

סיכום קורס הנדסת תוכנה :

שיעורים 1+2 :

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) דינמיים ומשתנים תוך כדי הפיתוח, כלומר האילוצים שהיו לי בהתחלה והפתרון שהיה לי כבר לא רלוונטיים לפעמים בתהליך הפיתוח עצמו.

בתכנון התוכנה יש להתחשב גם בקלות השימוש וזמן הקימפול.

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.

זזה העיקר בעיצוב התוכנה.

מודולציה לוגית – חלוקת המערכת למרכיבים לפי תפקיד.

מודולציה פיזית – חלוקת המערכת לחלקים בלי התחשבות בתפקיד.

המינימום נדרש והבסיסי כדי שהעיצוב יהיה ראוי זה שהלוגיקה המרכזית יהיה חלק והטיפול בנתונים יהיה חלק שונה.

עקרון SOC – כל פעם שיש יחידה בקוד או יחידה כלשהיא שעושה יותר מתפקיד אחד זו בעיה, לכן בעיקרון זה כל יחידה היא בעלת תפקיד מוגדר ויחיד, על פי עקרון זה אנו פועלים.

הדרישות הנצרכות כדי ליצור מוצר תוכנה –

- אין מוצר תוכנה שאין לו צורך ובעיה הדורשת פתרון, זה הבסיס לכל מוצר.
- רעיון, חזון, דרישות, התכנות קיימת בעולם.
- משאבים ושותפים לדרך.
- ידע – תמיד חסר בשוק ידע.
- תהליכי ניהול – יכולות שיווקיות פיננסיות והנדסיות.
- סביבות פיתוח – שלושה סביבות נדרשות כדי לפתח תוכנה – פיתוח בדיקות וייצור.
- תזמון נכון.

קשיים בפיתוח תוכנה – לו"ז, תקציב, התאמה לצרכים, איכות, שימוש חוזר, תחזוקה.

הסיבות המרכזיות לכשלים בפיתוח תוכנה –

- הגורם האנושי – תקשורת לקויה (פתרון: גיבוש קבוצתי והכרת הסיבה והאנשים בעבודה), ניהול לוקה, אינטרסים מנוגדים, סטנדרטים ושיטות עבודה שלא מוגדרות היטב, חוסר בבקרת איכות, חוסר מקצועיות.
- טבעו של עולם – אי וודאות, מורכבות, דרישות מוגזמות, לו"ז ותקציב.
- הפתרון: קבלת המציאות והתאמה של המציאות למצב, שילוב ושימוש בכלים שלך בצורה נכונה.

פעילויות בפיתוח מוצרי תוכנה –

- ייזום – היזם מזהה את הבעיה ואת ההזדמנות.
- הגדרת דרישות – הגדרת ותיאור הדרישות.
- ניתוח – ניתוח ושיפור הבעיות.
- עיצוב ומימוש.
- בדיקות.
- הטמעה והחזקה.

תהליך יזום יצירת תוכנה –

- "מישהו" יזום פיתוח מוצר
- בארגון – מנהל מחלקה, מנהל חטיבה, סמנכ"ל, מנכ"ל...
- בשוק חופשי - יזם בעל רעיון העונה לצורך
- בארגונים נדרש מסמך יזום
- אסמכתא מגורם ניהולי להתחיל בתהליך והגדרת אחריות לביצוע
- בשוק חופשי נדרש מסמך חזון (Vision)
- אמצעי תקשורת עם גורמים אותם עשוי לענן המיזם
- מרכיבים מרכזיים במסמך
 - הגדרת הצורך
 - הנחות ואילוצים
 - דרישות מרכזיות
 - תפוקות מרכזיות

מסמך יזום – יצירת מסמך שנותן "כותרת" לפרויקט בגדול.

המסמך אינו מסמך טכני אך הוא מתאר את המצב הקיים ואת הבעיות והפתרונות.

המסמך ישמש כקלט ובסיס לתהליך הפיתוח.

המסמך מורכב מהמטרה וההצדקה לפרויקט, יעדים ומדדים להשגתם, דרישות ותיאור כללי, אילוצים ול"ז משוער, הערכות תקציב ורשימת בעלי עניין.

מודל SMART לקביעת יעדים –

- **Specific** - יעדים ממוקדים ככל הניתן, ולא כלליים מידי.
- **Measurable** - יעדים הניתנים למדידה
- **Attainable** - יעדים ברי השגה- ריאליים והגיוניים.
- **Relevant** - יעדים המשרתים את הייעוד והאסטרטגיה הארגונית.
- **Time-bound** - יעדים התחומים בזמן.

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

- 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 3 :**בעלי תפקידים :****בעלי עניין :**

- אדם בעל אינטרס חיובי או שלילי היכול להשפיע על התפתחות הפרויקט.
- גורם שמעורב "מאחורי הקלעים" (משפיע על הפיתוח אבל לא בצורה ממשית) בצורה אקטיבית לדוגמא : לקוח, משתמש, משקיע, ספק, רגולטור.

מנתח מערכות :

- מנתח את המצבים הקיימים (מרחב הבעיה) 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 ובזמן.
- 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 התכנון, 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.
- רואה מערכתית – בקיא ומכיר את כל התהליכים וצריך להבין את הקשרים ביניהם.
- יודע להסתכל על המערכת גם מבחינה חומרית וגם מבחינה תכנותית.

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 low 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.

תוצרי ניתוח מערכת והנדסת דרישות :

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 :**מאפייני דרישות :**

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

סיווג דרישות – ישנם שלושה סיווגים :

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 פונקציונליות – 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 :

אוניברסיטת
הרצליה
בשומרון

דוגמא לדרישות פונקציונליות

המערכת תאפשר להזין הזמנות מלקוח למוצרים שבמלאי.
המערכת תייצר מספר הזמנה חד ערכי בעת שמירת ההזמנה.

כל הזמנה תאפשר לציין למוצר יחידת מידה וכמות. כל שינוי ביחידת מידה מחייב רישום כפריט נפרד בהזמנה.

לכל פריט בהזמנה יש לרשום יחידת מידה, כמות ומחיר ליחידת מידה.

ניתן להזמין מוצרים הן דרך מרכז ההזמנות והן בצורה ישירה באינטרנט.

עבור כל פריט שיוצג יש להציג את שמו, הברקוד שלו וצבעו.

תפעולית

תפעולית

תפעולית

מידע

מידע

מידע

אוניברסיטת
הרצליה
בשומרון

דוגמא לדרישות לא פונקציונליות

המערכת תתבסס על מחשב מסוג
המערכת תהיה זמינה ** שעות ביממה
עלויות הפיתוח לא יהיו גבוהות מ- 20M\$
תאריך היעד של המערכת הוא 1/1/2022
יש להשתמש בחישוב הפרמיה החודשית כפי שמחושב במערכת הקיימת
יש להחזיר תשובה לחיפוש תוך 2 שניות לכל היותר

אילוץ חומרה

מאפייני איכות

אילוץ ניהולי

אילוץ ניהולי

אילוץ מימוש

דרישות ביצועים

תיעוד 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.

סוגי תבניות לייצוג דרישות :

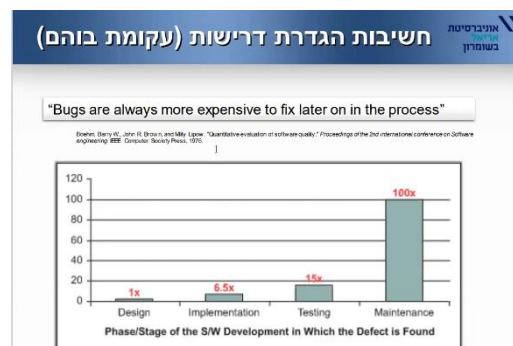
PRD – ייצוג של הפיצ'רים או החידושים מנקודת מבטו של המשתמש.

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 אילוף , 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 :

מקרי שימוש (תרחישים) USE CASE - ישנם שני דרכים להציג תרחישים ויזואלית (דיאגרמה) וטקסטוריאלי (תיאור מילולי מפורט).
 כמובן שתרחישים כאלו משמשים כמודל (מודל ויזואלי).

תרגום דרישות למודלים :**גישות :**

- גישת התהליכים – מה בעצם קורה פה ? בתהליכי משתמש ובתהליכי המערכת.
- גישת ישויות הנתונים – פירוק המערכת לחבילות (ישויות) של נתונים.
- גישות משולבות – גישה שמשלבת בין השניים הנ"ל.
- גישה תהליכית - data flow – השבילים שעובר הדאטה מהמקורות החיצוניים אל הפונקציות שעושות שינויים מסוימים ועד לאחסון שלהם.

תרשים DFD – תרשים זרימה :

מורכב מחמישה סמלים מרכזיים :

עיגול – פונקציה.

מלבן – ישות חיצונית .

מלבן מאורך – מאגר מידע.

משולש – יחידת זמן.

חץ – זרימה.

USE CASE – ראה לעיל.**ERD -Entity relation model:**

מודל גישה ריאליזציוני המבוסס נתונים.

שימוש בתרשימי ERD.

שימוש מודרני בתרשימי UML – בדיאגרמות ACTIVITY I CLASS.

סוגי מודלים :

1. מודל קונספטואלי (מחלקתי) – מתאר איזה ישויות (קונספטים, מחלקות) יש לנו ומה היחסים ביניהם.
 2. מודל לוגי – מדבר עם הגישה הטכנולוגית בלי המימוש המלא החיסרון הוא שהמשמעות והתכלס הולך לאיבוד.
 3. מודל פיסי - מדבר על איך אני מאחסן ושומר את הנתונים ממש – הקצאת נפחים , אופטימיזציה.
- לרוב נתחיל בפיתוח המודל הקונספטואלי ולאחר מכן נוסיף ונתרגם מודל לוגי ומודל נתונים לצורך המימוש. את היחסים והקשר ביניהם נבטא באמצעות דיאגרמת R-E מבוססת קשרים.

- **שאלת מבחן – תיאור תהליכים.**
ת. Long – תהליכים שרצים בזמן ארוך ומטפל בדברים שרצים ברקע
Short - תהליכים שמטפלים בדברים אינטראקטיביים עכשויים.
תהליכי משתמש – תהליכים שהמשתמש עובר כדי להשיג את יעדו.
תהליך ישים – תהליך שמבטיח את השלמת כל הפעולות עד קבלת התוצאה.
- **שאלת מבחן – מה זה Domain ?**
ת. Domain – זה תחום עיסוק (לדוגמא : רפואה, מסעדות וכו').

שיעור 5 :

מושגים נוספים (לא למבחן):

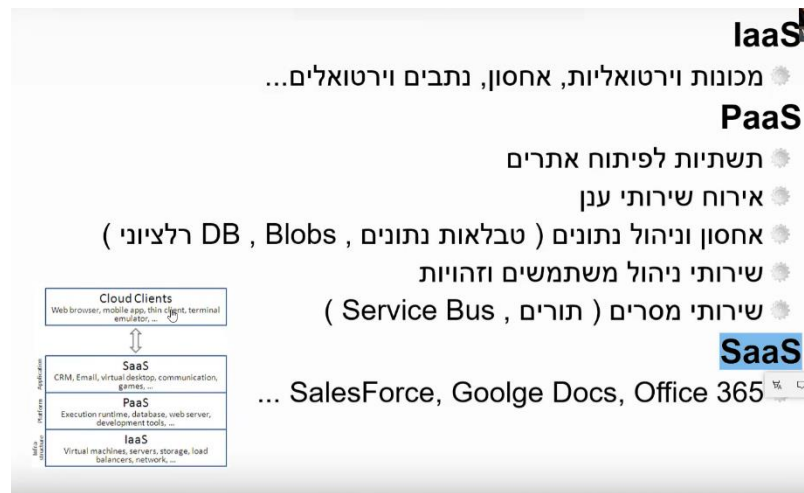
Software service – שירות זה קבלה של משהו כאשר אני לא הבעלים שלו זה השירות הקלאסי.

שירות תוכנה - יחידות הקצה שמהם אני בונה אפליקציה, אוסף של מחלקות (component) שגישות 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 תוכנה, לדוגמה: Platform as a service – PaaS – 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, לדוגמה: 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:

בשנות ה-80 כאשר רצו לפתח אפליקציה השתמשו במונולית – 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 מדי.

בשנות ה-2000 היו נפוצות תבנית השכבות בו כל שכבה תלויה בשכבה שמעליה לדוגמה השכבה שמביאה את הנתונים מ-DB 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 שיועד לדבר ב-HTTP (פרוטוקול תקשורת בשכבת האפליקציה) וכך יצרו ביזור של המערכת ממחשב אחד לשני מחשבים.

כעת לקחו את שני האספקטים (לוגיקה מרכזית, גישה לנתונים) ופיצלו אותם – כך שיצא שישנם שני שרתים שאני יכול לנצל בצורה עצמאית בהתאם לצרכי, 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 ל-microservices – 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 – לדוגמה, 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.

- 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 ו CSV מינימלית.
- מה ההבדלים בין web service ל Cloud service ?
ת. ב web service כל התחזוקה והאחריות היא עלי על מקבל השירות לעומת השירותים ש Cloud נותן בו כל התחזוקה, העדכונים והתפעול הוא על נותן השירות.

