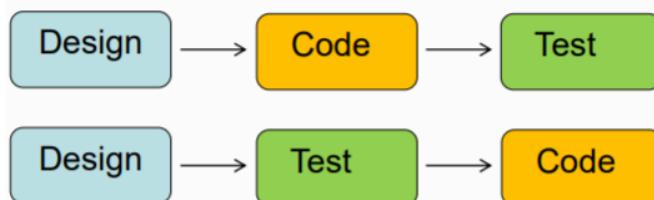
- מקרי בדיקת יחידה צריכים להיות עצמאיים. במקרה של שיפורים או שינוי בדרישות, אין להשפיע על מקרי בדיקת יחידות.
- בדוק קוד אחד כל פעם
- פועל לפוי מוסכמות שמורות ברורות ועקביות עבור מבחני היחידה שלך
- במקרה של שינוי בקוד במודול כלשהו, תודא שהוא עבר את כל הבדיקות יחידה לפני שאתה ממשיך הלאה.
- יש לתקן באגים שזוהו במהלך בדיקת היחידה לפני שתמשיך לשלב הבא-ב-SDLC (Development Life Cycle
- מצא גישת "בדיקות כמו שאתה כותב את הקוד". ככל שתכתוב יותר קוד ללא בדיקה, כך תצטרכך לבדוק יותר מקרים אם יש שגיאות

## דוגמאות לבדיקה יחידה:



TDD (Test Driven Design)

- TDD אינו עוסק בבדיקות אלא בגישה של עיצוב
- אף בסוף התהליך נוצר בדיקות יחידה המשמשות ב"רשות ביטחון" לפיתוח
- TDD עוסק בהסתכליות שונה שונה על דרך יצירת קוד
- מניעת הכנסת פגמים ע"י חשיבה מראש אודוט מה עלול להשتبש
- TDD מטיף לשינוי סדר פעולות הכתיבה (test first programing)



TDD היא פרקטיקה שמקורה בגישת XP

- כתוב בדיקה לפני הוספת הקוד הפונקציונלי הבא
- ממש את הקוד הפונקציונלי של שהבדיקה "עובדת"
- בצע "refactoring" כך שהקוד לא רק "עובד" אלא גם בני כהלה
- את השלבים הבאים לעיל מבצעים מחרוזית עד להשלמת המערכת
- יתרונות:

- קוד מסוגל "לבדק את עצמו" חלק מהפתרונות
- חשיבה מראש על בדיקות מובילת להתמקדות במשקיים ואופן שימוש במחalker
- בפרט מהימוש הקונקרטי
- התמודדות משופרת עם שינויים בדרישות או בהרכבת הוצאות
- כוונות המקודד מתעודת בצורה מפורשת

## • מדוע לא לכתוב קוד בדיקות אחרי קוד פונקציונלי?

- העדר משוב ישיר לעיצוב ולכתיבתה במהלך הפיתוח
- מציאות מלמדת כי אחרי שקוד עובד ישנה נטיה לא לכתוב בדיקות יחידה
- בדיקה הנכתבת בתום המימוש הפונקציונלי נוטה לתמוך בתניב שהוגדר על ידי
- מימוש זה ואינה מתמודדת עם תקלות אפשריות (טאוטולוגיה)

## • תוצאות השימוש ב-TDD

- שיפור עיצוב וניקיון קוד
- ביצוע Refactoring בטוח יותר
- הבדיקות משמשות כסוג של תיעוד לקוד הפונקציונלי (Documenting Intent)
- ניפוי שגיאות מהיר (קוד חדש בד"כ הוא הגורם לשגיאה)