**קבוצה מספר 1**

**לי חוגי** - 311178552 - [leehugi93@gmail.com](mailto:leehugi93@gmail.com)

**איתי דוד** - 308021591 - [itaydavid22@gmail.com](mailto:itaydavid22@gmail.com)

**עידו קדוש** - 301727277 - [Idokadosh1@gmail.com](mailto:Idokadosh1@gmail.com)

**ליאור קאופמן** - 315837997 - [123liorkauffman@gmail.com](%20123liorkauffman@gmail.com)

**רביב קומם** - 316217751 - [ravivkomem@gmail.com](mailto:ravivkomem@gmail.com)

**תאריך הגשה: 10.12.19**

***שאלה 1:***

תארו את תהליך הניתוח שביצעתם למרכיב: "בחינה והחלטה" של תהליך הטיפול בבקשה. פרטו מה הם השאלות/פרטים שהתייחסתם אליהם. בתשובה התייחסו לקשרים ולמעברים

1 .ממודל Use case - למודל תהליכי המיוצג בעזרת Activity Diagram

2. ממודל תהליכים למימושו בתוכנה.

תשובה:

כאשר ניתחנו את שלב "בחינה והחלטה" של תהליך הטיפול בבקשה דבר ראשון התייחסנו לכך שחברי הועדה סוקרים את דו"ח המעריך מהשלב הקודם.

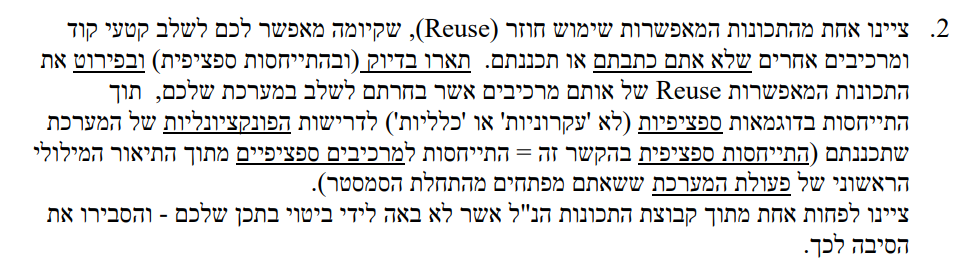
שאלנו את עצמו האם סקירת הדו"ח כוללת רק פתיחה של הדו"ח במערכת והאם הם מוסיפים הערות לאחר סקירת הדו"ח.

בשלב הבא התייחסנו לקבלת ההחלטה. השאלות שעלו הם האם כולם מקבלים החלטה אבל "המילה האחרונה" היא של היו"ר, או שרק היו"ר מקבל החלטה לפי הסקירה ששאר חברי הועדה עשו.

במודל ב- Use Case מדלנו כך שחברי הועדה סוקרים את הדו"ח ושכלול בכך קבלת החלטה, בנוסף מדלנו שהיו"ר נותן אישור סופי להחלטתם של חברי הועדה.

במודל ה- Activity Diagram מדלנו את שלב "בחינה והחלטה" כך שחברי הועדה סוקרים את הדו"ח וכותבים את חוות דעתם עליו, היו"ר מקבל את חוות דעתם ומחליט לפי הממצאים שלהם האם לאשר, לדחות או לבקש מידע נוסף בהתאם. כל החלטה שמתקבלת גוררת המשך לשלב אחר בהתאם לסיפור.

במימוש התוכנה מדלנו את השלב כך שחברי הועדה מציגים את דו"ח המעריך לפי מספר מזהה של בקשה, לאחר שהם עוברים על הדו"ח הם רושמים הערות ומסקנות שהם קיבלנו מקריאת הדו"ח בתוכנה. לאחר מכן היו"ר מקבל החלטה סופית בהתאם להערות והמסקנות שחברי הועדה כתבו בתוכנה, בהתאם להחלטה הסופית שקיבל היו"ר יחד עם עזרת חברי הועדה ממשיך הטיפול בבקשה.



***שאלה 2:***

בקוד שלנו מתבצע שימוש חוזר בפנייה למחלקה chatclient שאליה מועברת הודעה שצריך להפנות לServer.

Chatclient שיורשת מהמחלקה Abstractclient, מזמנת את המתודה SendToServer שממומשת בתוך Abstractclient ואותה קיבלנו כ"קופסה שחורה".

בסופו של דבר היא מגיע למתודה Handlemessagefromclient שנמצאת בתוך המחלקה Echoserver, אנו בעצם לא באמת מתייחסים לכל התהליך שקורה מהרגע שChatclient שלחה לServer את ההודעה עד הרגע שאנו מגיעים למתודה Handlemessagefromclient שנמצאת במחלקה Echoserver .

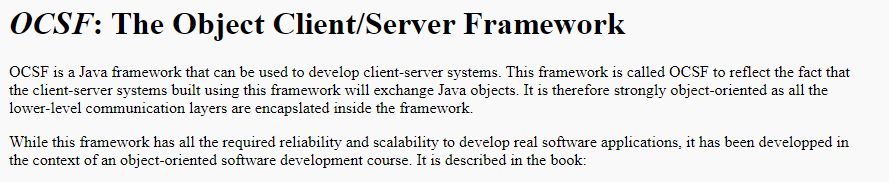
ב"בקופסה שחורה" זאת נעשה שימוש חוזר בכל פעם שChatclient פונה לServer, תכונה זו המאפשרת לנו שימוש חוזר (Reuse) נקראת הורשה (Generalization) ובעזרתה ניתן לגשת למחלקות שלא כתבנו ובעזרתן להתחבר לקוד שאנחנו כתבנו.

אחד ההתרונות לשימוש בהורשה היא שלא צריך לכתוב את אותה מתודה כמה פעמים במחלקות שונות וניתן לממש במחלקה אחת שממנה שאר המחלקות יירשו ויוכלו לממש מתודה זאת וכך להימנע מכתיבת קוד חוזר

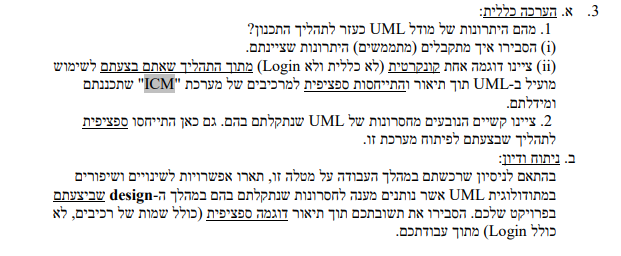
(תשובה של רביב בתהליך)

בקוד שלנו עלינו לממש יכולות תקשורת בצורת "לקוח – שרת", זוהי דרישת לקוח לגבי המערכת ולכן איננה ניתנת לשינוי או לגמישות בצד המימוש. מצד אחד דרישה זו היא דרישה קשיחה שלא מאפשרת גמישות, מצד שני זוהי דרישה הגיונית ומקובלת ומהווה סטנדרט במימוש מערכות מסוג זה בתעשייה. קיימים מימושים רבים לתקשורת זו, אשר גם עובדים בצורה מלאה וגם הכי חשוב נבדקו רבות במספר רב של פעמים במסגרת פרויקטים שונים. לכן הגענו למסקנה שאין לנו צורך אמיתי בלבצע מימוש תקשורת זו מאפס ויהיה יותר נוח ופרקטי במסגרת הזמן שניתן לביצוע העבודה להשתמש בקוד קיים ולבצע reuse עם התאמות מינוריות לצרכים של הפרויקט שלנו.

לאחר מחקר שנעשה בתוך הצוות וכמובן סיוע מקיף של צוות הקורס, בחרנו לממש תקשורת זו בעזרת תשתית OCSF (Object Client/Server Framework). זוהי תשתית שמאפשרת תקשורת לקוח שרת, כך שהם יעבירו ביניהם **(To Be Continued)**



***שאלה 3:***



1. **היתרונות של מודל UML כעזר לתהליך התכנון:**

דיאגרמות UML מהוות סטנדרט שמשמש לתאר בצורה ויזואלית וחזותית תוכנה, בעיקר עבור תוכנה מבוססת פיתוח מונחה עצמים. הדיאגרמות עוזרות לארגן, לתכנן ולראות את התוכנה. בנוסף לכך, כיוון שדיאגרמות אלו מהוות סטנדרט פיתוח הרי שהן משומשות ומקובלות בתעשייה כדרך לתאר תוכנות ומשמשות בתור שפה משותפת בין מתכנתים. כמו כן יתרון נוסף הוא הנוחות שבקריאת הדיאגרמות והאפשרות לשימוש חוזר בחלקים נרחבים מהדיאגרמה בפרויקטים שונים.

* **תיאור חזותי של המידע:**

דיאגרמת Class היא בעצם תיאור חזותי של הקשרים בין המחלקות והישויות בתוך אפליקציות ותוכנות מחשב. מחלקה היא בעצם אובייקט תכנותי שמסדר משתנים ומתודות במקום אחד על פי הקשר ביניהם. בכדי להבין את הקוד והתוכנה, זה הכרחי להבין מה כל מחלקה עושה, המידע שהיא שומרת וכיצד זה בא לידי ביטוי בתמונה הגדולה ביחס לשאר המחלקות בתוכנה. על ידי הצגת המידע בדיאגרמה, ניתן לראות ולהבין בצורה פשוטה את הקשרים בין החלקים והמחלקות השונות בתוכנה.

* **נוחות קריאה:**

דיאגרמות UML הן מאוד קריאות, הן נועדו להיות מובנות על ידי מפתחים מתחומים שונים ועוזרות להסביר את הקשרים של התוכנה בצורה פשוטה ומובנת. באופן מסורתי, בכדי להבין תוכנה, על המפתח לקרוא את הקוד בצורה ישירה. כאשר מדובר בתוכנה גדולה ייתכן ומדובר במאות אלפים ואף מיליונים של שורות קוד. כאשר ניתן לחסוך את זמן ההבנה בהרבה על ידי התבוננות בדיאגרמות אשר ממחישות את הקשרים והפעולות המתבצעות.

* **שימוש חוזר:**

באמצעות דיאגרמות של Activity וSequence ניתן למדל ולהראות קטעי קוד שרצים ופועלים בתוכנה. מפתח התוכנה יכול לזהות ולראות קטעי קוד מיותרים או לחילופין קטעי קוד שחוזרים על עצמם, ובכך בעת תהליך המימוש יוכל לבצע שימוש חוזר (re-use) בקטעי קוד אלה במקום לכתוב את אותן פונקציות מחדש ובכך לחסוך זמן.

* **סטנדרט פיתוח:**

דיאגרמות UML מהוות בעצם סטנדרט או פרוטוקול פיתוח עבור פיתוח תוכנות בשפות מונחות עצמים. כאשר יוצרים מחלקות ואובייקטים עם קשרים ביניהם, דיאגרמת UML היא מה שמאפשרת לנו לתאר בצורה ויזואלית את קשרים אלו. כיוון שהדיאגרמות משמשות כסטנדרט הן ניתנות להבנה על ידי מפתחים רבים, ובפרט זה מאפשר למפתחים חדשים לקפוץ לתוך הפרויקט בקלות ולהיות מועילים ופרודוקטיביים מהיום הראשון בעבודה.

* **כלי תכנון:**

דיאגרמות UML עוזרות לתכנן את התוכנה לפני שמבצעים את שלב המימוש והתכנות. בכלי שאנו משתמשים ישנן דרכים לבצע יצירת קוד בצורה אוטומטית על בסיס הרכיבים שתוארו בדיאגרמה. עובדה זו יכולה לעזור לחסוך זמן בשלב המימוש של התוכנה. כמו כן ניתן לבצע שינויים בדיאגרמות בצורה פשוטה בעוד שלבצע שינויים ותיקונים בקטעי קוד יכולה להיות עבודה סיזיפית אשר גוזלת זמן רב. לכן מומלץ לבצע תכנון של הפרויקט באמצעות הדיאגרמות.

**יתרון ספציפי של הדיאגרמה:**

בפרויקט שלנו ישנו תהליך מרכזי וחשוב של הגשת בקשה וטיפול בבקשה. כאשר הטיפול בבקשה מורכב ממספר בעלי תפקידים שונים אשר כל אחד מהם מבצע שלב אחד בודד מתוך תהליך גדול. לאחר ניתוח הסיפור והדרישות שנובעות ממנו, הצלחנו בצוות לאבחן ולנתח שתהליך זה מהווה את לב ליבה של המערכת והפונקציונליות העיקרית שלה. כלומר ישנה חשיבות גבוהה ביותר להבנה של התהליך ומימושו בצורה שתענה על דרישות הלקוח וכמובן תאפשר ניצול אידאלי ומיטבי של משאבינו. נדרש מימוש שיאפשר לנו לבצע שימוש חוזר בקטעי קוד שונים ולפשט את הפונקציונליות לתתי פונקציות אשר כל אחת מהן עושה משהו בסיסי ופשוט ובכך לאפשר לנו קוד קריא יותר. כמו כן יקל על הנוחות בסביבת עבודה בצוות כאשר העבודה הינה שיתופית.

דרך דיאגרמת הActivity הצלחנו לתאר בצורה חזותית וברורה את זרימת המידע, השלבים והשחקנים המשתתפים בכל שלב עבור תהליך הגשה וטיפול בבקשה. הצלחנו לזקק את הפעולות המתבצעות ולהבין את כל ההסתעפויות השונות שיכולות לקרות במהלך התהליך.

לדוגמא בשלב "בחינה והחלטה" ייתכנו שלוש אופציות שונות: דחייה, אישור או בקשת מידע נוסף. לאחר המידול ברור לנו שכל אופציה בהסתעפות בעצם תעביר את הבקשה לשלב אחר, אישור הבקשה יקדם אותנו אל שלב "ביצוע", דחיית הבקשה תקפיץ אותנו היישר אל שלב ה"סגירה" ובקשת פרטים נוספים תחזיר אותנו אחרוה אל שלב "הערכת משמעות".

כמו כן זיהינו תוך כדי המידול שאומנם אנו מבינים את הזרימה של התהליך בגדול אך כל שלב בפני עצמו מהווה רצף שלם של אירועים שצריכים להתרחש ולכן בחרנו לתאר כל שלב בתור תת דיאגרמה שגם היא תהיה דיאגרמת Activity.

מידול זה מאפשר לנו רמות שונות של רזולוציה על התהליך והבנה עמוקה יותר של כל אחד מהשלבים שמתבצעים, כמו כן בעזרת ההסתכלות העמוקה ניתן לזהות מספר פעולות שחוזרות על עצמן בכל שלב כגון: קביעת זמן הנדרש לביצוע השלב (בשלבי "הערכת משמעות" ו"ביצוע") והמתנה לאישור המפקח.

במהלך תכנון הפרויקט ומימוש מייצג ההדגמה ואב טיפוס פעיל, השתמשנו בדיאגרמת Class בכדי לתאר את כל המחלקות והרכיבים השונים של הפרויקט. החל מכל הצגים השונים, אשר נקראים GUI – Graphical User Interface, דרך כל הבקרים שלהם (Controllers), וכמובן הישויות (Entities) והאובייקטים שיהיו לנו בפרויקט. מידול זה עזר לנו לתכנן מראש כיצד יראה הפרויקט וכמובן אילו מתודות עלינו לממש. זה מאפשר לנו לתת הערכת זמן והיקף עבודה די טובה לביצוע כלל העבודה.

תוך כדי תכנון הפרויקט בדיאגרמה צפו כל מיני שאלות על ארכיטקטורת התוכנה שנרצה להשתמש בה, בהתחלה רצינו להשתמש בבקר אחד כולל עבור כל התהליך של הגשה וטיפול בבקשה. אבל במבט על הדיאגרמה הבנו שבקר זה יהיה עמוס במתודות שונות, ויהיה עמוס מידי, דבר שיקשה לנו על מימושו וכמובן במבט לעתיד על תחזוקתו ושמירת הקוד בצורה נקייה.

כמו כן זה עזר לנו להבין יותר טוב את זרימת המידע בפרויקט כאשר הוא יעבור מהפעולות של המשתמש בצג כלשהו (GUI) דרך הבקרים, אל מחלקת הלקוח שתדבר מול השרת שיפעיל את החיבור שלו למסד הנתונים וישלוף את המידע הרצוי ויחזירו באותו קו בכיוון הנגדי. ללא דיאגרמת Class היה לנו יותר קשה להבין את זרימת המידע הרצויה והדרך לממשה.

1. **קשיים הנובעים מחסרונות של UML**

* **יותר מידי דגש על עיצוב הפרויקט:**

דיאגרמות UML שמות דגש רב על עיצוב, דבר שיכול להיות בעייתי עבור צוותים קטנים כמו הצוות שלנו שמונה 5 מתכנתים. הרגשנו במהלך תכנון הדיאגרמות שאולי ניתחנו חלקים מסוימים מהעבודה והפרויקט יתר על המידה. כמו כן איבדנו ריכוז בכך שהשקענו זמן רב מאוד במחשבה על פיצ'רים של תוכנה שנרצה לכלול ולהוסיף.

בסופו של יום לא ניתן לפתור על בעיית תכנות באמצעות דיאגרמת UML, לבסוף צריך להפשיל את השרוולים ולהתחיל לקודד ולבצע בדיקות על הקוד. דיאגרמות UML אמורות לשמש למבט חזותי על התוכנה בכדי לעזור לנו לתכנן אותה תוך כדי פיתוח. אם נתקענו יותר מידי זמן בלאבחן ולפתור בעיות, זה יכול לגרום לעיכובים בעבודה שצריכה להתבצע בכדי לתקן את הבעיות.

* **דיאגרמות שנעשו גדולות ומסובכות:**

ככל שהתקדמנו יותר בדיאגרמות רצינו לכלול בתוכן כל תרחיש אפשרי, וכל סיטואציה שניתן להגיע אליה ולתת לה התייחסות בדיאגרמות. בסופו של יום זה גרם לנו לדיאגרמות די מסובכות ומבלבלות. לדוגמא עבור החלק של "מעקב ובקרה", יצרנו דיאגרמת Activity שלמה משלו. הרעיון היה שזוהי תהיה תת-דיאגרמה קטנה יחסית שתכלול התייחסות למקרים האלו עבור כל שלב בתהליך הגשת הבקשה, בכדי להקל על הנראות וההבנה של הדיאגרמות עבור כל שלב. בפועל מה שיצא זה דיאגרמה מאוד גדולה ומורכבת, שנעשתה אף יותר גדולה מהדיאגרמת Activty שמתארת במבט על את כל התהליך, דיאגרמה שעל פי התכנון הראשונה ציפינו שהיא תכלול את המסה העיקרית של המידע.

מחד גיסא הדיאגרמה מאפשרת לנו לתאר את התהליך בצורה מדויקת ומפורטת ומאידך גיסא יצרנו בלבול, הדיאגרמה גדולה וקשה להבנה. על אף כל הנוחות שמבט חזותי על הפעולות, יש צורך בבחינה מדוקדקת בכדי להבין את רצף הפעולות בדיאגרמה זו (Monitoring And Control).

1. **ניתוח ודיון:**

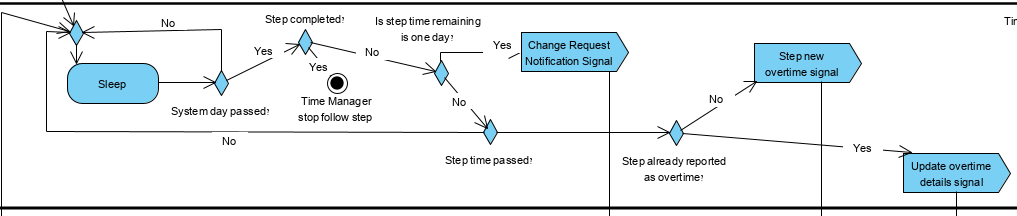
* בכדי לפתור את הבעיה של דגש יתר על העיצוב, ניתן אולי להוסיף תוסף לדיאגרמות כך שחלק מהן תוכל לתאר על ידי קוד פשוט בשפת מונחה עצמים \ פסאודו קוד. הדבר יאפשר לך להראות רצף האירועים על ידי מתודות ופונקציות כפי שהביצוע בפועל יתרחש, וכמו כן לעיתים ישנם דברים אשר מומחשים בצורה טובה יותר על ידי הקוד במקום לתאר בשרטוטים וציורים דברים שלא בהכרח מתאימים לכך.

דבר זה יאפשר לך לקפוץ ישירות מתהליך תכנון הפרויקט בדיאגרמות אל תהליך המימוש שכן פשוט נדרש לכתוב את הקוד כך שיתאים לשפה ולארכיטקטורה של הפרויקט.

כמו כן הוא יגרום למפתחים להרגיש יותר בסביבת העבודה הנוחה והטבעית שלהם כיוון שרוב ההתעסקות הינה בקוד, והצורך הוא הבנה מיטבית של הקוד ולא של השרטוטים.

חסרון אפשרי של תוסף זה יהיה סף מסוים של הבנה בתוכנה בכדי לייצר אותו, לקרוא אותו ולהבין אותו. כמובן שבסופו של יום כל צוות משתמש בדיאגרמות לצרכים האישיים שלו וכנראה לצרכי מימוש התוכנה והבנתה על ידי צוותי תכנה הוא יתאים יותר, ולצרכי הצגה ללקוח או למנהלים טכניים אולי הוא יהיה בשימוש פחות נרחב ויעדיפו לתאר תהליכים על ידי הדיאגרמות הקיימות כעת.

לדוגמא בתהליך מעקב ובקרה בדיאגרמת Activity ישנם הרבה צמתי החלטות ברצף:



בפועל על ידי תוכנה זה ימומש ככל הנראה במנגנון של פעולות If / Else, כנראה שעבור כל מתכנת ממוצע תהליך זה היה גם הרבה יותר נוח ליצירה וגם לקריאה על ידי קוד שיכיל פעולות If / Else ושמות פונקציות סתמיות של כל פעולה שתתבצע בכל אחד מהמקרים:

While (Current System Day == System Day)

{}

If (Step Complete)

Exit

Else

If(Step Time Remaining == 1)

{Send Notification}

Else If (Step Time != Passed)

Repeat line 1

Etc….

כמובן שבתור תוסף למערכת תהיה אפשרות לבצע זאת במתמלל קוד שיאפשר לך נוחות ביצירת כל הצמתים השונים אולי אף לתרגם אותם לדיאגרמת Activity כפי שמתואר בתרשים.

* על מנת לפתור דיאגרמות גדולות ומסובכות ישנה כבר אופציה קיימת של "תת-דיאגרמה" אשר מאפשרת לך לשים דיאגרמה שלמה בתור אובייקט בתוך דיאגרמה אחרת ובכך לייצר היררכיה ורמות שונות של רזולוציה.

אולי ניתן להתלבש על אפשרות זאת ולהרחיבה כך שגם יהיה יותר ברור שאכן אובייקט הוא בעצם תת-דיאגרמה, על ידי ממשק יותר נוח וסמלים בולטים יותר. אפשרות נוספות שניתן להוסיף היא שתהיה אפשרות להרחיב כל אובייקט שמתאר תת -דיאגרמה ולכווצו בדיאגרמה שאתה מסתכל עליה באותו הרגע. כך שאין צורך לבצע מעבר בין דפים שיכול לגרור לחוסר סדר ובלבול מסוים בתהליך היצירה של הדיאגרמות. כמו כן יחסוך עבודה כפולה של תיאור דמויות, משתמשים ותהליכים שחוזרים על עצמם בכל תת-דיאגרמה.

לדוגמא בתרשים שלנו ישנה את דיאגרמת Zoom-Out שמתארת את כל תהליך של הגשה בקשת שינוי וטיפול בבקשה על ידי כל הגורמים הרלוונטיים. בתוך דיאגרמה זו בחרנו לתאר כל שלב בתהליך בתור תת-דיאגרמה משלו. זה מאפשר לנו להיות יותר כלליים בתרשים העל ושהוא לא יהיה עמוס מידי, אך אם כאשר אנו רוצים לעבור על התהליך בפרטי פרטים ולהבינו בצורה מלאה מההתחלה ועד הסוף הייתה לנו את היכולת והאפשרות לפתוח כל תת-דיאגרמה ולהסתכל עליה בהקשר ובמקום של דיאגרמת העל אנו מניחים שזה היה תורם רבות להבנה של התהליך וגם לנוחות השימוש בדיאגרמות, במקום דפדוף בין מספר דיאגרמות כפי שיש לבצע בכלים הנתונים כיום.