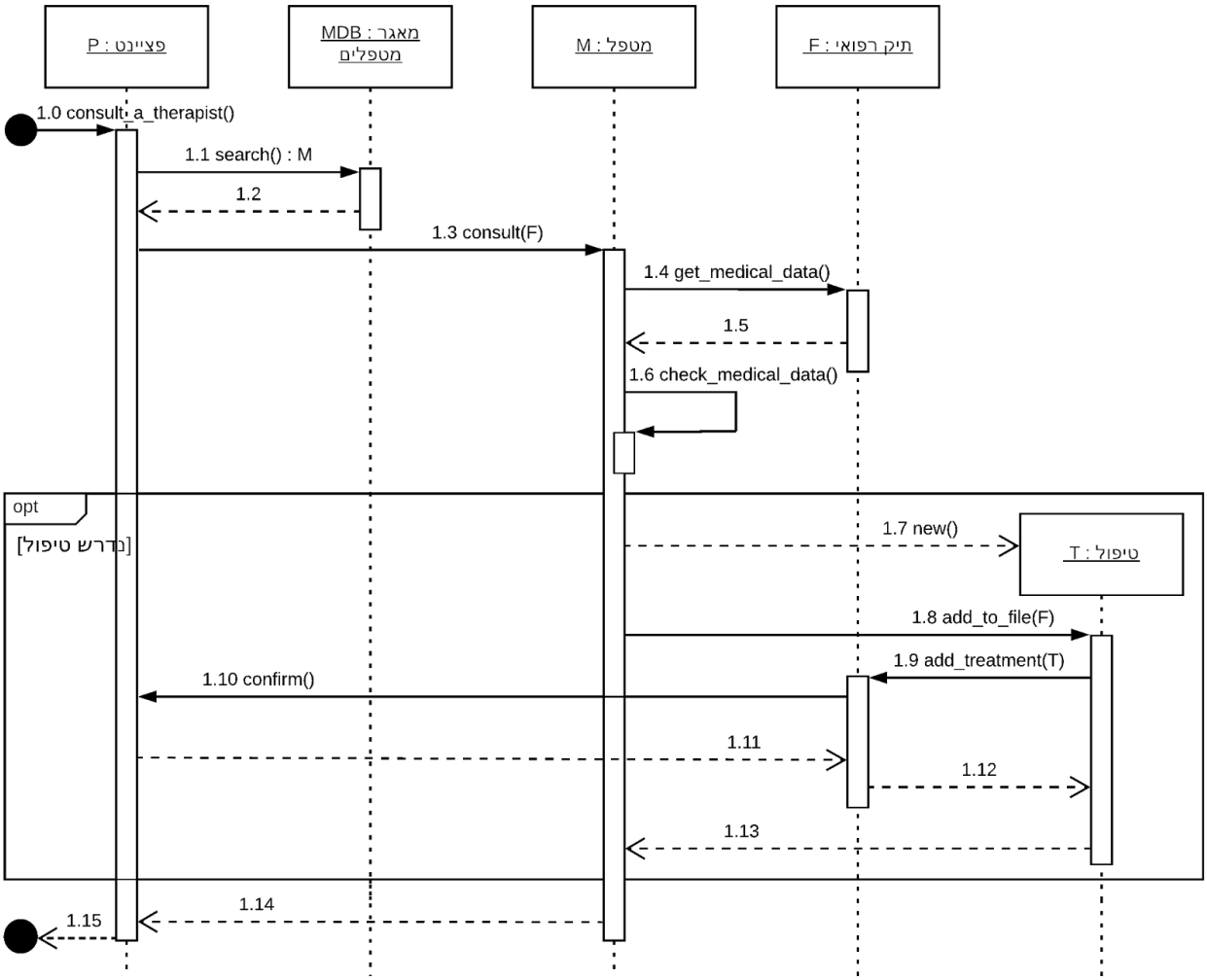
הנדסת תוכנה - מועד א' 2019

1. נתון תרשים sequence המתאר תהליך התייעצות של פציינט עם מטפל ובעקבותיו 10 משפטים.

לגבי כל משפט סמנו "נכון" אם הוא נובע מהאמור בתרשים, "לא נכון" אם הוא נמצא בסתירה לתרשים ו"לא יודע" אם לא ניתן לדעת מהתרשים האם הוא נכון או לא. (יש להתייחס אך ורק לתרשים!) (20 נק')



	משפט	תשובה
1	T הוא אובייקט ממחלקת טיפול הנוצר רק כאשר נדרש טיפול.	**נכון.** קוראים לבנאי של מחלקה T (מתודה new) רק בתוך חלון ה-opt ונכנסים לחלון הזה רק כאשר נדרש טיפול. אחרת לא נכנס לחלון ולא נקרא לבנאי.
2	האובייקט P נוצר בתחילת התרחיש ונמחק בסיומו.	**לא נכון.** לא נוצר בתחילת התהליך אובייקט P חדש, התהליך מתחיל כאשר קוראים למתודה consult_a_therapist של אובייקט P קיים. בסוף התהליך המתודה מסתיימת ומחזירה תשובה, ולא ידוע מה קורה ל-P אחר כך.

3	המתודה consult של "מטפל" מסתיימת תמיד לאחר קביעת טיפול.	**לא נכון.** קוראים למתודה consult לפני שנכנסים ל-opt, והיא מסתיימת אחרי שנגמר ה-opt. כך שהמתודה יכולה להסתיים לאחר קביעת טיפול, אם נכנסו לopt, והיא יכולה גם להסתיים בלי שנקבע טיפול, אם לא נכנסו ל-opt.
4	אובייקט T (טיפול) ייכנס לתיק הרפואי F אך ורק אם הפציינט P אישר אותו.	**לא נכון.** מי שקורא למתודה add_to_file הוא המטפל ולאובייקט P אין השפעה על המתרחש בתוך המתודה.
5	החץ המסומן במספר 1.14 מציין את הסיום והחזרה (return) של מתודת consult של מטפל.	**נכון.** 1.14 הוא קו מקווקו, כלומר הוא מציין סיום והחזרה של מתודה. המתודה האחרונה של M שהפעיל P לפני חץ 1.14 היא consult, ולכן החץ מתייחס אליה.
6	"add_treatment" היא מתודה של מחלקת "תיק רפואי" הקוראת למתודה "confirm" של מחלקת "פציינט".	**נכון.** החץ 1.9 הוא לא מקווקו, כלומר קריאה של מתודה, והוא מכוון לכיוון מחלקה F, כלומר זוהי מתודה של מחלקה F (תיק רפואי). לפני שהוחזרה תשובה לקריאה זו (התשובה היא חץ 1.12) נשלחת קריאה ממחלקה F למתודה confirm של מחלקת P.
7	למחלקת "תיק רפואי" יש 3 מתודות: get_medical_data, check_medical_data ו-add_treatment.	**לא נכון.** המתודה check_medical_data שייכת למחלקה "מטפל". ניתן לראות זה לפי חץ 1.6 שהוא קריאה למתודה הנ"ל, והוא מכוון לכיוון המחלקה "מטפל".
8	המתודה add_treatment מופעלת מתוך ה-constructor של אובייקט מסוג "טיפול".	**לא נכון.** קריאה לבנאי ב-sequence diagram מיוצגת ע"י חץ מקווקו מהמחלקה שקוראת לבנאי לאובייקט החדש (במקרה שלנו ממחלקה M לאובייקט החדש ממחלקה T, חץ 1.7). לאחר החץ הזה ממשיכים הלאה. הבנאי מחזיר אובייקט קיים, והמתודה add_treatment מופעלת אחרי שהוא כבר קיים, לכן לא יכול להיות שהמתודה הופעלה מתוך הבנאי.
9	התרשים מכסה את תרחיש ההצלחה הראשי של Use Case "התייעצות עם מטפל".	**לא ידוע.** לא נתון לנו תרשים ה-use case של המערכת.
10	כל ההחזרות (return) ממתודות מסומנות כחיצים מקווקוים מימין לשמאל.	**לא נכון.** מה שקובע את כיוון החץ הוא לאיזו מחלקה שייכת המתודה (משם יצא החץ) ואיזו מחלקה הפעילה את המתודה (לשם החץ יכוון), ואין שום מגבלה על כך שניתן לקרוא רק למתודות של מחלקות שנמצאות מימין למחלקה הקוראת. גם ניתן לראות מספר דוגמאות שמפריחות את הטענה בתרשים שלנו (1.7, 1.11).

2. (5 נק')

א. באיזה קובץ בפרויקט שלנו נוכל למצוא את הביטויים uses-feature, uses-permission? (1 נק')

ב. הסבירו את ההבדל בין uses-feature ל- uses-permission ולמה כל אחד מהם משמש. (2 נק')

ג. תנו דוגמא לביטוי אחר שנוכל להוסיף באותו קובץ (לא משהו שמופיע בו בברירת מחדל) (2 נק')

<u>תשובה</u>

א. בקובץ ה-manifest. הקובץ מכיל מידע חשוב על האפליקציה. לדוג': שם האפליקציה, גרסת האנרואיד המינימלית שהאפליקציה תומכת בה, אילו activities קיימים באפליקציה, מה ה-activity ההתחלתי, אילו הרשאות האפליקציה מבקשת ועוד הרבה.

ב. uses-permission אומר שהאפליקציה מבקשת רישות להשתמש ברכיב מסוים, אך גם מכשירים אשר לא מכילים את הרכיב יוכלו לראות את האפליקציה בחנות של גוגל.

uses-feature אומר שהאפליקציה משתמשת ברכיב מסוים, והחנות של גוגל תראה את האפליקציה רק למכשירים אשר מכילים את הרכיב.

ג. כמה דוגמאות:

בקשה לקבל הרשאה לשלוח הודעות SMS:

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

לגרום לכך שרק מכשירים עם רכיב מצפן יוכלו לראות את האפליקציה בחנות:

```
<uses-feature android:name="android.hardware.sensor.compass"
                  android:required="true" />
```

3. מה מהמשפטים הבאים מתאים לשיטת העבודה האג'ילית? סמנו מתאים/לא מתאים. **(10 נק')**

משפט	מתאים/לא מתאים
יש ליצור תשתית רחבה במערכת המתאימה לשינויים עתידיים.	לא מתאים.
קוד טוב, מסודר ויעיל הוא המדד העיקרי להתקדמות בפיתוח.	**לא מתאים.** המדד העיקרי להתקדמות לפי שיטת agile הוא תוכנה עובדת.
שלב העיצוב יבוצע ע"י צוות הפיתוח.	מתאים.
העדיפות העליונה היא לשחרר ללקוח גרסא עובדת כמה שיותר מוקדם.	מתאים.

4. **(15 נק')**

ברצוננו להעריך עלות של פרויקט.

ידוע שעלויות הרמת הפרויקט דורשים 100,000 ש"ח השקעה ראשונית ואחר כך כל חודש 10,000 ש"ח.

הפרויקט יפיק רווחים החל מהחודש הראשון של 30,000 ש"ח.

ערך היוון הוא 0.1.

א. מהו החודש הראשון שבו ה-NPV יהיה חיובי? הראו דרך חישוב!

ב. מה יהיה ה-ROI בחודש שמצאתם בסעיף 1?

נוסחת ה-NPV כפי שראינו בכיתה:

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

<u>תשובה</u>

חודש	0	1	2	3	4	5	6	7	8
הכנסות	0	27,273	24,793	22,539	20,490	18,628	16,934	15,395	13,995
סה"כ הכנסות	0	27,273	52,066	74,605	95,095	113,723	130,657	146,052	160,047
הוצאות	100,000	9,091	8,264	7,513	6,830	6,209	5,645	5,132	4,665
סה"כ הוצאות	100,000	109,091	117,355	124,868	131,698	137,907	143,552	148,684	153,349
הכנסות - הוצאות	-100,000	-81,818	-65,289	-50,263	-36,603	-24,184	-12,895	-2,632	6,698

א. החודש הראשון בו ה-NPV חיובי הוא החודש השמיני.

(הNPV יצא 6697 ו בעמודה האחרונה יצא 6698 בגלל שהמספרים בטבלה מעוגלים ולכן לא מדוייקים לגמרי.)

$$NPV(0.1,8) = \frac{-100000}{1.1^0} + \sum_{t=1}^{t=8} \frac{20000}{1.1^t}$$

$$NPV(0.1,8) = -100000 + 18182 + 16529 + 15026 + 13660 + 12418 + 11289 + 10263 + 9330 = 6697$$

ב. ה-ROI בחודש השמיני יהיה

$$ROI = 100 * \frac{benefits - costs}{costs} = 100 * \frac{6698}{153349} = 100 * 0.04367 = 4.367\%$$

5. נתונה טבלת משימות לביצוע פרויקט סודי וחשוב: **(15 נק')**

המשימה	תלויות	הערת זמן אופטימית	הערכת זמן ריאלית	הערכת זמן פסימית
A	-	2	3	4
B	-	4	5	6
C	A	4	6	8
D	A	1	3	5
E	B	5	7	9
F	D, C, E	1	2	4
G	E	8	10	10
H	G	4	5	8

א. אם יתרחשו המקרים הבאים, רשמו האם זה יאריך את זמן הפרויקט ואם כן בכמה:

1. משימה D נעשתה ב-6 חודשים.
2. משימה C התחילה 6 חודשים אחרי תחילת הפרויקט.

ב. הלקוח רוצה שזמן הפרויקט לא יעלה על 29 חודשים. האם נוכל לעמוד בזה לפי התכנון?
אם לא, הציעו דרכים שונות לעשות זאת.

ג. מה ה-slack של משימה E?

תשובה

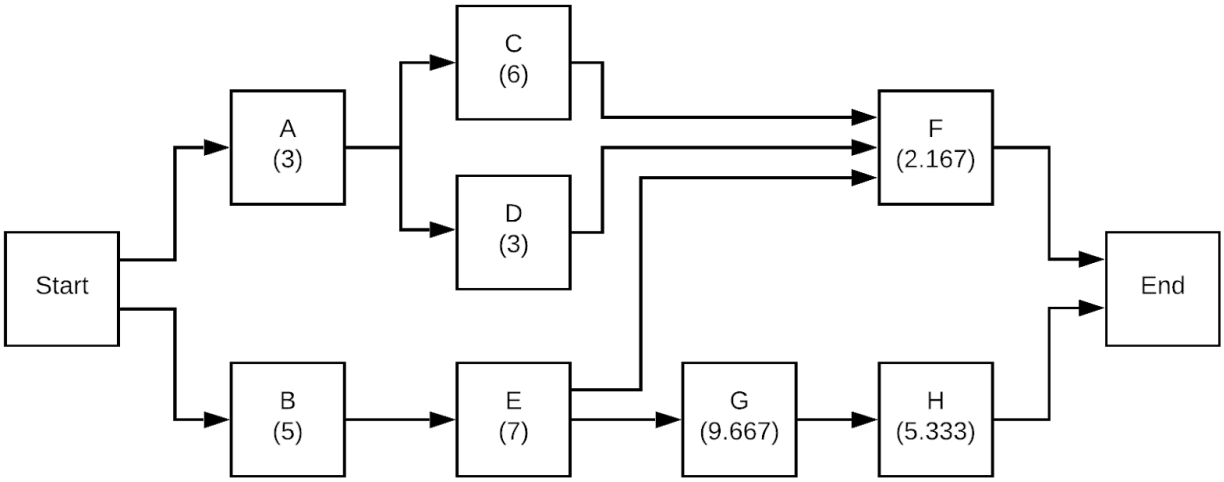
Duration (לפי נוסחת PERT) = (1/6) * (Optimistic + 4*Likely + Pessimistic)

Early Finish = Early Start + Duration

Late Start = Late Finish - Duration

Late FInish = ה-LS המינימלי מבין הפעולות התלויות בפעולה

Slack = Late Start - Early Start



	Duration	Early Start	Early FInish	Late Start	Late Finish	Slack
Start	0	0	0	0	0	0
A	3	0	3	15.833	18.833	15.833
B	5	0	5	0	5	0
C	6	3	9	18.833	24.833	15.833
D	3	3	6	21.833	24.833	18.833
E	7	5	12	5	12	0
F	2.167	12	14.167	24.833	27	12.833
G	9.667	12	21.667	12	21.667	0
H	5.333	21.667	27	21.667	27	0
End	0	27	27	27	27	0

א.

1. אם משימה D נעשתה ב6 חודשים במקום 3, זה לא יאריך את משך הפרויקט. ה-slack הוא כמות החודשים שהמשימה יכולה להתעכב מבלי להאריך את משך הפרויקט. ה-slack של משימה D הוא 18.833, זה יותר מ-3 (מספר החודשים שהמשימה התעכבה) ולכן אין בעיה.

2. משימה יכולה להתחיל בכל זמן גדול שווה ל-ES שלה וקטן שווה מה-LS מבלי להאריך את משך הפרויקט. ה-ES של משימה C הוא 3, וה-LS שלה הוא 18.833, לכן אם משימה C תתחיל בחודש 6, זה לא ישפיע על משך הפרויקט (ES = 3 < 6 < 18.833 = LS).

ב. משך הפרויקט הוא 27 חודשים, כלומר פחות מ-29, לכן נוכל לעמוד בזמנים.
במידה והיה יוצא שמשך הפרויקט המתוכנן הוא יותר מ-29 חודשים, אז היינו צריכים לתכנן מחדש את הפרויקט. אולי לוותר על חלק מהמשימות או להקטין אותן, או להשקיע יותר משאבים במשימות מסוימות כדי להקטין את המשך שלהן, וככה נוכל להקטין את משך הפרויקט.

ג. ה-Slack של משימה E הוא 0.

6. סווגו את רשימת הדרישות הבאה לפונקציונלי ולא פונקציונלי וכן סווגו לתתי הסוגים השונים.

בנוסף, ציינו עבור כל דרישה האם לדעתכם דרישה איכותית או לא. (10 נק')

<u>תשובה</u>

דרישה פונקציונלית - מה המערכת אמורה לעשות/להגיב מנקודת המבט של המשתמש.

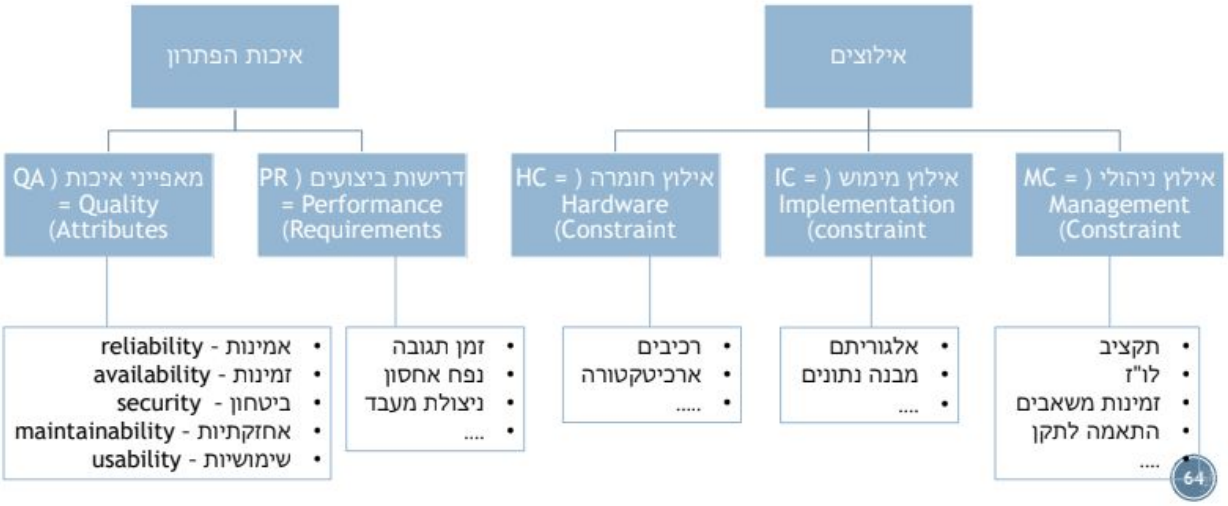
יש 2 סוגים של דרישות פונקציונליות:

1. **דרישה תפעולית (OR = Operational Requirement)** - דרישה המתייחסת לתפעול, לאינטרקציה או להתנהגות של המוצר. לדוג': פעולות, תרחישים, תגובות לאירועים וכו'.
2. **דרישת מידע (DR = Data Requirement)** - דרישה המתייחסת לישויות המידע ולנתונים בהן נדרשת התוכנה לטפל (לקלוט, לאחסן, לאחזר, לעבד, להפיק כפלט).

   לדוג': נתונים ומבני נתונים, מאגרי מידע/בסיסי נתונים, דרישות קלט/פלט.

דרישה לא פונקציונלית - דרישות המגדירות תכונות נוספות של הפתרון שצריכות להתמלא תוך כדי מילוי הדרישות הפונקציונליות. או: דרישות ותנאים המגבילים את חופש בחירת כיווני הפתרון.

סוגי דרישות לא פונקציונליות:



דרישה	פונצקיונלי?	תת סוג	איכותית?	הסביר/נימוק
בקליטת חשבון חדש יש לאמת ת.ז מול משרד הפנים.	כן	מידע	כן	פונקציונלית כי הדרישה מתייחסת לתהליך ספציפי שכלול בתהליך קליטת חשבון חדש. תת הסוג הוא מידע כי הדרישה מדברת על עיבוד מידע שמקבלים מהמשתמש. הדרישה איכותית כי ממוקדת ומוגדרת

				היטב.
פתיחת חשבון - הזמן שלוקח מרגע לחיצה על אישור ועד ההודעה כי החשבון נקלט לא יעלה על 3 שניות.	לא	דרישות ביצועים	כן	הדרישה לא פונקציונלית כי היא מגדירה אילוץ שמגביל את דרך הפתרון (הפתרון חייב לקחת בחשבון את האילוץ). תת הסוג הוא איכות פתרון כי היא מתייחסת לזמן תגובה. הדרישה איכותית כי היא מדידה ומדויקת.
המערכת צריכה להיות פשוטה להפעלה ועליה לתמוך ביכולות נגישות שונות.	לא	מאפייני איכות	לא	הדרישה לא פונקציונלית כי היא מדברת על האופי של המערכת ויוצרת אילוץ מסוים שצריך לקחת בחשבון ביצירת הפתרון. תת הסוג הוא מאפייני איכות כי מדובר על אופי ואיכות המערכת שלא קשורה לביצועים. היא לא איכותית כי היא כללית מאוד.
בעקבות תקן iso יש לאפשר שליפת חשבונות חסומים מעל שנה.	לא	אילוץ ניהולי	כן	הדרישה לא פונקציונלית כי היא מגדירה אילוץ שמגביל את הפתרון. תת הסוג הוא אילוץ ניהולי כי מדובר על עמידה בתקנים. הדרישה איכותית כי היא ממוקדת וספציפית.
על פי תקנות משרד העבודה נבדקת המעלית אחת לשישה חודשים בידי טכנאי מוסמך. בזמן הבדיקה לא ניתן להשתמש במעלית.	לא	אילוץ חומרה	כן	הדרישה לא פונקציונלית כי היא מגדירה אילוץ שמגביל את הפתרון. תת הסוג הוא אילוץ חומרה כי מתייחסים לחומרה (מעלית). הדרישה איכותית כי היא ממוקדת וספציפית.

7. מודול תוכנה עבר בהצלחה בדיקות "קופסא שחורה", כאשר הפיק פלט נכון לכל מקרי הקלט שפורטו במפרט הבדיקות.

איזה מהמשפטים הבאים נכון? **(5 נק')**

א. המודול יעבור בהצלחה כל בדיקת "קופסה לבנה".

ב. המודול לא יכשיל את בדיקות הקבלה, על בסיס אותם מפרטי בדיקות.

ג. המודול ניתן לשימוש חוזר, כפי שהוא, במערכות אחרות.

ד. מפרט הבדיקות כיסה, ככל הנראה, את כל המסלולים האפשריים בתוכו.

ה. כל המשפטים א-ד אינם נכונים.

<u>תשובה</u>

התשובה היא ה' - כל המשפטים א-ד אינם נכונים.

בדיקות קופסה שחורה - מתייחסים אך ורק לקלט ולפלט של המערכת; הבדיקה עוברת אם הפלט שהתקבל מהרכיב הנבדק (או הפעולה שביצע הרכיב) זהה לפלט שאמור להתקבל על פי הדרישות מרכיב תקין, כלומר: אם המערכת אכן עושה מה שהיא מתוכננת לעשות. בבדיקות קופסה שחורה יש חשיבות אך ורק למה, ולא לאיך: חשובה רק התוצאה, ולא הדרך שבה המערכת הגיעה אליה.

בדיקות קופסה לבנה - מתייחסים לא רק לקלט ולפלט של המערכת; הבדיקה עוברת אם הקוד של המערכת כתוב בצורה נכונה ולא נוצרים מצבים לא רצויים ולא צפויים בזמן ריצת הקוד (אובייקט מסויים נוצר כמה פעמים במקום פעם אחת, ערכים לא משתנים כצפוי וכו') כלומר: אם הקוד של המערכת עובד בצורה נכונה וכרצוי אז גם הפלטים אמורים להיות נכונים.

בדיקות קבלה - בדיקות כוללות של המערכת לפני הפעלה, במטרה לוודא עמידה בדרישות והתאמת המערכת לצרכי הלקוח. באחריות הלקוח/משתמש. מבוצע בסביבת הלקוח או בסביבת pre-productuon.

א' לא נכון כי מערכת יכולה להפיק קלט נכון בזמן שקיימות בעיות בקוד עצמו, כך שהמערכת תעבור בדיקת קופסה שחורה אך לא תעבור בדיקת קופסה לבנה.

ב' לא נכון כי יכול להיות שמודול התוכנה עובר את בדיקות הקופסה השחורה כי הוא מפיק פלט נכון, אבל לא בהכרח עונה על הדרישות של הלקוח. לדוג': זמן הביצועים ארוך מהרצוי.

ג' לא נכון כי יכולות להיות בעיות בקוד עצמו שלא מפריעות למערכת להפיק פלט נכון אך בשילוב עם מערכת אחרת יצרו מערכת שמפיקה פלט לא נכון.

ד' לא נכון כי בדיקת קופסה שחורה לא בהכרח עוברת על כל המסלולים האפשריים.

8. **(8 נק')**

א. למה חשוב שתהיה למערכת תוכנה יכולת שינוי? (2 נק')

ב. איזה מסוגי המורכבויות המהותיות באים תרשימי ה-uml לפשט? (2 נק')

ג. תנו 2 דוג' לשיטות שונות לצמצום המורכבות המהותית ו-2 דוג' לצמצום המורכבות המקרית. (4 נק')

<u>תשובה</u>

א. חשוב שתהיה למערכת תכונה יכולת שינוי כי תוכנה זה דבר משתנה, הלקוח יכול לבקש להוסיף עוד חלקים לתוכנה או להוריד חלקים קיימים, ואם התוכנה לא בנויה כך שניתן לשנות אותה בקלות - הרבה זמן ומשאבים יתבזבזו.

ב. תרשימי uml מפשטים מורכבות של להסביר ללקוח את התכונה תיראה ומורכבות של להגדיר למתכנתים על מה הם אחראים ולתת להם תמונה כללית של המערכת.

ג. **שיטות לצמצום מורכבות מקרית:**

1. פיתוח בשפות high level עוזר למתכנת להתעסק בעיקר במודל שלו, ולא לטפל בעניינים כמו ניהול זכרון.
2. סביבת פיתוח טובה עוזרת גם למקד את המתכנת בעיקר ולהשאיר את הפרטים הקטנים לסביבת הפיתוח.

שיטות לצמצום מורכבות מהותית:

1. ההבנה שתוכנה גדלה באיטרציות ותוך כדי למידה כמו המוח האנושי ובניית העבודה תחת הנחה זו.
2. שימוש ב-design טוב עוזר למתכנתים להבין "מי נגד מי", שימוש במודלים מוכתבים של הנדסת תוכנה בכלל עוזר לעשות סדר בבלגן.

9. **(12 נק')**

לפניכם קוד ב-java שמייצג ארגז כלים בו יש פטיש, מקדחה ומברג.

הסבירו באיזה עקרונות של solid הקוד לא עומד (לפחות 2).

```
sun.reflect.generics.reflectiveObjects.NotImplementedException;
import java.util.LinkedList;
import java.util.List;

class Surface {
}

class Hammer {
	public void apply(Surface s) {
		System.out.println("Applying a hammer to the surface");
	}
}

class Drill {
	public void apply(Surface s) {
		System.out.println("Applying a drill to the surface");
	}
}

class Screwdriver{
	public void apply(Surface s) {
		System.out.println("Applying a screwdriverto the surface");
	}
}


public class Toolbox {
	List<Object> tools = new LinkedList<>();

	public void addTool(Object tool) {
		tools.add(tool);
	}

	public List<Object> getTools() {
		return tools;
	}
```

```
	public void apply(Surface s) {
		for (Object tool : tools) {
			if (tool.getClass() == Hammer.class) {
				new Hammer().apply(s);
			} else if (tool.getClass() == Drill.class) {
				new Drill().apply(s);
			} else if (tool.getClass() == Screwdriver.class) {
				new Screwdriver().apply(s);
			} else {
				throw new NotImplementedException();
			}
		}
	}

	public float getPrice(Object o) {
		if (o.getClass() == Hammer.class) {
			return 21.0;
		} else if (o.getClass() == Drill.class) {
			return 18.0;
		} else if (o.getClass() == Screwdriver.class) {
			return 6.0;
		} else {
			throw new NotImplementedException();
		}
	}
}
```

<u>תשובה</u>

Single Responsibility Principle: המחלקה Toolbox אחראית על אחסון כלי עבודה וגם אחראית על לזכור את המחירים של כל סוגי כלי העבודה האפשריים, לכן יש לה 2 אחריויות. צריך להפריד את המחלקה לשתי מחלקות נפרדות - אחת שתטפל באחסון כלי עבודה, ושניה שתטפל בשמירה של מחירי סוגי הכלים השונים.

Open Close Principle: הקוד לא פתוח להתרחבות. אם נרצה להוסיף סוג חדש של כלי עבודה, נצטרך לשנות את הקוד של המתודות של המחלקה Toolbox. במקום לעשות באותן המתודות בדיקה של סוג כלי העבודה, צריך לעשות שכל כלי עבודה יממש ממשק Tool, ואז נוכל בתוך apply לקרוא ל-tool.apply(s) בלי וזה יקרא למתודה המתאימה לפי סוג כלי העבודה, כי לכל אחד יש מתודת apply לפי הממשק.