סיכום קורס הנדסת תוכנה :

שיעור 1 :

מושגי יסוד והגדרות :

הנדסה – תפקידה של ההנדסה זה ליצור דברים, אך לעומת אומנות לצורך העניין ההנדסה מבוססת ומסתמכת על העולם המדעי.

בהנדסה נרצה לצמצם את האי ודאות וננסה להיכנס לכמה שיותר תבניות בכדי שנוכל לייצר את המוצרים כמה שיותר דומים אחד לשני, באומנות לעומת זאת השונות היא לכתחילה וזה מה שמייחד אותה.

תוכנה – אוסף של שירותים דיגיטליים ,לדוגמא מערכת הפעלה זה אוסף של שירותים מסוימים, או שאפליקציה זה אוסף של שירותים ייעודיים (עיבוד תמונה וכו').

הנדסת תוכנה – עוסקת ביצירה של מוצרי תוכנה – אך בניגוד להנדסה אין שמירה על אחידות המוצרים והתוכנות שונות ומשונות בהתאם לדרישות הלקוח, אך בכל זאת אנו שואפים להיות הנדסה – אנחנו משתמשים בתבניות, תהליכים, שיטות עבודה ופרקטיקות כלליות.

מערכת – אוסף של חלקים שמקיפים את כל המציאות הקיימת (מערכת השמש וכו'), כלומר אנחנו לוקחים כמה יישויות מופשטות (העיקר שלו, התמצית שלו) שמקיימות קשרי גומלין או עם תלות הדדית מתמשכת ובעלת מטרה.

ניתן להגיד שכל מערכת בנויה מהפשטה, בעלת גבולות, מבנה (OOP למשל), יחסים פונקציונליים - מקבלת קלט ומחזירה פלט (יש לכל מערכת התנהגות), ובעלת מטרה.

מערכת סגורה – מערכת שאין לה קשר עם הסביבה החיצונית, היא רק בתיאוריה.

Stakeholders – אדם בעל עניין או אינטרס שנמצא בתוך המערכת והרבה פעמים הוא משנה את כל קבלת ההחלטות בשלב תכנון ויצירת התוכנה.

UI/UX – חייבת להיות התאמה כמה שיותר שלמה בין חווית המשתמש (UX) למשתמש עצמו (UI), לכן קודם נבין לעומק מי המשתמש כדי שנוכל להתאים את המוצר כמה שיותר.

מודל – לקיחת מערכת וייצוגה בצורה מופשטת.

תיכנון תוכנה (software design) – עיצוב זו הגדרת פתרון כאשר אתה כפוף לאילוצים שאני חייב לעמוד בהם שוב בניגוד לאומנות שלא מוגבלת באילוצים מהנדס צריך לעמוד בשלל אילוצים והגבלות ובהתאם אליהם הוא צריך למקסם את הפתרון (התוכנה) שלו.

כאמור בתהליך הפיתוח התוכנה החכמה זה להתמודד עם דרישות (אילוצים) דינאמים ומשתנים תוך כדי הפיתוח, כלומר האילוצים שהיו לי בהתחלה והפתרון שהיה לי כבר לא רלוונטיים לפעמים בתהליך הפיתוח עצמו.

ארכיטקטורה – זה סוג של עיצוב וההבדל בין ארכיטקטורה לעיצוב זה עומק השינוי , כלומר אם אעשה שינוי בארכיטקטורה אני מבצע שינוי כבד ועמוק מאוד.

עיצוב מודולרי – עיצוב תוכנה = מודולציה – לקיחת השלם ופריסה לחלקים כאשר כל חלק יקרא מודול אם הוא מתחלק בצורה נכונה ושלמה.

וזה העיקר בעיצוב התוכנה.

- ההבדל בין לקוח למשתמש – לקוח זה אחד שמשלם ואילו משתמש זה מי שמפעיל את התוכנה בקצה , האינטרסים של שניהם שונים לגמרי וצריך להבין מה הלקוח רוצה כדי למקסם את השימוש של המשתמש בקצה.
- תהליך התמודדות עם קשיים בתכנון והנדוס תוכנה :
  - תהליכים אדפטיביים – היכולת להגיב לשינויים.
  - עקרונות – עיסוק בכוח אדם, שיתוף פעולה , תקשורת.
  - פרקטיות הנדסיות – התאמת הקוד לשינויים (החלפת אלגוריתמים ,שינוי בהתאם לבאגים),TDD - תכנון ויצירת בדיקות לפני כתיבת הקוד (כתיבה מונחת בדיקות).

שיעור 2 :

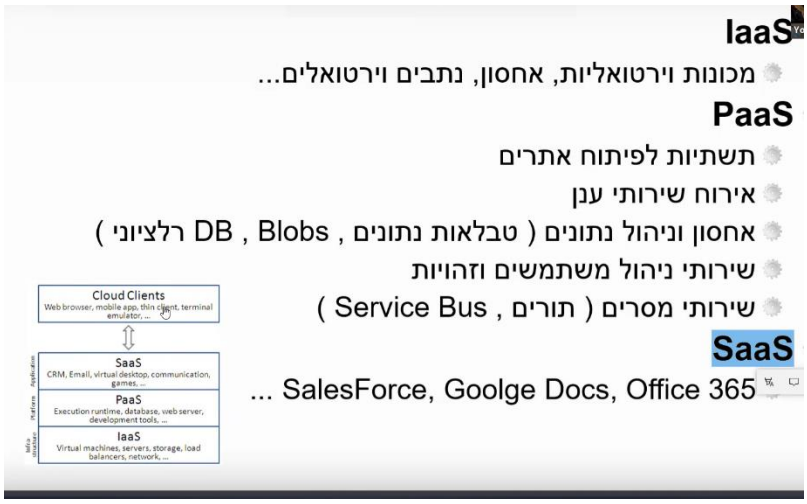
Software service – שירות זה קבלה של משהו כאשר אני לא הבעלים שלו זה השירות הקלאסי.

שירות תוכנה - יחידות הקצה שמהם אני בונה אפליקציה , אוסף של מחלקות (component) שנגישות למשתמשים בכל מקום, השירות כמובן לא חייב להיות בענן.

מתאפיינת בצימוד נמוך (כלומר, אני יכול להשתמש בפרט אחד בלי להביא את כל הקומפוננטה) כי אני לא יודע מי השתמש בה ומה הצרכים שלו.

Cloud Service – ענן – זה אוסף של שירותים בכל מיני רמות רגולציה שונות כאשר את השירות אני מקבל מהמאקרו או מהמיקרו.

המאקרו : SaaS, PaaS , IaaS. – שירות כתוכנה ,לדוגמא : PaaS– Platform as a service – מעיין ספריה המכילה מאות שירותים המוכנים לתת שירות בשלל נושאים כך שאין צורך שהשירות יהיה לוקאלי או פיזי בכתיבת הקוד שלי ויש לי גישה לשירות הזה מכל מקום.



מיקרו : web service . – קבלת שירות כחלקים מהתוכנה ,לדוגמא : AWS – AMAZOM WEB SERVICE שבו ניתן לקבל חלק מהענן כשירות ולא את כולו.

הבדלים בין המאקרו למיקרו (שאלת מבחן): במיקרו ב web service כל התחזוקה והאחריות היא עלי על המשתמש לעומת השירותים שהמאקרו נותן בו כל התחזוקה, העדכונים והתפעול הוא על נותן השירות.

SOA – Service oriented architecture - מחשוב ארגוני.

מהלך אופייני השווה כרבע מהמבחן :

בשנות ה80 כאשר רצו לפתח אפליקציה השתמשו במונוליט – חבילת קוד שלמה שהפירוק שלה מהווה בעיה גדולה, לאחר מכן פיתחו שיטות עבודה בה הקוד מגיע בחלקים וכך יש יותר גמישות "לפרק ולהרכיב" בהתאם לצורכי הפיתוח בלי לשלם מחיר כבד מידי.

בשנות ה2000 היו נפוצות תבנית השכבות בו כל שכבה תלויה בשכבה שמעליה לדוגמא השכבה שמביאה את הנתונים מDB תלוי בשכבת הקוד שמעליה, בתבנית זו יש שלושה

אספקטים – פרזנטציה ,לוגיקה מרכזית וגישה לנתונים , שכבה זו הייתה נחשבת מאוד טובה.

עם השנים הפרידו את שכבת הפרזנטציה משני האספקטים האחרים שמשמים כעת כסרבר של האפליקציה, כעת יצרו קוד שיודע לדבר בHTTP (פרוטוקול תקשורת בשכבת האפליקציה) וכך יצרו ביזור של המערכת ממחשב אחד לשני מחשבים.

כעת לקחו את שני האספקטים (לוגיקה מרכזית , וגישה לנתונים) ופיצלו אותם – כך שיצא שישנם שני שרתים שאני יכול לנצל בצורה עצמאית בהתאם לצרכיי , כעת לסרברים האלו הרבה יותר קל לגשת והשירות שלהם יותר ממוקד.

לזה קראו SOA – עיצוב של מערכות מידע בתוך הארגון – לבסס את הפיתוחים שלו על סרוויסים וכל אפליקציה ספציפית לוקחת מה שהיא צריכה.

Microservices – לקיחת פונקציה אחת והפיכתו לmicroservices – מבצע פעולה אחת בצורה מאוד טובה.

יש לארכיטקטורה זו יש לא מעט יתרונות :

- משוחרר מכל אילוץ – ניתן לכתוב אותו בכל שפה שבא לי.
- הפיתוח לא דורש משאבים גבוהים (מפתח אחד או שניים).
- קל מאוד לשימוש ופיתוח.
- התאמה מאוד נוחה לארגון – בהתאם למה שארגון צריך אני מפתח.
- ממוקד – קל מאוד לתקן באגים.
- במצב של קריסה – לא הכל קורס וניתן להמשיך לתפקד.
- אי תלות מערכת מרכזית אחת – לדוגמא, במידה ואתה תלוי כמפתח בספריה אחד שהפסיקו לפתח אותה אתה בבעיה.

חסרונות :

- ביצוע כמות בדיקות גדולה יותר לכל סרוויס בו אתה משתמש.
- מעבר אינפורמציה חסום.
- שני צוותים יכולים לעשות אותו דבר כי לא כולם מודעים למה שכולם עושים.
- יותר מידי use cases .
- אין חוקים וכללים שתופסים לכל משך הפיתוח – דבר שיוצר עיצוב וחלוקה של קוד בצורה שגויה.

Use case – מי בעל התפקיד ומה הוא רוצה מהמערכת ומה השימושים שלו (כל דרישה).

- **שאלת מבחן** – מה ההבדל העקרוני בין XML , JSON ל CSV ?
ת. מה שדומה זה שכולם מעבירים נתונים בצורה טקסטוריאלית אך רמת הדחיסות שלהם שונה ורמת הביטחון שהכלים האלה נותנים בהעברת הנתונים שונה – כאשר XML מקסימלית וCSV מינימלית.
- **שאלת מבחן** – מה ההבדלים בין web service ל Cloud service ?
ת. ב web service כל התחזוקה והאחריות היא עלי על מקבל השירות לעומת השירותים שCloud נותן בו כל התחזוקה, העדכונים והתפעול הוא על נותן השירות.

שיעור 3 :

בעלי תפקידים :

מנתח מערכות :

- מנתח את המצבים הקיימים ומבין את הבעיות הדרושות.
- מצליח להגדיר את הבעיה ואת קווי המתאר לפתרון שלה.
- קובע את מטרות המערכת ומגדיר את דרשותיה.
- אוסף ומסווג את הדרישות מהלקוחות ומהמשתמשים.

מנהל פרויקט :

- האחראי שצריך לדאוג לעמידה בדרישות ,בתקציב ובלו"ז.
- מכיר את כל התהליכים ומתאימים אותם לצורכי הפרויקט.
- נדרש ליכולות ניהוליות – יודע לארגן ,לחשוב ולהניע אנשים.
- יצירת gant.

מנהל מוצר (PO) :

- דואג לאינטרסים של הלקוח.
- קולו של המשתמש בצוות הפיתוח – מסתכל מהצד של המשתמש .
- מביא את המוצר הנכון למשתמשים – מעיין רפרנס בין הלקוח למפתח.
- חוקר את השוק ומבין את הצרכים ולאחר מכן בונה חזון לפתרון הבעיה.
- פיקוח על התכולה, הקצב ופתרון בעיות.
- כאשר הגרסה יוצאת תפקידו לעקוב ולקבל פידבקים מהמשתמשים.
- תפקיד אסטרטגי – קובע את הכיוון – לאן החברה הולכת.

מהנדס מערכת :

- אחראי טכנולוגית על הפיתוח.
- קובע את מיפוי הטכנולוגיות שבהם המערכת תשתמש.
- ראיה מערכתית – בקיא ומכיר את כל התהליכים וצריך להבין את הקשרים בינהם.
- יודע להסתכל על המערכת גם מבחינה חומרתית וגם מבחינה תיכנותית.

ארכיטקט תוכנה :

- מי שמעצב ומייצר את המערכת.
- High level design
- בוחר את תשתיות התוכנה.
- מחלק למודולים פיזיים ולוגים
- כללי העיצוב – איך מוסיפים רכיב חדש ,איך מאפיינים רכיב חדש .
- פונקציונאליות – מבין מה עושה כל חלק.
- הבנת הקשרים והתלויות בין כל חלקי המערכת.

ראש צוות :

- בונה את הצוות – בוחר את האנשים.
- תיאום עבודת הצוות.
- קביעת נהלים.
- אחראי על הlow level design.

מטמיע מערכת :

- התקנות, כתיבת מדריכים ,התקנה בפועל במחשבים שונים.
- תמיכה וליווי שוטפים לעובדים שנתקעים בתפעול המערכות השונות.
- משך זמן ההטמעה תלוי בגודל המערכת.

<u>תוצרי ניתוח מערכת והנדסת דרישות :</u>

הנדסת דרישות (RE) – דרישות, הכוונה לשירותים ואילוצים.

מה עושים עם דרישות ?

- אוספים אותם – תשאול בעלי עניין.
- ניתוח הדרישה – האם היא רלוונטית ?
- הכנסת מסמך דרישות.
- בדיקת אימות – נכונות ביחס לצרכים.

כיצד אוספים או מאתרים דרישות ?

- סיעור מוחות – אנשים מביעים את דעתם.
- יצירת demo - יצירת אב טיפוס שמדליק את הדמיון אצל אנשים – דבר שמניע דרישות ופיתוחים.
- אינטרנט, ראיונות ושאלונים.
- קבוצות מיקוד.

אפיון וסיווג דרישות :

<u>מאפייני דרישות :</u>

- יש דרישה כללית ויש דרישה ספציפית.
- הדרישות הן בסיסי לבקשת הצעות ולכן עליהם להיות פתוחות לשינויים.
- בניגוד לאמור לעיל הדרישות הם בסיס לחוזה ולכן עליהם להיות מוגדרות היטב וסגורות לשינויים וזה סוג שני של דרישות.
- דרישה טובה זו דרישה שהיא ברורה לגמרי ,עקבית ומתועדפת.

<u>סיווג דרישות – ישנם שלושה סיווגים :</u>

1. דרישות משתמש – משפטים בשפה פשוטה המלווים בדיאגרמות השירותים שהמערכת תספק - תיאור הדרישה היא עבור המשתמשים, כלומר המשתמש רואה ורוצה להשתמש במערכת הזו.

דרישות מערכת – מה הפונקציות של המערכת ? המטרה היא שונה – הדרישות הם בסיס לחוזה, פיתוח וכו' – איך המערכת תעבוד.

2. דרישות פונקציונליות – מה המערכת מספקת ? איך היא תגיב לקלט ומה הפלט שלו - לא איך היא עושה אלא מה היא עושה.

דרישות לא פונקציונאליות – כל מיני אילוצים , דברים שצריך לעשות – לדוגמא, רגולציה, אבטחה וסטנדרטים ,חלונות זמן לתגובה – כמה פעולות היא יכולה לקלוט ולעבד בזמן מסוים.

3. דרישה תפעולית – איך מתנהגת התוכנה, איך אני מדבר איתה ואיך היא מגיבה.

דרישת מידע – באיזה נתונים אנו מטפלים ? – איזה קלט נדרש לתהליך מסוים ומה הלקט בסופו.

דוגמאות :

תיעדוף דרישות – קיימים מודלים רבים ונבחר מודל בהתאם לסיטואציה אבל המיקום האידיאלי בו אנו נרצה להיות זה מינימום מאמץ ומקסימום תועלת.

תיעוד דרישות – את הדרישות נתעד כדי ליצור תקשורת טובה יותר.

הפורמט שבו אנו נשתמש זה מסמך SRS שמהווה בסיס לדיאגרמות בתקן UML.

<u>סוגי תבניות לייצוג דרישות :</u>

PRD – ייצוג של הפיצ'רים או החידושים מנקודת מבטו של המשתמש.

FRD – פונקציונאלי - משמש לאיך אני מתמודד טכנית עם דרישות מסוימות – וזו הפעם הראשונה שבו נדבר על טכנולוגיות מקצועיות.

SRS – התוכנית שבה נשתמש כדי לפתח – הבסיס לפיתוח מה שיהיה שם זה מה שיהיה איזה פיצ'רים צריכים להיות ומה ההתנהגות המצופה , השלד המרכזי איתו אנו יוצרים לדרך.

בסוגי התבניות לעיל ניתן להשתמש בהדרגה PRD -> FRD -> SRS .

- **שאלת מבחן** – מה אומרת עקומת בוהם ?
ת. עקומת בוהם באה להראות לנו כמה חשוב להגדיר דרישות מלכתחילה, בעצם היא מראה לנו את היחס בין הזמן שאיתרת שצריך משהו (דרישה) לבין כמות ההשקעה הנדרשת שצריך כדי לממש אותו.

- **שאלת מבחן –** מה יכול להיחשב כדרישה ?
ת. דרישה יכולה להיחשב כשירות או אילוץ , שירות ופיתוח או שירות ועיצוב.

- **שאלת מבחן –** מה זה התנהגות של מערכת ?
ת. התנהגות של מערכת זה הטרנספורמציה שהיא עושה לקלט – מגיעות בקשות והיא עושה את הטרנספורמציה.

שיעור 4 :