

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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 עליו (מצד הלוקו).
- תוכנה בעלת קוד נקי וברור, קללה לקרואיה (מצד המפתח) –> קללה לבדיקה ולתחזוקה.

מדוע פרויקטים נכשלים?

- עדי פרויקטים לא מציאותיים או לא ברורים מפסיק / הדרישות אינן מוגדרות היטב.
- אומדנים לא מדוייקים של המשתמשים הנדרשים.
- דיווח לキー לגבי מצב הפרויקט / סיכונים לא מנוהלים.
- תקשורת לקויה בין הלוקו למפתחים והמשתמשים / פוליטיקה של בעלי עניין.
- שימוש בטכנולוגיה לא בשלה / פרקטיקות פיתוח מורשות.
- אי יכולת לנוהל את מרכיבות הפרויקט / ניהול לキー של הפרויקט.
- לחצים מסחריים.

פיתוח תוכנה זה קשה

لتוכנה 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 מקנית

בפיתוח תוכנה יש 2 סוגים מרכיביות לפתור:

מרכיבות מקנית:

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

מרכבות מהותיות:

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

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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 ו��ינה.
- Invisibility: בלתי נראה. לדוגמה: מעצב/אדריכל/נגר יכול להציג בפני לקוחותיו שרטוט המתאר איך יראה המוצר בסוף. בתוכנה דבר זה אינו אפשרי.

דרכים לצמצום המורכבות

- שימוש ב high level languages גורם לשיפור של עד פי 5 בפרופודוקטיביות.
- שימוש במערכות מומחה, יכול להציג שיטות עבודה מומלצות ולעזר למפתחים מתחילה.
- DeepCoder Autonomic Programming
- מציאת באגים בתחילת התהילה.
- סביבות וכליים. לדוגמה Eclipse, Visual studio וכו' ו��ינה.
- שימוש בשיטות עבודה מתקדמות, כמו agile.
- קניית תוכנת מדף.
- חידוד הדרישות ובניה אב טיפוס.
- שינוי הגישה של פיתוח מערכת התוכנה, כמו שהמוה גדול ומתווספות יכולות, כך גם בתוכנה,
- התוכנה גדולה ומ�택תית ויש לה יותר יכולות.

הנדסת תוכנה

הנדסת תוכנה הchallenge להשתתף בסוף שנות השישים של ה-100 הקודמת, על רקע משבר תוכנה. הנדסת תוכנה מתיחסת למחזור 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מחזור החיים של מערכת תוכנה:

הנדסת תוכנה בראיה מערכית

הגדרה:

- מערכת היא צירוף של מרכיבים (אלמנטים) הפעלים במשותף ומאורגנים לצורך השגת מטרה מסוימת אחת או יותר.
- מערכת מאורגנת במבנה היררכי, רקורסיבי, מרכיב של מערכת יכול להיות מערכת עצמה.
- מערכת עניין, מערכת אליה תחול 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: BRAINTHE.
- רמת המערכת. למשל: מערכת ניהול שירות הקופה.
- רמת פריט התוכנה. למשל: מערכת ניהול תורים.
- רמת רכיב התוכנה. למשל: מסך משתמש של התור.
- רמת יחידת התוכנה. למשל: מסך הזדהות, מסך ביטול תור ועוד.

מערכת מבנה ותפקוד

מערכת (בכל רמה) מוגדרת על ידי המאפיינים הבאים:

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

מוצר תוכנה מסופק באחת משלושת הצורות:

1. מוצר עצמאי: מגע בדיסק קשיח או בהורדה מהאינטרנט מועד 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

"**אבולוציה של מערכת, מוצר, שירות, פרויקט או ישות אחרת מעשה ידי אדם משלב הקונספטציה ועד הוצאת המוצר משירות"**

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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 היוזם -> שלב הדרישות -> שלב הניתוח -> שלב התוכן -> שלב המימוש -> שלב השילוב -> שלב התחזקה -> פרישה

שלב היוזם

פעולות בשלב היוזם

- הגדרת הלקוחות.
- מציאת מומחה ישום.
- ניתוח מצב ק"ם, בדיקת מערכות דומות ק"מות.
- הגדרת מטרות יעדים.
- הגדרת אלוצים, מגבלות וסיכון.
- הגדרת תועלות וחסכנות.
- בדיקת יסודות ועלות/תועלות.
- גיבוש צוות התיכנון של המערכת.
- קביעות לוי'ז ומשאבים.

הגדרת הלקוק

זיהוי הלקוחות האפשריים:

- במקרה של מערכת בתוך הארגון:
 - זיהוי של סוג 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 <https://github.com/shaynaor>

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 אצלו את דרישות הארגון מהמערכת.
- מומחה היישום ילווה את המערכת משלב הייזום ועד הטמעת המערכת בארגון.
- מומחה היישום יחוות 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 מצבן? מה ניתן ללמידה מהן?

מטרות ויעדים

מטרות:

- תיאור כללי של התכונות שרצים שיהיו במערכת. דברים שרצים שהמערכת תדע/תבצע לדוגמה: שיפור השירות הנitin ללקוחות החברה.

יעדים:

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

מודל SMART:

יעדים צרכיים להיות מוגדרים לפי SMART:

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

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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.

אליזמים מגבלות וסיכון

ניתוח בעיות שועלות לנובע מאליזמים שונים:

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

סיכון:

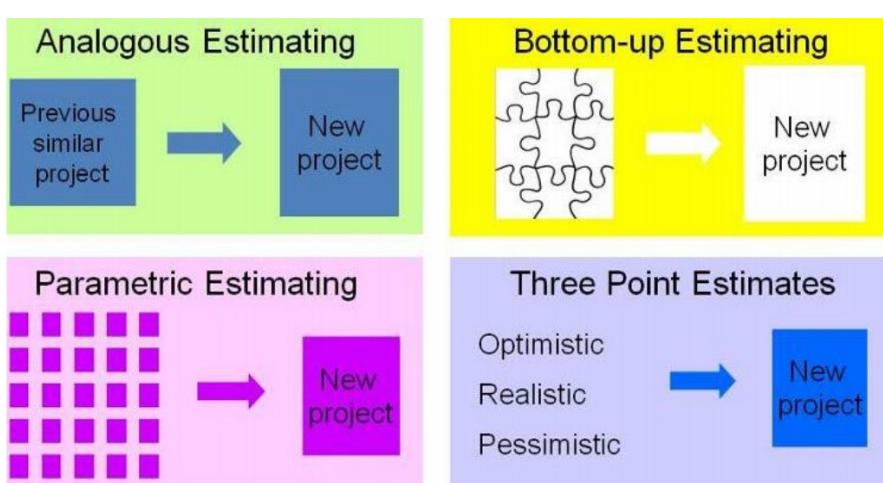
- כל דבר שועלול לקרוות ולהפחית או להקטין את הסיכויים להצלחה בפרויקט.

בדיקות שימוש ועלוות/תועלת

שאלות שנוצרן לשאול את עצמנו:

- האם צפויים קשיים/מגבליות בתחום הגדרת היישום ?
- האם מדובר בטכנולוגיה חדשה ובולטית 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שיטות לבדיקה כדיות כלכליות:

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

חשיבות של כלי המדידה:

- קיימת מוגמה למדוד הכל בכספי.
- עזר בהשגת תמיית ההנאה וקבלת תקציב.
- הגדרת היקף ההשקעה הכספי במערכת.
- כל' לבחינת אלטרנטיביות.
- ניסוי להפוך הנחות סובייקטיביות, השערות ותקויות לעובדות ומדוים.

האתגרים בכל' המדידה:

- הגדרת הפרמטרים המיצגים את התועלת והחסכנות של המערכת.
- הגדרת ערכי המצביע הקים לעומת הצפי בכל' אותו 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.

- PN זהה מונח מתחום המימון המשמש לניטוח רווחיות של השקעה או פרויקט מסוים.
- הערך הנוכחי הנקה מרכיב מהשו 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 צפואה מהמערכת לערך הנוכחי וכן נוכל לבדוק האם המערכת כdíaת או לא.

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

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

דוגמא

- המספרת שלנו צפואה במשר חמש שנים להבדיל את מס' היקחות ב- 50 כ"כ בשנה.

כל לילה מביבס קיופת חולים 1000 ש"ח לשנה.

ערך הריוון הוא 10%.

- מחירי לויות הונגה הוא 18,000 בתיהילא ואחר כרך עשרה 400,000 לישנה לתחזקה ושיפורים

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

שנה	5	4	3	2	1	0	סך
ערך הריוון	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
הוצאות	532,550	377,300	240,700	128,050	45,450	0	750,000
ערך הריוון	18,000	18,000	18,000	18,000	400,000	400,000	468,220
הוצאות	11,178	12,294	13,518	14,868	16,362	400,000	468,220
הכנסות	468,220	457,042	444,748	431,230	416,362	400,000	468,220

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

כללי אצבע:

ה NPV מראה אינדיקציה האם כדאי המשיך בפיתוח המערכת או לא, אך הוא לא חיבר להפשיע באופן חד משמעי.

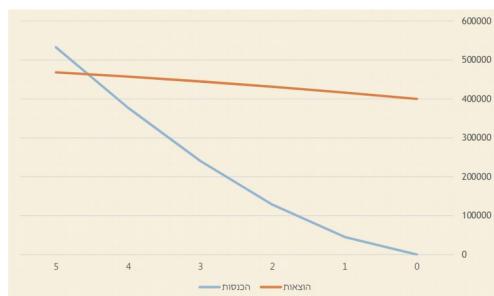
If...	It means...	Then...
NPV > 0	The investment would add value to the firm.	The project may be accepted.
NPV < 0	The investment would subtract value from the firm.	The project should be rejected.
NPV = 0	The investment would neither gain nor lose value for the firm.	We should be indifferent in the 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.
- מנוח המציין את היחס בין הכספי והמשאבים שהושקעו בפעולות עסקית כלשהי לבין ההכנסות שנבעו מהם ממשאים (פיתוח מערכת חדשה במקרה שלנו).
- נוסחת ROI בסיסית:

$$\text{הזרה השקעה \%} = \frac{\text{הרווח השקעה}}{\text{הוצאות}} * 100\%$$

: payback analysis



- ניתוח ההחזר, חשוב לקביעת משך הזמן שייקח לCAPEX להחזיר את השקעותיה.
- בדרכם כלל מציגים את זה בתגר גרף של ה蔚ות והתועלות וכן אפשר לראות בו מתי התועלות עלות על ה蔚ות.

גיבוש צוות התיכנון של המערכת (ועדת היגוי)

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

- הועדה כוללת נציגים מתחומי ידע שונים:
 - טכנולוגיה: מחשב, תשתיות, DB.
 - משאבי אנוש.
 - כספי.
 - נציגי הלוקוחות השונים.
- הועדה מתגבשת בשלב זה (יום) וממשיכה בעבודתה עד העלאת המערכת לאויר.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

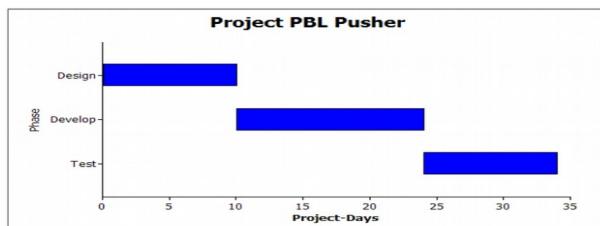
תפקידה של ועדת היגיון:

- 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:

- סיום האפיון.
- סיום הפיתוח.
- סיום הבדיקות.
- מועד העליה לאויר.



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

שליהם יכול להיות לחיבור או לשילילה.

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

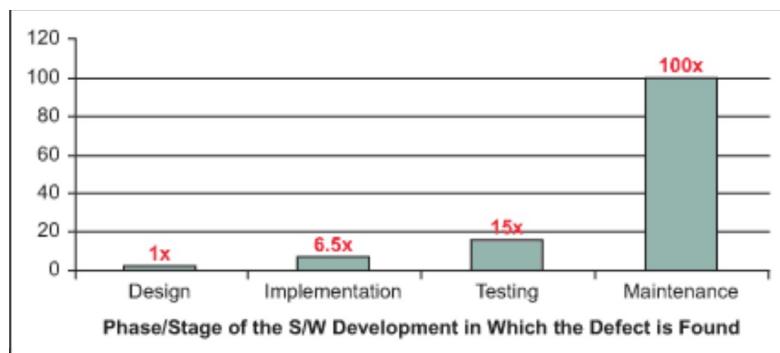
נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

חשיבות הגדרת הדרישות

- ניהול דרישות מקנה סיכי גבוה יותר שתוצרים עמו על דרישות הלוקו.
- ניהול ובקירה של הדרישות ממקדים את תהליכי האפיון והפיתוח של דרישות הלוקו.
- ניהול דרישות משפר את יכולת לבצע שינויים במרירות תוך בקרה ובוחנת השפעתם על דרישות אחרות.
- תיאום ציפיות בין ספק לлокוח על בסיס מנותח, מסוכם ומואושר.
- שיפור יכולת לבצע בקרה על התקדמות הפROYיקט.
- שיפור יכולת לביצוע אמידת עלויות ע"י ניתוח של עלות תועלות עבור כל הדרישות.

עלויות תיקון שגיאות בשלבים השונים



עודף דרישות

יש לשים לב להגדיר בשלב זה רק את מה שצורך. 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

: Functional Requirements (עסקיות)

Operational Requirement

דרישת המתיחסת לתפעול, לאינטראקציה או להתנהגות של המוצר.

לדוגמה: פעולות, תרחישים, תగובות לאירועים.

דוגמאות לדרישות:

○ המערכת תאפשר להזין הזמנות מלוקה למוצרים שבמלאי.

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

דרישת מידע Data Requirement

דרישת המתיחסת לשויות המידע ולנתונים בהן נדרשת התוכנה לטפל.

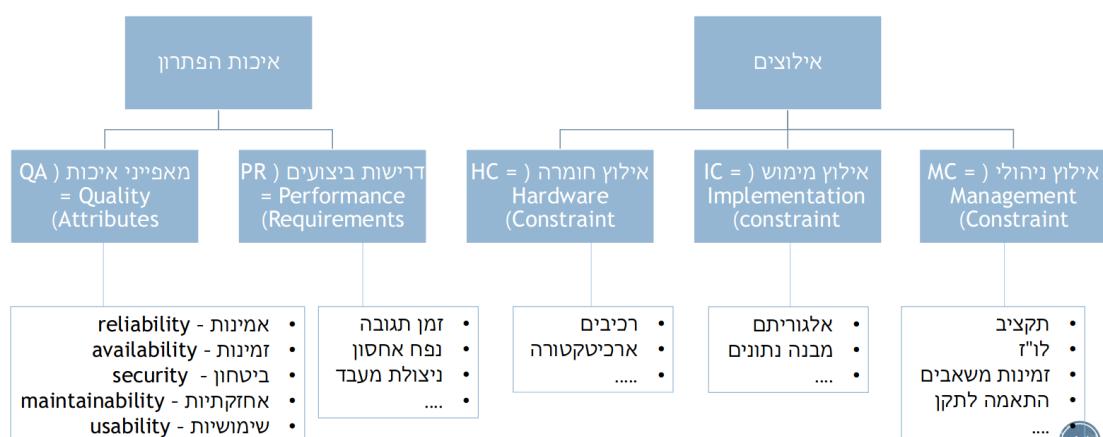
לדוגמה: נתונים ומבנה נתונים, מאגרי מידע / בסיסי נתונים, דרישות קלט ופלט.

דוגמאות לדרישות:

○ עבור כל פריט שיצג יש להציג את שמו,;brקוד שלו וצבעו.

○ לכל פריט בהזמנה יש לרשום יחידת מידת, כמות ומחיר ליחידה מידת.

דרישות לא פונקציונליות



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מהי דרישת אינטיטית ?

- **dentified:** בדידה, מזוהה חד ערכית, שייכות ברורה.
- **Understandable:** מובנת (ברורה, מדויקת) – מנוסחת בשפת הלשון.
- **Necessary:** הכרחית – בעלת תרומה שימושית לשיפור תהליכי העבודה.
- **Complete:** שלמה.
- **Consistent:** עקבית (לא סותרת דרישות אחרות).
- **Unambiguous:** לא עמויה (חד משמעות).
- **Verifiable:** ניתנת לבדיקה באמצעות מבחני קבלה.
- **Traceable:** עקיבה (גם לדרישות ברמה גבוהה יותר וגם בהמשך האפיון).
- **Prioritized:** מותעדפת.

שלב הניתנות

שלב זה נקרא גם requirements specification.

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

UML(Unified Modeling Language)

שפת סימולים גרפית וטקסטואלית לעיצוב מערכות מונחות עצמים. המודלים מתאימים הן ללקוק ולשימושם 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:

- רבוי 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

CASE כלים

תוכנות ייעודיות ליצירת מודלים של המערכת.

Visio, Rational Rose.

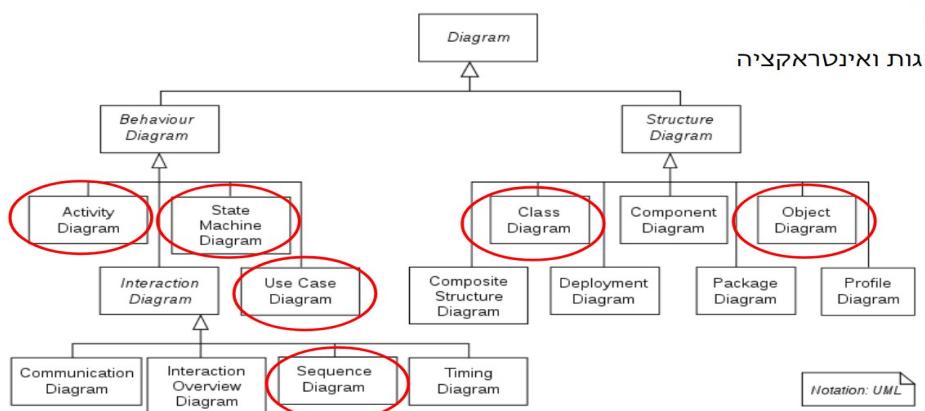
ישנים גם כלים חינמיים כמו:

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

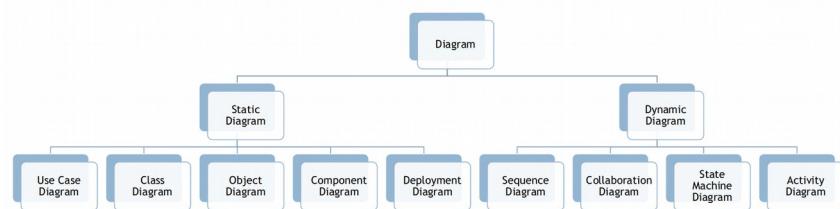
* <https://www.draw.io>

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.

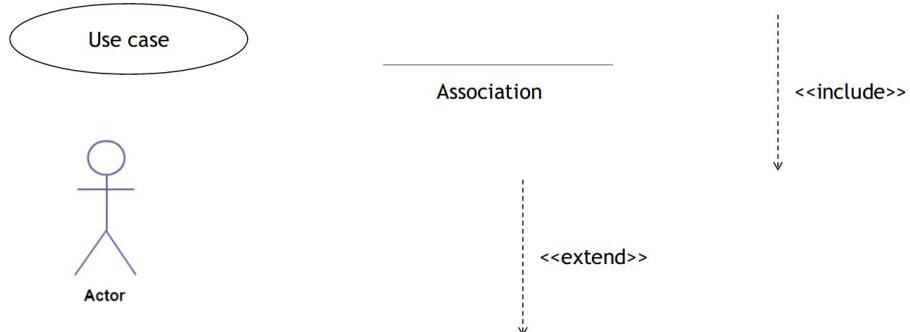
לכל דיאגרמה מוסיפים תיעוד הכלול את פירוט הפעולה, השחקנים, תדרות השימוש של הפונקציה, תנאים מקדים להפעלת הפונקציה ודברים שקורים אחריו ביצוע הפונקציה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

mbosus על המציגות של אביגיל שטקל ומירב שקרון

שיטות הSIMON:

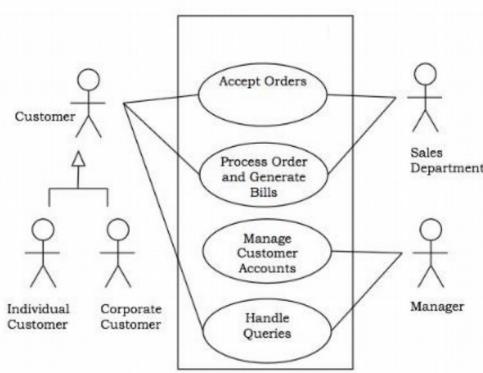


מי השחקנים ?

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

שאלות שיכלוט למצוא את השחקנים במערכת:

Use Case Diagram למערכת הזמן



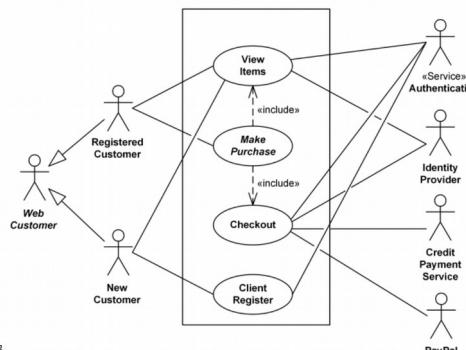
מי משתמש במערכת ?

מי מתќין את המערכת ?
מי מפעיל את המערכת ?
מי מתחזק את המערכת ?

דוגמאות ל use case :

- לקוח 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

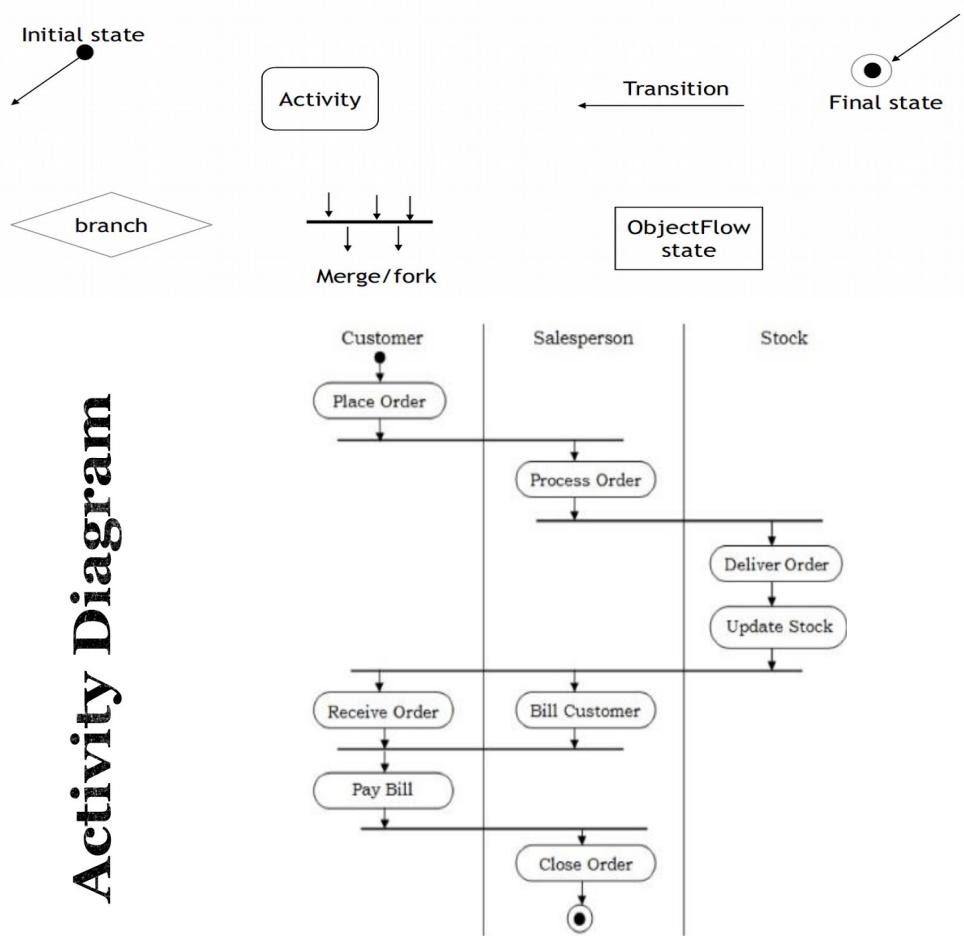
נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

Activity Diagram

תרשים פעילות מציג את הזרימה מפעולות לפעולות בתחום המערכת ובין השחקנים.
דרך להציג פעילות במבנה לדומה workflow.
כל פעולה בuse case הופכת לתרשים פעולה אחד.

שיטה הסימון



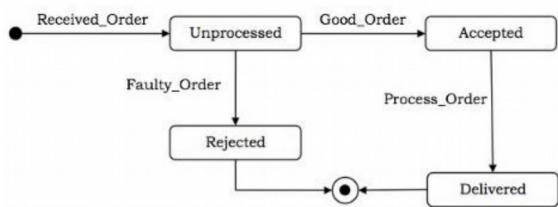
Activity Diagram

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

State Machine Diagram

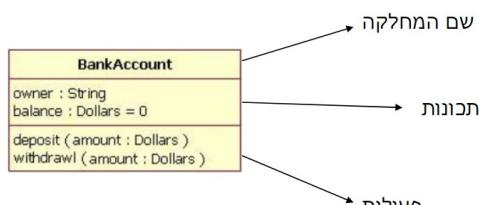


State Machine Diagram
 בתרשים מצבים שמים את האובייקט במרכז.
 התרשים יכול לתאר את זרימת האובייקט ממקום אחד
 לשנייה ואת המצביעים השונים שבו האובייקט יכול
 להיות לאורך ח' המערךת.
 תרשים זה מבוסס על האוטומט של פרופסור דוד
 הראל.
 כל תרשים מתרוך מחלוקת אחת.

שיטת הסימון



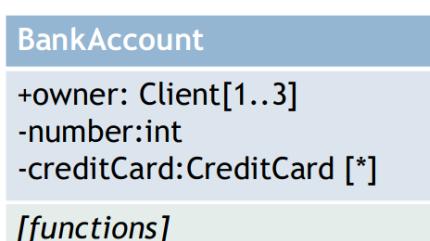
Class Diagram
 מודל של בניית המערכת על ידי מידול המחלקות, התכונות והפעולות.
 הוא תכנית האב של המחלקות הדרישות לבניית התוכנה.
 מתכוננים מפתחים את המערכת בהתאם על מודל המחלקות שהוגדר.
 זה המודל הנפוץ ביותר ב-UML.



סימן מחלוקת פשוטה:

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

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



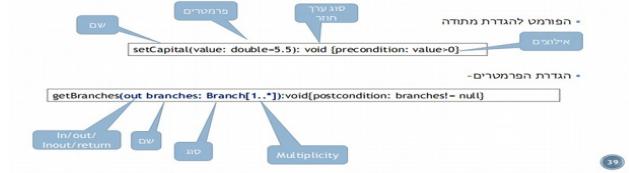
: Multiplicity
 מיד לאחר ציון ה type של המשתנה ניתן לציין את ה
 multiplicity שלו.
 את ה multiplicity מצינים באמצעות מתן טווח ערכים
 אפשרי.
 כאשר לא מצינים את ה multiplicity של משתנה
 ברירת המחדל היא 1.
 כאשר משתמשים בסימן * כדי לציין את ה
 multiplicity
 אז המשמעות ערך אחד או יותר.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מетодות שמודדרות במחלקה (Operations)

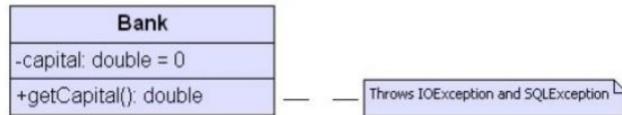


מетодות שמודדרות במחלקה (Operations)



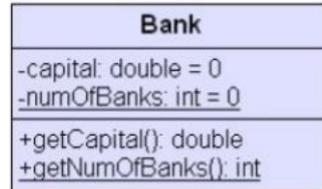
תיאור תקלות שאפשרות להתרכש:

כאשר בקריאה להפעלת METHOD יש סכנה שתתרחש תקלה (exception) מקובל להוסיף note וו תיאור החריגה.



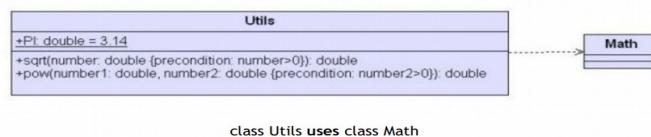
METHODS SPECIALIZATIONS:

METHOD SPECIALIZATION IS MARKED WITH AN exclamation POINT.

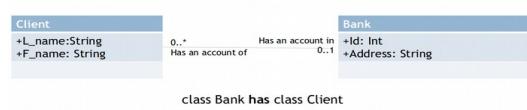


Dependency

LINKS BETWEEN CLASSES:

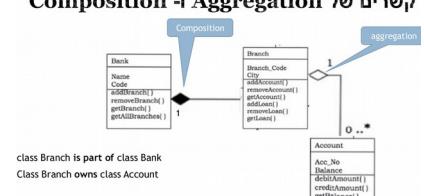


Association



class Bank has class Client

Composition - Aggregation

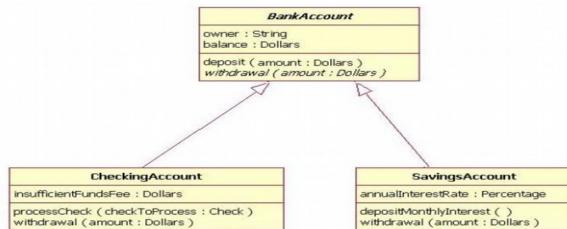


סיכום קורס הנדסת תוכנה

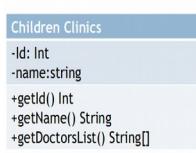
נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

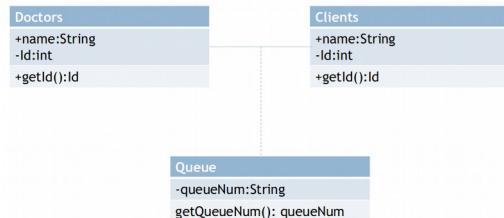
הורשה



תיאור Interfaces ומחלקות שימושיים אותם

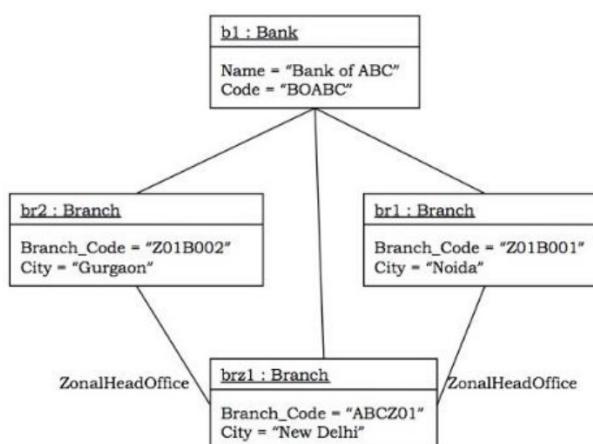
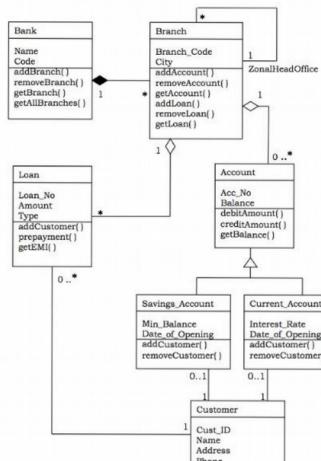


תיאור קשר בין מחלקות באמצעות Class Association



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 שיטת במודול האובייקטים דומה למודל המחלקות רק שהם מאפשרים בניית אובייקטים של המערכת מנוקדת מבט מעשית.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

Object diagram	Class diagram	
מכיל 2 חלקים: שם ורשימת פעולה	מכיל 3 חלקים: שם, רשימה תוכנות ורשימת מאפיינים	מבנה
+ הפורמט מורכב משם האובייקט+Nקודותיהם + שם המחלקה (Tom:Employee)	שם המחלקה עומדת בפני עצמה בחלק של שם המחלקה	שם המחלקה
מגדיר את הערך הנוכחי של כל תוכנה	מגדיר את התוכנות של המחלקה	רשימת התוכנות
לא כלולות	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

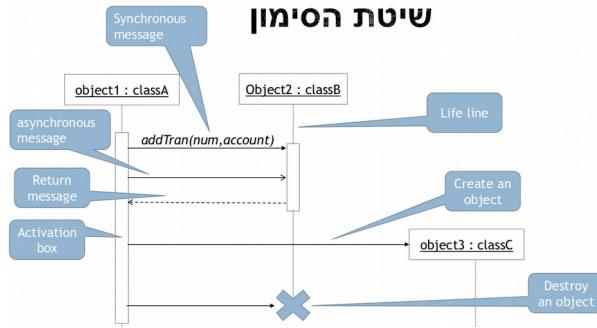
תרשים רצף הממחיש את סדר הפעולות ברגע הזמן.

תרשים רצף נכתב בצורה של תרשימים זו מימדי:

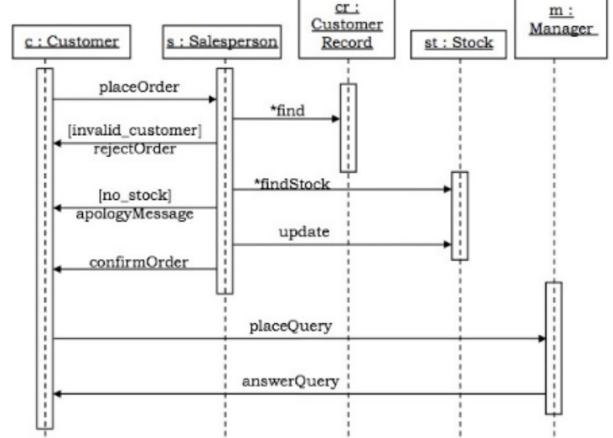
- על ציר ה-X נמצאים האובייקטים.
- על ציר ה-Y מוקומות ההודעות שהאובייקטים האלה שולחים.

לכל פונקציונליות נכון תרשימים רצף נפרד.

שיטת הסימון



Sequence Diagram



Activity	Sequence
Diagrammatic representations of interactions	Diagrammatic representations of interactions
Establishing initial conditions	Establishing initial conditions
Performing actions sequentially	Performing actions sequentially
Handling errors and exceptions	Handling errors and exceptions

אז מהו סדר הגיון?

אין!

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 <https://github.com/shaynaor>

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. זהו 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, חוליה בבית חולים.

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

אירוע: דיווח הוכחות, תנועה בחשבון בנק, כניסה פרט למלאי.

לישות יש מופעים: רשימת העובדים, רשימת הפריטים וכדומה.

ישות חזקה וישות חלשה

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

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

דוגמאות:

לקוחות, רפואיים יכולים להחשב לישות חזקה.

תור, מרשם ללקוח נחשבים לישות חלשה.

שלבי עיצוב בסיס הנתונים

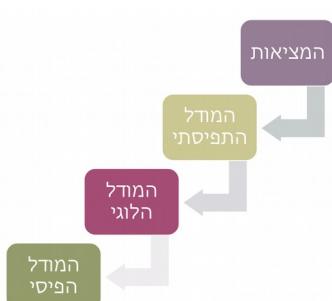
המציאות:

"באריאותא"

קליטה של החולים, רפואיים חדשים,

ניהול תורים, רזמנויות מבטי המרפקת, קשר בין לקוח לרופא, שיבוץ רפואיים במרפאות.

מי הם הישויות? לקוחות, 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

. Minimalism: קצר 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 לחלוקה נוספת.

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

דוגמה:

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.

דוגמה: ממוצע טיפולים לרופא.

ערך של תוכנה: מוגדר כתוכנה בנקודת זמן מסוימת.

דוגמה: boolean, decimal, integer.

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

בכל נקודת זמן ערך אחד או יותר.

בחזרה ל"בריאות", מהם התכונות של היישות ?

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

• מרפאות: שם רפואי, כתובות מחוץ.

• בתים מקרקחות: שם בית מקרקחת, כתובות, שם מנהל, טלפון.

מפתח של ישות key primary key

מפתח עיקרי: מוגדר zusätzlich המזהות 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.

דוגמה: 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.

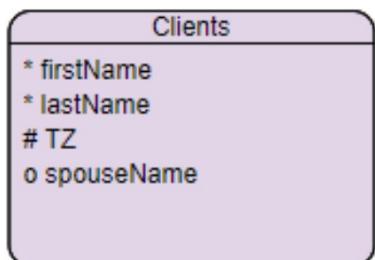
כלומר $FK(E1) = PK(E2)$

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

לדוגמה בישות "מרפאות" התוכנה מספר מחוץ היא מפתח ראשי.



בחזרה לבריאות מהם המפתחות של השיוויות ?

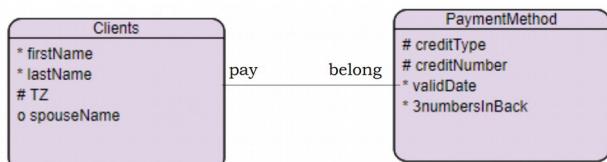
- "לקוחות": ת. זהות
- "מרפאות": קוד מרפאה.
- "תורים": ת.ז. לך, מספר רופא, תאריך ושעה.
- "רופאים": מספר רופא.

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, לקווח 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

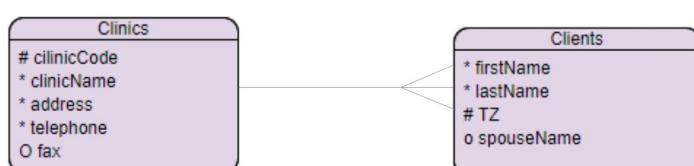
ישות אחת מתאימה לשווות אחרת.

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

קשר חד רב ערכי (1:M)

ישות אחת מתאימה לכמה ישויות בקבוצה אחרת.

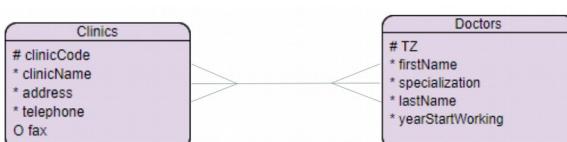
לדוגמה: כל לקוח משוייך למרפאה אחת ובכל מרפאה יש הרבה לקוחות.



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

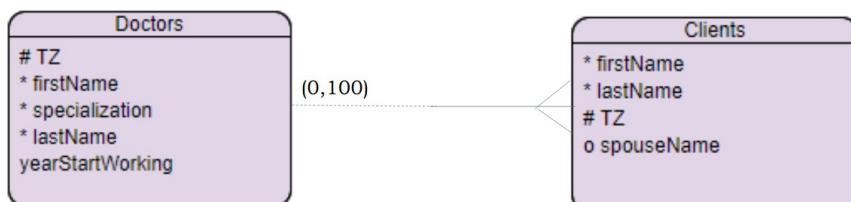
נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

קרדיינליות הקשר Relationship cardinality

קרדיינליות הקשר: מוגדר כמספר היחסות המינימלי והמקסימלי בקבוצת ישות A הקשורות לשות אחת בקבוצת ישות B.

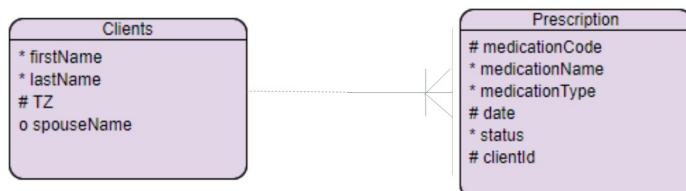
לדוגמה: רופא משפחה יכול לטפל בעד 100 לקוחות.



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.

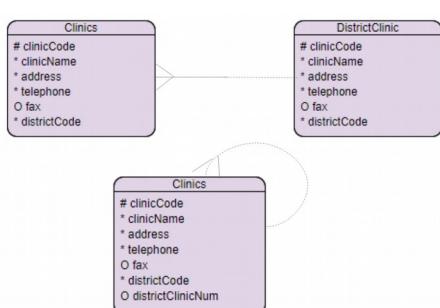
לדוגמה: לטבלה "מרשםים ללקוח" אין משמעות במידה והלקוח עזב את הקופה וכבר לא במושת בה.



קשר רקורסיבי recursive relationship

קשר רקורסיבי: מוגדר כקשר המחבר בין קבוצה A לעצמה.

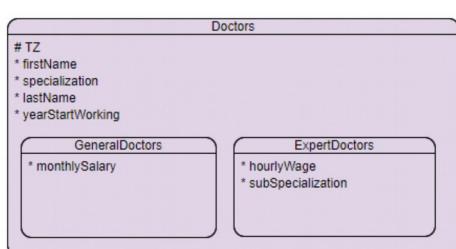
לדוגמה: מרפאה מחודשת.



ישות על ותת ישות

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

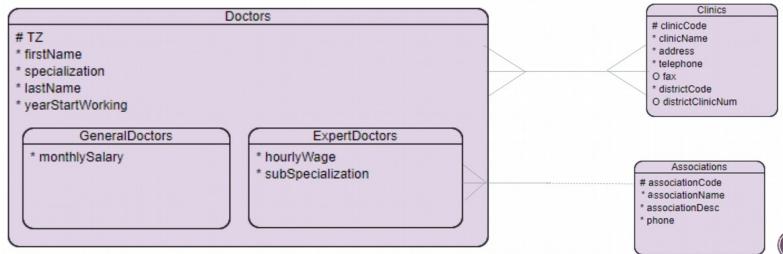
לדוגמה: כל רופא חייב להיות או רופא כללי או רופא מומחה.



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

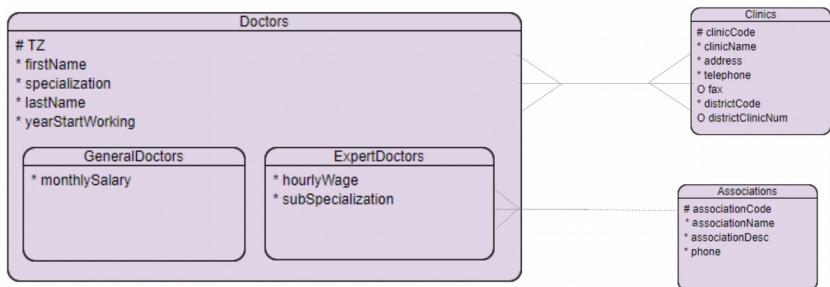
מבוסס על המציגות של אביגיל שטקל ומירב שקרון



קשר של ישות הуль הינו גם קשר של תת הישות.

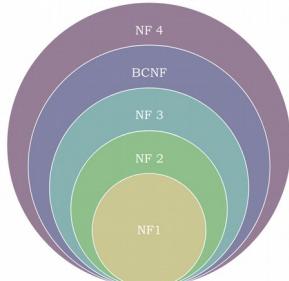
קשר של תת ישות מהוועה קשר שלה עצמה בלבד.

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 נתוניות או אנומלייה.



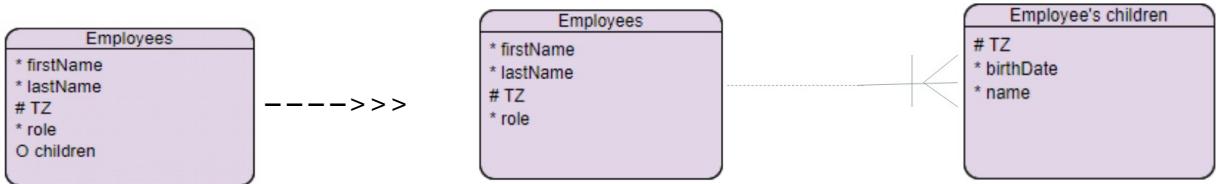
רמת נרמול	תיאור
1 NF	כל תכונה יכולה לקבל ערך אחד בלבד, ללא קבוצות
2 NF	תכונה שאינה מопיעת באך אחד מהמפתחות לא יכולה להיות תלולה בתת קבוצה ממש של מפתח
3 NF	תכונה שאינה מопיעת באך אחד מהמפתחות לא יכולה להיות תלולה בקבוצה שאינה סופר-מפתח
BCNF	תכונה לא יכולה להיות תלולה בקבוצה שאינה סופר-מפתח
4 NF	לא תלויות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

סיכום קורס הנדסת תוכנה

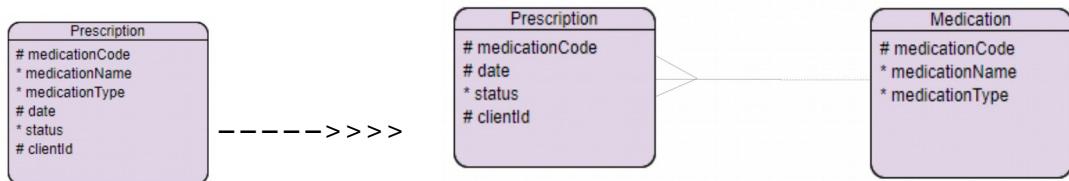
נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

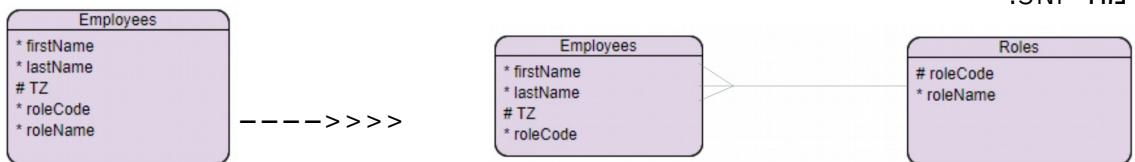
נរמול 1NF :



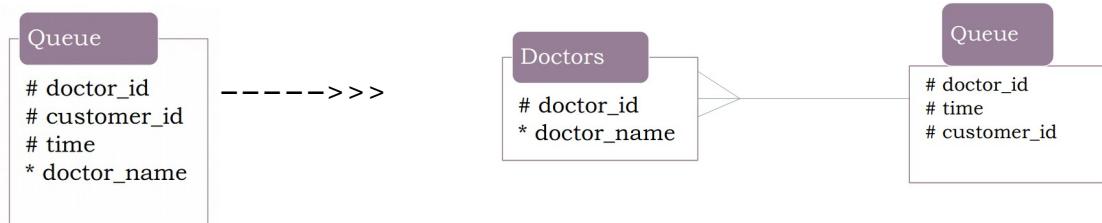
נរמול 2NF :



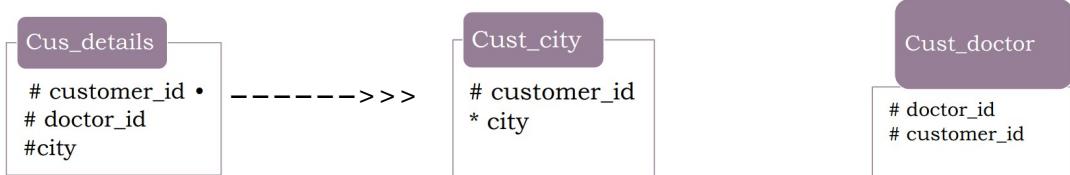
נរמול 3NF :



נរמול BCNF :



נរמול 4NF :



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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 קשיות.
- תרגום ישיות על ותתי ישיות.

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

מיפוי התכונות לעמודות בטבלה, מיפוי המפתחות הראשיים:

תכונה עם * מסמן כ null not-NN
 תכונה עם # מסמן עם (U) unique, not null (NN)
 במקורה מרכיב – נגדיר U לכולם ביחד.

הפיכת קשרים למפתחות זרים:

קשר יחיד לרבים – המפתח הזה יופיע בטבלת הרבים.

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

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

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

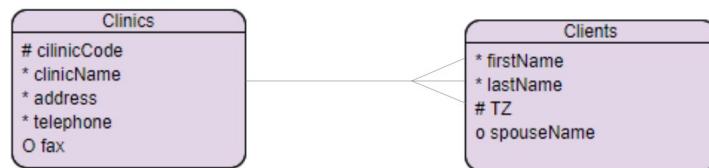


Table name: Clients

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 <https://github.com/shaynaor>

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 פטוטות פשוטות פשוטות, הביטוי הוטבע על ידי קל ג'ונסון, מהנדס מטוסים.

• שמות של משתנים ומחוקות:

- שמות משמעותיים. לדוגמה wagesPerHour לעמודה, מה.
- לשמור על עקביות בשיטת מתן השמות.
- למשתנים זמינים (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 העורות:

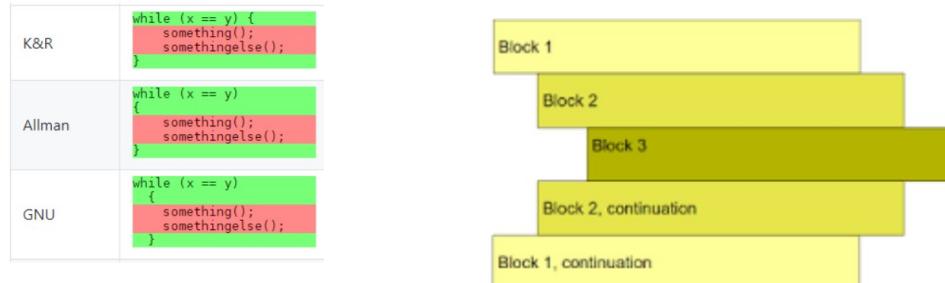
- חשוב במידה.
- הקוד-Amor להסביר את עצמו, העורות-Amor להסביר את הלמה.
- לא להשאיר קוד ישן בהערות.
- בראש המספר יש להוסיף העורה מי כתב את הקוד, מתי הוא נכתב ומה הוא אמר לעשות.
- בראש doc comments java – כל מגיע ב'יחד עם ה JDK. אפשר לנו ליצור תיעוד של הקוד שלו בפורמט של HTML. כתבים את העורות בתוך הטוויח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

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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 תכננות מרכזיות:

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

פרדיגמת הציוויל: imperative paradigm
 כתיבת רשות ציווים 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 שימוש (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

הפרדיגמה מודד שונה מהאחרות.
 הכי מתאימה לשימוש כאשר צריך להוציא מידע מעובדות בסיסיות או יחסים.
 הוכחות אוטומטיות עם בינה מלאכותית.
 מבוסס על אקסימות, כללי הסקה ושאלות.
 חיפוש שיטתי במערכת עבודות, תוך שימוש במערכת כללי הסקה.
 שפות התומכות בפרדיגמה: Prolog, Mercury, Alice

פרדיגמת מונחה עצמים: object oriented paradigm
 הפרדיגמה הכי נפוצה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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 בהיררכיה ירשה.

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, ואחריות 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()
    {
        ....
    }
}
```

פתרון

```
שוחח חישוב
/// <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;
    }
}

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

//<summary>
// Class draws a rectangle on a windows form object.
// </summary>
public class RectangleDraw
{
    public void Draw(Form form, RectangleShape rectangleShape)
    {
        SolidBrush myBrush = new SolidBrush(System.Drawing.Color.Red);
        Graphics formGraphics = form.CreateGraphics();
        formGraphics.FillRectangle(myBrush,
            rectangleShape.Width,rectangleShape.Height);
    }
}
```

- Open close principle
 המחלוקת צריכה להיות פתוחה להתרחבות אבל סגורה לשינויים. –> במקרה של שינויים, נוסיף 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();
    }
}
```

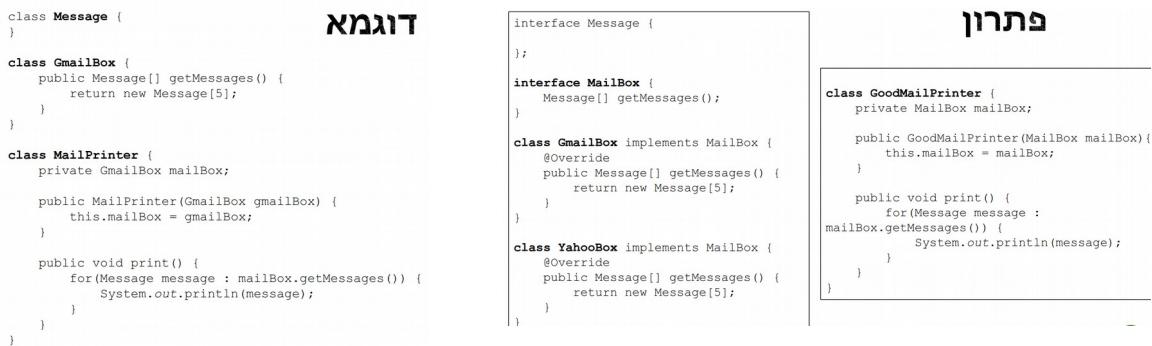
- Liskov substitution principle
 אם S היא תת מחלוקת (ירושת) של Z אז ניתן להחליף כל מופיע של Z במופיע של S מבלי שההתנהוגות תשתנה. –> מחלוקת ירושת יכולה לשנות את התנהוגות של מחלוקת האם. מי שפונה למוגדרת במחלוקת האם אינו אמור לדעת לאיזו מהמחלקות הבנות הוא פונה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

- Interface segregation principle
 - אין להכריח ל��וח 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 צרכות להיות תלויות בפרטים. –> פרטים צריכים להיות תלויים באבסטרקציות.



Code review

זהו בדיקה שיטית 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, שיפוריעילות ומורכבות הקוד, עוזר לשיתוף מידע בתוך הוצאות.
מה מתרחש? קוד מת, בעיותיעילות ומורכבות, שימוש חוזר בקוד, דיליפת זღגת מידע, דיליפת 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.
- מצטום הבעיות כתוצאה מראייה נוספת נספתח אשר משפיעה על איזות המערכת.
- אזהרה מוקדמת לגבי היבטים וחלקים חדשניים בקוד שניתן לאמוד אותם רק על ידי מדידה או מורכבות.
- שיפור רמת התחזקה של הקוד לשינויים עתידיים.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

בדיקות (שלב המימוש)

הגדרה

בדיקות הן מכלול תהליכי, שיטות, פעילויות וכליים המלאים את פיתוח המערכת והמצרים לא Orr מחזור החיים מתוך כוונה לבצע אימות ווחכת תקפות (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 ?

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

לבדוק שהיא עושה את זה עם כמה שפחות שגיאות (איכות).

לבדוק שהיא עושה את זה בצורה טובה מבחינה עצמאית בדרישות ביצועים, ובעומסים ונՓחים (=דרישות לא פונקציונליות).

מטרת עקיפה: לאסוף רישום של שגיאות תוכנה לצורך מניעת שגיאות עתידיות (פעולות מונעות או פעולהות מתקנות).

מי בודק ?

כולם!!!

תוכניתן כותב את הקוד שהוא כתב (בסביבת הפיתוח).

תוכניתנים עმיתים 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 בתהיליך 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.

הבדיקות מבוצעות בסביבת הלוקח או בסביבת pre-production.

שלבי בדיקות המסירה



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

היבטי בדיקות באפליקציות לנויידים

תהליך 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 נצרך לבדוק את כל הוויאציות!!

היבטי בדיקות באפליקציות לנויידים

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

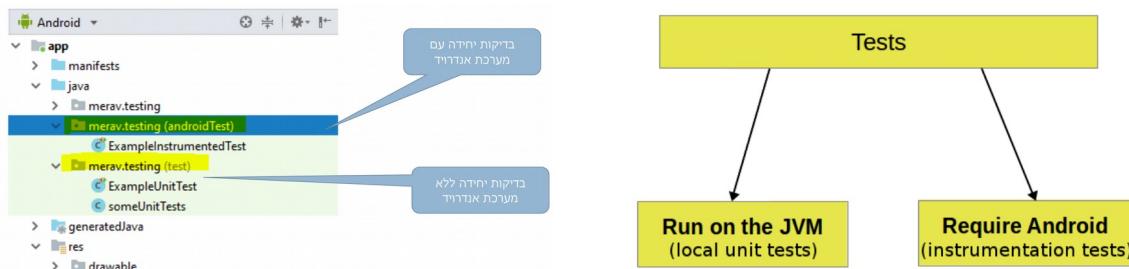
- מכשירים: גנטה להתמקד בסוגי המכשירים הזמינים באיזור היעד שלנו.
- גרסאות OS: נתמקד בגרסאות הנפוצות ביותר ונתן להם עדיפות.
- גודל המסך: באנдрואיד יש 4 אפשרויות של גודל מסך ו 4 אפשרויות של ציפוי המסך.
- נתמקד בבדיקות בגודל מסך medium ובציפויים dpi-high, dpi-medium-dpi, medium-dpi.
- פעולה סולולרי: כדאי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

דברים שכדי לבדוק באפליקציה מובייל

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

בדיקות ייחוד באנдрואיד סטודיו

שנם שני סוגי של בדיקות ייחוד:



רמת חומרת הבאגים

תיאור	שם	חווארה
- מונעתי ביצוע של רכיבים מרכזים במכשיר - ממכנתת כתוביות או בטיחון - תיקלה מוחותית בעונשו שהוגדר בעלי חשיבות גבוהה	Highly Critical	5
- משפשעה לעלה על ידו של רכיבים מרכזים במכשיר - בעיית מושגים או ביצועים המשפשעה על תיפקדת המערכת - כל קלה שברור שתשמש תכור לא יכול	Critical	4
- תיקלה בליטת במנוחה לעלי הלקטו (אם בפועל החשיפה שלה פחותה) - בעיית 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'	Medium	2
- תיקלות במנשך הגרפי לא אינטואיטיבי, טקסט או נתונים קשים לקוריאה או כתיכום, שגיאות כתיב וכו'	Minor	1

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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 בຄלות.
 שגיאות פנימיות מתגלות 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 <https://github.com/shaynaor>

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 מבקרים 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.

ה תלויות בין רכיבים חיצוניים, פנימיים ומשותפים – מובנת היטב.

מטרת התוכנה מובנת היטב.

משתמשי התוכנה מובנים היטב.

הסביבה בה השתמשו בתוכנה מובנת היטב.

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

דרישות התוכנה מובנות היטב.

:סוגי בדיקות מערכת:

בדיקות פונקציונליות, בדיקות תצוגה, בדיקת נתונים והסבות, 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 ומבחןת סטנדרטים של עיצוב.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

mbosus 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 ממפים את כל השדות, טקסטים ועוד.
2. לאחר המיפוי "בדק כל רכיב בצורה פרטנית מול ההגדרות באפין".

בדיקות נתונים והסבות

בדיקות נתונים לאחר ביצוע הסבה טכנית. לדוגמה, שינוי תשתית. בבדיקות נתונים 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 נתונים המציג את אפשרות ההסבה השונות.
2. ברמת הסיכון, בדיקה של סיכון הרשותות המוסבות בחתכים שונים למול הסיכון ברשומות הישנות.
3. ברמת השאלות, השוואת יישן 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 טולרנטיות של המערכת- מידת ההתאמה של המערכת לעבודתו של המשמש. מערכת עומסיה בתהליכיים, מסובכת עם תగובות לא ידידותיות לעבודת המשמש ולטעויות תזונה ע"י המשתמשים לטובת חלופות אחרות.

שלבי הבדיקה:

1. בדיקת נוחות התהליכים, ביצוע מספר צעדים קטנים ככל הצורך (לפחות לפעולות השכיחות), בדיקת צורת המ██ים ועוד.
2. יש להתאים את הבדיקות כמה שיותר לנסיבות הלוקו ולהתעד 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 באפין 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 לצרכים לעבוד יחד וכדומה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

בדיקות גרסיה

בדיקות רוחניות המבוצעות כל סבב בדיקות, לקרהת גרסה חדשה ועם סיום התקנת גרסה ביצור, בשביל 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 שימושshi הקצה הוי ואז קובע האם התוכנית פועלת כראוי.
 יתרונות:

- עלות נמוכה בטיבו הכספי.
- 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.
- רוטיני – 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 ומדוחחות האם התוצאות בפועל מתאימות לתוצאות הצפויות.

יתרונות:

- מהיר ויעיל – ההגדרה הראשונית לתקנת 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שלב העיצוב

UI User Interface

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

User Experience UX

חוויות משתמש: עוסק באלמנט החוויתי של השימוש ב מוצר – ולמעשה "מודד" את תחושותיו של המשתמש כתוצאה מהשימוש במוצר ואת שבעות רצונו ממנו ומהארגון שלו. האלמנט החוויתי נולד מתוך ההבנה שהצרכים זוקקים לו למענה פרקטី והן למענה רגשי – ולמעשה הם רוצים להציג שהarter, האפליקציה או התוכנה מבינים אותם ואת הצריכים שלהם, "מדובר" בשפה שלהם ויכולים להפוך את התהילה לנוח, ידידותי, פשוט ומהנה.

רקע UX

את המונח "חוויות משתמש" טבע לראשונה دونלד נורמן באמצעות שנות ה-90. UX מתיחס ל: ממשק משתמש, עיצוב, ארכיטקטורת המידע, אסטרטגיה, שיוק, טכנולוגיה, הבטים ריאטיבים, אנושיים ותחושתיים>.

מרכבי חוות משתמש

שימושיות Usability – הפקציית הקיימות במערכת וקלות התפעול שלהן (עיצוב פעולה, הזרת תוכן, קבלת משוב מתאים מן המערכת).

אסתטיות UI – המראה הכללי של ממשק המשתמש, מידת היוטו 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 הטעויות. 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 למשתמשים לבצע מטלות פשוטות ברגע שנתקלו בפעם הראשונה במשק ?

יעילות – Efficiency – ברגע שימושמשים למדו 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 חזות ?

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

סיפוק – Satisfaction – מה מידת האהדה המשתמש רוחש למערכת ?

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

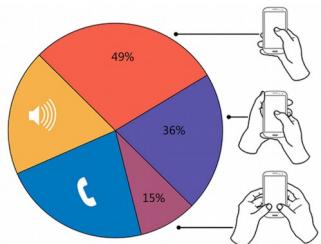
יתרונות בנית אס טוב

חוסך מאמצי הטמעה, מושך לכוחות, מצמצם את מספר הצעדים שהליך צריך לעשות כדי להציג למשתמש.

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 זיהמה.

המלצות כלליות לעיצוב המסך



הגודל המומלץ ללוחצים הוא 10-7מ"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 שפת עיצוב שפותחה על ידי גугл.

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

כל האפליקציות של גугл لأنדרואיד ורוב גרסאות הוויבר שללהן מישנות את השפה.

רוב אפליקציות האנדרואיד מישנות 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" :

ייצור שפה – ליצור שפה חזותית.

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

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

למה להשתמש ב Material Design ?

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

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

- נתן למשתמשים מראה מוכר ותחושת ביטחון.

למה לא?

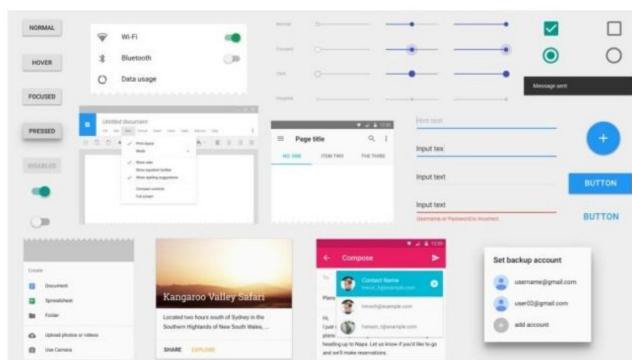
- לא תמיד נרצה להראות כמו אפליקציה של גугл.

- 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:

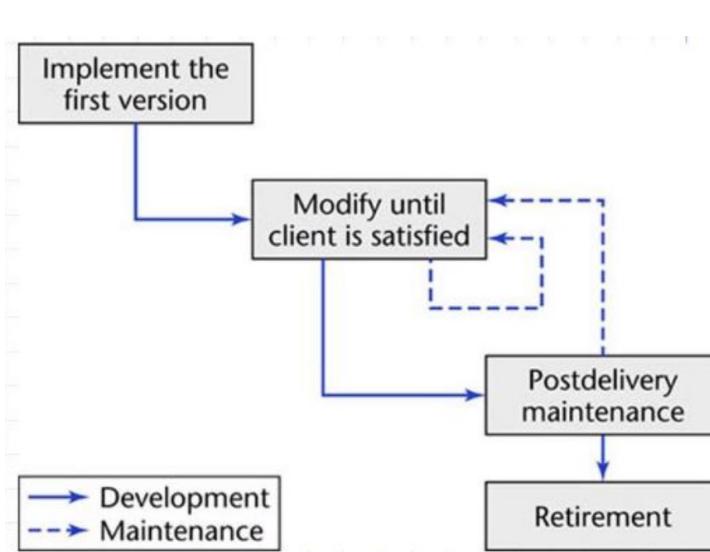
```
dependencies {  
    // ...  
    implementation 'com.android.support:design:28.0.0-beta01'  
    // ...  
}
```

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שיטות עבודה



Code and Fix

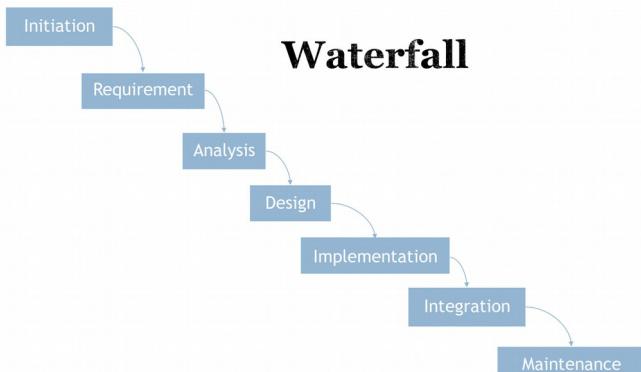
יתרונות: (במבט ראשון ובהתחלה..)

- קל לפיתוח
- מהיר
- קל לניהול
- גמיש

חסרונות:

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

Waterfall



Waterfall

יתרונות:

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

חסרונות:

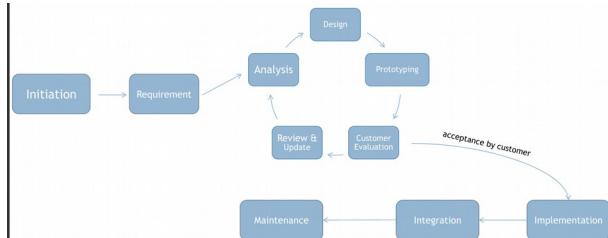
- שלב הפיתוח מגיע מאוחר בתהליך
- סיכון גבוה
- לא טוב לפרויקטים מורכבים, ארוכים או דינמיים
- קשה למדוד את ההתקדמות באמצעות שלב
- אין גמישות לשינויים

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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.

- ביצועים לא עליים.

משמש להדגמה בלבד, הוא לא המוצר הסופי!

שימושי במיחוד כאשר:

- דרישות המשתמש לא מלאות
- נושאים טכניים לא לגמרי ברורים

יתרונות:

- מבahir את דרישות הלוקו
- מקל במקדים של קשיים טכניים

חסרונות:

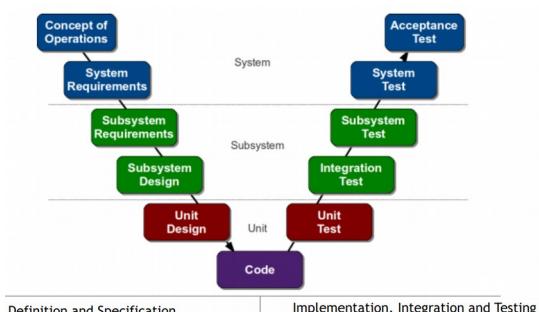
- מגדיל את עלות המערכת
- לא מקל על סיכונים שמתרגלים בתהיליך הפיתוח

V-model

המודל נועד לפשט ולשפר את תהליך הפיתוח לעומת מפל המים.

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

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



יתרונות:

- קל להבנה ולשימוש
- קל לניהול בגליל התיעוד והסדר
- השלבים ואבני הדרך מוגדרים היטב
- התהיליך והוצאות מתעדות היטב

חסרונות:

- תהליך הפיתוח מתחילה מאוחר בשלבי הפרויקט.
- חוסר וודאות, סיכון גבוה.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

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

From linear to iterative

קשה ומורכב לפתח תוכנה!

נעשות טיעות במהלך פיתוח התוכנה.

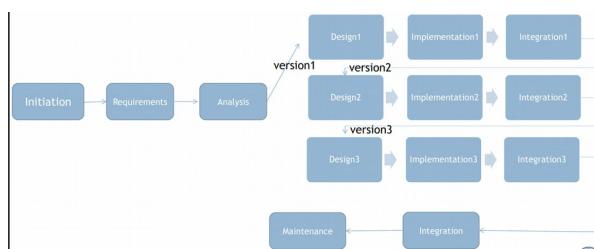
הדרישות משתנות תוך כדי הפיתוח.

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

Iterative Model

יתרונות:

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



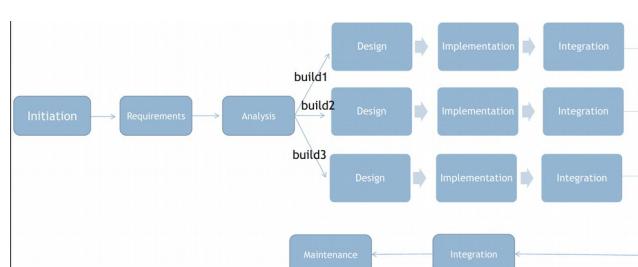
חסרונות:

- דרש משאבים גבוהים.
- אמנים העליונות 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 ובסlab מוקדם של תחילת הפיתוח.
- מודל גמיש יותר לשינויים, פחות יקר לשנות את התוכלה והדרישות.
- קל יותר לבדוק אינטרציות קטנות.
- הליקוי יכול להגיב לכל מודול.
- קל יותר לנוהל סיכונים, כי חלקיים מסוכנים מוגדרים ומטופלים באיטרציה שליהם.



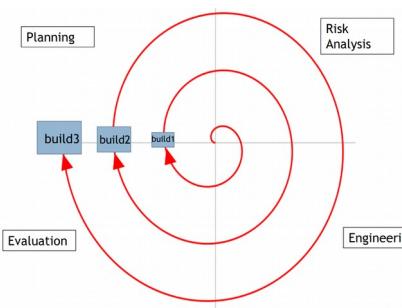
סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

חסרונות:

- דרוש תיכנון 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 להתחילה 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.

מетодולוגיות אג'יליות

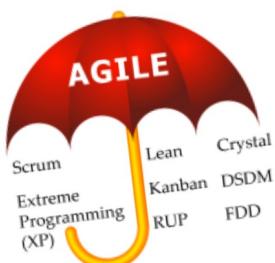
agile היא מетодולוגיה לפיתוח תוכנה זריזה, מетодולוגיה איטרטיבית שהומצאה לפיתוח תוכנה בצוותים קטנים תוך שימוש דגש על יעילות, זריזות וaicות.

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

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון



Agile

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

ישנם מספר שיטות עבודה אג'יליות, כל אחת מן השיטות דומה באג'יליות שלה, 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:
1. העדפות העלינה היא לשחרר ללקוח גרסה עובדת ובעל ערך כמו שיתר מוקדם.
 2. שינויים בדרישות הם טריגר לשיפור התוכנה ולכן נקדם בברכה.
 3. נרצה לשחרר גרסה עובדת לעיתים קרובות.
 4. נדרש שיתוף פעולה וקשר יומיומי בין מומחי היישום למפתחים לאורך כל הפרויקט.
 5. יש לחת בחברי הצוות אמון שיבצעו את העבודה ולהaddir בהם מוטיבציה.
 6. תקשורת פנים אל פנים היא הדרך העילית והטובה ביותר להעברת מידע.
 7. תוכנה עובדת היא המדריך העיקרי להתקרמות.
 8. תחזוקת המערכת היא חלק בלתי נפרד מפיתוח, וכל בעלי העניין והמפתחים צריכים לעשות זאת.
 9. מצוינות סכנית, עיצוב ותוכנן נכוון מגדים את היכולת האג'ילית.
 10. יש לפתח את מה שהכרח וללא לפתח את מה שלא.
 11. נשאיר לצוותים את האחריות ארכיטקטורה ולעיצוב.
 12. המצוות משתפר וניהיה יותר אפקטיבי לאורך זמן.

השוואה בין מודלים לינאריים ל Agile

Agile	לינארי	הדרישות בפרויקט
מעט דיען מראש, הרוב גנבה	דיעות מראש	הדרישות בפרויקט
במהשך התהליך	במהשך	שינויים בתוכנות הפרויקט
גמיש	גבוהה	שינויים בתוכנות הפרויקט
רמת הסיכון	גובהה	גובהה
מעורבות הלקוק בתהליכי הפיתוח	גובהה נאדי בהתחלה, נמוכה	גובהה נאדי בהתחלה, נמוכה
הפרויקט	במהשך	גובהה נאדי בהתחלה, נמוכה
באיוטציות	בפעם אחת	גובהה נאדי בהתחלה, נמוכה
מסירה ללקוח	לטהlixir	גובהה נאדי בהתחלה, נמוכה
אווינטציה של המודול	לטהlixir	גובהה נאדי בהתחלה, נמוכה
לأنשים	צחות גדול	גובהה נאדי בהתחלה, נמוכה
צוונים קטנים	גובהה נאדי בהתחלה, נמוכה	גובהה נאדי בהתחלה, נמוכה
הערך העסקי ללקוח	בוצעו כל הדרישות	הערך העסקי ללקוח
מודדים להצלחה של הפרויקט		הערך העסקי ללקוח

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

Agile vs. Waterfall

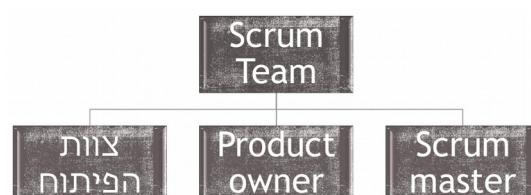
SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

Scrum

טכניקה לתחילה פיתוח זריזה המבוססת על עקרונות Agile. פותח בתחילת שנות ה-90 על ידי קן שובי וג'ף סאטרלנד. המתודולוגיה מבוססת על ההבנה שפיתוח תוכנה היא לא בעיה שנייה לפטור על ידי חיזוי או ניתוח, אלא בעיה אמפירית שנייה לפטור אותה בכלים המתאים לביעות אמפיריות. בעיות אמפיריות נפתרות בשיטה של ניסוי וטעייה – נתחילה בהדרגה, נלמד מטעויות ונשפר את המערכת כל הזמן.

*スクראם 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 תבוצע ותונשך העבודה הנדרשת על מנת לספק את התוסת המתוכננת ?

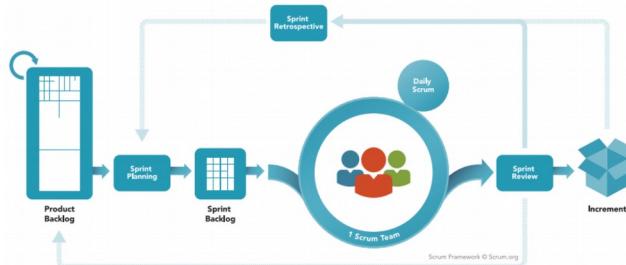
• daily scrum: פרישה קצרה של חברי הצוות המתקיימת בכל יום במהלך הספרינט ובها צוות הפיתוח מתכוון את העבודה ל 24 שעות הקרובות.

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

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 planning retrospective: פגישה המתבצעת לאחר sprint review ולפני sprint planning כדי לבדוק את עצמו וליצור תכנית שיפורים שיחולו הבא ומטרתה לתת הזדמנות לצוות scrum לבדוק את עצמו ולייצר תכנית שיפורים שיחולו על sprint הבא.

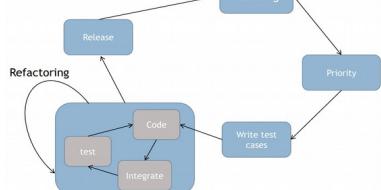


בעיות ב Scrum

- חוסר בהירות בנוגע לתקופה הסופית של הפרויקט: הלוקה 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 שהטהלה יצליח.
- נדרש שינויים מאד גדולים בארגון: גם ברמת הנהלה וגם ברמת הצוותים המפתחים.
- תפקיד scrum master: תפקיד קשה לביצוע וקשה לאיש.

XP model

Extreme Programming (XP) טכניקה לטהלה פיתוח זרי המבוסס על עקרונות Agile. נותנת דגש על מעורבות הלוקה בתהלה פיתוח התוכנה תקשורת טובה בתוך הצוותים ועבודה באינטרזיט פיתוח.



המודל פותח על ידי קנת 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

יתרונות:

- מעורבות הלקוּה מגדילה את הסיכוי שהתוכנה המיצרת תענה על הצרכים של המשתמשים.
- מקטין את הסיכון בפרויקט.
- בדיקות וaintergaczia מתמשכות מסיעות להגברת איכות העבודה.
- זמן הפיתוח קצר יותר, כי מכנים את הבדיקות בשלב ההגדרות.

חסרונות:

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

Pair Programming

2 מפתחים שושבים ביחד באותו מחשב.
 הזוגות דינמיים.

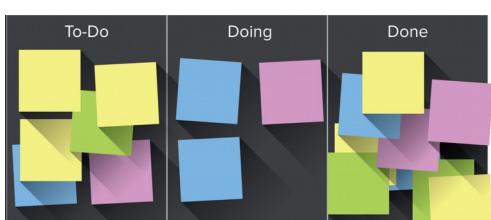
הרעיון הוא שני אנשים בודאות י כתבו קוד יותר טוב, יותר יעל ועם פחות באגים מאשר אחד.
 2 המתכנתים הם בעלי רמות שונות של ניסיון, אבל זה לא נועד לחניכה של מפתח חדש.
 ישם מקרים המראים שפיתוח בזוגות הוא מאוד מוצלח.

Kanban model

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

יתרונות:

- מוגברת הగמישות לשינויים.
- מצמצם בזבוז זמן, תמיד ממתינה למתקנת עוד משימה.
- קללה להבנה ולהטמעה, ניתן להשתמש על גבי מתודולוגיה אחרת.
- מקוצר 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

ניהול פרויקטים

הגדרה

לפי ה (Project Management Institute) PMI :

מאמץ זמני שיש לו התחלה וסוף מוגדרים.

מיוצר מוצר ייחודי (במקרה שלנו- מוצר תוכנה).

עשוי למן מטרה.

מורכב ממשימות הותלוויות ביןיהן.

למה פרויקטי תוכנה נכשלים לעיתים כל כך קרובות ?

עד הפרויקט לא מוציאותים ולא בורחים, אומדןים לא מדיינים של המשאבים הנדרשים, דרישות

מערכת אין מוגדרות היטב, דיווח לקוחות לא לגבי מצב הפרויקט, סיכונים לא מנוהלים, תקשורת לקויה בין

הלוקוט, המפתחים והמשתמשים, שימוש בטכנולוגיה לא בשלה, אי יכולת ניהול את מרכיבות

הפרויקט, פרקטיקות פיתוח מרושלות, ניהול לקוחות 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, מיומניות תוכנן ידע בשימוש בכל תכונן.

שלבי ניהול פרויקט

יום

סירה

תכנון

בקעה

ביצוע



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מחזור חיים + שלבי ניהול הפרויקט:



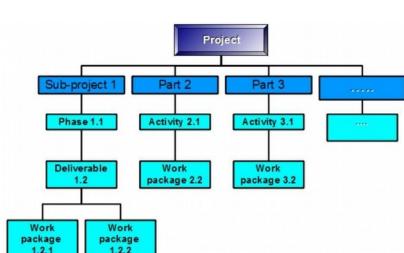
שלב הייזום

- הכרה ב הצורך לבצע: מה הרעיון שלנו/איזה בעיה אמורה להיפטר ?
- 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 לתת משימות.
- גיוס וגיבוש צוות העבודה.
- קביעת לו"ז לביצוע המשימות.
- הערכת עלויות הפרויקט.

כלים שונים בשלב התכנון:
WBS, תרשימים רשות, 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 רשות

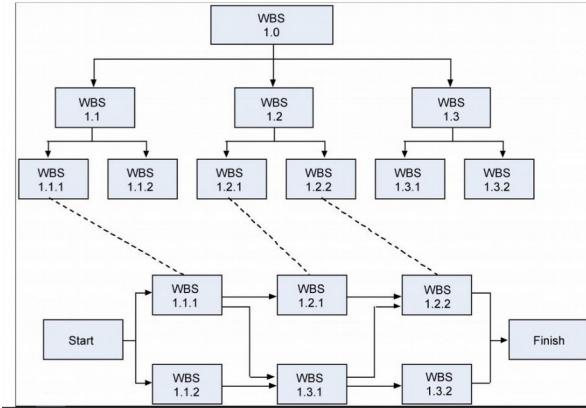
כלי להציג וניהול שרשרת המשימות בפרויקט.
מראת תלויות בין המטלות.
יכול לסייע לתכנון עיל, ארגון ובקרה הפרויקט.

סיכום קורס הנדסת תוכנה

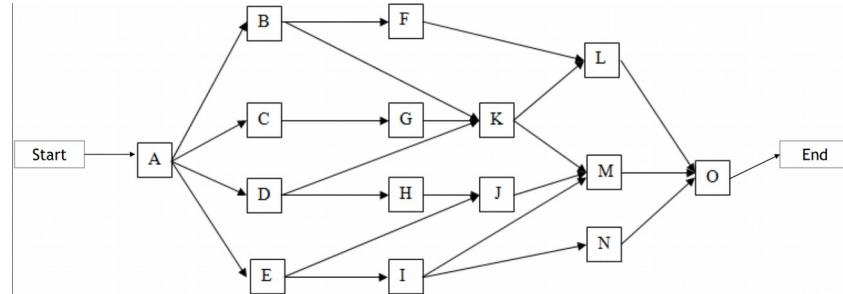
נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

מ WBS לתרשים רשת:



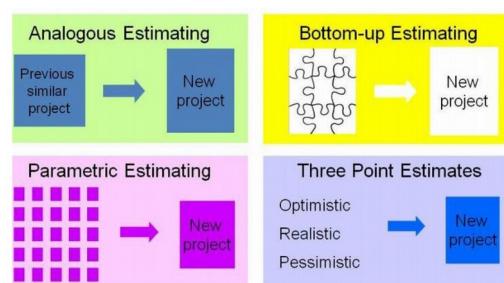
:schedule network diagram



תלות בין משימות:



אומדנים למשר הפעולות



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

תרשים (Program Evaluation Review Technique)

נוצר בשנות החמישים על מנת לסייע בניהול "צורך כל נشك בצד 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

הגדרה: שרשרת פעילותות הקשורות זו לזה, אשר כל שינוי בזמןיהם ישפיע על מועד סיום הפרויקט.

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

בפרויקט בו מתבצעות פעועלות במקביל, ישנו מספר נתיבים, נוצרים מרוחקים (slacks) המאפשרים לדוחות פעילותות מסוימות בלי לדוחות את סיום הפרויקט.

משך הזמן של הנתיב הקרייטי הוא גם המשך המינימלי של הפרויקט.

התחלת וסיום של פעילותות:

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 המטללות בפרויקט מסוים:			
תלויות	אומדן זמן	משך מללה	נש. מללה
	3	1	
1	3	2	
1	6	3	
2	8	4	
3,4	4	5	

.1 מה משך 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**

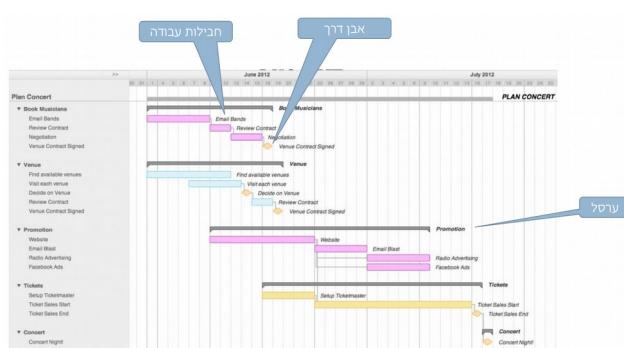
.6 נסופה מטללה 6 עם אומדן של 11 שבועות לפני מטללה 5 ואחרי מטללה 3. כמה זמן יימשך

.7 הפרויקט? **24w**

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון



Gantt

הומצא בתחילת המאה ה-20, השיטה הנפוצה ביותר להציג לוז' הפרויקט.

הציר האופקי מייצג את הזמן, הציר האנכי מייצג את חבילות 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 נתונים הטבלה ולמצוא את המידע הבא:

אומדן זמנים לפי נוסחת pert.

לשרטט תרשימים רשת.

למצוא את הנתיב הקրטי, מה המשכו ?

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

לחשב את ה-slack לכל המשימות.

חישוב נוסחת PERT:

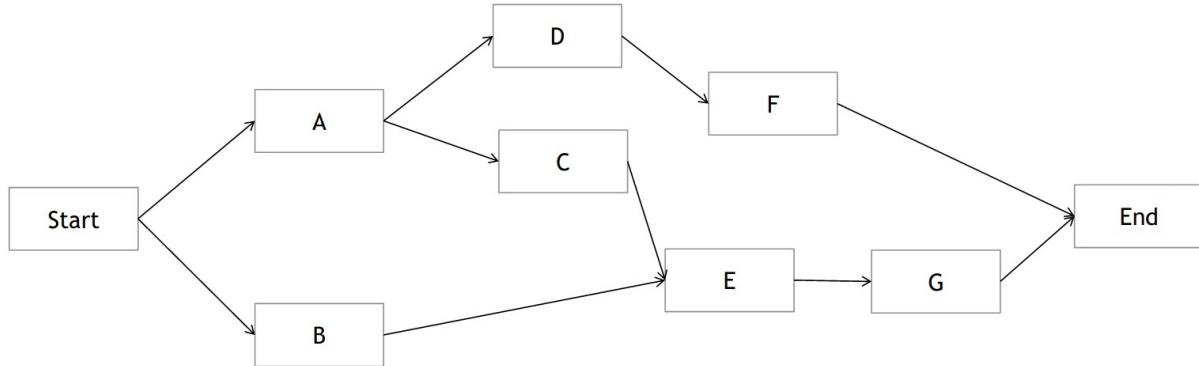
זמן צפוי	אומדן זמן				תלוויות	מטלה
	פסימי	סביר	אופטימלי			
4.00	6	4	2	--		A
5.33	9	5	3	--		B
5.17	7	5	4	A		C
6.33	10	6	4	A		D
5.17	7	5	4	B,C		E
4.50	8	4	3	D		F
5.17	8	5	3	E		G

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

תרשים רשות:



התחלת וסיום של הפעולות:

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

שאלות נוספת:

1. המשאבים שעבור מטלה D התפטר ובמקומו עובד משאב פחות מיום שייקח לו 8.33 ימים. כיצד ישפיע הפרויקט?
2. המשאבים שעבור מטלה B היה חלה ונעדר מהעבודה חמישה ימים. (מנהל הפרויקט לא מצא מחליף לימים שנעדר). כיצד ישפיע הפרויקט?
3. נוספה משימה new אחרי 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 <https://github.com/shaynaor>

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: הוספת יותר משאבים לנطיב הקרייטי, תוך שמירה על התכולה. דרש הגדרה של התקציב

Fast Tracking: ביצוע מטלות בנטיב הקרייטי 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 <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

דוגמה

תלויות	זמן	אומדן 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

נתונה רשימת המטלות בפרויקט מסוים:

הALKOC דורש שהפרויקט יסתיים תוך 30 חודשים.

Air נוכל לצמצם את הזמן?

Fast track – A–B–C–G–Crash.

העברת משאבים מ-B–C–G.

להוריד את H – צמצום תקופה.

להוריד את סטנדרטי האיכות (של משימות בנתיב הקרייטי).

ערך מזוכה

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

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

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

בשיטת זו מתרגמים את יכולת העבודה למונחים כספיים.

ביצוע מעקב אחריו התקדמות הפרויקט ובדיקה האם אנחנו מקדים או מאחרים בשלבzeit של הפרויקט

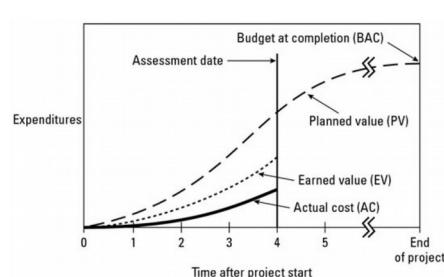
ואם אנחנו במסגרת התקציב או超נרגים ממנו.

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

היום), אשר מהויה נקודת ייחוס איחידה לכל הפרמטרים השונים.

השיטה למעשה תקופה רק לאחר שהפרויקט החל.

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



סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

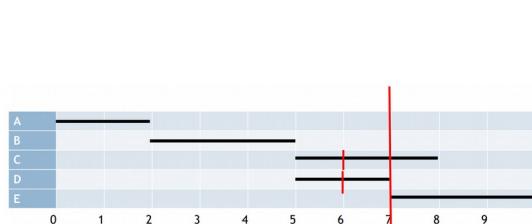
נתונים ומשתנים		
איך מוחسب	משמעות	סיביווין
$BAC * \% \text{ הביצוע בתכנון}$	Planned Value PV	
$BAC * \% \text{ הביצוע בפועל}$	Earned Value EV	
$EV - AC$	Cost Variance CV	
$EV - PV$	Schedule Variance SV	
$\frac{EV}{AC}$	Cost Performance Index CPI	
$\frac{EV}{PV}$	Schedule Performance Index SPI	
$\frac{BAC}{CPI}$	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) •



אל הסכומים
שוהים ב-7 ימים
ושווים של:

Activity	AC
A	600
B	1400
C	500
D	200
E	0
	2700

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

טלטוט ביצוע המשימות:

המ髦ינה - A

הסתמינה - B

נעשרה 1/3 מהעבודה - C

נעשרה חצי העבודה - D

אל התחילה - E

חוצים לחשב את עלות המומכה 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

עלות העבודה
שביצעו בפועל

עלות העבודה
שורייתה אמרורה
להתבצעה

What	Calculation	Answer	Interpretation
PV	$600 + 1200 + 800 + 400$	3000	הינו אמורים לעשות עבודה בעלות של 3000
EV	$600 + 1200 + 400 + 200$	2400	השלמים לעבודה 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 שאנחנו משקיעים
SPI	$2400 / 3000$	0.8	אנחנו מתקדמים ב-80%-85% המקורי
EAC	$3700 / 0.889$	4162	לפי המבזע עכשווי הפרויקט עולה 4162 בסך הכל
ETC	$4162 - 2700$	1461	צריך לוויא עד 1461 בשwil לסיים את הפרויקט
VAC	$3700 - 4162$	-462	נוצרר לויא עד 462 מעבר לתקציב בשביל לסיים את הפרויקט

סיכום קורס הנדסת תוכנה

נכתב על ידי: שי נאור <https://github.com/shaynaor>

מבוסס על המציגות של אביגיל שטקל ומירב שקרון

שלב הסגירה

שלב זה קורה אחרי שמשפיקים ללקוח את המוצר.
 בשלב זה עושים ניתוח על התהילה שעבורו חברי הצוות:

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