

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מבוא:

מה זה בכלל תוכנה?

תוכנה היא אוסף של פקודות המאפשרות למשתמש לתקשר עם המחשב, החומרה שלו, או לבצע משימות.

מהי תוכנה טובה?

- תוכנה שעובדת ומבצעת את מה שהוא אמור לעשות.
- תוכנה בעלת משך משתמש נוח, נעים להסתכל עלייה (מצד הלkoח).
- תוכנה בעלת קוד נקי וברור, קללה לקרואה (מצד המפתח) –> קללה לבדיקה ולתחזוקה.

מדוע פרויקטים נכשלים?

- עדי פרויקטים לא מציאותיים או לא ברורים מספיק / הדרישות אינן מוגדרות היטב.
- אומדן לא מדויק של המשאבים הנדרשים.
- דיווח לקרי לגבי מצב הפרויקט / סיכונים לא מנוהלים.
- תקשורת לקויה בין הלkoח למפתחים והמשתמשים / פוליטיקה של בעלי עניין.
- שימוש בטכנולוגיה לא בשלה / פרקטיקות פיתוח מרושלות.
- אי יכולת לנוהל את מרכיבות הפרויקט / ניהול לקרי של הפרויקט.
- לחצים מסחריים.

פיתוח תוכנה זה קשה

לתוכנה אין:

- טכנולוגיה אחת או טכנית ניהול אחת שאיתה בודאות ניתן לקבל שיפור בפרודקטיביות.
- פתרון קסם כלשהו עבור הקשי המובנה של פיתוח תוכנה.

מרכיבות מהותיות לעומת מרכיבות מקרית

בפיתוח תוכנה יש 2 סוגים מרכיביות לפתור:

מרכיבות מקרית:

- המרכיבות נוצרת עקב כתיבת תוכנה לא עיליה או כתיבת טעויות (באגים).
- אם נצליח להיפטר מהמרכבות המקרית נוכל לשפר את הפרודקטיביות.

מרכיבות מהותיות:

- לב התוכנה, הנתונים, האלגוריתמים ויחסים הגומלין ביניהם.
- המרכיבות טבועה בבעיה, הבעיה נפתרת כשכתבים את התוכנה.
- למרכיבות זאת אין פתרונות קסם.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

תכונות מהותיות בתוכנה

- Complexity, סיבוכיות. לדוגמה: מעקב אחרי מסלול של אדם על גבי מפה. נניח שיש 20 תחנות במסלול. אם נרצה לעקוב אחרי כל האפשרויות לעבור בתחנות. מספר האפשרויות הוא 20!.
- Compormity: קונפורמיות, התנהגות בהתאם, תאימות. לדוגמה:
 - שינויים והוספת יכולות בחומרה משפיעות על התוכנה.
 - תוספת מהירות ודיק.
 - הוספה או שינוי של יכולות.
 - התגברות על מגבלות פיזיקליות.
 - תיקון בעיות עבר.
- Changeability: ניתן לשינוי. לדוגמה: עדכוני גרסה וכדומה.
- Invisibility: בלתי נראה. לדוגמה: מעצב/אדריכל/נגר יכול להציג בפני לקוחותיו שרטוט המתאר איך יראה המוצר בסופו. בתוכנה דבר זה אינו אפשרי.

דרכים לצמצום המורכבות

- שימוש ב level languages high גורם לשיפור של עד פי 5 בפרופודוקטיביות
- שימוש במערכות מומחה, יכול להציג שיטות עבודה מומלצות ולעוזר למפתחים מתחילהים.
- DeepCoder Autonomic Programming
- מציאת באגים בתחילת התהילה.
- סביבות וכלים. לדוגמה Eclipse, Visual studio וכו' וצדומה.
- שימוש בשיטות עבודה מתקדמות, כמו agile.
- קניית תוכנת מדף.
- חידוד הדרישות ובניה אב טיפוס.
- שני הגישה של פיתוח מערכת התוכנה, כמו שהמוץ גדל ומתווספות יכולות, כך גם בתוכנה,
- התוכנה גדלתה ומפתחת ויש לה יותר יכולות.

הנדסת תוכנה

הנדסת תוכנה הchallenge להפתח בסוף שנות השישים של ה-100 הקודמת, על רקע משבר תוכנה. הנדסת תוכנה מתיחסת למבחן החיים של הפיתוח, החל משלב הרעיון, דרך התיכנון, הפיתוח, הבדיקות והתחזקה של המוצר.

המטרות של הנדסת תוכנה:

- הקטנת המורכבות שבפיתוח תוכנה.
- לשפר את אמינות התוכנה.
- להקטין את עלויות התחזקה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מחזור החיים של מערכת תוכנה:

הנדסת תוכנה בראיה מערכתי

הגדרה:

- מערכת היא צירוף של מרכיבים (אלמנטים) הפעלים במשותף ומאורגנים לצורך השגת מטרה מוצהרת אחת או יותר.
- מערכת מאורגנת במבנה היררכי, רקורסיבי, מרכיב של מערכת יכול להיות מערכת עצמאו.
- מערכת עניין, מערכת אליה תחוליך פיתוח נתון, כולל פעילויות ותוצרים.
- מערכת עתירת תוכנה, מע"ת, מערכת אשר התוכנה מהוות בה חלק משמעותית מבחינה הפונקציונליות, עלות הפיתוח, סיכון הפיתוח או משך הפיתוח.

רמות העניין האופייניות בפיתוח מערכות עתירות תוכנה

- רמת הארגון / העסק. למשל: בריאות.
- רמת המערכת. למשל: מערכת ניהול שירותים הקופה.
- רמת פריט התוכנה. למשל: מערכת ניהול תורים.
- רמת רכיב התוכנה. למשל: משק משתמש של התור.
- רמת יחידת התוכנה. למשל: מסר הזדהות, מסר ביטול תור ועוד.

מערכת מבנה ותפקיד

מערכת (בכל רמה) מוגדרת על ידי המאפיינים הבאים:

- תיחום: גבולות המערכת (מה נכלל בה ומה לא), ממשקים למערכת חיצונית.
- מבנה: מרכיבים, קשרים וממשקים פנימיים (פייזיים/לוגיים – פונקציונליים).
- תפקיד: תהליכי/התנהגות, אינטראקציה עם הסביבה.

מוצר תוכנה מסופק באחת משלושת הצורות:

1. מוצר עצמאי: מגע בדיסק קשיח או בהורדה מהאינטרנט מיועד לשימוש על המחשב ומספק מערכת הפעלה ותוכנה תשתייתית. דוגמה: מערכת אופיס (לפחות פעם).
2. מוצר משובץ (embedded): מגע ביחיד עם החומרה המתאימה לו ספציפית יש בו מערכת הפעלה ותוכנה תשתיית כמוצר שלם. למשל: טלפון סלולרי.
3. תוכנה כשירות (Software as a Service = SaaS): התוכנה עצמה אינה מגעה ללקוח ואין נשארת ברשותו אלא הוא מקבל שירותי אמצעותה. דוגמה: Gmail, Dropbox.

מחזור חיים – הגדרה

"**aboliczja** של מערכת, מוצר, שירות, פרויקט או ישות אחרת מעשה ידי אדם משלב הקונספטיה ועד הוצאת המוצר השירות"

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שלבי תהליכי הפיתוח

1. הבנת הבעיה/הצורך.
 2. גיבוש תפיסת פתרון (מערכת): תפיסת מבנה, תפיסת פעולה.
 3. תיקון פתרון (מערכת -> תת מערכת -> מצלולים -> מודלים).
 4. בוחנת הפתרון: תאימות לתיקון, מענה לבעיה/לצורך.
- "שם ספציפי לתהליכי זהה נקרא "מודול מחזור החיים"

מחזור החיים של מערכת תוכנה

- התחלתה: דרישת לקוחות.
- סיום: אין יותר שימוש בתוכנה.

שלב היום –> שלב הדרישות –> שלב הניתוח –> שלב התוכן –> שלב המימוש –> שלב השימוש
–> שלב התחזקה –> פרישה

שלב היום

פעולות בשלב היום

- הגדרת הלקוחות.
- מציאת מומחה יישום.
- ניתוח מצב ק"ם, בדיקת מערכות דומות קיימות.
- הגדרת מטרות ויעדים.
- הגדרת אילוצים, מגבלות וסיכון.
- הגדרת תועלות וחסכנות.
- בדיקת יישומיות ועלות/תועלות.
- גיבוש צוות התיכנון של המערכת.
- קביעות ליו"ז ומשאבים.

הגדרת הלקו

זיהוי הלקוחות האפשריים:

- במקרה של מערכת בתוך הארגון:
 - זיהוי של סוגי הלקוחות השונים (מנהל, עובד, משתמש קצה וכדומה).
- במקרה של מערכות מחוץ לארגון:
 - זיהוי הלקוחות הפוטנציאליים שיש סיכוי שהיו מעוניינים במערכת.

ישנם הרבה דרכים להגדיר את השוק בו המוצר שלנו יפעל.

אנחנו נתיחס לשניים:

- B2B: business to business
- B2C: business to customer

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מציאת מומחה "ישום"

- הלקוח ממנה נציג מטעמו אשר ילווה את המערכת, מנהלית ומקצועית.
- במערכות מידע "מומחה היישום" הוא בדרך כלל מחוץ ליחידת המחשב.
- במערכות תשתית המומחה עשוי להיות מתוך יחידת המחשב.
- מומחה היישום ירכז אצלן את דרישות הארגון מהמערכת.
- מומחה היישום ילווה את המערכת משלב הייזום ועד הטמעת המערכת בארגון.
- מומחה היישום יהווה גורם מקשר בין הגורמים השונים במרקם של התנאיות בדרישות מהמערכת.
- גם במערכת ללא ארגון יש למצוא מומחה "ישום" שהיא אחראית על שמירת האינטרסים של הלקוחות הפוטנציאליים.

ניתוח מצב קיימ

- במערכת עבור ארגון ספציפי יש לבדוק את הדברים הבאים:
 - האם יש מערכת דומה ממוכנת? אם אין, איך מתנהלת המערכת הידנית?
 - האם היה בעבר ניסיון להקים מערכת דומה? אם כן, מה היו תוצאותיו? מי היה מעורב?
 - האם יש מערכת דומה בארגון אחר?
 - מה המצב שם? מה ההיסטוריה, מה ניתן ללמידה מזה?
- במערכת ללא ארגון:
 - האם קיימות מערכות דומות?
 - מה מצב? מה ניתן ללמידה מהן?

מטרות ויעדים

מטרות:

- תיאור כללי של התכונות שרוצים שיהיו במערכת. דברים שרוצים שהמערכת תדע/תבצע לדוגמה: שיפור השירותים הנitinן ללקוחות החברה.

יעדים:

- פירוט המטרות ליעדים שאוטם אפשר לכמת.
- לדוגמה: עבור המטרה לעיל ניתן להגדיר את היעדים הבאים:
- מענה אנושי לשיחה תוך 2 دق' לכל היותר.
 - דיווח על תקלות בזמן אמת.

מודל SMART:

יעדים צריכים להיות מוגדרים לפי SMART:

- Specific: יעדים מוגדרים ככל הנitinן, ולא כללים מיד.
 - Measurable: יעדים ניתנים למדידה, אחרת לא יוכל לדעת האם היעדים הושגו או לא.
 - Attainable: יעדים ברิ השגה.
 - Relevant: יעדים המשרתים את הייעוד והסטרטגיה הארגונית. יעדים שימושיים לארגון.
 - Time bound: יעדים התוחמים בזמן.
- דוגמה ליעד שעומד בדרישות המודל: "קיצור תהליך קביעת תור ב 50% עד 2020".

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

דוגמה לעד שלא עומד בדרישות המודל: "הגדלת מספר הליקוחות".
 האם SMART באמת טוב ?
 למה לא ? הדגם הזה מסורבל מתייש וקשה לעמוד בווא ולרשום את כל המטרות שלנו.
 פיתרון:

מעבר למודל (Measurable, Time bound) MT

למה זה מכיל בתוכו גם את מודל SMART ?

- S – אם המטרה מוגדרת ומדויה היא ספציפית.
- A – בר השגה לאחד לא בהכרח בר השגה לאחר.
- R – דומה ל S. שונה אצל כל אדם.

אלוצים מגבלות וסיכון

ניתוח בעיות שועלות לבוא מאלוצים שונים:

- בתחום התפעול (מי יתפעל את המערכת ? האם המערכת תתאים לו ?)
- טכנולוגיים (באיזה מכשיר טלפון המערכת תתמוך ?)
- ארגוניים (אם מתאים ל רפואי ?)
- זמן ומשאבים (צריך להיות מוקן לפני 2020).
- תקציב (קופת החולים הקצתה למערכת סכום 5 מיליון ש"ח).

סיכון:

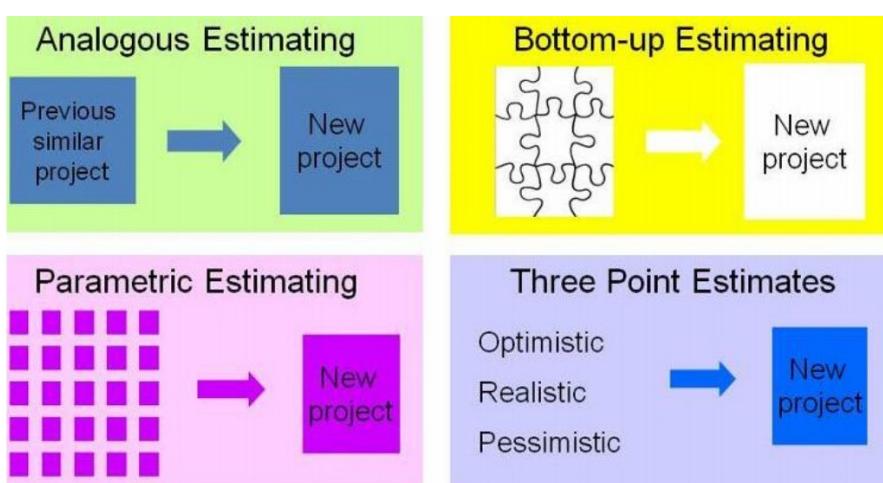
- כל דבר שעלול לקרו ולהפחית או להקטין את הסיכויים להצלחה בפרויקט.

בדיקות שימוש ועלות/תועלת

שאלות שנוצרו לשאול את עצמנו:

- האם צפויים קשיים/מגבליות בתחום הגדרת היישום ?
- האם מדובר בטכנולוגיה חדשה ובולטוי מוכרת ?
- האם צפויים קשיים/מגבליות במימוש המערכת ?
- האם יש בעיות היבנות ברכיבים מסוימים, למשל, ברכיב אבטחת מידע.
- האם צפויים קשיים בתחום הטמעת המערכת ?

שיטות לשערור עליות:



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שיטות לבדיקת כדיאות כלכלית:

- NPV: net present value
- ROI: Return On Investment
- Payback analysis

חשיבות של כלי המדידה:

- קיימת מגמה למדוד הכל בכספי.
- עזר בהשגת תמיית ההנהלה וקבלת תקציב.
- הגדרת היקף ההשקעה הכספי במערכת.
- כלי לבחינת אלטרנטיביות.
- ניסיונו להפוך הנחות סובייקטיביות, השערות ותקויות לעובדות ומדוים.

האתגרים בכלים המדידה:

- הגדרת הפרמטרים המציגים את התועלת והחסכנות של המערכת.
- הגדרת ערך המצביע הקים לעומת הצפי בכל אוטם הפרמטרים.
- כימות פער החיסכון והתועלת מול עלות ההוצאה.
- בנייה מודל להחזר ההשקעה.
- בנייה מנגנונים לאיסוף מידע על השיפור והחיסכון בפועל.

:NPV

בעברית: ענ"ג, ערך נוכחי נקי.

- PNV זהו מונח מתחום המימון המשמש לניטוח רווחיות של השקעה או פרוייקט מסוים. הערך הנוכחי הנקני מרכיב מהשווי הכספי של סך כל ההכנסות העתידיות של השקעה מסויימות פחות ההוצאות. פעולה זו, אשר מחשבת את הערך הנוכחי של הסכום שתתקבל בעתיד נקראת היוזן והוא לוקחת בחשבון משתנים כגון ריבית או אינפלציה. מהוונים כל הכנסה צפוייה מהמערכת לערך נוכחי וכך נוכל לבדוק האם המערכת כדאית או לא.

מס' שנים לחישוב - N
 הכנסות - הוצאות באותה שנה - R_t
 שיעור הרינו - i
 מתי הגיע הכנסה - t

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

דוגמא

* המשרת שלנו צפויו במשר חמש שנים להבדיל את מס' הלקוחות ב- 50 כ"כ בשנה.

* כל לקוח מקבל קיופת חווים 1000 ש"ח לשנה.

* שער הרינו הוא 10%.

* מחירי שירות הנותרים הוא 18,000 בתיהילה ואחר כרך עשרה ושיפורים לשנה לתחזוקה ו糸ופורם

* אחרי 5 שנים הNPV שלו יהיה 532,550 - 468,220 = 64,330

שנה	5	4	3	2	1	0	סך
ערך הינו	62.10%	68.30%	75.10%	82.60%	90.90%	100%	
הוצאות	250,000	200,000	150,000	100,000	50,000	0	750,000
הכנסות	532,550	155,250	136,600	112,650	82,600	45,450	0
saldo	532,550	377,300	240,700	128,050	45,450	0	
הוצאות	18,000	18,000	18,000	18,000	400,000	400,000	750,000
הכנסות	468,220	11,178	12,294	13,518	14,868	16,362	400,000
saldo	468,220	457,042	444,748	431,230	416,362	400,000	750,000

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

כללי אצבע:

ה NPV מהו אינדיקציה האם כדאי המשיך בפיתוח המערכת או לא, אך הוא לא ח'יב להפסיק באופן חד משמעי.

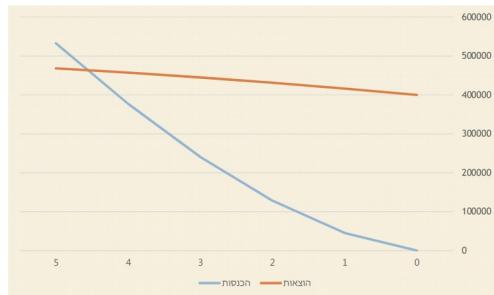
If...	It means...	Then...
$NPV > 0$	The investment would add value to the firm.	The project may be accepted.
$NPV < 0$	The investment would subtract value from the firm.	The project should be rejected.
$NPV = 0$	The investment would neither gain nor lose value for the firm.	We should be indifferent in the decision whether to accept or reject the project. This project adds no monetary value. Decision should be based on other criteria (e.g., strategic positioning or other factors not explicitly included in the calculation).

:ROI

- ראשי תיבות של Return On Investment.
- מונח המציין את היחס בין הכספי ומהשאים שהושקעו בפעולות עסקית כלשהי לבין ההכנסות שנבעו מאותם ממצים (פיתוח מערכת חדשה במקרה שלנו).
- נוסחת ROI בסיסית:

$$\text{הזרה השקעה \%} = \frac{\text{הכנסות}-\text{הוצאות}}{\text{הוצאות}} \times 100$$

:payback analysis



גיבוש צוות התיכנון של המערכת (ועדת היגוי)

ועדת היגוי היא ועדת מייעצת שהחברים בה הם בדרך כלל בעלי עניין או מומחים.

- הוועדה כוללת נציגים מתחומי ידע שונים:

- טכנולוגיה: מחשב, תשתיות, DB.
- משאבי אנוש.
- כספים.
- נציגי הלוקוחות השונים.

- הוועדה מתגבשת בשלב זה (יום) וממשיכה בעבודתה עד העלאת המערכת לאיור.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

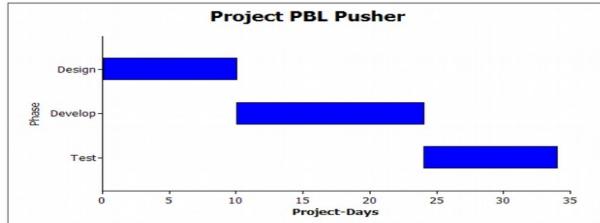
מבוסס על המציגות של אביגיל שטקל ומירב שקרון

תפקידה של ועדת הagi:

- קביעת מדיניות פיתוח.
- מעקב ובקרה אחרי הלו"ז והעלויות של המערכת.
- פתרון בעיות במהלך הפיתוח.
- שינויים בשלב הפיתוח.
- קבלת החלטות בשלבים השונים של פיתוח המערכת.
- יצירת מוערבות ומחויבות אצל הלוקחות ואנשי הפיתוח.

קבעת לו"ז ומשאבים

לוח זמנים לשכלי השינויים:



משאבים:

- משאבי חומרה ותוכנה.
- משאים אנושיים: לאפיון, לפיתוח, לבדיקות, להרצאה ולתחזוקה.

שלב הדרישות ובעל עניין

בעל עניין

בעל עניין: כל גורם המושפע מפיתוח התוכנה או אחראי באופן כלשהו על פיתוח התוכנה. האינטראס שלהם יכול להיות חיובי או שלילי.
דוגמאות לבעל עניין: משתמשים, לקוחות, מפתחים, מנהלים.
כל בעל עניין "ראה" את המערכת אחרת בעין רוחו ולכן חשוב להגיד מספר דרישות ברור.

מה זה דרישה?

- תנאי או יכולת הדורשים בעלי העניין כדי לפתור בעיה או להשיג מטרה.
- תנאי או יכולת שיש למלא או למצוא פתרון כדי לספק התcheinות, תקן, מפרט או מסמכים רשומים אחרים.
- דרישות יכולות לתאר "מה" המערכת אמורה לעשות או "איך" המערכת אמורה לעבוד.

שיטות למציאת הדרישות

ראיונות ושאלונים למשתמשים השונים, סיעור מוחות, יצרת DEMO, למידה מממשק קיימת / למידה ממצב קים, שימוש במודלים UML וכדומה.

סיכום קורס הנדסת תוכנה

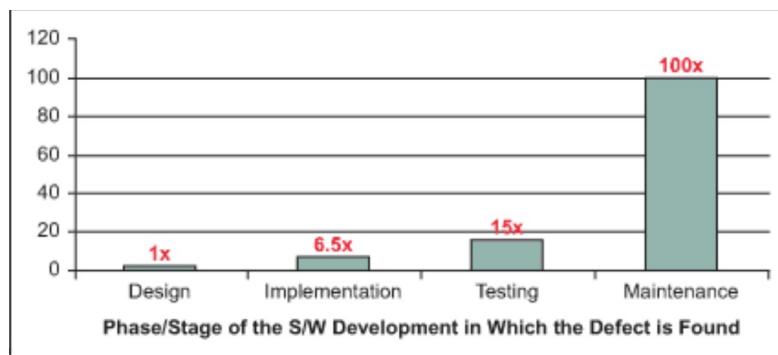
נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

חשיבות הגדלת הדרישות

- ניהול דרישות מקנה סיכון גבוה יותר שתוצרים ענו על דרישות הליקוי.
- ניהול ובקרה של הדרישות ממקדים את תהליכי האפיון והפיתוח של דרישות הליקוי.
- ניהול דרישות משפר את יכולת לבצע שינויים במרירות תוך בקרה ובחינת השפעתם על דרישות אחרות.
- תיאום ציפיות בין ספק לליקוי על בסיס מנוח, מסוכם ומואשר.
- שיפור יכולת לבצע בקרה על התקדמות הפroiיקט.
- שיפור יכולת לביצוע אמידת עלויות ע"י ירתוח של עלות תועלות עבור כל הדרישות.

עלויות תיקון שגיאות בשלבים השונים



עודף דרישות

יש לשים לב להגדיר בשלב זה רק את מה שצריך. כל דרישת משמעה עלויות כספיות, סיבוכיות למערכת וניהול של הבאים - לא נרצה לעשות את זה במקרה של דרישות לא שימושיות.

סוגי דרישות

- דרישות פונקציונליות (עסקיות) Functional Requirements מה המערכת אמורה לעשות/ להציג מנקודת המבט של המשתמש.
לדוגמה: פונקציות, שירותים.

- דרישות לא פונקציונליות (טכנולוגיות/פתרונות הפתרון) Non-Functional Requirements דרישות המגדירות תכונות נוספות של הפתרון שצריכות להתמלא תוך כדי מילוי הדרישות הפונקציונליות או דרישות ותנאים המגבילים את חופש בחירת כיווני הפתרון.
לדוגמה: זמן תגובה/ביצוע, נפח פועלות, אמינות, אבטחת מידע, אופני מימוש, תדרות ביצוע, עמידה בעומסים, שימושיות.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

mbos על המציגות של אביגיל שטקל ומירב שקרון

:Functional Requirements (עקריות)

- #### • דרישת פעולהית

דרישה המתיחסת לתפעול, לאינטראקציה או להתנהגות של המוצר.

לדוגמה: פעולות, תרחישים, תగובות לאירועים.

דוגמאות לדרישות:

- o ניתן להזמין מוצרים הן דרך מרכז הזמן והן בצורה ישירה באינטרנט.
 - o המערכת תאפשר להזין הצענות מלוקה למוצרים שבמלאי.

דרישת מידע .Data Requirement

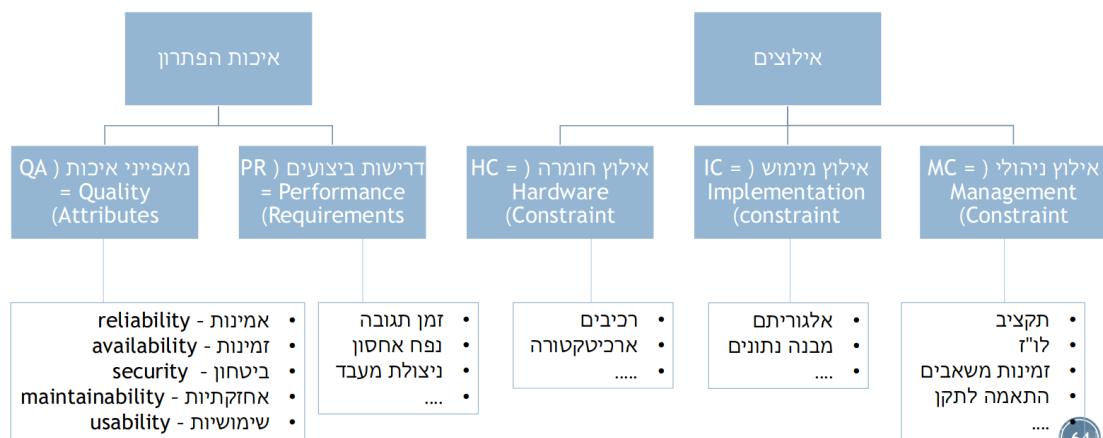
דרישה המתיחסת לשוויות המידע ולנתונים בהן נדרשות התוכנה לטפל.

לדוגמה: נתונים ומבנה נתונים, מאגרי מידע / בסיס נתונים, דרישות קלט ופלט.

דוגמאות לדרישות:

- עברו כל פריט שיצג יש להציג את שמו, הברקוד שלו וצבעו.
 - לכל פריט בהזמנה יש לרשום יחידת מידה, כמות ומחריך ליחידת מידה.

דרישות לא פונקציונליות



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מהי דרישת אינטואיטיבית?

- **辨认**: Identified (בdziada, מזוהה חד ערכית, שייכות ברורה).
- **可理解**: Understandable (ברורה, מדוייקת) – מנוסחת בשפת הלשון.
- **必要**: Necessary (הכרחית – בעלת תרומה שימושית לשיפור תהליכי העבודה).
- **完整**: Complete (שלמה).
- **一致**: Consistent (לא סותרת דרישות אחרות).
- **无歧义**: Unambiguous (חד משמעות).
- **可验证**: Verifiable (ניתנת לבדיקה באמצעות מבחני קבלה).
- **可追踪**: Traceable (גם לדרישות ברמה גבוהה יותר וגם בהמשך האפיון).
- **优先级**: Prioritized (מתועדת).

שלב הניתנות

שלב זה נקרא גם requirements specification.

- מטרת השלב: ניתוח הדרישות ומודלים
 - זיהוי הישויות הפעולות במערכת והכרת התכונות שלהם.
 - הכרת הקשרים בין הישויות.
- תוצר השלב: תרשימים שונים המתארים את המערכת.
 - class diagrams, use case diagrams

UML (Unified Modeling Language)

שפת סימולים גרפית וטקסטואלית לעיצוב מערכות מונחות עצמאיים. המודלים מתאימים הן ללקוק ולשימושים והן למפתחים.

השפה הנוכחית (2.5) כוללת 13 סוגי שונים של דיאגרמות אשר כל אחת מהן מתארת היבט אחר של המערכת המתוכננת.

יתרונות:

- התקדמות בתחום הידע של המשתמש.
- ההתאמה לדרך החשיבה של המשתמש.
- תרשימים ידידותיים יותר למשתמש, למפתח ולמנהלuproject.
- חוסר תלות בטכנולוגיה.

חסרונות:

- ריבוי מודלים וכלים עליונים לסביר את התהילה.
- מרכיבות הקשורות בין המודלים.
- קושי בשימוש במערכת מורכבת, כי קשה למצל אותה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

CASE כלים

תוכנות ייעודיות לייצור מודלים של המערכת.

Visio, Rational Rose.

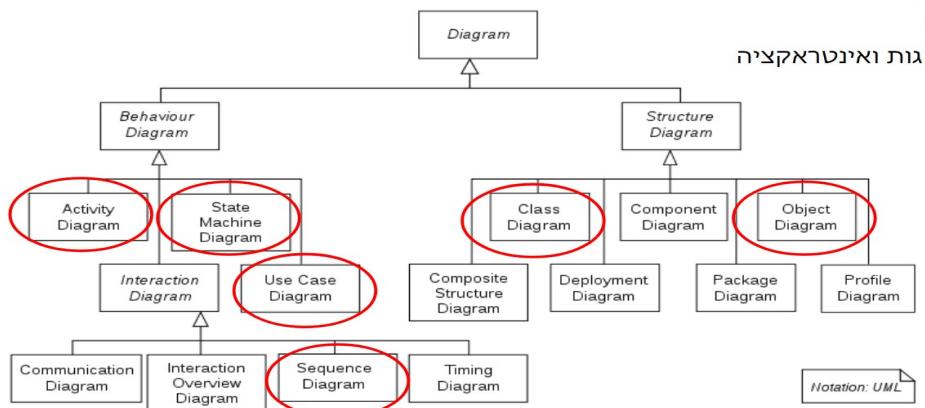
ישנים גם כלים חינמיים כמו:

* <https://www.smartdraw.com/uml-diagram/examples>

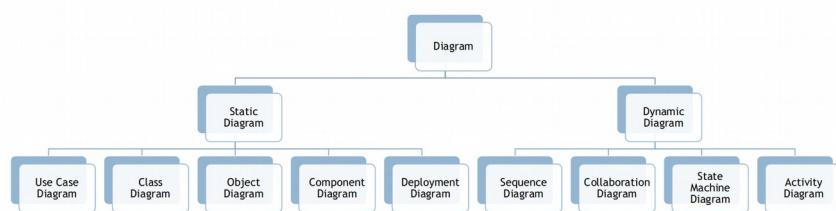
* <https://www.draw.io>

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

- דיאגרמות המתארות מבנה.
- דיאגרמות המתארות התנהלות או אינטראקציה.



ניתן לחלק לדיאגרמות סטטיות ודינמיות:



Use Case Diagram

מתארת את הפעולות שהמערכת מבצעת ויש להן תוצאות גלויות.

מראה את האינטראקציה בין דברים מחוץ למערכת למערכת עצמה.

מודל יכול להתייחס למערכת כולה או לחלקה.

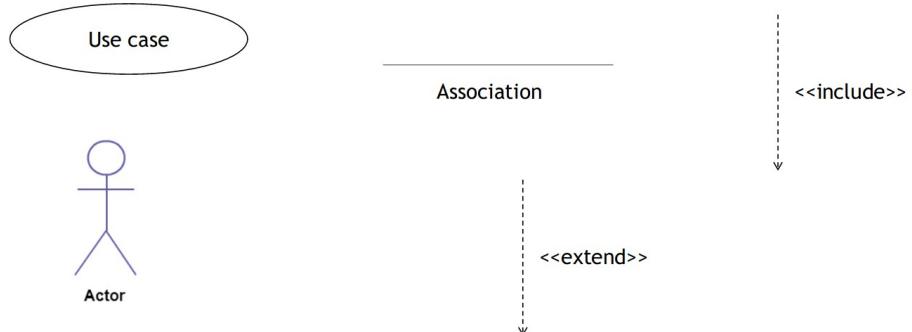
לכל דיאגרמה מסוימת תיעוד הכלול את פירוט הפעולה, השחקנים, תדרות השימוש של הפונקציה, תנאים מקדים להפעלת הפונקציה ודברים שקורים אחריו ביצוע הפונקציה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שיטות הSIMON:

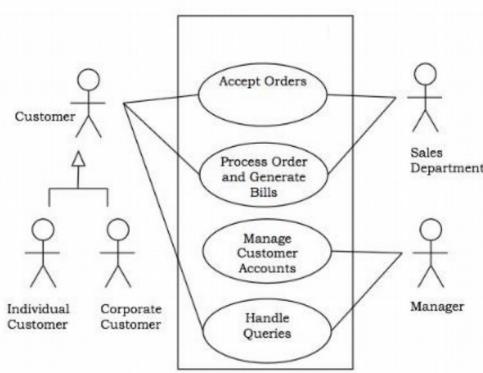


מי השחקנים ?

השחקנים הם יישות חיצונית אשר יש לה קשר עם המערכת. השחקנים מייצגים את תחומי המערכת אף אם לא חלק ממנה. שחקן יכול להיות ישות חיצונית, מערכת אחרת או כל אמצעי מחשב: עובד, לקוח, אורח, מדור בחברה, חברת משלוחים, קופה רושמת, מדפסת, מערכת כספים, ועוד...

שאלות שיכלוט למצוא את השחקנים במערכת:

Use Case Diagram למערכת הזמן



מי משתמש במערכת ?

מי מתќין את המערכת ?

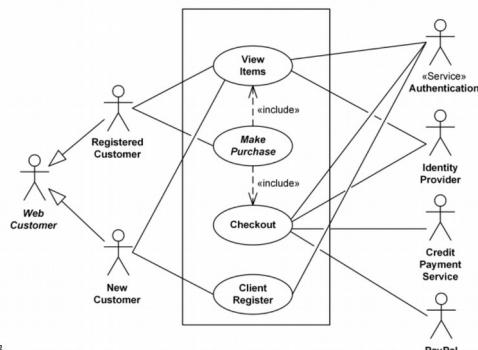
מי מפעיל את המערכת ?

מי מתחזק את המערכת ?

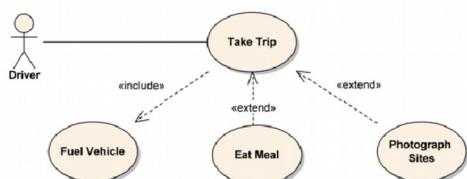
דוגמאות ל use case :

- לקוחות נרשם בקנייה למלון.
- הפקחת מול מכירות עברו מנהל המכירות.
- לקוח קונה מוצרים בסופרמרקט.
- ספק דרוש דרישת לתשלום.

Use Case Diagram למערכת הזמן



<<extend>> & <<Include>>



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

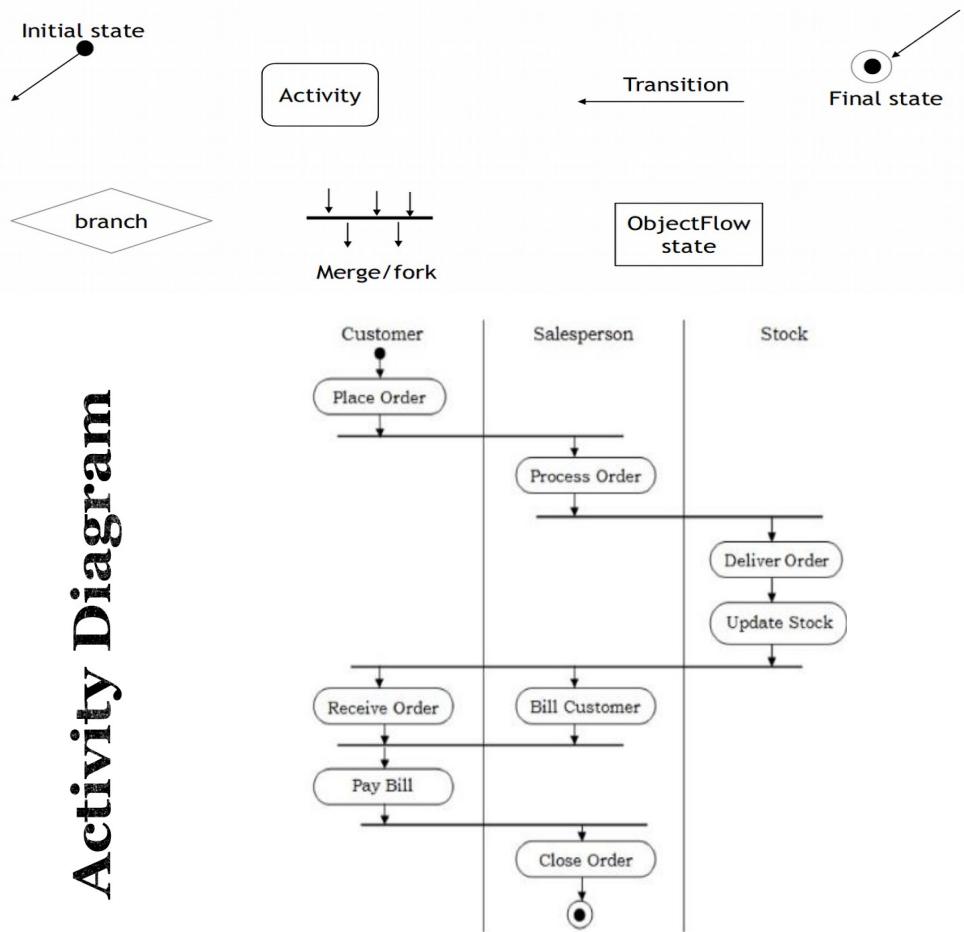
Activity Diagram

תרשים פעילות מציג את הזרימה מפעילות בתחום המערכת ובין השחקנים.

דרך להציג פעילות במודול בדוגמה ל-workflow.

כל פעולה בuse case הופכת לתרשים פעולה אחד.

שיטת הסימון

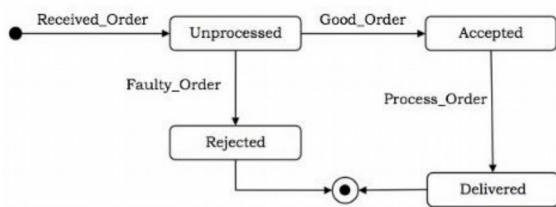


סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

State Machine Diagram



State Machine Diagram

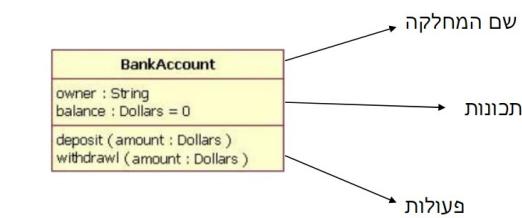
בתרשים מצבים שמים את האובייקט במרכז.
 התרשים יכול לתאר את זרימת האובייקט ממקום אחד
 לשני ואת הממצאים השונים שבו האובייקט נמצא
 להיות לאורך ח' המערךת.
 תרשימים זה מבוסס על האוטומט של פרופסור דוד
 הראל.
 כל תרשימים מתאר מחלוקת אחת.

שיטת הסימון



Class Diagram

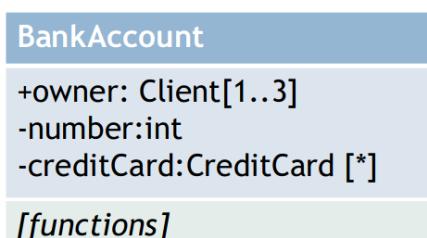
מודל של בניית המערכת על ידי מידול המחלקות, התכונות והפעולות.
 הוא תכנית האב של המחלקות הדרישות לבניית התוכנה.
 מתכוונים לפתחים את המערכת בהתאם על מודל המחלקות שהוגדר.
 זה המודל הנפוץ ביותר ב-UML.



סימון מחלוקת פשוטה:

- סימונים:
 - private
 - + public
 - # protected
 - ~ package

ניתן לסמן עם דרך דיפולטיבי על ידי השמת שיוון והערך אחרי שם המשתנה.



: Multiplicity
 מיד לאחר ציון ה type של המשתנה ניתן לציין את ה multiplicity שלו.

את ה multiplicity מצינים באמצעות מתן טווח ערכים אפשרי.

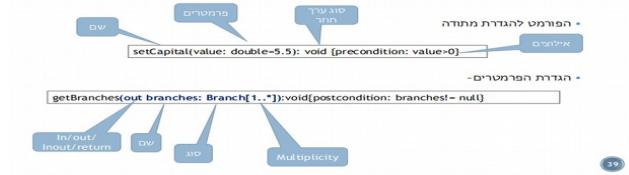
כאשר לא מצינים את ה multiplicity של משתנה ברירת המחדל היא 1.

כאשר משתמשים בסימן * כדי לציין את ה multiplicity אחד או יותר.

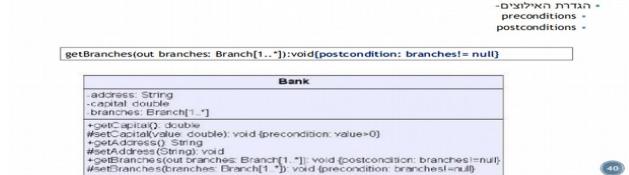
סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור
mboson על המציגות של אביגיל שטקל ומירב שקרון

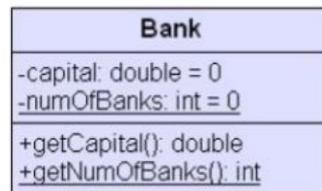
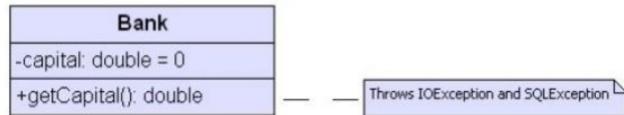
מетодות שמודדרות במחלקה (Operations)



מетодות שמודדרות במחלקה (Operations)



תיאור תקלות שאפשרות להתרחש:
כאשר בקריאה להפעלת METHOD יש סכנה שתתרחש תקלה (exception) מקובל להוסיף note בו תיאור החריגה.

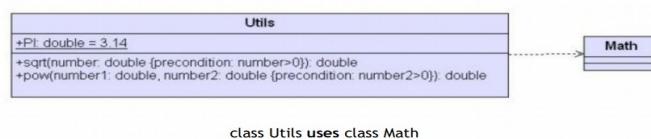


מетодות סטטיות:

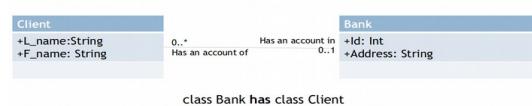
METHOD STATIC מסומן עם קו תחתון.

קשר של Dependency

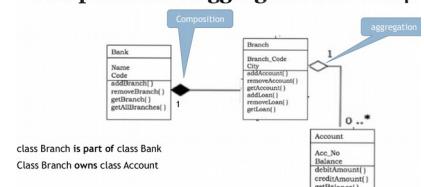
קשרים בין מחלקות:



קשר של Association



קשרים של Composition ו Aggregation

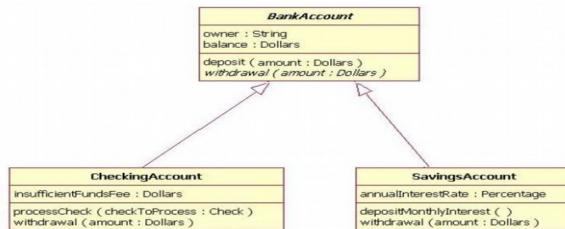


סיכום קורס הנדסת תוכנה

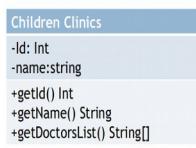
נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

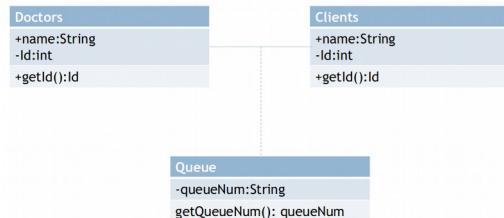
הורשה



תיאור Interfaces ומחלקות שימושיים אותם

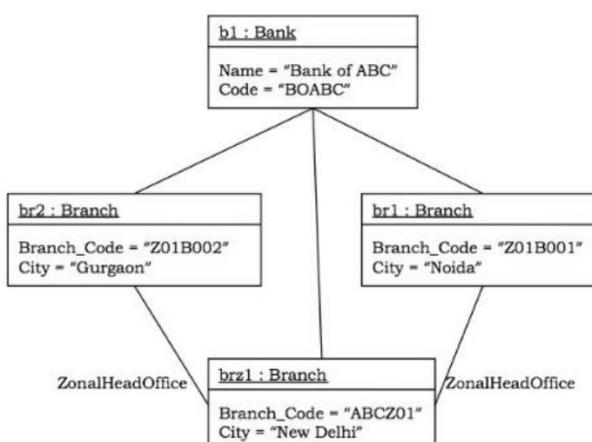
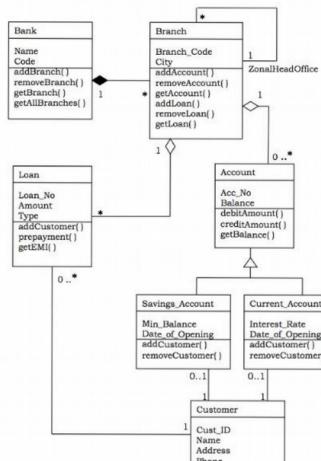


תיאור קשר בין מחלקות באמצעות Class Association



Class Diagram

www.tutorialspoint.com/



Object Diagram

דיאגרמת אובייקטים מתארת מופע של מודל המחלקות, בדומה לתרחישים האמיתיים שעיל הוא מודל סטטי (בדומה למודל המחלקות). דיאגרמת אובייקטים בסיסם בונים את המערכת. דיאגרמת אובייקטים היא שימוש במודל האובייקטים דומה למודל המחלקות רק שהםאפשרים בניית אובייקטים של המערכת מנוקדת מבט מעשית.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

Object diagram	Class diagram	
מכיל 2 חלקים: שם ורשימת פעולה מאפיינים	מכיל 3 חלקים: שם, רשימה תוכנות ורשימת המחלקה	מבנה
+ הפורט מורכב משם האובייקט + נקודותיהם + שם המחלקה (Tom:Employee)	שם המחלקה עומד בפני עצמו בחלק של שם המחלקה	שם המחלקה
מגדיר את הערך הנוכחי של כל תוכנה	מגדיר את התוכנות של המחלקה	רשימת התוכנות
לא כלולות	לא כלולות	רשימת הפעולות
מוגדר הרקשור בין מחלקות - שם הקשר וכמוות (לא רלוונטי כשמדוברים על ישות בודדת)	מוגדר הרקשור בין מחלקות - שם הקשר וכמוות הקשר.	קשרים

Sequence Diagram

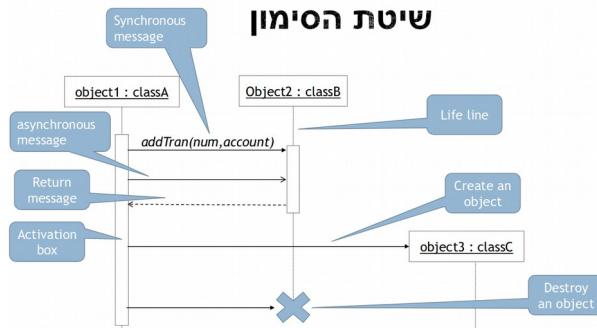
תרשים רצף הממחיש את סדר הפעולות ברגע הזמן.

תרשים רצף נכתב בצורה של תרשימים זו מימדי:

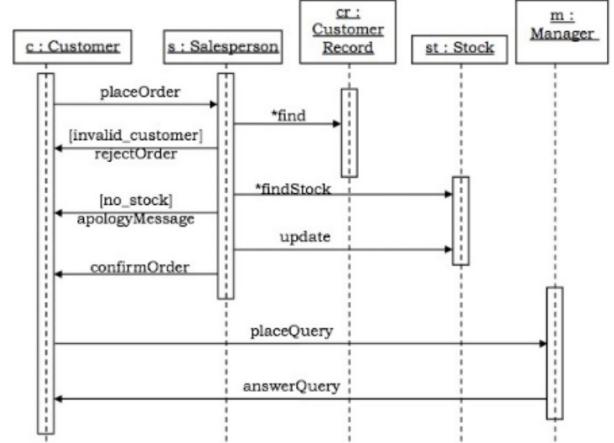
- על ציר ה-X נמצאים האובייקטים.
- על ציר ה-Y מוקומות ההודעות שהאובייקטים האלה שולחים.

לכל פונקציונליות נכון תרשימים רצף נפרד.

שיטת הסימון



Sequence Diagram



Activity	Sequence
Diagrammatic representation of interactions	Diagrammatic representation of interactions
Establishes initial context for the system	Establishes initial context for the system
Performs specific actions or calculations	Performs specific actions or calculations
Handles errors or exceptional cases	Handles errors or exceptional cases

אז מהו סדר הנכון ??

אנו!

אבל סדר הגיוני הוא:

- use case diagram
- state machine diagram
- activity diagram
- class diagram
- object diagram
- sequence diagram

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

ERD – Entity Relation Diagram

מודל תפיסתי ל-DB נפוץ ומקובל. פותח על ידי peter chen ב-1976. זה מודל גרפי המציג את מערכת המידע כאוסף של יישויות ושל קשרים בין היישויות.

ישות Entity

ישות מייצגת במודל אובייקט ממשי או מופשט, שיש לו קיום למציאות והוא בעל משמעות וענין בהקשר מסוים. לדוגמה:

עצם: כמו בניין, ספר, מכונית, מכונה במפעל וכדומה.

גוף חיה: עובד במפעל סטודנט, חוללה בבית חולים.

מושג מופשט או רעיון: קורס, מבחן ניהגה, טיסה, תעסוקה.

אריווע: דיווח הוכחות, תנועה בחשבון בנק, כניסה פריט למלאי.

לישות יש מופעים: רשימת העובדים, רשימת הפריטים וכדומה.

ישות חזקה וישות חלשה

ישות חזקה: מוגדרת כישות שקיומה העמצעי במודל אינו תלוי בקיומה של אישות אחרת.

ישות חלשה: מוגדרת כישות שקיומה העצמאי במודל מותנה בקיומה של ישות אחרת.

דוגמאות:

לקוק, רפואי יכולים להחשב לישויות חזקות.

טור, מרשם ללקוק נחשבים לישויות חלשות.

שלבי עיצוב בסיס הנתונים

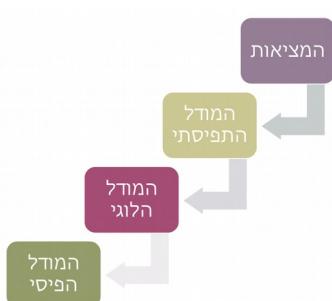
המציאות:

"באריאותא"

клиיטה של החולים, רפואיים חדשים,

ניהול תורים, הזמנות מבטי המركחת, קשר ביןלקוק לרופא, שיבוץ רפואיים במרפאות.

מי הם הישויות? לוחקות, רפואיים, תורים, בתים רפואיים, מרפאות.



המודל התפיסתי:

צוג מבנה הנתונים באמצעות מודל אבסטרקט (ללא מונחי מחשב). משמש ככלי קומוניקציה – בין לקוחות למשתמשים, בין לקוחות למפתחים.

תכונות רצויות למודל תפיסתי:

Expressive: אפשר ביטוי למגוון נתונים, קשרים ואילוצים.

Simplicity: קל להבין גם עבור לא-מקצוענים (משתמשים).

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

. Minimalism: קולל מספר קטן של מונחי יסוד שנובנים פשוט וברור.

Diagrammatic representation: ניתן לתיאור על ידי תרשים פשוט.

Formality: ניתן לתיאור פורמלי/מדויק.

מיפוי מדויק: (אלגוריתמי) לסתכמה לוגית של בסיס הנתונים.

הציגות ישות במודול

תוכנה attribute: מוגדרת כמאפיין כלשהו של אובייקט שהוא בעל משמעות וענין במודול.

דוגמה לאובייקט ללקוח: שם, תאריך לידיה, כתובת, טלפון.

תוכנה פשוטה: מוגדרת כתוכנה המכילה מרכיב אחד בלבד ואניינה ניתנת לחילוקה נוספת.

תוכנה מורכבת: מוגדרת כתוכנה המכילה מספר מרכיבים ונitinן לחלק אותה למרכיביה.

לדוגמה:

שם המרפאה – תוכנה פשוטה.

כתובות הלקוות – תוכנה מורכבת, ניתן לחלק לרחוב, מספר בית, עיר ומיקוד.

תוכנה מחושבת: מוגדרת כתוכנה אשר ערכה הינו תוצאה של חישוב נתונים אחרים במערכת.

לדוגמה: ממוצע טיפולים לרופא.

ערך של תוכנה: מוגדר כתוכן התוכנה בנקודת זמן מסוימת.

לדוגמה: boolean, decimal, integer.

תוכנה עם ערך בודד מכילה בכל נקודת זמן ערך אחד בלבד. תוכנה מרובה ערכים יכולה לקבל

בכל נקודת זמן ערך אחד או יותר.

בחזרה ל"בריאות", מהם התכונות של היישויות?

• ללקוחות: תעוזות זהות, שם פרטי, שם משפחה, כתובות.

• מרפאות: שם רפואי, כתובות מחוץ.

• בתים מקרקעין: שם בית מקרקעין, כתובות, שם מנהל, טלפון.

מפתח של ישות key primary key

מפתח עיקרי: מוגדר zusätzlich המזוהות באופן חד חד ערכי מופיע של ישות מסוימת.

דוגמאות: בלוקות תעוזות זהות היא המפתח.

מפתח פשוט לעומת מפתח מורכב: כאשר המפתח של קבוצת היישויות מורכב מתוכונה אחת בלבד,

מקובל לומר שהוא מפתח פשוט, כאשר הוא מורכב מספר תכונות מקובל לומר שהוא מפתח

מורכב.

לדוגמה: בישות "תור לרופא" המפתח מורכב מספר התור, מספר הלקוות, יום ורופא.

מפתח זר foreign key

מפתח זר: מוגדר כתוכנה המופיעה בישות E1, המשמשת גם כמפתח עיקרי בישות E2.

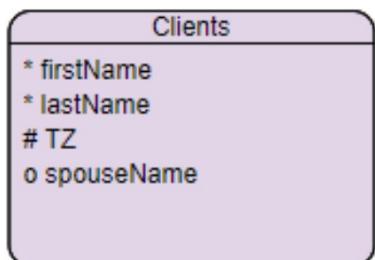
כלומר $FK(E1) = PK(E2)$

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

לדוגמה בישות "מרפאות" התוכנה מספר מחוץ היא מפתח זה. (בישות מוחזות היא מפתח ראשי).



בחזרה לבריאות מהם המפתחות של השיוויות ?

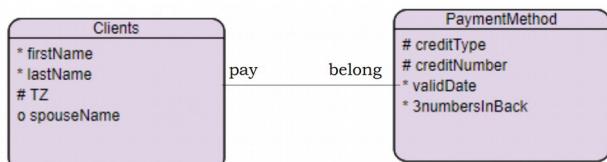
- "לקוחות": ת.זהות
- "מרפאות": קוד מרפאה.
- "תורים": ת.ז.לקוח, מספר רופא, תאריך ושעה.
- "רופאים": מספר רופא.

קשרים Relationship

קשר: מוגדר כיחס בעל משמעות בין ישויות שונות.

פונקציונליות הקשר: מוגדרת כסוג המיפוי הקיים בין הקבוצות המשותפות בקשר. הפונקציונליות יכולה להיות מסוג 1:1 , M:1 , M:N

לדוגמה: עבור הישויות מרפאה ולקווח קיימם קשר , לקוח שייר למרפאה אחת, במרפאה יש הרבה לקוחות. עבור הישות מזכירה ועמדת עבודה קיימם קשר – מזכירה עובדת בעמדת עבודה ועמדת עבודה שייכת למזכירה.



קשר חד חד ערכי 1:1

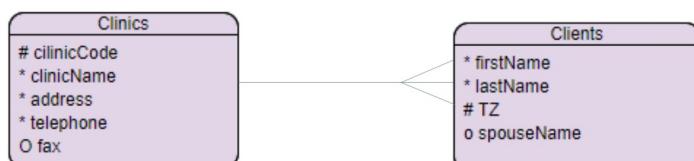
ישות אחת מתאימה לשווות אחרת.

לדוגמה: לכל לקוח מוגדר אמצעי תשלום ייחיד וכל אמצעי תשלום שייר לךות אחד בלבד.

קשר חד רב ערכי (1:M)

ישות אחת מתאימה לכמה ישויות בקבוצה אחרת.

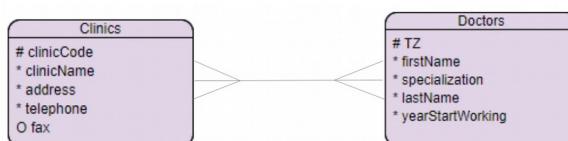
לדוגמה: כל לקוח משוייך למרפאה אחת ובכל מרפאה יש הרבה לקוחות.



קשר רב רב ערכי (N:M)

ישות אחת מתאימה לכמה ישויות בקבוצה אחרת, וישות בקבוצה האחרת מתאימה לכמה ישויות בקבומה הראשונה.

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



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

קרדיינליות הקשר Relationship cardinality

קרדיינליות הקשר: מוגדר כמספר היחסות המינימלי והמקסימלי בקבוצת ישות A הקשורות לשישות אחרת בקבוצת ישות B.

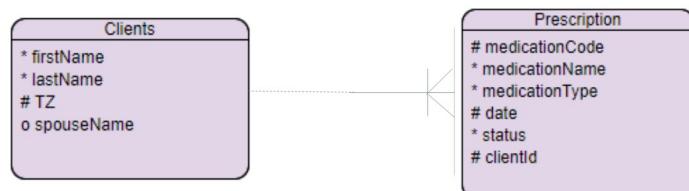
לדוגמה: רופא משפחה יכול לטפל בעד 100 לקוחות.



תלות קיומית existence dependence

תלות קיומית: בין קבוצה A לקבוצה B מוגדרת מצב שבו קיומ ישות בקבוצת ישות A מותנה בקיום ישות בקבוצת ישות אחרת B.

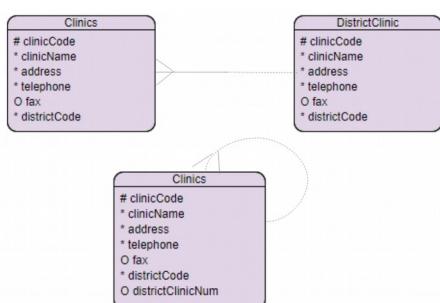
לדוגמה: לטבלה "מרשםים ללקוח" אין משמעות במידה והלקוח עזב את הקופה וכבר לא במושת בה.



קשר רקורסיבי recursive relationship

קשר רקורסיבי: מוגדר כקשר המחבר בין קבוצה A לעצמה.

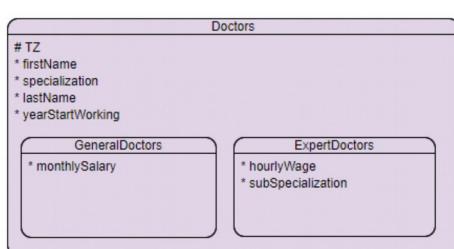
לדוגמה: מרפאה מחודשת.



ישות על ותת ישות

תת ישות היא סוג של ישות אשר יורשת את מאפייני ישות העל.

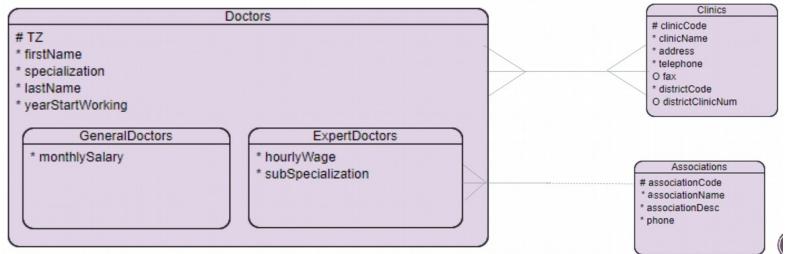
לדוגמה: כל רופא חייב להיות או רופא כללי או רופא מומחה.



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

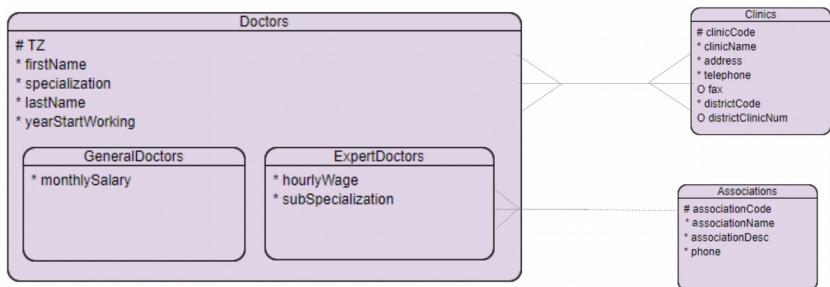
מבוסס על המציגות של אביגיל שטקל ומירב שקרון



קשר של ישות הуль הינו גם קשר של תת הישות.

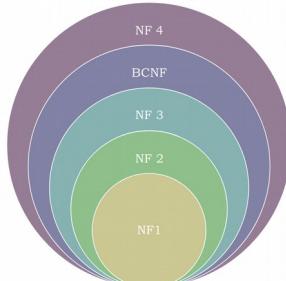
קשר של תת ישות מהוועה קשר שלה עצמה בלבד.

יחס בחירה בין קשרים: יחס בחירה בין מספר קשרים מוגדר כבחירה בקיים קשר אחד בלבד מתוך מספר אפשריים.



נормול - תזרורת

נормול: התליר שמטרתו לייצר אוסף של טבלאות שאין כוללות תופעות לא רצויות הנובעות מכפילותות נתונים או אנומלייה.



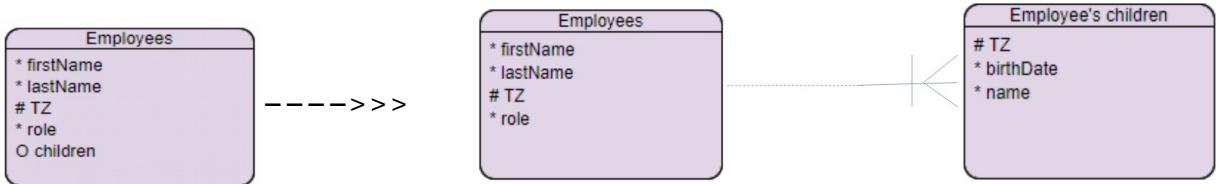
רמת נרמול	תיאור
1 NF	כל תוכנה יכולה לקבל ערך אחד בלבד, ללא קבוצות
2 NF	תוכנה שאינה מושגעה באך אחד מהמפתחות לא יכולה להיות תלולה בתת קבוצה ממש של מפתח
3 NF	תוכנה שאינה מושגעה באך אחד מהמפתחות לא יכולה להיות תלולה בקבוצה שאינה סופר-מפתח
BCNF	תוכנה לא יכולה להיות תלולה בקבוצה שאינה סופר-מפתח
4 NF	לא תלויות רב ערכיות. כל ישות צריכה להחזיק 'רעיון' אחד בלבד

סיכום קורס הנדסת תוכנה

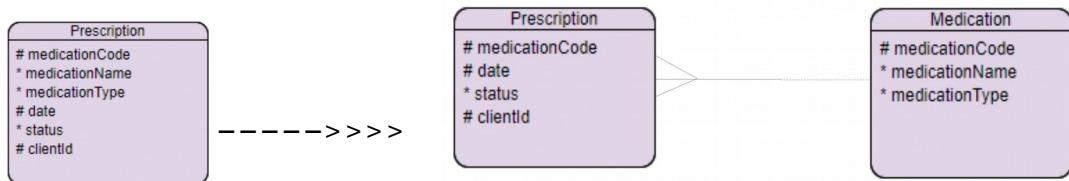
נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

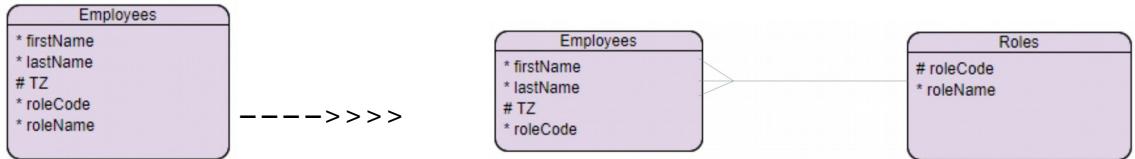
נרטול :1NF



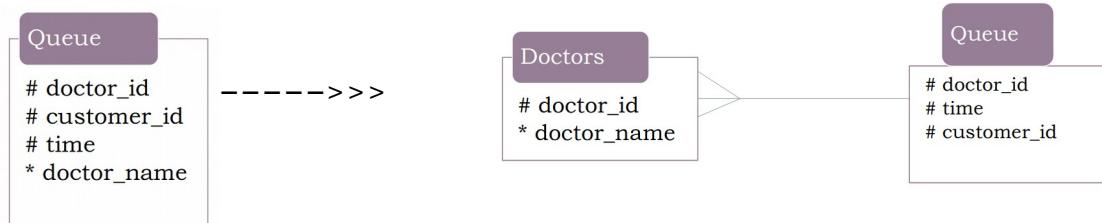
נרטול :2NF



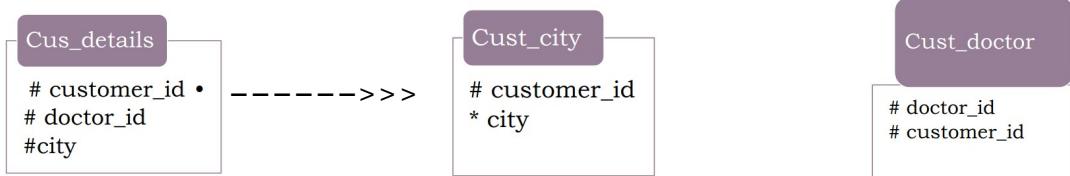
נרטול :3NF



נרטול :BCNF



נרטול :4NF



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

המודל הלוגי:

שלבי העבודה:

- מיפוי יישיות פשוטות לatable.
- מיפוי התכונות לעמודות בטבלה.
- מיפוי המפתחות הראשיים.
- הפיכת הקשרים למפתחות זרים.
- תרגום קשרות.
- תרגום יישיות על ותתי יישיות.

מיפוי יישיות פשוטות לatable: פשוט, כל ישות מקבל טבלה.

מיפוי התכונות לעמודות בטבלה, מיפוי המפתחות הראשיים:

תמונה עם * מסמן כ null not-NN
תמונה עם # מסמן עם (U) unique (NN) not null
במפתח מורכב – נגדיר U לכולם ביחד.

הפיכת קשרים למפתחות זרים:

קשר יחיד לרבים – המפתח הזר יופיע בטבלת הרבים.

קשר יחיד ליחיד – אם 2 צידי הקשר הם חובה, מיקום המפתח הזר יכול להיות בכל אחת מהטבלאות. אם אחד מצד אחד הראה רשות והשניה חובה, המפתח הזר צריך להיות לצד של החובה.
קשרים רבים לרבים – לא רלוונטי. בשלב זה גם לא יהיו קשרים כאלה.

אם הקשר הוא חובה, עמודת המפתח הזר מסמן כ null not

אם הקשר הוא חלק מהמפתח הראשי, מסמן את העמודה ב PK.

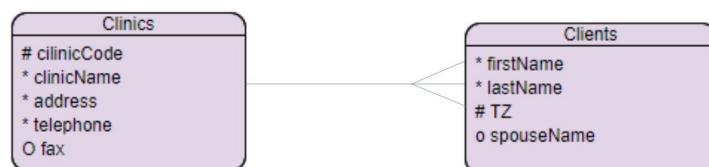


Table name: Clients

Column name	firstName	lastName	TZ	spouseName	clinicCode
Key type			PK		FK1
Null / unique	NN	NN	NN, U		NN

Table name: Clinics

Column name	clinicCode	clinicName	address	telephone	fax
Key type	PK				
Null / unique	NN, U	NN	NN	NN	

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שלב המימוש

פעולות בשלב המימוש:

- פיתוח התוכנה: התקנת סביבת פיתוח התוכנה, קידוד התוכנה, תיעוד התוכנה.
- בדיקת התוכנה: בדיקה של יחידות התוכנה השונות ובדיקת התוכנה של המערכת כולה.

עקרונות לכתיבת קוד "נכון"

ומה בכלל חשוב לכתוב קוד נכון? מכיון שכאשר כתבים קוד נכון קל לפתור בעיות, מבזבזים פחות זמן על התחזקה של המערכת, הקוד שלנו יותר ברור לקרוא.

עקרון ה **KISS**: פשוטות פשוטות פשוטות, הביטוי הוטבע על ידי קל ג'ונסון, מהנדס מטוסים.

• שמות של משתנים ומחלקות:

- שמות משמעותיים. לדוגמה wagesPerHour לעומת wph.
- לשמר על עקביות בשיטת מתן השמות.
- למשתנים זמניים (כמו מונה בלולאה) כדאי לתת שמות קצרים (j,i,...).

IDENTIFIER	NAMING RULES	EXAMPLE
Variables	A short, but meaningful, name that communicates to the casual observer what the variable represents, rather than how it is used. Begin with a lowercase letter and use camel case (mixed case, starting with lower case).	mass hourlyWage isPrime
Constant	Use all capital letters and separate internal words with the underscore character.	BOLTZMANN MAX_HEIGHT
Class	A noun that communicates what the class represents. Begin with an uppercase letter and use camel case for internal words.	class Complex class Charge class PhoneNumber
Method	A verb that communicates what the method does. Begin with a lowercase letter and use camelCase for internal words.	move() draw() enqueue()

• כתיבת הערות:

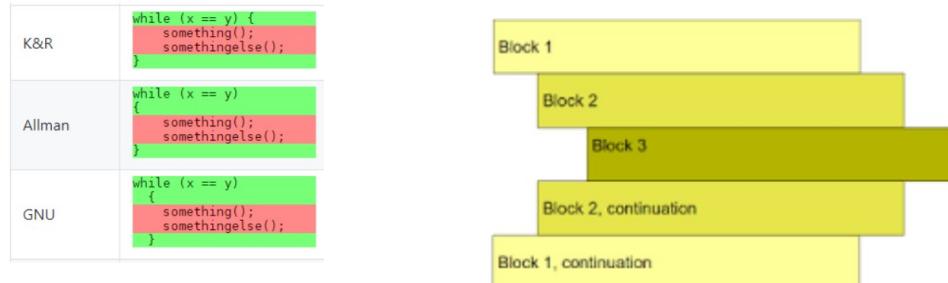
- חשוב במידה.
- הקוד-Amor להסביר את עצמו, הערות אמורים להסביר את הלמה.
- לא להשאיר קוד ישן בהערות.
- בראש המספר יש להוסיף הערה מי כתב את הקוד, מתי הוא נכתב ומה הוא אמר לעשות.
- מומלץ doc comments – כל מגיע ב'יחד עם ה-JDK. אפשר לנו ליצור תיעוד של הקוד שלו בפורמט של HTML. כתבים את הערות בתוך הטוויח שמתוחים עם */ ומסתיים ב/*.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

- הזרחות: גם הזרה היא לא חובה (ב JAVA) בהגדירה של השפה, אבל כדאי לנו להקפיד עליה.
ש שיטות הזרה רבות, כדאי לנו לשמר על עקביות בשיטה הנבחרת.



ישנים 4 פרדיגמות תכנות מרכזיות:

- פרדיגמת הציווי: imperative paradigm
- פרדיגמת הפונקציונליות: functional paradigm
- פרדיגמה לוגית: logical paradigm
- פרדיגמת מונחה עצמים: object oriented paradigm

פרדיגמת הציווי: imperative paradigm
 כתיבת רשות ציוויים commands. Katz דומה לכתיבה לתוכן. זהה כתיבה המדמה את הזרה בה המחשב פועל.
 שפות: Fortran, Algol, Pascal, Basic, C

פרדיגמת הפונקציונליות: functional paradigm
 פרדיגמה המגיעה מתחום המתמטיקה, תורה הפונקציונליות.
 הערכים המופקים בה אינם ניתנים לשינוי.
 בלתי אפשרי לשנות כל מרכיב בעל ערך מורכב.
 כל החישובים נעשים על ידי יישום (קריאה) של פונקציות.
 ההפשטה הטבעית היא פונקציה.
 השפות התומכות בפרדיגמה זו: ML, Haskell, Scala, Erlang, Lisp, ועוד

Answer a question via search for a solution

פרדיגמה לוגית: logical paradigm
 הפרדיגמה מאוד שונה מהאחרות.
 הci מתאימה ליישום כאשר צריך להוציא מידע מעובדות בסיסיות או יחסים.
 הוכחות אוטומטיות עם ביןיה מלאכותית.
 מבוסס על אקסומות, כללי הסקה ושאלות.
 חיפוש שיטתי במערכת עובדות, תוך שימוש במערכת כללי הסקה.

שפות התומכות בפרדיגמה: Prolog, Mercury, Alice
 הפרדיגמה הci נפוצה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

טוביה מאד לתוכניות גדולות.. תורה המושגים, מודלים של אינטראקציה אנושית עם תופעות בעולם האמתי.

נתונים ופעולות עוטופים באובייקטים.
 הסתרת מידע להגנה על תוכנות של אובייקטים.
 עצמים מקיימים אינטראקציה באמצעות הודעות.
 הקלאסים מאורגנים בהיררכיות ירושה.

Send messages between objects to simulate the temporal evolution of a set of real world phenomena

SOLID – עקרונות לתוכן יציב

- Singel responsibility principle •
 כל מחלוקת אחראית על פונקציונליות אחת, ואחראית עליה בצורה מלאה. –> לכל מחלוקת ש סיבה אחת להשתנות.

דוגמא

```
/// <summary>
/// Class calculates the area and can also draw it
/// on a windows form object.
/// </summary>

public class RectangleShape
{
    int height;
    int width;
    public int getHeight()
    {
        return height;
    }
    public int getWidth()
    {
        return width;
    }
    public void setHeight(height)
    {
        this.height = height;
    }
    public void setWidth(width)
    {
        this.width = width;
    }
    public int Area()
    {
        return Width * Height;
    }
    public void Draw()
    {
        ....
    }
}
```

שוחח חישוב
ארוחות
פירח צירוף

שוחח חישוב
ארוחות
פירח צירוף

פתרון

```
public int Area()
{
    return Width * Height;
}

// <summary>
// Class calculates the rectangle's area.
// </summary>
public class RectangleShape
{
    int height;
    int width;
    public int getHeight()
    {
        return height;
    }
    public int getWidth()
    {
        return width;
    }
    public void setHeight(height)
    {
        this.height = height;
    }
    public void setWidth(width)
    {
        this.width = width;
    }
}
```

מחלוקת צירוף
הווראה

- Open close principle •
 המחלוקת צריכה להיות פתוחה להתרחבות אבל סגורה לשינויים. –> במקרה של שינויים, נוסף תת מחלוקת הממשים אותו.

דוגמא

```
class Shape
{
    private Point _center;
    /* Center Get / Set*/
};

class Circle extends Shape
{
    private int _radius;
    /* Radius Get / Set*/
};

class Square extends Shape
{
    private int _side;
    /* Side Get / Set*/
};

class DrawShapes {
    // Draw all registered shapes
    public void drawShapes()
    {
        // This method is not conformed to the Open Closed Principle
        for(Shape s : _shapeList)
        {
            if (s instanceof Circle)
            {
                .drawCircle((Circle) s);
            }
            else if (s instanceof Square)
            {
                .drawSquare((Square) s);
            }
        }
    }
}
```

פתרון

```
abstract class Shape
{
    private Point _center;
    public abstract void Draw();
};

class Circle extends Shape
{
    private int _radius;
    @Override
    public void Draw()
    {
        ...
    }
};

class Square extends Shape
{
    private int _side;
    @Override
    public void Draw()
    {
        ...
    }
}
```

```
class DrawManager
{
    .....
};

public void drawShapes()
{
    for (Shape s : _shapeList)
        s.Draw();
}
```

- Liskov substitution principle •
 אם S היא תת מחלוקת (ירושת) של Z אז ניתן להחליף כל מופיע של Z במופיע של S מבלי שההתנהוגות תשתנה. –> מחלוקת ירושת יכולה לשנות את התנהוגות של מחלוקת האם. מי שפונה למוגדרת במחלוקת האם אינו אמור לדעת לאיזו מהמחלקות הבנות הוא פונה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

• Interface segregation principle

אין להכריח ל��ח להיות תלוי במשק שהוא אינו משתמש בו. –> יש להחליף משק "שם"
 במשקים "רזים", כל אחד מותאם ל��וח ספציפי.

• Dependency inversion principle

מודלים ברמה גבוהה אינם צריכים להיות תלויים במודלים ברמה נמוכה. –> שניהם צריכים להיות תלויים באבסטרקציה.
 אבסטרקציות אינן.CRITICAL. –> פרטיטים צריכים להיות תלויים באבסטרקציות.

```
class Message {  
}  
  
class GmailBox {  
    public Message[] getMessages() {  
        return new Message[5];  
    }  
}  
  
class MailPrinter {  
    private GmailBox mailbox;  
  
    public MailPrinter(GmailBox gmailBox) {  
        this.mailbox = gmailBox;  
    }  
  
    public void print() {  
        for(Message message : mailbox.getMessages()) {  
            System.out.println(message);  
        }  
    }  
}
```

דוגמה

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

פתרון

```
class GoodMailPrinter {  
    private MailBox mailbox;  
  
    public GoodMailPrinter(MailBox mailbox) {  
        this.mailbox = mailbox;  
    }  
  
    public void print() {  
        for(Message message :  
            mailbox.getMessages()) {  
            System.out.println(message);  
        }  
    }  
}
```

Code review

זהה בדיקה שיטתיות על קוד אשר נכתב לתוכנית או למערכת בצד' למצוא ולתקן בעיות שנמצאות בקוד עוד בשלבי פיתוח המוקדמים. הבדיקה נעשית על ידי עמייתים ולא על ידי המתכנת עצמו.

מטרות: מציאת בעיות ותיקונים, שיפור תהליכי ביצוע הפיתוח, שיפור יעילות ומורכבות הקוד, עוזר לשיתוף מידע בתוך הוצאות. מה מתרחש? קוד מת, בעיות יעילות ומורכבות, שימוש חוזר בקוד, דיליפט זולגת מידע, דיליפט זיכרון. יתרונות:

- מעבר על הקוד ומציאת בעיות בשלבים מוקדמים במחזור חי' המוצר/הפרוייקט.
- מציאת בעיות שלא ימצאו על ידי בדיקות רגילים. לדוגמה, אי שימוש במשתנים.
- צמצום הבעיות כתוצאה מראייה נוספת נספתח אשר משפיעה על איכות המערכת.
- אזהרה מוקדמת לגבי היבטים וחלקים חשודים בקוד שניtan לאמוד אותם רק על ידי מדידה או מורכבות.
- שיפור רמת התחזקה של הקוד לשינויים עתידיים.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

mbossum על המציגות של אביגיל שטקל ומירב שקרון

בדיקות (שלב המימוש)

הגדרה

בדיקות הן מכלול תהליכי, שיטות, פעילויות וכליים המלאים את פיתוח המערכת והמצרים לאחר מחזור החיים מתוך כוונה לבצע אימות ווחchtת תקיפות (Verification&Validation) לשם התאמת התוצרים לדרישות הלוקח, למפרטים הטכניים, לתקנים ולנהלים מחייבים. Validation: האם אנחנו מפתחים נכון המוצר. Verification: האם אנחנו מפתחים את המוצר שנדרשנו לפתח.

למה לבדוק ?

לבודק שהתוכנה אכן עושה את מה שהיא אמורה לעשות (=דרישות פונקציונליות).

לבודק שהיא עושה את זה עם כמה שפחות שגיאות (איקות).

לבודק שהיא עושה את זה בצורה טובה מבחינת עצמה בדרישות ביצועים, ובעומסים ונՓחים (=דרישות לא פונקציונליות).

מטרה עיקפה: לאסוף רישום של שגיאות תוכנה לצורכי מניעת שגיאות עתידיות (פעולות מונעות או פעולות מתקנות).

מי בודק ?

colsms!!!

תוכניתן כותב את הקוד שהוא כתב (בסביבת הפיתוח).

תוכניתנים עמיתים בודקים את הקוד במסגרת code review.

צוות הבדיקות בודק את הקוד שנכתב (בסביבת הבדיקות).

הלוקח בודק את המערכת.

בדיקות מסירה ובדיקות קבלה

בדיקות מסירה:

מכלול הבדיקות הנעשות בתהילך הפיתוח טרם מסירת המוצר לבדיקות הקבלה של הלוקח. הבדיקות הן באחריות צוות הפיתוח והבדיקות.

בדיקות קבלה:

בדיקה כוללת של המערכת לפני הפעלה, במטרה לוודא עמידה בדרישות והתאמת המערכת לצרכי הלוקח. הבדיקות הן באחריות הלוקח / משתמש.

בדיקות מבוצעות בסביבת הלוקח או בסביבת pre-production.

שלבי בדיקות המסירה



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

היבטי בדיקות באפליקציות לנויידים

תהלייך הבדיקות של אפליקציות לנויידים הוא מאוד מורכב!

מכיוון ש: קיימים למעלה מ-250 מכשירי אנדרואיד בשוק, כ-10 גרסאות של מערכות העפלה בשימוש, מגוון גודל של מסכים, מסכי מגע ומקלדות פיזיות, כ-63 פעולות סולולריות שונות שתומכות באנדרואיד.

בשביל לבדוק את האפליקציה שלנו כמו שצריך נctrיך לבדוק את כל הוויאציות!!

היבטי בדיקות באפליקציות לנויידים

כמובן שעבודת בדיקות זאת היא לא הגיונית ולכן גנטה להתמקד בדברים המרכזיים:

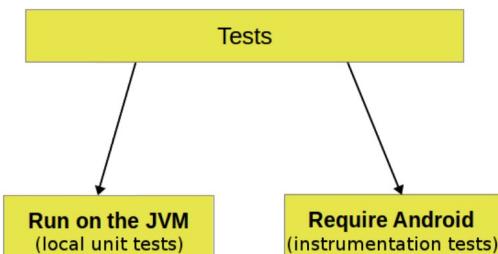
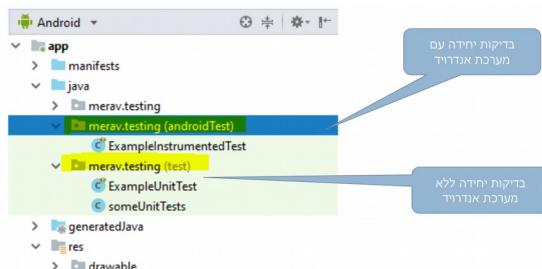
- מכשירים: גנטה להתמקד בסוגי המכשירים הזמינים באיזור היעד שלנו.
- גרסאות OS: נתמקד בגרסאות הנפוצות ביותר ונתן להם עדיפות.
- גודל המסך: באנдрואיד יש 4 אפשרויות של גודל מסך ו 4 אפשרויות של ציפוי המסך.
- נתמקד בבדיקות בגודל מסך medium ובציפויים dpi-high, dpi-medium-dpi, medium-dpi.
- פעולה סולולרי: כדאי גם לבדוק מול פעולהים שונים כי עצמת הרשת עלולה להשפיע על הביצועים.

דברים שכדי לבדוק באפליקציה מובייל

התקינה והסירה, תפריט, מקשיים ומקלדות, ניהול נתונים, זמן הסוללה, הפרעות, הודעות שגיאה.

בדיקות ייחוד באנדרואיד סטודיו

שנים שני סוגי של בדיקות ייחוד:



רמת חומרת הבאגים

תיאור	שם	חווארה
- מונעת ביצוע של רכיבים מרכזיים במכשיר - ממכנתת תוביה או ביטחון - תקללה מוחשית בסונה שהוגדר בעלי חשיבות גבוהה	Highly Critical	5
- ישפיע עליה על ביצוע של רכיבים מרכזיים במכשיר - בעיית מושגים או ביצועים המשפיע על תפקוד המערכת - כל קלה שברור שתשמש כבסיס לאיסכם לקל	Critical	4
- תקללה בליטת מוחשית לעלי הילוקו (אם בפועל החשיפה שלה פחותה) - בעית מושגים או ביצועים חמורה (אך עדין לאפורה שימוש)	Major	3
- תייקוד של המערכת אינו פעיל, אבל יש דרך סבירה לעקוף זאת. - סדר פעולות הדורש והמשתמש אינם טובים: לא אינטואיטיבי, מסובך, מבלבל וכו' - תקללות במנשך הגרפי לא אינטואיטיבי, טקסט או נתונים קשים לקויה או כתיכים, שגיאות כתיב וכו'	Medium	2
- תקלות קלות אחרות, בממשק הגרפי או בהיבטים אחרים	Minor	1

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

7 העקרונות של בדיקת תוכנה

1. הגדרה – מה זה בדיקה ?
2. בדיקות לעומת מפרטים.
3. מבדיקה לבדיקות רגסיה.
4. תוצאות הבדיקות הם .test oracles
5. בדיקות ידניות ואוטומטיות.
6. אסטרטגיות בדיקה.
7. קритריוני הערכה.

Testability

עד כמה ניתן לבדוק מערכת / רכיב.
 או במידה בה מערכת או רכיב מאפשרים את ההגדרה של קритריוני בדיקה ואת הביצוע של הבדיקות אשר יקבעו האם קритריונים אלה הושגו.

כללים להבטחת בדיקות התוכנה

:Controllability

“The better we can control it, the more testing can be automated and optimized”
 קיימים ממשק בו ניתן להגדיר תרחishi בדיקה, או אמצעי להפעלת בדיקות.
 מוצבי התוכנה והחומרה וכן משתני מערכת ניתנים לשילטה באופן ישיר על ידי מהנדס הבדיקות.
 אובייקטים, מודלים או שכבות פונקציונליות ניתנים לבדיקה באופן בלתי תלוי.

:observability

“What you see is what can be tested”
 מוצבי עבר ומערכות היסטוריים של משתני מערכת הינם גלוים או בני-תשאול.
 פלט שונה מיוצר עבור כל קלט.
 מוצבים ומשתני מערכת הינם גלוים או בני תשאול תוך כדי ריצה.
 כל הגורמים המשפיעים על הפלט הינם גלוים.
 פלט לא תקין – מזוהה בבעיות.
 שגיאות פנימיות מתגללות באופן אוטומטי ומדוחחות על ידי מנגנון בדיקה עצמאית.

:Availability

“To test it, we have to get at it”
 בתוכנה קיימים כמה באגים, אך באגים אינם מפריעים להרצת בדיקות.
 המוצר מתפתח בשלבים פונקציונליים, מאפשר פיתוח ובדיקה באופן סימולטני, חלקים שכבר נבדקו יכולים לשמש לבדיקת חלקים אחרים.
 קיימת נגישות לקוד המקור.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

:Simplicity

"The simpler it is, the less there is to test"

התוכן שומר על עקביות עצמית.

פשוטות פונקציונלית – תוכנות נדרשות ולא יותר מזו.

פשוטות מבנית – מבנה המחלקות והקבצים.

פשוטות הקוד – קוד פשוט וקריא.

:Stability

"The fewer the changes, the fewer the disruptions to testing"

שינויים בתוכנה אינם שכחחים.

שינויים בתוכנה מבוקרים ומפורטים.

שינויים בתוכנה אינם הופכים בבדיקות אוטומטיות לבלתי תקפות.

:Information

"The more information we have, the smarter we will test"

התוכן דומה למוצרים קודמים שאנו מכירים.

הטכנולוגיה עליה מבוסס המוצר מובנת היטב.

התלויות בין רכיבים חיצוניים, פנימיים ומשותפים – מובנת היטב.

מטרת התוכנה מובנת היטב.

משתמשי התוכנה מובנים היטב.

הסביבה בה ישמשו בתוכנה מובנת היטב.

התיעוד הטכני נגיד, מדוק, מאורגן היטב, ספציפי ומפורט.

דרישות התוכנה מובנות היטב.

:סוגי בדיקות מערכת:

בדיקות פונקציונליות, בדיקות תצוגה, בדיקות נתונים וhasil, בדיקות שימושיות, בדיקות ביצועים,

בדיקות רגסיה.

בדיקות פונקציונליות

בדיקות תפקוד המערכת על פי המוגדר במסמך האפין המפורט.

שלבי הבדיקה:

1. בדיקות נקודתיות מתבצעות על הפונקציות המקומיות של המערכת – בודקים את כל

הקומבינציות האפשריות – חוקיות ולא חוקיות.

2. תכנון תרחישי בדיקה והגדרת תוצאות צפויות לכל תרחיש.

3. ביצוע סבב בדיקות שלם – בדיקת התנהלות המערכת מול התוצאה הצפוייה.

בדיקות תצוגה

נודא שמשתמש המשתמש עומד בדרישות שהוגדרו לו מבחינת פונקציונליות, מבחינה ויזואלית

ובבחינת סטנדרטים של עיצוב.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

בסיום הבדיקה נוכל לומר האם משק המשמש מכיל את כל המרכיבים שהוא צריך להכיל והאם הם מתפקדים כיאות.

שלבי הבדיקה:

1. מפים את כל מסכי המערכת ובכל מסך מפים את כל השדות, טקסטים ועוד.
2. לאחר המיפוי יבדק כל רכיב בצורה פרטנית מול ההגדרות באפיון.

בדיקות נתונים והסבות

בדיקות נתוניים לאחר ביצוע הסבנה טכנית. לדוגמה, שינוי תשתית בדיקות נתוניים לאחר שינוי מבנה ועיצוב. לדוגמה, שדרוג המערכת לגרסה חדשה. שלבי הבדיקה:

1. ברמת השדה הבודד, השווואה של שדות המקור והיעד, על פי החוקיות שהוגדרה בהסבה. יש לבדוק על מוגדים נתוניים המציג את אפשרות ההסבה השונות.
2. ברמת הסיכון, בדיקה של סיכון הרשותות המוסבות בחתכים שונים למול הסיכומים ברשומות הישנות.
3. ברמת השאלות, השוואת ישן מול חדש.

בדיקות שימושיות

מטרות הבדיקה:

1. להעיר האם משק המשמש עיל, אסוציאטיבי, קל ונוח לשימוש, כך יוכל להשתלב בקלות בתהליכי עבודה המשתמשים ולא יכבר עליהם בשל בעיות עיצוב של משק לא נוח ולא ברור.
2. בדיקת טולרננסיות של המערכת- מידת ההתאמה של המערכת לעבודתו של המשמש. מערכת עומסיה בתהליכי, מסובכת עם תగובות לא ידידותיות לעבודת המשמש ולטעויות תזונה ע"י המשתמשים לטובת חלופות אחרות.

שלבי הבדיקה:

1. בדיקת נוחות התהליכים, ביצוע מספר צעדים קטן ככל הנitin (לפחות לפעולות השכיחות), בדיקת צורת המ██ים ועוד.
2. יש להתאים את הבדיקות כמה שיותר לנסיבות הלוקו ולתעד את תగובות המשמש ואופן תפקידו עם המערכת.

בדיקות ביצועים

מטרות הבדיקה:

1. עוזרות לבחון האם המערכת עומדת בדרישות הביצועים שהוגדרו לה.
2. בודק האם המוצר עומד בדרישות שהוגדרו באפיון או על פי התקנים המקבילים בארגון.

שלבי הבדיקה:

1. הפעלת המערכת תחת עומס ממוצע ועומס שיא ובחינת זמני הביצוע של תהליכי מרכזיים המערכת.
2. בדיקה של הנקודות המקיים של מידע במערכת בעומסים צפויים, מספר עמדות שצרכית לעבוד יחד וכדומה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

בדיקות גרסיה

בדיקות רוחניות המבוצעות כל סבב בדיקות, לקרהת גרסה חדשה ועם סיום התקנת גרסה ביצור, בשבייל לוודא שהמערכת עובדת תקין.

שלבי הבדיקה:
 יש להריץ תרחישי בדיקות שנכתבו עבור תהליכי שפותחו ונבדקו בעבר.

בדיקות תוכנה ידניות

בדיקות תוכנה הנעשות על ידי עובד שהוכשר לכך בדרך כלל על פי תוכנית בדיקות מסודרת ומוסכמת.

הבודק משתמש במערכת כפי שימוששי הקצה היי ואז קובע האם התוכנית פועלת כראוי.
 יתרונות:

- עלות נמוכה בטיבוח הקצר.
- הכי קרובה למציאות – لكن הסיכוי שייעלו בעיות אמיתיות גבוהה.
- גמישות – שינוי תסritis הבדיקה קל ומיד.

חסרונות:

- קשה – ישן פעולות שקשה לבחון אותן ידנית. לדוגמה, בדיקות גרסיה.
- רוטיניות – בדיקות מסוימות יכולות לחזור על עצמן, מה שמקשה על העבודה.
- חד פעמי – אחרי כל שינוי בקוד צריך לבצע את הבדיקות מחדש.

בדיקות תוכנה ממוכנות

בדיקות תוכנה הנעשות בצורה אוטומטית, רצוי עם מינימום התערבות אנושית.
 מגדירים את הבדיקות בכללי לבדיקות אוטומטיות והבדיקות רצות ומדוחות האם התוצאות בפועל מתאימות לתוצאות הצפויות.

יתרונות:

- מהיר ויעיל – ההגדרה הראשונית לנקחת זמן, אבל הריצות מהירות וזולות.
- מעניין – הופך את תפקיד הבודק לפחות טכני ודורש יותר מחשבה.
- כולם רואים תוצאות – הגישה לתוצאות הבדיקות פתוחה לכל צוות הבדיקות והן מתועדות בצורה אוטומטית.

חסרונות:

- עלויות רכישה.
- מוגבל – לא מבטל את הצורך בבדיקה ידנית. לדוגמה, בדיקות GUI.

אפקטיביות של בדיקות תוכנה

כמויות השגיאות שהתגלו בסבב הבדיקות

אפקטיביות =

סך כל השגיאות שהתגלו במהלך החיים, שמקורו בתוצר הנבדק

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שלב העיצוב

UI User Interface

משק המשתמש: החיבור בין המשתמש לטכנולוגיה – אותו רכיב הזמן לעניין המשתמש ווצג לפני בשעה שהוא מבצע פעולה כלשהי.

User Experience UX

חווית משתמש: עוסק באלמנט החוויתי של השימוש במכשיר – ולמעשה "מודד" את תחושותיו של המשתמש כתוצאה מהשימוש במכשיר ואת שיעור רצונו ממנו ומהארגון שלו. האלמנט החוויתי נולד מתוך ההבנה שהצרכים זוקקים לו למען פרקטן והן למען רגשי – ולמעשה הם רוצים להציג שהאתר, האפליקציה או התוכנה מבינים אותם ואת הזרים שלהם, "מדובר" בשפה שלהם ויכולים להפוך את התהילה לנוח, ידידותי, פשוט ומהנה.

רקע UX

את המונח "חויה משתמש" טבע לראשונה دونלד נורמן באמצעותו ה-90. UX מתיחס ל: משק משתמש, עיצוב, ארכיטקטורת המידע, אסטרטגיה, שיווק, טכנולוגיה, הבטים ריאטיבים, אנושיים ותחושתיים.

מרכבי חוות משתמש

שימושיות Usability – הפונקציות הקיימות במערכת וקלות התפעול שלהן (ביצוע פעולות, הזנת תוכן, קבלת מסוב מתאים מן המערכת).

אסתטיות UI – המראה הכללי של משק המשתמש, מידת היינו נעים לעין, שימוש באלמנטים גרפיים כגון, צורות צבעים, פונטים וכדומה שהיא אסתטיים עבור המשתמש. תנואה Motion & Feel – אינטיציה היא טרנד שלא ניתן להטעם ממנה ומעניקה למשתמש זרימה טבעיות ואינדיקטיביות קוגנטיביות למה קורה במסר. בין אם במערכות חלקים בין תצוגות ובין אם חלק מרכזי מאופי המוצר.

אמינות: העברת מידע מהימן למשתמש והימנעות וטעויות או הטיעות. נגישות: יצירת ארכיטקטורת מידע המתאימה לדרך בה המשתמש חושב או מחשש מידע במערכת.

שימושיות Usability

שימושיות מורכבת מ 5 מרכיבים עיקריים: לימודיות Learnability – כמה קל למשתמש לבצע מטלות פשוטות ברגע שנתקלו בפעם הראשונה במכשיר ?

יעילות – Efficiency – ברגע שימושים למדיו את המשק כמה מהר הם מסוגלים לבצע מטלות ? זכירות – Memorability – כאשר משתמשים חוזרים ל מערכת אחרי פרק זמן ללא שימוש בה, כמה קל להם לרכוש מינונות חזורת ?

שגיאות – Errors – כמה שגיאות משתמשים מבצעים, כמה חמורות השגיאות וכמה קל לתקן אותן ?

סיפוק – Satisfaction – מה מידת האהדה המשתמש רוחש למערכת ?

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

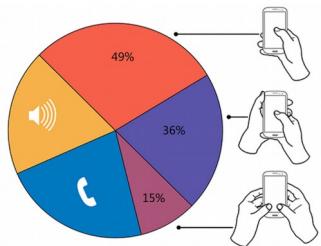
יתרונות בנית אס טוב

חוסך מאמץ הטעמה, מושך לckoחות, מצמצם את מספר הצעדים שהליך צריך לעשות כדי להציגו.

עקרונות לעיצוב ו-

נראות מצב המערכת, התאמת המערכת לעולם האמיתי, שליטה וחופש של המשתמש, עקביות, מניעת טעויות, זיהוי והთאוששות מטעויות, זיהוי לא זכרון, גמישות ויעילות בשימוש, עיצוב אסתטי ומינימליסטי, עזרה ותיעוד, הכינו ותרשים ויזמה.

המלצות כלליות לעיצוב המסך



הגודל המומלץ ללחצנים הוא 10-7" מ', אשר מאפשר מספיק מקום לאצבעות המשתמשים ללוחץ על הכפתור כשהקצוות עדין נראים בבירור מסביב. הקפידו למקם את התפריט הראשי, וכפתורים נפוצים או חשובים במיוחד באזורי נגישים של המסך. צמצמו את הצורך בהקלדה.

הגודל המומלץ לטקסט באפליקציות הוא 11 נק' לפחות. להקפיד על ניגודיות בצבאים בין הרקע לטקסט (במיוחד בטקסט קטן).

Material Design

היא שפה עיצובית שפותחה על ידי גугл.

הוכרזה לראשונה ב 25 ביוני 2014, בכנס המפתחים Io Google.

כל האפליקציות של גугл لأنדרואיד ורוב גרסאות הוו ששליה מיישמות את השפה. רוב אפליקציות האנדרואיד מיישמות את השפה, אך המפתחים אינם מחייבים להתאים את האפליקציה לשפה.

מה זה Material Design

"Material Design" היא שפה חזותית שמשלבת עקרונות קלאסיים של עיצוב טוב עם החידושים של הטכנולוגיה והמדע.

המטרות של "Material Design" :
 יצירת שפה – ליצור שפה חזותית.

איחדות – לפתח תשתיות שיוצרת חזותית משתמש אחידה ללא תלות במערכת מסוימת, במכשיר מסוים או בשיטת הזנה מסוימת (הקלדה/תנועות עבר/קול).

גמישות ויכולת התאמאה – יצירת מגנון הרחבה של השפה ליצירת תשתיות גמישה להכנסת שינויים של חדשנות אישית או סימני מיתוג.

למה להשתמש ב Material Design ?

- כדי להשתאר בקדמת הטכנולוגיה.
- מעניק ידע וניסיוני של גугл.
- חוסר זמן.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

- נתונים למשתמשים מראה מוכר ותחושת ביטחון.

למה לא?

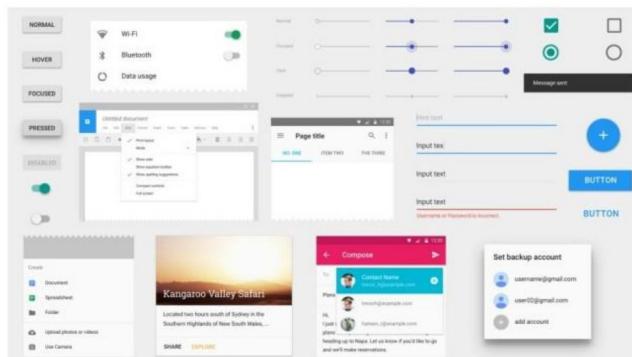
- לא תמיד נרצה להראות כמו אפליקציה של גугл.

- לעיתים נרצה להציג מראה חדש.

- מתאים לאפליקציה שיש בה דברים בטנדרתיים ולא למשחקים.

- יש גם מתחרים ל Material Design

Material Design Components



איך מתחילה השתמש?

צריך לוודא שיש לנו ברשימה התלוויות בקובץ gradle בرمת הפרויקט את ההפנייה ל google()

```
allprojects {  
    repositories {  
        google()  
        jcenter()  
    }  
}
```

בקובץ ה gradle בرمת המודול מוסיפים את הספרייה לרשימה התלוויות:

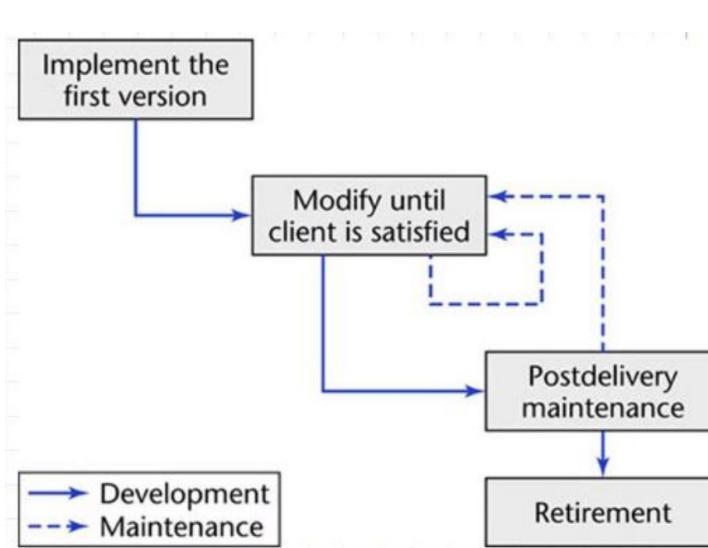
```
dependencies {  
    // ...  
    implementation 'com.android.support:design:28.0.0-beta01'  
    // ...  
}
```

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שיטות עבודה



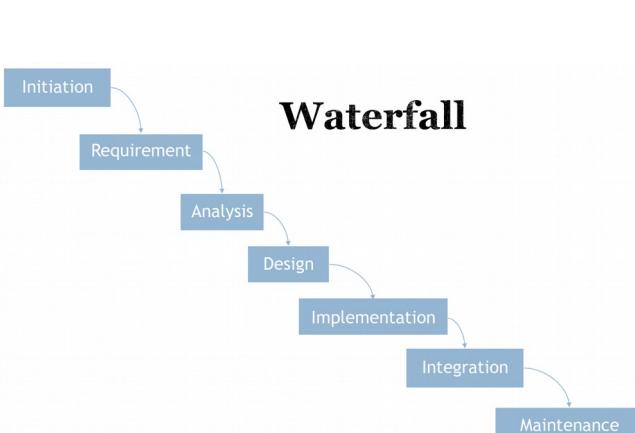
Code and Fix

יתרונות: (במבחן ראשוני ובהתחלה..)

- קל לפיתוח
- מהיר
- קל לניהול
- גמיש

חסרונות:

- סיכון מאוד גבוה
- גורע למערכות מורכבות
- לא מתאים לפרויקטים מתמשכים
- עלות תיקונים גבוהה
- אין לו"ז



Waterfall

יתרונות:

- פשוט, קל להבנה ולשימוש
- קל לניהול, בכלל שהכל קשה
- אבני דרך ברורות

חסרונות:

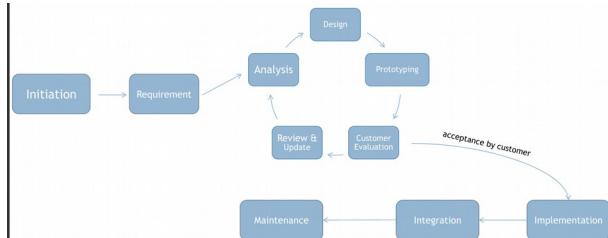
- שלב הפיתוח מגיע מאוחר בתהליך
- סיכון גבוה
- לא טוב לפרויקטים מורכבים, ארוכים או דינמיים
- קשה למדוד את ההתקדמות באמצעות שלב
- אין גמישות לשינויים

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מודל אב טיפוס מהיר



אב טיפוס, כשמו כן הוא, דגם של המוצר

- בעל יכולות מוגבלות.

- אמינותה נמוכה.

- ביטחעים לא עליים.

משמש להדגמה בלבד, הוא לא המוצר הסופי!

שימושי במיחוד כאשר:

- דרישות המשתמש לא מלאות
- נושאים טכניים לא לגמרי ברורים

יתרונות:

- מבהיר את דרישות הלוקו

- מקל במקדים של קשיים טכניים

חסרונות:

- מגדיל את עלות המערכת

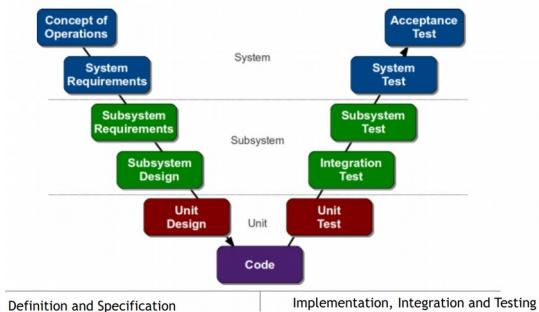
- לא מקל על סיכונים שמתרגלים בתהיליך הפיתוח

V-model

המודל נועד לפשט ולשפר את תהליך הפיתוח לעומת מפל המים.

השלבים הראשונים של הפרויקט הם ניתוח ברמה כללית ובשלבים הבאים הניתוח נהיה יותר ויותר מפורט עד רמת היחידה.

בשלב זה מפתחים את היחידות הפיתוח הקטנות ביותר ביחס למרכיבות (ביחד) את התמונה השלמה. בחלק השני של המודל נמצאים שלבי הבדיקות בסדר הפוך – מתחילה בבדיקות ברמת היחידה, דרך בדיקות אינטגרציה ותתי מערכות ועד בדיקות המערכת ובדיקות קבלה סופיות.



יתרונות:

- קל להבנה ולשימוש

- קל לניהול בגליל התיעוד והסדר

- השלבים ואבני הדרך מוגדרים היטב

- התהיליך והפתרונות מתעדות היטב

חסרונות:

- תהליך הפיתוח מתחילה מאוחר בשלבי הפרויקט.

- חוסר וודאות, סיכון גבוה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

- לא מתאים לפרויקטים מורכבים, דינמיים, ארוכים וمتמחכים.
- אין גמישות לשינויים במהלך הפרויקט.

From linear to iterative

קשה ומורכב לפתח תוכנה!

נעשות טיעות במהלך פיתוח התוכנה.

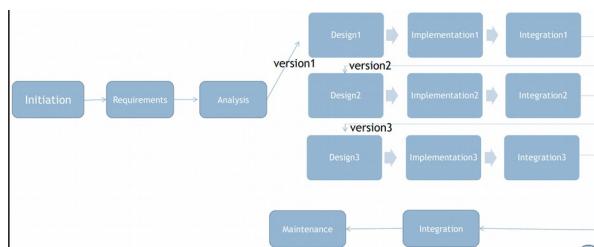
הדרישות משתנות תוך כדי הפיתוח.

כמו שהעולם כל הזמן משתנה כך גם מערכת התוכנה שאנו מפתחים.
 מודלים לינאריים לא גמישים לשינויים אלו ולכן נועבור למודלים איטרטיביים.

Iterative Model

יתרונות:

- ניתן לראות תוצאה כבר בשלב מוקדם של התהילה.
- ניתן לתקן פיתוח מקביל.
- ניתן למדוד את התקדמות הפרויקט.
- עלויות נמוכות יותר במקרה של שינויים בתוכנה או בדרישות.



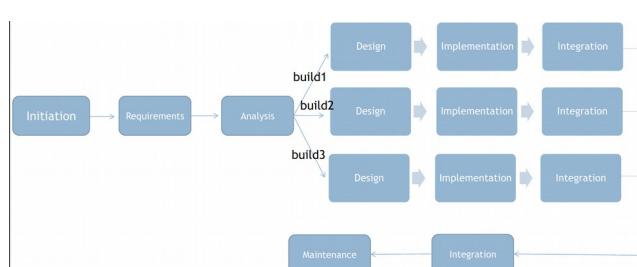
חסרונות:

- דרש משאבים גבוהים.
- אמנים העלוויות של השינויים נמוכות יותר, אבל עדין לא ממש מתאים לשינויים בדרישות.
- דרש ניהול צמוד וקפדי.
- אין תמונה מלאה של המערכת הנדרשת לפני תחילת התהילה.

Incremental model

יתרונות:

- יצירת תוכנה עובדת מהר ו בשלב מוקדם של התהילה הפיתוח.
- מודל גמיש יותר לשינויים, פחות יקר לשנות את התוכולה והדרישות.
- קל יותר לבדוק אינטרציות קטנות.
- הליקוי יכול להגיב לכל מודול.
- קל יותר לנוהל סיכונים, כי חלקיים מסוכנים מוגדרים ומטופלים באיטרציה שלham.



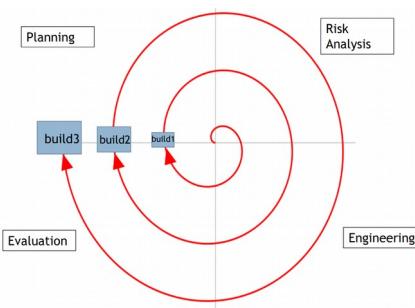
סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

חסרונות:

- דרוש תיקון ועיצוב קפדי.
- יש צורך להבהיר ולהגדיר את כל המערכת לפני שאפשר לחלק אותה למודלים.
- העלות הכלולת כנראה תהיה גבוהה יותר ממודל מפל המים.



Spiral model

דומה למודל האינקרמנטלי אבל עם דגש לניתוח סיכוניים.

שלבי הפיתוח:

- planning
- risk analysis
- engineering
- evaluation

בתחילת פיתוח תוכנה עוברים באיטרציות על ארבעת השלבים, כאשר כל איטרציה מבוססת על האיטרציה הקודמת.

יתרונות:

- ניהול סיכונים ברמה גבוהה.
- אפשר להתחיל פרויקט פשוט ולהוסיף סיבוכיות בהמשך.
- הפיתוח מתחילה מוקדם בתהליך.

חסרונות:

- עלויות גבוהות יותר לפROYיקט.
- הצלחת הפרויקט תלולה בעיקר בשלב ניתוח הסיכוניים.
- לא עובד טוב עבור פרויקטים קטנים.

מתודולוגיות אג'יליות

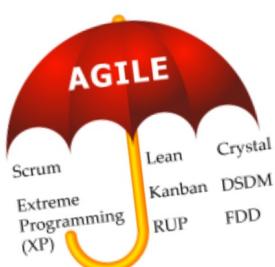
agile היא מתודולוגיה לפיתוח תוכנה זריזה, מתודולוגיה איטרטיבית שהומצאה לפיתוח תוכנה בצוותים קטנים תוך שימוש דגש על יעילות, זריזות וaicות.

הגישה הזריזה לפיתוח תוכנה מניחה שלא ניתן להגיד במילואה תוכנה מסויימת קודם לפיתוחה בפועל, ומתמקדת במקום זאת בשיפור יכולתו של היצור לספק תוצרים במהירות ולהגיב לדרישות העולות תוך כדי הפיתוח.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון



Agile

זה עיקרונו, מערכת חשיבה שאחרת בוגר לכל תהליכי פיתוח התוכנה.

ישנם מספר שיטות עבודה אג'יליות, כל אחת מן השיטות דומה באג'יליות אחרת, אך שונה באופן הימימוש.

Agile Manifesto

- ב 2001 נפגשו 17 אנשים, כולם מובילים בתחום התוכנה ורבים מובילים בתחום פיתוח מתודולוגיות לפיתוח תוכנה וניסו להסכים על כמה עקרונות משותפים:
- העדפות העלינה היא לשחרר ללקוח גרסה עובדת ובעל ערך כמה שיותר מוקדם.
 - שינויים בדרישות הם טריגר לשיפור התוכנה ולכן נקדם בברכה.
 - נרצה לשחרר גרסה עובדת לעיתים קרובות.
 - נדרש שיתוף פעולה וקשר יומיומי בין מומחי היישום למפתחים לאורך כל הפרוייקט.
 - יש לחת בחברי הצוות אמון שיבצעו את העבודה ולהaddir בהם מוטיבציה.
 - תקשורות פנים אל פנים היא הדרך העילית והטובה ביותר להעברת מידע.
 - תוכנה עובדת היא המדריך העיקרי להתקדמות.
 - תחזוקת המערכת היא חלק בלתי נפרד מפיתוח, וכל בעלי העניין והפתחים צריכים לעשות זאת.
 - מציאות טכנית, עיצוב ותוכנן נכון מגדים את היכולת האג'ילית.
 - יש לפתח את מה שהכרח וללא לפתח את מה שלא.
 - נסאייר לצוותים את האחריות וארקיטקטורה ולעיצוב.
 - המצוות משתפר ונוהיה יותר אפקטיבי לאורך זמן.

השוואה בין מודלים לינאריים ל Agile

Agile	לינארי	הדרישות בפרויקט
מעט דיעו מראש, הרוב גנבה	ודיעות מראש	הדרישות בפרויקט
במהשך התהליך	בקשה להכניס שינויים	שינויים בתוכנות הפרויקט
גיטש	גובהה	גובהה
רמת הסיכון	גובהה נא൰ בהתחלה, נמוכה בהמשך	גובהה נא൰ בהתחלה, נמוכה בהמשך
מעורבות הלקוק בתהליכי הפיתוח	גובהה לכל אורך התהליך, נמוכה	גובהה לכל אורך התהליך, נמוכה
הפרויקט	בפעם אחת	בפעם אחת
באיוטיזיות	לທהילר	אוינוונציה של המודול
מסירה ללקוח	לאנשים	גודל האזות
אוינוונציה של המודול	צוות גדול	בודדים להצלחה של הפרויקט
לאנשים	הערך העסקי ללקוח	בוצעו כל הדרישות
צוותים קטנים		
הערך העסקי ללקוח		

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

Agile vs. Waterfall

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

טכניתה לתחילה פיתוח זריזה המבוססת על עקרונות Agile. פותח בתחילת שנות ה-90 על ידי ken shaw וGF סאטרלנד. המתודולוגיה מבוססת על ההבנה שפיתוח תוכנה היא לא בעיה שנייה לפטור על ידי חיזי או ניתוח, אלא בעיה אמפירית שנייה לפטור אותה בכלים המתאים לביעות אמפיריות. בעיות אמפיריות נפתרות בשיטה של ניסוי וטעייה – נתחיל בהדרגה, נלמד מטעויות ונשפר את המערכת כל הזמן.

*סקראם היא תשתיית ניהול פרויקט תוכנה עם חדש פיתוח איטרטיבי/אינקרמנטלי (פיתוח בספרינטים), עבודה צוות מרוכזת, שקיפות ושיפור מתמיד.

השחקנים:



:Scrum events

Sprint: תקופה של חדש אחד או פחות בו נוצר מוצר בו כל המשימות עומדות בהגדלה של "בוצע", המשימות שימוש ובעלות פוטנציאלי לשחרור. Sprint חדש מתחילה מיד לאחר סיום הספרינט הקודם.

Sprint planning: העבודה שיש לבצע בספרינט מתוכננת ב-Sprint planning. תוכנית זו נוצרת על ידי עבודה משותפת על צוות ה scrum שלהם.

ה sprint planning עונה על השאלות הבאות:

▫ איזה תוסף ל מוצר ניתן לספק בסיום הספרינט ?

▫ כיצד תבצע ותונסг העבודה הנדרשת על מנת לספק את התוסת המתוכננת ?

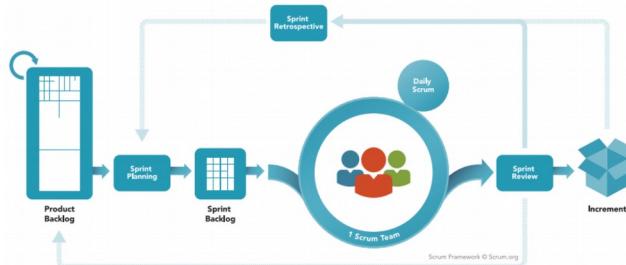
daily scrum: פרישה קצרה של חברי הצוות המתקיימת בכל יום במהלך הספרינט ובها צוות הפיתוח מתכנן את העבודה ל 24 שעות הקרובות.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

- sprint review: פגישה המתקיימת בסיוםו של sprint על מנת לבדוק את התוספת שפותחה ועל מנת להתאים את ה product backlog במידת הצורך.
- sprint retrospective: פגישה המתבצעת לאחר sprint review ולפני sprint planning כדי לבדוק את עצמו וליצור תכנית שיפורים שיחולו הבא ומטרתה לחתן הזדמנויות לצוות scrum לבדוק את עצמו וליצור תכנית שיפורים שיחולו על sprint הבא.

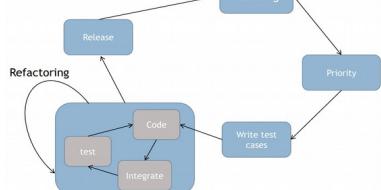


בעיות ב Scrum

- חוסר בהירות בנוגע לתקופה הסופית של הפרויקט: הלוקוח עלול להמשיך לבקש עוד פונקציונליות חדשות.
- "צוות שמנהל את עצמו": לא תמיד זה עובד טוב/קשה מאד לישום. אם חברי הצוות לא מוחיבים למטרה, הפרויקט יכשל.
- "עבודה ביחידות קטנות": על הניר נשמע פשוט, באופן מעשי יכול לגרום לביעות וקושי ביצוע.
- צוותים קטנים: חברי הצוות צריכים להיות מiomנים מאד בשבייל שהטהילה יצליח.
- נדרש שינויים מאד גדולים בארגון: גם ברמת הנהלה וגם ברמת הצוותים המפתחים.
- תפקיד ה scrum master: תפקיד קשה לביצוע וקשה לאיש.

XP model

Extreme Programming (XP) טכניקה לטהילה פיתוח זרי המבוסס על עקרונות Agile. נותנת דגש על מעורבות הלוקוח בתהילה פיתוח התוכנה תקשורת טובה בתוך הצוותים ועבודה באיטרציות פיתוח.



המודל פותח על ידי קנט בק בשנת 2000. העקרונות המרכזיים עליהם מושתת XP: תקשורת, פשוטות, משוב, אומץ, כבוד.

12 שיטות העבודה של XP:

The Planning Game, Small Releases, Metaphor, Simple Design, Continuous Testing, Refactoring, Pair Programming, Collective Ownership, Continuous Integration, 40-hour week, On-site Customer, Coding Standards.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

יתרונות:

- מעורבות הלקוּה מגדילה את הסיכוי שהתוכנה המיוצרת תענה על הצרכים של המשתמשים.
- מקטין את הסיכון בפרויקט.
- בדיקות ו互動גרציה מתמשכות מסייעות להגבר את איכות העבודה.
- זמן הפיתוח קצר יותר, כי מכנים את הבדיקות בשלב ההגדרות.

חסרונות:

- AX מיועד לפרוייקט אחד, שטפותה ומרתווזק על ידי צוות אחד.
- AX פגוע במיזוג עבור מפתחים סוליסטיים אשר לא עובדים טוב עם אחרים.
- AX לא יעבוד בסביבה בה הלקוּה או המנהל מתעקשים על מפרט מלא או עיצוב לפני שהם מתחילה לתוכנת.
- AX לא יעבוד בסביבה שבה מתכנתים מופרדים במחינה גיאוגרפית.

Pair Programming

2 מפתחים שושבים ביחד באותו מחשב.
 הזוגות דינמיים.

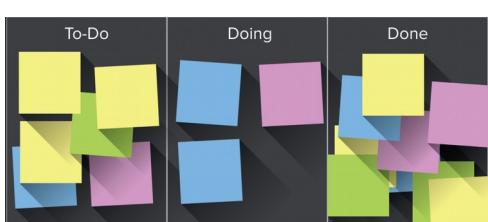
הרעיון הוא שני אנשים בודאות יכתבו קוד יותר טוב, יותר יעל ועם פחות באגים מאשר אחד.
 2 המתכנתים הם בעלי רמות שונות של ניסיון, אבל זה לא נועד לחניכה של מפתח חדש.
 ישם מקרים המראים שפיתוח בזוגות הוא מאוד מוצלח.

Kanban model

kanban בפניהם זה כרטיס או "סימן ויזואלי".
 טכניקה לתחילת פיתוח תוכנה זריזה המבוססת על עקרונות agile.
 Kanban נותן דגש לצד הייזואלי של התהילה: מה ליצר, מתי ליצר וכמה ליצר.
 השיטה מעודדת ביצוע שינויים קטנים והדרגתיים במערכת.

יתרונות:

- מגדיר את הגמישות לשינויים.
- מצמצם בזבוז זמן, תמיד ממתינה למתקנת עוד משימה.
- קללה להבנה ולהטמעה, ניתן להשתמש על גבי מתודולוגיה אחרת.
- מקצר את זמן המסירה של משימות.



חסרונות:

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

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

ניהול פרויקטים

הגדרה

לפי ה (Project Management Institute) PMI :
 מאמצז זמני שיש לו התחלה וסוף מוגדרים.
 מיוצר מוצר ייחודי (במקרה שלנו – מוצר תוכנה).
 עשויי למען מטרה.
 מורכב ממשימות הותלוויות ביןיהן.

למה פרויקטי תוכנה נכשלים לעיתים כל כך קרובות ?

עד הפרויקט לא מוציאות ולא ברורים, אומדן לא מדויקם של המשאבים הנדרשים, דרישות מערכת אין מוגדרות היטב, דיווח לקוי לגבי מצב הפרויקט, סיכונים לא מנוהלים, תקשורת לקויה בין הלוקוח, המפתחים והמשתמשים, שימוש בטכנולוגיה לא בשלה, אי יכולת לנהל את מרכיבות הפרויקט, פרקטיקות פיתוח מרושלות, ניהול לקוי של הפרויקט, פוליטיקה של בעלי עניין, לחצים מסחריים.

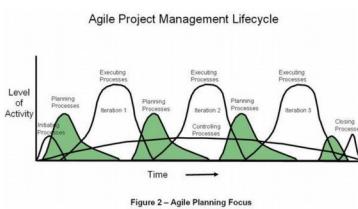
משולש האילוצים:



תכונות חשובות למנהל פרויקטים:

תקשורת טובה עם אנשים, מנהיגות, יכולת ניהול צוות, ידע לניהול זמן, יכולת ניהול סיכונים, בן אדם מאורגן, ידע לניהול משא ומתן בצורה טובה ובהרגשה נעימה ל-2 הצדדים, מומחיות בנושא, חשיבה ביקורתית, מיזמוניות תוכנן ידע בשימוש בכל תכונן.

שלבי ניהול פרויקט

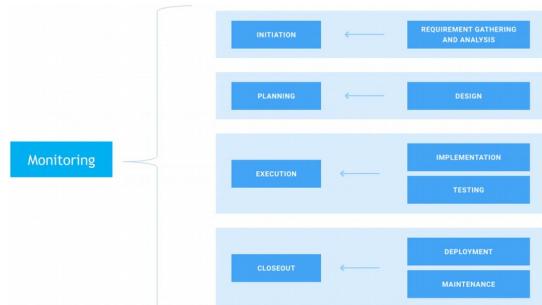


סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מחזור חיים + שלבי ניהול הפרויקט:



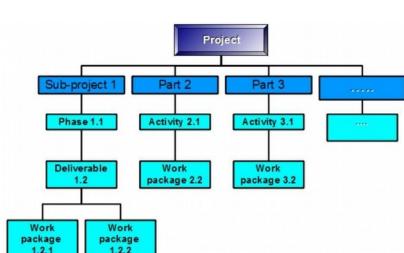
שלב הייזום

- הכרה ב הצורך לבצע: מה הרעיון שלנו/איזה בעיה אמורה להיפתר ?
- הגדרת מטרות העל של הפרויקט.
- הגדרת ציפיות הלקוחות, הנהלה ויתר בעלי העניין בפרויקט.
- הגדרת היקף הפרויקט.
- בחירת הצוות הראשוני לביצוע הפרויקט.

שלב התכנון

- חלוקת מטרות העל ליעדים ניתנים למדידה.
- חלוקת הפרויקט לתתי משימות.
- גיוס וגיבוש צוות העבודה.
- קביעת לו"ז לביצוע המשימות.
- הערכת עלויות הפרויקט.

כלים שונים בשלב התכנון:
WBS, תרשימים ראש, קביעת משך כל פעילות, PERT, CPM, Gantt



(WBS) Work Breakdown Structure
חלוקת הפרויקט לתת משימות הנונוטנות ביחד את התמונה המלאה.
הרמה העליונה מייצגת את המוצר הסופי.
הרמה התחתונה נקראת חבילת עבודה.
אפשר לקבל תמונה מלאה של דרישות הפרויקט ברמת המשימות.

תרשימים ראש

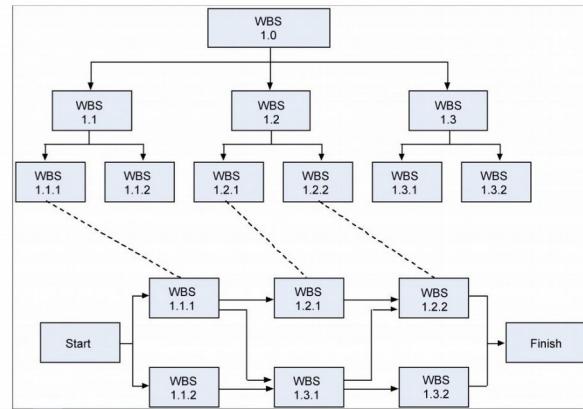
כל' להציג וניהול שרשרת המשימות בפרויקט.
מראת תלויות בין המטלות.
יכול לסייע לתכנון עיל, ארגון ובקרה הפרויקט.

סיכום קורס הנדסת תוכנה

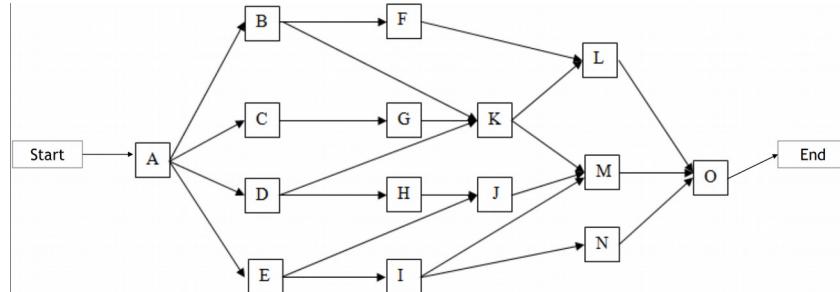
נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

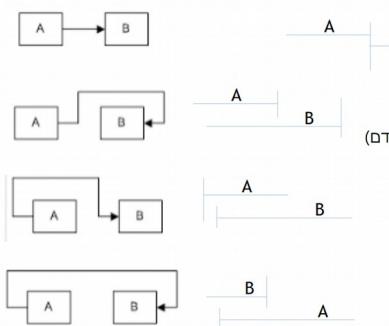
מ WBS לתרשים רשת:



:schedule network diagram



תלות בין משימות:



Finish to Start :

- יכול להתחיל רק כאשר A מסתיים
- זו דרך ליל בירתה המחדלה

Finish to Finish :

- יכול להסתיים יחד עם A או אחריו (חיה לסייעים קודם)
- או- B לא יכול להסתיים לפני A-

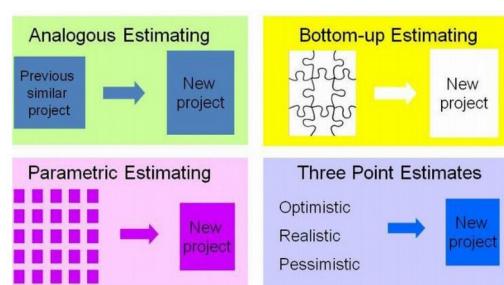
Start to Start :

- יכול להתחיל בזמןית עם A או אחריו
- אך לא יכול להתחיל לפני A-

Start to Finish :

- מסתיים כאשר A מתחילה
- או- חיבם להתחיל את A בשני ש- B יכול להסתיים
- זה סוג ייחודי נדיר

אומדנים למשר הפעולות



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

תרשים (Program Evaluation Review Technique)

נוצר בשנות החמישים על מנת לסייע בניהול "צורך כל נשק בצי האמריקני".
 מסיע בקביעת אומדן על עלות וזמן של משימות בפרויקט.
 נותן תמונה מלאה על הפרויקט ומאפשר מעקב אחריו ביצוע הפרויקט.

תרשים PERT מבוסס על תרשימים רשת, רק מוסיף את נושא אומדן הזמן והעלויות:

עבור כל פעולה ניתן 3 אומדנים:

אופטימי (shortest time)

פסימי (longest time)

סביר (likely time)

$$\frac{\text{shortest time} + 4 * \text{likely time} + \text{longest time}}{6}$$

נוסחת PERT:

CPM Critical Path Method

הדרה: שרשרת פעילות הקשורות זו לזה, אשר כל שינוי בזמןם ישפיע על מועד סיום הפרויקט.

בפרויקט בו כל המשימות תלויות זו בזה ולא מתבצעות פועלות במקביל, הפרויקט יכול נמצא בנתיב הקרייטי.

בפרויקט בו מתבצעות פועלות במקביל, ישנו מספר נתיבים, נוצרים מרוחקים (slacks) המאפשרים לדוחות פעילות מסוימת בלי לדוחות את סיום הפרויקט.

מושך הזמן של הנתיב הקרייטי הוא גם המושך המינימלי של הפרויקט.

התחלת וסיום של פעילות:

Early start: התאריך המוקדם ביותר בו משימה יכולה להתחילה ביחס לתלויות.

Early finish: התאריך המוקדם ביותר בו משימה יכולה להסתיים בהתייחס לתלויות (early start + duration).

Late start: התאריך המאוחר ביותר בו משימה יכולה להתחילה בלי לדוחות את מועד סיום הפרויקט (late start - duration).

Late finish: התאריך המאוחר ביותר בו משימה יכולה להסתיים בלי לדוחות את מועד סיום הפרויקט.

:Slack

זמן שבו המטללה יכולה להמתין ללא עיכוב בפרויקט.

חישוב ה-slack לשימה:

$$\text{Latest start} - \text{Early start}$$

דוגמה:

נתונה ורשות המטלות בפרויקט מסוים:		
תלויות	אומדן זמן	ນס' מטללה
	3	1
1	3	2
1	6	3
2	8	4
3,4	4	5

.1 מה מושך הנתיב הקרייטי? **18w**

.2 מה ה-slack של מטללה ?3?

.3 מה ה-slack של מטללה ?2?

.4 המשאב שעבד עבור מטללה 3 התפרק ובמקומו עבד משאב פחות ייומן שיקח לו 10 שבועות.

.5 כיצד יושפע הפרויקט? **לא ישפע**

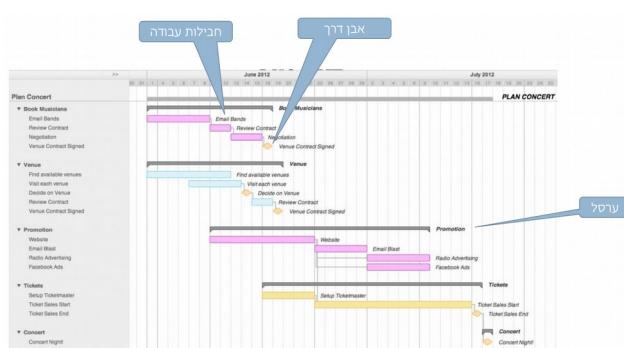
.6 נסופה מטללה 6 עם אומדן של 11 שבועות לפני מטללה 5 ואחרי מטללה 3. כמה זמן יימשך

.7 הפרויקט? **24w**

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון



Gantt

הומצא בתחילת המאה ה-20, השיטה הנפוצה ביותר להציג לוייז הפרויקט. הציר האופקי מייצג את הזמן, הציר האנכי מייצג את חברות העבודה. זה התרשים השימושי ביותר ע"י מנהלי פרויקטים.

דוגמה

נתונה טבלה המטלות בפרויקט מסויים:

מספר	אומדן זמן			תלוויות	מטלה
	פסימי	סביר	אופטימלי		
6	4	2	--		A
9	5	3	--		B
7	5	4	A		C
10	6	4	A		D
7	5	4	B,C		E
8	4	3	D		F
8	5	3	E		G

נרצה לנתח את נתוני הטבלה ולמצוא את המידע הבא:

אומדן זمانים לפי נוסחת pert.

לשרטט תרשימים רשת.

למצוא את הנתיב הקратי, מה המשכו?

לחשב את 4 סוגי הזמנים: early start, early finish, late start, late finish

לחשב את ה-slack לכל המשימות.

חישוב נוסחת PERT:

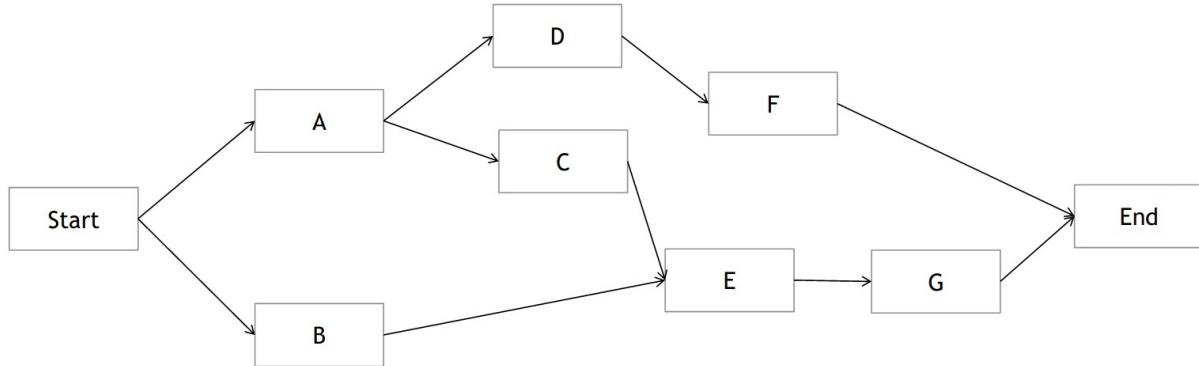
זמן צפוי	אומדן זמן				תלוויות	מטלה
	פסימי	סביר	אופטימלי			
4.00	6	4	2	--		A
5.33	9	5	3	--		B
5.17	7	5	4	A		C
6.33	10	6	4	A		D
5.17	7	5	4	B,C		E
4.50	8	4	3	D		F
5.17	8	5	3	E		G

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

תרשים רשת:



התחלת וסיום של הפעולות:

מטלה	מטלה	מטלה	מטלה	מטלה	מטלה	מטלה
Start	A	B	C	D	E	G
Start	0	0	0	0	0	0
	0	4	0	4	0	0
	3.84	9.17	3.84	5.33	0	4
	0	9.17	4	9.17	4	4
	4.68	15.01	8.68	10.33	4	4
	0	14.34	9.17	14.34	9.17	9.17
	4.68	19.51	15.01	14.83	10.33	10.33
	0	19.51	14.34	19.51	14.34	14.34
	0	19.51	19.51	19.51	19.51	19.51
End						

שאלות נוספת:

1. המשאבות שעבור מטלה D התפטר ובמקומו עבד משאב פחות מיום שיקח לו 8.33 ימים. כיצד ישפיע הפרויקט?
2. המשאבות שעבור מטלה B היה חלה ונעדר מהעבודה חמישה ימים. (מנהל הפרויקט לא מצא מחליף לימים שנעדר). כיצד ישפיע הפרויקט?
3. נוספה משימה חדשה לאחר משימה f עם אומדן של 5 ימים. כיצד ישפיע הפרויקט?

תשובות:

1. לא ישפיע.
2. הפרויקט יחרוג מהלוז שתוכנן ב-1.16 ימים.
3. הפרויקט יחרוג מהלוז שתוכנן ב-0.32 ימים.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שלב הביצוע

השלב הכי מקשור למנהל הפרויקט.

בשלב זה המנהל אחראי על:

- סיפוק תוצריים ללוקוח.
- מיקוד חברי הוצאות במשימות.
- שמירה על המשאבים המקצחים.
- עמידה על הוצאה לפועל של התכנון!
- התאמת לכל חבר צוות את המשימה שמתאימה לו.
- הסבר על המשימות: לדאוג שהכל מובן ואם לא לדאוג להדרכה.
- יצירת קשר עם הלוקוחות, חברי הוצאות והנהלה.
- הביצוע תלוי במידה רבה בשלב התכנון.

שלב הבקרה

תהיליך מתמיד של השוואת בין הביצוע לבין התכנון תוך נקיטת פעולות מתונות על מנת להקטין פערים בלתי רצויים.

סוג בקרות בפרויקט:

- בקרת תכולה.
- בקרת לו"ז.
- בקרת תקציב.
- בקרת סיכון.
- בקרת איכות.
- בקרת שינוי.

דרכים לצמצום הלוא"

Re-estimation: בדיקה מחודשת של ההנחות, דברים שלא היו ידועים בזמןנו וכו', Crashing: הוספה יותר משאבים לנútב הקritis, תוך שמירה על התכולה. דרש הגדרה של התקציב

Fast Tracking: ביצוע מטלות בנútב הקritis במקביל. מגביר סיכון וסיכון. לא אידיאלי, אבל אפשרי ו שימושי לעתים.

משולש האילוצים: להוריד באיכות או לצמצם תכולה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

דוגמאות

נתונה רשימת המטלות בפרויקט מסויים:

תღיות	זמן	מס' מטלה
	0	START
--	4	D
--	6	A
A,D	7	F
D	8	E
E,F	5	G
F	5	B
G	7	H
H	8	C
C,B	0	END

הלקוח דורש שהפרויקט יסתם תוך 30 חודשים.

איך נוכל לצמצם את הזמן?

Fast track – H-A במקביל – B-G.

העברת משאבים מ-Bל-G–Crash.

להוריד את H – צמצום תקופה.

להוריד את סטנדרטי האיכות (של משימות בנתיב הקרייטי).

ערך מזוכה

הביצועים הנוכחיים הם האינדיקטור הטוב ביותר לבדיקת הביצועים העתידיים, ולכן, באמצעות ניתוח

המגמה, ניתן לחזות את עלות הפרויקט ואיתו בלו"ז בשלב מוקדם של הפרויקט.

השיטה הנפוצה ביותר היא שיטת ערך מזוכה-Earned Value Method.

בשיטה זו מתרגמים את יכולת העבודה למונחים כספיים.

ביצוע מעקב אחרי התקדמות הפרויקט ובדיקה האם אנחנו מקדים או מאחרים בלו"ז של הפרויקט

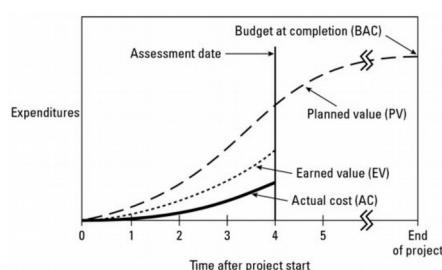
והאם אנחנו במסגרת התקציב או超 גרים ממנו.

בחינת הערך המזוכה תמיד נעשית סביר נקודת בקרה ספציפית, נקבעת נקודת חיתוך בזמן (למשל:

היום), אשר מהויה נקודת ייחודה לכל הפרמטרים השונים.

השיטה למעשה תקפה רק לאחר שהפרויקט החל.

כל החישובים נעשים ברמת הפעולות ומתנסכים באופן אגרגטיבי לרמת הפרויקט כולם.



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

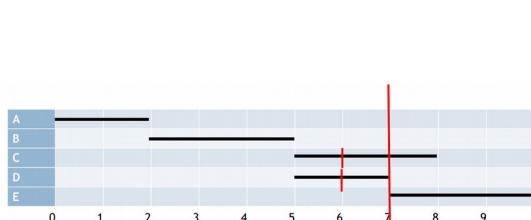
נתונים ומשתנים		
	משמעות	סיביון
$BAC * \% \text{ הביצוע בתכנון}$	Planned Value PV	
$BAC * \% \text{ הביצוע בפועל}$	Earned Value EV	
$EV - AC$	Cost Variance CV	
$EV - PV$	Schedule Variance SV	
$\frac{EV}{AC}$	Cost Performance Index CPI	
$\frac{EV}{PV}$	Schedule Performance Index SPI	
$\frac{BAC}{CPI}$	Estimated At Completion EAC	
$EAC - AC$	Estimated To Complete ETC	
$BAC - EAC$	Variance At Completion VAC	

נתונים ומשתנים:

נתונים שמנהל הפרויקט מעדכן לכל פעילות:

BAC (Budget At Completion) •

AC (Actual Cost) •



אילו הסכומים
שהוא יוציא ימם
ואלו ישלו

דוגמא

Activity	Predecessor	Duration (Days)	Cost/Day	Total Cost
A	-	2	300	600
B	A	3	400	1200
C	B	3	400	1200
D	B	2	200	400
E	D	3	100	300

סטודנטים ביצעו המשימות:

הטניה - A

הטניה - B

נעשרה 1/3 מהעבודה - C

נעשרה חצי העבודה - D

לא התחילו - E

חוצים להשאיב את הערך המומחה אחרי 7 ימים

Activity	AC	PV	EV
A	600	600	600
B	1400	1200	1200
C	500	800	400
D	200	400	200
E	0	0	0
	2700	3000	2400

שיעור אמרורה
להתבצעה

הוצאות העבודה
שביצינו בפועל

What	Calculation	Answer	Interpretation
PV	$600 + 1200 + 800 + 400$	3000	הינו אמורים לעשות עבודה בעלות של 3000
EV	$600 + 1200 + 400 + 200$	2400	השלמוני עבודה ששויה 2400
AC	$600 + 1400 + 500 + 200$	2700	בפועל, הוציאנו 2700
BAC	$600 + 1200 + 1200 + 400 + 300$	3700	עלות הפרויקט כולל
CV	$2400 - 2700$	-300	הפרויקט בחריגה של 300
SV	$2400 - 3000$	-600	הפרויקט באיחור של עבודה בערך של 600

What	Calculation	Answer	Interpretation
CPI	$2400 / 2700$	0.889	אנטו מקבלים 89 סנט על כל דולר שאנונו משקיעים
SPI	$2400 / 3000$	0.8	אנחנו מתקדמים ב-80%-85% המקורי
EAC	$3700 / 0.889$	4162	לפי המגביע עכשווי הפרויקט עולה בסך הכל 4162
ETC	$4162 - 2700$	1461	צריך לוויא עד 1461 בשwil לסיים את הפרויקט
VAC	$3700 - 4162$	-462	נוצרר לוויא עד 462 מעבר לתקציב בשביל לסיים את הפרויקט

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שלב הסגירה

שלב זה קורה אחרי שמשפקים ללקוח את המוצר.
 בשלב זה עושים ניתוח על התהילה שעברו חברי הוצאות:

- מה עבד ?
- מה לא עבד ?
 - למה ?
- איך המנהל יכול להשתפר ?
- כדי ליצור צוות ותהילה יותר טוב להבא ?